2006-4-(4): Manuscript for presentation at IPSJ-SIGPRO, 18 January 2007.


*Regular Paper*

# On-the-fly model checking of security protocols and its implementation by Maude

Guoqiang Li† and Mizuhito Ogawa†

Trace analysis for a security protocol represents every possible run as a trace and analyzes whether any insecure run is reachable. The number of traces will be infinite due to (1) infinitely many sessions of a protocol, (2) infinitely many principals in the network, and (3) infinitely many messages that intruders can generate. This paper presents an on-the-fly model checking method by restricting/abstracting these infinite factors to a finite model. First, we restrict a typed process calculus to avoid recursive operations, so that only finitely many sessions are considered. Next, a bound variable is introduced as an index of a message to represent its intended destination, so that an unbounded number of principals are finitely described. Then, messages in which irrelevant parts are reduced in a protocol are unified to a parametric message based on the type information. We implement the on-the-fly model checking method using Maude, and automatically detect the flaws of several security protocols, such as the NSPK protocol and the Woo-Lam protocol, etc..

## 1. Introduction

Trace analysis represents every possible run of a protocol as a trace, and analyzes whether any insecure state is reachable[1)~3)]. When applying a model checking as its execution engine, the main difficulty is that the number of traces is infinite. The reasons are:

- Each principal can initiate or act in response in an unlimited number of sessions, which causes unbounded length of each trace.
- Each principal may communicate with an unlimited number of principals, which causes infinitely many traces.
- Each intruder can produce, store, duplicate, hide, or replace an unlimited number of messages based on the messages sent in the network, following the Dolev-Yao model[4)]. This also causes infinitely many traces.

In this paper, a finite *parametric model* is proposed by restricting/abstracting the infinite factors of security protocols. Security properties, such as authentication and secrecy, are checked automatically by on-the-fly model checking. This model checking is sound and complete under the restriction of the bounded number of sessions, and implemented on Maude.

To describe security protocols, we set a typed process calculus based on a variant of Spi calculus[5)], in which new syntax, the *binder* and the *range* are introduced. The calculus uses environment-based communication, instead of the standard channel-based communication. Following the Dolev-Yao model, a deductive system is used in the environment to generate infinitely many messages[2)].

To restrict ourselves to a bounded number of sessions of a protocol, the calculus avoids recursive operations, such as replication. To represent an unbounded number of principals, a bound variable in a range is an index of a message representing its intended destination. To unify messages that cause the same behavior in a protocol, they are abstracted to a *parametric message* based on the type information, so that only a finite number of parametric messages are finally checked.

To represent an unbounded number of principals with which one may communicate, we assume that a principal may send a message to any of the principals. A binder is introduced such that a bound variable in it is regarded as an index of the possible destinations of the message. The variable is ranged over a set of principals' names. The usual way to restrict this kind of infinity is by bounding the number of principals in the network, so that each principal is described as only explicitly communicating with finitely many principals, including an intruder[1),3)].

Based on the type information, each subexpression whose type is a type variable will

† School of Information Science, Japan Advanced Institute of Science and Technology

be marked by a new kind of variable named a *parametric variable*. A parametric variable will not be further instantiated, since substructures of a message that are deeper than type variables do not affect behaviors of a principal. Thus the system is translated into a *parametric system*, in which all possible messages generated by each principal (including each intruder) can be simulated by finitely many parametric messages. In comparison,[3] imposes an upper-bound on the number of messages, restricting the state space to be finite.

In a parametric system, not all parametric traces have a corresponding trace, since a parametric trace may not have enough information to decide an equality between two parametric messages. Thus a refinement of the parametric trace procedure is proposed, in which if a parametric trace can be deduced to a *satisfiable normal form*, then it has a corresponding trace. The deduction procedure is decided dynamically. For this reason, we use on-the-fly model checking and implement the parametric system by Maude. The method successfully detects the flaws of several security protocols, such as the NSPK protocol and the Woo-Lam protocol, automatically. We use about 330 lines as the common part, and the protocol specific part for each protocol is about fifty lines in Maude.

The rest of the paper is organized as follows. In Section 2, an overview is introduced to illustrate how our system works. Section 3 presents the typed process calculus and its operational semantics. In Section 4, we introduce the parametric system, and prove its soundness and completeness with respect to the original system. In Section 5, we show how the security properties, such as secrecy and authentication, are represented and detected. Section 6 shows how to implement the parametric system using Maude. Section 7 presents related work, and in Section 8, we conclude the paper.

## 2. Overview

Because of the complexity of the whole system, we will illustrate how our system works by analyzing the Needham-Schroeder public-key protocol (referred to as NSPK protocol). An informal description of the NSPK protocol is given flow-by-flow as follows, in which $N_A$ and $N_B$ are nonces generated by $A$ and $B$, respectively.

$$A \longrightarrow B: \quad \{A, N_A\}_{+K_B} \qquad (1)$$
$$B \longrightarrow A: \quad \{N_A, N_B\}_{+K_A} \qquad (2)$$
$$A \longrightarrow B: \quad \{N_B\}_{+K_B} \qquad (3)$$

Our calculus is based on the Spi calculus[5]. Unlike the standard Spi calculus, this uses environment-based communication instead of channel-based communication. Each process sends messages to the environment and receives messages from the environment. Thus a message that a sender sends may be modified by intruders in the environment before the sender's intended receiver receives it.

In our calculus, the NSPK protocol is represented as follows:

$$\begin{aligned}
A \triangleq\ & (\nu x_a : \mathcal{I})\overline{a1}\{A, N_A\}_{+\mathtt{k}[x_a]}.a2(y_a).\\
& \mathit{case}\ y_a\ \mathit{of}\ \{y'_a\}_{-\mathtt{k}[A]}\ \mathit{in}\\
& \mathit{let}\ (z_a, z'_a) = y'_a\ \mathit{in}\\
& [z_a = N_A]\overline{a3}\{z'_a\}_{+\mathtt{k}[x_a]}.\mathbf{0}\\
B \triangleq\ & b1(x_b).\mathit{case}\ x_b\ \mathit{of}\{x'_b\}_{-\mathtt{k}[B]}\ \mathit{in}\\
& \mathit{let}\ (y_b, y'_b) = x'_b\ \mathit{in}\,[y_b = A]\\
& \overline{b2}\{y'_b, \mathtt{N_B}[A, B]\}_{+\mathtt{k}[A]}.b3(z_b).\\
& \mathit{case}\ z_b\ \mathit{of}\ \{u_b\}_{-\mathtt{k}[B]}\ \mathit{in}\\
& [u_b = \mathtt{N_B}[A, B]]\overline{acc}\, z_b.\mathbf{0}\\
SYS^{NSPK} \triangleq\ & A \| B
\end{aligned}$$

where

- $(\nu x_a : \mathcal{I})$ is a *range*. $x_a$ represents the possible destinations to which $A$ may send messages. It is ranged over the infinite set $\mathcal{I}$ of principals' names.
- $+\mathtt{k}[x_a]$ and $-\mathtt{k}[A]$ are key *binders* to represent any public key and $A$'s private key, respectively.
- $\mathtt{N_B}[A, B]$ is a nonce binder to represent the confidential datum $N_B$, which means that $B$ intends to send $N_B$ to $A$.
- $\overline{acc}\, z_b$ represents that after a sequence of checks, $B$ asserts that the message accepted by the $b3$ action comes from $A$.

A string of actions performed by principals, named a *trace*, is used to describe any possible run of protocols. In a trace, each variable in an input action is instantiated by a message that the environment generates. For example, a trace that represents a run with only the first two flows of the NSPK protocol, i.e. flow (1) and (2) is:

$\overline{a1}\{A, N_A\}_{+\mathtt{k}[B]}.b1(\{A, N_A\}_{-\mathtt{k}[B]}).$
$\overline{b2}\{N_A, \mathtt{N_B}[A, B]\}_{+\mathtt{k}[A]}.a2(\{N_A, \mathtt{N_B}[A, B]\}_{-\mathtt{k}[A]})$

It is well-known that the original NSPK protocol does not satisfy both secrecy and authentication by a man-in-middle attack[3], which is

given as:

$$A \longrightarrow I : \quad \{A, N_A\}_{+K_I} \quad \text{(A1)}$$
$$I(A) \longrightarrow B : \quad \{A, N_A\}_{+K_B} \quad \text{(B1)}$$
$$B \longrightarrow I(A) : \quad \{N_A, N_B\}_{+K_A} \quad \text{(B2)}$$
$$I \longrightarrow A : \quad \{N_A, N_B\}_{+K_A} \quad \text{(A2)}$$
$$A \longrightarrow I : \quad \{N_B\}_{+K_I} \quad \text{(A3)}$$
$$I(A) \longrightarrow B : \quad \{N_B\}_{+K_B} \quad \text{(B3)}$$

The confidential datum $N_B$ is leaked during the flow (A3), and thus the NSPK protocol violates the secrecy property. In (B3), $B$ "thinks" the message comes from $A$, while it actually comes from $I$, so this violates the authentication property.

In order to define these properties, some new expressions are defined: For the secrecy property, a guardian, $check(x)$, which means that any message that instantiates $x$ is observable, is introduced. The guardian can be inserted at any position in traces, to be instantiated by any messages leaked in each trace. Secrecy of $N_B$ is specified as $\neg check(\mathtt{N_B}[A, B])$. (The formal definition can be found in Subsection 5.1.)

The property that $A$ is authenticated to $B$ is specified as $\overline{a3}\,x \hookleftarrow \overline{acc}\,x$. The interpretation is that (1) if $\overline{acc}$ occurs in a trace, then the label $\overline{a3}$ must occur in the same trace before $\overline{acc}$, and (2) both $\overline{a3}$ and $\overline{acc}$ are attached to the same message. (The formal definition can also be found in Subsection 5.1.)

When one tries to detect such flaws, the main difficulty is that the number of runs is infinite. Such infinite nature is caused by: (1) an unlimited number of sessions of protocols, (2) an unlimited number of principals in the network, and (3) infinitely many messages that intruders may generate.

Our ideas are: (1) For an unlimited number of sessions, recursive definitions are avoided in our calculus, and thus only bounded number of sessions are considered. (2) For an unlimited number of principals, ranges and binders can represent communication with infinitely many principals, and later such communication is abstracted by a finite action. In comparison,[2),3)] impose an upper-bound on the number of principals. (3) For infinitely many messages, a parametric system is proposed to simulate the infinitely many traces with a finite number of parametric traces, while existing approaches impose an upper-bound on the number of messages [3)].

For an unlimited number of principals, a principal communicating to infinitely many other principals can be imitated by representing the intended destination by a bound variable in a range. The variable in the range later will not be instantiated in the parametric system, thus an unlimited number of principals can be extracted to a finite action. An alternative way to describe the communication is by using infinite process definition, such as replication, which is difficult to abstract to a finite system.

We apply a type system to abstract some unnecessary details of messages. Each subexpression whose type is a type variable will be marked by a new kind of variable named a parametric variable, which need not be further instantiated. Because substructures of a message that are deeper than type variables do not affect behaviors of principals. The process will not further decompose, decrypt or validate these submessages. By this abstraction, the resulting system is a parametric system (the formal definition is in Subsection 4.1). The NSPK protocol is translated into a parametric system as follows.

$$\begin{aligned}
A_p \triangleq\ & (\nu \hat{x}_a : \mathcal{I})\overline{a1}\{A, N_A\}_{+\mathtt{k}[\hat{x}_a]}. \\
& a2(\{\hat{z}_a, \hat{z}'_a\}_{-\mathtt{k}[\hat{x}_k]}). \\
& \mathit{case}\ \{\hat{z}_a, \hat{z}'_a\}_{-\mathtt{k}[\hat{x}_k]}\ \mathit{of}\ \{\hat{z}_a, \hat{z}'_a\}_{-\mathtt{k}[A]} \\
& \mathit{in\ let}\ (\hat{z}_a, \hat{z}'_a) = (\hat{z}_a, \hat{z}'_a)\ \mathit{in} \\
& [\hat{z}_a = N_A]\overline{a3}\{\hat{z}'_a\}_{+\mathtt{k}[\hat{x}_a]}.\mathbf{0} \\
B_p \triangleq\ & b1(\{\hat{x}_b, \hat{y}_b\}_{-\mathtt{k}[\hat{y}_k]}). \\
& \mathit{case}\ \{\hat{x}_b, \hat{y}_b\}_{-\mathtt{k}[\hat{y}_k]}\ \mathit{of} \\
& \{\hat{x}_b, \hat{y}_b\}_{-\mathtt{k}[B]}\ \mathit{in\ let}\ (\hat{x}_b, \hat{y}_b) = \\
& (\hat{x}_b, \hat{y}_b)\ \mathit{in}\ [\hat{x}_b = A] \\
& \overline{b2}\{\hat{y}_b, \mathtt{N_B}[A, B]\}_{+\mathtt{k}[A]}.b3(\{\hat{z}_b\}_{-\mathtt{k}[\hat{y}_k]}). \\
& \mathit{case}\ \{\hat{z}_b\}_{-\mathtt{k}[\hat{y}_k]}\ \mathit{of}\ \{\hat{z}_b\}_{-\mathtt{k}[B]}\ \mathit{in} \\
& [\hat{z}_b = \mathtt{N_B}[A, B]]\,\overline{acc}\,\{\hat{z}_b\}_{-\mathtt{k}[\hat{y}_k]}.\mathbf{0} \\
SYS_p^{NSPK} \triangleq\ & A_p \| B_p
\end{aligned}$$

In a parametric system, the number of parametric traces is finite. As Theorem 2 in Subsection 4.2 states, each trace in the original system has a corresponding parametric trace in its parametric system. However, not all parametric traces in a parametric system have a corresponding trace in the original system. To find whether a parametric trace has a corresponding trace, a refinement of parametric trace procedure is proposed, in which if a parametric trace can be deduced to a *satisfiable normal form*, then it has a corresponding trace (described by Theorem 3 in Subsection 4.3).

The secrecy and the authentication properties defined in an original system can also be defined equivalently in its corresponding parametric system and checked by performing a finite search (Theorems 4 and 5 in Subsection

5.2). For instance, in NSPK protocol, a counterexample to the secrecy property

$\overline{a1}\{A, N_A\}_{+\mathtt{k}[\hat{x}_a]}.b1(\{A, N_A\}_{-\mathtt{k}[B]}).$
$\overline{b2}\{N_A, \mathtt{N_B}[A, B]\}_{+\mathtt{k}[A]}.a2(\{N_A, \mathtt{N_B}[A, B]\}_{-\mathtt{k}[A]}).$
$\overline{a3}\{\mathtt{N_B}[A, B]\}_{+\mathtt{k}[\hat{x}_a]}.check(\mathtt{N_B}[A, B])$ (C1)

can be detected automatically. The counterexample shows that in $\overline{a3}$, $A$ may send the message he intends to send to $B$ to other principals (thus leaking the confidential message $\mathtt{N_B}[A, B]$). If we substitute $\hat{x}_a$ for an intruder name $I$, the same attack we have introduced in (A1–A3, B1–B3) is specified.

Similarly, we can detect a counterexample to the authentication

$\overline{a1}\{A, N_A\}_{+\mathtt{k}[\hat{x}_a]}.b1(\{A, N_A\}_{-\mathtt{k}[B]}).$
$\overline{b2}\{N_A, \mathtt{N_B}[A, B]\}_{+\mathtt{k}[A]}.a2(\{N_A, \mathtt{N_B}[A, B]\}_{-\mathtt{k}[A]}).$
$\overline{a3}\{\mathtt{N_B}[A, B]\}_{+\mathtt{k}[\hat{x}_a]}.b3(\{\mathtt{N_B}[A, B]\}_{-\mathtt{k}[B]}).$
$\overline{acc}\{\mathtt{N_B}[A, B]\}_{-\mathtt{k}[B]}$ (C2)

which means that $B$ thinks that he accepts the message from $A$, while actually $A$ can send the message to any one of possible principals. Similarly, if we substitute $\hat{x}_a$ for $I$, the trace also represents the same attack we have introduced in (A1–A3, B1–B3).

The fixed NSPK protocol[3] revises flow (2) to

$$B \longrightarrow A: \quad \{B, N_A, N_B\}_{+K_A} \qquad (2')$$

and avoids such attacks. In the flow (2′), $A$ will check whether the principal whom he intends to communicate with is identical to the principal name he has received. In an original system and in a parametric one, $A$ is represented as follows: (We omit the representation of $B$ here):

$A' \triangleq (\nu x_a : \mathcal{I})\overline{a1}\{A, N_A\}_{+\mathtt{k}[x_a]}.a2(y_a).$
$\quad case\ y_a\ of\ \{y'_a\}_{-\mathtt{k}[A]}\ in$
$\quad let\ (z_a, z'_a) = y'_a\ in\ [z_a = x_a]$
$\quad let\ (w_a, w'_a) = z'_a\ in$
$\quad [w_a = N_A]\overline{a3}\{w'_a\}_{+\mathtt{k}[z_a]}.\mathbf{0}$

$A'_p \triangleq (\nu \hat{x}_a : \mathcal{I})\overline{a1}\{A, N_A\}_{+\mathtt{k}[\hat{x}_a]}.$
$\quad a2(\{\hat{x}_a, \hat{w}_a, \hat{w}'_a\}_{-\mathtt{k}[\hat{x}_a]}).$
$\quad case\ \{\hat{x}_a, \hat{w}_a, \hat{w}'_a\}_{-\mathtt{k}[\hat{x}_k]}\ of$
$\quad \{\hat{x}_a, \hat{w}_a, \hat{w}'_a\}_{-\mathtt{k}[A]}\ in$
$\quad let\ (\hat{x}_a, \hat{w}_a, \hat{w}'_a) = (\hat{x}_a, \hat{w}_a, \hat{w}'_a)\ in$
$\quad [\hat{x}_a = \hat{x}_a]\ let\ (\hat{w}_a, \hat{w}'_a) = (\hat{w}_a, \hat{w}'_a)$
$\quad in\ [\hat{w}_a = N_A]\overline{a3}\{\hat{z}'_a\}_{+\mathtt{k}[\hat{w}'_a]}.\mathbf{0}$

With the match operation $[z_a = x_a]$, when we instantiate $z_a$ while generating a parametric trace, $x_a$ will be instantiated at the same time. Thus both the label $\overline{a3}$ and the label $\overline{acc}$ are attached to the same message. So following the same action order of the counterexample (C2), the trace is not a counterexample any more.

$\overline{a1}\{A, N_A\}_{+\mathtt{k}[B]}.b1(\{A, N_A\}_{-\mathtt{k}[B]}).$
$\overline{b2}\{B, N_A, \mathtt{N_B}[A, B]\}_{+\mathtt{k}[A]}.$
$a2(\{B, N_A, \mathtt{N_B}[A, B]\}_{-\mathtt{k}[A]}).$
$\overline{a3}\{\mathtt{N_B}[A, B]\}_{+\mathtt{k}[B]}.b3(\{\mathtt{N_B}[A, B]\}_{-\mathtt{k}[B]}).$
$\overline{acc}\{\mathtt{N_B}[A, B]\}_{-\mathtt{k}[B]}$

This means that $A$ will not send the message labeled $a3$ to any possible principal, because $A$ has accepted the communicator's name via $z_a$.

