Proof Scores in the OTS/CafeOBJ Method

Kazuhiro Ogata[1,2] and Kokichi Futatsugi[2]

[1] NEC Software Hokuriku, Ltd.
ogatak@acm.org
[2] Japan Advanced Institute of Science and Technology (JAIST)
kokichi@jaist.ac.jp

Abstract. A way to write proof scores showing that distributed systems have invariant properties in algebraic specification languages is described, which has been devised through several case studies. The way makes it possible to divide a formula stating an invariant property under discussion into reasonably small ones, each of which is proved by writing proof scores individually. This relieves the load to reduce logical formulas and can decrease the number of subcases into which the case is split in case analysis.
Keywords: algebraic specification, CafeOBJ, observational transition system, proof scores, the NSLPK authentication protocol, verification.

1 Introduction

Equations are the most basic logical formulas and equational reasoning is the most fundamental way of reasoning[1], which can moderate the difficulties of proofs that might otherwise become too hard to understand. Algebraic specification languages make it possible to describe systems in terms of equations and verify that systems have properties by means of equational reasoning. Writing proofs, or proof scores in algebraic specification languages has been mainly promoted by researchers of the OBJ community[2].

We have been successfully applying such algebraic techniques to modeling, specification and verification of distributed systems such as distributed mutual exclusion algorithms[3,4] and security protocols[5, 6]. In our method called the OTS/CafeOBJ method, systems are modeled as observational transition systems, or OTSs, which are described in CafeOBJ[7, 8], an algebraic specification language. The CafeOBJ description of OTSs can be regarded as restricted behavioral specification[9]. We verify that OTSs, which are models of systems, have properties by writing proof scores in CafeOBJ.

In this paper, we describe a way to write proof scores showing that distributed systems have invariant properties, which are most basic and important among various kinds of properties because proofs of other kinds of properties often need invariants. We have devised the way through several case studies[3–6]. The way makes it possible to divide a formula stating an invariant property under discussion into reasonably small ones, each of which is proved by writing proof scores individually. This relieves the load to reduce logical formulas and can

decrease the number of subcases into which the case is split in case analysis. The proofs of the small formulas may depend on each other in the sense that the proof of one uses some other to strengthen inductive hypotheses and vice versa.

The rest of the paper is organized as follows. Section 2 mentions CafeOBJ and OTSs. Section 3 describes compositional proofs of invariants. A way of writing proof scores based on the compositional proofs of invariants is described in Sect. 4. Section 5 uses the NSLPK authentication protocol[10, 11] as an example to demonstrate how to write proof scores. Section 6 discusses the advantages of our method and concludes the paper.

2 Preliminaries

2.1 CafeOBJ in a Nutshell

CafeOBJ[7, 8] can be used to specify abstract machines as well as abstract data types. A visible sort denotes an abstract data type, while a hidden sort the state space of an abstract machine. There are two kinds of operators to hidden sorts: action and observation operators. An action operator can change states of an abstract machine. Only observation operators can be used to observe the inside of an abstract machine. An action operator is basically specified with equations by describing how the value of each observation operator changes. Declarations of observation and action operators start with `bop` or `bops`, and those of other operators with `op` or `ops`. Declarations of equations start with `eq`, and those of conditional ones with `ceq`. The CafeOBJ system rewrites a given term by regarding equations as left-to-right rewrite rules.

2.2 Observational Transition Systems

We assume that there exists a universal state space called Υ. We also suppose that each data type used has been defined beforehand, including the equivalence between two data values v_1, v_2 denoted by $v_1 = v_2$. A system is modeled by observing, from the outside of each state of Υ, only quantities that are relevant to the system and how to change the quantities by state transition. An OTS (observational transition system) can be used to model a system in this way. An OTS $\mathcal{S} = \langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ consists of:

- $\mathcal{O}$: A set of observable values. Each $o \in \mathcal{O}$ is a function $o : \Upsilon \to D$, where D is a data type and may be different for each observable value. Given an OTS $\mathcal{S}$ and two states $v_1, v_2 \in \Upsilon$, the equivalence between two states, denoted by $v_1 =_{\mathcal{S}} v_2$, w.r.t. $\mathcal{S}$ is defined as $v_1 =_{\mathcal{S}} v_2 \stackrel{\text{def}}{=} \forall o \in \mathcal{O}.o(v_1) = o(v_2)$.
- $\mathcal{I}$: The set of initial states such that $\mathcal{I} \subset \Upsilon$.
- $\mathcal{T}$: A set of conditional transition rules. Each $\tau \in \mathcal{T}$ is a function $\tau : \Upsilon/=_{\mathcal{S}} \to \Upsilon/=_{\mathcal{S}}$ on equivalence classes of Υ w.r.t. $=_{\mathcal{S}}$. Let $\tau(v)$ be the representative element of $\tau([v])$ for each $v \in \Upsilon$ and it is called *the successor state* of v w.r.t. τ. The condition c_τ for a transition rule $\tau \in \mathcal{T}$, which is a predicate on states, is called *the effective condition*. The effective condition is supposed

to satisfy the following requirement: given a state $v \in \Upsilon$, if c_τ is false in v, namely τ is not *effective* in v, then $v =_{\mathcal{S}} \tau(v)$.

An OTS is described in CafeOBJ. Observable values are denoted by CafeOBJ observations, and transition rules by CafeOBJ actions.

Multiple similar observable values and transition rules may be indexed. Generally, observable values and transition rules are denoted by $o_{i_1,\ldots,i_m}$ and $\tau_{j_1,\ldots,j_n}$, respectively, provided that $m, n \geq 0$ and we assume that there exist data types D_k such that $k \in D_k$ $(k = i_1, \ldots, i_m, j_1, \ldots, j_n)$. For example, an integer array a possessed by a process p may be denoted by an observable value a_p, and the increment of the ith element of the array may be denoted by a transition rule $inc\text{-}a_{p,i}$.

An execution of $\mathcal{S}$ is an infinite sequence $v_0, v_1, \ldots$ of states satisfying[3]:

- *Initiation*: $v_0 \in \mathcal{I}$.
- *Consecution*: For each $i \in \{0, 1, \ldots\}$, $v_{i+1} =_{\mathcal{S}} \tau(v_i)$ for some $\tau \in \mathcal{T}$.

