From the Applied Pi Calculus to Horn Clauses for Protocols with Lists

Miriam Paiola and Bruno Blanchet

INRIA Paris-Rocquencourt
{paiola,blanchet}@inria.fr

Abstract. Recently [7], we presented an automatic technique for proving secrecy and authentication properties for security protocols that manipulate lists of unbounded length, for an unbounded number of sessions. That work relies on an extension of Horn clauses, generalized Horn clauses, designed to support unbounded lists, and on a resolution algorithm on these clauses. However, in that previous work, we had to model protocols manually with generalized Horn clauses, which is unpractical. In this work, we present an extension of the input language of ProVerif, a variant of the applied pi calculus, to model protocols with lists of unbounded length, give its formal semantics, and translate it automatically to generalized Horn clauses. We prove that this translation is sound.

1 Introduction

Security protocols rely on cryptography for securing communication on insecure networks such as Internet. However, attacks are often found against protocols that were thought correct. Furthermore, security flaws cannot be detected by testing since they appear only in the presence of an attacker. Formal verification can then be used to increase the confidence in these protocols. To ease formal verification, one often uses the symbolic, so-called Dolev-Yao model [11], which considers cryptographic primitives as black boxes and messages as terms on these primitives. In this work, we also rely on this model.

The formal verification of security protocols with fixed-size data structures has been extensively studied. However, some protocols, for instance XML protocols of web services or some group protocols, use more complex data structures, such as lists. The verification of protocols that manipulate such data structures has been less studied and presents additional difficulties, since these complex data structures add another cause of undecidability.

Recently [7], we started to extend the automatic verifier ProVerif [5] to protocols with lists of unbounded length. ProVerif takes as input a protocol written in a variant of the applied pi calculus [1], translates it into a representation in Horn clauses, and uses a resolution algorithm to determine whether a fact is derivable from the clauses. One can then infer security properties of the protocol. For instance, ProVerif uses a fact $\text{att}(M)$ to mean that the attacker may have the message M. If $\text{att}(s)$ is not derivable from the clauses, then s is secret. The main goal of this approach is to prove security properties of protocols without bounding the number of sessions of the protocol.

In [7], we introduced generalized Horn clauses, to be able to represent lists of any length, and we adapted the resolution algorithm of ProVerif to deal with these new clauses. Using this algorithm, we can prove secrecy and authentication properties of protocols with lists of any length, without bounding the number of sessions of the protocol. However, to use this algorithm, one has to write the generalized Horn clauses that model the protocol manually, which is delicate and error-prone. In this paper, our goal is to solve this problem by providing a more convenient input language for protocols. More precisely, we extend the input language of ProVerif to model protocols with lists of unbounded length. We give a formal semantics to the new process calculus, and an automatic translation to generalized Horn clauses. We prove that this translation is sound. One can apply the resolution algorithm of [7] to the generalized Horn clauses obtained by our translation, to prove secrecy and authentication properties of the initial protocol. We illustrate our work on a small protocol that relies on XML signatures; it could obviously be applied to other protocols such as those considered in [7]. This work is only theoretical: the implementation is planned for future work.

Related Work The first approach considered for proving protocols with recursive data structures was interactive theorem proving: a recursive authentication protocol was studied for an unbounded number of participants, using Isabelle/HOL [15], and using rank functions and PVS [9]. However, this approach requires considerable human effort.

Truderung [17] showed a decidability result (in NEXPTIME) for secrecy in recursive protocols, which include transformations of lists, for a bounded number of sessions. This result was extended to a class of recursive protocols with XOR [13] in 3-NEXPTIME. Chridi et al [10] present an extension of the constraint-based approach in symbolic protocol verification to handle a class of protocols (Well-Tagged protocols with Autonomous keys) with unbounded lists in messages. They prove that the insecurity problem for Well-Tagged protocols with Autonomous keys is decidable for a bounded number of sessions.

Several approaches were considered for verifying XML protocols [4,16,12,3], by translating them to the input format of a standard protocol verifier: the tool TulaFale [4] uses ProVerif as back-end; Kleiner and Roscoe [16,12] translate WS-Security protocols to FDR; Backes et al [3] use AVISPA. All these approaches have little or no support for lists of unbounded length. For instance, TulaFale has support for list membership with unbounded lists, but does not go further.

In [14], we showed that, for a certain class of Horn clauses, if secrecy is proved by ProVerif for lists of length one, then secrecy also holds for lists of unbounded length. However, this work is limited to secrecy and to protocols that treat all elements of lists uniformly. When this reduction result does not apply, a different approach is needed: in our previous work [7], we proposed such an approach.

Outline In the next section, we recall the process calculus used by ProVerif and we extend it with the non-deterministic choice. We also adapt its translation into Horn clauses. In Section 3, we recall the syntax of generalized Horn clauses and their translation into Horn clauses. Section 5 defines the new process calculus and its semantics. Section 6 gives the automatic translation of generalized processes into generalized Horn clauses. In Section 7, we prove that this translation is sound. Because of space constraints, the proofs and additional details are postponed to the appendix.

2 ProVerif

ProVerif [5] takes as input a process written in a variant of the applied pi calculus [1]. ProVerif then translates this process into an abstract representation by Horn clauses. It uses a resolution algorithm to determine whether some facts are derivable from these clauses, and infer security properties on the initial process.

2.1 The Process Calculus: Syntax and Semantics

The syntax of the process calculus assumes an infinite set of names a, b, c, k, s, to be used for representing atomic data items, such as keys or nonces, and an infinite set of variables x, y, z. There is also a set of function symbols for constructors (f) and destructors (g), each with an arity. Constructors build new terms of the form $f(M_1, \dots, M_n)$. Therefore, messages are represented by terms M, N, which can be a variable, a name, or a constructor application $f(M_1, \dots, M_n)$. Destructors manipulate terms in processes; they are defined by rewrite rules as detailed below.

Protocols are represented by processes P, Q, of the following forms:

- The output process $\mathsf{out}(M, N).P$ outputs the message N on the channel M and then executes P.
- The input process $\mathsf{in}(M, x).P$ inputs a message on the channel M and then executes P with x bound to the input message.
- The nil process $\mathbf{0}$ does nothing.
- The process $P \mid Q$ is the parallel composition of P and Q.
- The replication $!P$ represents an infinite number of copies of P in parallel.
- The restriction $(\nu a)P$ creates a new name a and then executes P.
- The destructor application $\mathsf{let}\ x = g(M_1, \dots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q$ tries to evaluate $g(M_1, \dots, M_n)$; if this succeeds, then x is bound to the result and P is executed, else Q is executed. More precisely, a destructor g is defined by a set $\mathit{def}(g)$ of rewrite rules of the form $g(M_1, \dots, M_n) \to M$ where $M_1, \dots, M_n, M$ are terms without free names, and the variables of M also occur in $M_1, \dots, M_n$. Then $g(M_1, \dots, M_n)$ evaluates to M if and only if it reduces to M by a rewrite rule of $\mathit{def}(g)$. Using constructors and destructors, one can represent data structures and cryptographic operations:
  - The constructor pk builds a new public key $pk(M)$ from a secret key M. The constructor $sign$ is such that $sign(M, N)$ represents the signature of M under the key N. It has one corresponding destructor:

$$checksign(sign(x, y), pk(y), x) \to x$$

Hence, $\mathit{checksign}(\mathit{sign}(M,N), \mathit{pk}(N), M)$ checks if $\mathit{sign}(M,N)$ is a correct signature of message M under the secret key N; if yes, it returns the message M; otherwise, it fails.

- A data constructor is a constructor f of arity n that comes with n associated destructors f_i^{-1} $(1 \leq i \leq n)$, defined by rewrite rules $f_i^{-1}(f(x_1, \dots, x_n)) \rightarrow x_i$, so that the arguments of f can be recovered. Data constructors are typically used to represent data structures.

– The pattern-matching $\mathsf{let}\ \mathit{pat} = M\ \mathsf{in}\ P\ \mathsf{else}\ Q$ matches M with the pattern pat, and executes P when the matching succeeds and Q when it fails. The pattern pat can be a variable x or a data constructor application $f(\mathit{pat}_1, \dots, \mathit{pat}_n)$. Patterns pat are linear, that is, they never contain several occurrences of the same variable. Pattern-matching can be encoded using destructor application: $\mathsf{let}\ x = M\ \mathsf{in}\ P\ \mathsf{else}\ Q$ is an abbreviation for $\mathsf{let}\ x = \mathit{id}(M)\ \mathsf{in}\ P\ \mathsf{else}\ Q$, where the destructor id is defined by $\mathit{id}(x) \rightarrow x$ and $\mathsf{let}\ f(\mathit{pat}_1, \dots, \mathit{pat}_n) = M\ \mathsf{in}\ P\ \mathsf{else}\ Q$ is an abbreviation for

$$\begin{array}{l}\mathsf{let}\ x_1 = f_1^{-1}(M)\ \mathsf{in}\ \dots\ \mathsf{let}\ x_n = f_n^{-1}(M)\ \mathsf{in}\\ \quad \mathsf{let}\ \mathit{pat}_1 = x_1\ \mathsf{in}\ \dots\ \mathsf{let}\ \mathit{pat}_n = x_n\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \dots\ \mathsf{else}\ Q\\ \mathsf{else}\ Q\ \dots\ \mathsf{else}\ Q\end{array}$$

where the variables $x_1, \dots, x_n$ are fresh and the variables of $\mathit{pat}_1, \dots, \mathit{pat}_n$ do not occur in Q.

– ProVerif models authentication as correspondence assertions, such as "if event $e(x)$ has been executed, then event $e'(x)$ has been executed". The process calculus provides an instruction for executing such events: the process $\mathsf{event}(e(M)).P$ executes the event $e(M)$, then executes P.

– We add a construct for internal choice, which was not present in [5]: the process $P + Q$ behaves either as P or as Q, non-deterministically. This construct will be helpful for defining our extension to lists.

The conditional $\mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q$ can be encoded as the destructor application $\mathsf{let}\ x = \mathit{equal}(M,N)\ \mathsf{in}\ P\ \mathsf{else}\ Q$ where x does not occur in P and the destructor equal, defined by $\mathit{equal}(x,x) \rightarrow x$, succeeds if and only if its two arguments are equal. We often omit a trailing $\mathbf{0}$.

The name a is bound in P in the process $(\nu a)P$. The variable x is bound in P in the processes $\mathsf{in}(M,x).P$ and $\mathsf{let}\ x = g(M_1, \dots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q$. The variables of pat are bound in P in the process $\mathsf{let}\ \mathit{pat} = M\ \mathsf{in}\ P\ \mathsf{else}\ Q$. A process is closed if it has no free variables; it may have free names. We denote by $\mathit{fn}(P)$ the free names of P.

The formal semantics of this calculus can be defined either by a structural equivalence and a reduction relations, in the style of [1], or by a reduction relation on semantic configurations, as in [5]. These semantics can easily be extended with the internal choice, by adding rules such that the process $P + Q$ reduces into P and also into Q.

2.2 Horn Clauses

ProVerif translates a protocol written in the process calculus into a set of Horn clauses. The syntax of these clauses is defined as follows.

The terms, named *clause terms* to distinguish them from the terms that occur in processes, represent the messages of the protocol. A term p can be a variable x, a name $a[p_1, \dots, p_n]$, or a constructor application $f(p_1, \dots, p_n)$. A variable can represent any term. Instead of representing each fresh name by a different symbol in the clauses, the fresh names are considered as functions represented by the clause term $a[p_1, \dots, p_n]$. These functions take as arguments the messages previously received by the principal that creates the name as well as session identifiers, which are variables that take a different value at each run of the protocol, to distinguish names created in different runs. As shown in, e.g., [5], this representation of names is a sound approximation.

A fact $F = \mathit{pred}(p_1, \dots, p_n)$ can be of the following forms: $\mathrm{message}(p, p')$ means that the message p' may appear on channel p; $\mathrm{att}(p)$ means that the attacker may have the message p; $\text{m-event}(p)$ represents that the event p must have been executed; $\mathrm{event}(p)$ represents that the event p may have been executed.

A clause $F_1 \wedge \dots \wedge F_n \Rightarrow F$ means that, if all facts F_i are true, then the conclusion F is also true. We use R for a clause, H for its hypothesis, and C for its conclusion. The hypothesis of a clause is considered as a multiset of facts. A clause with no hypothesis $\Rightarrow F$ is written simply F.

2.3 Translation from the Process Calculus to Horn Clauses

As explained in [5], ProVerif uses two sets of clauses: the clauses for the attacker and the clauses for the protocol.

Clauses for the Attacker. Initially the attacker has all the names in a set S, hence the clauses $\mathrm{att}(a[\,])$ for each $a \in S$. Moreover, the abilities of the attacker are represented by the following clauses:

$$\mathrm{att}(b[x]) \tag{Rn}$$

for each constructor f of arity n,

$$\mathrm{att}(x_1) \wedge \dots \wedge \mathrm{att}(x_n) \Rightarrow \mathrm{att}(f(x_1, \dots x_n)) \tag{Rf}$$

for each destructor g, for each rule $g(M_1, \dots, M_n) \rightarrow M$ in $\mathit{def}(g)$,

$$\mathrm{att}(M_1) \wedge \dots \wedge \mathrm{att}(M_n) \Rightarrow \mathrm{att}(M) \tag{Rg}$$

$$\mathrm{message}(x, y) \wedge \mathrm{att}(x) \Rightarrow \mathrm{att}(y) \tag{Rl}$$

$$\mathrm{att}(x) \wedge \mathrm{att}(y) \Rightarrow \mathrm{message}(x, y) \tag{Rs}$$

Clause (Rn) represents the ability of the attacker to create fresh names: all fresh names that the attacker may create are represented by the names $b[x]$ for any x. Clauses (Rf) and (Rg) mean that if the attacker has some terms, than he can apply constructors and destructors to them. Clause (Rl) means that if the attacker has a channel x then he can listen on it and clause (Rs) means that the attacker can send messages in all the channels he has.

Clauses for the Protocol. The protocol is represented by a closed process P_0. To compute the clauses, we first rename the bound names of P_0 so that they are pairwise distinct and distinct from free names of P_0. This renaming is important because bound names are also used as function symbols in terms in the generated clauses. We make an exception for the new construct $P+Q$: the bound names in P need not be distinct from those in Q. Using the same symbols for both names in P and Q does not cause problems because P and Q cannot be both executed. We say that the renamed process, denoted P'_0, is a *suitable renaming* of P_0.

Next, we instrument the process P'_0 by labeling each replication $!P$ with a distinct session identifier s, so that it becomes $!^s P$, and labeling each restriction (νa) with the clause term that corresponds to the fresh name a, $a[x_1, \dots, x_n, s_1, \dots, s_{n'}]$, where $x_1, \dots, x_n$ are the variables that store the messages received in inputs above (νa) in P'_0 and $s_1, \dots, s_{n'}$ are the session identifiers that label replications above (νa) in the instrumentation of P'_0. We denote the instrumentation of P'_0 by $\mathrm{instr}(P'_0)$.

Then we compute the clauses as follows. Let ρ be a function that associates a clause term with each name and variable. We extend ρ as a substitution by $\rho(f(M_1, \dots, M_n)) = f(\rho(M_1), \dots, \rho(M_n))$ if f is a constructor.

The translation $[\![P]\!]\rho H$ of an instrumented process P is a set of clauses, where the environment ρ is a function defined as above and H is a sequence of facts $\mathrm{message}(\cdot,\cdot)$ and $\text{m-event}(\cdot)$. The empty sequence is $\emptyset$ and the concatenation of a fact F to the sequence H is denoted by $H \wedge F$. The translation $[\![P]\!]\rho H$ is defined as follows, and explained below.

- $[\![\mathsf{out}(M, N).P]\!]\rho H = [\![P]\!]\rho H \cup \{H \Rightarrow \mathrm{message}(\rho(M), \rho(N))\}$.
- $[\![\mathsf{in}(M, x).P]\!]\rho H = [\![P]\!](\rho[x \mapsto x])(H \wedge \mathrm{message}(\rho(M), x))$.
- $[\![\mathbf{0}]\!]\rho H = \emptyset$.
- $[\![P \mid Q]\!]\rho H = [\![P]\!]\rho H \cup [\![Q]\!]\rho H$.
- $[\![!^s P]\!]\rho H = [\![P]\!](\rho[s \mapsto s])H$.
- $[\![(\nu a : a'[x_1, \dots, x_n, s_1, \dots, s_{n'}])P]\!]\rho H =$
  $[\![P]\!](\rho[a \mapsto a'[\rho(x_1), \dots, \rho(x_n), \rho(s_1), \dots, \rho(s_{n'})]])H$.
- $[\![\mathsf{let}\ x = g(M_1, \dots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]\rho H = \bigcup\{[\![P]\!]((\sigma\rho)[x \mapsto \sigma' p'])(\sigma H) \mid g(p'_1, \dots, p'_n) \to p'$ is in $\mathit{def}(g)$ and (σ, σ') is a most general pair of substitutions such that $\sigma\rho(M_1) = \sigma' p'_1, \dots, \sigma\rho(M_n) = \sigma' p'_n\} \cup [\![Q]\!]\rho h$.
- $[\![\mathsf{event}(e(M)).P]\!]\rho H = [\![P]\!]\rho(H \wedge \text{m-event}(e(\rho(M)))) \cup \{H \Rightarrow \mathrm{event}(e(\rho(M)))\}$.
- $[\![P + Q]\!]\rho H = [\![P]\!]\rho H \cup [\![Q]\!]\rho H$.

The translation of an output $\mathsf{out}(M, N).P$ adds a clause, meaning that the reception of the messages in H can produce the output in question. The translation of an input $\mathsf{in}(M, x).P$ is the translation of P with the concatenation of the input to H. The translation of $\mathbf{0}$ is empty, as this process does nothing. The translation of the parallel composition $P \mid Q$ is the union of the translation of P and Q. The translation of the replication adds the session identifier to ρ; as the clauses can be applied many times, replication is otherwise ignored. The translation of a restriction $(\nu a)P$ is the translation of P in which a is replaced with the corresponding clause term that depends on previously received messages and on session identifiers of replications above the restriction. The translation of a destructor application is the union of the translation for the case where the destructor succeeds and that for the case where it fails, so the translation does not have to determine whether the destructor succeeds or not, but considers both the possibilities. We consider that the else branch may always be executed, which overapproximates the possible behaviors of the process. The translation of an event adds the hypothesis $\text{m-event}(e(\rho(M)))$ to H, meaning that P can be executed only if the event $e(M)$ has been executed first. Furthermore, it adds a clause that concludes $\text{event}(e(\rho(M)))$, meaning that the event $e(M)$ is triggered when all conditions in H are true. The translation of the choice $P + Q$ is the union of the translation of P and Q, since $P + Q$ behaves either as P or as Q. The choice was not included in [5]; we have easily extended the proofs of the results of [5] to the internal choice. (It is also possible to encode $P + Q$ as $(\nu a)(a\langle a\rangle \mid a(x).P \mid a(x).Q)$ where a and x do not occur in P and Q. This encoding leads to more complex clauses so we preferred defining $P+Q$ as a new construct.)

Summary and correctness. Let $\rho_0 = \{a \mapsto a[\,] \mid a \in fn(P_0)\}$. The set of the clauses corresponding to the closed process P_0 is defined as:

$$\mathcal{R}_{P_0',S} = [\![\text{instr}(P_0')]\!]\rho_0\emptyset \cup \{\text{att}(a[\,]) \mid a \in S\} \cup \{(\text{Rn}), (\text{Rf}), (\text{Rg}), (\text{Rl}), (\text{Rs})\}$$

where P_0' is a suitable renaming of P_0 and S is the set of names initially known by the attacker.

By testing derivability of facts from these clauses, we can prove security properties of the protocol P_0, as shown by the following two results. These results are applications of [5, Theorem 1] to the particular properties of secrecy and authentication modeled as non-injective agreement. The formal definitions of these properties can be found in [5]. For this paper, it is sufficient to know that the following results hold. Let $\mathcal{F}_{\text{me}}$ be any set of facts of the form $\text{m-event}(p)$; this set corresponds to the set of allowed events. As explained in [5], this set is useful to prove the desired correspondence for authentication. We refer the reader to [5, Section 4] for further details.

Theorem 1 (Secrecy). *Let P_0' be a suitable renaming of P_0. Let M be a term. Let p be the clause term obtained by replacing names a with $a[\,]$ in M. If $\text{att}(p)$ is not derivable from $\mathcal{R}_{P_0',S} \cup \mathcal{F}_{\text{me}}$ for any $\mathcal{F}_{\text{me}}$, then P_0 preserves the secrecy of M from adversaries with initial knowledge S.*

Theorem 2 (Authentication). *Let P_0' be a suitable renaming of P_0. Suppose that, for all $\mathcal{F}_{\text{me}}$, for all p, if $\text{event}(e(p))$ is derivable from $\mathcal{R}_{P_0',S} \cup \mathcal{F}_{\text{me}}$, then $\text{m-event}(e'(p)) \in \mathcal{F}_{\text{me}}$. Then P_0 satisfies the correspondence "if $e(x)$ has been executed, then $e'(x)$ has been executed" against adversaries with initial knowledge S.*

3 Generalized Horn Clauses

This section recalls the syntax and semantics of *generalized Horn Clauses*, which extend Horn clauses to lists and were introduced in [7].