## 3. Process calculus with binders and types, and its trace

The syntax of our calculus is based on the Spi calculus[5]. We also introduce new syntax, binder and range.

### 3.1 Process and trace

Assume four countable disjoint sets: $\mathcal{L}$ for labels, $\mathcal{N}$ for names, $\mathcal{B}$ for binder names and $\mathcal{V}$ for variables. Let $a, b, c, \ldots$ indicate labels, $m, n, k, \ldots$ indicate names, $\mathtt{m}, \mathtt{n}, \mathtt{k}, \ldots$ indicate binder names, and $x, y, z, \ldots$ indicate variables.

Messages $M, N, L \ldots$ in a set $\mathcal{M}$ are defined as follows:

$$M, N, L ::= \quad n \mid x \mid (M, N) \mid \{M\}_L \mid \mathtt{m}[M_1, \ldots, M_n]$$

$(M, N)$ represents a pair of which each element is a message. $\{M\}_L$ is an encrypted message where $M$ is its plain message and $L$ is its encryption key. A binder $\mathtt{m}[M_1, \ldots, M_n]$ is regarded as a special name with some relation to other messages. $M_1, \ldots, M_n$ are parameters of $\mathtt{m}$. For simplicity, we usually use $\tilde{M}$ to represent a tuple of messages, and thus a binder $\mathtt{m}[M_1, \ldots, M_n]$ can also be denoted as $\mathtt{M}[\tilde{M}]$. One usage of binders is to represent encryption keys. For instance, binder $\mathtt{k}[A, S]$ represents a symmetric key shared with principals $A$ and $S$; $+\mathtt{k}[A]$ and $-\mathtt{k}[A]$ represent $A$'s public key and private key, respectively. $+\mathtt{k}[x]$ represents any public key in the network. We say a message $M$ is in a message $N$, if $M$ is a subterm of $N$; a message is ground, if it does not contain any variable.

Let $\mathcal{P}$ be a countable set of processes which is indicated by $P, Q, R, \ldots$. The syntax of pro-

cesses is defined as follows:

$P, Q, R ::=$

| | |
|---|---|
| $\mathbf{0}$ | Nil |
| $\overline{a}M.P$ | output |
| $a(x).P$ | input |
| $[M = N]\,P$ | match |
| $(\nu x : \mathcal{A})P$ | range |
| $let\ (x, y) = M\ in\ P$ | pair splitting |
| $case\ M\ of\ \{x\}_L\ in\ P$ | decryption |
| $P \| Q$ | composition |

Intuitively understanding,

- $\mathbf{0}$ is *Nil* process that does nothing.
- $\overline{a}M.P$ sends message $M$ to the environment and then behaves like $P$.
- $a(x).P$ awaits an input message $M$ and behaves like $P\{M/x\}$.
- If $M = N$, $[M = N]\,P$ acts as $P$; Otherwise it will be stuck.
- $(\nu x : \mathcal{A})P$ means that $x$ in $P$ ranges over $\mathcal{A}$ ($\subseteq \mathcal{N}$), and acts as $P\{m/x\}$, where $m \in \mathcal{A}$.
- If $M$ is a pair $(N, L)$, $let\ (x, y) = M\ in\ P$ is reduced to $P\{N/x, L/y\}$; Otherwise it will be stuck.
- Process $case\ M\ of\ \{x\}_L\ in\ P$ is reduced to $P\{N/x\}$ when $M$ is an encrypted message $\{N\}_{L'}$ that $L$ can decrypt; Otherwise it will be stuck.
- $P \| Q$ means that $P$ and $Q$ run concurrently.

Variables $x$ and $y$ are bound in $a(x).P$, $(\nu x : \mathcal{A})P$, $let\ (x, y) = M\ in\ P$, and $case\ M\ of\ \{x\}_L\ in\ P$. We denote the sets of free variables and bound variables in $P$ by $f_v(P)$ and $b_v(P)$, respectively. A process $P$ is closed if $f_v(P) = \emptyset$.

A process is used to represent behaviors of each principal in a security protocol. Here, we take a naive example; a more complex example can be found in Subsection 3.4.

**Example 1.** *(Wide-mouthed frog protocol) A principal $A$ shares a shared-key $\mathtt{k}[A, S]$ with a server $S$, and another principal $B$ shares a shared-key $\mathtt{k}[B, S]$ with $S$. The purpose of the protocol is to establish a new secret key $\mathtt{k}[A, B]$ between $A$ and $B$, which $A$ may use to send a confidential datum $M$ to $B$. The protocol flows and its representation by above syntax $SYS^{WMF}$, are described below:*

$$A \longrightarrow S : \{K_{AB}\}_{K_{AS}}$$
$$S \longrightarrow B : \{K_{AB}\}_{K_{BS}}$$
$$A \longrightarrow B : \{M\}_{K_{AB}}$$

$$A \triangleq \overline{a1}\{\mathtt{k}[A, B]\}_{\mathtt{k}[A,S]}.\overline{a2}\{M\}_{\mathtt{k}[A,B]}.\mathbf{0}$$
$$B \triangleq b1(x).case\ x\ of \{y\}_{\mathtt{k}[B,S]}\ in\ b2(z).case\ z\ of\ \{u\}_y\ in.\mathbf{0}$$
$$S \triangleq s1(x).case\ x\ of\ \{y\}_{\mathtt{k}[A,S]}\ in\ \overline{s2}\{y\}_{\mathtt{k}[B,S]}.\mathbf{0}$$
$$SYS^{WMF} \triangleq A \| S \| B$$

Messages that the environment can generate are started from the current finite knowledge, denoted by $\mathcal{S}$ ($\subseteq \mathcal{M}$), and deduced by a deductive system. Here, we presuppose a countable set $\mathcal{E}$ ($\subseteq \mathcal{M}$), for those environmental names and ground binders such as public keys, intruders' names and their keys, etc. For example, $I, -\mathtt{k}[I'], \mathtt{k}[I, S], +\mathtt{k}[A] \ldots \in \mathcal{E}$. Let $\rhd$ be the least binary relation generated by the deductive system in Fig. 1:

An *action* is a term of form $\overline{a}M$ or $a(M)$, in which $M$ is a message. *Act* is defined as an action set. An action is ground if its attached message is ground. A string of ground actions can represent a possible run of the protocol when each input message can be deduced by messages in its prefix string. We named such a kind of string *concrete trace*, or a *trace*. represented by $s$. The messages in a trace $s$, represented by $msg(s)$, are those messages in output actions of the trace $s$. We use $s \rhd M$ to represent $msg(s) \rhd M$.

**Definition 1.** *A* concrete trace[☆] *$s$ is a ground action string $s \in Act^*$ such that $s = s'.a(M).s''$ implies $s' \rhd M$ for each $s'$, $s''$ and $a(M)$. A* concrete configuration *is a pair $\langle s, P\rangle$, in which $s$ is a trace and $P$ is a closed process.*

### 3.2 Operational Semantics

The transition relation of configurations is defined by the rules in Fig. 2, in which each rule has the following form: $\langle s, P\rangle \longrightarrow \langle s', P'\rangle\ \mathcal{C}$, meaning that if a condition $\mathcal{C}$ is satisfied, $\langle s, P\rangle$ will transit to $\langle s', P'\rangle$. Note that in rules $LCOM$ and $RCOM$, no reaction is provided between two composed processes, and both processes communicate with the environment. Furthermore, a function $\mathtt{Opp}$ is defined for complementary key in decryption and encryption. Thus we have $\mathtt{Opp}(+\mathtt{k}[A]) = -\mathtt{k}[A]$, $\mathtt{Opp}(-\mathtt{k}[A]) = +\mathtt{k}[A]$ and $\mathtt{Opp}(\mathtt{k}[A, B]) = \mathtt{k}[A, B]$.

The rules $INPUT$ and $RANGE$ may lead to an infinite system. For the former, let's take a process $A \triangleq \overline{a1}M.a2(x).\mathbf{0}$ for example. $\langle \epsilon, A\rangle$ will transit to $\langle \overline{a1}M, a2(x).\mathbf{0}\rangle$ by the $OUTPUT$ rule, which can then transit to in-

[☆] Before Section 4, a concrete trace is identified with a trace, for simplicity.

$$\frac{}{S \rhd n}\ n \in \mathcal{E}\ \ Env \qquad \frac{}{S \rhd M}\ M \in S\ \ Ax$$

$$\frac{S \rhd M \quad S \rhd N}{S \rhd (M,N)}\ Pair\_intro \qquad \frac{S \rhd (M,N)}{S \rhd M}\ Pair\_elim1 \qquad \frac{S \rhd (M,N)}{S \rhd N}\ Pair\_elim2$$

$$\frac{S \rhd \{M\}_{\mathtt{k}[A,B]} \quad S \rhd \mathtt{k}[A,B]}{S \rhd M}\ Senc\_elim \qquad \frac{S \rhd M \quad S \rhd \mathtt{k}[A,B]}{S \rhd \{M\}_{\mathtt{k}[A,B]}}\ Senc\_intro$$

$$\frac{S \rhd \{M\}_{\pm\mathtt{k}[A]} \quad S \rhd \mp\mathtt{k}[A]}{S \rhd M}\ Penc\_elim \qquad \frac{S \rhd M \quad S \rhd \pm\mathtt{k}[A]}{S \rhd \{M\}_{\pm\mathtt{k}[A]}}\ Penc\_intro$$

**Fig. 1** Environmental deductive system

$$(INPUT)\quad \langle s, a(x).P\rangle \longrightarrow \langle s.a(M), P\{M/x\}\rangle \ \ s \rhd M$$
$$(OUTPUT)\quad \langle s, \overline{a}M.P\rangle \longrightarrow \langle s.\overline{a}M, P\rangle$$
$$(DEC)\quad \langle s, case\ \{M\}_L\ of\ \{x\}_{L'}\ in\ P\rangle \longrightarrow \langle s, P\{M/x\}\rangle \quad L' = \mathtt{Dpp}(L)$$
$$(PAIR)\quad \langle s, let\ (x,y) = (M,N)\ in\ P\rangle \longrightarrow \langle s, P\{M/x, N/y\}\rangle$$
$$(RANGE)\quad \langle s, (\nu x : \mathcal{A})P\rangle \longrightarrow \langle s, P\{m/x\}\rangle \quad m \in \mathcal{A}$$
$$(MATCH)\quad \langle s, [M = M]P\rangle \longrightarrow \langle s, P\rangle$$
$$(LCOM)\quad \frac{\langle s, P\rangle \longrightarrow \langle s', P'\rangle}{\langle s, P\|Q\rangle \longrightarrow \langle s', P'\|Q\rangle}$$
$$(RCOM)\quad \frac{\langle s, Q\rangle \longrightarrow \langle s', Q'\rangle}{\langle s, P\|Q\rangle \longrightarrow \langle s', P\|Q'\rangle}$$

**Fig. 2** Transition rules

finitely many configurations. Since the trace $\overline{a1}M$ can deduce infinitely many messages, and according to the $INPUT$ rule, each message can instantiate $x$. Thus it generates infinitely many traces. For the latter, let's take another process $B \triangleq (\nu x : \mathcal{I})\overline{b1}\{M\}_{+\mathtt{k}[x]}$ to illustrate the infinity, where $\mathcal{I}$ is an infinite set of principals' names. According to the $RANGE$ rule, $x$ will be instantiated to any name in $\mathcal{I}$, which thereafter leads to infinitely many traces by $OUTPUT$ rules.

### 3.3 Type

The type information of an input variable can be inferred through looking up the process. A message whose type cannot unify the type of an input variable will be stuck when a protocol runs. Stuck messages cannot attack a protocol, so we will exclude such messages from being checked.

Let $\mathcal{T}$ be the set of types. Its syntax is inductively defined as follows:

$$\tau ::= \alpha \mid b \mid \tau * \tau \mid \sigma[\tau_1, \ldots, \tau_n] \mid \ominus\tau \mid \ominus_+\tau \mid \ominus_-\tau \mid \ominus_?\tau \mid unit \mid \tau + \tau \mid \tau \to \tau$$

- $\alpha$ ranges over a countable set of type variables.
- $b$ ranges over the set of base types, which consists of an identity type $i$ for names of principals, a nonce type $n$ for nonces, and other kinds of base types, for instance, $int$, $char$, etc.
- The pair type $\tau * \tau$ is given to a pair message.
- The binder type $\sigma[\tau_1, \ldots, \tau_n]$ is given to a binder $\mathtt{m}[M_1, \ldots, M_n]$, where $\sigma$ ranges over the set of binder name types, and $\tau_1, \ldots, \tau_n$ are the types of the binder's parameters $M_1, \ldots, M_n$, respectively. For example $k, k_+, k_-, \ldots$ are binder name types for binder names $\mathtt{k}, +\mathtt{k}, -\mathtt{k}$ respectively, so the type of a binder $\mathtt{k}[A,B]$ is $k[i,i]$. Since given a binder name type, its parameters' types will be fixed, for simplicity, we usually use a binder name type to represent a binder's type. For instance, the type of $\mathtt{k}[A,B]$ can be abbreviated to $k$.
- The shared-key encryption type, $\ominus\tau$, is given to an encrypted message encrypted by a shared-key, and $\tau$ is the type for its plain message. Similarly, $\ominus_+\tau$ is for a public-key encrypted message, and $\ominus_-\tau$ is for a private-key encrypted message, say, a digital signature. $\ominus_?\tau$ is for an encrypted message whose key cannot be decided statically.
- The type $unit$ is a nil type for the $\mathbf{0}$ process.
- The disjoint type $\tau_1 + \tau_2$ is given to a composition process, $P\|Q$.
- The arrow type $\tau \to \tau$ is given to an input process, which is similar to the type of the abstraction in $\lambda$-calculus.

We use an expression $e$ where $e \in \mathcal{M} \cup \mathcal{P}$ to describe a message or a process. Let $\Gamma$ be a type environment mapping from the set of variables $\mathcal{V}$, to the set of types $\mathcal{T}$. The typing inference

$$\frac{}{\Gamma \vdash x : \tau}\ (x,\tau) \in \Gamma \quad \text{Msg\_Variable} \qquad \frac{}{\Gamma \vdash n : b}\ b = \texttt{TypeOf}(n) \quad \text{Msg\_Name}$$

$$\frac{\Gamma \vdash \tilde{M} : \tau_1 * \ldots * \tau_n}{\Gamma \vdash \mathtt{m}[\tilde{M}] : \sigma[\tau_1, \ldots, \tau_n]}\ \sigma = \texttt{TypeOf}(\mathtt{m}) \ \text{Msg\_Term} \qquad \frac{\Gamma \vdash M : \tau_1 \quad \Gamma \vdash N : \tau_2}{\Gamma \vdash (M,N) : \tau_1 * \tau_2}\ \text{Msg\_Pair}$$

$$\frac{\Gamma \vdash M : \tau \quad \Gamma \vdash L : k^*}{\Gamma \vdash \{M\}_L : \ominus^* \tau}\qquad \ominus^* \tau = \begin{cases} \ominus\tau & k^* = k \\ \ominus_+\tau & k^* = k_+ \\ \ominus_-\tau & k^* = k_- \\ \ominus_?\tau & k^* = \alpha \end{cases}\ \text{Msg\_Enc}$$

$$\frac{}{\Gamma \vdash \mathbf{0} : unit}\ \text{Nil} \qquad \frac{\Gamma, \{x : \tau_1\} \vdash P : \tau_2}{\Gamma \vdash (\nu x : \mathcal{A})P : \tau_2}\ \tau_1 = \texttt{TypeOf}(\mathcal{A}) \quad \text{Range}$$

$$\frac{\Gamma, \{x : \tau_1\} \vdash P : \tau_2}{\Gamma \vdash a(x).P : \tau_1 \to \tau_2}\ \text{Input} \qquad \frac{\Gamma \vdash P : \tau_1 \quad \Gamma \vdash M : \tau_2}{\Gamma \vdash \overline{a}M.P : \tau_1}\ \text{Output}$$

$$\frac{\Gamma, \{x : \tau_1, y : \tau_2\} \vdash P : \tau_3 \quad \Gamma \vdash M : \tau_1 * \tau_2}{\Gamma \vdash let\ (x,y) = M\ in\ P : \tau_3}\ \text{Pair}$$

$$\frac{\Gamma, \{x : \tau_1\} \vdash P : \tau_2 \quad \Gamma \vdash M : \ominus^* \tau_1 \quad \Gamma \vdash L : k^*}{\Gamma \vdash case\ M\ of\ \{x\}_L\ in\ P : \tau_2}\qquad \ominus^* \tau = \begin{cases} \ominus\tau & k^* = k \\ \ominus_+\tau & k^* = k_- \\ \ominus_-\tau & k^* = k_+ \\ \ominus_?\tau & k^* = \alpha \end{cases}\ \text{Dec}$$

$$\frac{\Gamma \vdash M : \tau_1 \quad \Gamma \vdash N : \tau_1 \quad \Gamma \vdash P : \tau_2}{\Gamma \vdash [M = N]P : \tau_2}\ \text{Match} \qquad \frac{\Gamma \vdash P : \tau_1 \quad \Gamma \vdash N : \tau_2}{\Gamma \vdash P \| Q : \tau_1 + \tau_2}\ \text{Composition}$$

**Fig. 3** Typing rules

system has the form $\Gamma \vdash e : \tau$, in which $\Gamma$ is a type environment, $e$ is the expression whose type will be inferred and $\tau$ is the type of $e$. If the type environment is an empty set, the form will be abbreviated to $\vdash e : \tau$. Furthermore, we assume a function $\texttt{TypeOf} : \mathrm{P}(\mathcal{N}) \cup \mathrm{P}(\mathcal{B}) \to \mathcal{T}$ that assigns a type to a set of names or binder names. Here $\texttt{TypeOf}(n)$ is the abbreviation of $\texttt{TypeOf}(\{n\})$. The typing rules for the expressions are given in Fig. 3.

Among the transition rules in Fig. 2, the $INPUT$ rule needs to be modified with a type constraint, that is, a message $M$ deduced by the $s$ in the configuration $\langle s, a(x).P\rangle$, whose type can be unified with the type of $x$, will be instantiated to $x$.

$$(TINPUT)\quad \langle s, a(x).P : \tau_1 \to \tau_2\rangle \longrightarrow \langle s.a(M), P\{M/x\}\rangle \quad s \rhd M, \vdash M : \tau_1$$

Even with the type constraint, the rule $TINPUT$ may also lead to an infinity of a system, since a type variable can be unified to any type, thus a variable whose type is a type variable can be instantiated to any possible message.

The typing system does not provide an easy way to assign a type to an expression $e$. Thus a type algorithm is provided and its correctness is verified. In the algorithm, given a typing environment $\Gamma$ and an expression $e$, a substitution $\theta$ mapping from type variables to types and a type $\tau$ can be calculated, which satisfy

$$\Gamma\theta \vdash e : \tau$$

Before defining the algorithm, we provide a type unification algorithm, which can be used both in the type inference algorithm and the message type matching in $TINPUT$ rule. An occurrence check function $FTV(\tau, \alpha)$ is presupposed as usual, which satisfies that $\alpha$ does not occur in $\tau$ if $FTV(\tau, \alpha) = True$. The unification algorithm has the following form $\texttt{Unify}(\tau, \tau') = (\theta, \sigma)$. That is, given two types $\tau$ and $\tau'$, it either returns a substitution $\theta$ and a type $\sigma$ that satisfy $\tau\theta = \tau'\theta = \sigma$, or raises failure.

