Induction-Guided Falsification

Kazuhiro Ogata[1], Masahiro Nakano[1],
Weiqiang Kong[1], and Kokichi Futatsugi[1]

School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
{ogata, m-nakano, weiqiang, kokichi}@jaist.ac.jp

Abstract. The induction-guided falsification searches a bounded reachable state space of a transition system for a counterexample that the system satisfies an invariant property. If no counterexamples are found, it tries to verify that the system satisfies the property by mathematical induction on the structure of the reachable state space of the system, from which some other invariant properties may be obtained as lemmas. The verification and falsification process is repeated for each of the properties until a counterexample is found or the verification is completed. The NSPK authentication protocol is used as an example to demonstrate the induction-guided falsification.
Keywords: CafeOBJ, counterexample, induction, invariant, Maude, observational transition system (OTS)

1 Introduction

The OTS/CafeOBJ method [1] is a modeling, specification and verification method. In the method, a system is modeled as an *observational transition system*, or an *OTS*, the OTS is specified in *CafeOBJ* [2], an algebraic specification language, and it is verified that the OTS satisfies a property using the CafeOBJ system as an interactive theorem prover. OTSs are transition systems. Unlike the conventional definition of transition systems, however, the structure of states are not specified explicitly. Instead of use of variables, functions from states to data types are used to obtain the values that characterize states. Such functions are called observers. We have conducted some case studies [3–10] so as to demonstrate the effectiveness of the method and refine the method.

Although CafeOBJ does not have any model checking facilities, *Maude* [11], which is a sibling language of CafeOBJ, is equipped with such facilities. Although the state space of a system to be model checked by Maude does not have to be finite, its reachable state space should be finite. The reachable state space of an OTS is generally infinite, even if the number of some entities such as principals is made finite. Therefore, a way to search a bounded reachable state space of an OTS for a counterexample that the OTS satisfies an invariant property has been proposed [12], which is inspired by *Bounded Model Checking*, or *BMC* [13].

What if no counterexamples that an OTS satisfies an invariant property are found in the bounded reachable state space whose depth is n and the bounded

reachable state space whose depth is $n+1$ or more is too large to be exhaustively traversed within a reasonable time? If that is the case, we start verifying that the OTS satisfies the invariant property by mathematical induction on the structure of the reachable state space of the OTS. Some other invariant properties may be obtained as lemmas from the induction. If such invariant properties are obtained, we search the bounded reachable state space whose depth is n for a counterexample that the OTS satisfies each of the invariant properties. If at least one such counterexample is found, the OTS does not satisfy the original invariant property. Otherwise, the verification and falsification process called *the induction-guided falsification* is repeated for each of the invariant properties until a counterexample is found or the verification is completed.

The rest of the paper is organized as follows. Section 2 describes OTSs. Section 3 mentions how to write OTSs in CafeOBJ and Maude. Section 4 outlines how to search a bounded reachable state space of an OTS for a counterexample that the OTS satisfies an invariant property. Sections 5 and 6 describe the induction-guided falsification. Section 7 reports on a case study. The NSPK authentication protocol [14] is used as an example in Sections 3, 4 and 7. Section 8 mentions some related work. Section 9 concludes the paper.

2 Observational Transition Systems (OTSs)

We suppose that there exists a universal state space denoted Υ and that each data type used in OTSs is provided. The data types include Bool for truth values. A data type is denoted D_*.

Definition 1 (OTSs). *An OTS $\mathcal{S}$ [1] is $\langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ such that*

- *$\mathcal{O}$: A finite set of observers. Each* observer $o_{x_1:D_{o1},\ldots,x_m:D_{om}} : \Upsilon \to D_o$ *is an indexed function that has m indexes $x_1, \ldots, x_m$ whose types are $D_{o1}, \ldots, D_{om}$. The equivalence relation ($\upsilon_1 =_{\mathcal{S}} \upsilon_2$) between two states $\upsilon_1, \upsilon_2 \in \Upsilon$ is defined as $\forall o_{x_1,\ldots,x_m} : \mathcal{O}.\, (o_{x_1,\ldots,x_m}(\upsilon_1) = o_{x_1,\ldots,x_m}(\upsilon_2))$, where $\forall o_{x_1,\ldots,x_m} : \mathcal{O}$ is the abbreviation of $\forall o_{x_1,\ldots,x_m} : \mathcal{O}. \forall x_1 : D_{o1} \ldots \forall x_m : D_{om}$.*
- *$\mathcal{I}$: The set of initial states such that $\mathcal{I} \subseteq \Upsilon$.*
- *$\mathcal{T}$: A finite set of transitions. Each* transition $t_{y_1:D_{t1},\ldots,y_n:D_{tn}} : \Upsilon \to \Upsilon$ *is an indexed function that has n indexes $y_1, \ldots, y_n$ whose types are $D_{t1}, \ldots, D_{tn}$ provided that $t_{y_1,\ldots,y_n}(\upsilon_1) =_{\mathcal{S}} t_{y_1,\ldots,y_n}(\upsilon_2)$ for each $[\upsilon] \in \Upsilon/=_{\mathcal{S}}$, each $\upsilon_1, \upsilon_2 \in [\upsilon]$ and each $y_k : D_{tk}$ for $k = 1, \ldots, n$. $t_{y_1,\ldots,y_n}(\upsilon)$ is called* the successor state *of υ with respect to (wrt) $\mathcal{S}$. Each transition $t_{y_1,\ldots,y_n}$ has the condition c-$t_{y_1:D_{t1},\ldots,y_n:D_{tn}} : \Upsilon \to$ Bool, which is called* the effective condition *of the transition. If c-$t_{y_1,\ldots,y_n}(\upsilon)$ does not hold, then $t_{y_1,\ldots,y_n}(\upsilon) =_{\mathcal{S}} \upsilon$.* □

Note that although the number of indexed functions is finite, the instances of the indexed functions may be infinite. For example, the number of instances of transition $\mathrm{send1}_{p:\mathrm{Prin},q:\mathrm{Prin}} : \Upsilon \to \Upsilon$ is infinite if Prin is infinite, namely that the number of principals is infinite.

Definition 2 (Reachable states). *Given an OTS $\mathcal{S}$,* reachable states *wrt $\mathcal{S}$ are inductively defined:*

- *Each $v_{\mathrm{init}} \in \mathcal{I}$ is reachable wrt $\mathcal{S}$.*
- *For each $t_{y_1,\ldots,y_n} \in \mathcal{T}$ and each $y_k : D_{tk}$ for $k = 1\ldots,n$, $t_{x_1,\ldots,x_n}(v)$ is reachable wrt $\mathcal{S}$ if $v \in \Upsilon$ is reachable wrt $\mathcal{S}$.*

Let $\mathcal{R}_{\mathcal{S}}$ be the set of all reachable states wrt $\mathcal{S}$. $\mathcal{R}_{\mathcal{S}}$ may be called the reachable state space wrt $\mathcal{S}$. □

Predicates whose types are $\Upsilon \to \mathrm{Bool}$ are called *state predicates*. We suppose that each state predicate includes a finite number of logical connectives. We also suppose that each state predicate p considered in this paper has the form $\forall z_1 : D_{p1} \ldots \forall z_M : D_{pM}.\, P(v, z_1, \ldots, z_M)$, where $v, z_1, \ldots, z_M$ are all variables in p and $P(v, z_1, \ldots, z_M)$ does not contain any quantifiers.

Definition 3 (Invariants). *Any state predicate $p : \Upsilon \to \mathrm{Bool}$ is called* invariant *wrt $\mathcal{S}$ if p holds in all reachable states wrt $\mathcal{S}$, i.e. $\forall v : \mathcal{R}_{\mathcal{S}}.\, p(v)$.* □

Definition 4 (Execution fragments). *Given an OTS $\mathcal{S}$, execution fragments wrt $\mathcal{S}$ are inductively defined:*

- *Each $v_{\mathrm{init}} \in \mathcal{I}$ is an execution fragment (to v_{init}) wrt $\mathcal{S}$.*
- *For each $t_{y_1,\ldots,y_n} \in \mathcal{T}$ and each $y_k : D_{tk}$ for $k = 1\ldots,n$, $v_0, \ldots, v_m, t_{y_1,\ldots,y_n}(v_m)$ is also an execution fragment (to $t_{y_1,\ldots,y_n}(v_m)$) wrt $\mathcal{S}$ if $v_0, \ldots, v_m$ is an execution fragment wrt $\mathcal{S}$.*

Let $\mathcal{EF}_{\mathcal{S}}$ be the set of all execution fragments wrt $\mathcal{S}$. □

Proposition 1 (Reachable states and Execution fragments). *(1) For each reachable state $v \in \mathcal{R}_{\mathcal{S}}$, there exists an execution fragment to v wrt $\mathcal{S}$, and (2) for each execution fragment $v_0, \ldots, v_m \in \mathcal{EF}_{\mathcal{S}}$, each v_k is reachable wrt $\mathcal{S}$ for $k = 0, \ldots, m$.*

Proof. (1) By mathematical induction on v. (2) By mathematical induction on m. □

Given an execution fragment $e \in \mathcal{EF}_{\mathcal{S}}$, let $\mathrm{depth}(e)$ denote the length of the execution fragment, e.g. $\mathrm{depth}(v_0, \ldots, v_n) = n$, and let $\mathrm{ef2set}(e)$ denote the set of the states that appear in e, e.g. $\mathrm{ef2set}(v_0, \ldots, v_n) = \{v_0, \ldots, v_n\}$. Let $\mathcal{EF}_{\mathcal{S},\leq n}$ be $\{e \in \mathcal{EF}_{\mathcal{S}} \mid \mathrm{depth}(e) \leq n\}$, the set of all execution fragments wrt $\mathcal{S}$ whose lengths are less than or equal to n.

