International Journal of Software Engineering and Knowledge Engineering

Proof Score Approach to Analysis of Electronic Commerce Protocols

Kazuhiro Ogata*

School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan
ogata@jaist.ac.jp

Kokichi Futatsugi

School of Information Science
Japan Advanced Institute of Science and Technology (JAIST),
1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan
kokichi@jaist.ac.jp

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Proof scores are documents of comprehensible plans to prove theorems. The proof score approach to systems analysis is a method in which proof scores are used to verify that systems enjoy properties (or analyze systems). In this paper, we describe a way to analyze electronic commerce protocols with the proof score approach, which has been developed and refined through several case studies conducted.

Keywords: Algebraic specification; formal methods; security; rewriting; specification; verification.

1. Introduction

The advance of network technology and cryptography have made it possible to conduct commercial transactions electronically. Many electronic commerce (e-commerce) protocols have been then proposed, among which are *i*KP[4,3], SET[37], NetBill[8], Horn-Preneel[26], TLS[12] and Mondex[a]. E-commerce protocols can be considered one of the key infrastructures in the modern and future advanced information society. On the other hand, e-commerce protocols are subject to subtle flaws, which can be difficult to reveal by testing, like other security protocols such as the NSPK authentication protocol[44] whose flaw has been reported by Lowe[33,34]. Therefore, such protocols are worth analyzing formally. Moreover, formal analyzes should

*He was also with NEC Software Hokuriku, Ltd. (1 Anyoji, Hakusan, Ishikawa 920-2141, Japan) when earlier versions of the paper were written.
[a]`http://www.mondex.com/`

be supported by tools so as to avoid as many human errors as possible.

There are mainly two kinds of tools available in order to formally analyze systems including e-commerce protocols: model checkers[38,15] and interactive theorem provers[45,5,20]. Model checkers and interactive theorem provers are complementary in the following sense: (1) Model checkers can verify automatically that systems enjoy properties, provided that systems have bounded number of (reachable) states; (2) Model checkers can provide automatically counterexamples when systems whose (reachable) state spaces are bounded do not enjoy properties; (3) Interactive theorem provers can verify that systems whose reachable state spaces are even unbounded enjoy properties, although computer-human interaction is basically needed; (4) Interactive theorem provers can help humans understand systems more profoundly by revealing hidden facts (lemmas) of the systems. Moreover, dedicated security protocol analysis tools have been proposed[35,39,28,61], which are equipped with specification languages in which security protocols can be straightforwardly written and translators that takes abstract descriptions of security protocols written in such specification languages and generate those that can be dealt with by model checkers and/or theorem provers.

Proof scores are documents of comprehensible plans to prove theorems. The proof score approach to systems analysis is a method in which proof scores are used to verify that systems enjoy properties (or analyze systems). The proof score approach to systems analysis has been mainly devoted by researchers in the OBJ community[22,20]. In the approach, algebraic specification language processors are used as interactive theorem provers. Roughly speaking, systems are analyzed as follows: (1) Write a system in an algebraic specification language L as an algebraic specification S, which is basically a set of equations; (2) Write properties in L; Let P be the set of such properties and let P' be the empty set; (3) If P is empty, the analysis has been successfully finished, which means that the system enjoys all the properties in P'; Otherwise, extract a property p from P and go next; (4) Write a proof plan called a *proof score* in L to prove that p holds for S, which involves case analyzes and/or lemma discoveries; (5) Execute (or play) the proof score with an L's processor based on equational reasoning using rewriting; If results are as expected for all cases, which means that p holds for S, provided that all lemmas used are proved, then put p into P', put lemmas used that are not in P' into P and go to (3); Otherwise, do further case analyzes and/or discover/use other lemmas to rewrite the proof score, and go to (5); a counterexample may be found in this stage.