The `Unify` and `Infer` algorithms are given in Appendix A.1.

We will prove that every type calculated by the algorithm `Infer` can be inferred by the typing rules in Fig. 3.

**Lemma 1.** *Let $e$ be an expression, $\tau$ be a type, $\Gamma$ be a type environment and $\theta$ be a substitution mapping from type variables to types. If $\Gamma \vdash e : \tau$, then $\Gamma\theta \vdash e : \tau\theta$.*

*Proof.* Applying structural induction to the expression $e$, we only show the base cases and two inductive steps here; the remaining cases are quite similar.

(1) Case $e = x$: Because $\Gamma \vdash x : \tau$, we have $(x, \tau) \in \Gamma$. Let $\Gamma_1 = \Gamma \backslash \{(x, \tau)\}$, so $\Gamma\theta = \Gamma_1\theta \cup \{(x, \tau)\}\theta = \Gamma_1\theta \cup \{(x, \tau\theta)\}$, and thus $\Gamma\theta \vdash x : \tau\theta$ according to typing rule

$Msg\_Variable$.

(2) Case $e = n$: Because $\Gamma \vdash n : \tau$, we have $\tau = \texttt{TypeOf}(n)$, and $\tau\theta = \tau$ for every substitution $\theta$. By typing rule $Msg\_Name$, $\Gamma\theta \vdash n : \texttt{TypeOf}(n)$.

(3) Case $e = (M, N)$: By typing rules, $\Gamma \vdash M : \tau_1$ and $\Gamma \vdash N : \tau_2$ are satisfied. Given a substitution $\theta$, we have $\Gamma\theta \vdash M : \tau_1\theta$ and $\Gamma\theta \vdash M : \tau_2\theta$ according to induction hypothesis. So $\Gamma\theta \vdash (M, N) : (\tau_1 * \tau_2)\theta$.

(4) Case $e = a(x).P$: By induction hypothesis, $\Gamma\theta \cup \{x, \tau_1\}\theta = \Gamma\theta \cup \{x, \tau_1\theta\} \vdash P : \tau_2\theta$ is satisfied. Thus $\Gamma\theta \vdash a(x)P : (\tau_1 \to \tau_2)\theta$.

□

**Theorem 1 (Soundness of type inference).** *Let $e$ be an expression, $\Gamma$ be a typing environment and $\theta$ be a substitution. If $\texttt{Infer}(\Gamma, e) = (\theta, \tau)$, $\Gamma\theta \vdash e : \tau$ can be inferred.*

*Proof.* Applying structural induction to the expression $e$. one base step and three inductive steps are proposed; the remaining cases are simple and similar. We use the same notations as in the algorithm.

(1) Case $e = x$: We have $Infer(\Gamma, x) = (Id, \tau)$, because $\Gamma Id = \Gamma$ and $(x, \tau) \in \Gamma$, $\Gamma \vdash x : \tau$ is inferred by the typing rule $Msg\_Variable$.

(2) Case $e = (M, N)$: by induction hypothesis, we have two derivations:

$$Infer(\Gamma, M) = (\theta_1, \tau_1) \mapsto \Gamma\theta_1 \vdash M : \tau_1 \quad (1)$$

$$Infer(\Gamma\theta_1, N) = (\theta_2, \tau_2) \mapsto \Gamma\theta_1\theta_2 \vdash N : \tau_2 \quad (2)$$

By applying Lemma 1 to (1), we have

$$\Gamma\theta_1\theta_2 \vdash M : \tau_1\theta_2 \quad (3)$$

By applying typing rule $Msg\_Pair$ to (2) and (3), we have

$$\Gamma\theta_1\theta_2 \vdash (M, N) : \tau_1\theta_2 * \tau_2$$

which is the expected result of

$$Infer(\Gamma, (M, N)) = (\theta_1\theta_2, (\tau_1\theta_2) * \tau_2)$$

(3) Case $e = a(x).P$: By induction hypothesis, we get a derivation:

$$Infer(\Gamma \cup \{(x, \alpha)\}, P) = (\theta, \tau) \mapsto \Gamma\theta \cup \{(x, \alpha)\}\theta \vdash P : \tau \quad (4)$$

By applying typing rule $Input$ to (4), we have

$$\Gamma\theta \vdash a(x).P : \alpha\theta \to \tau$$

which is the result of $Infer(\Gamma, a(x).P) = (\theta, \alpha\theta \to \tau)$.

(4) Case $e = case\ M\ of\ \{x\}_L\ in\ P$: By the induction hypothesis, we have two derivations and a unification:

$$\texttt{Infer}(\Gamma, M) = (\theta_1, \tau_1) \mapsto \Gamma\theta_1 \vdash M : \tau_1 \quad (5)$$

$$\texttt{Infer}(\Gamma\theta_1, L) = (\theta_2, \tau_2) \mapsto \Gamma\theta_1\theta_2 \vdash L : \tau_2 \quad (6)$$

$$\texttt{Unify}(\ominus^*\alpha, \tau_1\theta_2) = (\vartheta, \varrho) \mapsto \ominus^*(\alpha\vartheta) = \tau_1\theta_2\vartheta = \varrho \quad (7)$$

$$\texttt{Infer}(\Gamma\theta_1\theta_2\vartheta \cup \{(x, \alpha)\}\theta_2\vartheta, P) = (\theta_3, \tau_3) \mapsto \Gamma\theta_1\theta_2\vartheta\theta_3 \cup \{(x, \alpha)\vartheta\theta_3\} \vdash P : \tau_3 \quad (8)$$

By applying Lemma 1 to (5), we have

$$\Gamma\theta_1\theta_2\vartheta\theta_3 \vdash M : \tau_1\theta_2\vartheta\theta_3 \quad (9)$$

Note that $\alpha$ does not occur in $\theta_1$ and $\theta_2$, and thus $\{(x, \alpha)\}\vartheta\theta_3 = \{(x, \alpha)\}\theta_1\theta_2\vartheta\theta_3$. So after unification, the type of $x$ is $\alpha\vartheta\theta_2$, and the type of $\{x\}_L$ is $\ominus^*(\alpha\vartheta\theta_2)$. And because of (7), we have

$$\ominus^*(\alpha\vartheta\theta_3) = \tau_1\theta_2\vartheta\theta_3$$

and thus

$$\Gamma\theta_1\theta_2\vartheta\theta_3 \vdash M : \ominus^*(\alpha\vartheta\theta_3)$$

Furthermore, because of (7),

$$\Gamma\theta_1\theta_2\vartheta\theta_3 \vdash L : \tau_2\vartheta_2\theta_3$$

So, using typing rule $Dec$, we have

$$\Gamma\theta_1\theta_2\vartheta\theta_3 \vdash case\ M\ of\ \{x\}_L\ in\ P : \tau_3$$

which is the result of

$$Infer(\Gamma, case\ M\ of\ \{x\}_L\ in\ P) = (\theta_1\theta_2\vartheta\theta_3, \tau_3)$$

□

### 3.4 Representation of security protocols using binders

In this subsection, we will describe how to represent a protocol with the purpose of model checking some security properties. We assume there are infinitely many principals in the network, and consider two arbitrary principals $A$ and $B$ who are willing to communicate with each other, as well as with any other principals by a protocol. The secrecy property means during the communication, a confidential message will not be leaked to other principals except $A$ and $B$. The authentication property means that when $B$ thinks it has received a message from $A$, the message really comes from $A$.

According to these two properties, a principal who intends to send a message is supposed to send the message to any one of the possible principals in the network, if he cannot gain the information about the principal to whom he intends to send the message. An abstraction will

be employed by using a range and some binders to finitely describe such an assumption. That is, the principal may send the same message to different principals, and such a sending procedure is performed only once. An alternative way to describe the communication is by using infinite process definition, such as replication. By this way, a principal with intention to communicate with infinitely many principals can be described as the principal that communicates with each principal in different sessions. It is difficult to abstract to a finite system.

As a receiver, we will fix its potential sender to the sender we represented in one session. That is, a receiver will "think" he has received the message from some principal he has known. Such an assumption is necessary when defining authentication and secrecy properties, since otherwise the sender and the receiver we represented may have no connections with each other, and thus these properties between them can not be defined. For example, if $A$ sent a message to $C$, and $B$ received a message from $D$, it is certain that the message $B$ received is different from the message $A$ sent. In Subsection 5.3, when we define the security properties in multiple sessions, we will loosen the restriction, assuming that other than in one session, the receiver can communicate with any principal. With these assumptions, we define secrecy and authentication properties in multiple sessions.

Let's consider Abadi-Gordon protocol introduced in[5] as an example. The informal description of the protocol is given flow-by-flow as follows:

$$\begin{array}{ll} A \longrightarrow S: & A, \{B, K_{AB}\}_{K_{AS}} \\ S \longrightarrow B: & \{A, K_{AB}\}_{K_{SB}} \\ A \longrightarrow B: & A, \{A, M\}_{K_{AB}} \end{array}$$

Intuitively interpreting, the principal $A$ wants to send a message $M$ to $B$ encrypted by a new key $K_{AB}$ that he generates. Firstly, he sends the $K_{AB}$ and $B$'s name to a trusted third party(TTP) server $S$. After the TTP sends the new key to $B$, $A$ sends a message encrypted by the key $K_{AB}$ to $B$. We will represent the protocol as follows:

$$\begin{array}{l} A \triangleq (\nu x : \mathcal{I})\overline{a1}(A, \{x, \mathtt{k}[A,x]\}_{\mathtt{k}[A,S]}). \\ \quad \overline{a2}(A, \{A, M\}_{\mathtt{k}[A,x]}).\mathbf{0} \\ B \triangleq b1(x).case\ x\ of\ \{x'\}_{\mathtt{k}[B,S]}\ in \\ \quad let\ (y,z) = x'\ in\ [y = A]\, b2(w). \\ \quad let\ (w', w'') = w\ in\ [w' = A] \\ \quad case\ w''\ of\ \{u\}_z\ in \\ \quad let\ (u', u'') = u\ in\ [u' = A]\, F(u'') \\ S \triangleq s1(x).let\ (y,z) = x\ in \\ \quad case\ z\ of\ \{u\}_{\mathtt{k}[y,S]}\ in \\ \quad let\ (u', u'') = u\ in\ \overline{s2}\{y, u''\}_{\mathtt{k}[u',S]}.\mathbf{0} \\ SYS^{AG} \triangleq A \| S \| B \end{array}$$

A range and a binder are used when a principal sends a message that contains the information of its intended receiver, and the principal cannot gain such information by previous communications. For example, during the communication of the Abadi-Gordon protocol, the principal $A$ cannot obtain any information of its corresponding receiver. Thus a binder $\mathtt{k}[A, x]$ is used to describe that $A$ can communicate with any one of principals in the network. Furthermore, a range $(\nu x : \mathcal{I})$ is used to bind the variable $x$ to an infinite set $\mathcal{I}$, which contains the names of all principals in the network.

$F(u'')$ in (5) means that $B$ runs a process $F$ with the message that it "thinks" came from $A$. Usually, we define the process $F$ as a declaration of accepting a specific message from the intended principal. For instance, in this protocol, we define $F(u'') = \overline{acc}\, w.\mathbf{0}$, meaning principal $B$ thinks itself successfully accepting the message through $w$ from $A$.

## 4. Parametric system

The typed system has reduced the number of traces by excluding messages whose type cannot unify the type of an input variable. However, the typed system may still be infinite, since a variable (or a sub-expression) whose type is a type variable can be instantiated to any possible message. For example, a process $P$ is defined as follows:

$$P \triangleq a1(x).\overline{a2}x.\mathbf{0}$$

After inferring the type of $P$, the type of $x$ is a type variable $\alpha$, which means a configuration $\langle s, P\rangle$ can transit to $\langle s.a1(M), \overline{a2}\, M.\mathbf{0}\rangle$ for any message that satisfies $s \rhd M$. However, these messages have the same effect on the following actions of the process, since the receiver process will not further decompose, decrypt or validate the message that was received, before it sends the message through $\overline{a2}$. To reduce the system, a special variable named a *parametric variable* $\hat{x}$ is proposed to mark a sub-expression with a type variable as its type. A parametric variable will not be further instantiated. In the above example, the trace will become $s.a1(\hat{x}).\overline{a2}\hat{x}$ when $P$ transits to $\mathbf{0}$, which

is finite. According to the above approach, a new system named a *parametric system* is introduced to simulate the previous system.

### 4.1 Parametric process and parametric trace

We use a new set $\hat{\mathcal{V}}$ for parametric variables and assume $\hat{\mathcal{V}} \cap \mathcal{V} = \emptyset$. As convention, elements in $\hat{\mathcal{V}}$ are denoted by $\hat{x}, \hat{y}, \ldots$. *Parametric messages* in a set $\hat{\mathcal{M}}$ and *parametric processes* in a set $\hat{\mathcal{P}}$ can be defined as follows:

$$\begin{aligned} \hat{M} ::= \;& \hat{x} \mid n \mid (\hat{M}, \hat{N}) \mid \{\hat{M}\}_{\hat{L}} \\ & \mid \mathtt{m}[\hat{M}_1, \hat{M}_2, \ldots \hat{M}_1] \\ \hat{P} ::= \;& \mathbf{0} \mid a(\hat{M}).\hat{P} \mid \overline{a}\hat{M}.\hat{P} \mid [\hat{M} = \hat{N}]\hat{P} \\ & \mid (\nu \hat{x} : \mathcal{A})\hat{P} \mid let\ (\hat{M}, \hat{N}) = \hat{L}\ in\ \hat{P} \\ & \mid case\ \hat{M}\ of\ \{\hat{N}\}_{\hat{L}}\ in\ \hat{P} \mid \hat{P} \| \hat{Q} \end{aligned}$$

Given a closed process $P$, we try to mark each sub-expression whose type is a type variable with a parametric variable. Thus $P$ can be translated into a parametric process $\hat{P}$. For this purpose, given a closed process $P$ and its type $\tau$, an inference system is proposed, which infers a substitution $\theta$ mapping from $\mathcal{V}$ to $\hat{\mathcal{M}}$. The inference system has the form $\Delta \vdash_p P : \tau \Rightarrow \theta$, in which $\Delta$ is a context environment mapping from $\mathcal{V}$ to $\hat{\mathcal{M}}$, $P$ is a closed process and $\tau$ is its type. The corresponding parametric process $\hat{P}$ is obtained by applying the substitution $\theta$ to the process $P$. We name $\hat{P}$ the *abstraction* of $P$, and $P$ the *concretization* of $\hat{P}$. Note that there do not exist any variables in a parametric process, since each variable is substituted to a parametric message.

In the inference system, some functions are predefined. $\mathtt{NewParVar} : \Delta \to \hat{\mathcal{V}}$ generates a new parametric variable that does not occur in $\mathtt{Dom}(\Delta)$. $\mathtt{Binder} : \mathcal{T} \to \mathcal{I}$ obtains the corresponding binder name from a binder type. The inference system is given in Fig. 4.

By the inference system, the formal definition of Abadi-Gordon protocol described in Subsection 3.4 will be translated into the following parametric processes:

$$\begin{aligned} A_p \triangleq\; & (\nu \hat{x}_1 : \mathcal{I})\overline{a1}(A, \{\hat{x}_1, \mathtt{k}[A, \hat{x}_1]\}_{\mathtt{k}[A,S]}). \\ & \overline{a2}(A, \{A, M\}_{\mathtt{k}[A,\hat{x}_1]}).\mathbf{0} \\ B_p \triangleq\; & b1(\{\hat{y}_1, \hat{z}_1\}_{\mathtt{k}[\hat{x}_k, \hat{y}_k]}).\ case\ \{\hat{y}_1, \hat{z}_1\}_{\mathtt{k}[\hat{x}_k,\hat{y}_k]} \\ & of\ \{\hat{y}_1, \hat{z}_1\}_{\mathtt{k}[B,S]}\ in\ let\ (\hat{y}_1, \hat{z}_1) = \\ & (\hat{y}_1, \hat{z}_1)\ in\ [\hat{y}_1 = A]\, b2(\hat{w}_1, \{\hat{w}_1', \hat{w}_1''\}_{\hat{t}_1}). \\ & let\ (\hat{w}_1, (\hat{w}_1, \{\hat{w}_1', \hat{w}_1''\}_{\hat{t}_1})) = \\ & (\hat{w}_1, (\hat{w}_1, \{\hat{w}_1', \hat{w}_1''\}_{\hat{t}_1}))\ in \\ & [\hat{w}_1 = A]\ case\ \{\hat{w}_1', \hat{w}_1''\}_{\hat{t}_1}\ of\ \{\hat{w}_1', \hat{w}_1''\}_{\hat{z}_1} \\ & in\ let\ (\hat{w}_1', \hat{w}_1'') = (\hat{w}_1', \hat{w}_1'')\ in\ [\hat{w}_1' = A] \\ & \overline{acc}(\hat{w}_1, \{\hat{w}_1', \hat{w}_1''\}_{\hat{t}_1}) \end{aligned}$$

$$\begin{aligned} S_p \triangleq\; & s1(\hat{x}, \{\hat{y}, \hat{z}\}_{\mathtt{k}[\hat{x}_k,S]}).let\ (\hat{x}, \{\hat{y}, \hat{z}\}_{\mathtt{k}[\hat{x}_k,S]}) \\ & = (\hat{x}, \{\hat{y}, \hat{z}\}_{\mathtt{k}[\hat{x}_k,S]})\ in\ case\ \{\hat{y}, \hat{z}\}_{\mathtt{k}[\hat{x}_k,S]} \\ & of\ \{\hat{y}, \hat{z}\}_{\mathtt{k}[\hat{x},S]}\ in\ let\ (\hat{y}, \hat{z}) = (\hat{y}, \hat{z})\ in \\ & \overline{s2}\{\hat{x}, \hat{z}\}_{\mathtt{k}[\hat{y},S]}.\mathbf{0} \\ SYS_p^{AG} \triangleq\; & A_p \| S_p \| B_p \end{aligned}$$

Similar to a concrete system, a *parametric action* is a term of form $\overline{a}\hat{M}$ or $a(\hat{M})$, in which $\hat{M}$ is a parametric message. A *parametric trace* is a string of parametric actions. Note that in a concrete trace, any message in an input action should be deduced by the prefix trace of the input action. However, a parametric trace may not have enough information to decide an equality between two parametric messages, and thus whether a parametric message is deducible is unknown. Thus we loosen this restriction and define a parametric trace as follows. Such a loosening may lead to a divergence between a trace and a parametric trace, which will be discussed later.

**Definition 2.** *A* parametric trace $\hat{s}$ *is a string of parametric actions. A pair* $\langle \hat{s}, \hat{P} \rangle$ *is* a parametric configuration *if* $\hat{s}$ *is a parametric trace and* $\hat{P}$ *is a parametric process.*

Since parametric variables are not instantiated during model transitions, the equality check (in $MATCH$ and $DEC$) of two messages cannot be judged explicitly. Instead, a parametric message unification function $\mathtt{Uni}$, whose parameters enjoy the same type, is applied to them. The transitions of a parametric system are given in Fig. 5.