Definition 5 (Bounded reachable state space). $(\bigcup_{e \in \mathcal{EF}_{\mathcal{S},\leq n}} \mathrm{ef2set}(e))$ *is called* the (n-)bounded reachable state space *wrt $\mathcal{S}$. Let $\mathcal{R}_{\mathcal{S},\leq n}$ denote the set of states.*

From Prop. 1, it is clear that every $v \in \mathcal{R}_{\mathcal{S},\leq n}$ is reachable wrt $\mathcal{S}$. For a set $\mathcal{A} \subseteq \Upsilon$ of states to be (in)finite wrt $\mathcal{S}$ means that $\mathcal{A}/{=_{\mathcal{S}}}$ consists of (in)finite elements.

Theorem 1 (Sufficient condition that $\mathcal{R}_{\mathcal{S},\leq n}$ is finite). *If $\mathcal{I}$ is finite wrt $\mathcal{S}$ and the number of the instances of transitions whose effective conditions hold in each state of $\mathcal{R}_{\mathcal{S},\leq(n-1)}$ is finite, then $\mathcal{R}_{\mathcal{S},\leq n}$ is finite wrt $\mathcal{S}$.*

Proof. By mathematical induction on n. □

If $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$ does not hold, then there must exist a reachable state $\upsilon \in \mathcal{R}_{\mathcal{S}}$ such that $\neg p(\upsilon)$, and there must exit an execution fragment to υ wrt $\mathcal{S}$ from Prop. 1.

Definition 6 (Counterexamples). *Any execution fragment to $\upsilon \in \mathcal{R}_{\mathcal{S}}$ such that $\neg p(\upsilon)$ is called* a counterexample *for an invariant $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$. Let $\mathcal{CX}_{\mathcal{S},p}$ be all counterexamples for $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$.* □

Any counterexample $cx \in \mathcal{CX}_{\mathcal{S},p}$ such that $\neg(\exists ex' : \mathcal{CX}_{\mathcal{S},p}.\, (\mathrm{depth}(cx') < \mathrm{depth}(cx)))$ is called a *shortest counterexample* for $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$. When $\mathcal{CX}_{\mathcal{S},p}$ is not empty, let $cx^{\min}_{\mathcal{S},p} \in \mathcal{CX}_{\mathcal{S},p}$ be a shortest counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$.

3 Specifying OTSs

OTSs are defined so that they can be straightforwardly specified as behavioral specifications in CafeOBJ. But, OTSs can be specified in Maude as well [15, 12]. In this paper, the NSPK authentication protocol [14] is used as an example to describe how to specify OTSs in Maude as well as CafeOBJ.

The protocol can be described as the three message exchanges:

$$\begin{array}{l} \mathrm{Msg}\,1\ p \longrightarrow q : \mathcal{E}_q(n_p, p) \\ \mathrm{Msg}\,2\ q \longrightarrow p : \mathcal{E}_p(n_p, n_q) \\ \mathrm{Msg}\,3\ p \longrightarrow q : \mathcal{E}_q(n_q) \end{array}$$

Each principal is given a pair of keys: public and private keys. $\mathcal{E}_p(m)$ is the message m encrypted with the principal p's public key. n_p is a nonce (a random number) generated by principal p.

3.1 OTS $\mathcal{S}_{\mathrm{NSPK}}$ Modeling NSPK

One of the desired invariant properties that the protocol should have is (*Nonce*) *Secrecy Property* that any nonces cannot be leaked. The protocol is modeled as an OTS $\mathcal{S}_{\mathrm{NSPK}}$ by taking into account the intruder so as to verify that the protocol has Secrecy Property. The data types used in $\mathcal{S}_{\mathrm{NSPK}}$ are: (1) Bool for truth values, (2) Prin for principals; intr denoting the intruder, (3) Rand for random numbers; seed denoting a random number available initially; next(r) denoting a random number that has never been generated so far, (4) Nonce for nonces; n(p, q, r) denoting the nonce (generated by principal p for principal q) whose uniqueness is guaranteed by random number r, (5) Cipher for ciphertexts; enc1(p, n, q) denoting $\mathcal{E}_p(n, q)$; enc2($p, n1, n2$) denoting $\mathcal{E}_p(n1, n2)$; enc3(p, n) denoting $\mathcal{E}_p(n)$, (6) SetNonce for sets of nonces; empty denoting the empty set;

n , s denoting $\{n\} \cup s$; $s1$, $s2$ denoting $s1 \cup s2$, and (7) Network for multisets of ciphertexts; empty denoting the empty multiset; e , m denoting $\{\!|\, e \,|\!\} \uplus m$; $m1$, $m2$ denoting $m1 \uplus m2$.

$\mathcal{S}_{\mathrm{NSPK}}$ is $\langle \mathcal{O}_{\mathrm{NSPK}}, \mathcal{I}_{\mathrm{NSPK}}, \mathcal{T}_{\mathrm{NSPK}} \rangle$ such that

$$\begin{aligned}
\mathcal{O}_{\mathrm{NSLPK}} &\triangleq \{\mathrm{rand} : \Upsilon \to \mathrm{Rand}, \mathrm{nw} : \Upsilon \to \mathrm{Network}, \mathrm{nonces} : \Upsilon \to \mathrm{SetNonce}\} \\
\mathcal{I}_{\mathrm{NSLPK}} &\triangleq \{\upsilon_{\mathrm{init}} \in \Upsilon \mid \mathrm{rand}(\upsilon_{\mathrm{init}}) = \mathrm{seed} \wedge \mathrm{nw}(\upsilon_{\mathrm{init}}) = \mathrm{empty} \wedge \\
&\qquad\qquad \mathrm{nonces}(\upsilon_{\mathrm{init}}) = \mathrm{empty}\} \\
\mathcal{T}_{\mathrm{NSLPK}} &\triangleq \{\mathrm{send1}_{p:\mathrm{Prin},q:\mathrm{Prin}} : \Upsilon \to \Upsilon, \\
&\qquad \mathrm{send2}_{p:\mathrm{Prin},q:\mathrm{Prin},n:\mathrm{Nonce},nw:\mathrm{Network}} : \Upsilon \to \Upsilon, \\
&\qquad \mathrm{send3}_{p:\mathrm{Prin},q:\mathrm{Prin},n1,n2:\mathrm{Nonce},nw:\mathrm{Network}} : \Upsilon \to \Upsilon, \\
&\qquad \mathrm{fake1}_{p:\mathrm{Prin},q:\mathrm{Prin},n:\mathrm{Nonce},ns:\mathrm{SetNonce}} : \Upsilon \to \Upsilon, \\
&\qquad \mathrm{fake2}_{p:\mathrm{Prin},n1,n2:\mathrm{Nonce},ns:\mathrm{SetNonce}} : \Upsilon \to \Upsilon, \\
&\qquad \mathrm{fake3}_{p:\mathrm{Prin},n:\mathrm{Nonce},ns:\mathrm{SetNonce}} : \Upsilon \to \Upsilon\}
\end{aligned}$$

Given a state $\upsilon \in \Upsilon$, rand returns a random number available in υ, nw returns a multiset of ciphertexts (denoting the network) that have been sent up to υ, and nonces returns a set of nonces that have been gleaned by the intruder up to υ. The first three transitions model sending messages exactly following the protocol, while the last three transitions model the intruder's faking messages based on the gleaned nonces. The transitions are defined as follows:

- $\mathrm{send1}_{p,q} : \mathrm{send1}_{p,q}(\upsilon) \triangleq \upsilon'$ such that

  $\mathrm{rand}(\upsilon') \triangleq \mathrm{next}(\mathrm{rand}(\upsilon))$, $\mathrm{nw}(\upsilon') \triangleq \mathrm{enc1}(q, n(p, q, \mathrm{rand}(\upsilon)), p)$, $\mathrm{nw}(\upsilon)$, and
  $\mathrm{nonces}(\upsilon') \triangleq$ **if** $q = \mathrm{intr}$ **then** $n(p, q, \mathrm{rand}(\upsilon))$, $\mathrm{nonces}(\upsilon)$ **else** $\mathrm{nonces}(\upsilon)$.

- $\mathrm{send2}_{p,q,n,nw} : c_{\mathrm{send2}_{p,q,n,nw}}(\upsilon) \triangleq (\mathrm{nw}(\upsilon) = \mathrm{enc1}(p, n, q)$, $nw)$.
  If $c_{\mathrm{send2}_{p,q,n,nw}}(\upsilon)$, then $\mathrm{send2}_{p,q,n,nw}(\upsilon) \triangleq \upsilon'$ such that

  $\mathrm{rand}(\upsilon') \triangleq \mathrm{next}(\mathrm{rand}(\upsilon))$, $\mathrm{nw}(\upsilon') \triangleq \mathrm{enc2}(q, n, n(p, q, \mathrm{rand}(\upsilon)))$, $\mathrm{nw}(\upsilon)$, and
  $\mathrm{nonces}(\upsilon') \triangleq$ **if** $q = \mathrm{intr}$ **then** n , $n(p, q, \mathrm{rand}(\upsilon))$, $\mathrm{nonces}(\upsilon)$ **else** $\mathrm{nonces}(\upsilon)$.