$\iota ::=$	index terms
$\quad i$	index variable
$\quad \phi(\iota_1, \ldots, \iota_h)$	function application
$p^G ::=$	clause terms
$\quad x_{\iota_1,\ldots,\iota_h}$	variable ($h \geq 0$)
$\quad f(p_1^G, \ldots, p_n^G)$	function application
$\quad a_{\iota_1,\ldots,\iota_h}^{L_1,\ldots,L_h}[p_1^G, \ldots, p_n^G]$	indexed names
$\quad \mathit{list}(i \leq L, p^G)$	list constructor
$\mathcal{C} ::= \bigwedge_{(i_1,\ldots i_h) \in [1,L_1] \times \cdots \times [1,L_h]}$	conjunctions
$F^G = \mathcal{C}\, \mathit{pred}(p_1^G, \ldots, p_l^G)$	facts
$E ::= \mathcal{C}\, p^G \doteq p'^G$	equations
$\mathcal{E} ::= \{E_1, \ldots, E_n\}$	set of equations
$R^G ::= F_1^G \wedge \cdots \wedge F_n^G \wedge \mathcal{E} \Rightarrow \mathit{pred}(p_1^G, \ldots, p_l^G)$	generalized Horn clauses

Fig. 1. Syntax of generalized Horn clauses

3.1 Syntax

The syntax of these clauses is defined in Figure 1. Clause terms p^G represent messages: variables may have indices $x_{\iota_1,\ldots,\iota_h}$; these indices ι are build from index variables and application of functions on indices. The term $f(p_1^G, \ldots, p_n^G)$ represents constructor application. For each integer n, we introduce a new data constructor $\langle p_1^G, \ldots, p_n^G \rangle$, which represents lists of fixed length n. The clause term $a_{\iota_1,\ldots,\iota_h}^{L_1,\ldots,L_h}[p_1^G, \ldots, p_n^G]$ represents a fresh name a indexed by $\iota_1, \ldots, \iota_h$ in $[1, L_1]$, $\ldots$, $[1, L_h]$ respectively. The construct $\mathit{list}(i \leq L, p^G)$ represents lists of variable length L: $\mathit{list}(i \leq L, p^G)$ represents intuitively the list $\langle p^G\{1/i\}, \ldots, p^G\{L/i\} \rangle$.

Facts are represented by $\bigwedge_{(i_1,\ldots,i_h) \in [1,L_1] \times \cdots \times [1,L_h]} \mathit{pred}(p_1^G, \ldots, p_l^G)$. The symbol $[1, L]$ represents the set $\{1, \ldots, L\}$. The set of equations $\mathcal{E}$ serves to remember equalities between terms. Keeping equations is especially useful when they cannot be immediately used to infer the value of some variables and substitute them in the rest of clause. For instance, the equation $x_i \doteq p^G$ does not determine the value of all instances of x_ι, because the equation holds for a single index i and not for all indices, so the equation remains for future use. The clause $F_1^G \wedge \cdots \wedge F_n^G \wedge \mathcal{E} \Rightarrow \mathit{pred}(p_1^G, \ldots, p_l^G)$ means that, if the facts F_1^G, $\ldots$, F_n^G and the equations in $\mathcal{E}$ hold, then the fact $\mathit{pred}(p_1^G, \ldots, p_l^G)$ also holds. The conclusion of a clause does not contain a conjunction $\mathcal{C}$: we can simply leave the indices of $\mathit{pred}(p_1^G, \ldots, p_l^G)$ free to mean that $\mathit{pred}(p_1^G, \ldots, p_l^G)$ can be concluded for any value of these indices. We use H^G for hypothesis and C^G for conclusions.

These clauses are simplified with respect to [7]: in [7], we considered conjunctions over arbitrary subsets of $[1, L_1] \times \cdots \times [1, L_h]$ and equations on indices. These two features appear during the resolution algorithm on generalized Horn clauses, but are not needed in the initial clauses, so we omit them here. We still introduce two minor extensions with respect to [7]: we consider names with any number of indices instead of just 0 or 1 index, and predicates of any arity instead of just arity 1. (The predicate message has arity 2.) It is straightforward to extend the resolution algorithm of [7] to this more general situation.

3.2 Translation from Generalized Horn Clauses to Horn Clauses

A generalized Horn clause represents several Horn clauses: for each value of the bounds L, functions ϕ, and free indices i that occur in a generalized Horn clause R^G, R^G corresponds to a certain Horn clause. This section defines this correspondence, which gives the formal semantics of the generalized Horn clauses.

As in [7], we can define a type system for generalized Horn clauses, to guarantee that the indices of all variables vary in the appropriate interval. The judgment $\Gamma \vdash R^G$ means that the clause R^G is well-typed in the type environment Γ, which is a sequence of type declarations of the following forms: $i : [1, L]$ means that index i can vary between 1 and L; $\phi : [1, L_1] \times \ldots \times [1, L_h] \to [1, L]$ means that function ϕ expects as arguments h indices of types $[1, L_j]$ for $j = 1, \ldots, h$, and returns an index of type $[1, L]$; $x\_ : [1, L_1] \times \ldots \times [1, L_h]$ means that the variable x expects indices of types $[1, L_j]$ for $j = 1, \ldots, h$.

Definition 1 *Given a well-typed generalized Horn clause $\Gamma \vdash R^G$, an environment T for $\Gamma \vdash R^G$ is a function that associates:*

- *to each bound L that appears in R^G or Γ an integer L^T;*
- *to each index i such that $i : [1, L] \in \Gamma$, an index $i^T \in \{1, \ldots, L^T\}$;*
- *to each index function ϕ such that $\phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L] \in \Gamma$, a function $\phi^T : \{1, \ldots, L_1^T\} \times \cdots \times \{1, \ldots, L_h^T\} \to \{1, \ldots, L^T\}$.*

Given an environment T and values $v_1, \ldots, v_h$, we write $T[i_1 \mapsto v_1, \ldots, i_h \mapsto v_h]$ for the environment that associates to indices $i_1, \ldots, i_h$ the values $v_1, \ldots, v_h$ respectively and that maps all other values like T.

Given an environment T for $\Gamma \vdash R^G$, the generalized Horn clause R^G is translated into the standard Horn clause R^{GT} defined as follows. We denote respectively $p^{GT}, E^T, \ldots$ the translation of $p^G, E, \ldots$ using the environment T.

The translation of an index term ι such that $\Gamma \vdash \iota : [1, L]$ is an integer $\iota^T \in \{1, \ldots, L^T\}$ defined as follows:

$$\iota^T = \begin{cases} i^T & \text{if } \iota = i \\ \phi^T(\iota_1^T, \ldots, \iota_h^T) & \text{if } \iota = \phi(\iota_1, \ldots, \iota_h) \end{cases}$$

The translation of a clause term p^G is defined as follows:

$$\begin{aligned} (x_{\iota_1, \ldots, \iota_h})^T &= x_{\iota_1^T, \ldots, \iota_h^T} \\ f(p_1^G, \ldots, p_n^G)^T &= f(p_1^{GT}, \ldots, p_n^{GT}) \\ a_{\iota_1, \ldots, \iota_h}^{L_1, \ldots, L_h}[p_1^G, \ldots, p_n^G]^T &= a_{\iota_1^T, \ldots, \iota_h^T}^{L_1^T, \ldots, L_h^T}[p_1^{GT}, \ldots, p_n^{GT}] \\ \mathit{list}(i \leq L, p^G)^T &= \langle p^{GT[i \mapsto 1]}, \ldots, p^{GT[i \mapsto L^T]} \rangle \end{aligned}$$

The symbol $x_{\iota_1^T,\dots,\iota_h^T}$ is considered as a variable x; the symbol $a^{L_1^T,\dots,L_h^T}_{\iota_1^T,\dots,\iota_h^T}$ is considered as a name function symbol a. (There is a different symbol for each value of the indices $\iota_1^T,\dots,\iota_h^T$ and bounds $L_1^T,\dots,L_h^T$.) The translation of $\mathit{list}(i \leq L, p^G)$ is a list of length L^T.

Given a conjunction $\mathcal{C} = \bigwedge_{(i_1,\dots,i_h)\in[1,L_1]\times\dots\times[1,L_h]}$ and an environment T, we define the set of environments $T^{\mathcal{C}} = \{T[i_1 \mapsto v_1,\dots,i_h \mapsto v_h] \mid v_j \in \{1,\dots,L_j^T\}$ for $j = 1,\dots,h\}$: these environments map the indices i_j of the conjunction to all their possible values in $\{1,\dots,L_j^T\}$, and map all other values like T.

The translation of a fact $F^G = \mathcal{C}\,\mathit{pred}(p_1^G,\dots,p_l^G)$ is

$$(\mathcal{C}\,\mathit{pred}(p_1^G,\dots,p_l^G))^T = F_1 \wedge \dots \wedge F_k$$

where $\{F_1,\dots,F_k\} = \{\mathit{pred}(p_1^{GT'},\dots,p_l^{GT'}) \mid T' \in T^{\mathcal{C}}\}$ and $(F_1^G \wedge \dots \wedge F_n^G)^T = F_1^{GT} \wedge \dots \wedge F_n^{GT}$.

The translation of a set of equations $\mathcal{E}$ is the set $\mathcal{E}^T$ obtained by translating the equations as follows:

- $(\mathcal{C}\,p^G \doteq p'^G)^T = \{p^{GT'} = p'^{GT'} \mid T' \in T^{\mathcal{C}}\}$.
- $\mathcal{E}^T = \bigcup_{E\in\mathcal{E}} E^T$.

Given a set of equations $\{p_1 = p'_1,\dots,p_n = p'_n\}$ over standard clause terms, we define as usual its most general unifier $\textsc{mgu}\,(\{p_1 = p'_1,\dots,p_n = p'_n\})$ as a most general substitution σ such that $\sigma p_i = \sigma p'_i$ for all $i \in \{1,\dots,n\}$, $\mathit{dom}(\sigma) \cup \mathit{fv}(\mathit{im}(\sigma)) \subseteq \mathit{fv}(p_1,p'_1,\dots,p_n,p'_n)$, and $\mathit{dom}(\sigma) \cap \mathit{fv}(\mathit{im}(\sigma)) = \emptyset$, where $\mathit{fv}(p)$ designates the (free) variables of p, $\mathit{dom}(\sigma)$ is the domain of σ: $\mathit{dom}(\sigma) = \{x \mid \sigma x \neq x\}$, and $\mathit{im}(\sigma)$ is the image of σ: $\mathit{im}(\sigma) = \{\sigma x \mid \sigma x \neq x\}$. We denote by $\{x_1 \mapsto p_1,\dots,x_n \mapsto p_n\}$ the substitution that maps x_i to p_i for all $i = 1,\dots,n$.

Finally, we define the translation of the generalized Horn clause $R^G = H^G \wedge \mathcal{E} \Rightarrow \mathit{pred}(p_1^G,\dots,p_l^G)$ as follows. If the unification of $\mathcal{E}^T$ fails, then R^{GT} is undefined. Otherwise, $R^{GT} = \textsc{mgu}\,(\mathcal{E}^T)H^{GT} \Rightarrow \textsc{mgu}\,(\mathcal{E}^T)\mathit{pred}(p_1^{GT},\dots,p_l^{GT})$.

When $\mathcal{R}^G$ is a set of well-typed generalized Horn clauses (i.e., a set of pairs of a type environment Γ and a clause R^G such that $\Gamma \vdash R^G$), we define $\mathcal{R}^{G\mathcal{T}} = \{R^{GT} \mid \Gamma \vdash R^G \in \mathcal{R}^G,\ T$ is an environment for $\Gamma \vdash R^G$ and R^{GT} is defined$\}$, the set of all Horn clauses corresponding to clauses in $\mathcal{R}^G$.

4 Motivation

4.1 Running Example

As a running example, we consider a simple protocol based on the SOAP extension to XML signatures [8]. SOAP envelopes are XML documents with a mandatory Body together with an optional Header. The Body may contain a request, response or a fault message. The Header contains information about the message: in particular, the SOAP header can carry a digital signature, as follows:

```
<Envelope>
  <Header>
    <Signature>
      <SignedInfo>
        <Reference URI="#theBody">
          <DigestValue> hash of the body </DigestValue>
        </Reference>
        <Reference URI="#x1">
          <DigestValue> hash of the content of x1
          </DigestValue>
        </Reference>
        ...
      </SignedInfo>
      <SignatureValue>
        signature of SignedInfo with key sk_C
      </SignatureValue>
    </Signature>
  </Header>
  <Body Id="#theBody"> request </Body>
</Envelope>
```

The Signature header contains two components. The first component is a SignedInfo element: it contains a list of references to the elements of the message that are signed. Each reference is designated by its identifier and carries a DigestValue, a hash of the corresponding content. This hash may be computed with the hash function SHA-1. The second component of the Signature header is the signature of the SignedInfo element with a secret key sk_C.

We consider a simple protocol in which a client C sends such a document to a server S. The server processes the document and checks the signature before authorizing the request given in the Body: if the SignedInfo contains a Reference to an element with tag Body and the content of this element is the request, then he will authorize the request.

4.2 Need for a New Process Calculus

In order to model this protocol, we suppose that the XML parser parses the SOAP envelope as a pair. The first component is a list of triplets (tag, id, corresponding content) and the second component is the content of the body (that is, the request). The list in the first component is useful to retrieve the content of an element from its id by looking up the list. The content of the Signature header is modeled as a pair (*SignedInfo*, *SignatureValue*). *SignedInfo* is a list of pairs containing an id and the hash of the corresponding content. *SignatureValue* is the signature of *SignedInfo* with a secret key sk_C.

We would like to model this running example with the process calculus introduced in Section 2.1. However since the length of the header and the length of the list of references of the signature can be different from a document to another, we encounter several problems.

First, since the receiver of the SOAP envelope accepts messages containing any number of headers, we need lists of variable length in order to model the expected message. We solve this problem by adding a new construct to the syntax of terms: $list(i \leq L, M_i)$ for the list of elements M_i with index i in the set $\{1, \dots, L\}$.

Suppose now to have the following process let $list(i \leq L, y_i) = x$ in P else $\mathbf{0}$: we would like to bind y_i $(i \leq L)$ to the element of the list x, without knowing the length of the list in advance. To do this, we introduce a new construct choose L in P that chooses non-deterministically a bound L and then executes P.

Hence the beginning of the process P_S, that describes the receiver of the SOAP envelope, will be:

$P_S :=$ in(c, x).choose L in
 let $(list(j \leq L, (tag_j, id_j, cont_j)), w) = x$ in
 ...

Next, the server has to check the signature, before authorizing the request he receives. He has to verify that the list contains a tag tag_k equal to Signature and that $cont_k$ contains a correct signature. In other words, the server has to choose a k and test whether tag_k is equal to Signature and $cont_k$ contains a correct signature. We introduce a new process choose $k \leq L$ in P that chooses non-deterministically an index $k \in \{1, \dots, L\}$ and then executes P. This construct allows us to handle protocols that treat elements of lists non-uniformly: we can in fact perform a look-up in a list.

Hence, we can represent the beginning of the check of the signature as:

...
choose $k \leq L$ in
if $tag_k =$ Signature then
let $(sinfo, sinfosign) = cont_k$ in
...

We will give the final representation of this protocol with the new process calculus in Section 5.2.

Suppose now, that we want to model the following simple message between L participants A_i, with $i = 1, \dots, L$ and a chair of the communication C:

$$A_i \rightarrow C : a_i$$

Since we have L participants we would like to describe each participant with a process A_i and replicate A_i L times. Moreover we may need to create L identifiers a_i, each for one participant A_i. We solve these two issues by introducing two new constructs: $\Pi_{i \leq L} P$ and (for all $i \leq L, \nu a_i)P$. The first represents L copies of P running in parallel; the second creates L names $a_1, \dots, a_L$ and then executes P. Such components appears when modeling of groups protocols, such as the Asokan-Ginzboorg protocol [2].

Finally, suppose to apply a destructor $g(r_i, s_i)$ to each element y_i of a list $list(i \leq L, y_i)$. Since L is not fixed we cannot model this destructor application as let $y_1 = g(r_1, s_1)$ in ... let $y_L = g(r_L, s_L)$ in P else Q ... else Q. Hence we introduce a new destructor application let for all $i_1 \leq L_1, \dots, i_h \leq L_h, x_{i_1,\dots,i_h} = g(M_1, \dots, M_n)$ in P else Q: it tries to evaluate $g(M_1, \dots, M_n)$ for each $i_1 \in \{1, \dots, L_1\}, \dots, i_h \in \{1, \dots, L_h\}$; if this succeeds, then $x_{i_1,\dots,i_h}$ is bound to the result and P is executed, else Q is executed. This construct allows us to perform a map on the list: the destructor g is in fact applied to all the elements of the list.

5 Generalized Process Calculus

This section formally defines the syntax and the semantics of the new process calculus that we introduce to represent protocols with lists of unbounded lengths. We will refer to this new process calculus as *generalized process calculus*.

5.1 Syntax and Type System

The syntax of the generalized process calculus is described in Figure 2. Terms are enriched with several new constructs. Variables may have indices $x_{\iota_1,\dots,\iota_h}$, and so do names a_ι. We use the construct $list(i \leq L, M^G)$ to represent lists of variable length L. Lists of fixed length are represented by a data constructor $\langle M_1^G, \dots, M_n^G \rangle$ for each length n. We use $\widetilde{i}$ to represent a tuple of indices $i_1, \dots, i_h$, and we use the notation $x_{\widetilde{i}}$ for $x_{i_1,\dots,i_h}$ and $list(\widetilde{i} \leq \widetilde{L}, M^G)$ for $list(i_1 \leq L_1, list(i_2 \leq L_2, \dots, list(i_h \leq L_h, M^G) \dots))$.

Processes are also enriched with new constructs:

- The indexed replication $\Pi_{i \leq L} P^G$ represents L copies of P^G in parallel. It may represent L participants of a group protocol, where L is not fixed.
- The restriction (for all $i \leq L, \nu a_i) P^G$ creates L names $a_1, \dots, a_L$ and then executes P^G. The names $a_1, \dots, a_L$ may for instance be a secret key for each member of a group of L participants.
- The destructor application let for all $i_1 \leq L_1, \dots, i_h \leq L_h, x_{i_1,\dots,i_h} = g(M_1^G, \dots, M_n^G)$ in P^G else Q^G tries to evaluate $g(M_1^G, \dots, M_n^G)$ for each $i_1 \in \{1, \dots, L_1\}, \dots, i_h \in \{1, \dots, L_h\}$; if this succeeds, then $x_{i_1,\dots,i_h}$ is bound to the result and P^G is executed, else Q^G is executed.
- The pattern matching let for all $i_1 \leq L_1, \dots, i_h \leq L_h, pat^G = M^G$ in P^G else Q^G matches M^G with the pattern pat^G for each $i_1 \in \{1, \dots, L_1\}, \dots, i_h \in \{1, \dots, L_h\}$ and executes P^G when the matching succeeds, Q^G otherwise. The pattern pat^G can be a variable $x_{i_1,\dots,i_h}$, a data constructor application $f(pat_1^G, \dots, pat_h^G)$, or a list of variable length $list(i \leq L, pat^G)$; the latter pattern is essential to be able to decompose lists without fixing their length, since we do not have destructors to perform this decomposition. When a variable occurs in the pattern pat^G in the process let for all $i_1 \leq L_1, \dots, i_{h'} \leq L_{h'}, list(i_{h'+1} \leq L_{h'+1}, \dots list(i_h \leq L_h, pat^G) \dots) = M^G$ in P^G else Q^G, its indices must be $i_1, \dots, i_h$. Patterns are linear.
- The bound choice choose L in P^G chooses non-deterministically a bound L and then executes P^G. For example, in the process choose L in let $list(i \leq$

$\iota ::=$	index terms
i	index variable
$\phi(\iota_1, \dots, \iota_h)$	function application
$M^G, N^G ::=$	terms
$x_{\iota_1,\dots,\iota_h}$	variable ($h \geq 0$)
$f(M_1^G, \dots, M_n^G)$	function application
a	name
a_ι	indexed name
$\mathit{list}(i \leq L, M^G)$	list constructor
$\mathit{pat}^G :=$	patterns
$x_{i_1,\dots,i_h}$	variable
$f(\mathit{pat}_1^G, \dots, \mathit{pat}_n^G)$	data constructor
$\mathit{list}(i \leq L, \mathit{pat}^G)$	list pattern
$P^G, Q^G ::=$	processes
$\mathsf{out}(M^G, N^G).P^G$	output
$\mathsf{in}(M^G, x).P^G$	input
$\mathbf{0}$	nil
$P^G \mid Q^G$	parallel composition
$!P^G$	replication
$\Pi_{i \leq L} P^G$	indexed replication
$(\nu a)P^G$	restriction
$(\mathsf{for\ all}\ i \leq L, \nu a_i)P^G$	restriction
$\mathsf{let\ for\ all}\ i_1 \leq L_1, \dots, i_h \leq L_h, x_{i_1,\dots,i_h} = g(M_1^G, \dots, M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G$	destructor application
$\mathsf{let\ for\ all}\ i_1 \leq L_1, \dots, i_h \leq L_h, \mathit{pat}^G = M^G\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G$	pattern matching
$\mathsf{event}(e(M^G)).P^G$	event
$\mathsf{choose}\ L\ \mathsf{in}\ P^G$	bound choice
$\mathsf{choose}\ k \leq L\ \mathsf{in}\ P^G$	index choice
$\mathsf{choose}\ \phi : [1, L_1] \times \dots \times [1, L_h] \to [1, L']\ \mathsf{in}\ P^G$	function choice

Fig. 2. Syntax of the generalized process calculus

$L, y_i) = x$ in P^G else $\mathbf{0}$, the non-deterministic choice of the bound L allows us to bind y_i ($i \leq L$) to the elements of the list x, without knowing the length of the list in advance.