Given a parametric trace $\hat{s}$, if there exists a substitution $\vartheta$ that assigns each parametric variable to a ground message, and satisfies $s = \hat{s}\vartheta$, where $s$ is a concrete trace, we say that $s$ is a *concretization* of $\hat{s}$ and $\hat{s}$ is an *abstraction* of $s$. $\vartheta$ is named *concretized ground substitution.*

### 4.2 Sound and complete simulation

We hope that each concrete trace in a concrete system has an abstraction in its corresponding parametric system, and that each parametric trace in the parametric system has at least one concretization, so a bisimulation relation can be defined between them. However, although each concrete trace does have an abstraction, some parametric traces may have no concretizations. Let's take a simple example, suppose a process $P$ is defined as $P \triangleq a1(x).[x = b]\,\overline{a2}\,x.\mathbf{0}$, in the concrete system $\langle \epsilon, P \rangle$, $\overline{a2}$ will never occur in traces during transitions since any process will be stuck when

$$\frac{}{\Delta \vdash_p x : \alpha \Rightarrow \{\hat{x}/x\}}\ \hat{x} = \texttt{NewParVar}(\Delta)\ \text{Type\_variable}$$

$$\frac{}{\Delta \vdash_p x : b \Rightarrow \{\hat{x}_b/x\}}\ \hat{x}_b = \texttt{NewParVar}(\Delta)\ \text{Base\_type}$$

$$\frac{\Delta \vdash_p x' : \tau_1 * \ldots * \tau_n \Rightarrow \theta}{\Delta \vdash_p x : \sigma[\tau_1, \ldots, \tau_n] \Rightarrow \{\mathtt{m}[\theta(x')]/x\}}\ \mathtt{m} = \texttt{Binder}(\sigma), x' = \texttt{NewParVar}(\Delta)\ \text{Binder\_type}$$

$$\frac{\Delta \vdash_p x' : \tau_1 \Rightarrow \theta_1 \quad \Delta \vdash_p x'' : \tau_2 \Rightarrow \theta_2}{\Delta \vdash_p x : \tau_1 * \tau_2 \Rightarrow \{(\theta_1(x'), \theta_2(x''))/x\}}\ x', x'' = \texttt{NewParVar}(\Delta)\text{Pair\_type}$$

$$\frac{\Delta \vdash_p x' : \tau \Rightarrow \theta}{\Delta \vdash_p x : \ominus\tau \Rightarrow \{\{\theta(x')\}_{\mathtt{k}[\hat{x}_a, \hat{x}_b]}/x\}}\ \hat{x}_a, \hat{x}_b, x' = \texttt{NewParVar}(\Delta)\ \text{Sencryption\_type}$$

$$\frac{\Delta \vdash_p x' : \tau \Rightarrow \theta}{\Delta \vdash_p x : \ominus_+\tau \Rightarrow \{\{\theta(x')\}_{+\mathtt{k}[\hat{x}_a]}/x\}}\ \hat{x}_a, x' = \texttt{NewParVar}(\Delta)\ \text{Pencryption\_type}$$

$$\frac{\Delta \vdash_p x' : \tau \Rightarrow \theta}{\Delta \vdash_p x : \ominus_-\tau \Rightarrow \{\{\theta(x')\}_{-\mathtt{k}[\hat{x}_a]}/x\}}\ \hat{x}_a, x' = \texttt{NewParVar}(\Delta)\ \text{Signature\_type}$$

$$\frac{\Delta \vdash_p x' : \tau \Rightarrow \theta}{\Delta \vdash_p x : \ominus_?\tau \Rightarrow \{\{\theta(x')\}_{\hat{x}_a}/x\}}\ \hat{x}_a, x' = \texttt{NewParVar}(\Delta)\ \text{Gencryption\_type}$$

$$\frac{}{\Delta \vdash_p \mathbf{0} : \mathtt{unit} \Rightarrow \{\}}\text{Nil} \qquad \frac{\Delta \vdash_p x : \tau_1 \Rightarrow \theta_1 \quad \Delta, \theta_1 \vdash_p P : \tau_2 \Rightarrow \theta_2}{\Delta \vdash_p a(x).P : \tau_1 \to \tau_2 \Rightarrow \theta_1 \cup \theta_2}\text{Input}$$

$$\frac{\Delta \vdash_p P : \tau \Rightarrow \theta}{\Delta \vdash_p \overline{a}M.P : \tau \Rightarrow \theta}\text{Output} \qquad \frac{\Delta \vdash_p P : \tau \Rightarrow \theta}{\Delta \vdash_p (\nu x : \mathcal{A})\, P : \tau \Rightarrow \theta \cup \{\hat{x}/x\}}\ \hat{x} = \texttt{NewParVar}(\Delta)\ \text{Range}$$

$$\frac{\Delta, \{\hat{M}_1/x, \hat{M}_2/y\} \vdash_p P : \tau \Rightarrow \theta}{\Delta \vdash_p let\ (x,y) = M\ in\ P : \tau \Rightarrow \theta \cup \{\hat{M}_1/x, \hat{M}_2/y\}}\ (\Delta(M) = (\hat{M}_1, \hat{M}_2))\text{Pair}$$

$$\frac{\Delta, \{\hat{M}/x\} \vdash_p P : \tau \Rightarrow \theta}{\Delta \vdash_p case\ M\ of \{x\}_L\ in\ P : \tau \Rightarrow \theta \cup \{\hat{M}/x\}}\ (\Delta(M) = \{\hat{M}\}_{\hat{L}})\text{Dec}$$

$$\frac{\Delta \vdash_p P : \tau \Rightarrow \theta}{\Delta \vdash_p [M = N]P : \tau \Rightarrow \theta}\ \text{Match} \qquad \frac{\Delta \vdash_p P : \tau_1 \Rightarrow \theta_1 \quad \Delta \vdash_p Q : \tau_2 \Rightarrow \theta_2}{\Delta \vdash_p P \| Q : \tau_1 + \tau_2 \Rightarrow \theta_1 \cup \theta_2}\text{Composition}$$

**Fig. 4** Inference rules for parametric process

$$(PINPUT)\quad \langle \hat{s}, a(\hat{M}).\hat{P}\rangle \longrightarrow_p \langle \hat{s}.a(\hat{M}), \hat{P}\rangle$$

$$(POUTPUT)\quad \langle \hat{s}, \overline{a}\hat{M}.\hat{P}\rangle \longrightarrow_p \langle \hat{s}.\overline{a}\hat{M}, \hat{P}\rangle$$

$$(PDEC)\quad \langle \hat{s}, case\ \{\hat{M}\}_{\hat{L}}\ of\ \{\hat{M}\}_{\hat{L}'}\ in\ \hat{P}\rangle \longrightarrow_p \langle \hat{s}\hat{\theta}, \hat{P}\hat{\theta}\rangle\ \ \hat{\theta} = \texttt{Uni}(\hat{L}, \texttt{Opp}(\hat{L}'))$$

$$(PPAIR)\quad \langle \hat{s}, let\ (\hat{M}, \hat{N}) = (\hat{M}, \hat{N})\ in\ \hat{P}\rangle \longrightarrow_p \langle \hat{s}, \hat{P}\rangle$$

$$(PRANGLE)\quad \langle \hat{s}, (\nu\hat{x} : \mathcal{A})\hat{P}\rangle \longrightarrow_p \langle \hat{s}, \hat{P}\rangle$$

$$(PMATCH)\quad \langle \hat{s}, [\hat{M} = \hat{M}']\hat{P}\rangle \longrightarrow_p \langle \hat{s}\hat{\theta}, \hat{P}\hat{\theta}\rangle \qquad \hat{\theta} = \texttt{Uni}(\hat{M}, \hat{M}')$$

$$(PLCOM)\quad \frac{\langle \hat{s}, \hat{P}\rangle \longrightarrow_p \langle \hat{s}', \hat{P}'\rangle}{\langle \hat{s}, \hat{P}\|\hat{Q}\rangle \longrightarrow_p \langle \hat{s}', \hat{P}'\|\hat{Q}\rangle}\quad \hat{Q}' = \hat{Q}\theta \text{ if } \hat{s}' = \hat{s}\theta \text{ else } \hat{Q}' = \hat{Q}$$

$$(PRCOM)\quad \frac{\langle \hat{s}, \hat{Q}\rangle \longrightarrow_p \langle \hat{s}', \hat{Q}'\rangle}{\langle \hat{s}, \hat{P}\|\hat{Q}\rangle \longrightarrow_p \langle \hat{s}', \hat{P}'\|\hat{Q}'\rangle}\quad \hat{P}' = \hat{P}\theta \text{ if } \hat{s}' = \hat{s}\theta \text{ else } \hat{P}' = \hat{P}$$

**Fig. 5** Parametric transition relation

$[x = b]$ is considered, since the trace cannot deduce the name $b$. However, a parametric trace $a1(b).\overline{a2}(b)$ is in its corresponding parametric system.

However, a parametric system can still cover its concrete one. That is: if a parametric trace has a concretization, then the concretization is a trace in its counterpart concrete system, otherwise the parametric trace cannot be instantiated to any concrete trace. As shown in the above example, parametric trace $a1(b).\overline{a2}(b)$ cannot be instantiated to any concrete trace since $\epsilon \not\vdash b$. Here we explain the soundness and completeness theorem.

**Theorem 2.** *(Soundness and completeness) Let $\langle \epsilon, P\rangle$ be a configuration, $s'$ be a trace, and $\hat{P}$ be the abstraction of $P$, then $\langle \epsilon, P\rangle \longrightarrow^* \langle s', P'\rangle$ for some $P'$, if and only if there exists $\hat{s}'$, such that $\langle \epsilon, \hat{P}\rangle \longrightarrow_p^* \langle \hat{s}', \hat{P}'\rangle$ for some $\hat{P}'$, and $s'$ is a concretization of $\hat{s}'$.*

*Proof.* "⇒": By an induction on the number of transitions $\longrightarrow$ and $\longrightarrow_p$, the proof is trivial in the zero-step. We assume in the $n$-th step the

property holds. That is, for each trace $s$ gained in the $n$-th $\longrightarrow$ step, there exists an $\hat{s}$ obtained by the $n$-th $\longrightarrow_p$ step, and $\hat{s}\vartheta = s$ holds for some substitution $\vartheta$ from parametric variables to ground messages. Now, we perform a case analysis on the $n+1$ step:

(1) Case $\langle s, 0\rangle$: Obviously.

(2) Case $\langle s, a(x).P\rangle$: If $\langle s, a(x).P\rangle \longrightarrow \langle s.a(M), P\{M/x\}\rangle$, then the type of the ground message $M$ can be unified with the type $\tau$ of $x$. So after applying the inference system in Fig. 4 to the process, $x$ in the corresponding parametric process will be substituted to a parametric message $\hat{M}$ that can be unified by $M$. Let $\vartheta' = \mathtt{Uni}(M, \hat{M})$, and $\hat{P}$ be the abstraction of $P$, then we have $\langle \hat{s}, a(\hat{M}).\hat{P}\rangle \longrightarrow_p \langle \hat{s}.a(\hat{M}), \hat{P}\rangle$ and $s.a(M) = \hat{s}.a(\hat{M})(\vartheta \cup \vartheta')$.

(3) Case $\langle s, \overline{a}M.P\rangle$: Compared with the transition $\langle s, \overline{a}M.P\rangle \longrightarrow \langle s.\overline{a}M, P\rangle$, the parametric configuration has the transition $\langle \hat{s}, \overline{a}\hat{M}.\hat{P}\rangle \longrightarrow \langle \hat{s}.\overline{a}\hat{M}, \hat{P}\rangle$. Since each parametric variable in $\hat{M}$ is already in the domain of $\vartheta$, we have $M = \hat{M}\vartheta$ and $s.\overline{a}M = (\hat{s}.\overline{a}\hat{M})\vartheta$.

(4) Case $\langle s, let\ (x,y) = (M,N)\ in\ P\rangle$: Obviously.

(5) Case $\langle s, case\ \{M\}_L\ of\{x\}_{L'}\ in\ P\rangle$: If $\langle s, case\ \{M\}_L\ of\{x\}_{L'}\ in\ P\rangle$ can transit to $\langle s, P\rangle$ where $L' = \mathtt{Opp}(L)$, and its counterpart parametric configuration is $\langle \hat{s}, case\ \{\hat{M}\}_{\hat{L}}\ of\{\hat{M}\}_{\hat{L}'}\ in\ \hat{P}\rangle$, then the parametric variables in $\hat{L}$ and $\hat{L}'$ are in the domain of $\vartheta$. Thus $L = \hat{L}\vartheta$ and $L' = \hat{L}'\vartheta$. So $s\vartheta = (\hat{s}\hat{\theta})\vartheta$ where $\hat{\theta} = \mathtt{Uni}(\hat{L}, \mathtt{Opp}(\hat{L}'))$.

(6) Case $\langle s, [M = M]P\rangle$: If $\langle s, [M = M]P\rangle \longrightarrow \langle s, P\rangle$ and its counterpart configuration is $\langle \hat{s}, [\hat{M} = \hat{M}']\hat{P}\rangle$, then the parametric variables in $\hat{M}$ and $\hat{M}'$ are in the domain of $\vartheta$, and variable$\hat{M}\vartheta = \hat{M}'\vartheta = M$. Thus if $\hat{\theta} = \mathtt{Uni}(\hat{M}, \hat{M}')$, then $\hat{\theta} \subseteq \vartheta$ since the $\hat{\theta}$ is the most general unifier of $\hat{M}$ and $\hat{M}'$ and $\vartheta$ is a unifier of them. So we have $s\vartheta = (\hat{s}\hat{\theta})\vartheta$.

(7) $\langle s, (\nu x : \mathcal{A})P\rangle$: Then we have $\langle s, (\nu x : \mathcal{A})P\rangle \longrightarrow \langle s, P\{m/x\}\rangle$ for each $m \in \mathcal{A}$. Its counterpart configuration is $\langle \hat{s}, (\nu \hat{x} : \mathcal{A})\hat{P}$ and $s = \hat{s}(\vartheta \cup \{m/\hat{x}\})$.

(8) Case $\langle s, P \| Q\rangle$: Obviously.

"$\Leftarrow$": By an induction on the number of transitions $\longrightarrow_p$ and $\longrightarrow$, the proof is trivial in the zero-step. We assume in the $n$-th step the property holds, that is, for each parametric trace $\hat{s}$ gained by the $n$-th $\longrightarrow_p$ step, if there exists a substitution $\vartheta$ from parametric variables to ground messages, and a trace $s$ that satisfies $s = \hat{s}\vartheta$, then $s$ can be obtained by the $n$-th step of $\longrightarrow$. Now, we perform a case analysis on the $n+1$-th step:

(1) Case $\langle \hat{s}, 0\rangle$: obviously.

(2) Case $\langle \hat{s}, a(\hat{M}).\hat{P}\rangle$: If there exists a step in which $\langle \hat{s}, a(\hat{M}).\hat{P}\rangle \longrightarrow_p \langle \hat{s}.a(\hat{M}).\hat{P}\rangle$, and a ground substitution $\vartheta$ where $\hat{s}\vartheta$ is a trace, then $\hat{M}\vartheta$ is a ground message which can be deduced by $s\vartheta$, and its type can be unified by the type of $x$ because of the inference system in Fig. 4. So $\langle s, a(x).P\rangle \longrightarrow \langle s.a(\hat{M}\vartheta), P\{\hat{M}\vartheta/x\}\rangle$.

(3) Case $\langle \hat{s}, \overline{a}\hat{M}.\hat{P}\rangle$: We have $\langle \hat{s}, \overline{a}\hat{M}.\hat{P}\rangle \longrightarrow_p \langle \hat{s}.\overline{a}\hat{M}, \hat{P}\rangle$. Note that there exists a ground substitution $\vartheta$, and there do not exist any fresh parametric variables in $\hat{M}$ due to *Output* rule in the inference system in Fig. 4, so we have $\langle \hat{s}\vartheta, \overline{a}\hat{M}\vartheta.P\rangle \longrightarrow \langle (\hat{s}.\overline{a}\hat{M})\vartheta, P\rangle$.

(4) Case $\langle \hat{s}, let\ (\hat{M},\hat{N}) = (\hat{M},\hat{N})\ in\ \hat{P}\rangle$: Obviously.

(5) Case $\langle \hat{s}, case\ \{\hat{M}\}_{\hat{L}}\ of\{\hat{M}\}_{\hat{L}'}\ in\ \hat{P}\rangle$: We have $\langle \hat{s}, case\ \{\hat{M}\}_{\hat{L}}\ of\{\hat{M}\}_{\hat{L}'}\ in\ \hat{P}\rangle \longrightarrow_p \langle \hat{s}\hat{\theta}, \hat{P}\hat{\theta}\rangle$, where $\hat{\theta} = \mathtt{Uni}(\hat{L}, \mathtt{Opp}(\hat{L}'))$, and there exists a substitution $\vartheta$. So in $\hat{s}\hat{\theta}$, $\hat{L}\hat{\theta} = \mathtt{Opp}(\hat{L}'\hat{\theta})$. Furthermore, in the inference system in Fig. 4, *Dec* rule does not introduce any new parametric variables in $\hat{L}$ and $\hat{L}'$, then $\hat{L}\hat{\theta}\vartheta = \mathtt{Opp}(\hat{L}'\hat{\theta}\vartheta)$ and thus the counterpart transition can be performed.

(6) Case $\langle \hat{s}, [\hat{M} = \hat{M}']\hat{P}\rangle$: cf. the former case analysis, after applying the substitution $\hat{\theta} = \mathtt{Uni}(\hat{M}, \hat{M}')$ to both the parametric trace and process, $\hat{M}\hat{\theta} = \hat{M}'\hat{\theta}$ will be the same, and there do not exist any new parametric variables in them. By applying the ground substitution $\vartheta$, $\hat{M}\hat{\theta}\vartheta = \hat{M}'\hat{\theta}\vartheta$. Thus the counterpart transition will be performed successfully.

(7) The last two cases, *New* and *Composition*, are obvious.

□

### 4.3 Satisfiable normal form

Theorem 2 shows that each trace in a concrete system has an abstraction in its parametric system. However, a parametric trace may

not have concretizations. Let's take a simple example to show the reason for the divergence.