- $\mathrm{send3}_{p,q,n1,n2,nw}$:
  $c_{\mathrm{send2}_{p,q,n1,n2,nw}}(\upsilon) \triangleq (\mathrm{nw}(\upsilon) = \mathrm{enc2}(p, n1, n2)$, $\mathrm{enc1}(q, n1, p)$, $nw)$.
  If $c_{\mathrm{send3}_{p,q,n1,n2,nw}}(\upsilon)$, then $\mathrm{send3}_{p,q,n1,n2,nw}(\upsilon) \triangleq \upsilon'$ such that

  $\mathrm{rand}(\upsilon') \triangleq \mathrm{rand}(\upsilon)$, $\mathrm{nw}(\upsilon') \triangleq \mathrm{enc3}(q, n2)$, $\mathrm{nw}(\upsilon)$, and
  $\mathrm{nonces}(\upsilon') \triangleq$ **if** $q = \mathrm{intr}$ **then** $n2$, $\mathrm{nonces}(\upsilon)$ **else** $\mathrm{nonces}(\upsilon)$.

- $\mathrm{fake1}_{p,q,n,ns} : c_{\mathrm{fake1}_{p,q,n,ns}}(\upsilon) \triangleq (\mathrm{nonces}(\upsilon) = n$, $ns)$.
  If $c_{\mathrm{fake1}_{p,q,n,ns}}(\upsilon)$, then $\mathrm{fake1}_{p,q,n,ns}(\upsilon) \triangleq \upsilon'$ such that

  $\mathrm{rand}(\upsilon') \triangleq \mathrm{rand}(\upsilon)$, $\mathrm{nw}(\upsilon') \triangleq \mathrm{enc1}(q, n, p)$, $\mathrm{nw}(\upsilon)$, and $\mathrm{nonces}(\upsilon)' \triangleq \mathrm{nonces}(\upsilon)$

- $\mathrm{fake2}_{p,n1,n2,ns} : c_{\mathrm{fake2}_{p,n1,n2,ns}}(\upsilon) \triangleq (\mathrm{nonces}(\upsilon) = n1$, $n2$, $ns)$.
  If $c_{\mathrm{fake2}_{p,n1,n2,ns}}(\upsilon)$, then $\mathrm{fake2}_{p,n1,n2,ns}(\upsilon) \triangleq \upsilon'$ such that

  $\mathrm{rand}(\upsilon') \triangleq \mathrm{rand}(\upsilon)$, $\mathrm{nw}(\upsilon') \triangleq \mathrm{enc2}(p, n1, n2)$, $\mathrm{nw}(\upsilon)$, and
  $\mathrm{nonces}(\upsilon)' \triangleq \mathrm{nonces}(\upsilon)$.

– $\mathrm{fake3}_{p,n,ns} : c_{\mathrm{fake3}_{p,n,ns}}(\upsilon) \triangleq (\mathrm{nonces}(\upsilon) = n\ ,\ ns)$.
If $c_{\mathrm{fake3}_{p,n,ns}}(\upsilon)$, then $\mathrm{fake3}_{p,n,ns}(\upsilon) \triangleq \upsilon'$ such that
$\mathrm{rand}(\upsilon') \triangleq \mathrm{rand}(\upsilon)$, $\mathrm{nw}(\upsilon') \triangleq \mathrm{enc3}(p,n)\ ,\ \mathrm{nw}(\upsilon)$, and $\mathrm{nonces}(\upsilon)' \triangleq \mathrm{nonces}(\upsilon)$.

Secrecy Property can be expressed as $\forall \upsilon : \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}.\mathrm{SP}(\upsilon)$, where $\mathrm{SP}(\upsilon) \triangleq \forall n : \mathrm{Nonce}\,(n \in \mathrm{nonces}(\upsilon) \Rightarrow (\mathrm{p1}(n) = \mathrm{intr} \vee \mathrm{p2}(n) = \mathrm{intr}))$, $\mathrm{p1}(\mathrm{n}(p,q,r)) \triangleq p$ and $\mathrm{p2}(\mathrm{n}(p,q,r)) \triangleq q$.

3.2 Specifying $\mathcal{S}_{\mathbf{NSPK}}$ in CafeOBJ

We suppose that there exist visible sorts `Bool`, `Prin`, `Rand`, `Nonce`, `Cipher`, `SetNonce` and `Network` corresponding to the data types used in $\mathcal{S}_{\mathrm{NSPK}}$. $\mathcal{S}_{\mathrm{NSPK}}$ is specified as a module `NSPK`. The signature of the module is as follows:

```
op  init   : -> Sys
bop rand   : Sys -> Rand
bop nw     : Sys -> Network
bop nonces : Sys -> SetNonce
bop send1  : Sys Prin Prin -> Sys
bop send2  : Sys Prin Prin Nonce Network -> Sys
bop send3  : Sys Prin Prin Nonce Nonce Network -> Sys
bop fake1  : Sys Prin Prin Nonce SetNonce -> Sys
bop fake2  : Sys Prin Nonce Nonce SetNonce -> Sys
bop fake3  : Sys Prin Nonce SetNonce -> Sys
```

`Sys` is the hidden sort denoting the state space. `bop` is the keyword to declare observation and action operators, while `op` is the keyword to declare other operators. Constant `init` denotes an arbitrary initial state of $\mathcal{S}_{\mathrm{NSPK}}$. The three observation operators correspond to the three observers, and the six action operators correspond to the six transitions. In this paper, the definition of action operator `send3` is shown, which is as follows:

```
op c-send3 : Sys Prin Prin Nonce Nonce Network -> Bool
eq c-send3(S,P1,P2,N1,N2,NW)
   = (nw(S) = enc2(P1,N1,N2), enc1(P2,N1,P1) , NW) .
eq  rand(send3(S,P1,P2,N1,N2,NW)) = rand(S) .
ceq nw(send3(S,P1,P2,N1,N2,NW))
    = (enc3(P2,N2) , nw(S)) if c-send3(S,P1,P2,N1,N2,NW) .
ceq nonces(send3(S,P1,P2,N1,N2,NW))
    = (if P2 = intr then (N2 , nonces(S)) else nonces(S) fi)
    if c-send3(S,P1,P2,N1,N2,NW) .
ceq send3(S,P1,P2,N1,N2,NW) = S if not c-send3(S,P1,P2,N1,N2,NW) .
```

`eq` is the keyword to declare equations, while `ceq` is the keyword to declare conditional equations.

Constant `init` is defined as follows:

```
eq rand(init)   = seed .
eq nw(init)     = empty .
eq nonces(init) = empty .
```

3.3 Specifying $\mathcal{S}_{\mathrm{NSPK}}$ in Maude

We suppose that there exist sorts `Bool`, `Prin`, `Rand`, `Nonce`, `Cipher`, `SetNonce` and `Network` corresponding to the data types used in $\mathcal{S}_{\mathrm{NSPK}}$. $\mathcal{S}_{\mathrm{NSPK}}$ is specified as a module `NSPK`. The signature of the module is as follows:

```
subsorts TRule OValue < Sys .
op none      : -> Sys .
op __        : Sys Sys -> Sys [assoc comm id: none] .
op rand :_   : Rand -> OValue .
op nw :_     : Network -> OValue .
op nonces :_ : SetNonce -> OValue .
op send1     : Prin Prin -> TRule .
op send2     : -> TRule .
op send3     : -> TRule .
op fake1     : Prin Prin -> TRule .
op fake2     : Prin -> TRule .
op fake3     : Prin -> TRule .
```

`Sys` is the sort denoting the state space. A state is represented by a multiset of variables (which correspond to observers) and transitions. `OValue` is the sort denoting variables and `TRule` is the sort denoting transitions. `TRule` and `OValue` are declared as subsorts of `Sys`. Constant `none` denotes the empty state, and the juxtaposition operator `__`, which is given associativity, commutativity and `none` as its identity, is the data constructor of non-empty states. The next three operators denote the three variables, which correspond to the three observers, and the last six operators denote the six transitions. In this paper, the definition of operator `send3` is shown, which is as follows:

```
rl [send3] : send3 (rand : R)
  (nw : (enc2(P1,N1,N2), enc1(P2,N1,P1) , NW)) (nonces : Ns)
  => send3 (rand : R)
     (nw : (enc3(P2,N2) , enc2(P1,N1,N2), enc1(P2,N1,P1) , NW))
     (nonces : (if P2 == intr then N2 , Ns else Ns fi)) .
```

`rl` is the keyword to declare rewriting rules, while `crl` is the keyword to declare conditional rewriting rules. `send3` is the label given to this rewriting rule.