A state is called *reachable* w.r.t. $\mathcal{S}$ iff it appears in an execution of $\mathcal{S}$. Let $\mathcal{R}_{\mathcal{S}}$ be the set of all the reachable states w.r.t. $\mathcal{S}$.

All properties considered in this paper are invariants[4], which are defined as follows:

$$\mathsf{invariant}\ p \stackrel{\text{def}}{=} (\forall v \in \mathcal{I}.\, p(v)) \wedge (\forall v \in \mathcal{R}_{\mathcal{S}}.\forall \tau \in \mathcal{T}.(p(v) \Rightarrow p(\tau(v))))\,,$$

which means that the predicate p is true in any reachable state of $\mathcal{S}$. Let $\boldsymbol{x}$ be all free variables except for one for states in p. We suppose that $\mathsf{invariant}\, p$ is interpreted as $\forall \boldsymbol{x}.(\mathsf{invariant}\, p)$ in this paper.

2.3 Description of OTSs in CafeOBJ

An OTS $\mathcal{S}$ is described in CafeOBJ. The universal state space Υ is denoted by a hidden sort, say H. An observable value $o_{i_1,\ldots,i_m} \in \mathcal{O}$ is denoted by a CafeOBJ observation operator. We assume that the data types D_k $(k = i_1, \ldots, i_m)$ and D are described in initial algebra and there exist visible sorts V_k $(k = i_1, \ldots, i_m)$ and V corresponding to the data types. The CafeOBJ observation operator denoting $o_{i_1,\ldots,i_m}$ is declared as follows:

```
bop o : H V_{i_1} ... V_{i_m} -> V .
```

Any initial state in $\mathcal{I}$ is denoted by a constant (an operator with no arguments), say *init*, which is declared as follows:

[3] If we want to discuss liveness properties, an execution of $\mathcal{S}$ should also satisfy *Fairness*: for each $\tau \in \mathcal{T}$, there exist an infinite number of indexes $i \in \{0, 1, \ldots\}$ such that $v_{i+1} =_S \tau(v_i)$.

[4] In addition to invariant properties, there are unless, stable, ensures and leads-to properties, which are inspired by UNITY[12]. The way to write proof scores described in this paper can also be applied to unless, stable, ensures properties.

`op` *init* : `->` H

Suppose that the initial value of $o_{i_1,\ldots,i_m}$ is $f(i_1,\ldots,i_m)$. This is expressed by the following equation:

`eq` $o(init, X_{i_1},\ldots,X_{i_m}) = f(X_{i_1},\ldots,X_{i_m})$.

$X_k\,(k = i_1,\ldots,i_m)$ is a CafeOBJ variable for V_k and $f(X_{i_1},\ldots,X_{i_m})$ is a term denoting $f(i_1,\ldots,i_m)$.

A transition rule $\tau_{j_1,\ldots,j_n} \in \mathcal{T}$ is denoted by a CafeOBJ action operator. We assume that the data types $D_k\,(k = j_1,\ldots,j_n)$ are described in initial algebra and there exist visible sorts $V_k\,(k = j_1,\ldots,j_n)$ corresponding to the data types. The CafeOBJ action operator denoting $\tau_{j_1,\ldots,j_n}$ is declared as follows:

`bop` a : $H\ V_{j_1}\ \ldots\ V_{j_n}$ `->` H .

If $\tau_{j_1,\ldots,j_n}$ is applied in a state in which it is effective, the value of $o_{i_1,\ldots,i_m}$ may be changed, which can be described in CafeOBJ generally as follows:

`ceq` $o(a(W, X_{j_1},\ldots,X_{j_n}), X_{i_1},\ldots,X_{i_m}) = e\text{-}a(W, X_{j_1},\ldots,X_{j_n}, X_{i_1},\ldots,X_{i_m})$
`if` $c\text{-}a(W, X_{j_1},\ldots,X_{j_n})$.

W is a CafeOBJ variable for H and $X_k\,(k = j_1,\ldots,i_n)$ is a CafeOBJ variable for V_k. $a(W, X_{j_1},\ldots,X_{j_n})$ denotes the successor state of W w.r.t. $\tau_{j_1,\ldots,j_n}$. $e\text{-}a(W, X_{j_1},\ldots,X_{j_n}, X_{i_1},\ldots,X_{i_m})$ denotes the value of $o_{i_1,\ldots,i_m}$ in the successor state. $c\text{-}a(W, X_{j_1},\ldots,X_{j_n})$ denotes the effective condition $c_{\tau_{j_1,\ldots,j_n}}$ of $\tau_{j_1,\ldots,j_n}$.

If $\tau_{j_1,\ldots,j_n}$ is applied in a state in which it is not effective, the value of any observable value is not changed. Therefore all we have to do is to declare the following equation:

`ceq` $a(W, X_{j_1},\ldots,X_{j_n}) = W$ `if not` $c\text{-}a(W, X_{j_1},\ldots,X_{j_n})$.

If the value of $o_{i_1,\ldots,i_m}$ is not affected by applying $\tau_{j_1,\ldots,j_n}$ in any state (regardless of the truth value of $c_{\tau_{j_1,\ldots,j_n}}$), the following equation may be declared:

`eq` $o(a(W, X_{j_1},\ldots,X_{j_n}), X_{i_1},\ldots,X_{i_m}) = o(W, X_{i_1},\ldots,X_{i_m})$.

3 Compositional Proofs of Invariants

Suppose that we prove that a system has an invariant property. The system is first modeled as an OTS, which is described in CafeOBJ. Let H be the hidden sort denoting the state space Υ, and let the invariant be invariant $pred_1(s, \boldsymbol{x}_1)$, where s is a free variable for states and $\boldsymbol{x}_1$ the other free variables. It is often impossible to prove invariant $pred_1(s, \boldsymbol{x}_1)$ alone. Suppose that it is possible to prove that $pred_1(s, \boldsymbol{x}_1)$, together with $n-1$ other predicates, is invariant to the OTS. Let the $n-1$ predicates be $pred_2(s, \boldsymbol{x}_2),\ldots,pred_n(s, \boldsymbol{x}_n)$. That is, we prove invariant $(pred_1(s, \boldsymbol{x}_1) \wedge \ldots \wedge pred_n(s, \boldsymbol{x}_n))$, instead of the original invariant, from which the original invariant can be deduced. Let $pred(s, \boldsymbol{x}_1,\ldots,\boldsymbol{x}_n)$ be $pred_1(s, \boldsymbol{x}_1) \wedge \ldots \wedge pred_n(s, \boldsymbol{x}_n)$.