- The index choice choose $k \leq L$ in P^G chooses non-deterministically an index $k \in \{1, \dots, L\}$ and then executes P^G. In particular, this construct allows us to perform a lookup in a list. For example, let $\mathit{list}(i \leq L, x_i) = z$ in choose $k \leq L$ in if $f(x_k) = M^G$ then P^G else $\mathbf{0}$ looks for an element x_k of the list z such that $f(x_k) = M^G$.
- The function choice choose $\phi : [1, L_1] \times \dots \times [1, L_h] \to [1, L']$ in P^G chooses non-deterministically an index function $\phi : \{1, \dots, L_1\} \times \dots \times \{1, \dots, L_h\} \to \{1, \dots, L\}$ and then executes P^G. For instance, this construct allows us to verify that the elements of a list are a subset of the elements of another list,

by non-deterministically choosing the mapping between the indices of the two lists, as we do in Section 5.2.

We will use the notation for all $\widetilde{i} \leq \widetilde{L}$ instead of for all $i_1 \leq L_1, \ldots, i_h \leq L_h$, and simply omit "for all " when $h = 0$. As for the process calculus of Section 2.1, we can encode the process if for all $i_1 \leq L_1, \ldots, i_h \leq L_h, M^G = N^G$ then P^G else Q^G in the generalized process calculus as let $x = equal(list(\widetilde{i} \leq \widetilde{L}, M^G), list(\widetilde{i} \leq \widetilde{L}, N^G))$ in P^G else Q^G, where x does not occur in P^G. The "else" branches can be omitted when they are "else **0**".

We also define a simple type system for the generalized process calculus, to guarantee that the indices of all variables vary in the appropriate interval. In the type system, the type environment Γ is a list of type declarations:

- $i : [1, L]$ means that i is of type $[1, L]$, that is, intuitively, the value of index i can vary between 1 and the value of the bound L;
- $\phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L]$ means that the function ϕ expects as input h indices of types $[1, L_j]$, for $j = 1, \ldots, h$ and computes an index of type $[1, L]$;
- $x\_ : [1, L_1] \times \cdots \times [1, L_h]$ means that the variable x expects indices of types $[1, L_1], \ldots, [1, L_h]$; we write $x\_ : [\,]$ when x expects no index (that is, $h = 0$);
- $a\_ : [1, L]$ means that the name a expects an index of type $[1, L]$, and $a\_ : [\,]$ means that the name a expects no index.

The type system defines the judgment $\Gamma \vdash P^G$, which means that P^G is well-typed in the type environment Γ. The type rules are detailed in Appendix A.

We have notions of bound indices i, functions ϕ, variables x, names a, and bounds L. For example, the index k is bound in P^G in the process choose $k \leq L$ in P^G. In the pattern matching let for all $i_1 \leq L_1, \ldots, i_h \leq L_h, pat^G = M^G$ in P^G else Q^G, the indices $i_1, \ldots, i_h$ are bound in $pat^G = M^G$, but not in P^G or Q^G. The bound L is bound in P^G in the process choose L in P^G. A closed process has no free bounds, indices, functions, and variables. It may have free names.

We suppose that all processes are well-typed. A closed process P_0^G is well-typed as follows: $\Gamma_0 \vdash P^G$ where $\Gamma_0 = \{a\_ : [\,] \mid a \in fn(P^G)\}$.

5.2 Example

The representation of the protocol introduced in Section 4.1 in our process calculus is given in Figure 3. As explained in Section 4.2, we represent an XML message as a pair, containing as first component a list of triplets (tag, identifier, corresponding content) and as second component the content of the body. The client process P_C first executes an event $b(\mathsf{Req})$, meaning that he starts the protocol with a request Req. Then he builds his message and sends it on the public channel c. We suppose that the only element signed by the client is the Body. Since the receiver of the SOAP envelope accepts messages containing any number of headers, we need lists of variable length in order to model the expected message. This is why we model a generic XML message

$$
\begin{array}{ll}
P_C := & \mathsf{event}(b(\mathsf{Req})).\mathsf{out}(c, (\langle\langle(\mathsf{Signature}, \mathsf{ids}, (\langle\langle(\mathsf{idb}, \mathit{sha1}(\mathsf{Req}))\rangle\rangle, \\
& \quad \mathit{sign}((\langle\langle(\mathsf{idb}, \mathit{sha1}(\mathsf{Req}))\rangle\rangle, \mathsf{sk}_C)))), (\mathsf{Body}, \mathsf{idb}, \mathsf{Req})\rangle\rangle, \mathsf{Req}))) \\
P_S := & \mathsf{in}(c, x).\mathsf{choose}\ L\ \mathsf{in} \\
& \mathsf{let}\ (\mathit{list}(j \le L, (\mathit{tag}_j, \mathit{id}_j, \mathit{cont}_j)), w) = x\ \mathsf{in} \\
& \mathsf{choose}\ k \le L\ \mathsf{in} \\
& \mathsf{if}\ \mathit{tag}_k = \mathsf{Signature}\ \mathsf{then} \\
& \mathsf{let}\ (\mathit{sinfo}, \mathit{sinfosign}) = \mathit{cont}_k\ \mathsf{in} \\
& \mathsf{let}\ z = \mathit{checksign}(\mathit{sinfosign}, \mathit{pk}_C, \mathit{sinfo})\ \mathsf{in} \\
& \mathsf{choose}\ L'\ \mathsf{in}\ \mathsf{choose}\ \phi : [1, L'] \to [1, L]\ \mathsf{in} \\
& \mathsf{if}\ \mathit{sinfo} = \mathit{list}(l \le L', (\mathit{id}_{\phi(l)}, \mathit{sha1}(\mathit{cont}_{\phi(l)})))\ \mathsf{then} \\
& \mathsf{choose}\ d \le L'\ \mathsf{in} \\
& \mathsf{if}\ \mathit{tag}_{\phi(d)} = \mathsf{Body}\ \mathsf{then}\ \mathsf{if}\ \mathit{cont}_{\phi(d)} = w\ \mathsf{then}\ \mathsf{event}(e(w)) \\
P := & (\nu \mathsf{sk}_C)\mathsf{let}\ \mathit{pk}_C = \mathit{pk}(\mathsf{sk}_C)\ \mathsf{in}\ \mathsf{out}(c, \mathit{pk}_C).(!P_C \mid !P_S)
\end{array}
$$

Fig. 3. Representation of our running example

as $(\mathit{list}(j \le L, (\mathit{tag}_j, \mathit{id}_j, \mathit{cont}_j)), w)$, where tag_j, id_j, and cont_j are variables representing tags, identifiers, and contents respectively and w is the variable for the body. Therefore, the server process P_S receives on channel c the document x consisting of $\mathit{list}(j \le L, (\mathit{tag}_j, \mathit{id}_j, \mathit{cont}_j))$ together with the body w. Then he looks for an element with tag $\mathit{tag}_k = \mathsf{Signature}$ and tries to match the corresponding content cont_k to $(\mathit{sinfo}, \mathit{sinfosign})$, where $\mathit{sinfosign}$ is the signature of sinfo under the secret key sk_C. He checks that sinfo is a list of references to elements of the message $\mathit{list}(l \le L', (\mathit{id}_{\phi(l)}, \mathit{sha1}(\mathit{cont}_{\phi(l)})))$, and that in this list, there is an element with tag $\mathit{tag}_{\phi(d)} = \mathsf{Body}$ and with content $\mathit{cont}_{\phi(d)}$ equal to the content of the body w. When all checks succeed, he authorizes the request w, which is modeled by the event $e(w)$. Our goal is to show that, if the server authorizes a request w, then the client has sent this request, that is, if event $e(w)$ is executed, then event $b(w)$ has been executed.

5.3 Semantics

We define the semantics of a generalized process by translating it into a corresponding standard process. To define this translation, we need an environment that gives a value to each free bound, index, and index function of the process.

Definition 2 *Given a generalized process $\Gamma \vdash P^G$, an environment T for $\Gamma \vdash P^G$ is a function that associates:*

- *to each bound L free in P^G or that appears in Γ an integer L^T;*
- *to each index i such that $i : [1, L] \in \Gamma$, an index $i^T \in \{1, \ldots, L^T\}$;*
- *to each index function ϕ such that $\phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L] \in \Gamma$, a function $\phi^T : \{1, \ldots, L_1^T\} \times \cdots \times \{1, \ldots, L_h^T\} \to \{1, \ldots, L^T\}$.*

In order to abbreviate notations, we write:

- $T[\widetilde{i} \mapsto \widetilde{v}]$ instead of $T[i_1 \mapsto v_1, \ldots, i_h \mapsto v_h]$;

- $T[\widetilde{i} \mapsto \widetilde{1}]$ instead of $T[i_1 \mapsto 1, \ldots, i_h \mapsto 1]$;
- $T[\widetilde{i} \mapsto \widetilde{L}^T]$ instead of $T[i_1 \mapsto L_1^T, \ldots, i_h \mapsto L_h^T]$;
- $\widetilde{v} \leq \widetilde{L}^T$ instead of $\forall j \in \{1, \ldots, h\}, v_j \in \{1, \ldots, L_j^T\}$;
- $\widetilde{i} : \widetilde{L}$ instead of $i_1 : [1, L_1], \ldots, i_h : [1, L_h]$;
- $x\_ : \widetilde{L}$ instead of $x\_ : [1, L_1] \times \cdots \times [1, L_h]$;
- $\bigwedge_{\widetilde{i} \in \widetilde{L}}$ instead of $\bigwedge_{(i_1, \ldots, i_h) \in [1, L_1] \times \cdots \times [1, L_h]}$.

Given an environment T for $\Gamma \vdash P^G$, the generalized process P^G is translated into the standard process P^{GT} defined as follows. The translation ι^T of an index term ι is defined exactly as in Section 3.2. The translation M^{GT} of a term M^G is defined as follows:

$$\begin{aligned}
(x_{\iota_1, \ldots, \iota_h})^T &= x_{\iota_1^T, \ldots, \iota_h^T} \\
f(M_1^G, \ldots, M_n^G)^T &= f(M_1^{GT}, \ldots, M_n^{GT}) \\
a^T &= a \\
a_\iota^T &= a_{\iota^T} \\
\mathit{list}(i \leq L, M^G)^T &= \langle M^{GT[i \mapsto 1]}, \ldots, M^{GT[i \mapsto L^T]} \rangle
\end{aligned}$$

The translation of $\mathit{list}(i \leq L, M^G)$ is a list of length L^T. Patterns pat^G are translated exactly in the same way as terms M^G.

Finally the translation of a generalized process is defined as follows and explained below.

- $(\mathsf{out}(M^G, N^G).P^G)^T = \mathsf{out}(M^{GT}, N^{GT}).P^{GT}$.
- $(\mathsf{in}(M^G, x).P^G)^T = \mathsf{in}(M^{GT}, x).P^{GT}$.
- $\mathbf{0}^T = \mathbf{0}$.
- $(P^G \mid Q^G)^T = P^{GT} \mid Q^{GT}$.
- $(!P^G)^T = !P^{GT}$.
- $(\Pi_{i \leq L} P^G)^T = P^{GT[i \mapsto 1]} \mid \cdots \mid P^{GT[i \mapsto L^T]}$.
- $((\nu a)P^G)^T = (\nu a)P^{GT}$.
- $((\text{for all } i \leq L, \nu a_i)P^G)^T = (\nu a_1^{L^T}) \ldots (\nu a_{L^T}^{L^T})P^{GT}$.
- $(\text{let for all } \widetilde{i} \leq \widetilde{L}, x_{\widetilde{i}} = g(M_1^G, \ldots M_n^G) \text{ in } P^G \text{ else } Q^G)^T = \text{let } E_1 \text{ in } \ldots \text{ let } E_l \text{ in } P^T \text{ else } Q^T \ \ldots \text{ else } Q^T$, where $\{E_1, \ldots E_l\} = \{x_{\widetilde{i}}^{T'} = g(M_1^{GT'}, \ldots, M_n^{GT'}) \mid T' = T[\widetilde{i} \mapsto \widetilde{v}], \widetilde{v} \leq \widetilde{L}^T\}$.
- $(\text{let for all } \widetilde{i} \leq \widetilde{L}, \mathit{pat}^G = M^G \text{ in } P^G \text{ else } Q^G)^T = \text{let } E_1 \text{ in } \ldots \text{ let } E_l \text{ in } P^T \text{ else } Q^T \ldots \text{ else } Q^T$, where $\{E_1, \ldots E_l\} = \{\mathit{pat}^{GT'} = M^{GT'} \mid T' = T[\widetilde{i} \mapsto \widetilde{v}], \widetilde{v} \leq \widetilde{L}^T\}$.
- $(\mathsf{event}(e(M^G)).P^G)^T = \mathsf{event}(e(M^{GT})).P^{GT}$.
- $(\text{choose } L \text{ in } P^G)^T = P^{GT[L \mapsto 1]} + \cdots + P^{GT[L \mapsto n]} + \cdots$.
- $(\text{choose } k \leq L \text{ in } P^G)^T = P^{GT[k \mapsto 1]} + \cdots + P^{GT[k \mapsto L^T]}$.
- $(\text{choose } \phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L'] \text{ in } P^G)^T = P^{GT[\phi \mapsto \phi_1]} + \cdots + P^{GT[\phi \mapsto \phi_l]}$, where $\{\phi_1, \ldots, \phi_l\} = \{\phi \mid \phi : \{1, \ldots, L_1^T\} \times \cdots \times \{1, \ldots, L_h^T\} \to \{1, \ldots, L^T\}\}$.

In most cases, a construct of the generalized process calculus is translated into the corresponding construct of the standard process calculus. The translation of $(\mathsf{for\ all}\ i \leq L, \nu a_i)P^G$ creates L^T names and then executes P^{GT}. The translation of the process $\mathsf{let}\ \widetilde{i} \leq \widetilde{L}, x_{\widetilde{i}} = g(M_1^G, \dots, M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G$ computes $g(M_1^G, \dots, M_h^G)$ and stores it in $x_{\widetilde{i}}$, for all values of the indices $\widetilde{i}$. We define the translation of the pattern matching similarly. The choice processes are translated into a non-deterministic choice between all the values that L, i, or ϕ can assume. The translation of the process $\mathsf{choose}\ L\ \mathsf{in}\ P^G$ is an infinite process: this translation cannot be handled by ProVerif and our work solves this problem.

When P^G is a closed process, it can be translated in the empty environment, which we denote by T_0.

6 Translation into Generalized Horn Clauses

As for the standard process calculus, we define the translation of the generalized process calculus into generalized Horn clauses, by giving the clauses for the attacker and those for the protocol.

Clauses for the Attacker. The clauses for the attacker are the same as in ProVerif, that is, the clauses $\mathrm{att}(a[\,])$ for each $a \in S$ and the clauses (Rn), (Rf), (Rg), (Rl), (Rs), except that the following two clauses for lists are added:

$$\textstyle\bigwedge_{i \in [1,M]} \mathrm{att}(x_i) \Rightarrow \mathrm{att}(\mathit{list}(j \leq M, x_j)) \qquad \text{(Rf-list)}$$

$$\mathrm{att}(\mathit{list}(j \leq M, x_j)) \Rightarrow \mathrm{att}(x_i) \qquad \text{(Rg-list)}$$

and the clauses (Rf) and (Rg) for lists of fixed length $\langle \cdots \rangle$ are removed. (The two clauses above are sufficient for all lists.)

Clauses for the Protocol. The protocol is represented by a closed process P_0^G. To compute the clauses, we assume that the bound names in P_0^G have been renamed so that they are pairwise distinct and distinct from free names of P_0^G.

Next, we instrument the process P_0^G by labeling each replication $!P^G$ with a distinct session identifier s, so that it becomes $!^s P^G$, and labeling each restriction $(\mathsf{for\ all}\ i \leq L, \nu a_i)$ with the clause term that corresponds to the fresh name a_i, $a_{i,i_1,\dots,i_h}^{L,L_1,\dots,L_h}[x_1, \dots, x_n, s_1, \dots, s_{n'}]$, where $x_1, \dots, x_n$ are the variables that store the messages received in inputs above $(\mathsf{for\ all}\ i \leq L, \nu a_i)$ in P_0^G, $s_1, \dots, s_{n'}$ are the session identifiers that label replications above $(\mathsf{for\ all}\ i \leq L, \nu a_i)$ in the instrumentation of P_0^G and $i_1, \dots, i_h$ and $L_1, \dots, L_h$ are the indices that label indexed replications above $(\mathsf{for\ all}\ i \leq L, \nu a_i)$ in P_0^G. The construct (νa) is instrumented in the same way, so that it becomes $(\nu a : a_{i_1,\dots,i_h}^{L_1,\dots,L_h}[x_1, \dots, x_n, s_1, \dots, s_{n'}])$. We denote the instrumentation of P_0^G by $\mathrm{instr}^G(P_0^G)$.

The translation $[\![P^G]\!]\rho^G H^G \mathcal{E} \Gamma$ of a well-typed instrumented process $\Gamma_P \vdash P^G$ is a set of clauses, where the environment ρ^G is a mapping that associates each name and variable, possibly with indices, to a clause term, H^G is a sequence of facts $\mathrm{message}(\cdot, \cdot)$ and $\mathrm{m\text{-}event}(\cdot)$, $\mathcal{E}$ is a set of equations, and Γ is a type environment for generalized Horn clauses such that:

- $\Gamma \vdash H^G$;
- $\Gamma \vdash \mathcal{E}$;
- $\Gamma_P, \Gamma \vdash \rho^G$: for each mapping $x_{\widetilde{i}} \mapsto p^G$ in ρ^G, if $\Gamma_P \vdash x\_ : \widetilde{L}$ then $\Gamma, \widetilde{i} : \widetilde{L} \vdash p^G$, for each mapping $a \mapsto p^G$ in ρ^G, then $\Gamma \vdash p^G$, for each mapping $a_i \mapsto p^G$, if $\Gamma_P \vdash a\_ : [1, L]$ then $\Gamma, i : [1, L] \vdash p^G$, and for each declaration $i : [1, L] \in \Gamma_P$ (resp. $\phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L] \in \Gamma_P$) we have $i : [1, L] \in \Gamma$ (resp. $\phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L] \in \Gamma$).

The mapping ρ^G is then extended into a substitution that maps terms M^G to clause terms $p^G = \rho^G(M^G)$, by replacing each name and variable with the corresponding clause term, as follows:

$$\begin{aligned}
\rho^G(x_{\widetilde{\iota}}) &= p^G\{\widetilde{\iota}/\widetilde{i}\} \text{ if } \rho^G(x_{\widetilde{i}}) = p^G \\
\rho^G(f(M_1^G, \dots, M_n^G)) &= f(\rho^G(M_1^G), \dots, \rho^G(M_n^G)) \\
\rho^G(a_\iota) &= p^G\{\iota/i\} \text{ if } \rho^G(a_i) = p^G \\
\rho^G(\mathit{list}(i \leq L, M^G)) &= \mathit{list}(i \leq L, \rho^G(M^G)) \text{ if } i \notin \mathit{fi}(\mathit{im}(\rho^G))
\end{aligned}$$

The side condition $i \notin \mathit{fi}(\mathit{im}(\rho))$ in the last formula can be guaranteed by renaming i if needed; it avoids the capture of bound indices.

- $[\![\mathsf{out}(M^G, N^G).P^G]\!]\rho^G H^G \mathcal{E} \Gamma =$
  $[\![P^G]\!]\rho^G H^G \mathcal{E} \Gamma \cup \{\Gamma \vdash H^G \wedge \mathcal{E} \Rightarrow \text{message}(\rho^G(M^G), \rho^G(N^G))\}$.
- $[\![\mathsf{in}(M^G, x).P^G]\!]\rho^G H^G \mathcal{E} \Gamma =$
  $[\![P^G]\!](\rho^G[x \mapsto x])(H^G \wedge \text{message}(\rho^G(M^G), x))\mathcal{E}(\Gamma, x\_ : [\,])$.
- $[\![\mathbf{0}]\!]\rho^G H^G \mathcal{E} \Gamma = \emptyset$.
- $[\![P^G \mid Q^G]\!]\rho^G H^G \mathcal{E} \Gamma = [\![P^G]\!]\rho^G H^G \mathcal{E} \Gamma \cup [\![Q^G]\!]\rho^G H^G \mathcal{E} \Gamma$.
- $[\![!^s P^G]\!]\rho^G H^G \mathcal{E} \Gamma = [\![P^G]\!](\rho^G[s \mapsto s])H^G \mathcal{E} \Gamma$.
- $[\![\Pi_{i \leq L} P^G]\!]\rho^G H^G \mathcal{E} \Gamma = [\![P^G]\!]\rho^G H^G \mathcal{E}(\Gamma, i : [1, L])$.
- $[\![(\nu a : a_{\widetilde{i}}^{\widetilde{L}}[x_1, \dots, x_n, s_1, \dots, s_{n'}])P^G]\!]\rho^G H^G \mathcal{E} \Gamma = [\![P^G]\!](\rho^G[a \mapsto a_{\widetilde{i}}^{\widetilde{L}}[\rho^G(x_1), \dots, \rho^G(x_n), \rho^G(s_1), \dots, \rho^G(s_{n'})]\,])H^G \mathcal{E} \Gamma$.
- $[\![(\mathsf{for\ all}\ i \leq L, \nu a_i : a_{i,\widetilde{i}}^{L,\widetilde{L}}[x_1, \dots, x_n, s_1, \dots, s_{n'}])P^G]\!]\rho^G H^G \mathcal{E} \Gamma =$
  $[\![P^G]\!](\rho^G[a_i \mapsto a_{i,\widetilde{i}}^{L,\widetilde{L}}[\rho^G(x_1), \dots, \rho^G(x_n), \rho^G(s_1), \dots, \rho^G(s_{n'})]\,])H^G \mathcal{E} \Gamma$.
- $[\![\mathsf{let\ for\ all}\ \widetilde{i} \leq \widetilde{L}, x_{\widetilde{i}} = g(M_1^G, \dots M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G]\!]\rho^G H^G \mathcal{E} \Gamma =$
  $[\![Q^G]\!]\rho^G H^G \mathcal{E} \Gamma \cup [\![P^G]\!](\rho^G[x_{\widetilde{i}} \mapsto p'^G])H^G(\mathcal{E} \cup \mathcal{E}')\Gamma'$,
  where p'^G, $\mathcal{E}'$, and Γ' are defined as follows. Let $g(p_1, \dots, p_n) \to p$ be the rewrite rule in $\mathit{def}(g)$. The rewrite rule $g(p_1'^G, \dots, p_n'^G) \to p'^G$ is obtained from $g(p_1, \dots, p_n) \to p$ by replacing all variables y of this rule with fresh variables with indices $\widetilde{i}$: $y'_{\widetilde{i}}$. Then $\mathcal{E}' = \{\bigwedge_{\widetilde{i} \leq \widetilde{L}} p_1'^G \doteq \rho^G(M_1^G), \dots, \bigwedge_{\widetilde{i} \leq \widetilde{L}} p_n'^G \doteq \rho^G(M_n^G)\}$ and Γ' is Γ extended with $x\_ : \widetilde{L}$ and $y'\_ : \widetilde{L}$ for each variable $y'_{\widetilde{i}}$ in $p_1'^G, \dots, p_n'^G, p'^G$.
- $[\![\mathsf{let\ for\ all}\ \widetilde{i} \leq \widetilde{L}, \mathit{pat}^G = M^G\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G]\!]\rho^G H^G \mathcal{E} \Gamma = [\![Q^G]\!]\rho^G H^G \mathcal{E} \Gamma \cup [\![P^G]\!](\rho^G[x_{\widetilde{i}'} \mapsto x_{\widetilde{i}'} \mid x_{\widetilde{i}'} \text{ occurs in } \mathit{pat}^G])H^G(\mathcal{E} \cup \{\bigwedge_{\widetilde{i} \leq \widetilde{L}} \mathit{pat}^G \doteq \rho^G(M^G)\})\Gamma'$, where Γ' is Γ extended for the variables in pat^G.