When three principals including the intruder participate in the protocol, the initial state is represented as follows:

```
op init : -> Sys .
eq init = send1(p1,p2) send1(p1,intr) send1(p2,p1) send1(p2,intr)
    send1(intr,p1) send1(intr,p2) send2 send3 fake1(p1,p2) fake1(p1,intr)
    fake1(p2,p1) fake1(p2,intr) fake1(intr,p1) fake1(intr,p2) fake2(p1)
    fake2(p2) fake2(intr) fake3(p1) fake3(p2) fake3(intr)
    (rand : seed) (nw : empty) (nonces : empty) .
```

4 Falsification of OTSs

Maude is used to falsify $\forall o : \mathcal{R}_{\mathcal{S}}.\, p(o)$, i.e. to find a counterexample for $\forall o : \mathcal{R}_{\mathcal{S}}.\, p(o)$. The way [12] used in this paper is to search $\mathcal{R}_{\mathcal{S},\leq n}$ for a counterexample for $\forall o : \mathcal{R}_{\mathcal{S}}.\, p(o)$. If $\mathcal{R}_{\mathcal{S},\leq n}$ is finite wrt $\mathcal{S}$, this search can be completed within a finite time. A sufficient condition that $\mathcal{R}_{\mathcal{S},\leq n}$ is finite wrt $\mathcal{S}$ is given in Theorem 1. Since Maude is not equipped with any facilities that can be used to search only $\mathcal{R}_{\mathcal{S},\leq n}$ for a counterexample for $\forall o : \mathcal{R}_{\mathcal{S}}.\, p(o)$, however, we need to make a little modification to $\mathcal{S}$.

Definition 7 (Bounded OTSs). *One observer* steps : $\Upsilon \rightarrow$ Nat *is added to* $\mathcal{S}$*, where* Nat *is the type for natural numbers. The initial value returned by* steps *is* 0*, and the inequality* steps$(o) < n$ *is added to the effective condition of each transition. The value returned by* steps *is incremented whenever each transition is applied in a state where the effective condition holds. The OTS obtained by modifying* $\mathcal{S}$ *in this way is called* the (n-)bounded OTS $\mathcal{S}$ *and denoted* $\mathcal{S}^{\leq n}$. □

We have the theorem that guarantees that the search of $\mathcal{R}_{\mathcal{S}^{\leq n}}$ for a counterexample for $\forall o : \mathcal{R}_{\mathcal{S}^{\leq n}}.\, p(o)$ coincides with the search of $\mathcal{R}_{\mathcal{S},\leq n}$ for a counterexample for $\forall o : \mathcal{R}_{\mathcal{S}}.\, p(o)$ if the observer steps is not used in p.

Theorem 2 (Coincidence of counterexamples [12]). *If the observer* steps *is not used in a state predicate* $p : \Upsilon \rightarrow$ Bool*, then (1) any counterexample for* $\forall o : \mathcal{R}_{\mathcal{S}^{\leq n}}.\, p(o)$ *is also a counterexample for* $\forall o : \mathcal{R}_{\mathcal{S}}.\, p(o)$*, and (2) for any counterexample* $o_0, \ldots, o_m$ *for* $\forall o : \mathcal{R}_{\mathcal{S}}.\, p(o)$ *such that* $m \leq n$*, there exists a counterexample* $o'_0, \ldots, o'_m$ *for* $\forall o : \mathcal{R}_{\mathcal{S}^{\leq n}}.\, p(o)$ *such that* $o'_k =_{\mathcal{S}} o_k$ *for* $k = 0, \ldots, m$. □

In the Maude specification of $\mathcal{S}_{\mathrm{NSPK}}$, the following operator declaration is added:

```
op steps :_ : Nat -> OValue .
```

The term `(steps : 0)` is added to constant `init` in Subsect. 3.3. Then, the rewriting rules defining each transition is modified such that the value returned by steps is incremented whenever each transition is applied and the inequality steps$(o) < n$ is added to the condition of each of the rewriting rules. The rewriting rule labeled `send3` is modified as follows:

```
crl [send3] : send3 (rand : R)
  (nw : (enc2(P1,N1,N2), enc1(P2,N1,P1) , NW)) (nonces : Ns) (steps : X)
  => send3 (rand : R)
     (nw : (enc3(P2,N2) , enc2(P1,N1,N2), enc1(P2,N1,P1) , NW))
     (nonces : (if P2 == intr then N2 , Ns else Ns fi)) (steps : (X + 1))
  if X < bound .
```

where constant `bound` corresponds to n.

The Maude model checker can be used to search $\mathcal{R}^{\leq n}_{\mathcal{S}_{\mathrm{NSPK}}}$ for a counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}. \mathrm{SP}(\upsilon)$, and so can command `search`. In this paper, we use command `search`. One way to use command `search` is as follows:

`search` [1] *start* `=>*` *pattern* `such that` *condition* .

Command `search` performs a breadth-first search to find one state that matches *pattern* and that can be reached from *start* by applying zero or more rewriting rules.

To search $\mathcal{R}^{\leq n}_{\mathcal{S}_{\mathrm{NSPK}}}$ for a counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}. \mathrm{SP}(\upsilon)$, all we have to do is to feed the following line to the Maude system:

```
search [1] init =>* (nonces : (N , Ns)) S
                    such that not(p1(N) == intr or p2(N) == intr) .
```

When `bound` is 4, command `search` finds a state υ in which $\mathrm{SP}(\upsilon)$ does not hold. Command `show path` can be used to show the path to the state, which is a shortest counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}. \mathrm{SP}(\upsilon)$.

5 Interaction between Verification and Falsification

When `bound` is less than 4, command `search` does not find any states υ in which $\mathrm{SP}(\upsilon)$ does not hold. What if $\mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}},\leq 4}$ is too large for the Maude system to search it within a reasonable time? If so, we start verifying $\forall \upsilon : \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}. \mathrm{SP}(\upsilon)$. One standard way to prove $\forall \upsilon : \mathcal{R}_{\mathcal{S}}. p(\upsilon)$ is to use mathematical induction on υ. In the rest of the paper, let $p(\upsilon)$ be $\forall z_1 : D_{p1} \ldots \forall z_M : D_{pM}. P(\upsilon, z_1, \ldots, z_M)$.

Theorem 3 (Mathematical induction on $\mathcal{R}_{\mathcal{S}}$). *Let (I) be $\forall \upsilon_{\mathrm{init}} : \mathcal{I}. p(\upsilon_{\mathrm{init}})$, (II) be $\forall \upsilon : \mathcal{R}_{\mathcal{S}}. (p(\upsilon) \Rightarrow A. p(t_{y_1,\ldots,y_n}(\upsilon)))$, let (III) be $\forall \upsilon : \mathcal{R}_{\mathcal{S}}. B. (P(\upsilon, z_{\iota 1}, \ldots, z_M) \Rightarrow A. P(t_{y_1,\ldots,y_n}(\upsilon), z_1, \ldots, z_M))$, where A is $\forall t_{y_1,\ldots,y_n} : \mathcal{T}. \forall y_1 : D_{t1} \ldots \forall y_n : D_{tn}$ and B is $\forall z_1 : D_{p1} \ldots \forall z_M : D_{pM}$. Then, (1) $\forall \upsilon : \mathcal{R}_{\mathcal{S}}. p(\upsilon) \Leftrightarrow ((I) \wedge (II))$ and (2) $((I) \wedge (II)) \Leftrightarrow ((I) \wedge (III))$.*

Proof. (1) From the mathematical induction principle. (2) $\Leftarrow$: Straightforward. $\Rightarrow$: It is clear that $((I) \wedge (II)) \Rightarrow (I)$. We assume $(I) \wedge (II)$. From (1), we have $\forall \upsilon : \mathcal{R}_{\mathcal{S}}. p(\upsilon)$, which implies (III). □

We use $\forall \upsilon : \mathcal{R}_{\mathcal{S}}. p(\upsilon) \Leftrightarrow ((I) \wedge (III))$ from Theorem 3 in order to prove (and disprove) $\forall \upsilon : \mathcal{R}_{\mathcal{S}}. p(\upsilon)$. We often need lemmas to prove $\forall \upsilon : \mathcal{R}_{\mathcal{S}}. p(\upsilon)$.