Compared to systems analysis using other existing interactive theorem provers such as Isabelle/HOL[45] and Coq[5], the proof score approach to systems analysis has some advantages[18]: (1) balanced human-computer interaction and (2) flexible but clear structure of proof scores. The former means that humans are able to focus on proof plans, while tedious and detailed computations can be left to computers; humans do not necessarily have to know what deductive rules or equations should be applied to goals to prove. The latter means that lemmas do not need to be proved in advance and proof scores can help humans comprehend the corresponding

proofs; a proof that a property holds for a system can be conducted even when all lemmas used have not been proved, and results obtained by case analyzes and lemmas discovered are explicitly and clearly written in proof scores. On the other hand, the proof score approach to systems analysis is less rigorous than systems analysis using other existing interactive theorem provers such as Isabelle/HOL and Coq. Proofs constructed under the support of such an interactive theorem prover are basically guaranteed to be correct, while humans might overlook some cases to consider and/or proofs of some lemmas in the proof score approach because there are no tools available to check such oversights. We have been developing some tools[60,41,42,43] to make the proof score approach more rigorous.

We have conducted case studies[47,50,51,48,52], with the proof score approach, in which we have formally analyzed *i*KP[4,3], SET[37], NetBill[8], Horn-Preneel[26], TLS[12] and Mondex[30]. Precisely, the OTS/CafeOBJ method[49,11,53], which is an instance of the proof score approach to systems analysis, has been used. In the OTS/CafeOBJ method, observational transition systems (OTSs) are used as models of systems and CafeOBJ[9], an algebraic specification language, is used; OTSs are transition systems, which are straightforwardly written in equations. CafeOBJ is equipped with its processor called the CafeOBJ system, which is used as an interactive theorem prover. In one case study[47] with the OTS/CafeOBJ method, we have found out a counterexample showing that 2KP and 3KP do not enjoy a property, which is called Payment Agreement, and proposed one possible modification that lets 2KP and 3KP enjoy the property[46].

What can be done for e-commerce protocols with the method describe in this paper is to prove that such protocols whose reachable state spaces are even unbounded enjoy security properties expressed as invariant properties on the assumption that there exist malicious principals. Thanks to our way of modeling messages and networks, we can express various properties such that some events always precede others in terms of invariant properties only.

In this paper, we describe a way to analyze e-commerce protocols with the OTS/CafeOBJ method, which has been developed and refined through our experiences. A modified version of (simplified) 3KP[46] is used as an example. The protocol is called AM3KP. Payment Agreement[46] is taken into account to describe how to express security properties and how to write proof scores to verify that security properties hold for e-commerce protocols.

1.1. *Related Work*

We mention some related work: (1) analyses of e-commerce protocols with model checkers, (2) analyses of e-commerce protocols with interactive theorem provers, (3) dedicated security protocol analysis tools, and (4) Mondex case studies.

Analyses with Model Checkers

Lu and Smolka[36] analyze a system consisting of two cardholders, one merchant and one payment gateway that perform payment transactions according to SET with the FDR model checker. One protocol run is considered. One of the cardholders has a legitimate credit card, and the other does not. Both cardholders may try to pay less than the amount agreed on previously with the merchant. The merchant may try to overcharge a cardholder. The payment gateway is trustable. Five properties are checked to the system with FDR.

Heintze, Tygar, Wing and Wong[24] analyze NetBill and a digital cash protocol with FDR concerning money atomicity and goods atomicity. Money atomicity means that money should neither be created nor destroyed by e-commerce protocols, and goods atomicity that a merchant should receive payment if and only if the consumer receives the goods. For each of the protocols, a finite model consisting one consumer, one merchant and one bank is made and one protocol run is considered. FDR is used to check if the models satisfy money atomicity and goods atomicity. As the results of the analyses, NetBill satisfies both properties, while the digital cash protocol does neither property.

Mitchell, Shmatikov and Sternet[40] use the Murφ model checker to check seven simple protocols derived from the SSL 3.0 handshake protocol. The primal reason why they have analyzed the protocols is to identify the purpose of certain message fields (version number, nonce, etc.) in some steps of the protocol. The analysis starts with the simplest version of the handshake protocol from which some fields are omitted, and the fields are gradually added to the protocol. The first six protocols have been found badly flawed and the model checker has found many attacks. The model checker has been used to check the final protocol with two clients, one server, no more than two simultaneous open sessions per server and no more than one resumption per session, and no attacks have been discovered.