- $[\![\mathsf{event}(e(M^G)).P^G]\!]\rho^G H^G \mathcal{E}\Gamma = [\![P^G]\!]\rho^G(H^G \wedge \text{m-event}(e(\rho^G(M^G))))\mathcal{E}\Gamma \cup \{\Gamma \vdash H^G \wedge \mathcal{E} \Rightarrow \text{event}(e(\rho^G(M^G)))\}$.
- $[\![\mathsf{choose}\ L\ \mathsf{in}\ P^G]\!]\rho^G H^G \mathcal{E}\Gamma = [\![P^G]\!]\rho^G H^G \mathcal{E}\Gamma$.
- $[\![\mathsf{choose}\ k \leq L\ \mathsf{in}\ P^G]\!]\rho^G H^G \mathcal{E}\Gamma = [\![P^G]\!]\rho^G H^G \mathcal{E}(\Gamma, k : [1, L])$.
- $[\![\mathsf{choose}\ \phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L']\ \mathsf{in}\ P^G]\!]\rho^G H^G \mathcal{E}\Gamma =$
  $[\![P^G]\!]\rho^G H^G \mathcal{E}(\Gamma, \phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L'])$.

In most cases, the translation is similar to the one of the standard process calculus. The translation of the process $(\mathsf{for\ all}\ i \leq L, \nu a_i : a^{L,\widetilde{L}}_{i,\widetilde{i}}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}])P^G$ extends ρ^G to the name a_i for all possible values of $i \in \{1, \ldots, L\}$. The translation of the destructor application $\mathsf{let\ for\ all}\ \widetilde{i} \leq \widetilde{L}, x_{\widetilde{i}} = g(M_1^G, \ldots M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G$ is the union of the clauses for the case where the destructor succeeds and for the case where it fails. In particular, when the destructor succeeds, instead of performing unification, we add the equations $\rho(M_i^G) = p_i'^G$ for every $\widetilde{i} \leq \widetilde{L}$ to $\mathcal{E}$ and extend ρ^G to the variable $x_{\widetilde{i}}$. We define the translation of the pattern matching similarly. Finally, the type environment Γ is extended with the chosen index or function in the choice processes and in the indexed replication; this is sufficient since the chosen bound, index, or function can take any value in the generalized Horn clauses.

Summary. Let $\rho_0 = \{a \mapsto a[\,] \mid a \in \mathit{fn}(P_0^G)\}$. The set of generalized Horn clauses corresponding to the closed process P_0^G is defined as:

$$\mathcal{R}^G_{P_0^G,S} = [\![\mathrm{instr}^G(P_0^G)]\!]\rho_0\emptyset\emptyset\Gamma_0 \cup \{\mathrm{att}(a[\,]) \mid a \in S\} \cup \{(\mathrm{Rn}), (\mathrm{Rf}), (\mathrm{Rg}), (\mathrm{Rl}), (\mathrm{Rs}), (\text{Rf-list}), (\text{Rg-list})\}$$

where S is the set of names initially known by the attacker.

For example, by translating the process P_S of our running example, we obtain the following clause:

$$\begin{aligned}&\mathrm{message}(c, x) \wedge \{s \doteq pk(\mathsf{sk}_C), pk_C \doteq s, (list(j \leq L, (tag_j, id_j, cont_j)), w) \doteq x,\\ &tag_k \doteq \mathsf{Signature}, cont_k \doteq (sinfo, sinfosign), sinfosign \doteq sign(v, y),\\ &sinfo \doteq v, pk_C \doteq pk(y), sinfo \doteq list(l \leq L', (id_{\phi(l)}, sha1(cont_{\phi(l)}))),\\ &tag_{\phi(d)} \doteq \mathsf{Body}, cont_{\phi(d)} \doteq w\} \Rightarrow \mathrm{event}(e(w))\end{aligned}$$

which means that the server process P_S executes event $e(w)$ when it has received a message x that satisfies all the checks. This clause will be simplified by the resolution algorithm presented in [7].

7 Soundness of the Generalized Horn Clauses

In this section, we relate the generalized Horn clauses generated from a closed well-typed generalized process $\Gamma_0 \vdash P_0^G$, to the Horn clauses generated from $P_0^{GT_0}$, to show that our generated Horn clauses are correct. The proofs of the

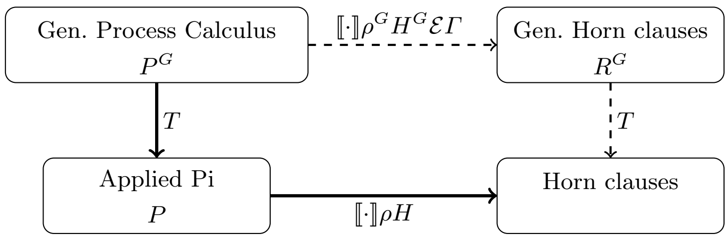

Fig. 4. Basic idea of Theorem 3

results of this section can be found in Appendix B. We assume that the bound names in P_0^G have been renamed so that they are pairwise distinct and distinct from free names of P_0^G.

The bound names in $P_0^{GT_0}$ need not be pairwise distinct, so we first need to rename them, before generating the Horn clauses. Hence, we define a function Tren that combines the translation P^{GT} with that renaming of bound names.

Definition 3 *Given a well-typed generalized process $\Gamma \vdash P^G$, an environment T for $\Gamma \vdash P^G$, and a list of indices $\widetilde{i} \leq \widetilde{L}$, let* Tren *be defined by:*

- $\mathrm{Tren}(\Pi_{i \leq L} P^G, T, \widetilde{i} \leq \widetilde{L}) = \mathrm{Tren}(P^G, T[i \mapsto 1], (\widetilde{i}, i) \leq (\widetilde{L}, L)) \mid \cdots \mid \mathrm{Tren}(P^G, T[i \mapsto L^T], (\widetilde{i}, i) \leq (\widetilde{L}, L))$;
- $\mathrm{Tren}((\nu a)P^G, T, \widetilde{i} \leq \widetilde{L}) = (\nu a_{\widetilde{i}^T}^{\widetilde{L}^T})\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L})\{a_{\widetilde{i}^T}^{\widetilde{L}^T}/a\}$;
- $\mathrm{Tren}((\textsf{for all } i \leq L, \nu a_i)P^G, T, \widetilde{i} \leq \widetilde{L}) = (\nu a_{1,\widetilde{i}^T}^{L^T,\widetilde{L}^T}) \ldots (\nu a_{L^T,\widetilde{i}^T}^{L^T,\widetilde{L}^T})\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L})\{a_{1,\widetilde{i}^T}^{L^T,\widetilde{L}^T}/a_1, \ldots, a_{L^T,\widetilde{i}^T}^{L^T,\widetilde{L}^T}/a_{L^T}\}$;
- *In all other cases,* $\mathrm{Tren}(P^G, T, \leq \widetilde{L})$ *is defined like P^{GT} except that it recursively calls* $\mathrm{Tren}(P'^G, T, \leq \widetilde{L})$ *instead of P'^{GT} on the subprocesses. For instance,* $\mathrm{Tren}(\textsf{choose } k \leq L \textsf{ in } P^G, T, \widetilde{i} \leq \widetilde{L}) = \mathrm{Tren}(P^G, T[k \mapsto 1], \widetilde{i} \leq \widetilde{L}) + \cdots + \mathrm{Tren}(P^G, T[k \mapsto L^T], \widetilde{i} \leq \widetilde{L})$.

Lemma 1. $\mathrm{Tren}(P_0^G, T_0, \emptyset \leq \emptyset)$ *is a suitable renaming of $P_0^{GT_0}$.*

We say that R_1 subsumes R_2 when R_2 can be obtained by adding hypotheses to an instance of R_1. In this case, all facts derivable using R_2 can also be derived by R_1, so R_2 can be eliminated. Formally, subsumption is defined by:

Definition 4 (Subsumption) *We say that $R_1 = H_1 \Rightarrow C_1$ subsumes $R_2 = H_2 \Rightarrow C_2$, and we write $R_1 \sqsupseteq R_2$, if and only if there exists a substitution σ such that $\sigma C_1 = C_2$ and $\sigma H_1 \subseteq H_2$ (multiset inclusion).*

We extend subsumption to sets of clauses as follows. Let $\mathcal{R}_1, \mathcal{R}_2$ be two sets of Horn clauses. We say that $\mathcal{R}_1 \sqsupseteq \mathcal{R}_2$ if for every clause $R_2 \in \mathcal{R}_2$, there exists a clause $R_1 \in \mathcal{R}_1$ such that $R_1 \sqsupseteq R_2$.

The following theorem shows the soundness of our generalized Horn clauses. Its main idea is summarized in Figure 4.

Theorem 3. *Let $\Gamma_0 \vdash P_0^G$ be a closed well-typed generalized process, and S be a set of names. Let $P'_0 = \mathrm{Tren}(P_0^G, T_0, \emptyset \leq \emptyset)$. We have $\mathcal{R}^{G\mathcal{T}}_{P_0^G,S} \sqsupseteq \mathcal{R}_{P'_0,S}$.*

Furthermore, if $\mathcal{R}_1 \sqsupseteq \mathcal{R}_2$ and a fact F is derivable from $\mathcal{R}_1$, then it is also derivable from $\mathcal{R}_2$. So, by Theorems 1, 2, and 3 and Lemma 1, we obtain the following results.

Corollary 1 (Secrecy). *Let M^G be a term. Let p^G be the clause term obtained by replacing names a with $a[\,]$ and names a_i with $a_i[\,]$ in M^G. If for all environments T, $\mathrm{att}(p^{GT})$ is not derivable from $\mathcal{R}^{G\mathcal{T}}_{P_0^G,S} \cup \mathcal{F}_{\mathrm{me}}$ for any $\mathcal{F}_{\mathrm{me}}$, then for all environments T, $P_0^{GT_0}$ preserves the secrecy of M^{GT} from adversaries with initial knowledge S.*

Corollary 2 (Authentication). *Suppose that, for all $\mathcal{F}_{\mathrm{me}}$, for all p, if $\mathrm{event}(e(p))$ is derivable from $\mathcal{R}^{G\mathcal{T}}_{P_0^G,S} \cup \mathcal{F}_{\mathrm{me}}$, then $\text{m-event}(e'(p)) \in \mathcal{F}_{\mathrm{me}}$. Then $P_0^{GT_0}$ satisfies the correspondence "if $e(x)$ has been executed, then $e'(x)$ has been executed" against adversaries with initial knowledge S.*

The hypotheses of these two corollaries are precisely those that can be proved using the resolution algorithm we developed in [7], as shown by [7, Corollaries 3 and 4]. In particular, in [7] we prove the soundness of the resolution algorithm for generalized Horn clauses by considering their translation into Horn clauses. So by combining the results of this paper with [7], we can prove secrecy and authentication for protocols that use lists of any length.

For example, after translating our running example into generalized Horn clauses, we can run the tool developed in [7] and obtain that the hypothesis of Corollary 2 holds for events e and b. Therefore, by Corollary 2, the process of Section 5.2 satisfies the desired correspondence: if $e(x)$ is executed, then $b(x)$ has been executed.

8 Conclusion

We have proposed a new process calculus, useful to represent protocols that manipulate lists of unbounded length. We have defined its semantics and provided an automatic translation from this calculus into generalized Horn clauses. We have proved that this translation is sound. By combining these results with [7], we obtain an automatic technique for proving secrecy and authentication properties of protocols that manipulate unbounded lists, for an unbounded number of sessions, represented in a process calculus. Implementing the translation into generalized Horn clauses is planned for future work. We also plan to adapt to the generalized process calculus the result presented in [6] on observational equivalences between processes.

Acknowledgments. This work was partly supported by the ANR project ProSe (decision number ANR-2010-VERS-004-01).

References

1. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: POPL'01. pp. 104–115. ACM Press, London, United Kingdom (Jan 2001)
2. Asokan, N., Ginzboorg, P.: Key agreement in ad hoc networks. Computer Communications 23(17), 1627–1637 (2000)
3. Backes, M., Mödersheim, S., Pfitzmann, B., Viganò, L.: Symbolic and cryptographic analysis of the secure WS-ReliableMessaging scenario. In: Aceto, L., Ingólfsdóttir, A. (eds.) FoSSaCS'06. LNCS, vol. 3921, pp. 428–445. Springer (2006)
4. Bhargavan, K., Fournet, C., Gordon, A.D., Pucella, R.: TulaFale: A security tool for web services. In: FMCO'03. LNCS, vol. 3188, pp. 197–222. Springer (2004)
5. Blanchet, B.: Automatic verification of correspondences for security protocols. Journal of Computer Security 17(4), 363–434 (Jul 2009)
6. Blanchet, B., Abadi, M., Fournet, C.: Automated verification of selected equivalences for security protocols. Journal of Logic and Algebraic Programming 75(1), 3–51 (Feb–Mar 2008)
7. Blanchet, B., Paiola, M.: Automatic verification of protocols with lists of unbounded length. In: CCS'13. ACM (Nov 2013), to appear.
8. Brown, A., Fox, B., Hada, S., LaMacchia, B., Maruyama, H.: SOAP security extensions: Digital signature, available at `http://www.w3.org/TR/SOAP-dsig/`
9. Bryans, J., Schneider, S.: CSP, PVS and recursive authentication protocol. In: DIMACS Workshop on Formal Verification of Security Protocols (Sep 1997)
10. Chridi, N., Turuani, M., Rusinowitch, M.: Decidable analysis for a class of cryptographic group protocols with unbounded lists. In: CSF'09. pp. 277–289. IEEE Computer Society (2009)
11. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Transactions on Information Theory IT-29(12), 198–208 (Mar 1983)
12. Kleiner, E., Roscoe, A.W.: On the relationship between web services security and traditional protocols. In: MFPS 21. Electronic Notes in Theoretical Computer Science, vol. 155, pp. 583–603 (2006)
13. Küsters, R., Truderung, T.: On the automatic analysis of recursive security protocols with XOR. In: Thomas, W., Weil, P. (eds.) STACS'07. LNCS, vol. 4393, pp. 646–657. Springer (2007)
14. Paiola, M., Blanchet, B.: Verification of security protocols with lists: from length one to unbounded length. In: Degano, P., Guttman, J.D. (eds.) POST'12. LNCS, vol. 7215, pp. 69–88. Springer (2012)
15. Paulson, L.C.: Mechanized proofs for a recursive authentication protocol. In: CSFW'97. pp. 84–95. IEEE Computer Society Press (1997)
16. Roscoe, A.W., Kleiner, E.: Web Services Security: a preliminary study using Casper and FDR. In: Automated Reasoning for Security Protocol Analysis (2004)
17. Truderung, T.: Selecting theories and recursive protocols. In: CONCUR 2005. LNCS, vol. 3653, pp. 217–232. Springer (2005)

Appendix

A Type System

The type rules are given in Figure 5. The type system defines the following judgments:

$$\frac{i:[1,L]\in\Gamma}{\Gamma\vdash i:[1,L]}$$

$$\frac{\phi:[1,L_1]\times\cdots\times[1,L_h]\to[1,L]\in\Gamma \qquad \Gamma\vdash\iota_1:[1,L_1]\dots\Gamma\vdash\iota_h:[1,L_h]}{\Gamma\vdash\phi(\iota_1,\dots,\iota_h):[1,L]}$$

$$\frac{x\_:[1,L_1]\times\cdots\times[1,L_h]\in\Gamma \qquad \Gamma\vdash\iota_1:[1,L_1]\dots\Gamma\vdash\iota_h:[1,L_h]}{\Gamma\vdash x_{\iota_1,\dots,\iota_h}}(\mathrm{Var})$$

$$\frac{\Gamma\vdash M_1^G\dots\Gamma\vdash M_n^G}{\Gamma\vdash f(M_1^G,\dots,M_n^G)}$$

$$\frac{a\_:[\,]\in\Gamma}{\Gamma\vdash a} \qquad \frac{a\_:[1,L]\in\Gamma \qquad \Gamma\vdash\iota:[1,L]}{\Gamma\vdash a_\iota} \qquad \frac{\Gamma,i:[1,L]\vdash M^G}{\Gamma\vdash \mathit{list}(i\le L,M^G)}$$

$$\frac{\Gamma\vdash M^G \qquad \Gamma\vdash N^G \qquad \Gamma\vdash P^G}{\Gamma\vdash \mathsf{out}(M^G,N^G).P^G}$$

$$\frac{\Gamma\vdash M^G \qquad \Gamma,x\_:[\,]\vdash P^G}{\Gamma\vdash \mathsf{in}(M^G,x).P^G}$$

$$\Gamma\vdash\mathbf{0} \qquad \frac{\Gamma\vdash P^G \qquad \Gamma\vdash Q^G}{\Gamma\vdash P^G\mid Q^G} \qquad \frac{\Gamma\vdash P^G}{\Gamma\vdash !P^G}$$

$$\frac{\Gamma,i:[1,L]\vdash P^G}{\Gamma\vdash \Pi_{i\le L}P^G} \qquad \frac{\Gamma,a\_:[\,]\vdash P^G}{\Gamma\vdash(\nu a)P^G} \qquad \frac{\Gamma,a\_:[1,L]\vdash P^G}{\Gamma\vdash(\mathsf{for\ all}\ i\le L,\nu a_i)P^G}$$

$$\frac{\begin{array}{cc}\Gamma,i_1:[1,L_1],\dots,i_h:[1,L_h]\vdash M_1^G & \\ \dots & \Gamma,x\_:[1,L_1]\times\cdots\times[1,L_h]\vdash P^G \\ \Gamma,i_1:[1,L_1],\dots,i_h:[1,L_h]\vdash M_n^G & \Gamma\vdash Q^G\end{array}}{\mathsf{let\ for\ all}\ i_1\le L_1,\dots,i_h\le L_h, x_{i_1,\dots,i_h}=g(M_1^G,\dots M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G}$$

$$i_1:[1,L_1],\dots,i_h[1,L_h]\vdash x_{i_1,\dots,i_h}\rightsquigarrow(x\_:[1,L_1]\times\cdots\times[1,L_h])(\mathrm{PatVar})$$

$$\frac{\text{for all } j\le n\text{, we have } i_1:[1,L_1],\dots,i_h:[1,L_h]\vdash \mathit{pat}_j^G\rightsquigarrow\Gamma_j}{i_1:[1,L_1],\dots,i_h:[1,L_h]\vdash f(\mathit{pat}_1^G,\dots,\mathit{pat}_n^G)\rightsquigarrow\Gamma_1,\dots,\Gamma_n}(\mathrm{PatData})$$

$$\frac{i_1:[1,L_1],\dots,i_h:[1,L_h],i:[1,L]\vdash \mathit{pat}^G\rightsquigarrow\Gamma}{i_1:[1,L_1],\dots,i_h:[1,L_h]\vdash \mathit{list}(i\le L,\mathit{pat}^G)\rightsquigarrow\Gamma}(\mathrm{PatList})$$

$$\frac{\begin{array}{cc}i_1:[1,L_1],\dots,i_h:[1,L_h]\vdash \mathit{pat}^G\rightsquigarrow\Gamma' & \Gamma,\Gamma'\vdash P^G \\ \Gamma,i_1:[1,L_1],\dots,i_h:[1,L_h]\vdash M^G & \Gamma\vdash Q^G\end{array}}{\mathsf{let\ for\ all}\ i_1\le L_1,\dots,i_h\le L_h,\mathit{pat}^G=M^G\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G}$$

$$\frac{\Gamma\vdash M^G \qquad \Gamma\vdash P^G}{\Gamma\vdash \mathsf{event}(e(M^G)).P^G} \qquad \frac{\Gamma\vdash P^G}{\Gamma\vdash \mathsf{choose}\ L\ \mathsf{in}\ P^G}$$

$$\frac{\Gamma,k:[1,L]\vdash P^G}{\Gamma\vdash \mathsf{choose}\ k\le L\ \mathsf{in}\ P^G} \qquad \frac{\Gamma,\phi:[1,L_1]\times\cdots\times[1,L_h]\to[1,L]\ \vdash P^G}{\Gamma\vdash \mathsf{choose}\ \phi:[1,L_1]\times\cdots\times[1,L_h]\to[1,L]\ \mathsf{in}\ P^G}$$

Fig. 5. Type system for the generalized process calculus

– $\Gamma \vdash \iota : [1, L]$, which means that ι has type $[1, L]$ in the type environment Γ;
– $\Gamma \vdash M^G$, $\Gamma \vdash P^G$, which mean that M^G, P^G, respectively, are well-typed in the type environment Γ.
– $i_1 : [1, L_1], \dots, i_h : [1, L_h] \vdash \mathit{pat}^G \rightsquigarrow \Gamma$, which means that the pattern pat^G has free indices $i_1, \dots, i_h$ of types $[1, L_1], \dots, [1, L_h]$ respectively, and binds the variables in Γ.