Definition 8 (Effective case splits and Necessary lemmas). *Let us consider proving $\forall \upsilon : \mathcal{R}_{\mathcal{S}}. p(\upsilon)$ by mathematical induction on υ. In an induction case where $t_{y_1,\ldots,y_n} \in \mathcal{T}$ is taken into account, all we have to do is to prove $P(\upsilon^{\mathrm{c}}, z^{\mathrm{c}}_1, \ldots, z^{\mathrm{c}}_M) \Rightarrow P(t_{y^{\mathrm{c}}_1,\ldots,y^{\mathrm{c}}_n}(\upsilon^{\mathrm{c}}), z^{\mathrm{c}}_1, \ldots, z^{\mathrm{c}}_M)$, where υ^{c} is a constant denoting an arbitrary state and each y^{c}_k (z^{c}_k) is a constant denoting an arbitrary value of D_{tk} (D_{pk}). We suppose that a proposition $q_1 \vee \ldots \vee q_L$ is a tautology, where each q_l is in the form $Q_l(\upsilon^{\mathrm{c}}, y^{\mathrm{c}}_1, \ldots, y^{\mathrm{c}}_n, z^{\mathrm{c}}_1, \ldots, z^{\mathrm{c}}_M)$. If the truth value*

of $P(\upsilon^{\mathrm{c}}, z_1^{\mathrm{c}}, \ldots, z_M^{\mathrm{c}}) \Rightarrow P(t_{y_1^{\mathrm{c}},\ldots,y_n^{\mathrm{c}}}(\upsilon^{\mathrm{c}}), z_1^{\mathrm{c}}, \ldots, z_M^{\mathrm{c}})$ *can be determined assuming each* q_l*, then* $q_1 \vee \ldots \vee q_L$ *is called* an effective case split *for this induction case. Moreover, if the truth value is* false*, then* $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\forall y_1 : D_{t1}, \ldots, \forall y_n : D_{tn}, \forall z_1 : D_{p1}, \ldots, \forall z_M : D_{pM}.\, \neg Q_l(\upsilon, y_1, \ldots, y_n, z_1, \ldots, z_M)$ *is called* a necessary lemma *of* $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$*. Given an effective case split for each induction case, let* $\mathcal{NL}_{\mathcal{S},p}$ *be the set of all necessary lemmas of* $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$ *obtained by the effective case splits. Generally, there are multiple such sets, which depend on effective case splits.* □

In the rest of this section, let $q(\upsilon)$ be $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\forall y_1 : D_{t1}, \ldots, \forall y_n : D_{tn}, \forall z_1 : D_{p1}, \ldots, \forall z_M : D_{pM}.\, \neg Q_l(\upsilon, y_1, \ldots, y_n, z_1, \ldots, z_M)$, and let q_l be $Q_l(\upsilon^{\mathrm{c}}, y_1^{\mathrm{c}}, \ldots, y_n^{\mathrm{c}}, z_1^{\mathrm{c}}, \ldots, z_M^{\mathrm{c}})$.

Lemma 1 (Counterexamples induced by necessary lemmas). *Let* $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, q(\upsilon)$ *be a necessary lemma of* $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$*. If there exists a counterexample* $ce_q \in \mathcal{CX}_{\mathcal{S},q}$ *such that* $\mathrm{depth}(ce_q) = N$*, then* $ce_q \in \mathcal{CX}_{\mathcal{S},p}$ *or there exists a counterexample* $ce_p \in \mathcal{CX}_{\mathcal{S},p}$ *such that* $\mathrm{depth}(ce_p) = N + 1$*.*

Proof. We suppose that $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, q(\upsilon)$ is found in an induction case where a transition $t_{y_1,\ldots,y_n} \in \mathcal{T}$ is taken into account. Let ce_q be $\upsilon_0, \ldots, \upsilon_N$. From the assumption, there exist $y_1^{\mathrm{d}}, \ldots, y_n^{\mathrm{d}}, z_1^{\mathrm{d}}, \ldots, z_M^{\mathrm{d}}$ such that $Q(\upsilon_N, y_1^{\mathrm{d}}, \ldots, y_n^{\mathrm{d}}, z_1^{\mathrm{d}}, \ldots, z_M^{\mathrm{d}})$ holds. (1) $\neg p(\upsilon_N)$: Clearly $ce_q \in \mathcal{CX}_{\mathcal{S},p}$. (2) $p(\upsilon_N)$: Since both $p(\upsilon_N)$ and $Q(\upsilon_N, d_{j_1}, \ldots, d_{j_n}, d_{\iota_1}, \ldots, d_{\iota_\alpha})$ holds, $P(t_{d_{j_1},\ldots,d_{j_n}}(\upsilon_N), d_{\iota_1}, \ldots, d_{\iota_\alpha})$ must not hold because $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, q(\upsilon)$ is a necessary lemma of $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$ and is found in the induction case concerned. Therefore, $\upsilon_0, \ldots, \upsilon_N, t_{y_1^{\mathrm{d}},\ldots,y_n^{\mathrm{d}}}(\upsilon_N)$ is a counterexample of $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$. □

Lemma 2 (Existence of necessary lemmas that induce counterexamples). *If* $\mathcal{CX}_{\mathcal{S},p}$ *is not empty and* $\mathrm{depth}(cx_{\mathcal{S},p}^{\min}) = N + 1$*, then there exists a necessary lemma* $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, q(\upsilon)$ *of* $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$ *such that* $\mathcal{CX}_{\mathcal{S},q}$ *is not empty and* $\mathrm{depth}(cx_{\mathcal{S},q}^{\min}) = N$*, and such a necessary lemma can be found in any* $\mathcal{NL}_{\mathcal{S},p}$*.*

Proof. Let $cx_{\mathcal{S},p}^{\min}$ be $\upsilon_0, \ldots, \upsilon_N, \upsilon_{N+1}$. From the assumption, $p(\upsilon_N)$ holds and there exist $t_{y_1,\ldots,y_n} \in \mathcal{T}$ and $y_1^{\mathrm{d}}, \ldots, y_n^{\mathrm{d}}, z_1^{\mathrm{d}}, \ldots, z_M^{\mathrm{d}}$ such that $\upsilon_{N+1} =_{\mathcal{S}} t_{y_1^{\mathrm{d}},\ldots,y_n^{\mathrm{d}}}(\upsilon_N)$ and $\neg P(t_{y_1^{\mathrm{d}},\ldots,y_n^{\mathrm{d}}}(\upsilon_N), z_1^{\mathrm{d}}, \ldots, z_M^{\mathrm{d}})$. Let $q_1 \vee \ldots \vee q_L$ be an arbitrary effective case split for the induction case where $t_{y_1,\ldots,y_n}$ is taken into account. There must exist $l \in \{1, \ldots, L\}$ such that the truth value of $P(\upsilon^{\mathrm{c}}, z_1^{\mathrm{c}}, \ldots, z_M^{\mathrm{c}}) \Rightarrow P(t_{y_1^{\mathrm{c}},\ldots,y_n^{\mathrm{c}}}(\upsilon^{\mathrm{c}}), z_1^{\mathrm{c}}, \ldots, z_M^{\mathrm{c}})$ is false assuming q_l because otherwise there does not exist the supposed counterexample. Therefore, $Q_l(\upsilon^{\mathrm{c}}, y_1^{\mathrm{c}}, \ldots, y_n^{\mathrm{c}}, z_1^{\mathrm{c}}, \ldots, z_M^{\mathrm{c}})$ holds and then $\upsilon_1, \ldots, \upsilon_N$ is a counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, q(\upsilon)$. We suppose that $\mathrm{depth}(cx_{\mathcal{S},q}^{\min}) < N$. If so, $\mathrm{depth}(cx_{\mathcal{S},p}^{\min}) < N + 1$ from Lemma 1, which contradicts the assumption. □

We give a procedure with which we alternately falsify and verify $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$.

Definition 9 (Procedure IGF). *Given an OTS* $\mathcal{S}$*, a state predicate* p *and a natural number* n*, the procedure is defined as follows:*

1. $\mathcal{P} := \{p\}$ *and* $\mathcal{Q} := \emptyset$.
2. *Repeat the following until* $\mathcal{P} = \emptyset$.
   (a) *Choose a state predicate* q *from* $\mathcal{P}$ *and* $\mathcal{P} := (\mathcal{P} - \{q\})$.
   (b) *Search* $\mathcal{R}_{\mathcal{S}^{\leq n}}$ *for a counterexample for* $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, q(\upsilon)$.
   If a counterexample is found, terminate and return Falsified.
   (c) *Try to prove* $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, q(\upsilon)$ *by mathematical induction on* υ *and compute* $\mathcal{NL}_{\mathcal{S},q}$.
   (d) $\mathcal{Q} := \mathcal{Q} \cup \{q\}$ *and* $\mathcal{P} := \mathcal{P} \cup (\mathcal{NL}_{\mathcal{S},q} - \mathcal{Q})$.
3. *Terminate and return* Verified. □

We have the the soundness and completeness theorem on procedure IGF.

Theorem 4 (Soundness and Completeness of IGF wrt Falsification). *Given an arbitrary OTS* $\mathcal{S}$*, an arbitrary state predicate* p *and an arbitrary natural number* n*, (1) if* IGF *terminates and returns* Falsified*, then* $\neg(\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon))$*, and (2) if* $\mathcal{CX}_{\mathcal{S},p}$ *is not empty,* $\mathrm{depth}(cx^{\min}_{\mathcal{S},p})$ *is finite,* $\mathcal{R}_{\mathcal{S}^{\leq n}}$ *is finite wrt* $\mathcal{S}$ *and* $\mathcal{NL}_{\mathcal{S},q}$ *can be computed for an arbitrary state predicate* q*, then* IGF *terminates and returns* Falsified.

Proof. From Lemmas 1 and 2. □

Note that when n is large, the search of $\mathcal{R}_{\mathcal{S}^{\leq n}}$ may not be completed within a reasonable time, which implies that IGF may not terminate within a reasonable time, and when $\mathrm{depth}(cx^{\min}_{\mathcal{S},p})$ is large, IGF may not terminate within a reasonable time.