Although sometimes invariants may be proved by reduction and/or case analysis only, we often need to use induction, especially induction on the number of transition rules applied or executed.

Suppose that invariant $pred(s, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ is proved by induction on the number of transition rules applied. Let us consider an inductive case in which it is shown that any transition rule denoted by a CafeOBJ action operator a preserves $pred(s, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$. To this end, it is sufficient to show this formula:

$$pred(s, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \Rightarrow pred(\mathrm{a}(s, \boldsymbol{y}), \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \tag{1}$$

for any $s, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \boldsymbol{y}$, where $\boldsymbol{y}$ is the arguments of the CafeOBJ action operator except for s. It is often the case that we cannot prove the formula as it is because the inductive hypothesis $pred(s, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ is too weak. Then, we can strengthen the inductive hypothesis by adding a formula, say SIH, of the following form:

$$pred(s, \boldsymbol{t}_1^1, \ldots, \boldsymbol{t}_n^1) \wedge \ldots \wedge pred(s, \boldsymbol{t}_1^m, \ldots, \boldsymbol{t}_n^m)\,,$$

where $\boldsymbol{t}_1^i, \ldots, \boldsymbol{t}_n^i$ $(i = 1, \ldots, m)$ are lists of terms. Then, the proof of (1) can be replaced with the proof of the following formula:

$$(SIH \wedge pred(s, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)) \Rightarrow pred(\mathrm{a}(s, \boldsymbol{y}), \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)\,.$$

This formula can be proved compositionally. The proof of the formula is equivalent to the proofs of the following n formulas:

$$\begin{aligned} (SIH \wedge pred(s, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)) &\Rightarrow pred_1(\mathrm{a}(s, \boldsymbol{y}), \boldsymbol{x}_1)\,, \\ &\vdots \\ (SIH \wedge pred(s, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)) &\Rightarrow pred_n(\mathrm{a}(s, \boldsymbol{y}), \boldsymbol{x}_n)\,. \end{aligned} \tag{2}$$

Moreover, it suffices to prove the following n formulas, if possible, instead of the above n formulas:

$$\begin{aligned} pred_1(s, \boldsymbol{x}_1) &\Rightarrow pred_1(\mathrm{a}(s, \boldsymbol{y}), \boldsymbol{x}_1)\,, \\ &\vdots \\ pred_n(s, \boldsymbol{x}_n) &\Rightarrow pred_n(\mathrm{a}(s, \boldsymbol{y}), \boldsymbol{x}_n)\,, \end{aligned} \tag{3}$$

because the ith formula of (2) can be deduced from the ith formula of (3), where $1 \leq i \leq n$.

Some of (3) cannot be proved as they are because their inductive hypotheses are too weak. Let $pred_i(s, \boldsymbol{x}_i) \Rightarrow pred_i(\mathrm{a}(s, \boldsymbol{y}), \boldsymbol{x}_i)$, where $1 \leq i \leq n$, be one of such formulas. Suppose that $pred_j(s, \boldsymbol{u}_j)$, where $1 \leq j \leq n$ and $\boldsymbol{u}_j$ is $\boldsymbol{x}_j$, $\boldsymbol{t}_j^1$, $\ldots$, or $\boldsymbol{t}_j^m$, can be used to strengthen the inductive hypothesis $pred_i(s, \boldsymbol{x}_i)$ in order to prove the formula. The proof of the formula can be replaced with the proof of the following formula[5]:

$$(pred_j(s, \boldsymbol{u}_j) \wedge pred_i(s, \boldsymbol{x}_i)) \Rightarrow pred_i(\mathrm{a}(s, \boldsymbol{y}), \boldsymbol{x}_i)\,,$$

[5] If invariant $pred_j(s, \boldsymbol{x}_j)$ has been proved independent of invariant $pred_i(s, \boldsymbol{x}_i)$, the proof can also be replaced with the proof of the following:

$$(pred_j(\mathrm{a}(s, \boldsymbol{y}), \boldsymbol{u}_j) \wedge pred_i(s, \boldsymbol{x}_i)) \Rightarrow pred_i(\mathrm{a}(s, \boldsymbol{y}), \boldsymbol{x}_i)\,.$$

In this case, the jth invariant is used as usual lemma for the proof of the ith invariant.

because the ith formula of (2) can be deduced from this formula. Generally what strengthens the inductive hypothesis can be $pred_{j_1}(s, \boldsymbol{u}_{j_1}) \wedge \ldots \wedge pred_{j_k}(s, \boldsymbol{u}_{j_k})$, where $1 \leq j_1, \ldots, j_k \leq n$ and $\boldsymbol{u}_j$ $(j = j_1, \ldots, j_k)$ is $\boldsymbol{x}_j$, $\boldsymbol{t}_j^1$, $\ldots$, or $\boldsymbol{t}_j^m$. Let SIH_i be this formula to strengthen the inductive hypothesis $pred_i(s, \boldsymbol{x}_i)$. Then, the proof of the ith formula of (3) can be replaced with the proof of the following:

$$(SIH_i \wedge pred_i(s, \boldsymbol{x}_i)) \Rightarrow pred_i(\mathrm{a}(s, \boldsymbol{y}), \boldsymbol{x}_i)\,. \tag{4}$$

Moreover, we may have to split the case into multiple subcases in order to prove (4). Suppose that the case is split into l subcases. The l subcases are denoted by l formulas $case_1^i, \ldots, case_l^i$, which should satisfy the following:

$$(case_1^i \vee \ldots \vee case_l^i) = \mathrm{true}.$$

Then, the proof of (4) can be replaced with the proofs of the following l formulas:

$$\begin{aligned} &(case_1^i \wedge SIH_i \wedge pred_i(s, \boldsymbol{x}_i)) \Rightarrow pred_i(\mathrm{a}(s, \boldsymbol{y}), \boldsymbol{x}_i)\,, \\ &\qquad\vdots \\ &(case_l^i \wedge SIH_i \wedge pred_i(s, \boldsymbol{x}_i)) \Rightarrow pred_i(\mathrm{a}(s, \boldsymbol{y}), \boldsymbol{x}_i)\,. \end{aligned} \tag{5}$$

SIH_i may not be needed for some subcases.