Most type rules are straightforward. For instance, the rule (Var) means that $x_{i_1,\dots,i_h}$ is well-typed when the types expected by x for its indices match the types of $i_1, \dots, i_h$. The rules (PatVar), (PatData), and (PatList) deal with the patterns $x_{i_1,\dots,i_h}, f(\mathit{pat}_1^G, \dots, \mathit{pat}_n^G)$, and $\mathit{list}(i \leq L, \mathit{pat}^G)$, respectively. They build the type environment that gives types to the variables bound in the pattern.

B Proofs

We write $P \equiv_\alpha Q$ when the process P is equal to Q up to renaming of bound names: in an instrumented process $(\nu a : a'[x_1, \dots, x_n, s_1, \dots, s_{n'}])P$, the name a can be renamed, but the function symbol a' remains unchanged. This is why we may end up with instrumented processes in which the name a is different from the function symbol a'.

B.1 Proof of Lemma 1

Lemma 1 is an immediate consequence of the following lemma.

Lemma 2. *Let $\Gamma \vdash P^G$ be a well-typed generalized process, such that the bound names of P^G are pairwise distinct and distinct from free names of P^G. Given an environment T for $\Gamma \vdash P^G$, and a list of indices $\widetilde{i} \leq \widetilde{L}$, we have:*

$$\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}) \equiv_\alpha P^{GT} .$$

Furthermore, we have the following two properties:

P1. The bound names in $\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L})$ are pairwise distinct and distinct from free names, except that in processes $P + Q$, the bound names in P need not be distinct from those in Q.

P2. All bound names in $\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L})$ are of the form $a^{\widetilde{L}^T,\dots}_{\widetilde{i}^T,\dots}$ when they come from (νa) in P^G and of the form $a^{L^T,\widetilde{L}^T,\dots}_{v,\widetilde{i}^T,\dots}$ when they come from (for all $i \leq L, \nu a_i$) in P^G.

Proof. The property $\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}) \equiv_\alpha P^{GT}$ is proved by an easy induction on the syntax of P^G.

Properties P1 and P2 are proved by simultaneous induction on the syntax of P^G.

- Case $\Pi_{i\leq L} P^G$: for each $v \leq L^T$, by induction hypothesis, the bound names in $\mathrm{Tren}(P^G, T[i \mapsto v], (\widetilde{i}, i) \leq (\widetilde{L}, L))$ are pairwise distinct (except that in processes $P + Q$, the bound names in P need not be distinct from those in Q) and distinct from free names. Furthermore, they are of the form $a^{\widetilde{L}^T, L^T, \dots}_{\widetilde{i}^T, v, \dots}$ when they come from (νa) in P^G and of the form $a^{L'^T, \widetilde{L}^T, L^T, \dots}_{v', \widetilde{i}^T, v, \dots}$ when they come from $(\mathsf{for\ all}\ i' \leq L', \nu a_{i'})$ in P^G, so P2 holds. Hence the names $\mathrm{Tren}(P^G, T[i \mapsto v], (\widetilde{i}, i) \leq (\widetilde{L}, L))$ are distinct from the names in $\mathrm{Tren}(P^G, T[i \mapsto v'], (\widetilde{i}, i) \leq (\widetilde{L}, L))$ when $v \neq v'$, so P1 holds.
- Case $(\mathsf{for\ all}\ i \leq L, \nu a_i) P^G$: by induction hypothesis, the bound names in $\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L})$ are pairwise distinct (except that in processes $P+Q$, the bound names in P need not be distinct from those in Q) and distinct from free names. Furthermore, they are of the form $a'^{\widetilde{L}^T, \dots}_{\widetilde{i}^T, \dots}$ when they come from $(\nu a')$ in P^G and of the form $a'^{L^T, \widetilde{L}^T, \dots}_{v, \widetilde{i}^T, \dots}$ when they come from $(\mathsf{for\ all}\ i \leq L, \nu a'_i)$ in P^G. The new bound names $a^{L^T, \widetilde{L}^T}_{v, \widetilde{i}^T}$ for $v \leq L^T$ are of the required form, so P2 holds. They are distinct from the free names and from the bound names of $\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L})$, since the bound names in $(\mathsf{for\ all}\ i \leq L, \nu a_i) P^G$ are pairwise distinct and distinct from free names, so they do not use the same symbol a. So P1 holds.
- The case $(\nu a) P^G$ is similar to the previous one. All other cases follow easily using the induction hypothesis. We use the property that the bound names of P^G are pairwise distinct and distinct from free names of P^G. In the cases "choose", we also use that in processes $P+Q$, the bound names in P need not be distinct from those in Q, so the induction hypothesis already guarantees that names are distinct when desired. □

B.2 Proof of Theorem 3

Theorem 3 comes from the combination of two different results. The first result (Lemma 4) shows that the translation from generalized processes to processes commutes with the instrumentation (provided the translation is suitably renamed using Tren). The second result (Lemma 10) shows the soundness of the translation from instrumented processes to generalized Horn clauses.

Instrumentation Before proving the first result, we define the instrumentation of processes and generalized processes more formally by induction on the syntax of the processes, as follows.

Definition 5 *Given a process P, a list of variables $Vars = x_1, \dots, x_n$, and a list of session identifiers $SessId = s_1, \dots, s_{n'}$, we define the instrumented process as follows:*

- $\mathrm{instr}(\mathsf{in}(M, x).P, Vars, SessId) = \mathsf{in}(M, x).\mathrm{instr}(P, (Vars, x), SessId)$;
- $\mathrm{instr}(!P, Vars, SessId) = !^s \mathrm{instr}(P, Vars, (SessId, s))$;

– $\mathrm{instr}((\nu a)P, \mathit{Vars}, \mathit{SessId}) = (\nu a : a[\mathit{Vars}, \mathit{SessId}])\mathrm{instr}(P, \mathit{Vars}, \mathit{SessId})$;
– *In all other cases, the same instrumentation is applied recursively on the subprocesses. For instance,* $\mathrm{instr}(P \mid Q, \mathit{Vars}, \mathit{SessId}) = \mathrm{instr}(P, \mathit{Vars}, \mathit{SessId}) \mid \mathrm{instr}(Q, \mathit{Vars}, \mathit{SessId})$.

We let $\mathrm{instr}(P) = \mathrm{instr}(P, \emptyset, \emptyset)$.

Definition 6 *Given a generalized process P^G, a list of variables $\mathit{Vars} = x_1, \dots, x_n$, a list of session identifiers $\mathit{SessId} = s_1, \dots, s_{n'}$, and a list of indices $\widetilde{i} \leq \widetilde{L}$, we define the instrumented generalized process as follows:*

– $\mathrm{instr}^G(\mathsf{in}(M^G, x).P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}) =$
  $\mathsf{in}(M^G, x).\mathrm{instr}^G(P^G, (\mathit{Vars}, x), \mathit{SessId}, \widetilde{i} \leq \widetilde{L})$;
– $\mathrm{instr}^G(!P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}) = !^s\mathrm{instr}^G(P^G, \mathit{Vars}, (\mathit{SessId}, s), \widetilde{i} \leq \widetilde{L})$;
– $\mathrm{instr}^G(\Pi_{i \leq L} P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}) =$
  $\Pi_{i \leq L}\mathrm{instr}^G(P^G, \mathit{Vars}, \mathit{SessId}, (\widetilde{i}, i \leq \widetilde{L}, L))$;
– $\mathrm{instr}^G((\mathsf{for\ all}\ i \leq L, \nu a_i)P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}) =$
  $(\mathsf{for\ all}\ i \leq L, \nu a_i : a_{i,\widetilde{i}}^{L,\widetilde{L}}[\mathit{Vars}, \mathit{SessId}])\mathrm{instr}^G(P, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L})$;
– $\mathrm{instr}^G((\nu a)P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}) =$
  $(\nu a : a_{\widetilde{i}}^{\widetilde{L}}[\mathit{Vars}, \mathit{SessId}])\mathrm{instr}^G(P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L})$;
– *In all other cases, the same instrumentation is applied recursively on the subprocesses. For instance,* $\mathrm{instr}^G(P^G \mid Q^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}) = \mathrm{instr}^G(P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}) \mid \mathrm{instr}^G(Q^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L})$.

We let $\mathrm{instr}^G(P^G) = \mathrm{instr}^G(P^G, \emptyset, \emptyset, \emptyset \leq \emptyset)$.

The translation P^{GT} on instrumented processes is defined similarly to the translation on non-instrumented processes; the cases that differ are as follows:

– $(!^s P^G)^T = !^s P^{GT}$
– $((\nu a : a'^{\widetilde{L}}_{\widetilde{i}}[x_1, \dots, x_n, s_1, \dots, s_{n'}])P^G)^T = (\nu a : a'^{\widetilde{L}^T}_{\widetilde{i}^T}[x_1, \dots, x_n, s_1, \dots, s_{n'}])P^{GT}$
– $((\mathsf{for\ all}\ i \leq L, \nu a_i : a'^{L,\widetilde{L}}_{i,\widetilde{i}}[x_1, \dots, x_n, s_1, \dots, s_{n'}])P^G)^T = (\nu a_1 : a'^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}[x_1, \dots, x_n, s_1, \dots, s_{n'}]) \dots (\nu a_{L^T} : a'^{L^T,\widetilde{L}^T}_{L^T,\widetilde{i}^T}[x_1, \dots, x_n, s_1, \dots, s_{n'}])P^{GT}$

Lemma 3. *Given a well-typed generalized process $\Gamma \vdash P^G$, an environment T for $\Gamma \vdash P^G$, a list of variables $\mathit{Vars} = x_1, \dots x_n$, a list of session identifiers $\mathit{SessId} = s_1, \dots, s_{n'}$, and a list of indices $\widetilde{i} \leq \widetilde{L}$, we have:*

$$(\mathrm{instr}^G(P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T \equiv_\alpha \mathrm{instr}(\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}), \mathit{Vars}, \mathit{SessId})\,.$$

Proof. This proof is done by structural induction on the process P^G. We detail here the most interesting cases.

– Case $\mathsf{in}(M^G, x).P^G$:

$$\begin{aligned}
&(\mathrm{instr}^G(\mathsf{in}(M^G, x).P^G,\ \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T \\
&\quad = (\mathsf{in}(M^G, x).\mathrm{instr}^G(P^G, (\mathit{Vars}, x), \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T \\
&\quad = \mathsf{in}(M^{GT}, x).(\mathrm{instr}^G(P^G, (\mathit{Vars}, x), \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T \\
&\quad \equiv_\alpha \mathsf{in}(M^{GT}, x).\mathrm{instr}(\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}), (\mathit{Vars}, x), \mathit{SessId}) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{by induction hypothesis} \\
&\quad \equiv_\alpha \mathrm{instr}(\mathsf{in}(M^{GT}, x).\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}),\ \mathit{Vars}, \mathit{SessId}) \\
&\quad \equiv_\alpha \mathrm{instr}(\mathrm{Tren}(\mathsf{in}(M^G, x).P^G, T, \widetilde{i} \leq \widetilde{L}),\ \mathit{Vars}, \mathit{SessId})
\end{aligned}$$

– Case $!P^G$:

$$\begin{aligned}
&(\mathrm{instr}^G(!P^G,\ \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T \\
&\quad = (!^s\mathrm{instr}^G(P^G,\ \mathit{Vars}, (\mathit{SessId}, s), \widetilde{i} \leq \widetilde{L}))^T \\
&\quad = !^s(\mathrm{instr}^G(P^G,\ \mathit{Vars}, (\mathit{SessId}, s), \widetilde{i} \leq \widetilde{L}))^T \\
&\quad \equiv_\alpha !^s\mathrm{instr}(\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}),\ \mathit{Vars}, (\mathit{SessId}, s)) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{by induction hypothesis} \\
&\quad \equiv_\alpha \mathrm{instr}(!\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}),\ \mathit{Vars}, \mathit{SessId}) \\
&\quad \equiv_\alpha \mathrm{instr}(\mathrm{Tren}(!P^G, T, \widetilde{i} \leq \widetilde{L}),\ \mathit{Vars}, \mathit{SessId})
\end{aligned}$$

– Case $\Pi_{i \leq L} P^G$:

$$\begin{aligned}
&(\mathrm{instr}^G(\Pi_{i \leq L} P^G,\ \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T \\
&\quad = \Pi_{i \leq L}(\mathrm{instr}^G(P^G,\ \mathit{Vars}, \mathit{SessId}, (\widetilde{i}, i) \leq (\widetilde{L}, L)))^T \\
&\quad = (\mathrm{instr}^G(P^G,\ \mathit{Vars}, \mathit{SessId}, (\widetilde{i}, i) \leq (\widetilde{L}, L)))^{T[i \mapsto 1]} \mid \cdots \mid \\
&\qquad\quad (\mathrm{instr}^G(P^G,\ \mathit{Vars}, \mathit{SessId}, (\widetilde{i}, i) \leq (\widetilde{L}, L)))^{T[i \mapsto L^T]}
\end{aligned}$$

For each $v \leq L^T$, we have by induction hypothesis:

$$\begin{aligned}
&\mathrm{instr}^G(P^G,\ \mathit{Vars}, \mathit{SessId}, (\widetilde{i}, i) \leq (\widetilde{L}, L))^{T[i \mapsto v]} \equiv_\alpha \\
&\quad \mathrm{instr}(\mathrm{Tren}(P^G, T[i \mapsto v], (\widetilde{i}, i) \leq (\widetilde{L}, L)),\ \mathit{Vars}, \mathit{SessId}).
\end{aligned}$$

Hence:

$$\begin{aligned}
&(\mathrm{instr}^G(\Pi_{i \leq L} P^G,\ \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T \\
&\quad \equiv_\alpha \mathrm{instr}(\mathrm{Tren}(P^G, T[i \mapsto 1], (\widetilde{i}, i) \leq (\widetilde{L}, L)),\ \mathit{Vars}, \mathit{SessId}) \mid \cdots \mid \\
&\qquad\quad \mathrm{instr}(\mathrm{Tren}(P^G, T[i \mapsto L^T], (\widetilde{i}, i) \leq (\widetilde{L}, L)),\ \mathit{Vars}, \mathit{SessId}) \\
&\quad \equiv_\alpha \mathrm{instr}(\mathrm{Tren}(P^G, T[i \mapsto 1], (\widetilde{i}, i) \leq (\widetilde{L}, L)) \mid \cdots \mid \\
&\qquad\qquad \mathrm{Tren}(P^G, T[i \mapsto L^T], (\widetilde{i}, i) \leq (\widetilde{L}, L)),\ \mathit{Vars}, \mathit{SessId}) \\
&\quad \equiv_\alpha \mathrm{instr}(\mathrm{Tren}(\Pi_{i \leq L} P^G, T, \widetilde{i} \leq \widetilde{L}),\ \mathit{Vars}, \mathit{SessId})
\end{aligned}$$

– Case $(\mathsf{for\ all}\ i \leq L, \nu a_i) P^G$:

$$\begin{aligned}
&(\mathrm{instr}^G((\mathsf{for\ all}\ i \leq L, \nu a_i) P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T \\
&\quad = ((\mathsf{for\ all}\ i \leq L, \nu a_i : a^{L,\widetilde{L}}_{i,\widetilde{i}}[\mathit{Vars}, \mathit{SessId}]) \\
&\qquad \mathrm{instr}^G(P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T \\
&\quad = (\nu a_1 : a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}[\mathit{Vars}, \mathit{SessId}]) \dots (\nu a_{L^T} : a^{L^T,\widetilde{L}^T}_{L^T,\widetilde{i}^T}[\mathit{Vars}, \mathit{SessId}]) \\
&\qquad (\mathrm{instr}^G(P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T
\end{aligned}$$

Moreover, by induction hypothesis, $(\mathrm{instr}^G(P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T \equiv_\alpha \mathrm{instr}(\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}), \mathit{Vars}, \mathit{SessId})$. Therefore,

$$\begin{aligned}
&(\mathrm{instr}^G((\mathsf{for\ all}\ i \leq L, \nu a_i) P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T \\
&\equiv_\alpha (\nu a_1 : a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}[\mathit{Vars}, \mathit{SessId}]) \dots (\nu a_{L^T} : a^{L^T,\widetilde{L}^T}_{L^T,\widetilde{i}^T}[\mathit{Vars}, \mathit{SessId}]) \\
&\quad \mathrm{instr}(\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}), \mathit{Vars}, \mathit{SessId}) \\
&\equiv_\alpha (\nu a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T} : a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}[\mathit{Vars}, \mathit{SessId}]) \dots (\nu a^{L^T,\widetilde{L}^T}_{L^T,\widetilde{i}^T} : a^{L^T,\widetilde{L}^T}_{L^T,\widetilde{i}^T}[\mathit{Vars}, \mathit{SessId}]) \\
&\quad (\mathrm{instr}(\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}), \mathit{Vars}, \mathit{SessId})\{a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}/a_1, \dots, a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}/a_{L^T}\}) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{by renaming bound names} \\
&\equiv_\alpha \mathrm{instr}((\nu a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}) \dots (\nu a^{L^T,\widetilde{L}^T}_{L^T,\widetilde{i}^T}) \mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L})\{a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}/a_1, \dots, \\
&\quad a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}/a_{L^T}\}, \mathit{Vars}, \mathit{SessId}) \\
&\equiv_\alpha \mathrm{instr}(\mathrm{Tren}((\mathsf{for\ all}\ i \leq L, \nu a_i) P^G, T, \widetilde{i} \leq \widetilde{L}), \mathit{Vars}, \mathit{SessId})
\end{aligned}$$

– The case $(\nu a) P^G$ can be handled similarly to the previous case. All other cases follow easily from the induction hypothesis. □

Lemma 4. *Given a well-typed generalized process $\Gamma_0 \vdash P^G_0$, we have:*

$$(\mathrm{instr}^G(P^G_0))^{T_0} \equiv_\alpha \mathrm{instr}(\mathrm{Tren}(P^G_0, T_0, \emptyset \leq \emptyset))\,.$$

Proof. This result comes immediately from Lemma 3 applied to $\mathrm{instr}^G(P^G_0) = \mathrm{instr}^G(P^G_0, \emptyset, \emptyset, \emptyset \leq \emptyset)$. □

Translation from Instrumented Processes to Clauses We use the following standard result.

Lemma 5. *Let $\mathcal{E}_1$, $\mathcal{E}_2$ be two sets of equations over standard clause terms. Then* $\textsc{mgu}\,(\mathcal{E}_1 \cup \mathcal{E}_2)$ *is defined if and only if* $\textsc{mgu}\,(\textsc{mgu}\,(\mathcal{E}_2)\mathcal{E}_1)\textsc{mgu}\,(\mathcal{E}_2)$ *is defined, and* $\textsc{mgu}\,(\mathcal{E}_1 \cup \mathcal{E}_2) = \textsc{mgu}\,(\textsc{mgu}\,(\mathcal{E}_2)\mathcal{E}_1)\textsc{mgu}\,(\mathcal{E}_2)$.

Lemma 6. *Let P be an instrumented process, ρ a function that associates a clause term with each name and variable, and H a sequence of facts. Given a substitution σ over the variables in ρ, we have that:*

$$[\![P]\!](\sigma\rho)(\sigma H) \sqsubseteq [\![P]\!]\rho H\,.$$

Proof. The proof of this lemma is done by structural induction on the process P. We detail here the most interesting cases.