The following should be noted. The number of some entities such as principals may have to be made finite so as to make the n-bounded reachable state space wrt an OTS finite. Even when there exists a counterexample for an invariant in the n-bounded reachable state space wrt an OTS $\mathcal{S}$ in which there are an infinite number of some entities, no such counterexamples may be found in the n-bounded reachable state space wrt $\mathcal{S}$ in which there are a finite number of the entities, which depends on the number of the entities. Let us consider $\mathcal{S}_{\mathrm{NSPK}}$ for example. When the number of principals is infinite, $\mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}},\leq n}$ is also infinite if $n \geq 2$. The number of principals should be made finite to make $\mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}},\leq n}$ finite. When there are three or more principals, one of which is the intruder, a counterexample that $\mathcal{S}_{\mathrm{NSPK}}$ satisfies Secrecy Property is found in $\mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}},\leq n}$ if $n \geq 4$. Otherwise, however, no such counterexamples are found in $\mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}},\leq n}$ for any n.

6 A Way to Compute Necessary Lemmas

Since Theorem 4 relies on whether $\mathcal{NL}_{\mathcal{S},p}$ can be computed for an arbitrary state predicate p, we need to argue the feasibility. Given an arbitrary OTS $\mathcal{S}$ and an arbitrary state predicate p, we show a way to compute an effective case split for each induction case when we prove $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$ by mathematical induction on υ and to obtain $\mathcal{NL}_{\mathcal{S},p}$ based on the effective case splits. The

solution employs the CafeOBJ system that uses the Hsiang TRS [16] as a decision procedure of propositional logic. The CafeOBJ system reduces a proposition that is always true (false) to `true` (`false`). Generally, the CafeOBJ system reduces a proposition to an exclusive-or normal form.

We suppose that $\mathcal{S}$ is written as a module $\mathtt{M}_{\mathcal{S}}$ in CafeOBJ. We also suppose that when all equations available in $\mathtt{M}_{\mathcal{S}}$ are regarded as a set of left-to-right rewrite rules, the set, i.e. the TRS, is confluent and terminating. The TRS will be referred as $\mathrm{TRS}_{\mathcal{S}}$. In a module `INV`, which imports $\mathtt{M}_{\mathcal{S}}$, we declare the following operator and equation:

$\mathtt{op\ inv}_p : \mathtt{H\ V}_{p1} \ \ldots\ \mathtt{V}_{pM}\ \mathtt{->\ Bool}$
$\mathtt{eq\ inv}_p(\mathtt{S}, \mathtt{Z}_1, \ldots, \mathtt{Z}_M) = \mathtt{P}(\mathtt{S}, \mathtt{Z}_1, \ldots, \mathtt{Z}_M)\ .$

where `H` is a hidden sort denoting Υ, `S` is a CafeOBJ variable of sort `H`, each $\mathtt{Z}_k$ is a CafeOBJ variable of sort $\mathtt{V}_{pk}$, and $\mathtt{P}(\mathtt{S}, \mathtt{Z}_1, \ldots, \mathtt{Z}_M)$ is a term denoting $P(\upsilon, z_1, \ldots, z_M)$. In `INV`, for each $\mathtt{V}_{pk}$, we also declare a constant $\mathtt{y}^{\mathrm{c}}_k$ of the sort, which denotes an arbitrary value of the sort. In a module `ISTEP`, which imports `INV`, we declare the following operator and equation:

$\mathtt{op\ istep}_p : \mathtt{V}_{p1} \ \ldots\ \mathtt{V}_{pM}\ \mathtt{->\ Bool}$
$\mathtt{eq\ istep}_p(\mathtt{Z}_1, \ldots, \mathtt{Z}_M) = \mathtt{inv}_p(\mathtt{s}, \mathtt{Z}_1, \ldots, \mathtt{Z}_M)\ \mathtt{implies}\ \mathtt{inv}_p(\mathtt{s'}, \mathtt{Z}_1, \ldots, \mathtt{Z}_M)\ .$

where `s` and `s'` are constants of sort `H` declared in the module, and the operator `_implies_` corresponds to the logical implication. Constant `s` denotes an arbitrary state, and constant `s'` denotes a successor state of the state.

Let us consider an induction case in which a transition $t_{y_1,\ldots,y_n} \in \mathcal{T}$ is taken into account. We suppose that the transition and its effective condition are represented by the action operator `t` and the operator `c-t`, respectively, declared in $\mathtt{M}_{\mathcal{S}}$ as follows:

$\mathtt{bop\ t} : \mathtt{H\ V}_{t1} \ \ldots\ \mathtt{V}_{tn}\ \mathtt{->\ H}$
$\mathtt{op\ c\text{-}t} : \mathtt{H\ V}_{t1} \ \ldots\ \mathtt{V}_{tn}\ \mathtt{->\ Bool}$

We also have the following equation:

$\mathtt{eq\ s'} = \mathtt{t}(\mathtt{s}, \mathtt{y}^{\mathrm{c}}_{j_1}, \ldots, \mathtt{y}^{\mathrm{c}}_{j_n})\ .$

where each $\mathtt{y}^{\mathrm{c}}_k$ is a constant of $\mathtt{V}_{tk}$ denoting an arbitrary value of $\mathtt{V}_{tk}$.

We give a procedure that computes an effective case split for the induction case.

Definition 10 (Procedure CaseSplit). *The procedure is defined as follows:*

1. $\mathcal{C} := \{\mathtt{c\text{-}t}(\mathtt{s}, \mathtt{y}^{\mathrm{c}}_{j_1}, \ldots, \mathtt{y}^{\mathrm{c}}_{j_n}), \neg\mathtt{c\text{-}t}(\mathtt{s}, \mathtt{y}^{\mathrm{c}}_{j_1}, \ldots, \mathtt{y}^{\mathrm{c}}_{j_n})\}$ *and* $\mathcal{C}' := \emptyset$.
2. *Repeat the following until* $\mathcal{C} = \emptyset$.
   (a) *Choose a proposition* q *from* $\mathcal{C}$ *and* $\mathcal{C} := \mathcal{C} - \{q\}$.
   (b) *Reduce* $\mathtt{istep}_p(\mathtt{z}^{\mathrm{c}}_1, \ldots, \mathtt{y}^{\mathrm{c}}_M)$ *assuming* q *in module* `ISTEP`. *Let* r *be the result.*
      – *If* r *is* `true`, *do nothing.*

– *If* r *is* `false`, $\mathcal{C}' := \mathcal{C}' \cup \{q\}$.
– *Otherwise, choose a primitive proposition* ρ *from* r *and* $\mathcal{C} := \mathcal{C} \cup \{q \wedge \rho, q \wedge \neg\rho\}$.

3. *Terminate and return* $\mathcal{C}'$. □

When $\mathtt{istep}_p(\mathtt{z}_1^{\mathrm{c}}, \ldots, \mathtt{y}_M^{\mathrm{c}})$ is reduced assuming q in module `ISTEP`, q should be written as one or more equations. A way to write q in equations is described in [17]. Since $\mathrm{TRS}_{\mathcal{S}}$ is terminating and p includes a finite number of logical connectives, procedure CaseSplit terminates. CaseSplit clearly computes an effective case split for the induction case, and when CaseSplit terminates, $\mathcal{C}'$ consists of all the propositions in the effective case split such that $\mathtt{istep}_p(\mathtt{z}_1^{\mathrm{c}}, \ldots, \mathtt{y}_M^{\mathrm{c}})$ reduces to `false` assuming each of the propositions. From $\mathcal{C}'$, it is straightforward to construct all necessary lemmas (of $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$) that are found in the induction case.

7 A Case Study

We try to prove $\forall \upsilon \in \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}.\mathrm{SP}(\upsilon)$ by mathematical induction on υ based on the CafeOBJ specification of $\mathcal{S}_{\mathrm{NSPK}}$. We first declare a module `INV`, which imports module `NSPK`. In module `INV`, the following operator and equation are declared:

```
op inv1 : Sys Nonce -> Bool
eq inv1(S,N)
   = ((N \in nonces(S)) implies (p1(N) = intr or p2(N) = intr)) .
```

where the operator `_or_` corresponds to the logical disjunction. We also declare a constant `n` of sort `Nonce` in module `INV`. We next declare a module `ISTEP`, which imports module `INV`. In module `ISTEP`, the following operator and equation are declared:

```
op istep1 : Nonce -> Bool
eq istep1(N) = inv1(s,N) implies inv1(s',N) .
```

where `s` and `s'` are constants of sort `Sys` declared in module `ISTEP`.

We have the two cases in which `istep1(n)` reduces to `false`. The corresponding proof passages (basic fragments of a proof, or a proof score) are as follows:

```
open ISTEP
-- arbitrary values
  ops p1 p2 : -> Prin .  op m : -> Nonce .  op nw : -> Network .
-- assumptions
  -- eq c-send2(s,p1,p2,m,nw) = true .
  eq nw(s) = enc1(p1,m,p2) , nw .
  --
  eq p2 = intr .  eq (p1(n) = intr) = false .
  eq (p2(n) = intr) = false .  eq m = n .  eq n \in nonces(s) = false .
```

```
-- successor state
  eq s' = send2(s,p1,p2,m,nw) .
-- check
  red istep1(n) .
close
open ISTEP
-- arbitrary values
  ops p1 p2 : -> Prin .  ops m1 m2 : -> Nonce .  op nw : -> Network .
-- assumptions
  -- eq c-send3(s,p1,p2,m1,m2,nw) = true  .
  eq nw(s) = enc2(p1,m1,m2), enc1(p2,m1,p1) , nw .
  --
  eq p2 = intr .  eq m2 = n .  eq (p1(n) = intr) = false .
  eq (p2(n) = intr) = false .  eq n \in nonces(s) = false .
-- successor state
  eq s' = send3(s,p1,p2,m1,m2,nw) .
-- check
  red istep1(n) .
close
```

The CafeOBJ command `open` constructs a temporary module that imports a given module and the CafeOBJ command `close` destroys such a temporary module. A comment starts with `--` and terminates at the end of the line.