**Example 2.** *Consider a naive protocol:*

$$A \longrightarrow B : \{A, M\}_{K_{AB}}$$

*In its parametric system, there exists a parametric trace* $b1(\{A, \hat{x}\}_{\mathtt{k}[A,B]})$*, while in its concrete system, since* $\mathtt{k}[A, B]$ *was not leaked in the environment, before* $A$ *sends the message* $\{A, M\}_{\mathtt{k}[A,B]}$*,* $B$ *cannot accept any message encrypted by* $\mathtt{k}[A, B]$*. Thus, the parametric trace* $b1(\{A, \hat{x}\}_{\mathtt{k}[A,B]})$ *has no concretizations.*

We name a message like $\{A, \hat{x}\}_{\mathtt{k}[A,B]}$ a *rigid message*. Intuitively interpreting, a rigid message in the pattern of a requirement in an input action. The requirement can only be satisfied by the messages generated by a proper principal, not by intruders. These messages are contained in an output action of a parametric trace. If there are no appropriate messages to satisfy the requirement, then the parametric trace has no concretizations. The definition of a rigid message is as follows:

**Definition 3 (Rigid message).** *Given a parametric trace* $\hat{s} = \hat{s}'.a(\hat{M}).\hat{s}''$*,* $\{\hat{N}\}_{\hat{L}} \in \hat{M}$ *is a rigid message if the following conditions are satisfied:*

- $\hat{L}$ *is a ground binder, and there exists a name, a binder or a rigid message in* $\hat{N}$*;*
- *If* $\hat{L}$ *is a shared key, then* $\hat{s}' \not\vdash \hat{L}$ *and* $\hat{s}' \not\vdash \{\hat{N}\}_{\hat{L}}$*;*
- *If* $\hat{L}$ *is a private key, then there exists some rigid message, or at least one name or binder in* $\hat{N}$ *cannot be deduced by the* $\hat{s}'$*, and* $\hat{s}' \not\vdash \{\hat{N}\}_{\mathtt{Opp}(\hat{L})}$*;*
- *If* $\hat{L}$ *is a public key, then* $\hat{s}' \not\vdash \mathtt{Opp}(\hat{L})$ *and* $\hat{s}' \not\vdash \{\hat{N}\}_{\mathtt{Opp}(\hat{L})}$*.*

Some researchers also regard an encrypted message where a variable is encrypted by shared key as a rigid message[2)], for example, to represent a protocol through which $A$ sends to $B$ an encrypted message, $\{M\}_{\mathtt{k}[A,B]}$. One of the parametric traces will be $\overline{a1}\{M\}_{\mathtt{k}[A,B]}.b1(\{\hat{x}\}_{\mathtt{k}[A,B]})$. It seems $\hat{x}$ can only be substituted by $M$, and thus $\{\hat{x}\}_{\mathtt{k}[A,B]}$ is a rigid message. However, the communicated messages are nothing but bit streams in the network. In such a case, any bit stream with the same length as $\{M\}_{\mathtt{k}[A,B]}$ can fake the message, since without comparing some plain message to other messages already known, $B$ cannot distinguish whether the plain message will be meaningful after decrypting the message he receives. So in our definition, $\{\hat{x}\}_{\mathtt{k}[A,B]}$ is not a rigid message.

A parametric trace with a rigid message needs to be further substituted by trying to unify a rigid message to the messages in output actions of its prefix parametric traces. Such unification procedures will terminate because each rigid message can only be unified by the irreducible messages in some output actions of its prefix parametric trace, and the number of these messages is finite. We name these messages *elementary messages*, and use $el(\hat{s})$ to represent the set of elementary messages in $\hat{s}$. Here is the definition of $el(\hat{s})$.

**Definition 4 (Elementary messages).** *Let* $\hat{\mathcal{U}}$ *be a set of parametric messages,* $dec(\hat{\mathcal{U}})$ *is a minimal set that satisfies*

- $\hat{\mathcal{U}} \subseteq dec(\hat{\mathcal{U}})$*;*
- *If* $(\hat{M}, \hat{N}) \in dec(\hat{\mathcal{U}})$*, then* $\hat{M}, \hat{N} \in dec(\hat{\mathcal{U}})$*;*
- *If* $\{\hat{M}\}_{\hat{L}} \in dec(\hat{\mathcal{U}})$*,* $\hat{L}$ *is ground, and* $\mathtt{Opp}(\hat{L}) \in dec(\hat{\mathcal{U}})$*, then* $\hat{M}, \hat{L} \in dec(\hat{\mathcal{U}})$*;*
- *If* $\{\hat{M}\}_{\hat{L}} \in dec(\hat{\mathcal{U}})$*, and* $\hat{L}$ *is not ground, then* $\hat{M} \in dec(\hat{\mathcal{U}})$*.*

*Given a parametric trace* $\hat{s}$*, let* $out(\hat{s})$ *be the set of all parametric messages in output actions of* $\hat{s}$*, then* $el(\hat{s})$ *is the set of minimal terms with respect to the subterm relation in* $dec(out(\hat{s}))$*.*

Given a parametric trace $\hat{s}$ and a parametric message $\hat{N}$, we say $\hat{N}$ is $\hat{\rho}$*-unifiable* in $\hat{s}$, if there exists $\hat{N}' \in el(\hat{s})$ such that $\hat{\rho} = \mathtt{Uni}(\hat{N}, \hat{N}')$. A parametric trace deductive relation between two parametric traces, $\hat{s} \rightsquigarrow \hat{s}'$ is defined as follows:

**Definition 5 (Deductive relation).** *Let* $\rightsquigarrow$ *be the least binary relation of two parametric traces and* $\hat{s}$ *be a parametric trace such that* $\hat{s} = \hat{s}_1.a(\hat{M}).\hat{s}_2$*. If there exists a rigid message* $\hat{N}$ *in* $\hat{M}$ *such that* $\hat{N} \notin el(\hat{s}_1)$*, and* $\hat{N}$ *is* $\hat{\rho}$*-unifiable in* $\hat{s}_1$*, then* $\hat{s} \rightsquigarrow \hat{s}\hat{\rho}$*.*

For two parametric traces $\hat{s}$ and $\hat{s}'$, if $\hat{s} \rightsquigarrow^* \hat{s}'$ and there is no $\hat{s}''$ that satisfies $\hat{s}' \rightsquigarrow \hat{s}''$, we name $\hat{s}'$ the normal form of $\hat{s}$. The set of normal forms of $\hat{s}$ is denoted by $\mathtt{nf}_{\rightsquigarrow}(\hat{s})$.

**Remark 1.** *Given a parametric trace* $\hat{s}$*,* $\mathtt{nf}_{\rightsquigarrow}(\hat{s})$ *is finite.*

**Example 3.** *One of the parametric traces generated by the Abadi-Gordon protocol described in Subsection 3.4 is as follows. By the deductive relation, it has deduced to a normal form.*

$$\overline{a1}(A, \{\hat{x}_1, \mathtt{k}[A, \hat{x}_1]\}_{\mathtt{k}[A,S]}).\overline{a2}(A, \{A, M\}_{\mathtt{k}[A,\hat{x}_1]}).$$
$$s1(\hat{x}, \{\hat{y}, \hat{z}\}_{\mathtt{k}[\hat{x},S]}).\overline{s2}\{\hat{x}, \hat{z}\}_{\mathtt{k}[\hat{y},S]}.b1(\{A, \hat{t}_1\}_{\mathtt{k}[B,S]}).$$
$$b2(A, \{A, \hat{w}_1''\}_{\hat{t}_1}).\overline{acc}(A, \{A, \hat{w}_1''\}_{\hat{t}_1})$$
$$\rightsquigarrow$$

$\overline{a1}(A,\{\hat{x}_1,\mathtt{k}[A,\hat{x}_1]\}_{\mathtt{k}[A,S]}).\overline{a2}(A,\{A,M\}_{\mathtt{k}[A,\hat{x}_1]}).$
$s1(A,\{B,\hat{z}\}_{\mathtt{k}[A,S]}).\overline{s2}\{A,\hat{z}\}_{\mathtt{k}[B,S]}.b1(\{A,\hat{z}\}_{\mathtt{k}[B,S]}).$
$b2(A,\{A,\hat{w}_1''\}_{\hat{z}}).\overline{acc}\,(A,\{A,\hat{w}_1''\}_{\hat{z}})$
$\rightsquigarrow$
$\overline{a1}(A,\{B,\mathtt{k}[A,B]\}_{\mathtt{k}[A,S]}).\overline{a2}(A,\{A,M\}_{\mathtt{k}[A,B]}).$
$s1(A,\{B,\mathtt{k}[A,B]\}_{\mathtt{k}[A,S]}).\overline{s2}\{A,\mathtt{k}[A,B]\}_{\mathtt{k}[B,S]}.$
$b1(\{A,\mathtt{k}[A,B]\}_{\mathtt{k}[B,S]}).b2(A,\{A,\hat{w}_1''\}_{\mathtt{k}[A,B]}).$
$\overline{acc}\,(A,\{A,\hat{w}_1''\}_{\mathtt{k}[A,B]})$
$\rightsquigarrow$
$\overline{a1}(A,\{B,\mathtt{k}[A,B]\}_{\mathtt{k}[A,S]}).\overline{a2}(A,\{A,M\}_{\mathtt{k}[A,B]}).$
$s1(A,\{B,\mathtt{k}[A,B]\}_{\mathtt{k}[A,S]}).\overline{s2}\{A,\mathtt{k}[A,B]\}_{\mathtt{k}[B,S]}.$
$b1(\{A,\mathtt{k}[A,B]\}_{\mathtt{k}[B,S]}).b2(A,\{A,M\}_{\mathtt{k}[A,B]}).$
$\overline{acc}\,(A,\{A,M\}_{\mathtt{k}[A,B]})$

By the following lemma, we can see that a concretization of a parametric trace $\hat{s}$ is still the concretization of $\hat{s}'$ if $\hat{s} \rightsquigarrow \hat{s}'$. Thus, to decide whether a parametric trace has concretizations just requires checking whether there exists some parametric trace in its $\mathtt{nf}_{\rightsquigarrow}(\hat{s})$ that has concretizations.

**Lemma 2.** *If $\hat{s}$ is a parametric trace, and $s$ is a concretization satisfying $s = \hat{s}\vartheta$ where $\vartheta$ is a concretized ground substitution, then $\hat{s}$ is either a normal form, or there exists an $\hat{s}'$ such that $\hat{s} \rightsquigarrow \hat{s}'$ with $\hat{s}\vartheta = \hat{s}'\vartheta$.*

*Proof.* Let $\hat{s} = \hat{s}'.a(\hat{M}).\hat{s}''$. If $\hat{s}$ is not a normal form, there exists some rigid message $\{\hat{N}\}_{\hat{L}}$ in $\hat{M}$, such that $\{\hat{N}\}_{\hat{L}} \notin el(\hat{s}')$. Since $s = \hat{s}\vartheta$ and $s$ is a trace, and thus $\hat{s}'\vartheta \rhd \hat{M}\vartheta$, then $\{\hat{N}\}_{\hat{L}}\vartheta \in el(\hat{s}'\vartheta)$. By the definition of a rigid message, $\hat{L} \notin el(\hat{s}')$, and thus $\hat{L}\vartheta \notin el(\hat{s}')\vartheta$. Since $\{\hat{N}\}_{\hat{L}}\vartheta \in el(\hat{s}'\vartheta) = el(\hat{s}')\vartheta$, there exists $\{\hat{N}'\}_{\hat{L}} \in el(\hat{s}')$ such that $\{\hat{N}\}_{\hat{L}}\vartheta = \{\hat{N}'\}_{\hat{L}}\vartheta$. Thus $\{\hat{N}\}_{\hat{L}}$ and $\{\hat{N}'\}_{\hat{L}}$ are unifiable. Let $\hat{\rho} = \mathtt{Uni}(\{\hat{N}\}_{\hat{L}},\{\hat{N}'\}_{\hat{L}})$, then $\hat{s} \rightsquigarrow \hat{s}\hat{\rho}$. Since $\{\hat{N}\}_{\hat{L}}\vartheta = \{\hat{N}'\}_{\hat{L}}\vartheta$, each corresponding parametric variable in two messages will be assigned to the same ground message. Thus, $\hat{s}\vartheta = \hat{s}\hat{\rho}\vartheta$. □

**Lemma 3.** *Let $\hat{s}$ be a parametric trace, and $\hat{s}'$ be a normal form in $\mathtt{nf}_{\rightsquigarrow}(\hat{s})$. $\hat{s}'$ has a concretization, if and only if, for each decomposition $\hat{s}' = \hat{s}'_1.a(\hat{M}).\hat{s}'_2$,*

- *each rigid message $\hat{N}$ in $\hat{M}$ satisfies $\hat{N} \in el(\hat{s}'_1)$, and*
- *each name $n$, and ground binder $\mathtt{m}[\tilde{N}]$ in $\hat{M}$ in $\hat{P}_a$ satisfies $n,\mathtt{m}[\tilde{N}] \in el(\hat{s}'_1)$, where $\hat{P}_a$ is the process containing label $a$.*

*Proof.* "⇒": Prove by contradiction. Assume a normal form $\hat{s}'$ has concretizations $s$ such that $s = \hat{s}'\vartheta$. If $\hat{s}'$ does not satisfy the first requirement, there exists at least one rigid message $\{\hat{N}\}_L$ in $\hat{s}'$ that is not $\hat{\rho}$-unifiable in its prefix $\hat{s}'_1$. Thus $\{\hat{N}\}_L\vartheta \notin el(\hat{s}'_1)\vartheta$. By definition of a rigid message, $\hat{s}'_1\vartheta \not\rhd L$, then $\hat{s}'_1\vartheta \not\rhd \{\hat{N}\}_L\vartheta$. This contradicts the definition of a trace. If $\hat{s}'$ does not satisfy the second requirement, that is, there exists either a name $n$, or a ground binder $\mathtt{m}[\tilde{N}]$ in $\hat{M}$ that is local in $\hat{P}_a$ and $n,\mathtt{m}[\tilde{N}] \in el(\hat{s}'_1)$. Then $\hat{s}'_1\vartheta \not\rhd n,\mathtt{m}[\tilde{N}]$, and thus $\hat{s}'_1\vartheta \not\rhd \hat{M}\vartheta$. This again contradicts the definition of a trace.

"⇐": Since the first occurrence of a parametric variable is in an input action, let $\vartheta$ be an arbitrary concretized ground substitution that assigns each parametric variable in $\hat{s}'$ to a name in $\mathcal{EN}$, then for each decomposition $\hat{s}'\vartheta = \hat{s}'_1\vartheta.a(\hat{M}\vartheta).\hat{s}'_2\vartheta$, $\hat{s}'_1\vartheta \rhd \hat{M}\vartheta$ is satisfiable. Thus $\hat{s}'\vartheta$ is a trace, and also a concretization of $\hat{s}'$ □

We name a normal form of $\hat{s}$ that satisfies the requirements in Lemma 3 a satisfiable normal form, and use $\mathtt{snf}_{\rightsquigarrow}(\hat{s})$ to denote the set of satisfiable normal forms of $\hat{s}$. Since $\mathtt{snf}_{\rightsquigarrow}(\hat{s}) \subseteq \mathtt{nf}_{\rightsquigarrow}(\hat{s})$, the set is finite.

**Remark 2.** *Given a parametric trace $\hat{s}$, $\mathtt{snf}_{\rightsquigarrow}(\hat{s})$ is finite.*

The following theorem shows that a parametric trace has a concretization if $\mathtt{snf}_{\rightsquigarrow}(\hat{s}) \neq \emptyset$.

**Lemma 4.** *Let $\hat{s}$ be a parametric trace, and $s$ be a trace. $s$ is a concretization of $\hat{s}$ if and only if $s$ is a concretization of some $\hat{s}'$ with $\hat{s}' \in \mathtt{snf}_{\rightsquigarrow}(\hat{s})$.*

*Proof.* "⇒" If $s$ is a concretization of $\hat{s}$, then there exists a concretized ground substitution $\vartheta$ with $s = \hat{s}\vartheta$. By Lemma 2 we can get either $\hat{s}$ is a normal form or $\hat{s}$ can be deduce to a parametric trace $\hat{s}'$ by $\rightsquigarrow$ such that $s = \hat{s}'\vartheta$. If $\hat{s}$ is a normal form and it has a concretization $s$, so $\hat{s}$ is also a satisfiable normal form according to Lemma 3 . If $\hat{s}$ is not a normal form, the number of rigid messages in $\hat{s}$ is finite, so $\hat{s}\vartheta = \hat{s}'\vartheta$, where $\hat{s}'$ is a normal form, by repeatedly applying lemma 2. Since $\hat{s}'$ has the concretization $s$, $\hat{s}' \in \mathtt{snf}_{\rightsquigarrow}(\hat{s})$.

"⇐" If $s$ is a concretization of the satisfiable normal form $\hat{s}'$ such that $\hat{s}' \in \mathtt{snf}_{\rightsquigarrow}(\hat{s})$, we have $s = \hat{s}'\vartheta$ for some concretized ground substitution $\vartheta$. $\hat{s}'$ is a normal form of $\hat{s}$, so $\hat{s}' = \hat{s}\hat{\rho}$ for some $\hat{\rho}$, in which $s = \hat{s}'\vartheta = \hat{s}\hat{\rho}\vartheta$. Thus $s$ is a concretization of $\hat{s}$. □

**Theorem 3.** *A parametric trace $\hat{s}$ has a concretization if and only if* $\mathtt{snf}_{\rightsquigarrow}(\hat{s}) \neq \emptyset$.

The theorem is a corollary of Lemma 4.

## 5. Representing and checking security properties

The security properties, such as secrecy and authentication, can be defined in a concrete system, and be detected equivalently in the corresponding parametric system. In Subsection 5.1, we propose two definitions in the concrete system for secrecy and authentication properties, then characterize how to define secrecy and authentication for NSPK protocol (described in Section 2) and Abadi-Gordon protocol (described in Subsection 3.4), respectively. In Subsection 5.2, we interpret how to detect these properties in the parametric system, and prove the methods are equivalent to those defined in the corresponding concrete system.

### 5.1 Representing the security properties

#### 5.1.1 Secrecy

The secrecy property intuitively means that the environment should never learn a confidential data the principals communicate. For example, in the Abadi-Gordon protocol described in Subsection 3.4, a confidential datum is $M$, which should be guaranteed never to occur in the environment without any protection. A usual way to define the secrecy property is by proposing a guardian to the system, checking at any time whether a confidential datum is leaked, as shown in[1].

In our system, the secrecy property cannot be defined so easily, since a sender may send a message to any possible principal, if he cannot gain the information about his destination from previous messages. Thus we cannot confirm whether the message is sent to the specific receiver we represented. In order to define the secrecy property, we will use a binder instead of a name to represent a confidential datum $M$ in the Abadi-Gordon protocol, that is, $\mathtt{M}[A, B]$, which means that the datum is only shared by $A$ and $B$. With this modification, the message labeled $a2$ should be modified to $\overline{a2}(A, \{A, \mathtt{M}[A, x]\}_{\mathtt{k}[A,x]})$ in the representation of Abadi-Gordon protocol (in Subsection 3.4). The modified system is defined as $SYS^{AG'}$. Thus we define the system with a guardian as follows:

$$SYS_s^{AG} \triangleq SYS^{AG'} \| check(x).\mathtt{0}$$

To define the secrecy property of protocols, we have the following definition:

**Definition 6.** *Let $\alpha$ be an action and $s$ be a trace. We define $s \models \neg\alpha$ if for each ground substitution $\rho$ from a variable to a ground message, $\alpha\rho$ does not occur in $s$. We say that a configuration satisfies $\neg\alpha$, denoted by $\langle s, P\rangle \models \neg\alpha$, if $s' \models \neg\alpha$ for each concrete trace $s'$ that satisfies $\langle s, P\rangle \longrightarrow^* \langle s', P'\rangle$ for some $P'$.*

With the above definition, the secrecy property of the Abadi-Gordon protocol can be characterized as follows. Similarly, we can also formally characterize the secrecy property of NSPK protocol introduced in Section 2 here.