– Case $M(x).P$:

$$\begin{aligned}
&[\![M(x).P]\!](\sigma\rho)(\sigma H)\\
&\quad= [\![P]\!]((\sigma\rho)[x \mapsto x])(\sigma H \wedge \text{message}(\sigma\rho(M), x))\\
&\quad= [\![P]\!](\sigma'(\rho[x \mapsto x]))(\sigma'(H \wedge \text{message}(\rho(M), x)))\\
&\qquad\qquad\qquad\qquad\text{where we define the substitution } \sigma' = \sigma[x \mapsto x]\\
&\quad\sqsubseteq [\![P]\!](\rho[x \mapsto x])(H \wedge \text{message}(\rho(M), x)) \qquad \text{by induction hypothesis}\\
&\quad\sqsubseteq [\![M(x).P]\!]\rho H
\end{aligned}$$

– Case $\mathsf{let}\ x = g(M_1, \dots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q$:

$$\begin{aligned}
&[\![\mathsf{let}\ x = g(M_1, \dots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!](\sigma\rho)(\sigma H)\\
&\quad= [\![Q]\!](\sigma\rho)(\sigma H) \cup \bigcup\{[\![P]\!](\sigma_1\sigma\rho[x \mapsto \sigma'_1 p'])(\sigma_1\sigma H) \mid g(p'_1, \dots, p'_n) \to p'\\
&\qquad\quad \text{is in } \mathit{def}(g) \text{ and } (\sigma_1, \sigma'_1) \text{ is a most general pair of substitutions}\\
&\qquad\quad \text{such that } \sigma_1\sigma\rho(M_i) = \sigma'_1 p'_i, \text{for each } i = 1, \dots n\}
\end{aligned}$$

By induction hypothesis, we have $[\![Q]\!](\sigma\rho)(\sigma H) \sqsubseteq [\![Q]\!]\rho H$. Let $g(p'_1, \dots, p'_n) \to p'$ be a rule in $\mathit{def}(g)$, and (σ_1, σ'_1) be a most general pair of substitutions such that $\sigma_1\sigma\rho(M_i) = \sigma'_1 p'_i$, for each $i = 1, \dots n$. Let $\sigma_2 = \sigma_1\sigma$ and $\sigma'_2 = \sigma'_1$. For each $i = 1, \dots, n$, we have $\sigma_2\rho(M_i) = \sigma'_2 p'_i$. Let (σ_3, σ'_3) be a most general pair of substitutions such that for each $i = 1, \dots, n$: $\sigma_3\rho(M_i) = \sigma'_3 p'_i$. As (σ_2, σ'_2) is such a pair (but maybe not a most general one), there exists a substitution σ_4 such that $\sigma_2 = \sigma_4\sigma_3$ and $\sigma'_2 = \sigma_4\sigma'_3$. Hence we have that

$$\begin{aligned}
&[\![P]\!](\sigma_1\sigma\rho[x \mapsto \sigma'_1 p'])(\sigma_1\sigma H)\\
&\quad= [\![P]\!](\sigma_4\sigma_3\rho[x \mapsto \sigma_4\sigma'_3 p'])(\sigma_4\sigma_3\sigma H)\\
&\quad= [\![P]\!](\sigma_4(\sigma_3\rho[x \mapsto \sigma'_3 p']))(\sigma_4(\sigma_3\sigma H))\\
&\quad\sqsubseteq [\![P]\!](\sigma_3\rho[x \mapsto \sigma'_3 p'])(\sigma_3\sigma H)
\end{aligned}$$

by induction hypothesis. Therefore,

$$\begin{aligned}
&[\![\mathsf{let}\ x = g(M_1, \dots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!](\sigma\rho)(\sigma H)\\
&\quad\sqsubseteq [\![Q]\!]\rho H \cup \bigcup\{[\![P]\!](\sigma_3\rho[x \mapsto \sigma'_3 p'])(\sigma_3\sigma H) \mid g(p'_1, \dots, p'_n) \to p'\\
&\qquad\quad \text{is in } \mathit{def}(g) \text{ and } (\sigma_3, \sigma'_3) \text{ is a most general pair of substitutions}\\
&\qquad\quad \text{such that } \sigma_3\rho(M_i) = \sigma'_3 p'_i, \text{for each } i = 1, \dots n\}\\
&\quad\sqsubseteq [\![\mathsf{let}\ x = g(M_1, \dots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]\rho H.
\end{aligned}$$

– The other cases are straightforward using the induction hypothesis. □

Lemma 7. *We have*

$$[\![\mathsf{let}\ E_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ E_l\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q]\!]\rho H \\ \sqsubseteq [\![Q]\!]\rho H \cup [\![P]\!](\textsc{mgu}\,(\mathcal{E})(\rho[x_1 \mapsto p'_1, \ldots, x_l \mapsto p'_l]))(\textsc{mgu}\,(\mathcal{E})H)$$

where

- *for each* $i \leq l$, E_i *is* $x_i = g_i(M_{i,1}, \ldots, M_{i,n_i})$;
- *for each* $i \leq l$, x_i *does not occur in* Q *nor in* $M_{k,j}$ *for all* $k = 1, \ldots, l$ *and* $j = 1, \ldots, n_k$;
- *for each* $i \leq l$, $g_i(p^0_{i,1}, \ldots, p^0_{i,n_i}) \rightarrow p'^0_i$ *is the rewriting rule of* g_i *and* $p_{i,1}, \ldots, p_{i,n_i}, p'_i$ *are obtained by renaming* $p^0_{i,1}, \ldots, p^0_{i,n_i}, p'^0_i$ *with fresh variables;*
- $\mathcal{E} = \{\rho(M_{k,j}) = p_{k,j} \mid k = 1, \ldots, l$ *and* $j = 1, \ldots, n_k\}$.

When the equations in $\mathcal{E}$ *cannot be unified,* $\textsc{mgu}\,(\mathcal{E})$ *is not defined, and the second component of the union is omitted.*

Proof. The proof is done by induction on l.

– Base case: $l = 1$.

$$[\![\mathsf{let}\ E_1\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]\rho H = [\![Q]\!]\rho H \cup [\![P]\!](\sigma\rho[x_1 \mapsto \sigma p'_1])(\sigma H)$$

where σ is a most general substitution such that $\sigma\rho(M_{1,j}) = \sigma p_{1,j}$ for each $j = 1, \ldots, n_1$, assuming that σ exists. (Finding such a σ is equivalent to finding a most general pair of substitutions (σ', σ'') such that $\sigma'\rho(M_{1,j}) = \sigma'' p^0_{1,j}$: we can define σ by $\sigma x = \sigma''\alpha^{-1}x$ where α is the renaming of $p^0_{i,j}$ into $p_{i,j}$ and x is a fresh variable introduced by this renaming, and $\sigma x = \sigma' x$ otherwise.) Hence $\sigma = \textsc{mgu}\,(\mathcal{E})$ where $\mathcal{E} = \{\rho(M_{1,j}) = p_{1,j} \mid j = 1, \ldots, n_1\}$ and we can conclude that

$$[\![\mathsf{let}\ E_1\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]\rho H = [\![Q]\!]\rho H \cup [\![P]\!](\textsc{mgu}\,(\mathcal{E})(\rho[x_1 \mapsto p'_1]))(\textsc{mgu}\,(\mathcal{E})H)$$

When $\textsc{mgu}\,(\mathcal{E})$ is not defined, that is, σ does not exist, the second component of the union is omitted.

– Inductive step. We have

$$\begin{aligned}
&[\![\mathsf{let}\ E_1\ \mathsf{in}\ \mathsf{let}\ E_2\ \mathsf{in}\ \ldots\ \mathsf{let}\ E_l\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q\ \mathsf{else}\ Q]\!]\rho H \\
&\quad = [\![Q]\!]\rho H \cup [\![\mathsf{let}\ E_2\ \mathsf{in}\ \ldots\ \mathsf{let}\ E_l\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q]\!]\rho_1 H_1 \\
&\qquad\quad \text{where } \rho_1 = \textsc{mgu}\,(\mathcal{E}_1)(\rho[x_1 \mapsto p'_1]),\ H_1 = \textsc{mgu}\,(\mathcal{E}_1)H, \text{ and} \\
&\qquad\quad \mathcal{E}_1 = \{\rho(M_{1,j}) = p_{1,j} \mid j = 1, \ldots, n_1\}, \text{ by the base case} \\
&\quad \sqsubseteq [\![Q]\!]\rho H \cup [\![Q]\!]\rho_1 H_1 \cup \\
&\qquad [\![P]\!](\textsc{mgu}\,(\mathcal{E}_2)(\rho_1[x_2 \mapsto p'_2, \ldots, x_l \mapsto p'_l]))(\textsc{mgu}\,(\mathcal{E}_2)H_1)
\end{aligned}$$

where $\mathcal{E}_2 = \{\rho_1(M_{k,j}) = p_{k,j} \mid k = 2,\dots,l \text{ and } j = 1,\dots,n_k\}$, by induction hypothesis, assuming that $\text{MGU}(\mathcal{E}_1)$ and $\text{MGU}(\mathcal{E}_2)$ are defined. We have $[\![Q]\!]\rho_1 H_1 = [\![Q]\!](\text{MGU}(\mathcal{E}_1)\rho)(\text{MGU}(\mathcal{E}_1)H)$ since x_1 does not occur in Q, so $[\![Q]\!]\rho_1 H_1 \sqsubseteq [\![Q]\!]\rho H$ by Lemma 6.
Let $\mathcal{E}'_2 = \{\rho(M_{k,j}) = p_{k,j} \mid k = 2,\dots,l \text{ and } j = 1,\dots,n_k\}$. The variables of $p_{k,j}$ ($k \geq 2$) are fresh, so they are untouched by $\text{MGU}(\mathcal{E}_1)$, so we have $\mathcal{E}_2 = \text{MGU}(\mathcal{E}_1)\mathcal{E}'_2$ and $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}'_2$, so

$$\text{MGU}(\mathcal{E}_2)\text{MGU}(\mathcal{E}_1) = \text{MGU}(\text{MGU}(\mathcal{E}_1)\mathcal{E}'_2)\text{MGU}(\mathcal{E}_1) = \text{MGU}(\mathcal{E}_1 \cup \mathcal{E}'_2) = \text{MGU}(\mathcal{E})$$

by Lemma 5. Moreover, the variables of $p'_2,\dots,p'_l$ are fresh, so they are untouched by $\text{MGU}(\mathcal{E}_1)$. Hence

$$\begin{aligned}&\text{MGU}(\mathcal{E}_2)(\rho_1[x_2 \mapsto p'_2,\dots,x_l \mapsto p'_l])\\ &\quad = \text{MGU}(\mathcal{E}_2)\text{MGU}(\mathcal{E}_1)(\rho[x_1 \mapsto p'_1, x_2 \mapsto p'_2,\dots,x_l \mapsto p'_l])\\ &\quad = \text{MGU}(\mathcal{E})(\rho[x_1 \mapsto p'_1,\dots,x_l \mapsto p'_l])\end{aligned}$$

and $\text{MGU}(\mathcal{E}_2)H_1 = \text{MGU}(\mathcal{E}_2)\text{MGU}(\mathcal{E}_1)H = \text{MGU}(\mathcal{E})H$. Therefore,

$$\begin{aligned}&[\![\mathsf{let}\ E_1\ \mathsf{in}\ \mathsf{let}\ E_2\ \mathsf{in}\ \dots\ \mathsf{let}\ E_l\ \mathsf{in}\ P\ \mathsf{else}\ Q\dots\ \mathsf{else}\ Q\ \mathsf{else}\ Q]\!]\rho H\\ &\quad \sqsubseteq [\![Q]\!]\rho H \cup [\![P]\!](\text{MGU}(\mathcal{E})(\rho[x_1 \mapsto p'_1,\dots,x_l \mapsto p'_l]))(\text{MGU}(\mathcal{E})H)\end{aligned}$$

As before, when $\text{MGU}(\mathcal{E})$ is not defined, that is, $\text{MGU}(\mathcal{E}_2)\text{MGU}(\mathcal{E}_1)$ is not defined, the second component of the union is omitted. □

From this lemma, we obtain the following result for the special case of the decomposition of data constructors.

Corollary 3. *Let f be a data constructor of arity n and f_1^{-1}, …, f_n^{-1} be its associated destructors.*

$$\begin{aligned}&[\![\mathsf{let}\ x_1 = f_1^{-1}(M)\ \mathsf{in}\ \dots\ \mathsf{let}\ x_n = f_n^{-1}(M)\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \dots\ \mathsf{else}\ Q]\!]\rho H\\ &\quad \sqsubseteq [\![Q]\!]\rho H \cup [\![P]\!](\text{MGU}(\mathcal{E})(\rho[x_1 \mapsto v_1,\dots,x_n \mapsto v_n]))(\text{MGU}(\mathcal{E})H)\end{aligned}$$

where $x_1,\dots,x_n$ do not occur in Q nor in M, and $\mathcal{E} = \{f(v_1,\dots,v_n) = \rho(M)\}$. When $\text{MGU}(\mathcal{E})$ is not defined, the second component of the union is omitted.

Proof. By Lemma 7, we obtain

$$\begin{aligned}&[\![\mathsf{let}\ x_1 = f_1^{-1}(M)\ \mathsf{in}\ \dots\ \mathsf{let}\ x_n = f_n^{-1}(M)\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \dots\ \mathsf{else}\ Q]\!]\rho H\\ &\quad \sqsubseteq [\![Q]\!]\rho H \cup [\![P]\!](\text{MGU}(\mathcal{E}')(\rho[x_1 \mapsto v_{1,1},\dots,x_n \mapsto v_{n,n}]))(\text{MGU}(\mathcal{E}')H)\end{aligned}$$

where $\mathcal{E}' = \{\rho(M) = f(v_{k,1},\dots,v_{k,n}) \mid k = 1,\dots,n\}$ and the variables $v_{k,j}$ ($k = 1,\dots,n$, $j = 1,\dots,n$) are fresh. We have $\text{MGU}(\mathcal{E}')v_{k,j} = \text{MGU}(\mathcal{E}')v_{k',j}$ for all k, k', j, so for all $j = 1,\dots,n$, we can rename the variables $v_{k,j}$ for all k into the same variable v_j. After this renaming, we obtain the announced result. □

Lemma 8. *Suppose that the variables of $pat_1, \dots, pat_n$ are pairwise distinct and fresh (that is, they do not occur in ρ, H, M_1, $\dots$, M_n, and Q).*

$$\begin{aligned}&[\![\mathsf{let}\ pat_1 = M_1\ \mathsf{in}\ \dots\ \mathsf{let}\ pat_n = M_n\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \dots\ \mathsf{else}\ Q]\!]\rho H\\ &\quad\sqsubseteq [\![Q]\!]\rho H \cup [\![P]\!](\textsc{mgu}\,(\mathcal{E})(\rho[x \mapsto x \mid x \text{ occurs in } pat_1, \dots, pat_n]))(\textsc{mgu}\,(\mathcal{E})H)\end{aligned}$$

where $\mathcal{E} = \{pat_i = \rho(M_i) \mid i = 1, \dots, n\}$.

Proof. The proof is done by induction on the total size of the patterns $pat_1, \dots, pat_n$.

– Case 1: there is a single pattern $pat = x$.

$$\begin{aligned}&[\![\mathsf{let}\ x = M\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]\rho H\\ &\quad= [\![\mathsf{let}\ x = id(M)\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]\rho H\\ &\quad= [\![Q]\!]\rho H \cup [\![P]\!](\textsc{mgu}\,(\{\rho(M) = y\})(\rho[x \mapsto y]))(\textsc{mgu}\,(\{\rho(M) = y\})H)\end{aligned}$$

where y is a fresh variable and the rewrite rule for destructor id is renamed into $id(y) \rightarrow y$ (see the base case of Lemma 7).

$$\quad\sqsubseteq [\![Q]\!]\rho H \cup [\![P]\!](\textsc{mgu}\,(\{\rho(M) = x\})(\rho[x \mapsto x]))(\textsc{mgu}\,(\{\rho(M) = x\})H)$$

by renaming x into y since x and y do not occur in ρ, $\rho(M)$, and H.

– Case 2: there is a single pattern $pat = f(pat_1, \dots, pat_n)$.

$$\begin{aligned}&[\![\mathsf{let}\ f(pat_1, \dots, pat_n) = M\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]\rho H\\ &\quad= [\![\mathsf{let}\ x_1 = f_1^{-1}(M)\ \mathsf{in}\ \dots\ \mathsf{let}\ x_n = f_n^{-1}(M)\ \mathsf{in}\\ &\qquad\mathsf{let}\ pat_1 = x_1\ \mathsf{in}\ \dots\ \mathsf{let}\ pat_n = x_n\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \dots\ \mathsf{else}\ Q\\ &\qquad\mathsf{else}\ Q\ \dots\ \mathsf{else}\ Q]\!]\rho H\end{aligned}$$

where $x_1, \dots, x_n$ are fresh variables

$$\begin{aligned}&\quad\sqsubseteq [\![Q]\!]\rho H \cup [\![\mathsf{let}\ pat_1 = x_1\ \mathsf{in}\ \dots\ \mathsf{let}\ pat_n = x_n\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \dots\ \mathsf{else}\ Q]\!]\\ &\qquad(\textsc{mgu}\,(\mathcal{E})(\rho[x_1 \mapsto v_1, \dots, x_n \mapsto v_n]))(\textsc{mgu}\,(\mathcal{E})H)\end{aligned}$$

where $\mathcal{E} = \{f(v_1, \dots, v_n) = \rho(M)\}$, by Corollary 3

$$\begin{aligned}&\quad\sqsubseteq [\![Q]\!]\rho H \cup [\![Q]\!]\rho' H' \cup\\ &\qquad[\![P]\!](\textsc{mgu}\,(\mathcal{E}')(\rho'[x \mapsto x \mid x \text{ occurs in } pat_1, \dots, pat_n]))(\textsc{mgu}\,(\mathcal{E}')H')\end{aligned}$$

where $\rho' = \textsc{mgu}\,(\mathcal{E})(\rho[x_1 \mapsto v_1, \dots, x_n \mapsto v_n])$, $H' = \textsc{mgu}\,(\mathcal{E})H$, and $\mathcal{E}' = \{pat_1 = \rho'(x_1), \dots, pat_n = \rho'(x_n)\}$, by induction hypothesis (since the total size of $pat_1, \dots, pat_n$ is less than the size of $f(pat_1, \dots, pat_n)$).
As $x_1, \dots, x_n$ do not appear in Q, $[\![Q]\!]\rho' H' = [\![Q]\!](\textsc{mgu}\,(\mathcal{E})\rho)(\textsc{mgu}\,(\mathcal{E})H) \sqsubseteq [\![Q]\!]\rho H$, by Lemma 6.
We have $\mathcal{E}' = \{pat_i = \textsc{mgu}\,(\mathcal{E})v_i \mid i = 1, \dots, n\} = \textsc{mgu}\,(\mathcal{E})\{pat_i = v_i \mid i = 1, \dots, n\}$, so by Lemma 5, $\textsc{mgu}\,(\mathcal{E}')\textsc{mgu}\,(\mathcal{E}) = \textsc{mgu}\,(\{f(v_1, \dots, v_n) = \rho(M)\} \cup \{pat_i = v_i \mid i = 1, \dots, n\}) = \textsc{mgu}\,(\{f(pat_1, \dots, pat_n) = \rho(M)\} \cup \{pat_i = v_i \mid i = 1, \dots, n\})$. Let $\mathcal{E}'' = \{f(pat_1, \dots, pat_n) = \rho(M)\}$. Then

we have $\text{MGU}(\mathcal{E}')\text{MGU}(\mathcal{E}) = (\text{MGU}(\mathcal{E}''))[v_i \mapsto \text{MGU}(\mathcal{E}'')pat_i]$. Therefore we obtain that:

$$\begin{aligned}&[\![\mathsf{let}\ f(pat_1,\ldots,pat_n) = M\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]\rho H \sqsubseteq [\![Q]\!]\rho H \\ &\quad \cup [\![P]\!](\text{MGU}(\mathcal{E}'')(\rho[x \mapsto x \mid x \text{ occurs in } pat_1,\ldots,pat_n]))(\text{MGU}(\mathcal{E}'')H)\end{aligned}$$

since the variables $v_1,\ldots,v_n$ do not occur in ρ and H, and the variables x_1, $\ldots, x_n$ can be removed from the environment since they do not occur in P.

– Case 3: there are several patterns.

$$\begin{aligned}&[\![\mathsf{let}\ pat_1 = M_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ pat_n = M_n\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q]\!]\rho H \\ &\quad \sqsubseteq [\![Q]\!]\rho H \cup [\![\mathsf{let}\ pat_2 = M_2\ \mathsf{in}\ \ldots\ \mathsf{let}\ pat_n = M_n\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q]\!] \\ &\qquad (\text{MGU}(\mathcal{E}_1)(\rho[x \mapsto x \mid x \text{ occurs in } pat_1]))(\text{MGU}(\mathcal{E}_1)H) \\ &\qquad \text{where } \mathcal{E}_1 = \{pat_1 = \rho(M_1)\}, \text{ by induction hypothesis applied to } pat_1 \\ &\quad \sqsubseteq [\![Q]\!]\rho H \cup [\![Q]\!]\rho' H' \cup \\ &\qquad [\![P]\!](\text{MGU}(\mathcal{E}_2)(\rho'[x \mapsto x \mid x \text{ occurs in } pat_2,\ldots,pat_n]))(\text{MGU}(\mathcal{E}_2)H')\end{aligned}$$

where $\rho' = \text{MGU}(\mathcal{E}_1)(\rho[x \mapsto x \mid x \text{ occurs in } pat_1])$, $H' = \text{MGU}(\mathcal{E}_1)H$, and $\mathcal{E}_2 = \{pat_i = \rho'(M_i) \mid i = 2,\ldots,n\}$, by induction hypothesis applied to pat_2, $\ldots, pat_n$.
Since the variables of pat_1 do not occur in the process Q, we have $[\![Q]\!]\rho' H' = [\![Q]\!](\text{MGU}(\mathcal{E}_1)\rho)(\text{MGU}(\mathcal{E}_1)H) \sqsubseteq [\![Q]\!]\rho H$ by Lemma 6.
Let $\mathcal{E}_2' = \{pat_i = \rho(M_i) \mid i = 2,\ldots,n\}$ and $\mathcal{E} = \{pat_i = \rho(M_i) \mid i = 1,\ldots,n\}$. Since the variables of pat_i for $i \geq 2$ do not occur in $\mathcal{E}_1$, we have $\text{MGU}(\mathcal{E}_2)\text{MGU}(\mathcal{E}_1) = \text{MGU}(\text{MGU}(\mathcal{E}_1)\mathcal{E}_2')\text{MGU}(\mathcal{E}_1) = \text{MGU}(\mathcal{E}_1 \cup \mathcal{E}_2') = \text{MGU}(\mathcal{E})$ by Lemma 5. So

$$\begin{aligned}&\text{MGU}(\mathcal{E}_2)(\rho'[x \mapsto x \mid x \text{ occurs in } pat_2,\ldots,pat_n]) \\ &\quad = \text{MGU}(\mathcal{E}_2)\text{MGU}(\mathcal{E}_1)(\rho[x \mapsto x \mid x \text{ occurs in } pat_1,\ldots,pat_n]) \\ &\quad = \text{MGU}(\mathcal{E})(\rho[x \mapsto x \mid x \text{ occurs in } pat_1,\ldots,pat_n])\end{aligned}$$

and $\text{MGU}(\mathcal{E}_2)H' = \text{MGU}(\mathcal{E}_2)\text{MGU}(\mathcal{E}_1)H = \text{MGU}(\mathcal{E})H$. Therefore,

$$\begin{aligned}&[\![\mathsf{let}\ pat_1 = M_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ pat_n = M_n\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q]\!]\rho H \sqsubseteq [\![Q]\!]\rho H \\ &\quad \cup [\![P]\!](\text{MGU}(\mathcal{E})(\rho[x \mapsto x \mid x \text{ occurs in } pat_1,\ldots,pat_n]))(\text{MGU}(\mathcal{E})H)\end{aligned}$$

□

Lemma 9. *Let $\Gamma_P \vdash M^G$ be a well-typed pattern, ρ^G a function that associates a clause term with each name and variable, possibly with indices, and Γ an environment for generalized Horn clauses such that $\Gamma_P, \Gamma \vdash \rho^G$. Then $\Gamma \vdash \rho^G(M^G)$*

Proof. We detail here the three interesting cases.