From what has been discussed, it follows that the n invariants can be proved compositionally even if they depend on each other, namely that the ith invariant is used to strengthen inductive hypotheses for the proof of the jth invariant and vice versa. The original invariant $\mathsf{invariant}\ pred_1(s, \boldsymbol{x}_1)$, which we would like to prove, may be divided into multiple invariants. Proof scores in the OTS/CafeOBJ method are based on what has been discussed, especially (5), and therefore, we can write proof scores of the n invariants individually.

4 Proof Scores of Invariants

Let us consider that we write proof scores of the n invariants discussed in the previous section. We first write a module, say `INV`, where $pred_i(s, \boldsymbol{x}_i)$ $(i = 1, \ldots, n)$ is expressed as a CafeOBJ term as follows:

```
op inv1 : H V1 -> Bool
...
op invn : H Vn -> Bool
eq inv1(W, X1) = pred1(W, X1) .
...
eq invn(W, Xn) = predn(W, Xn) .
```

$\boldsymbol{V}_i$ $(i = 1, \ldots, n)$ is a list of visible sorts corresponding to $\boldsymbol{x}_i$, W is a CafeOBJ variable for the hidden sort H and $\boldsymbol{X}_i$ $(i = 1, \ldots, n)$ is a list of CafeOBJ variables for $\boldsymbol{V}_i$. The term $pred_i(W, \boldsymbol{X}_i)$ $(i = 1, \ldots, n)$ denotes $pred_i(s, \boldsymbol{x}_i)$.

In the module, we also declare constants $\boldsymbol{x}_i$ $(i = 1, \ldots, n)$ for $\boldsymbol{V}_i$. In proof scores, a constant that is not constrained is used for denoting an arbitrary object

for the intended sort. For example, if we declare a constant x for `Nat` that is the visible sort for natural numbers in a proof score, x can be used to denote an arbitrary natural number. Such constants are constrained with equations, which make it possible to split the state space, or the case. Suppose that the case is split into two: one where x equals 0 and the other where x does not, namely that x is greater than 0. The former is expressed by declaring the following equation:

```
eq x = 0 .
```

The latter is expressed by declaring the following equation:

```
eq (x > 0) = true .
```

We are going to mainly describe the proof score of the ith invariant. Let *init* denote any initial state of the system under consideration. All we have to do to show that $pred_i(s, \boldsymbol{x}_i)$ holds in any initial state is to write the CafeOBJ code, which looks like this:

```
open INV
  red inv_i(init, x_i) .
close
```

We next write a module, say `ISTEP`, where two constants s, s' are declared, denoting any state and the successor state after applying a transition rule in the state, and the predicates to prove in each inductive case are expressed as CafeOBJ terms as follows:

```
op istep_1 : V_1 -> Bool
...
op istep_n : V_n -> Bool
eq istep_1(X) = inv_1(s, X_1) implies inv_1(s', X_1) .
...
eq istep_n(X) = inv_n(s, X_n) implies inv_n(s', X_n) .
```

These predicates correspond to (3) in the previous section.

In each inductive case, the case is usually split into multiple subcases with basic predicates declared in the CafeOBJ specification. Suppose that we prove that any transition rule denoted by a CafeOBJ action operator a preserves $pred_i(s, \boldsymbol{x}_i)$. As described in the previous section, the case is supposed to be split into the l subcases $case_1^i, \ldots, case_l^i$. Then, the CafeOBJ code showing that the transition rule preserves $pred_i(s, \boldsymbol{x}_i)$ for $case_j^i$ $(j = 1, \ldots, l)$ looks like this:

```
open ISTEP
  Declare constants denoting arbitrary objects.
  Declare equations denoting case_j^i.
  Declare equations denoting facts if necessary.
  eq s' = a(s, y) .
  red istep_i(x_i) .
close
```

$\boldsymbol{y}$ is a list of constants that are used as the arguments of the CafeOBJ action operator a, which are declared in this CafeOBJ code and denote arbitrary objects for the intended sorts. In addition to $\boldsymbol{y}$, other constants may be declared in the CafeOBJ code for the case split. Equations are used to express $case_j^i$. If necessary, equations denoting facts about data structures used, etc. may be declared as well. The equation with s' as its left-hand side specifies that s' denotes the successor state after applying any transition rule denoted by a in the state denoted by s.

If $istep_i(\boldsymbol{x}_i)$ is reduced to `true`, it is shown that the transition rule preserves $pred_i(p, \boldsymbol{x})$ in the subcase j, which corresponds to the proof of the jth formula of (5) in the previous section. Otherwise, we have to strengthen the inductive hypothesis in the way described in the previous section. Let SIH_i be the term denoting SIH_i. Then, instead of $istep_i(\boldsymbol{x}_i)$, we reduce the following term

$$(SIH_i \texttt{ and } inv_i(s, \boldsymbol{x}_i)) \texttt{ implies } inv_i(s', \boldsymbol{x}_i),$$

or

$$SIH_i \texttt{ implies } istep_i(\boldsymbol{x}_i).$$

5 Example: The NSLPK Authentication Protocol

The NSLPK authentication protocol[10, 11] is the modified version of the NSPK authentication protocol[13] by G. Lowe. The protocol can be described as follows:

$$\begin{aligned} &\text{Msg1 } p \to q : \mathcal{E}_q(n_p, p) \\ &\text{Msg2 } q \to p : \mathcal{E}_p(n_p, n_q, q) \\ &\text{Msg3 } p \to q : \mathcal{E}_q(n_q) \end{aligned}$$

Suppose that each principal is given a private/public key pair, and the public counterpart is available to all principals but the private counterpart to its owner only. Given a message m, the one encrypted with the public key given to a principal p is denoted by $\mathcal{E}_p(m)$.