**Characterization 1.** *[Secrecy in AG protocol] Given the formal description of Abadi-Gordon protocol, it satisfies secrecy property, if*

$$\langle \epsilon, SYS_s^{AG}\rangle \models \neg check(\mathtt{M}[A, B])$$

**Characterization 2.** *[Secrecy in NSPK protocol] Given the formal description of NSPK protocol, it satisfies secrecy property, if*

$$\langle \epsilon, SYS_s^{NSPK}\rangle \models \neg check(\mathtt{N_B}[A, B])$$

#### 5.1.2 Authentication

The authentication property is another important security property that has been studied in security protocol analysis. We exploit an already existing and widely used way to specify authentication properties, called *correspondence assertion*, which was first introduced by Woo and Lam in[6]. The following method to define authentication property comes originally from [2].

**Definition 7.** *Let $\alpha$ and $\beta$ be actions, with $f_v(\alpha) \subseteq f_v(\beta)$, and let $s$ be a trace. We use $s \models \alpha \hookleftarrow \beta$ to represent that for each ground substitution $\rho$ from a variable to a ground message, if $\beta\rho$ occurs in $s$, then $\alpha\rho$ occurs in $s$ before $\beta\rho$. We say that a configuration satisfies $\alpha \hookleftarrow \beta$, denoted by $\langle s, P\rangle \models \alpha \hookleftarrow \beta$, if $s' \models \alpha \hookleftarrow \beta$ for each trace $s'$ that satisfies $\langle s, P\rangle \longrightarrow^* \langle s', P'\rangle$ for some $P'$.*

So the authentication property of Abadi-Gordon protocol is characterized formally as follows. Similarly, authentication in NSPK protocol introduced in Section 2 is also characterized formally.

**Characterization 3.** *[Authentication in AG protocol] Given the formal description of Abadi-Gordon protocol, the sender is correctly authenticated to the receiver, if*

$$\langle \epsilon, SYS^{AG}\rangle \models \overline{a2}\, x \hookleftarrow \overline{acc}\, x$$

**Characterization 4.** *[Authentication in NSPK*

*protocol] Given the formal description of NSPK protocol, the sender is correctly authenticated to the receiver, if*

$$\langle \epsilon, SYS^{NSPK} \rangle \models \overline{a3}\, x \hookleftarrow \overline{acc}\, x$$

### 5.2 Checking the security properties

In order to check security properties in a parametric system, Definition 8 and Definition 9 simulate the specifications defined in Definition 6 and Definition 7. Theorem 4 and Theorem 5 guarantee that checking these two specifications is decidable in a parametric system.

To define the secrecy property in a parametric system, a specification that simulates the one defined in Definition 6 is given as follows:

**Definition 8.** *Let $\alpha$ be an action, and $\hat{s}$ be a parametric trace which has concretizations. $\hat{s} \models \neg\alpha$, if for each concretization $s$ of $\hat{s}$, $s \models \neg\alpha$. We say that a parametric configuration satisfies $\neg\alpha$, denoted by $\langle \hat{s}, \hat{P} \rangle \models \neg\alpha$, if $\hat{s}' \models \neg\alpha$ for each parametric trace $\hat{s}'$ that satisfies $\langle \hat{s}, \hat{P} \rangle \longrightarrow^* \langle \hat{s}', \hat{P}' \rangle$ for some $\hat{P}'$.*

A parametric action $\alpha$ is *$\hat{\rho}$-unifiable* in a parametric trace $\hat{s}$ if the parametric message in $\alpha$ can be unified to the parametric message attached to the same label as $\alpha$ in $\hat{s}$, and $\hat{\rho}$ is the result of the unification.

**Lemma 5.** *$\hat{s}$ is a parametric trace and $\hat{s} \models \neg\alpha$, if and only if $\mathtt{snf}_{\rightsquigarrow}(\hat{s}\hat{\rho}) = \emptyset$ when $\alpha$ is $\hat{\rho}$-unifiable in $\hat{s}$.*

*Proof.* "$\Rightarrow$": Prove by contradictions: If $\alpha$ is $\hat{\rho}$-unifiable in $\hat{s}$, and $\mathtt{snf}_{\rightsquigarrow}(\hat{s}\hat{\rho}) \neq \emptyset$, by Theorem 3, $\hat{s}\hat{\rho}$ has concretizations. We choose an arbitrary concretization $s$ that satisfies $s = \hat{s}\hat{\rho}\vartheta$, and then $\alpha\hat{\rho}\vartheta$ occurs in $s$ and thus $\hat{s} \not\models \neg\alpha$, which contradicts the assumption.

"$\Leftarrow$": If $\alpha$ is $\hat{\rho}$-unifiable in $\hat{s}$ with $\mathtt{snf}_{\rightsquigarrow}(\hat{s}\hat{\rho}) = \emptyset$, by Theorem 3, $\hat{s}\hat{\rho}$ has no concretization. Since a concretization of $\hat{s}$ in which $\alpha$ is $\hat{\rho}$-unifiable is also a concretization of $\hat{s}\hat{\rho}$, then for each concretization $s$ of $\hat{s}$, $s \models \neg\alpha$. Thus $\hat{s} \models \neg\alpha$. □

**Theorem 4.** *Given a concrete configuration $\langle \epsilon, P \rangle$ and an action $\alpha$, let $\hat{P}$ be the abstraction of $P$, $\langle \epsilon, P \rangle \models \neg\alpha$ if and only if $\langle \epsilon, \hat{P} \rangle \models \neg\alpha$.*

This theorem is a corollary of Theorem 2 and Lemma 5.

By Theorem 4, the definition of the secrecy property for a protocol (for instance, characterized in Characterizations 1 and 2) can be equivalently defined in its corresponding parametric system, and detected by finitely searching on the satisfiable normal form set of each parametric trace in the parametric system.

To define the authentication property in a parametric system, a specification that simulates the one defined in Definition 7 is given as follows:

**Definition 9.** *Let $\alpha$ and $\beta$ be actions, with $f_v(\alpha) \subseteq f_v(\beta)$, and let $\hat{s}$ be a parametric trace that has concretizations. $\hat{s} \models \alpha \hookleftarrow \beta$, if $s \models \alpha \hookleftarrow \beta$ for each concretization $s$ of $\hat{s}$. We say that a parametric configuration satisfies $\alpha \hookleftarrow \beta$, denoted by $\langle \hat{s}, \hat{P} \rangle \models \alpha \hookleftarrow \beta$, if $\hat{s}' \models \alpha \hookleftarrow \beta$ for each trace $\hat{s}'$ that satisfies $\langle \hat{s}, \hat{P} \rangle \longrightarrow^* \langle \hat{s}', \hat{P}' \rangle$ for some $\hat{P}'$.*

**Lemma 6.** *Given a parametric trace $\hat{s}$, $\hat{s} \models \alpha \hookleftarrow \beta$ if and only if, $\alpha$ is $\hat{\rho}$-unifiable in $\hat{s}$, and for each satisfiable normal form in $\mathtt{snf}_{\rightsquigarrow}(\hat{s})$ satisfying $\hat{s}\hat{\rho}'$, $\alpha\hat{\rho}'\hat{\rho}$ occurs before $\beta\hat{\rho}'\hat{\rho}$ in $\hat{s}\hat{\rho}'$.*

*Proof.* "$\Rightarrow$": Prove by contradictions: If $\alpha$ is not $\hat{\rho}$-unifiable in $\hat{s}$, then $\hat{s} \not\models \alpha \hookleftarrow \beta$, which contradicts to our assumption. Otherwise, assume a satisfiable normal form $\hat{s}\hat{\rho}'$ in $\mathtt{snf}_{\rightsquigarrow}(\hat{s})$, and $\alpha\hat{\rho}'\hat{\rho}$ does not occur before $\beta\hat{\rho}'\hat{\rho}$ in $\hat{s}\hat{\rho}'$. Let $s'$ be a concretization of $\hat{s}\hat{\rho}'$ satisfying $s' = \hat{s}\hat{\rho}'\vartheta$. Thus $\alpha\hat{\rho}'\hat{\rho}\vartheta$ does not occur before $\beta\hat{\rho}'\hat{\rho}\vartheta$ in $s'$, that is, $s' \not\models \alpha \hookleftarrow \beta$. $s'$ is also the concretization of $\hat{s}$. Thus $\hat{s} \not\models \alpha \hookleftarrow \beta$, which contradicts the assumption.

"$\Leftarrow$" If $\alpha$ is $\hat{\rho}$-unifiable in $\hat{s}$, and for each satisfiable normal form in $\mathtt{snf}_{\rightsquigarrow}(\hat{s})$ satisfying $\hat{s}\hat{\rho}'$, $\alpha\hat{\rho}'\hat{\rho}$ occurs before $\beta\hat{\rho}'\hat{\rho}$ in $\hat{s}\hat{\rho}'$, then for each concretization satisfying $\hat{s}\hat{\rho}'\vartheta$, $\hat{s}\hat{\rho}'\vartheta \models \alpha \hookleftarrow \beta$. By Lemma 4, the concretization is also a concretization of $\hat{s}$, so $\hat{s} \models \alpha \hookleftarrow \beta$. □

**Theorem 5.** *Given a concrete configuration $\langle \epsilon, P \rangle$ and two actions $\alpha$ and $\beta$ with $f_v(\alpha) \subseteq f_v(\beta)$, let $\hat{P}$ be the abstraction of $P$, $\langle \epsilon, P \rangle \models \alpha \hookleftarrow \beta$ if and only if $\langle \epsilon, \hat{P} \rangle \models \alpha \hookleftarrow \beta$.*

This theorem is a corollary of Lemma 6.

By Theorem 5, the definition of the authentication property for a protocol (for instance, characterized in Characterizations 3 and 4) can be equivalently defined in its corresponding parametric system, and detected by finitely searching on the satisfiable normal form set of each parametric trace in the parametric system.

To distinguish a parametric trace generated by a configuration and the one that is further substituted during the deduction procedure to a normal form, we name the former parametric trace a *most general parametric trace.*

### 5.3 An example of attacks in multiple sessions

In model checking the NSPK protocol, a sender and a receiver just perform sending or receiving actions once each, and an intruder simultaneously communicates with the sender and the receiver by imitating to be the sender. Such an attack is called a *man-in-middle attack*. It can be detected during a session of a protocol. However, some intruders can use a message leaked from a previous session of a protocol, and attack the protocol during a later session. This kind of attack is called a *replay attack*. Such an attack needs to be detected in two sessions of the protocol. In this section, a version of Woo-Lam protocol[7] will be analyzed as an example of how to detect such an attack by our method. The Woo-Lam protocol is defined flow-by-flow as follows:

$$\begin{array}{lll} A \longrightarrow B: & A & (1)\\ B \longrightarrow A: & N_B & (2)\\ A \longrightarrow B: & \{N_B\}_{K_{AS}} & (3)\\ B \longrightarrow S: & B, \{A, \{N_B\}_{K_{AS}}\}_{K_{BS}} & (4)\\ S \longrightarrow B: & \{N_B\}_{K_{BS}} & (5) \end{array}$$

We will perform model checking on two sessions of the Woo-Lam protocol. $A^{(2)}$ is composed of two sessions of $A$, and assigns each session a unique set of labels in order to distinguish them. For $B^{(2)}$, without loss of generality, we assume that in the second session, $B$ is willing to communicate with any principal, rather than the specific $A$. $S^{(2)}$ is just composed of two sessions of $S$. So the two-session Woo-lam protocol is described as follows:

$$\begin{array}{rl} A^{(2)} \triangleq & \overline{a1}\, A.a2(x_a).\overline{a3}\,\{x_a\}_{\mathtt{k}[A,S]}.\mathbf{0} \| \\ & \overline{a'1}\, A.a'2(x'_a).\overline{a'3}\,\{x'_a\}_{\mathtt{k}[A,S]}.\mathbf{0} \\ B^{(2)} \triangleq & b1(x_b).[x_b = A]\,\overline{b2}\, N_B.b3(y_b). \\ & \overline{b4}\,(B, \{x_b, y_b\}_{\mathtt{k}[B,S]}).b5(z_b).case\ z_b \\ & of\ \{u_b\}_{\mathtt{k}[B,S]}\ in\ [u_b = N_B]\,\overline{acc}\, y_b.\mathbf{0} \| \\ & b'1(x'_b).\overline{b'2}\, N'_B.b3(y'_b). \\ & \overline{b4}\,(B, \{x'_b, y'_b\}_{\mathtt{k}[B,S]}).b5(z'_b).\ case\ z'_b \\ & of\ \{u'_b\}_{\mathtt{k}[B,S]}\ in\ [u'_b = N'_B]\,\mathbf{0} \\ S \triangleq & s1(x_s).let\ (x'_s, x''_s) = x_s\ in\ case\ x''_s \\ & of\ \{y_s\}_{\mathtt{k}[x'_s,S]}\ in\ let\ (z_s, w_s) = y_s\ in \\ & case\ w_s\ of\ \{u_s\}_{\mathtt{k}[z_s,S]}\ in \\ & \overline{s2}\,\{u_s\}_{\mathtt{k}[x'_s,S]}.\mathbf{0} \\ S^{(2)} \triangleq & S \| S \\ SYS^{(2)} \triangleq & A^{(2)} \| S^{(2)} \| B^{(2)} \end{array}$$

We define the authentication property as follows: if the label $\overline{acc}$ occurs in a trace attached to a message, then at least one label in $\overline{a3}$ and $\overline{a'3}$ attached to the same message occurs in the same trace before $\overline{acc}$.

**Characterization 5.** *[Authentication in two-session Woo-Lam protocol] Given the formal description of two-session Woo-Lam protocol, the sender is correctly authenticated to the receiver, if*

$$\langle \epsilon, SYS^{(2)} \rangle \models (\overline{a3}\, x \vee \overline{a'3} x) \hookleftarrow \overline{acc}\, x$$

The model and the specification can be translated into a parametric system, and counterexamples can be detected automatically in its parametric system. One of them is shown as follows:

$b1(A).b'1(\hat{x}'_b).\overline{b2}\, N_B.\overline{b'2}\, N'_B.b3(\hat{y}_b).$
$b'3(\{N_B\}_{\mathtt{k}[\hat{x}'_b,S]}).\overline{b'4}(B, \{\hat{x}'_b, \{N_B\}_{\mathtt{k}[\hat{x}'_b,S]}\}_{\mathtt{k}[B,S]}).$
$\overline{b4}\,(B, \{A, \hat{y}_b\}_{\mathtt{k}[B,S]}).s1(\hat{x}_s, \{\hat{y}_s, \{\hat{z}_s\}_{\mathtt{k}[\hat{y}_s,S]}\}_{\mathtt{k}[\hat{x}_s,S]}).$
$s1(B, \{\hat{x}'_b, \{N_B\}_{\mathtt{k}[\hat{x}'_b,S]}\}_{\mathtt{k}[B,S]}).$
$\overline{s2}\,\{\hat{z}_s\}_{\mathtt{k}[\hat{x}_s,S]}.\overline{s2}\,\{N_B\}_{\mathtt{k}[B,S]}.b5(\{N_B\}_{\mathtt{k}[B,S]}).\overline{acc}\,\hat{y}_b$

In this counterexample, there is no action labeled $ax$, which means that an intruder can completely imitate $A$. It is a bit difficult to understand the counterexample, which actually represents the following attack.

$$\begin{array}{lll} I(A) \longrightarrow B: & A & (a1)\\ B \longrightarrow I(A): & N_B & (a2)\\ I \longrightarrow B: & I & (b1)\\ B \longrightarrow I: & N'_B & (b2)\\ I(A) \longrightarrow B: & \hat{y}_b & (a3)\\ B \longrightarrow S: & B, \{A, \hat{y}_b\}_{K_{BS}} & (a4)\\ I \longrightarrow B: & \{N_B\}_{K_{IS}} & (b3)\\ B \longrightarrow S: & B, \{I, \{N_B\}_{K_{IS}}\}_{K_{BS}} & (b4)\\ S \longrightarrow B: & \{N_B\}_{K_{BS}} & (a5(b5)) \end{array}$$

The reason that the attack occurs is that $B$ cannot distinguish which session the last message belongs to. To refine the protocol, one possible solution is for the server $S$ to append the information of $B$'s communication principal in the encrypted message in the last flow[15]:

$$S \longrightarrow B: \quad \{A, N_B\}_{K_{BS}} \qquad (5')$$

However, the modified protocol has a replay attack even in a single session! A counterexample is detected as follows:

$b1(A).\overline{b2}\, N_B.b3(N_B).\overline{b4}\,(B, \{A, N_B\}_{\mathtt{k}[B,S]}).$
$b5(\{A, N_B\}_{\mathtt{k}[B,S]}).\overline{acc}\, N_B$

which can be interpreted as follows:

$$\begin{array}{lll} A \longrightarrow B: & A & (1'')\\ B \longrightarrow A: & N_B & (2'')\\ I(A) \longrightarrow B: & N_B & (3'')\\ B \longrightarrow S: & B, \{A, N_B\}_{K_{BS}} & (4'')\\ I(S) \longrightarrow B: & \{A, N_B\}_{K_{BS}} & (5'') \end{array}$$

One of the correct modifications of Woo-Lam protocol is in Message (3), $A$ sends an encrypted message whose plain message is not only $N_B$ but also $A$'s and $B$'s names. Such a modification can prevent both replay attacks

we introduce above.

$A \longrightarrow B:$ $A$
$B \longrightarrow A:$ $N_B$
$A \longrightarrow B:$ $\{A, B, N_B\}_{K_{AS}}$
$B \longrightarrow S:$ $B, \{A, \{A, B, N_B\}_{K_{AS}}\}_{K_{BS}}$
$S \longrightarrow B:$ $\{A, B, N_B\}_{K_{BS}}$

## 6. Implementation by Maude

According to Theorems 4 and 5, we can detect the specifications for security properties by generating the satisfiable normal form set of each parametric trace that a parametric system for a protocol produces, then performing model checking on each satisfiable normal form. A satisfiable normal form is deduced on-the-fly by applying deductive rules in Definition 5. Thus we propose an on-the-fly model checking method, and Maude is chosen to implement it.

Maude[8] is a language and system supporting both equational and rewriting logic computation for a wide range of applications. The basic units of Maude specifications are modules. In Core Maude, there are two kinds of modules: functional modules and system modules. Functional modules define data types and operations on them by means of equational theories whose equations are assumed to be confluent and terminating. System modules specify a model by a rewrite theory, and the model is a transition system with an initial term. For a finite system, Maude `search` command explores all possible execution paths from the initial term for reachable states satisfying some property.

A basic functional module mainly has four parts: sorts, operations, variables and equations. Maude can define a sort or several sorts each time, with the key words `sort` and `sorts` respectively. Variables are declared with the key words `var` or `vars`. The key word of operation is `op`. There are two uses of operations: as a constructor of a sort, and as a declaration of a function. `[ctor]` is a key attribute of a constructor. A function can be implemented by a set of equations, by key words `eq` and `ceq`(conditional equation). The use of variables in equations does not carry actual values. Rather, they stand for any instance of a certain sort. Anything defined in a function module can be defined in a system module in the same way. Also, a system module can define a transition system by a set of rewrite laws, whose key words are `rl` and `crl`(conditional rewrite law).

We implement each elementary definition and function in the parametric system by functional modules, and implement a *trace generating system* by using a system module. Then we use `search` command to find whether the negations of the specifications are reachable.