Analyses with Interactive Theorem Provers

Bolignano[6] proposes a way of expressing and verifying properties of e-commerce protocols such as SET. Bolignano claims that some properties relevant to those protocols cannot be directly expressed by simple invariants and should rely on the history of each protocol run. Such properties include one that a principal can be sure as the time when the principal receives a message that this message really originates from some intended principal and has not been tempered with. Hence such a property is expressed by a pair of a regular language or a finite automaton L and a filtering function $ff_x(y)$ where x is used to parameterize the function and y ranges in the domain of actions corresponding to transmission and receipt of messages. L is used to observe that any finite protocol run follows the property, and $ff_x(y)$ selects actions relevant to the property from the protocol run. A property expressed by L and $ff_x(y)$ is satisfied if and only if for any finite trace t and for any x, $ff_x(y) \in L$. The proof can be reduced to that of a simple invariant property,

which is supported by Coq.

Paulson's inductive method[57] is used to analyze Cardholder Registration phase of SET and the SET purchase protocols (the simplified and combined Payment Authorization with Purchase Request). The inductive method models a system in which an arbitrary number of principals including the intruder take part in a protocol by inductively defining traces generated by a set of rules that correspond to the possible actions of the principals including the intruder. Security properties can be stated as predicates over the traces. It can be inductively proved that a certain property holds of all possible traces for a certain protocol. The proof can be supported by Isabelle/HOL. On Cardholder Registration phase of SET, it is verified that if a trusted certificate authority (CA) keeps track of the registered keys, the protocol is robust enough to guarantee that two different agents will never get the same key certified by the same CA; however, different agents may collude and register the same key with different CAs[2]. On the SET purchase protocols, it is verified that the protocols enjoy several desirable properties[1]. Besides, it is found that the Cardholder and Payment Gateway cannot agree on the latter's identity, giving rise to potential vulnerabilities. A possible modification is proposed.

Paulson[58] analyzes the TLS handshake protocol with his inductive method. In the protocol analyzed by Paulson, servers always sent their certificates to clients, the key exchange method considered is RSA, and clients optionally send their certificates and ClientKeyExchange messages to servers. In his model of the protocol, a malicious principal called the spy is taken into account and it is supposed that any session key, if used, may end up in the hands of the spy, which is denoted by the rule *Oops*. One of the results of the analysis is that session resumption turns out to be safe even if the spy has obtained session keys from earlier sessions.

Dedicated Security Protocol Analysis Tools

Casper[35] takes an abstract description of a security protocol and a property to be checked, and produces CSP processes that can be analyzed by the FDR model checker. Since FDR can deal with finite-state CSP processes only, users need to feed the number of principals involved, nonces used, etc. into Casper. The main purpose of Casper is to find security flaws lurked in security protocols with model checking, and the main target is authentication protocols.

CAPSL[39] is a specification language for security protocols. A specification of a security protocol and some properties written in CAPSL is translated into another description written in CIL, which is the CAPSL intermediate language. Protocol descriptions in CIL are state transition representations, which can be straightforwardly translated into those that can be directly dealt with by some generic analysis tool. Although any generic analysis tool can be used, Maude is mainly concerned. The main purpose of CAPSL is the same as that of Casper, and the main target is authentication protocols.

CASRUL[28] has a similar flavor of Casper, but produces security protocol de-

scriptions that can be fed into the theorem prover daTac, which is based on first order deduction modulo associativity and commutativity axioms. Although CASRUL could be used to prove that some security properties hold for security protocols that have unbounded number of states because daTac is a theorem prover, the paper[28] only describes how to find security flaws lurked in security protocols, and the main target is authentication protocols.