If a principal p wants a principal q to authenticate herself/himself and wants to authenticate q, she/he newly generates a nonce n_p and sends it to q, together with her/his ID, encrypted with q's public key. On receipt of the message, q first decrypts it, obtains a nonce and a principal ID, and checks if the principal ID matches the sender of the message. Then, q newly generates a nonce n_q and sends it to p, together with the received nonce and her/his ID, encrypted with p's public key. On receipt of the message, p first decrypts it and obtains two nonces and a principal ID, and checks if the principal ID equals the sender of the message and one of the nonces is the exact one that p has sent to the sender in this session, which is supposed to convince p that the responder is really q. Then, p sends the other nonce to q, encrypted with q's public key. On receipt of the message, q decrypts it, obtains a nonce and checks if the nonce is the exact one that q has sent to the sender in this session, which supposedly assures q that the initiator is really p.

5.1 Modeling and Description of the Protocol

We suppose that there exist untrustable principals as well as trustable ones. Trustable principals exactly follow the protocol, but untrustable ones may do something against the protocol as well, namely eavesdropping and/or faking messages. The combination and cooperation of untrustable principals is modeled as the most general intruder à la Dolev and Yao[14]. The intruder can do the following:

- Eavesdrop any message flowing in the network.
- Glean any nonce and cipher from the message; however the intruder can decrypt a cipher only if she/he knows the key to decrypt.
- Fake and send messages based on the gleaned information; however the intruder cannot guess unknown nonces.

We first describe the basic data types used to model the protocol. The visible sorts and the corresponding data constructors are as follows:

- `Principal` denotes principals.
- `Random` denotes random numbers, which make nonces unguessable and unique.
- `Nonce` denotes nonces. Given principals p, q and a random number r, $\mathtt{n}(p, q, r)$ denotes a nonce created by p for q. Projections `creator`, `forwhom` and `random` return the first, second and third arguments.
- `Cipher1` denotes ciphers used in Msg1's. Given principals p, q and a nonce n, $\mathtt{enc1}(p, n, q)$ denotes $\mathcal{E}_p(n, q)$. Projections `key`, `nonce` and `principal` return the first, second and third arguments.
- `Cipher2` denotes ciphers used in Msg2's. Given principals p, q and nonces $n1, n2$, $\mathtt{enc2}(p, n1, n2, q)$ denotes $\mathcal{E}_p(n1, n2, q)$. Projections `key`, `nonce1`, `nonce2` and `principal` return the first, second, third and fourth arguments.
- `Cipher3` denotes ciphers used in Msg3's. Given a principal p and a nonce n, $\mathtt{enc3}(p, n)$ denotes $\mathcal{E}_p(n)$. Projections `key` and `nonce` return the first and second arguments.

In addition to those visible sorts, we use the visible sort `Bool` that denotes truth values, declared in the built-in module `BOOL`, where the constants `true` and `false` with usual meanings, and the operators `not_` for negation, `_and_` for conjunction, `_or_` for disjunction and `_implies_` for implication are declared. An underscore `_` indicates where an argument is put.

The three operators (data constructors) to denote the three kinds of messages are declared as follows:

```
op m1 : Principal Principal Principal Cipher1 -> Message
op m2 : Principal Principal Principal Cipher2 -> Message
op m3 : Principal Principal Principal Cipher3 -> Message
```

The visible sort `Message` denotes messages. Projections `creator`, `sender` and `receiver` return the first, second and third arguments of each message. A projection `cipher`i $(i = 1, 2, 3)$ returns the fourth argument of Msg$i$. A predicate `m`i`?` checks if a given message is Msg$i$. The first, second and third arguments of

each constructor mean the actual creator, the seeming sender and the receiver of the corresponding message. The first argument is meta-information that is only available to the outside observer and the principal that has sent the corresponding message, and that cannot be forged by the intruder, while the remaining arguments may be forged by the intruder.

The network is modeled as a bag (multiset) of messages, which is used as the storage that the intruder can use. The network is also used as each principal's private memory that reminds the principal to send messages, of which the first argument is the principal. Any message that has been sent or put once into the network is supposed to be never deleted from the network because the intruder can replay the message repeatedly, although the intruder cannot forge the first argument. Consequently, the emptiness of the network means that no messages have been sent.

The intruder tries to glean four kinds of quantities from the network. The four kinds of quantities are nonces and three kinds of ciphers. The collections of those quantities gleaned by the intruder are denoted by the following operators:

```
op cnonce : Network -> ColNonce      op cenc1  : Network -> ColCipher1
op cenc2  : Network -> ColCipher2    op cenc3  : Network -> ColCipher3
```

The visible sort `Network` denotes networks and the visible sort `Col`X denotes collections of quantities denoted by the visible sort X. Those operators are defined with equations.

`cnonce` is defined as follows:

```
eq  N \in cnonce(void) = (creator(N) = intruder) .
ceq N \in cnonce(M,NW) = true
    if m1?(M) and key(cipher1(M)) = intruder and nonce(cipher1(M)) = N .
ceq N \in cnonce(M,NW) = true
    if m2?(M) and key(cipher2(M)) = intruder and nonce1(cipher2(M)) = N .
ceq N \in cnonce(M,NW) = true
    if m2?(M) and key(cipher2(M)) = intruder and nonce2(cipher2(M)) = N .
ceq N \in cnonce(M,NW) = true
    if m3?(M) and key(cipher3(M)) = intruder and nonce(cipher3(M)) = N .
ceq N \in cnonce(M,NW) = N \in cnonce(NW)
    if not(m1?(M) and key(cipher1(M)) = intruder and nonce(cipher1(M)) = N) and
       not(m2?(M) and key(cipher2(M)) = intruder and nonce1(cipher2(M)) = N) and
       not(m2?(M) and key(cipher2(M)) = intruder and nonce2(cipher2(M)) = N) and
       not(m3?(M) and key(cipher3(M)) = intruder and nonce(cipher3(M)) = N) .
```

The constant `void` denotes the empty bag and the operator `_,_` denotes the data constructor of nonempty bags. The operator `_\in_` is the membership predicate of bags. The equations say that nonces created by the intruder are always available to the intruder, and a nonce created by one of the other principals is available to the intruder iff there exists a message in the network, and the cipher in the message is encrypted with the intruder's public key and includes the nonce.