There are slight differences between implementing a shared-key protocol model and a public-key protocol model, since according to Definition 3, a shared-key rigid message is "context-free", while a public-key one, which is decided in a context of parametric messages, is "context-sensitive". In Maude, the functions that judge whether a message is a rigid message are declared as follows:

```
op isSharedRigid : Message -> Bool .
op isPublicRigid : Message Messagelist
                             -> Bool .
```

In this section, we introduce our implementation based mainly on a shared-key protocol system. The implementation can be naturally encoded in a public-key system, which we have also implemented. In the following subsections, we first introduce some basic types, definitions and functions used for trace system generating and on-the-fly model checking. They are defined in some functional modules. Then we introduce a trace generating system used for model checking, which is defined in a system module.

### 6.1 Types, definitions, and key functions in functional modules

In a parametric system, types and constructors of the parametric message, the parametric action (which is a 3-tuple consisting of a label, an input/output signal, and a parametric message) and the parametric trace (which is a list of actions) are defined in their function modules, named `MESSAGE`, `ACTION`, and `TRACE` respectively, as follows. These definitions coincide with the definitions in Subsection 4.1.

```
sort Message .
  op  name : Nat -> Message [ctor] .
  op   px  : Nat -> Message [ctor] .
  op (_,_) : Message Message ->
                        Message [ctor] .
  op {_}_  : Message Message ->
               Message [ctor prec 20] .
  op k[_,_] : Message Message ->
               Message [ctor prec 15] .

sort Action .
  op  <_,_,_> : Label IO Message ->
               Action [ctor prec 23 ] .

sort Trace .
```

```
  subsort Action < Trace .
  op Nil : -> Trace [ctor] .
  op _._ : Trace Trace ->
   Trace [ctor assoc id: Nil prec 25] .
```

The following are some key functions used in our system.

**Decompose**

According to Definition 4, an elementary message set can be obtained by decomposing each pair message and encrypted message whose key is also in the set. We implement a `decompose` function as follows. It accepts two message lists (the latter is used for an environment), and returns a message list. A function `elementary` is defined by applying the `decompose` function.

```
op decompose : Messagelist Messagelist
                          -> Messagelist .
  eq decompose(nil, ML2 ) = nil .
  eq decompose((MES1,MES2) # ML1,ML2)=
              decompose(MES1 # MES2 #
              ML1, MES1 # MES2 # ML2) .
  ceq decompose({MES1}k[name(N1),
              name(N2)] # ML1, ML2) =
        {MES1}k[name(N1), name(N2)] #
                  decompose (ML1, ML2)
           if not in(k[name(N1),
                     name(N2)], ML2) .
  ceq decompose({MES1}k[name(N1),
               name(N2)] # ML1, ML2) =
            decompose(MES1 # ML1,
                              MES1 # ML2)
    if in(k[name(N1), name(N2)], ML2) .
  eq decompose({MES1}k[MES2, MES3] #
                              ML1,ML2) =
          {MES1}k[MES2, MES3] #
                decompose (MES1 # ML1,
                  MES1 # ML2)[owise] .
  eq decompose(MES1 # ML1 , ML2) =
                    MES1 # decompose
                    (ML1, ML2)[owise] .
op elementary : Messagelist ->
                             Messagelist .
  eq elementary (ML1) =
                decompose (ML1, ML1) .
```

**Unification**

A unification function, `unifying`, is used for unifying a rigid message and each message in an elementary message set (defined as a list in the implementation). A unification function accepts two messages and returns a boolean and a substitution. Maude does not allow the definition of product type, so we need to define a `Result` type, which is a pair type of `Substitutions` and `Bool`. For simplicity, here we only illustrate the unification function by proposing its base cases and some inductive cases. Before defining the unification function, we need an occurrence check function, `oCheck`, which checks whether a parametric variable occurs in a parametric message.

```
sort Result .
  op (_,_) : Substitutions Bool ->
                          Result [ctor] .
  op getSubstitution : Result ->
                          Substitutions .
  op getBool : Result -> Bool .

  op oCheck  : Message Message -> Bool .
    eq oCheck (px(X), px(X)) = true .
    eq oCheck (px(X), (M1,M2)) =
               oCheck(px(X), M1) or
                     oCheck(px(X), M2) .
    eq oCheck (px(X), {M1}M2) =
              oCheck(px(X), M1) or
                     oCheck(px(X), M2) .
    eq oCheck (px(X), k[M1,M2]) =
              oCheck(px(X), M1) or
                     oCheck(px(X), M2) .
    eq oCheck (M1, M2) = false [owise] .

  op unifying : Message Message ->
                        Result [ comm ] .
    eq unifying(px(X),px(Y)) =
                   (X |-> px(Y), true) .
    eq unifying(px(X),name(Y)) =
                (X |-> name(Y), true) .
    eq unifying(name(X),name(X))=
                           (nil, true) .
    ceq unifying(px(X),(M1,M2)) =
                  (X |-> (M1,M2), true)
         if not oCheck(px(X), (M1,M2)) .
    ... ...
    eq unifying((M1,M2),(M3,M4))=
     ((getSubstitution(unifying(M1,M3)),
      getSubstitution( unifying
      (substitutions(M2,getSubstitution
      (unifying(M1,M3))),substitutions
      (M4, getSubstitution
      (unifying(M1,M3))))))),
      ( getBool(unifying(M1,M3)) and
        getBool(unifying(substitutions
        (M2, getSubstitution
         (unifying(M1,M3))),
       substitutions(M4,getSubstitution
                 (unifying(M1,M3)))))) .
    ... ...
    eq unifying(M1,M2) =
```

```
                    (nil, false) [owise] .
```

**Trace analyzing**

A trace analyzing function, `analyzingTrace`, is a core function of our system. It accepts a trace and a message list as an environment of the trace, and returns the trace's first rigid message, a boolean (true if there exists some rigid message in the trace), and a message list (as an elementary message list ready for unifying the rigid message). The basic strategy is, the function will return the first rigid message and its elementary message list. For the same reason as defined in the function `unifying`, we need to define a 3-tuple type as the return type.

```
sort Anares .
  op [_,_,_] : Message Bool Messagelist
                              -> Anares .
  op getMessage : Anares -> Message .
  op getBool    : Anares -> Bool .
  op getMessagelist : Anares ->
                             Messagelist .

  op analyzingTrace : Trace Messagelist
                              -> Anares .
    eq analyzingTrace(Nil, ML1) =
              [name(100), false, nil ] .
    ceq analyzingTrace (< LA1 , i ,
                 MES1 > . TR1 ,  ML1 ) =
      [getMesRes(getSharedRigid(MES1)),
                               true , ML1 ]
   if getBoolRes(getSharedRigid(MES1)) .
    ceq analyzingTrace(< LA1, i , MES1 >
                    . TR1 ,  ML1 ) =
                analyzingTrace (TR1, ML1)
   if not getBoolRes
                 (getSharedRigid(MES1)) .
    eq analyzingTrace(< LA1, o , MES1 >
                     . TR1 ,  ML1 )=
        analyzingTrace (TR1, MES1 # ML1) .
```

Here we do not show how to define the function `getSharedRigid`, which accepts a message and returns its first rigid message and a boolean (true if the message contains a rigid message).

### 6.2 Trace generating system in a system module

A trace generating system is embedded in a system module. In the trace generating system, there are two kinds of trace generating rules. The first kind of rules comes from the parametric transition relation (in Fig. 5). These rules are specific rules for a protocol that describe behaviors of each principal in the protocol. We name a parametric trace generated by these rules an *original trace*, which are the most general parametric traces, that are not further instantiated. We will illustrate these rules by an example, as one fragment of the protocol-specific part in the next subsection.

The second kind of rules is a common part of each protocol, which comes from the parametric trace deductive relation, $\rightsquigarrow$ (defined in Definition 5). These rules deduce a parametric trace to a new one by applying a substitution, which is the result of unifying a rigid message and an elementary message in the old parametric trace. We name a parametric trace that is generated by these rules, and that needs to be further substituted, a *pending trace*. Furthermore, we name a parametric trace which is a satisfiable normal form of some original trace a *satisfiable trace*.

We define the state of the trace generating system as a 3-tuple, $\langle tr, S, k\rangle$, where

- $tr$ is a parametric trace.
- $S$ is a list of substitutions.
- $k$ is a type of $tr$, where $k \in \{ot, st, pt\}$. $ot$ denotes an original trace, $st$ denotes a satisfiable trace and $pt$ represents a pending trace.

In Maude, the state is defined as follows:

```
sort Tracestate Tracetype State .
  ops  ot st pt : -> Tracetype [ctor] .
  op [_]  : Trace ->
                    Tracestate [frozen] .
  op <_,_,_> : Tracestate
  Substitutionlist Tracetype -> State .
```

Let's specify how the parametric deductive relation is represented in our trace generating system. If a parametric trace is labeled $pt$, and its substitution list is not empty, it will be deduced to a new parametric trace by applying the first substitution in its substitution list. At the same time, a new substitution list of the new trace will be calculated on-the-fly by applying `analyzingTrace` to the new trace. Furthermore, given a state, we also shrink its substitution list by removing the first substitution so that other substitutions can be applied.

```
crl [sub_per_pt] : < [ TR1 ] , SUBS @
                           SUBLIST, pt >
  => <[substitutingTrace (TR1, SUBS)],
        getSubstitutionlist(getMessage
     (analyzingTrace(substitutingTrace
       (TR1, SUBS), nil)),
       elementary(getMessagelist
     (analyzingTrace(substitutingTrace
       (TR1, SUBS), nil))), NIL), pt >
   if not isSatisfiableNF
```

```
     (substitutingTrace (TR1, SUBS)) .
rl [sub_dis] : < [ TR1 ] , SUBS @
                          SUBLIST, pt >
       => < [ TR1 ] , SUBLIST, ht > .
```

Furthermore, an original trace can naturally transfer to a pending trace if it is not a satisfiable normal form, and the pending trace's substitution list is obtained on-the-fly. An original trace can also transfer to a satisfiable trace if it is a satisfiable normal form (checked by `isSatisfiableNF`). A pending trace can transfer to a satisfiable trace if all rigid messages are unified, which is also checked by `isSatisfiableNF`, and thus it is a satisfiable normal form of some original trace.

```
crl [ot_to_pt] : < [ TR1 ],SUBLIST,ot >
   => < [ TR1 ], getSubstitutionlist
  (getMessage(analyzingTrace (TR1,nil)),
   elementary(getMessagelist
  (analyzingTrace(TR1,nil)), NIL), pt >
        if not isSatisfiableNF (TR1) .
crl [ot_to_st] : < [ TR1 ],SUBLIST,ot >
   => < [ TR1 ],NIL ,st >
            if  isSatisfiableNF(TR1) .
crl [pt_to_st] :  < [ TR1 ] , SUBS @
                          SUBLIST, pt >
   =>< [ substitutingTrace(TR1,SUBS) ],
                             NIL, st >
     if isSatisfiableNF
        SubstitutingTrace(TR1, SUBS)) .
```

The system starts with an initial state:

```
eq init =  < [ Nil ] , NIL , ot > .
```

### 6.3 Protocol-specific description

By our implementation, the protocol-specific part is surprisingly short, and only contains about fifty lines for each protocol. This part mainly has two fragments. One is used to describe the parametric transition relation, which is a kind of rules in the trace generating system we introduced in Section 6.2. The other fragment is used to describe the specifications, that is, security properties of each protocol.

As defined in the parametric transition relation (in Fig. 5), each action in a parametric process can be added to the tail of its parametric trace only once. Since each label occurs only once in one process, we will check whether an action has been added to its parametric trace by searching its label in the parametric trace. For example, in the Woo-Lam protocol given in Section 5, the first flow will be implemented as follows:

```
crl [A_1] : < [ TR1 ], SUBLIST, ot >
   => < [ (TR1 . < a(1), o,
            name(0) >) ], SUBLIST, ot >
     if not labelinTrace (TR1, a(1)) .
crl [B_1] : < [ TR1 ], SUBLIST, ot >
   => < [ (TR1 . < b(1), i,
            name(0) >) ], SUBLIST, ot >
     if not labelinTrace (TR1, b(1)) .
```

In the trace generating system, each satisfiable trace represents a successful run of the protocol. So we will search whether the negation of a specification is reachable in a satisfiable trace. For example, the negation of the authentication property for Woo-Lam protocol defined in Characterization 5 is represented as follows:

```
search [1] in WOOLAMPROTOCOL : init =>*
   < [ TR1 ], NIL, st > such that not
           ( labelinTrace(TR1, acc)
       implies (
        ( labelinTrace(TR1, a(3)) and
        labelbefore(TR1, a(3), acc) and
      equal((getLabelMessage(TR1,acc)),
           (getLabelMessage(TR1,a(3))))
     )
     or
     ( labelinTrace(TR1, a'(3)) and
      labelbefore (TR1,a'(3),acc) and
      equal((getLabelMessage(TR1,acc)),
          (getLabelMessage(TR1,a'(3))))
     )
    ) ) .
```

The total protocol-specific part of the two-session Woo-Lam protocol is about 40 lines. We will illustrate the whole protocol-specific part of the Yahalom protocol in Appendix A.2 as another example.

### 6.4 Experimental result

We have focused on the authentication property, and performed several tests for some security protocols described in the security protocols repository[☆]. The results are summarized in Fig. 6. In the figure, "Woo-lam protocol" is what we introduced in Subsection 5.3, (1)-(5), and "Woo-lam protocol*" is a variation described in Subsection 5.3, (1)-(4), (5′). Furthermore, the number in the column "sessions" is the number of sessions we have modeled when checking the properties. In the column "protocol spec.", the number means the line number for a protocol specific part. Besides that, each Maude file also contains about 330 lines for a common part. The number in the column

[☆] The descriptions of these protocols come from the following security protocols repository, http://www.lsv.ens-cachan.fr/spore/table.html.

"states" means the states generated by the system, and the column "times" shows how many milliseconds that checking the protocol takes; In the column "flaws", "detected" means we detected an attack, while"undetected" means in the number of sessions, we did not detect any attacks.

There are two possibilities when representing a two-session protocol. One possibility is that, each principal acts in the same role (i.e. a sender or a receiver). This case actually means that a principal initiates two sessions by communicating with an unlimited number of principals, because of the usage of binders. The second possibility is that, each principal acts in two different roles in the two sessions. This case represents that each of two different principals initiates a session by communicating with an unlimited number of principals, respectively. In both cases, one principal should intend to receive messages from the other in one of two sessions. Otherwise the two principals may have no communications with each other, and thus we could not define any security properties between them. In Fig. 6, a two-session protocol in which each principal acts in the same role is labeled by †, and in different roles is labeled by ‡.

The tests were performed on a Pentium 1.4 GHz, 1.5 G memory PC, under Windows XP. By the experimental results, we could find that a protocol that is secure in one session is not necessarily really secure. For instance, we could not detect any flaws in one-session Yahalom protocol, Otway-Ree protocol, and Woo-Lam protocol, while in the two-session of these protocols, flaws do exist. Furthermore, the checking is quite time-consuming when number of sessions increases. By our experience, checking a protocol with more than three sessions often takes several hours.

## 7. Related work

Trace analysis is one of the formal approaches in analyzing security protocols, both in the model checking and in theorem proving.

Gavin Lowe first used trace analysis on process calculus CSP, and implemented a model-checker FDR to discover numerous attacks[3),9)]. In his work, the intruder is represented as a recursive process. He restricts the state space to be finite by imposing upper-bounds upon messages the intruder generates, and also upon the principals in the network.

Many of our ideas are inspired by Michele Boreale's symbolic approach[2)]. In his research, he restricted the number of principals and intruders, and represented that each principal explicitly communicates with an intruder. Our model finitely represents an unlimited number of principals and intruders in the network. This is more powerful than his.

Trace analysis is also used in theorem proving. Paulson et al. first defined the traces and the security properties inductively, and proved whether a security protocol satisfies a property by Isabelle/HOL[10),11)]. The approach need not restrict the number of traces to be finite, however it cannot be fully automated.

Several other research papers also used process calculi: M. Abadi and A. Gordon developed the Spi calculus with primitives representing the cryptographic operations of encryption and decryption. They used some equivalences [5),12)] to define the security properties. Unfortunately, these equivalences are usually undecidable. Another approach based on process calculi was a static analysis based on type system [13)~15)]. The limitation of this method is that the intruder's model is weaker than the Dolev-Yao model, assuming that the intruder is partially trusted.

David Basin et al. proposed an On-the-fly model checking method (OFMC)[16)]. They used a high-level language, HLPSL, to represent a protocol, which then translates automatically to a low-level language, IF. An intruder's messages are instantiated when necessary, which is similar to the occasion when a rigid message occurs in our model. In their work, an intruder's role is explicitly assigned, for instance, as an initiator. This is flexible and efficient, but the process needs to be performed several times to insure that in no role can an intruder attack the protocol. In our work, we do not explicitly define an intruder, and we have to check all situations in which intruders act in different roles at one time.

The strand space formalism[17),18)] is a framework for studying security protocol analysis. There are some similarities between our parametric trace and the strand. The difference is that in Strand Space, intruder abilities are explicitly represented as some strands.

It has been shown by J. Heather et al. in[19)] that a tag system can prevent type flaw attacks. Their tag definition is similar to our type system. Their work infers that the depth of ground

| protocols | session | protocol spec. | states | times(ms) | flaws |
|---|---|---|---|---|---|
| NSPK protocol | 1 | 20 | 46 | 130 | detected |
| Woo-Lam protocol* | 1 | 25 | 168 | 160 | detected |
| fixed NSPK protocol | 1 | 20 | 164 | 637 | undetected |
| fixed NSPK protocol ‡ | 2 | 29 | 16,468 | 243,460 | undetected |
| Abadi-Gordon protocol | 1 | 20 | 238 | 713 | undetected |
| Abadi-Gordon protocol † | 2 | 30 | 4,802 | 30,499 | undetected |
| Yahalom protocol | 1 | 26 | 279 | 2,111 | undetected |
| Yahalom protocol ‡ | 2 | 36 | 536 | 1,039 | detected |
| Otway-Ree protocol | 1 | 25 | 461 | 8,185 | undetected |
| Otway-Ree protocol ‡ | 2 | 34 | 2,164 | 22,316 | detected |
| Woo-lam protocol | 1 | 25 | 552 | 2460 | undetected |
| Woo-lam protocol † | 2 | 51 | 105,423 | 476,507 | detected |

**Fig. 6** Experimental results

messages can be bounded in the search for an attack, which yields decidability by exhaustive search. Our conclusion is the same as theirs.

Recently, a more general and efficient verification approach based on Horn clauses and resolution has been proposed by B. Blanchet et al. in [20),21)]. It verifies the properties in infinite sessions of a protocol with infinite principals by some approximations on both sessions and principals. The method sometimes does not terminate, as the author noted. In[21)], a tag system that assigns each encrypted message a unique tag is added to the system to make each run terminate. Then the authors proved that security of a tagged protocol does not imply the security of an untagged version. Our model avoids recursive executions of protocols by restricting replication, and thus the model terminates.

The research of Comon-Lundh and Cortier [22)] is also based on the Horn clauses. They proved that it is sufficient to consider only a bounded number of principals when verifying some security properties. They distinguish intruders as compromised principals and eavesdroppers, reducing a system with infinite principals to one with finite principals.