– Case $M^G = x_{\widetilde{\imath}}$. Since Γ_P types $x_{\widetilde{\imath}}$, we have two judgments $x\_ : \widetilde{L} \in \Gamma_P$ and $\Gamma_P \vdash \widetilde{\imath} : \widetilde{L}$. From the definition of $\Gamma_P, \Gamma \vdash \rho^G$, if $\{x_{\widetilde{i}} \mapsto p^G\} \in \rho^G$, then $\Gamma, \widetilde{i} : \widetilde{L} \vdash p^G$. Moreover, as $\Gamma_P \vdash \widetilde{\imath} : \widetilde{L}$, we have $\Gamma \vdash \widetilde{\imath} : \widetilde{L}$. Hence $\rho^G(M^G) = p^G\{\widetilde{\imath}/\widetilde{i}\}$ and $\Gamma \vdash \rho^G(M^G)$.

- Case $M^G = a$. From the definition of $\Gamma \vdash \rho^G$, if $\{a \mapsto p^G\} \in \rho^G$, then $\Gamma \vdash p^G = \rho^G(M^G)$.
- Case $M^G = a_\iota$. Since Γ_P types a_ι, we have two judgments $a\_ : [1, L] \in \Gamma_P$ and $\Gamma_P \vdash i : [1, L]$. From the definition of $\Gamma_P, \Gamma \vdash \rho^G$, if $\{a_i \mapsto p^G\} \in \rho^G$, then $\Gamma, i : [1, L] \vdash p^G$. Moreover, as $\Gamma_P \vdash i : [1, L]$, we have $\Gamma \vdash i : [1, L]$. Hence $\rho^G(M^G) = p^G\{\iota/i\}$ and $\Gamma \vdash \rho^G(M^G)$.

We write $T' \mathit{ext}\, T$ to mean that T' is an extension of the environment T. Given a type environment Γ_P for processes and a type environment Γ for generalized Horn clauses, we define $\{x_{\widetilde{i}} \mapsto p^G\}^T = \{x_{\widetilde{v}} \mapsto p^{GT[\widetilde{i} \mapsto \widetilde{v}]} \mid \widetilde{v} \leq \widetilde{L}\}$ when $x\_ : \widetilde{L} \in \Gamma$, $\{a \mapsto p^G\}^T = \{a \mapsto p^{GT}\}$, and $\{a_i \mapsto p^G\}^T = \{a_v \mapsto p^{GT[i \mapsto v]} \mid v \leq L\}$ when $a\_ : [1, L] \in \Gamma_P$. We extend this definition naturally to ρ^{GT}.

Lemma 10. *Let $\Gamma_P \vdash P^G$ be a well-typed instrumented generalized process, ρ^G a function that associates a clause term with each name and variable, possibly with indices, H^G a sequence of facts, $\mathcal{E}$ a set of equations, and Γ is an environment for generalized Horn clauses such that:*

- *$\Gamma \vdash H^G$;*
- *$\Gamma \vdash \mathcal{E}$;*
- *$\Gamma_P, \Gamma \vdash \rho^G$.*

Then

$$[\![P^{GT}]\!](\mathrm{MGU}\,(\mathcal{E}^T)\rho^{GT})(\mathrm{MGU}\,(\mathcal{E}^T)H^{GT}) \sqsubseteq \bigcup_{T'\,\mathit{ext}\,T} ([\![P^G]\!]\rho^G H^G \mathcal{E} \Gamma)^{T'}$$

and the clauses in the right hand side are well-typed.

Proof. The proof is done by structural induction on the process P^G. Let $\rho = \mathrm{MGU}\,(\mathcal{E}^T)\rho^{GT}$ and $H = \mathrm{MGU}\,(\mathcal{E}^T)H^{GT}$, and let us show that $[\![P^{GT}]\!]\rho H \sqsubseteq \bigcup_{T'\,\mathit{ext}\,T}([\![P^G]\!]\rho^G H^G \mathcal{E} \Gamma)^{T'}$.

- Case $\mathsf{out}(M^G, N^G).P^G$:

$$\begin{aligned}
&[\![(\mathsf{out}(M^G, N^G).P^G)^T]\!]\rho H \\
&\quad = [\![\mathsf{out}(M^{GT}, N^{GT})\rangle.P^{GT}]\!]\rho H \\
&\quad = [\![P^{GT}]\!]\rho H \cup \{(\mathrm{MGU}\,(\mathcal{E}^T)H^{GT} \\
&\qquad\qquad \Rightarrow \mathrm{message}(\mathrm{MGU}\,(\mathcal{E}^T)\rho^{GT}(M^{GT}), \mathrm{MGU}\,(\mathcal{E}^T)\rho^{GT}(N^{GT}))\} \\
&\quad = [\![P^{GT}]\!]\rho H \cup (\{\Gamma \vdash H^G \wedge \mathcal{E} \Rightarrow \mathrm{message}(\rho^G(M^G), \rho^G(N^G))\})^T \\
&\quad \sqsubseteq \bigcup_{T'\,\mathit{ext}\,T} ([\![P^G]\!]\rho^G H^G \mathcal{E} \Gamma)^{T'} \cup \\
&\qquad\qquad \bigcup_{T'\,\mathit{ext}\,T} (\{\Gamma \vdash H^G \wedge \mathcal{E} \Rightarrow \mathrm{message}(\rho^G(M^G), \rho^G(N^G))\})^{T'} \\
&\qquad \text{by induction hypothesis and using that } T \text{ is an extension of itself} \\
&\quad \sqsubseteq \bigcup_{T'\,\mathit{ext}\,T} ([\![\mathsf{out}(M^G, N^G).P^G]\!]\rho^G H^G \mathcal{E} \Gamma)^{T'}
\end{aligned}$$

– Case $\mathsf{in}(M^G, x).P^G$:

$$\begin{aligned}[\![(\mathsf{in}(M^G, x).P^G)^T]\!]\rho H &= [\![\mathsf{in}(M^{GT}, x).P^{GT}]\!]\rho H \\ &= [\![P^{GT}]\!](\rho[x \mapsto x])(H \wedge \mathrm{message}(\rho(M^{GT}), x))\end{aligned}$$

The right-hand side of the theorem develops in

$$\bigcup_{T'\,ext\,T}([\![\mathsf{in}(M^G, x).P^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'} = \bigcup_{T'\,ext\,T}([\![P^G]\!]\rho_1^G H_1^G \mathcal{E}\Gamma_1)^{T'}$$

where $\rho_1^G = \rho^G[x \mapsto x]$, $H_1^G = H^G \wedge \mathrm{message}(\rho^G(M^G), x)$, and $\Gamma_1 = \Gamma, x\_ : [\,]$. We show that $\rho[x \mapsto x] = \textsc{mgu}\,(\mathcal{E}^T)\rho_1^{GT}$:

$$\textsc{mgu}\,(\mathcal{E}^T)\rho_1^{GT} = \textsc{mgu}\,(\mathcal{E}^T)\rho^{GT}[x \mapsto x] = \rho[x \mapsto x]$$

and $H \wedge \mathrm{message}(\rho(M^{GT}), x) = \textsc{mgu}\,(\mathcal{E}^T)H_1^{GT}$:

$$\begin{aligned}\textsc{mgu}\,(\mathcal{E}^T)H_1^{GT} &= \textsc{mgu}\,(\mathcal{E}^T)(H^G \wedge \mathrm{message}(\rho^G(M^G), x))^T \\ &= \textsc{mgu}\,(\mathcal{E}^T)H^{GT} \wedge \textsc{mgu}\,(\mathcal{E}^T)(\mathrm{message}(\rho^{GT}(M^{GT}), x) \\ &= H \wedge \mathrm{message}(\textsc{mgu}\,(\mathcal{E}^T)\rho^{GT}(M^{GT}), x) \\ &= H \wedge \mathrm{message}(\rho(M^{GT}), x)\end{aligned}$$

Let Γ'_P the environment that types P^G, $\Gamma'_P = \Gamma_P, x\_ : [\,]$. Before applying the induction hypothesis we need to show that $\Gamma'_P, \Gamma_1 \vdash \rho_1^G$ and $\Gamma_1 \vdash H_1^G$ (clearly, $\Gamma_1 \vdash \mathcal{E}$). Since $\Gamma_P, \Gamma \vdash \rho^G$, we have $\Gamma'_P, \Gamma_1 \vdash \rho^G$. For the new map $[x \mapsto x] \in \rho_1^G$ we have that $x\_ : [\,] \in \Gamma'_P$ and $\Gamma_1 \vdash x$. Hence $\Gamma'_P, \Gamma_1 \vdash \rho_1^G$. Since $\Gamma \vdash H^G$, we have $\Gamma_1 \vdash H^G$. From Lemma 9 we have that $\Gamma \vdash \rho^G(M^G)$, as $\Gamma_P, \Gamma \vdash \rho^G$ and $\Gamma_P \vdash M^G$. Finally $\Gamma_1 \vdash x$. Hence $\Gamma_1 \vdash \mathrm{message}(\rho^G(M^G), x)$, and thus $\Gamma_1 \vdash H_1^G$. Therefore, we can apply the induction hypothesis and conclude.

– Case $\mathbf{0}$: $[\![\mathbf{0}^T]\!]\rho H = \emptyset = \bigcup_{T'\,ext\,T}([\![\mathbf{0}]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}$.

– Case $P^G \mid Q^G$:

$$\begin{aligned}[\![(P^G \mid Q^G)^T]\!]\rho H &= [\![P^{GT} \mid Q^{GT}]\!]\rho H = [\![P^{GT}]\!]\rho H \cup [\![Q^{GT}]\!]\rho H \\ &\sqsubseteq \bigcup_{T'\,ext\,T}([\![P^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'} \cup \bigcup_{T'\,ext\,T}([\![Q^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'} \\ &\sqsubseteq \bigcup_{T'\,ext\,T}([\![P^G \mid Q^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}\end{aligned}$$

– Case $!^s P^G$:

$$\begin{aligned}[\![(!^s P^G)^T]\!]\rho h &= [\![!^s P^{GT}]\!]\rho H = [\![P^{GT}]\!](\rho[s \mapsto s])H \\ &\sqsubseteq \bigcup_{T'\,ext\,T}([\![P^G]\!](\rho^G[s \mapsto s])H^G \mathcal{E}\Gamma)^{T'} \\ &\sqsubseteq \bigcup_{T'\,ext\,T}([\![!^s P^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}\end{aligned}$$

– Case $\Pi_{i \leq L} P$:

$$\begin{aligned}\llbracket (\Pi_{i \leq L} P)^T \rrbracket \rho H &= \llbracket P^{GT[i \mapsto 1]} \mid \ldots \mid P^{GT[i \mapsto L^T]} \rrbracket \rho H \\ &= \llbracket P^{GT[i \mapsto 1]} \rrbracket \rho H \cup \cdots \cup \llbracket P^{GT[i \mapsto L^T]} \rrbracket \rho H\end{aligned}$$

By induction hypothesis, $\llbracket P^{GT[i \mapsto v]} \rrbracket \rho H \sqsubseteq \bigcup_{T' \, ext \, T[i \mapsto v]} (\llbracket P^G \rrbracket \rho^G H^G \mathcal{E}(\Gamma, i : [1, L]))^{T'}$ for each $v \in \{1, \ldots, L^T\}$. Therefore

$$\begin{aligned}\llbracket (\Pi_{i \leq L} P)^T \rrbracket \rho H \sqsubseteq & \bigcup_{T' \, ext \, T[i \mapsto 1]} (\llbracket P^G \rrbracket \rho^G H^G \mathcal{E}(\Gamma, i : [1, L]))^{T'} \cup \cdots \cup \\ & \bigcup_{T' \, ext \, T[i \mapsto L^T]} (\llbracket P^G \rrbracket \rho^G H^G \mathcal{E}(\Gamma, i : [1, L]))^{T'} \\ \sqsubseteq & \bigcup_{T' \, ext \, T} (\llbracket P^G \rrbracket \rho^G H^G \mathcal{E}(\Gamma, i : [1, L]))^{T'} \\ \sqsubseteq & \bigcup_{T' \, ext \, T} (\llbracket \Pi_{i \leq L} P^G \rrbracket \rho^G H^G \mathcal{E} \Gamma)^{T'}\end{aligned}$$

since $T[i \mapsto v]$ is an extension of T for each $v \in \{1, \ldots, L^T\}$.

– Case $(\mathsf{for\ all}\ i \leq L, \nu a_i : a_{i,\tilde{i}}^{L,\tilde{L}}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}]) P^G$:

$$\begin{aligned}&\llbracket ((\mathsf{for\ all}\ i \leq L, \nu a_i : a_{i,\tilde{i}}^{L,\tilde{L}}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}]) P^G)^T \rrbracket \rho H \\ &\quad = \llbracket (\nu a_1 : a_{1,\tilde{i}^T}^{L^T,\tilde{L}^T}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}]) \ldots \\ &\qquad (\nu a_{L^T} : a_{L^T,\tilde{i}^T}^{L^T,\tilde{L}^T}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}]) P^{GT} \rrbracket \rho H \\ &\quad = \llbracket P^{GT} \rrbracket \rho_1 H\end{aligned}$$

where

$$\begin{aligned}\rho_1 = \rho[&a_1 \mapsto a_{1,\tilde{i}^T}^{L^T,\tilde{L}^T}[\rho(x_1), \ldots, \rho(x_n), \rho(s_1), \ldots, \rho(s_{n'})], \ldots, \\ &a_{L^T} \mapsto a_{L^T,\tilde{i}^T}^{L^T,\tilde{L}^T}[\rho(x_1), \ldots, \rho(x_n), \rho(s_1), \ldots, \rho(s_{n'})]] \,.\end{aligned}$$

The right-hand side of the theorem develops in

$$\begin{aligned}&\bigcup_{T' \, ext \, T} (\llbracket (\mathsf{for\ all}\ i \leq L, \nu a_i : a_{i,\tilde{i}}^{L,\tilde{L}}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}]) P^G \rrbracket \rho^G H^G \mathcal{E} \Gamma)^{T'} \\ &\quad = \bigcup_{T' \, ext \, T} (\llbracket P^G \rrbracket \rho_1^G H^G \mathcal{E} \Gamma)^{T'}\end{aligned}$$

where $\rho_1^G = \rho^G[a_i \mapsto a_{i,\widetilde{i}}^{L,\widetilde{L}}[\rho^G(x_1), \dots, \rho^G(x_n), \rho^G(s_1), \dots, \rho^G(s_{n'})]]$. We show that $\rho_1 = \mathrm{MGU}\,(\mathcal{E}^T)\rho_1^{GT}$:

$$\begin{aligned}\rho_1 &= \rho[a_1 \mapsto a_{1,\widetilde{i}^T}^{L^T,\widetilde{L}^T}[\rho(x_1), \dots, \rho(x_n), \rho(s_1), \dots, \rho(s_{n'})], \dots,\\ &\qquad a_{L^T} \mapsto a_{L^T,\widetilde{i}^T}^{L^T,\widetilde{L}^T}[\rho(x_1), \dots, \rho(x_n), \rho(s_1), \dots, \rho(s_{n'})]]\\ &= \mathrm{MGU}\,(\mathcal{E}^T)\\ &\qquad (\rho^{GT}[a_1 \mapsto a_{1,\widetilde{i}^T}^{L^T,\widetilde{L}^T}[\rho^{GT}(x_1), \dots, \rho^{GT}(x_n), \rho^{GT}(s_1), \dots, \rho^{GT}(s_{n'})],\\ &\qquad \dots, a_{L^T} \mapsto a_{L^T,\widetilde{i}^T}^{L^T,\widetilde{L}^T}[\rho^{GT}(x_1), \dots, \rho^{GT}(x_n), \rho^{GT}(s_1), \dots, \rho^{GT}(s_{n'})]])\\ &= \mathrm{MGU}\,(\mathcal{E}^T)\rho_1^{GT}\end{aligned}$$

Let Γ'_P the environment that types P^G, $\Gamma'_P = \Gamma_P, a\_ : [1, L]$. Before applying the induction hypothesis we need to show that $\Gamma'_P, \Gamma \vdash \rho_1^G$. Since $\Gamma_P, \Gamma \vdash \rho^G$, we have $\Gamma'_P, \Gamma \vdash \rho^G$. For the new map $[a_i \mapsto a_{i,\widetilde{i}}^{L,\widetilde{L}}[\rho^G(x_1), \dots, \rho^G(x_n), \rho^G(s_1), \dots, \rho^G(s_{n'})]] \in \rho_1^G$ we have that $a\_ : [1, L] \in \Gamma'_P$ and $\Gamma \vdash a_{i,\widetilde{i}}^{L,\widetilde{L}}[\rho^G(x_1), \dots, \rho^G(x_n), \rho^G(s_1), \dots, \rho^G(s_{n'})$ by Lemma 9. Hence $\Gamma'_P, \Gamma \vdash \rho_1^G$. We can then apply the induction hypothesis and conclude.

- Case $(\nu a)P^G$: this case is similar to the previous one.
- Case $\mathsf{let\ for\ all}\ \widetilde{i} \le \widetilde{L}, x_{\widetilde{i}} = g(M_1^G, \dots, M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G$: let $g(p_1, \dots, p_n) \to p'$ be the rewrite rule for the destructor g. We suppose that the tuples of indices $\widetilde{v} \le \widetilde{L}^T$ are indexed by $1, \dots, l$, that is, we define $\{\widetilde{v}_1, \dots, \widetilde{v}_l\} = \{\widetilde{1}, \dots, \widetilde{L}^T\}$. We let $T'_k = T[\widetilde{i} \mapsto \widetilde{v}_k]$ for $k = 1, \dots, l$.

$$\begin{aligned}&[\![(\mathsf{let\ for\ all}\ \widetilde{i} \le \widetilde{L}, x_{\widetilde{i}} = g(M_1^G, \dots, M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G)^T]\!]\rho H\\ &\quad = [\![\mathsf{let}\ E_1\ \mathsf{in}\ \dots\ \mathsf{let}\ E_l\ \mathsf{in}\ P^{GT}\ \mathsf{else}\ Q^{GT}\ \dots\ \mathsf{else}\ Q^{GT}]\!]\rho H\\ &\qquad\quad \text{where } E_k \text{ is } x_{\widetilde{i}}^{T'_k} = g(M_1^{GT'_k}, \dots, M_n^{GT'_k}) \text{ for } k = 1, \dots, l.\\ &\quad \sqsubseteq [\![Q^{GT}]\!]\rho H \cup [\![P^{GT}]\!](\mathrm{MGU}\,(\mathcal{E}_1)(\rho[x_{\widetilde{v}_1} \mapsto p'_1, \dots, x_{\widetilde{v}_l} \mapsto p'_l]))(\mathrm{MGU}\,(\mathcal{E}_1)H)\end{aligned}$$

by Lemma 7, where $p_{k,1}, \dots, p_{k,n}, p'_k$ are the patterns $p_1, \dots, p_n, p'$ renamed with distinct fresh variables for each $k = 1, \dots, l$ and $\mathcal{E}_1 = \{\rho(M_j^{GT'_k}) = p_{k,j} \mid k = 1, \dots, l \text{ and } j = 1, \dots, n\}$, assuming that $\mathrm{MGU}\,(\mathcal{E}_1)$ exists. (When $\mathrm{MGU}\,(\mathcal{E}_1)$ does not exist, the second component of the union is omitted, and the rest of the proof can easily be adapted.) The right-hand side of the theorem develops in

$$\begin{aligned}&\bigcup_{T'\,ext\,T} ([\![\mathsf{let\ for\ all}\ \widetilde{i} \le \widetilde{L}, x_{\widetilde{i}} = g(M_1^G, \dots, M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}\\ &\quad = \bigcup_{T'\,ext\,T} ([\![Q^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'} \cup \bigcup_{T'\,ext\,T} ([\![P^G]\!](\rho^G[x_{\widetilde{i}} \mapsto p'^G])H^G(\mathcal{E} \cup \mathcal{E}')\Gamma')^{T'}\end{aligned}$$

where $\mathcal{E}'$ and Γ' are defined as follows. The rewrite rule $g(p_1'^G, \dots, p_n'^G) \to p'^G$ is obtained from $g(p_1, \dots, p_n) \to p'$ by replacing all variables y of this

rule with fresh variables with indices $\widetilde{i}$: $y'_{\widetilde{i}}$. Then $\mathcal{E}'$ is the set of equations $\{\bigwedge_{\widetilde{i}\in\widetilde{L}} p_1'^G \doteq \rho^G(M_1^G), \ldots, \bigwedge_{\widetilde{i}\in\widetilde{L}} p_n'^G \doteq M_n^G\}$ and Γ' is Γ extended with $x\_ : \widetilde{L}$ and $y'\_ : \widetilde{L}$ for each variable $y'_{\widetilde{i}}$ in $p_1'^G, \ldots p_n'^G, p'^G$.
We analyze now the relation between $\text{MGU}\,(\mathcal{E}_1)$ and $\mathcal{E}'^T$. We have $\mathcal{E}'^T = \{\rho^{GT}(M_j^{GT[\widetilde{i}\mapsto\widetilde{v}]}) = p_j'^{GT[\widetilde{i}\mapsto\widetilde{v}]} \mid \widetilde{v} \leq \widetilde{L}^T \text{ and } j = 1, \ldots n\}$. Given the construction of $p_{k,j}$, p'_k, $p_j'^G$, p'^G, there is a renaming α such that, for all $k = 1, \ldots, l$, we have $\alpha p_{k,j} = p_j'^{GT'_k}$ for each $j = 1, \ldots, n$ and $\alpha p'_i = p'^{GT'_k}$. Hence we have

$$\begin{aligned}
\text{MGU}\,(\mathcal{E}^T)\mathcal{E}'^T &= \{\text{MGU}\,(\mathcal{E}^T)\rho^{GT}(M_j^{GT[\widetilde{i}\mapsto\widetilde{v}]}) = \text{MGU}\,(\mathcal{E}^T)p_j'^{GT[\widetilde{i}\mapsto\widetilde{v}]} \\
&\qquad \mid \widetilde{v} \leq \widetilde{L}^T \text{ and } j = 1, \ldots n\} \\
&= \{\rho(M_j^{GT[\widetilde{i}\mapsto\widetilde{v}]}) = p_j'^{GT[\widetilde{i}\mapsto\widetilde{v}]} \mid \widetilde{v} \leq \widetilde{L}^T \text{ and } j = 1, \ldots n\} \\
&\qquad \text{since the variables of } p_j'^{GT[\widetilde{i}\mapsto\widetilde{v}]} \text{ are fresh,} \\
&\qquad \text{so they are not touched by } \text{MGU}\,(\mathcal{E}^T) \\
&= \{\rho(M_j^{GT'_k}) = \alpha p_{k,j} \mid k = 1, \ldots, l \text{ and } j = 1, \ldots n\} \\
&= \alpha\mathcal{E}_1
\end{aligned}$$