AVISPA[61] is equipped with High-Level Protocol Specification Language (HLPSL) and Intermediate Format (IF), which correspond to CAPSL and CIL, respectively. Abstract descriptions of security protocols in HLPSL are translated into those written in IF, which are then converted into those that can be directly analyzed by one of the four analysis tools, which are (1) an on-the-fly model checker, (2) a CL-based attack searcher, (3) a SAT-based model checker, and (4) a tree-automata-based protocol analyzer. Although a tree-automata-based protocol analyzer can be used to prove that some security properties hold for security protocols that have unbounded number of states, the main purpose seems to find security flaws lurked in security protocols. Unlike Casper, CAPSL and CASRUL, however, AVISPA has been used to analyze SET.

Mondex Case Studies

Mondex case studies[29] have been conducted as part of the world-wide Grand Challenge in Verified Software[25]. Mondex was originally analyzed as follows[62]: (1) three models (abstract, intermediate and concrete models) of Mondex were described in the Z notation, and (2) it was proved that the intermediate model is a refinement of the abstract model and the concrete model is a refinement of the intermediate model by hand. Several research groups have mechanized the proof of correctness of Mondex with different formal methods and tools.

Freitas and Woodcock[17] have mechanized the proof with the Z/Eves theorem prover.

George and Haxthausen[19] have mechanized the proof by translating models written in the RAISE specification language (RSL) into those for the PVS theorem provers. They have also translated models in RSL into those for the SAL model checkers.

Butler and Yadav[7] have used Event-B to make 10 models of Mondex and performed nine refinement proofs with its associated proof tools. The reason why they have made more than three models is because the gap between two consecutive models can be smaller, making refinement proofs easier.

Ramananandro[59] has used Alloy to model check the refinement steps.

The Mondex case studies mentioned above do not suppose that there exist malicious principals but just assume that messages exchanged are protected by some appropriate cryptographic mechanisms. Haneberg, Schellhorn, Grandy and Reif[23] have made models of Mondex as abstract state machines (ASMs) and performed the refinement proofs with the KIV theorem provers. Unlike the other Mondex case

studies, they have taken into account malicious principals.

Kuhlmann and Gogolla[32] have used UML and OCL to validate the abstract model of Mondex using both positive and negative test cases.

All the Mondex case studies except for the final one, which does not use any formal verification techniques, make multiple different models of Mondex and use refinement techniques to prove that Mondex enjoys some security properties. On the other hand, the authors' group[30] makes one model of Mondex and proves that Mondex enjoys some security properties with the method described in this paper.

1.2. *Outline of the Paper*

Section 2 introduces the OTS/CafeOBJ method. Section 3 gives a description of AM3KP and Payment Agreement. Section 4 describes our way to model and specify e-commerce protocols; AM3KP is used as an example. Section 5 describes our way of expressing security properties; Payment Agreement is used as an example. Section 6 describes our way of writing proof scores; part of the proof scores to verify that Payment Agreement holds for AM3KP are used as examples. Section 7 concludes the paper, mentioning some future issues as well.

2. The OTS/CafeOBJ Method

In this section, we describe CafeOBJ, observational transition systems (OTSs), and how to specify OTSs in CafeOBJ. We will describe how to write proof scores of invariant properties in CafeOBJ in Subsection 6.1.

2.1. *CafeOBJ*

CafeOBJ[b][9] is based on three logical foundations: (1) *order-sorted algebras*[20], (2) *hidden algebras*[10,21], and (3) *preorder algebras* (which are similar to rewriting logic on which Maude, a sibling language of CafeOBJ, is based). The OTS/CafeOBJ method uses the first two: order-sorted algebras and hidden algebras. Abstract machines as well as abstract data types can be specified in CafeOBJ. There are two kinds of sorts in CafeOBJ, which are *visible* and *hidden sorts*. A visible sort denotes an abstract data type, while a hidden sort denotes the state space of an abstract machine. There are three kinds of operators (or operations) with respect to hidden sorts, which are *hidden constants*, *action operators* and *observation operators*. Hidden constants can be used to denote initial states of abstract machines, action operators denote state transitions of abstract machines, and observation operators let us know the situation where abstract machines are located. A hidden constant takes zero or more (typically zero) data and returns a state. Both an action operator and an observation operator take a state of an abstract machine and zero or more data. The action operator returns the successor state of the state with respect to

[b] `http://www.ldl.jaist.ac.jp/cafeobj/`

the state transition denoted by the action operator plus the data. The observation operator returns a value that characterizes the situation where the abstract machine is located.