From the two proof passages, we obtain the two necessary lemmas of $\forall o \in \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}.\,\mathrm{SP}(o)$. The two necessary lemmas are $\forall o \in \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}.\,\mathrm{NL}_1(o)$ and $\forall o \in \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}.\,\mathrm{NL}_2(o)$, where $\mathrm{NL}_1(o) \triangleq \forall n : \mathrm{Nonce}.\forall q : \mathrm{Prin}.\,(\mathrm{enc1}(q,n,\mathrm{intr}) \in \mathrm{nw}(o) \Rightarrow (n \in \mathrm{nonces}(o) \vee \mathrm{p1}(n) = \mathrm{intr} \vee \mathrm{p2}(n) = \mathrm{intr}))$ and $\mathrm{NL}_2(o) \triangleq \forall n1, n2 : \mathrm{Nonce}.\forall q : \mathrm{Prin}.\,((\mathrm{enc2}(q,n1,n2) \in nw(o) \wedge \mathrm{enc1}(\mathrm{intr},n1,q) \in \mathrm{nw}(o)) \Rightarrow (n2 \in \mathrm{nonces}(o) \vee \mathrm{p1}(n2) = \mathrm{intr} \vee \mathrm{p2}(n2) = \mathrm{intr}))$.

To search $\mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}},\leq n}$ for a counterexample for $\forall o : \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}.\,\mathrm{NL}_1(o)$, all we have to do is to feed the following line to the Maude system:

```
search [1] init =>* (nw : (enc1(Q,N,intr) , Ms)) (nonces : Ns) S
             such that not(N \in Ns or p1(N) == intr or p2(N) == intr) .
```

Command `search` does not find any states o such that $\mathrm{NL}_1(o)$ does not hold when `bound` is up to 5. Actually, we have proved $\forall o \in \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}.\,\mathrm{NL}_1(o)$ by mathematical induction on $\mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}$ without any lemmas.

To search $\mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}},\leq n}$ for a counterexample for $\forall o : \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}.\,\mathrm{NL}_2(o)$, all we have to do is to feed the following line to the Maude system:

```
search [1] init =>*
    (nw : (enc2(Q1,N1,N2) , enc1(intr,N1,Q1) , Ms)) (nonces : Ns) S
        such that not(N2 \in Ns or p1(N2) == intr or p2(N2) == intr) .
```

When `bound` is 3, command `search` finds a state o in which $\mathrm{NL}_2(o)$ does not hold. Command `show path` can be used to show the path to the state, which is a shortest counterexample for $\forall o : \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}.\,\mathrm{NL}_2(o)$. An excerpt from the counterexample generated is shown in Fig. 1.

```
state 0, Sys: send2 send3 rand : seed nw : empty nonces : empty fake2(intr)
    fake2(p1) fake2(p2) fake3(intr) fake3(p1) fake3(p2) steps : 0 send1(intr,
    p1) send1(intr, p2) send1(p1, intr) send1(p1, p2) send1(p2, intr) send1(p2,
    p1) fake1(intr, p1) fake1(intr, p2) fake1(p1, intr) fake1(p1, p2) fake1(p2,
    intr) fake1(p2, p1)
===[ ... [label send1] ... ]===>
state 3, Sys: send2 send3 rand : next(seed) nw : enc1(intr, n(p1, intr, seed),
    p1) nonces : n(p1, intr, seed) fake2(intr) fake2(p1) fake2(p2) fake3(intr)
    fake3(p1) fake3(p2) steps : 1 send1(intr, p1) send1(intr, p2) send1(p1,
    intr) send1(p1, p2) send1(p2, intr) send1(p2, p1) fake1(intr, p1) fake1(
    intr, p2) fake1(p1, intr) fake1(p1, p2) fake1(p2, intr) fake1(p2, p1)
===[ ... [label fake1] ... ]===>
state 31, Sys: send2 send3 rand : next(seed) nw : (enc1(intr, n(p1, intr,
    seed), p1),enc1(p2, n(p1, intr, seed), p1)) nonces : n(p1, intr, seed)
    fake2(intr) fake2(p1) fake2(p2) fake3(intr) fake3(p1) fake3(p2) steps : 2
    send1(intr, p1) send1(intr, p2) send1(p1, intr) send1(p1, p2) send1(p2,
    intr) send1(p2, p1) fake1(intr, p1) fake1(intr, p2) fake1(p1, intr) fake1(
    p1, p2) fake1(p2, intr) fake1(p2, p1)
===[ ... [label send2] ... ]===>
state 436, Sys: send2 send3 rand : next(next(seed)) nw : (enc1(intr, n(p1,
    intr, seed), p1),enc1(p2, n(p1, intr, seed), p1),enc2(p1, n(p1, intr,
    seed), n(p2, p1, next(seed)))) nonces : n(p1, intr, seed) fake2(intr)
    fake2(p1) fake2(p2) fake3(intr) fake3(p1) fake3(p2) steps : 3 send1(intr,
    p1) send1(intr, p2) send1(p1, intr) send1(p1, p2) send1(p2, intr) send1(p2,
    p1) fake1(intr, p1) fake1(intr, p2) fake1(p1, intr) fake1(p1, p2) fake1(p2,
    intr) fake1(p2, p1)
```

Fig. 1. An excerpt from the counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}.\,\mathrm{NL}_2(\upsilon)$.

```
state 0, Sys: send2 send3 rand : next(next(seed)) nw : (enc1(intr, n(p1, intr,
    seed), p1),enc1(p2, n(p1, intr, seed), p1),enc2(p1, n(p1, intr, seed), n(
    p2, p1, next(seed)))) nonces : n(p1, intr, seed) fake2(intr) fake2(p1)
    fake2(p2) fake3(intr) fake3(p1) fake3(p2) steps : 3 send1(intr, p1) send1(
    intr, p2) send1(p1, intr) send1(p1, p2) send1(p2, intr) send1(p2, p1)
    fake1(intr, p1) fake1(intr, p2) fake1(p1, intr) fake1(p1, p2) fake1(p2,
    intr) fake1(p2, p1)
===[ ... [label send3] ... ]===>
state 9, Sys: send2 send3 rand : next(next(seed)) nw : (enc3(intr, n(p2, p1,
    next(seed))),enc1(intr, n(p1, intr, seed), p1),enc1(p2, n(p1, intr, seed),
    p1),enc2(p1, n(p1, intr, seed), n(p2, p1, next(seed)))) nonces : (n(p1,
    intr, seed),n(p2, p1, next(seed))) fake2(intr) fake2(p1) fake2(p2) fake3(
    intr) fake3(p1) fake3(p2) steps : 4 send1(intr, p1) send1(intr, p2) send1(
    p1, intr) send1(p1, p2) send1(p2, intr) send1(p2, p1) fake1(intr, p1)
    fake1(intr, p2) fake1(p1, intr) fake1(p1, p2) fake1(p2, intr) fake1(p2, p1)
```

Fig. 2. An excerpt from the path to a state υ such that $\neg\mathrm{SP}(\upsilon)$ from `s436`.

The counterexample and $\mathrm{send3}_{p,q,n1,n2,nw}$ make a counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}.\,\mathrm{SP}(\upsilon)$. Command `search` can also be used to make such a counterexample. Let a constant `s436` equal the term of state 436 appearing in Fig. 1. Instead of `init`, `s436` is used to find a state such υ that $\mathrm{SP}(\upsilon)$ does not hold by feeding the following line into the Maude system:

```
search [1] s436 =>* (nonces : (N , Ns)) S
                    such that not(p1(N) == intr or p2(N) == intr) .
```

When `bound` is 1, such a state is found. An excerpt from the path to the state from `s436` is shown in Fig. 2. The two paths shown in Fig. 1 and Fig. 2 are combined to make a counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}.\,\mathrm{SP}(\upsilon)$.

8 Related Work

There are two main methods of falsifying (software and/or hardware) systems: testing and model checking [18]. Model checking is superior to testing in terms of coverage provided that systems should be basically modeled as finite-state transition systems. Even when a system can be modeled as a finite-state transition system, the system may not be model checked because the state space is too large for a computer on which model checking is performed. Bounded model checking, or BMC [13] can alleviate the problem. BMC uses a propositional SAT solver to search $\mathcal{R}_{\mathcal{S},\leq n}$ for a counterexample for a property written in propositional LTL for a fixed n, although a Kripke structure is used instead of an OTS. If no counterexample is found, BMC repeatedly increments n and performs the search until a counterexample is found, the search becomes intractable, or some pre-computed completeness threshold is reached.