So, by Lemma 5,

$$\text{MGU}\,(\alpha\mathcal{E}_1)\text{MGU}\,(\mathcal{E}^T) = \text{MGU}\,(\text{MGU}\,(\mathcal{E}^T)\mathcal{E}'^T)\text{MGU}\,(\mathcal{E}^T) = \text{MGU}\,((\mathcal{E}\cup\mathcal{E}')^T)$$

Hence

$$\begin{aligned}
&\text{MGU}\,(\alpha\mathcal{E}_1)(\rho[x_{\widetilde{v}_1} \mapsto \alpha p'_1, \ldots, x_{\widetilde{v}_l} \mapsto \alpha p'_l]) \\
&\quad = \text{MGU}\,(\alpha\mathcal{E}_1)\text{MGU}\,(\mathcal{E}^T)(\rho^{GT}[x_{\widetilde{1}} \mapsto p'^{GT[\widetilde{i}\mapsto\widetilde{1}]}, \ldots, x_{\widetilde{L}^T} \mapsto p'^{GT[\widetilde{i}\mapsto\widetilde{L}^T]}]) \\
&\quad = \text{MGU}\,((\mathcal{E}\cup\mathcal{E}')^T)(\rho^G[x_{\widetilde{i}} \mapsto p'^G])^T
\end{aligned}$$

Similarly, $\text{MGU}\,(\alpha\mathcal{E}_1)H = \text{MGU}\,(\alpha\mathcal{E}_1)\text{MGU}\,(\mathcal{E}^T)H^{GT} = \text{MGU}\,((\mathcal{E}\cup\mathcal{E}')^T)H^{GT}$.
Let Γ'_P the environment that types P^G: by the typing rules we have that $\Gamma'_P = \Gamma_P, x\_ : \widetilde{L}$. Before applying induction we need to show that $\Gamma'_P, \Gamma' \vdash \rho^G[x_{\widetilde{i}} \mapsto p'^G]$ and $\Gamma' \vdash \mathcal{E}\cup\mathcal{E}'$. At first, notice that $p_1'^G, \ldots, p_n'^G, p'^G$ are obtained from $p_1, \ldots, p_n, p'$ by replacing all variables y with fresh variables with indices $y'_{\widetilde{i}}$ and that Γ' types each variable $y'\_$ with type $\widetilde{L}$. Hence all variables in $p_1'^G, \ldots, p_n'^G, p'^G$ are typed by Γ'.
We have that $\Gamma'_P, \Gamma' \vdash \rho^G$ because Γ' extends Γ, Γ'_P extends Γ_P, and $\Gamma_P, \Gamma \vdash \rho^G$ by hypothesis. Since $x\_ : \widetilde{L} \in \Gamma'_P$ and $\Gamma', \widetilde{i} : \widetilde{L} \vdash p'^G$ (all variables in p'^G are typed by Γ'), we have $\Gamma'_P, \Gamma' \vdash \rho^G[x_{\widetilde{i}} \mapsto p'^G]$.
For each equation $\bigwedge_{\widetilde{i}\in\widetilde{L}} p_j'^G \doteq \rho^G(M_j^G)$, $j = 1, \ldots n$ we have that:

- $\Gamma', \widetilde{i} : \widetilde{L} \vdash \rho^G(M_j^G)$: this comes from Lemma 9 applied to $\Gamma_P, \widetilde{i} : \widetilde{L} \vdash M_j^G$ and $(\Gamma_P, \widetilde{i} : \widetilde{L}), \Gamma \vdash \rho^G$ and from the fact that Γ' extends Γ.
- $\Gamma', \widetilde{i} : \widetilde{L} \vdash p_j'^G$: all variables in $p_j'^G$ are typed in Γ'.

This means that each equation in $\mathcal{E}'$ is well typed in Γ'; moreover $\Gamma' \vdash \mathcal{E}$ because Γ' extends Γ and $\Gamma \vdash \mathcal{E}$ by hypothesis. Thus $\Gamma' \vdash \mathcal{E} \cup \mathcal{E}'$.
We can then apply the induction hypothesis, which yields

$$\begin{aligned}&[\![P^{GT}]\!](\textsc{mgu}\,(\alpha\mathcal{E}_1)(\rho[x_{\widetilde{v}_1} \mapsto \alpha p'_1, \dots, x_{\widetilde{v}_l} \mapsto \alpha p'_l]))(\textsc{mgu}\,(\alpha\mathcal{E}_1)H)\\ &\quad\sqsubseteq \bigcup_{T'\,ext\,T} ([\![P^G]\!](\rho^G[x_{\widetilde{i}} \mapsto p'^G])H^G(\mathcal{E}\cup\mathcal{E}')\Gamma')^{T'}\end{aligned}$$

and $[\![Q^{GT}]\!]\rho H \sqsubseteq \bigcup_{T'\,ext\,T}([\![Q^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}$.
Moreover,

$$\begin{aligned}&[\![P^{GT}]\!](\textsc{mgu}\,(\mathcal{E}_1)(\rho[x_{\widetilde{v}_1} \mapsto p'_1, \dots, x_{\widetilde{v}_l} \mapsto p'_l]))(\textsc{mgu}\,(\mathcal{E}_1)H)\\ &\quad\sqsubseteq [\![P^{GT}]\!](\textsc{mgu}\,(\alpha\mathcal{E}_1)(\rho[x_{\widetilde{v}_1} \mapsto \alpha p'_1, \dots, x_{\widetilde{v}_l} \mapsto \alpha p'_l]))(\textsc{mgu}\,(\alpha\mathcal{E}_1)H)\end{aligned}$$

(These two sets of clauses are in fact equal up to renaming of variables, by construction.)
Hence we can conclude that:

$$\begin{aligned}&[\![(\mathsf{let\ for\ all}\ \widetilde{i} \leq \widetilde{L}, x_{\widetilde{i}} = g(M_1^G, \dots, M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G)^T]\!]\rho H \sqsubseteq\\ &\bigcup_{T'\,ext\,T} ([\![\mathsf{let\ for\ all}\ \widetilde{i} \leq \widetilde{L}, x_{\widetilde{i}} = g(M_1^G, \dots, M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}\end{aligned}$$

– Case $\mathsf{let\ for\ all}\ \widetilde{i} \leq \widetilde{L}, pat^G = M^G\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G$: as in the previous case, we suppose that the tuples of indices $\widetilde{v} \leq \widetilde{L}^T$ are indexed by $1, \dots, l$, that is, we define $\{\widetilde{v}_1, \dots, \widetilde{v}_l\} = \{\widetilde{1}, \dots, \widetilde{L}^T\}$.

$$\begin{aligned}&[\![(\mathsf{let\ for\ all}\ \widetilde{i} \leq \widetilde{L}, pat^G = M^G\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G)^T]\!]\rho H\\ &\quad= [\![\mathsf{let}\ E_1\ \mathsf{in}\ \dots\ \mathsf{let}\ E_l\ \mathsf{in}\ P^{GT}\ \mathsf{else}\ Q^{GT}\ \dots\ \mathsf{else}\ Q^{GT}]\!]\rho H\\ &\qquad\quad \text{where } E_i \text{ is the equation } pat^{GT[\widetilde{i}\mapsto\widetilde{v}_i]} = M^{GT[\widetilde{i}\mapsto\widetilde{v}_i]}.\\ &\quad\sqsubseteq [\![Q]\!]\rho H \cup [\![P]\!]\rho' H'\end{aligned}$$

by Lemma 8, where $\mathcal{E}_1 = \{pat^{GT''} = \rho(M^{GT''}) \mid T'' = T[\widetilde{i} \mapsto \widetilde{v}], \widetilde{v} \leq \widetilde{L}^T\}$, $\rho' = \textsc{mgu}\,(\mathcal{E}_1)(\rho[x \mapsto x \mid x \text{ occurs in } pat^{GT[\widetilde{i}\mapsto\widetilde{v}]}, \widetilde{v} \leq \widetilde{L}^T])$, and $H' = \textsc{mgu}\,(\mathcal{E}_1)H$, assuming that $\textsc{mgu}\,(\mathcal{E}_1)$ exists. (When $\textsc{mgu}\,(\mathcal{E}_1)$ does not exist, the second component of the union is omitted, and the rest of the proof can easily be adapted.)
The right-hand side of the theorem develops in:

$$\begin{aligned}&\bigcup_{T'\,ext\,T} ([\![\mathsf{let\ for\ all}\ \widetilde{i} \leq \widetilde{L}, pat^G = M^G\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}\\ &\quad= \bigcup_{T'\,ext\,T} ([\![Q]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'} \cup\\ &\qquad\quad [\![P]\!](\rho^G[x_{\widetilde{i}'} \mapsto x_{\widetilde{i}'} \mid x_{\widetilde{i}'} \text{ occurs in } pat^G])H^G(\mathcal{E}\cup\mathcal{E}')\Gamma')^{T'}\end{aligned}$$

where $\mathcal{E}' = \bigwedge_{\widetilde{i} \leq \widetilde{L}} pat^G \doteq \rho^G(M^G)$ and Γ' is Γ extended for the variables occurring in pat^G. More precisely, if the typing rule for the process let for all $\widetilde{i} \leq \widetilde{L}, pat^G = M^G$ in P^G else Q^G has $i_1 : [1, L_1], \dots, i_h : [1, L_h] \vdash pat^G \rightsquigarrow \Gamma''$ as a premise, then $\Gamma' = \Gamma, \Gamma''$. Hence $\mathcal{E}'^T = \{pat^{GT''} = \rho^{GT}(M^{GT''}) \mid T'' = T[\widetilde{i} \mapsto \widetilde{v}], \forall \widetilde{v} \leq \widetilde{L}^T\}$. Hence we have that:

$$\begin{aligned} &\text{MGU}\,(\mathcal{E}^T)\mathcal{E}'^T \\ &\quad = \{\text{MGU}\,(\mathcal{E}^T) pat^{GT[\widetilde{i} \mapsto \widetilde{v}]} = \text{MGU}\,(\mathcal{E}^T)\rho^{GT}(M^{GT[\widetilde{i} \mapsto \widetilde{v}]}) \mid \forall \widetilde{v} \leq \widetilde{L}^T\} \\ &\quad = \{pat^{GT[\widetilde{i} \mapsto \widetilde{v}]} = \rho(M^{GT[\widetilde{i} \mapsto \widetilde{v}]}) \mid \forall \widetilde{v} \leq \widetilde{L}^T\} \\ &\quad = \mathcal{E}_1 \end{aligned}$$

By Lemma 5,

$$\text{MGU}\,(\mathcal{E}_1)\text{MGU}\,(\mathcal{E}^T) = \text{MGU}\,(\text{MGU}\,(\mathcal{E}^T)\mathcal{E}'^T)\text{MGU}\,(\mathcal{E}^T) = \text{MGU}\,((\mathcal{E} \cup \mathcal{E}')^T)$$

so

$$\begin{aligned} \rho' &= \text{MGU}\,(\mathcal{E}_1)(\rho[x \mapsto x \mid x \text{ occurs in } pat^{GT[\widetilde{i} \mapsto \widetilde{v}_i]}, \widetilde{v}_i \leq \widetilde{L}^T]) \\ &= \text{MGU}\,(\mathcal{E}_1)\text{MGU}\,(\mathcal{E}^T)(\rho^{GT}[x \mapsto x \mid x \text{ occurs in } pat^{GT[\widetilde{i} \mapsto \widetilde{v}_i]}, \widetilde{v}_i \leq \widetilde{L}^T]) \\ &= \text{MGU}\,((\mathcal{E} \cup \mathcal{E}')^T)(\rho^G[x_{\widetilde{i}'} \mapsto x_{\widetilde{i}'} \mid x_{\widetilde{i}'} \text{ occurs in } pat^G])^T \end{aligned}$$

Similarly,

$$H' = \text{MGU}\,(\mathcal{E}_1)H = \text{MGU}\,(\mathcal{E}_1)\text{MGU}\,(\mathcal{E}^T)H^{GT} = \text{MGU}\,((\mathcal{E} \cup \mathcal{E}')^T)H^{GT}\,.$$

Let Γ'_P the environment that types P^G: by the typing rules we have that $\Gamma'_P = \Gamma_P, \Gamma''_P$, where $\widetilde{i} : \widetilde{L} \vdash pat^G \rightsquigarrow \Gamma''_P$. Before applying induction we need to show that $\Gamma'_P, \Gamma' \vdash \rho^G[x_{\widetilde{i}'} \mapsto x_{\widetilde{i}'} \mid x_{\widetilde{i}'} \text{ occurs in } pat^G]$ and $\Gamma' \vdash \mathcal{E} \cup \mathcal{E}'$.
We have that $\Gamma'_P, \Gamma' \vdash \rho^G$ because Γ' extends Γ, Γ'_P extends Γ_P, and $\Gamma_P, \Gamma \vdash \rho^G$ by hypothesis. Clearly $x\_ : \widetilde{L} \in \Gamma''_P$ (that is $x\_ : \widetilde{L} \in \Gamma'_P$) and $\Gamma', \widetilde{i} : \widetilde{L} \vdash x_{\widetilde{i}}$ (all variables in pat^G are typed by Γ') then $\Gamma'_P, \Gamma' \vdash \rho^G[x_{\widetilde{i}'} \mapsto x_{\widetilde{i}'} \mid x_{\widetilde{i}'} \text{ occurs in } pat^G]$.
For the equation $\bigwedge_{\widetilde{i} \in \widetilde{L}} pat^G \doteq \rho^G(M^G)$ we have that:

- $\Gamma', \widetilde{i} : \widetilde{L} \vdash \rho^G(M^G)$: this comes from Lemma 9 applied to $\Gamma_P, \widetilde{i} : \widetilde{L} \vdash M^G$ and $(\Gamma_P, \widetilde{i} : \widetilde{L}), \Gamma \vdash \rho^G$ and from the fact that Γ' extends Γ.
- $\Gamma', \widetilde{i} : \widetilde{L} \vdash pat^G$: all variables in pat'^G are typed in Γ'.

This means that the equation in $\mathcal{E}'$ is well typed in Γ'; moreover $\Gamma' \vdash \mathcal{E}$ because Γ' extends Γ and $\Gamma \vdash \mathcal{E}$ by hypothesis. Thus $\Gamma' \vdash \mathcal{E} \cup \mathcal{E}'$.
We can then apply the induction hypothesis:

$$[\![P^{GT}]\!]\rho' H' \sqsubseteq \bigcup_{T'\,ext\,T} [\![P]\!](\rho^G[x_{\widetilde{i}'} \mapsto x_{\widetilde{i}'} \mid x_{\widetilde{i}'} \text{ occurs in } pat^G])H^G(\mathcal{E} \cup \mathcal{E}')\Gamma')^{T'}$$

and $[\![Q^{GT}]\!]\rho H \sqsubseteq \bigcup_{T'\,ext\,T}([\![Q^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}$. Therefore we can conclude.

– Case $\mathsf{choose}\ L\ \mathsf{in}\ P^G$:

$$
\begin{aligned}
&[\![(\mathsf{choose}\ L\ \mathsf{in}\ P^G)^T]\!]\rho H\\
&\quad = [\![P^{GT[L\mapsto 1]} + \dots + P^{GT[L\mapsto n]} + \dots]\!]\rho H\\
&\quad = [\![P^{GT[L\mapsto 1]}]\!]\rho H \cup \dots \cup [\![P^{GT[L\mapsto n]}]\!]\rho H \cup \dots\\
&\quad \sqsubseteq \bigcup_{T'\, ext\, T[L\mapsto 1]} ([\![P^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'} \cup \dots \cup \bigcup_{T'\, ext\, T[L\mapsto n]} ([\![P^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'} \cup \dots\\
&\quad \sqsubseteq \bigcup_{T'\, ext\, T} ([\![\mathsf{choose}\ L\ \mathsf{in}\ P^G]\!]\rho^G H^G \mathcal{E}\Gamma^{T'})
\end{aligned}
$$

– Case $\mathsf{choose}\ k \leq L\ \mathsf{in}\ P^G$:

$$
\begin{aligned}
&[\![(\mathsf{choose}\ k \leq L\ \mathsf{in}\ P^G)^T]\!]\rho H\\
&\quad = [\![P^{GT[k\mapsto 1]} + \dots + P^{GT[k\mapsto L^T]}]\!]\rho H\\
&\quad = [\![P^{GT[k\mapsto 1]}]\!]\rho H \cup \dots \cup [\![P^{GT[k\mapsto L^T]}]\!]\rho H\\
&\quad \sqsubseteq \bigcup_{T'\, ext\, T[k\mapsto 1]} ([\![P^G]\!]\rho^G H^G \mathcal{E}(\Gamma, k : [1, L]))^{T'} \cup \dots \cup\\
&\qquad\quad \bigcup_{T'\, ext\, T[k\mapsto L^T]} ([\![P^G]\!]\rho^G H^G \mathcal{E}(\Gamma, k : [1, L]))^{T'}\\
&\quad \sqsubseteq \bigcup_{T'\, ext\, T} ([\![\mathsf{choose}\ k \leq L\ \mathsf{in}\ P^G]\!]\rho^G H^G \mathcal{E}\Gamma^{T'})
\end{aligned}
$$

– Case $\mathsf{choose}\ \phi : L_1 \times \dots \times L_h \to L'\ \mathsf{in}\ P^G$: this case is similar to previous one. □

Combining the results From the previous results, we easily obtain Theorem 3.

Proof (of Theorem 3). By Lemma 10, $([\![P_1^G]\!]\rho_0\emptyset\emptyset\Gamma_0)^{\mathcal{T}} = \bigcup_T ([\![P_1^G]\!]\rho_0\emptyset\emptyset\Gamma_0)^T \sqsupseteq [\![P_1^{GT_0}]\!]\rho_0\emptyset$, where $P_1^G = \mathrm{instr}^G(P_0^G)$. By Lemma 4, $\mathrm{instr}(P_0') = \mathrm{instr}(\mathrm{Tren}(P_0^G, T_0, \emptyset \leq \emptyset)) \equiv_\alpha \mathrm{instr}^G(P_0^G)^{T_0} = P_1^{GT_0}$, so we have $([\![\mathrm{instr}^G(P_0^G)]\!]\rho_0\emptyset\emptyset\Gamma_0)^{\mathcal{T}} \sqsupseteq [\![\mathrm{instr}(P_0')]\!]\rho_0\emptyset$ since the translation to Horn clauses $[\![\cdot]\!]$ is invariant by renaming of bound names.

Moreover, for each clause R in $\{\mathrm{att}(a[\,]) \mid a \in S\} \cup \{$(Rn), (Rf), (Rg), (Rl), (Rs)$\}$ except the clauses (Rf) and (Rg) for lists of fixed length, R is also a generalized Horn clause and we have $\{R\}^{\mathcal{T}} = \{R\}$. The clauses (Rf) for lists of fixed length are in $\{R^G\}^{\mathcal{T}} = \{\mathrm{att}(x_1) \wedge \dots \wedge \mathrm{att}(x_n) \Rightarrow \mathrm{att}(\langle x_1, \dots, x_n\rangle) \mid n \in \mathbb{N}\}$, where $R^G =$ (Rf-list). The clauses (Rg) for lists of fixed length are in $\{R^G\}^{\mathcal{T}} = \{\mathrm{att}(\langle x_1, \dots, x_n\rangle) \Rightarrow \mathrm{att}(x_v) \mid n \in \mathbb{N}, v \leq n\}$ where $R^G =$ (Rg-list).

So we obtain $\mathcal{R}^{G\mathcal{T}}_{P_0^G,S} \sqsupseteq \mathcal{R}_{P_0',S}$. □