## 8. Conclusion

In this paper, a finite *parametric model* is proposed by restricting/abstracting the infinite factors of security protocols. Security properties are checked automatically by on-the-fly model checking. This model checking is sound and complete under the restriction of the bounded number of sessions, and implemented on Maude. To describe security protocols, we set a typed process calculus is in which new syntax, the *binder* and the *range* are introduced. A deductive environment is used to represent intruders based on Dolev-Yao model[4)]. The calculus avoids recursive operations, so that only bounded sessions of a protocol are considered.

The idea of ranges and binders is, that a principal is assumed to communicate with any possible principal with the same message if he does not know his intended destination. For instance, a principal $A$ sending a message $M$ encrypted by a shared key is represented as $A \triangleq (\nu x : \mathcal{I})\overline{a1}\{M\}_{\mathrm{k}[A,x]}$, where $(\nu x : \mathcal{I})$ is a range that binds $x$ within the infinite set $\mathcal{I}$, and $\mathrm{k}[A,x]$ is a binder to represent arbitrary shared keys $A$ has. An approximation that $A$ sends the same message $M$ randomly to different principals is used.

The idea of parametric messages is, that each sub-expression whose type is a type variable will be marked with a *parametric variable* that will not need to be further instantiated. For example, consider a principal $A$ such that $A \triangleq a1(x).\ let\ (y,z) = x\ in\ [z = m]\ \mathbf{0}$, and any message can be instantiated to $y$ because of its type $\alpha$, which results in infinite branches. Now $y$ is marked as a parametric variable $\hat{y}$ that will not be instantiated, thus the number of branches becomes finite. With the inference rules given in Fig. 4, an original process is translated into a *parametric process* based on its type. Traces then were reduced to be finitely many parametric traces.

In a parametric system, if a parametric trace can deduce to a *satisfiable normal form*, then it has a corresponding trace, and the deduction procedure is decided on-the-fly. For this reason, we used on-the-fly model checking and implemented the parametric system using Maude. The method successfully detected the flaws of

several security protocols automatically.

Our future work will be: First, to develop a translator from an original protocol description in the process calculus to the Maude description. Second, to perform model checking on other security properties, such as nonrepudiation, fairness, anonymity. Third, to extend with pushdown model checking to cover an extension of the calculus with the recursive process.

## 9. Acknowledgement

We thank Bochao Liu and Xin Li for fruitful discussions on the research, Jianwen Xiang for his skilled Maude technique and Shin Nakajima for the information of related references. We also grateful to the anonymous referees for their helpful suggestions. This research is supported by the 21st Century COE "Verifiable and Evolvable e-Society" of JAIST, funded by Japanese Ministry of Education, Culture, Sports, Science and Technology.

## References

1) Amadio, R. M. and Prasad, S.: The Game of the Name in Cryptographic Tables, *PPDP'99*, LNCS, Vol. 1702, Springer, pp. 15–26 (1999).
2) Boreale, M.: Symbolic Trace Analysis of Cryptographic Protocols, *ICALP'01*, LNCS, Vol. 2076, Springer, pp. 667–681 (2001).
3) Lowe, G.: Breaking and Fixing the Needham-schroeder Public-key Using FDR, *TACAS'96*, LNCS, Vol. 1055, Springer, pp. 147–166 (1996).
4) Dolev, D. and Yao, A.: On the Security of Public Key Protocols, *IEEE Transactions on Information Theory*, Vol. 29, No. 2, pp. 198–208 (1983).
5) Abadi, M. and Gordon, A. D.: A Calculus for Cryptographic Protocols: The Spi Calculus, *Proceedings of the Fourth ACM Conference on Computer and Communications Security* (1997).
6) Woo, T. Y. and Lam, S. S.: A Semantic Model for Authentication Protocols, *RSP: IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 178–194 (1993).
7) Woo, T. Y. and Lam, S. S.: A Lesson on Authenticated Protocol Design, *Operating Systems Review*, Vol.28(3), No.3, pp.24–37 (1994).
8) Clavel, B., Durán, F., Eker, S., Lincoln, P., Martì-Oliet, N., Meseguer, J., and Talcott, C.: Maude Manual (Version 2.2), *http://maude.cs.uiuc.edu/maude2-manual/* (2005).
9) Lowe, G.: Some New Attacks upon Security Protocols., *CSFW'96*, IEEE Computer Society Press, pp. 162–169 (1996).
10) Bella, G.: *Inductive Verification of Cryptographic Protocols*, PhD Thesis, University of Cambridge (2000).
11) Paulson, L. C.: The Inductive Approach to Verifying Cryptographic Protocols, *Journal of Computer Security*, Vol. 6, pp. 85–128 (1998).
12) Abadi, M. and Gordon, A. D.: A Bisimulation Method for Cryptographic Protocols, *Nordic Journal of Computing*, Vol. 5, pp. 267–303 (1997).
13) Abadi, M. and Blanchet, B.: Secrecy Types for Asymmetric Communication, *FoSSaCS'01*, LNCS, Vol. 2030, Springer (2001).
14) Abadi, M.: Secrecy by Typing in Security Protocols, *Journal of the ACM*, Vol. 46, No. 5, pp. 749–786 (1999).
15) Gordon, A. and Jeffrey, A.: Authenticity by typing for security protocols, *14th IEEE Computer Security Foundations Workshop*, pp.145–159 (2001).
16) Basin, D., Mödersheim, S., Viganó, L.: OFMC: A symbolic model checker for security protocols, *International Journal of Information Security*, Vol. 4(3), pp. 181–208 (2005).
17) Guttman, J.D., Thayer, F. J.: Protocol Independence through Disjoint encryption, *13th IEEE Computer Security Foundations Workshop (CSFW-13)*, IEEE Computer Society, pp. 24–34 (2000).
18) Guttman, J.D., Thayer, F.J., Carlson, J. A., Herzon, J. C., Ramsdell,J. D., and Snifen, B.: Trust Management in Strand Spaces: a Rely-guarantee method, *European Symposium on Programming (ESOP 2004)*, LNCS, Vol. 2986, Springer, pp. 325–339 (2004).
19) Heather, J., Lowe, G., and Schneider, S.: How to Prevent Type Flaw attacks on Security Protocols, *13th IEEE Computer Security Foundations Workshop (CSFW-13)*, IEEE Computer Society, pp. 255–268 (2000).
20) Blanchet, B.: An Efficient Cryptographic Protocol Verifier Based on Prolog Rules, *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, Cape Breton, Nova Scotia, Canada, IEEE Computer Society, pp. 82–96 (2001).
21) Blanchet, B. and Podelski, A.: Verification of Cryptographic Protocols: Tagging Enforces Termination, *Theoretical Computer Science*, Vol. 333, No. 1-2, pp. 67–90 (2005). Special issue FoSSaCS'03.
22) Comon-Lundh, H. and Cortier, V.: Security Properties : Two Agents are Sufficient, *Science of Computer Programming*, Vol. 50, pp. 51–71 (2004).

## Appendix

### A.1 Type inference algorithm

$\texttt{Unify}(\tau, \tau') = (\theta, \sigma)$

(1) $\texttt{Unify}(\alpha, \tau') = (\{\tau'/\alpha\}, \tau')$, if $FTV(\tau', \alpha)$.

(2) $\texttt{Unify}(\tau, \tau) = (Id, \tau)$.

(3) let $\texttt{Unify}(\tau_1, \tau_1') = (\theta_1, \sigma_1)$,
$\texttt{Unify}(\tau_2\theta_1, \tau_2'\theta_1) = (\theta_2, \sigma_2)$
in
$\texttt{Unify}(\tau_1 * \tau_2, \tau_1' * \tau_2') = (\theta_1\theta_2, \sigma_1\theta_2 * \sigma_2)$.

(4) let $\texttt{Unify}(\tau_1, \tau_1') = (\theta, \sigma_1)$, $\sigma = \ominus^*\sigma_1$
in
$\texttt{Unify}(\ominus^*\tau_1, \ominus^*\tau_1') = (\theta, \sigma)$ $(\ominus^* \in \{\ominus_+, \ominus_-, \ominus, \ominus_?\})$.

(5) let $\texttt{Unify}(\tau_1, \tau_1') = (\theta_1, \sigma_1)$,
$\texttt{Unify}(\tau_2\theta_1, \tau_2'\theta_1) = (\theta_2, \sigma_2)$
in
$\texttt{Unify}(\tau_1 + \tau_2, \tau_1' + \tau_2') = (\theta_1\theta_2, \sigma_1\theta_2 + \sigma_2)$.

(6) let $\texttt{Unify}(\tau_1, \tau_1') = (\theta_1, \sigma_1)$,
$\texttt{Unify}(\tau_2\theta_1, \tau_2'\theta_1) = (\theta_2, \sigma_2)$
in
$\texttt{Unify}(\tau_1 \to \tau_2, \tau_1' \to \tau_2') = (\theta_1\theta_2, \sigma_1\theta_2 \to \sigma_2)$.

(7) raise error.

$\texttt{Infer}(\Gamma, e) = (\theta, \tau)$

(1) $\texttt{Infer}(\Gamma, x) = (Id, \tau)$ where $(x, \tau) \in \Gamma$.

(2) $\texttt{Infer}(\Gamma, n) = (Id, b)$ where $b = \texttt{TypeOf}(n)$.

(3) Let $\texttt{Infer}(\Gamma, \tilde{M}) = (\theta_1, \tau_1 * \ldots * \tau_n)$
in $\texttt{Infer}(\Gamma, \texttt{m}[\tilde{M}]) = (\theta_1, \sigma[\tau_1, \ldots, \tau_n])$
where $\sigma = \texttt{TypeOf}(\texttt{m})$

(4) Let $\texttt{Infer}(\Gamma, M) = (\theta_1, \tau_1)$,
$\texttt{Infer}(\Gamma\theta_1, N) = (\theta_2, \tau_2)$
in $\texttt{Infer}(\Gamma, (M, N)) = (\theta_1\theta_2, (\tau_1\theta_2) * \tau_2)$.

(5) Let $\texttt{Infer}(\Gamma, M) = (\theta_1, \tau_1)$,
$\texttt{Infer}(\Gamma\theta_1, L) = (\theta_2, \tau_2)$, $\texttt{Unify}(\tau_2, k^*) = (\vartheta, k^*)$
in $\texttt{Infer}(\Gamma, \{M\}_L) = (\theta_1\theta_2\vartheta, \ominus^*(\tau_1\theta_2))$
where $\ominus^*(\tau_1\theta_2) = \begin{cases} \ominus(\tau_1\theta_2) & k^* = k \\ \ominus_+(\tau_1\theta_2) & k^* = k_+ \\ \ominus_-(\tau_1\theta_2) & k^* = k_- \\ \ominus_?(\tau_1\theta_2) & k^* = \alpha \end{cases}$.

(6) $\texttt{Infer}(\Gamma, \mathbf{0}) = (Id, unit)$.

(7) Let $\texttt{Infer}(\Gamma \cup \{(x, \tau_1)\}, P) = (\theta, \tau_2)$ in
$\texttt{Infer}(\Gamma, (\nu x : \mathcal{A})P) = (\theta_2, \tau_2)$ where
$\tau_1 = \texttt{TypeOf}(\mathcal{A})$.

(8) Let $\texttt{Infer}(\Gamma \cup \{(x, \alpha)\}, P) = (\theta, \tau)$, in
$\texttt{Infer}(\Gamma, a(x).P) = (\theta, \alpha\theta \to \tau)$, where
$\forall\tau.(x', \tau) \in \Gamma$, $FTV(\tau, \alpha) = True$.

(9) Let $\texttt{Infer}(\Gamma, P) = (\theta, \tau)$
in $\texttt{Infer}(\Gamma, \overline{a}M.P) = (\theta, \tau)$.

(10) Let $\texttt{Infer}(\Gamma, M) = (\theta_1, \tau_1)$, $\texttt{Unify}(\alpha * \beta, \tau_1) = (\vartheta, \varrho)$,
$\texttt{Infer}(\Gamma\theta_1\vartheta \cup \{(x, \alpha), (y, \beta)\}\vartheta, P) = (\theta_2, \tau_2)$
in $\texttt{Infer}(\Gamma, let\ (x, y) = M\ in\ P) = (\theta_1\vartheta\theta_2, \tau_2)$ where $\forall\tau.(x', \tau) \in \Gamma$,
$FTV(\tau, \alpha) = FTV(\tau, \beta) = True$.

(11) Let $\texttt{Infer}(\Gamma, M) = (\theta_1, \tau_1)$,
$\texttt{Infer}(\Gamma\theta_1, L) = (\theta_2, \tau_2)$, $\texttt{Unify}(\ominus^*\alpha, \tau_1\theta_2) = (\vartheta, \varrho)$, $\texttt{Infer}(\Gamma\theta_1\theta_2\vartheta \cup \{(x, \alpha)\}\vartheta, P) = (\theta_3, \tau_3)$ in $\texttt{Infer}(\Gamma, case\ M\ of\ \{x\}_L\ in\ P) = (\theta_1\theta_2\vartheta\theta_3, \tau_3)$ where $\forall\tau.(x', \tau) \in \Gamma$, $FTV(\tau, \alpha) = True$, and
$\ominus^*\alpha = \begin{cases} \ominus(\tau_1\theta_2) & \tau_2 = k \\ \ominus_+(\tau_1\theta_2) & \tau_2 = k_- \\ \ominus_-(\tau_1\theta_2) & \tau_2 = k_+ \\ \ominus_?(\tau_1\theta_2) & \tau_2 = \beta \end{cases}$.

(12) Let $\texttt{Infer}(\Gamma, M) = (\theta_1, \tau_1)$,
$\texttt{Infer}(\Gamma\theta_1, N) = (\theta_2, \tau_2)$,
$\texttt{Unify}(\tau_1\theta_2, \tau_2) = (\vartheta, \varrho)$
$\texttt{Infer}(\Gamma\theta_1\theta_2\varrho, P) = (\theta_3, \tau_3)$
in $\texttt{Infer}(\Gamma, [M = N]P) = (\theta_1\theta_2\vartheta\theta_3, \tau_3)$.

(13) Let $\texttt{Infer}(\Gamma, P) = (\theta_1, \tau_1)$,
$\texttt{Infer}(\Gamma\theta_1, Q) = (\theta_2, \tau_2)$
in $\texttt{Infer}(\Gamma, P \| Q) = (\theta_1\theta_2, \tau_1\theta_2 + \tau_2)$.

### A.2 The protocol-specific part of the Yahalom protocol

The protocol-specific part of the Yahalom protocol in the source code mainly has two code fragments. One is to describe behaviors of each principal in Yahalom protocol, which is represented as follows:

```
crl [A_1] : < [ TR1 ], SUBLIST, ot >
      =>
 < [ (TR1 . < a(1), o, (name(0),
      name(10) ) >) ], SUBLIST, ot >
   if not labelinTrace (TR1, a(1)) .
crl [A_2] : < [ TR1 ], SUBLIST, ot >
      =>
 < [ TR1 .  < a(2), i, ((px(0),
     {(name(1), px(1)), name(10)}
    k[name(1),name(0)]), px(2)) > .
< a(3),o,(px(2), {px(0)}px(1)) > ],
                       SUBLIST, ot >
   if labelinTrace(TR1, a(1)) and
       not labelinTrace(TR1,a(2)) .
crl [A'_1] : < [ TR1 ],SUBLIST,ot >
     =>
 < [ (TR1 . < a'(1), o, (name(30),
     name(39) ) >) ], SUBLIST, ot >
   if not labelinTrace(TR1,a'(1)) .
crl [A'_2] :< [ TR1 ],SUBLIST, ot >
     =>
 < [ TR1 .  < a'(2), i, ((px(30),
 {(name(31), px(31)), name(39)}
 k[name(31),name(30)]),px(32)) > .
 < a'(3), o, (px(32),
```

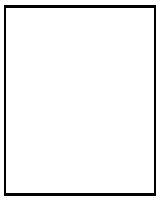
```
  {px(30)}px(31)) > ],SUBLIST,ot >
  if  labelinTrace(TR1, a'(1)) and
     not labelinTrace(TR1,a'(2)) .
 crl [B_1] :< [ TR1 ],SUBLIST,ot >
      =>
  < [ (TR1 . < b(1), i,
             (name(0), px(10)) > .
  < b(2), o, ((name(1),name(11)),
{name(0),px(10)}k[name(1),name(2)])
              > )], SUBLIST, ot >
   if not labelinTrace (TR1,b(1)) .
 crl [B_3] : < [ TR1 ],SUBLIST,ot >
      =>
 < [ (TR1 . < b(3), i,
({(name(0), px(11)), name(11)}
              k[name(1),name(2)],
            {name(11)}px(11) ) > .
     < acc, o, ({(name(0), px(11)),
        name(11)}k[name(1),name(2)],
{name(11)}px(11) ) >) ],SUBLIST,ot >
     if labelinTrace(TR1, b(1)) and
       labelinTrace(TR1, b(2)) and
     not labelinTrace (TR1, b(3)) .
 crl [B'_1]: < [ TR1 ],SUBLIST,ot >
      =>
  < [ (TR1 . < b'(1), i,
             (px(27), px(20)) > .
  < b'(2), o, ((name(1),name(21)),
{px(27), px(20)}k[name(1),name(2)])
     > )], SUBLIST, ot >
 if not labelinTrace (TR1, b'(1)) .
 crl [B'_3] : < [ TR1 ],SUBLIST,ot >
      =>
  < [ (TR1 . < b'(3), i,
     ({(px(27), px(21)), name(21)}
k[name(1),name(2)],{name(21)}px(21))
               > ) ], SUBLIST, ot >
   if labelinTrace (TR1, b'(1)) and
      labelinTrace (TR1, b'(2)) and
     not labelinTrace (TR1, b'(3)) .
 crl [S_1] : < [ TR1 ],SUBLIST,ot >=>
< [ (TR1 . < s(1),i,((px(20),px(21)),
{px(22),px(23)}k[px(20),name(2)]) > .
    < s(2), o,((px(21),{(px(20),
       k[name(0),name(1)]),px(23)}
        k[px(22),name(2)]),{(px(22),
      k[name(0),name(1)]),px(21)}
k[px(20),name(2)]) >) ],SUBLIST, ot >
    if not labelinTrace (TR1, s(1)) .
```

The other fragment is to describe the authentication specification for Yahalom protocol, which is represented as follows:

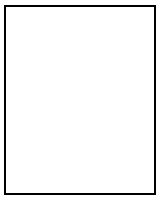
```
search [1] in YAHALOMPROTOCOL :init=>*

< [ TR1 ], NIL, st > such that not
    ( labelinTrace(TR1, acc) implies
     (
      ( (labelinTrace(TR1, a(3)) and
         labelbefore (TR1, a(3),acc))
        and
    equal((getLabelMessage(TR1,acc)),
      (getLabelMessage(TR1,a(3)))))
    )
   ) .
```

(Received December 18, 18)
(Accepted April 1, 19)



**Guoqiang Li** Guoqiang Li received his M.S. in 2002 from Shanghai Jiao Tong University, China. His research interest includes process calculus and verification methodology, such as model checking and theorem proving.



**Mizuhito Ogawa** Mizuhito Ogawa received his M.S. in 1985 and Ph.D degree in 2002, both from University of Tokyo. He worked in NTT Basic Research Laboratory from 1985 until 2001, and in JST from 2002 to 2003. From 2003, he has been working at Japan Advanced Institute of Science and Technology. His research interest includes from theory in rewriting, formal language, and combinatorics, to program verification methodology, such as theorem provers and model checkers.