`cenc1` is defined as follows:

```
eq  E1 \in cenc1(void) = false .
ceq E1 \in cenc1(M,NW) = true
    if m1?(M) and not(key(cipher1(M)) = intruder) and E1 = cipher1(M) .
ceq E1 \in cenc1(M,NW) = E1 \in cenc1(NW)
    if not(m1?(M) and not(key(cipher1(M)) = intruder) and E1 = cipher1(M)) .
```

The equations say that no ciphers appearing in Msg1's are available if the network is empty, and the intruder glean such a cipher from the network iff there

exists a Msg1 in the network and the cipher in the message is not encrypted with the intruder's public key. If the cipher is encrypted with the intruder's public key, the intruder can rebuild the cipher and it is not necessary to collect it. `cenc2` and `cenc3` can be defined likewise.

We are about to describe the OTS modeling the protocol. Two observable values and nine kinds of transition rules are used. The corresponding CafeOBJ observations and actions are as follows:

```
-- observations
bop ur : System -> URand
bop nw : System -> Network
-- actions
bop sdm1  : System Principal Principal Random       -> System
bop sdm2  : System Principal Random Message         -> System
bop sdm3  : System Principal Random Message Message -> System
bop fkm11 : System Principal Principal Cipher1      -> System
bop fkm12 : System Principal Principal Nonce        -> System
bop fkm21 : System Principal Principal Cipher2      -> System
bop fkm22 : System Principal Principal Nonce Nonce  -> System
bop fkm31 : System Principal Principal Cipher3      -> System
bop fkm32 : System Principal Principal Nonce        -> System
```

The hidden sort `System` denotes the state space. The observation `ur` denotes the set of used random numbers and the observation `nw` denotes the network. The first three actions formalize sending messages following to the protocol, and the remaining the intruder's faking messages.

The equations to define `sdm2` are as follows:

```
op c-sdm2 : System Principal Random Message -> Bool
eq c-sdm2(S,Q,R,M)
   = (M \in nw(S) and m1?(M) and receiver(M) = Q and key(cipher1(M)) = Q and
      principal(cipher1(M)) = sender(M) and not(R \in ur(S))) .
--
ceq ur(sdm2(S,Q,R,M)) = R ur(S) if c-sdm2(S,Q,R,M) .
ceq nw(sdm2(S,Q,R,M))
    = m2(Q,Q,sender(M),enc2(sender(M),nonce(cipher1(M)),n(Q,sender(M),R),Q)) , nw(S)
    if c-sdm2(S,Q,R,M) .
ceq sdm2(S,Q,R,M)     = S if not c-sdm2(S,Q,R,M) .
```

The operator `c-sdm2` denotes the effective condition of any transition rule denoted by `sdm2`. `c-sdm2`(s, q, r, m) means that in a state s, there exists a Msg1 m in the network that is addressed to q, the cipher in m is encrypted with q's public key, the principal in the cipher equals the seeming sender, and the nonce generated by q for replying to m is really fresh. If this condition holds, the Msg2 denoted by the term `m2`$(\ldots)$ is put into the network. The juxtaposition operator `__` of '`R ur(S)`' is the data constructor of nonempty sets.

The equations to define `fkm22` are as follows:

```
op c-fkm22 : System Principal Principal Nonce Nonce -> Bool
eq c-fkm22(S,P,Q,N1,N2) = N1 \in cnonce(nw(S)) and N2 \in cnonce(nw(S)) .
--
eq  ur(fkm22(S,P,Q,N1,N2)) = ur(S) .
ceq nw(fkm22(S,P,Q,N1,N2)) = m2(intruder,P,Q,enc2(Q,N1,N2,P)) , nw(S)
    if c-fkm22(S,P,Q,N1,N2) .
ceq fkm22(S,P,Q,N1,N2)     = S if not c-fkm22(S,P,Q,N1,N2) .
```

The equations say that if two nonces are available to the intruder, the intruder can fake and send a Msg2. The constant `intruder` denotes the intruder.

5.2 Proof Scores of Nonce Secrecy

We describe the proof scores showing that nonces are really secret in the protocol, which means that the intruder cannot obtains nonces generated by another principal for yet another one illegally. This can be expressed by the following invariant:

$$\begin{array}{l}\mathsf{invariant}\ (n\ \backslash\mathtt{in}\ \mathtt{cnonce}(\mathtt{nw}(s))\ \mathtt{implies}\\ \qquad (\mathtt{creator}(n) = \mathtt{intruder}\ \mathtt{or}\ \mathtt{forwhom}(n) = \mathtt{intruder}))\,.\end{array} \tag{6}$$

It is impossible to prove this invariant alone. We need six more invariants, which are as follows:

$$\mathsf{invariant}\ (e1\ \backslash\mathtt{in}\ \mathtt{cenc1}(\mathtt{nw}(s))\ \mathtt{implies}\ \mathtt{not}(\mathtt{key}(e1) = \mathtt{intruder}))\,, \tag{7}$$

$$\mathsf{invariant}\ (e2\ \backslash\mathtt{in}\ \mathtt{cenc2}(\mathtt{nw}(s))\ \mathtt{implies}\ \mathtt{not}(\mathtt{key}(e2) = \mathtt{intruder}))\,, \tag{8}$$

$$\mathsf{invariant}\ (e3\ \backslash\mathtt{in}\ \mathtt{cenc3}(\mathtt{nw}(s))\ \mathtt{implies}\ \mathtt{not}(\mathtt{key}(e3) = \mathtt{intruder}))\,, \tag{9}$$