In addition to modeling systems as finite-state transition systems, abstract data types such as lists and queues should be encoded in basic data types such as arrays and bounded integers because most existing model checkers do not allow to use abstract data types freely in a system to be model checked. The Maude model checker [19] allows to use abstract data types including inductively defined data types in a system to be model checked and does not require the state space of a system to be finite, although the reachable state space of a system should be finite. That is why we have decided to use Maude to falsify OTSs. Since Maude is not equipped with any BMC facilities, however, a way to search $\mathcal{R}_{\mathcal{S}^{\leq n}}$ for a counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}}. p(\upsilon)$ has been devised [12]. Note that the `search` command can be used to search an infinite state space of an OTS for a counterexample that the OTS satisfies an invariant property, but the termination is not guaranteed, which is required by procedure IGF.

A way to implement a local (or bounded) μ-calculus model checker in Maude using the Maude reflective facilities has been proposed [20]. The primary purpose of implementing or specifying the model checker in Maude is toward verification of the model checker. The bounded μ-calculus model checker could be used to search the bounded reachable state space $\mathcal{R}_{\mathcal{S},\leq n}$ for a counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}}. p(\upsilon)$. In terms of speed, however, the Maude `search` command and the Maude model checker are superior to the bounded μ-calculus model checker.

The induction-guided falsification can be considered a possible solution to the state explosion problem, which we often encounter when we try to model check if a system satisfies a property. Several possible solutions to the problem have been proposed. Their primary purpose is verification. One of the most popular methods is abstraction [21], which requires an original transition system and property to be modified. Instead of abstraction, our solution uses mathematical induction on the structure of the reachable state space of a transition system, which does not require an original transition system to be modified.

The induction-guided falsification can also be regarded as one possible combination of BMC and mathematical induction. There exists another possible combination of them: k-induction [22]. k-induction has been implemented in SAL

(Symbolic Analysis Laboratory) [23], which is a toolkit for analyzing transition systems. The primary purpose of k-induction is verification.

9 Conclusion

The induction-guided falsification has been described. The NSPK authentication protocol has been used as an example to demonstrate the induction-guided falsification. We have been developing a translator [24], which takes a CafeOBJ specification of an OTS and generates a Maude specifications of the OTS, and an automatic invariant verifier [25, 26] for OTSs, which uses an automatic case splitter that computes necessary lemmas. One piece of our future work is to use the translator and the automatic case splitter to automate the induction-guided falsification.

The basic idea in the proposed solution to find a counterexample that an OTS $\mathcal{S}$ satisfies an invariant property is as follows. When no counterexamples are found in the bounded reachable state space $\mathcal{R}_{\mathcal{S}}^{\leq n}$ and it is impossible to search $\mathcal{R}_{\mathcal{S}}^{\leq (n+1)}$ entirely, first discover all necessary lemmas of the invariant property and then search $\mathcal{R}_{\mathcal{S}}^{\leq n}$ for each of the necessary lemmas. The proposed solution guarantees if there exists a counterexample for the invariant property in $\mathcal{R}_{\mathcal{S}}^{\leq (n+1)}$, there exists a counterexample for at least one of the necessary lemmas in $\mathcal{R}_{\mathcal{S}}^{\leq n}$, and vice versa. Some may wonder how efficient it is to search $\mathcal{R}_{\mathcal{S}}^{\leq n}$ when compared to the search of $\mathcal{R}_{\mathcal{S}}^{\leq (n+1)}$. We suppose that $\mathcal{S}$ has one initial state and there are $x\,(\geq 2)$ (instances of) transitions whose effective conditions hold in each state. Then, the number of states in $\mathcal{R}_{\mathcal{S}}^{\leq n}$ is $\sum_{i=0}^{n} x^i$, which equals $(x^{n+1}-1)/(x-1)$. The difference between the number of states in $\mathcal{R}_{\mathcal{S}}^{\leq (n+1)}$ and that in $\mathcal{R}_{\mathcal{S}}^{\leq n}$ is x^{n+1}, which is greater than the number of states in $\mathcal{R}_{\mathcal{S}}^{\leq n}$ because $x^{n+1} - \sum_{i=0}^{n} x^i$ is $(x^{n+1}(x-2)+1)/(x-1)$. The greater x is, the greater the difference is. There are more than two (instances of) transitions whose effective conditions hold in each state in most applications. Therefore, the search of $\mathcal{R}_{\mathcal{S}}^{\leq n}$ is more efficient than that of $\mathcal{R}_{\mathcal{S}}^{\leq (n+1)}$. When three principles including the intruder participate the NSPK protocol, the number of states in $\mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}^{\leq 3}$ is 807, while that in $\mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}^{\leq 4}$ is 11323 and that in $\mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}^{\leq 5}$ is 180475.

Although procedure IGF can be used to verify that a state predicate p is invariant wrt an OTS $\mathcal{S}$, it is not efficient for the verification. This is because necessary lemmas are useful for finding counterexamples, i.e. falsification but they may not for verification. It is often the case that necessary lemmas should be strengthened to make the corresponding proofs more tractable. It is another piece of our future work to make the procedure useful for both verification and falsification.

As described at the end of Section 5, it depends on the number of some entities in an OTS whether procedure IGF works effectively if the number of the entities should be made finite. Therefore, we need to come up with something that can decide how many entities in an OTS $\mathcal{S}$ are enough to make sure that

there exists a counterexample for an invariant in the n-bounded reachable state space wrt $\mathcal{S}$ in which there are a finite number of the entities if there does so in the n-bounded reachable state space wrt $\mathcal{S}$ in which there are an infinite number of the entities.

References

1. Ogata, K., Futatsugi, K.: Proof scores in the OTS/CafeOBJ method. In: FMOODS 2003. LNCS 2884, Springer (2003) 170–184
2. Diaconescu, R., Futatsugi, K.: CafeOBJ Report. Volume 6 of AMAST Series in Computing. World Scientific (1998)
3. Ogata, K., Futatsugi, K.: Formally modeling and verifying Ricart&Agrawala distributed mutual exclusion algorithm. In: 2nd APAQS, IEEE CS Press (2001) 357–366
4. Ogata, K., Futatsugi, K.: Formal analysis of Suzuki&Kasami distributed mutual exclusion algorithm. In: 5th FMOODS, Kluwer (2002) 181–195
5. Ogata, K., Futatsugi, K.: Rewriting-based verification of authentication protocols. In: 4th WRLA 2002. ENTCS 71, Elsevier (2002)
6. Ogata, K., Futatsugi, K.: Formal analysis of the iKP electronic payment protocols. In: 1st ISSS. LNCS 2609, Springer (2003) 441–460
7. Ogata, K., Futatsugi, K.: Formal verification of the Horn-Preneel micropayment protocol. In: 4th VMCAI. LNCS 2575, Springer (2003) 238–252
8. Ogata, K., Futatsugi, K.: Equational approach to formal verification of SET. In: 4th QSIC, IEEE CS Press (2004) 50–59
9. Ogata, K., Futatsugi, K.: Formal analysis of the NetBill electronic commerce protocol. In: 2nd ISSS. LNCS 3233, Springer (2004) 45–64
10. Ogata, K., Futatsugi, K.: Equational approach to formal analysis of TLS. In: 25th ICDCS, IEEE CS Press (2005) 795–804
11. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.: Maude: Spesification and programming language in rewriting logic. TCS **285** (2002) 187–243
12. Ogata, K., Kong, W., Futatsugi, K.: Falsification of OTSs by searches of bounded reachable state spaces. In: 18th SEKE. (2006) 440–445
13. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. In: Advances in Computers. 58. Academic Press (2003)
14. Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. CACM **21** (1978) 993–999
15. Kong, W., Ogata, K., Futatsugi, K.: Model-checking observational transition system with Maude. In: 20th ITC-CSCC. (2005) 5–6
16. Hsiang, J., Dershowitz, N.: Rewrite methods for clausal and nonclausal theorem proving. In: 10th ICALP. LNCS 154, Springer (1983) 331–346
17. Ogata, K., Futatsugi, K.: Some tips on writing proof scores in the OTS/CafeOBJ method. In: Algebra, Meaning, and Computation: A Festschrift Symposium in Honor of Joseph Goguen. LNCS 4060, Springer (2006) 596–615
18. Edmund M. Clarke, J., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press (2001)
19. Eker, S., Meseguer, J., Sridharanarayanan, A.: The Maude LTL model checker. In: WRLA 2002. ENTCS 71, Elsevier (2002) 143–168

20. Wang, B.Y.: Specification of an infinite-state local model checker in rewriting logic. In: 17th SEKE. (2005) 442–447
21. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM TOPLAS **16** (1994) 1512–1542
22. de Moura, L., Rueß, H., Sorea, M.: Bounded model checking and induction: From refutation to verification. In: 15th CAV. LNCS 2392, Springer (2003) 14–26
23. de Moura, L., Owre, S., Rueß, H., Rushby, J., Shankar, N., Sorea, M., Tiwari, A.: SAL 2. In: 16th CAV. LNCS 3114, Springer (2004) 496–500
24. Kong, W., Ogata, K., Seino, T., Futatsugi, K.: Lightweight integration of theorem proving and model checking for system verification. In: 12th APSEC, IEEE CS Press (2005) 59–66
25. Nakano, M., Ogata, K., Nakamura, M., Futatsugi, K.: Automatic verification of the STS authentication protocol with Crème. In: 20th ITC-CSCC. (2005) 15–16
26. Nakano, M., Ogata, K., Nakamura, M., Futatsugi, K.: Automating invariant verification of behavioral specifications. In: 6th QSIC, IEEE CS Press (2006)