$$\begin{array}{l}\mathsf{invariant}\ (e1\ \backslash\mathtt{in}\ \mathtt{cenc1}(\mathtt{nw}(s))\ \mathtt{and}\ \mathtt{principal}(e1) = \mathtt{intruder}\\ \qquad \mathtt{implies}\ \mathtt{nonce}(e1)\ \backslash\mathtt{in}\ \mathtt{cnonce}(\mathtt{nw}(s)))\,,\end{array} \tag{10}$$

$$\begin{array}{l}\mathsf{invariant}\ (e2\ \backslash\mathtt{in}\ \mathtt{cenc2}(\mathtt{nw}(s))\ \mathtt{and}\ \mathtt{principal}(e2) = \mathtt{intruder}\\ \qquad \mathtt{implies}\ \mathtt{nonce}(e2)\ \backslash\mathtt{in}\ \mathtt{cnonce}(\mathtt{nw}(s)))\,,\end{array} \tag{11}$$

$$\mathsf{invariant}\ (\mathtt{creator}(n) = \mathtt{intruder}\ \mathtt{implies}\ n\ \backslash\mathtt{in}\ \mathtt{cnonce}(\mathtt{nw}(s)))\,. \tag{12}$$

The proof of (6) uses (7), (8), (9), (10) and (11) to strengthen inductive hypotheses. The proofs of (10) and (11) use (7), (8), (9) and (12) to strengthen inductive hypotheses. (7), (8), (9) and (12) can be proved independently.

In this paper, we partly show the proof scores showing that any transition rule denoted by `sdm2` preserves (6), which uses (10) to strengthen inductive hypotheses. As described in Sect. 4, (6) and (10) are first expressed as CafeOBJ terms, which are denoted by the operators `inv1` and `inv2` that are declared and defined as follows:

```
op inv1 : System Nonce -> Bool
op inv2 : System Cipher1 -> Bool
eq inv1(S,N) = (N \in cnonce(nw(S))
                implies (creator(N) = intruder or forwhom(N) = intruder)) .
eq inv2(S,E1) = (E1 \in cenc1(nw(S)) and principal(E1) = intruder
                 implies nonce(E1) \in cnonce(nw(S))) .
```

A constant `n` for `Nonce` and a constant `e1` for `Cipher1` are also declared. The predicates to prove in each inductive case are next expressed as CafeOBJ terms, which are denoted by the operators `istep1` and `istep2` that are declared and defined as follows:

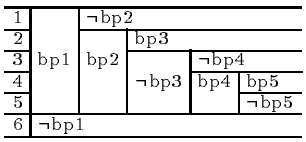
```
op istep1 : Nonce -> Bool
op istep2 : Cipher1 -> Bool
eq istep1(N) = inv1(s,N) implies inv1(s',N) .
eq istep2(E1) = inv2(s,E1) implies inv2(s',E1) .
```

The constants `s, s'` for `System` are also declared.

In the inductive case under consideration, the case is split into six subcases based on the following predicates:

$$\text{bp1} \stackrel{\text{def}}{=} \texttt{c-sdm2(s,p10,r10,m10)}$$
$$\text{bp2} \stackrel{\text{def}}{=} \texttt{(sender(m10) = intruder)}$$
$$\text{bp3} \stackrel{\text{def}}{=} \texttt{(n(p10,intruder,r10) = n)}$$
$$\text{bp4} \stackrel{\text{def}}{=} \texttt{(nonce(cipher1(m10)) = n)}$$
$$\text{bp5} \stackrel{\text{def}}{=} \texttt{(p10 = intruder)}$$

The constants `p10` for `Principal`, `r10` for `Random` and `m10` for `Message` are used as the arguments of `c-sdm2`. Then, the case is split as follows:



Each case is denoted by the predicate obtained by connecting ones appearing in the row with conjunction.

The proof score for subcase 5 is shown.

```
open ISTEP
-- arbitrary objects
  op p10 : -> Principal .  op r10 : -> Random .
  op m10 : -> Message .    op nw10 : -> Network .
-- assumptions
  -- eq c-sdm2(s,p10,r10,m10) = true .
  eq nw(s) = m10 , nw10 .                      eq m1?(m10) = true .
  eq receiver(m10) = p10 .                     eq key(cipher1(m10)) = p10 .
  eq principal(cipher1(m10)) = sender(m10) .   eq r10 \in ur(s) = false .
  --
  eq sender(m10) = intruder .
  eq (n(p10,intruder,r10) = n) = false .
  eq nonce(cipher1(m10)) = n .
  eq (p10 = intruder) = false .
-- successor state
  eq s' = sdm2(s,p10,r10,m10) .
-- check if the predicate is true.
  red inv2(s,cipher1(m10)) implies istep1(n) .
close
```

The condition that there exists a message denoted by `m10` in the network denoted by `nw(s)` is expressed by the equation "`eq nw(s) = m10 , nw10 .`" Except for the conjunct corresponding to this condition, each conjunct in bp1 is expressed by one equation.

For the remaining five subcases, similar proof scores can be written, which do not use any other invariant to strengthen inductive hypotheses. For subcase 6 where the effective condition is false, it is not necessary to write the proof score in theory because nothing changes. But, from an engineering point of view, the proof score is worth writing because the specification may be miswritten.

6 Discussion

If we write proof scores of invariants in CafeOBJ, the compositional proofs of invariants described in Sect. 3 has the following advantages:

1. CafeOBJ reduces a logical formula into an exclusive-or normal form à la Hsiang[15], of which response time crucially depends on the length of the formula, essentially the possible number of `or`'s in it. The compositional proofs of invariants make it possible to focus on each conjunct $pred_i(s, \boldsymbol{x}_i)\,(i = 1, \ldots, n)$ of a large formula $pred(s, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ and relieve the complexity of the reduction.
2. Since we prove each conjunct of a large formula individually, case analyses can be done w.r.t. this conjunct only. This can ease the complexity of case analyses. Suppose that we have to consider N_i subcases for $pred_i(s, \boldsymbol{x}_i)$ and N_j subcases for $pred_j(s, \boldsymbol{x}_j)$ in an inductive case. If we try to prove $pred_i(s, \boldsymbol{x}_i) \wedge pred_j(s, \boldsymbol{x}_j)$ together, then we may have to consider $N_i \times N_j$ subcases in the inductive case.

It is often the case that a large formula is divided into smaller ones to ease the proof. In our method, however, any two of such smaller formulas may depend on each other in the sense that the proof of one uses the other to strengthen inductive hypotheses in inductive cases and vice versa. Existing proof assistants such as Isabelle/HOL[16] may not allow to divide a formula into such two subformulas.

It is significant to split the case into multiple subcases in an inductive case and find another inductive hypothesis (or a lemma) to strengthen the direct one of what being proved in order to make progress on a proof. The former can be done based on predicates used in a specification such as `_=_` and `_\in_` in the NSLPK protocol. If you encounter a subcase where the formula stating a property under discussion is reduced to `false`, the subcase may be unreachable and you may conjecture another invariant.

Although the OTS/CafeOBJ method is not specific to object-orientation, it can be easily applied to object-based distributed systems by modeling an object as an OTS. In the behavioral specification in CafeOBJ, modeling and verification techniques that are specific to object-orientation have been studied[17], which can be incorporated in the OTS/CafeOBJ method.

In addition to the proof that nonces are really secret in the NSLPK protocol, we have proved that the protocol has one-to-many agreement property[18], which is expressed as the following two invariants:

$$
\begin{aligned}
&\mathsf{invariant}\ \ (\mathtt{not}(p = \mathtt{intruder})\ \mathtt{and}\\
&\qquad \mathtt{m1}(p, p, q, \mathtt{enc1}(q, \mathtt{n}(p, q, r), p))\ \mathtt{\backslash in}\ \mathtt{nw}(s)\ \mathtt{and}\\
&\qquad \mathtt{m2}(q1, q, p, \mathtt{enc2}(p, \mathtt{n}(p, q, r), n, q))\ \mathtt{\backslash in}\ \mathtt{nw}(s)\\
&\qquad \mathtt{implies}\ \mathtt{m2}(q, q, p, \mathtt{enc2}(p, \mathtt{n}(p, q, r), n, q))\ \mathtt{\backslash in}\ \mathtt{nw}(s))\,,\\
&\mathsf{invariant}\ \ (\mathtt{not}(q = \mathtt{intruder})\ \mathtt{and}\\
&\qquad \mathtt{m2}(q, q, p, \mathtt{enc2}(p, n, \mathtt{n}(q, p, r), q))\ \mathtt{\backslash in}\ \mathtt{nw}(s)\ \mathtt{and}\\
&\qquad \mathtt{m3}(p1, p, q, \mathtt{enc3}(q, \mathtt{n}(q, p, r)))\ \mathtt{\backslash in}\ \mathtt{nw}(s)\\
&\qquad \mathtt{implies}\ \mathtt{m3}(p, p, q, \mathtt{enc3}(q, \mathtt{n}(q, p, r)))\ \mathtt{\backslash in}\ \mathtt{nw}(s))\,.
\end{aligned}
$$

The former describes that if a principal p has sent a Msg1 to a principal q and has received a Msg2 in response to the Msg1 seemingly from q, although the actual sender $q1$ might be the intruder, then the Msg2 originates from q, and

the latter describes a similar phenomenon. We need (6) and eight more invariants to prove these two invariants. The proof has been done by writing proof scores in CafeOBJ likewise.

The modeling and proof scores described in [19] are different than those described in this paper.

References

1. Gries, D., Schneider, F.B.: A Logical Approach to Discrete Math. Texts and Monographs in Computer Science. Springer, NY (1993)
2. Goguen, J., Malcolm, G., eds.: Software Engineering with OBJ: Algebraic Specification in Action. Kluwer Academic Publishers (2000)
3. Ogata, K., Futatsugi, K.: Formally modeling and verifying Ricart&Agrawala distributed mutual exclusion algorithm. In: APAQS '01, IEEE CS Press (2001) 357–366
4. Ogata, K., Futatsugi, K.: Formal analysis of Suzuki&Kasami distributed mutual exclusion algorithm. In: FMOODS '02, Kluwer Academic Publishers (2002) 181–195
5. Ogata, K., Futatsugi, K.: Formal analysis of the *i*KP electronic payment protocols. In: ISSS 2002. Volume 2609 of LNCS., Springer (2003) 441–460
6. Ogata, K., Futatsugi, K.: Formal verification of the Horn-Preneel micropayment protocol. In: VMCAI 2003. Volume 2575 of LNCS., Springer (2003) 238–252
7. CafeOBJ: CafeOBJ web page. `http://www.ldl.jaist.ac.jp/cafeobj/` (2001)
8. Diaconescu, R., Futatsugi, K.: CafeOBJ report. AMAST Series in Computing, 6. World Scientific, Singapore (1998)
9. Diaconescu, R., Futatsugi, K.: Behavioural coherence in object-oriented algebraic specification. Journal of Universal Computer Science **6** (2000) 74–96
10. Lowe, G.: An attack on the Needham-Schroeder public-key authentication protocol. Inf. Process. Lett. **56** (1995) 131–133
11. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In: TACAS '96. LNCS 1055, Springer (1996) 147–166
12. Chandy, K.M., Misra, J.: Parallel Program Design: A Foundation. Addison-Wesley, Reading, MA (1988)
13. Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Comm. ACM **21** (1978) 993–999
14. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Trans. Inform. Theory **IT-29** (1983) 198–208
15. Hsiang, J.: Refutational Theorem Proving using Term Rewriting Systems. PhD thesis, University of Illinois at Champaign-Urbana (1981)
16. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL – A Proof Assistant for Higer-Order Logic. LNCS 2283. Springer (2002)
17. Diaconescu, R., Futatsugi, K., Iida, S.: Component-based algebraic specification and verification in CafeOBJ. In: World Congress on Formal Methods. Volume 1709 of LNCS., Springer (1999) 1644–1663
18. Lowe, G.: A hierarchy of authentication specifications. In: 10th IEEE Computer Security Foundations Workshop, IEEE CS Press (1997) 31–43
19. Ogata, K., Futatsugi, K.: Rewriting-based verification of authentication protocols. In: WRLA '02. Volume 71 of ENTCS., Elsevier Science Publishers (2002)