Visible sorts are declared by enclosing `[` and `]`, and hidden sorts are declared by enclosing `*[` and `]*`. Action and observation operators are declared by starting with `bop`, and other operators including hidden constants are declared by starting with `op`. After `bop` or `op`, an operator name is written, followed by a colon `:` and a list of sorts, and then, `->` and a sort are written. The list of sorts is called the arity of the operator, and the sort after `->` is called the coarity of the operator. The pair of the arity and coarity is called the rank of the operator. When declaring more than one operator whose rank is the same simultaneously, `bops` and `ops` are used instead of `bop` and `op`. Operators with the empty arity are called *constants*.

Properties of operators are specified (or properties are defined) in equations. There are two kinds of equations, which are *conventional* and *behavioral equations*. Both can have conditions. A conventional equation says that two data values are equal, and a behavioral equation says that two states are equal in that any observation returns a same data value in the two states and any sequence of actions preserves it. A nonconditional conventional (behavioral) equation is declared by starting with `eq` (`beq`), and a conditional conventional (behavioral) equation with `ceq` (`cbeq`). After `eq` (`beq`), two terms connected with `=` are written, ended with a full stop. After `ceq` (`cbeq`), two terms connected with `=` are written, followed by `if`, and then, a term denoting the condition and a full stop are written.

The CafeOBJ system uses declared equations as left-to-right rewrite rules and rewrites (or reduces) a given term. The command `red` is used to reduce a given term. This executability makes it possible to simulate a specified system and verify that a specified system enjoys properties.

Basic units of CafeOBJ specifications are modules. The CafeOBJ system provides built-in modules where basic data types such as truth values are specified. The module of truth values is `BOOL`.

Since truth values are indispensable for conditional equations, `BOOL` is automatically imported by almost every module unless otherwise stated. The import of `BOOL` lets us use the visible sort `Bool` denoting truth values, the constants `true` and `false` denoting true and false, and operators denoting some basic logical operators. Among the operators are `not_`, `_and_`, `_or_`, `_xor_`, `_implies_` and `_iff_` denoting negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$), exclusive disjunction (xor), implication ($\Rightarrow$) and logical equivalence ($\Leftrightarrow$), respectively. The operator `if_then_else_fi` corresponding to **if** statements in programming languages is also available. An underscore `_` indicates the place where an argument is put.

`BOOL` plays an essential role in verification with the CafeOBJ system. If the equations available in the module are regarded as left-to-right rewrite rules, they are complete with respect to propositional logic[27]. Therefore, any term denoting a propositional formula that is always true (or false) is surely reduced to `true` (or `false`). Generally, a term denoting a propositional formula is reduced to a term

denoting an exclusively disjunctive normal form of the propositional formula.

2.2. *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 with a subscript such as D_{o1}.

Definition 1. (OTSs) An OTS[49,53] $\mathcal{S}$ is $\langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ such that

- $\mathcal{O}$: A finite set of observers. Each *observer* $o_{x_1:D_{o1},\ldots,x_m:D_{om}} : \Upsilon \rightarrow D_o$ 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* $t_{y_1:D_{t1},\ldots,y_n:D_{tn}} : \Upsilon \rightarrow \Upsilon$ 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 $\mathcal{S}$. Each transition $t_{y_1,\ldots,y_n}$ has the condition $c\text{-}t_{y_1:D_{t1},\ldots,y_n:D_{tn}} : \Upsilon \rightarrow$ Bool, which is called *the effective condition* of the transition. If $c\text{-}t_{y_1,\ldots,y_n}(\upsilon)$ does not hold, then $t_{y_1,\ldots,y_n}(\upsilon) =_{\mathcal{S}} \upsilon$.

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

- Each $\upsilon_{\text{init}} \in \mathcal{I}$ is reachable with respect to $\mathcal{S}$.
- For each $t_{y_1,\ldots,y_n} \in \mathcal{T}$ and each $y_k : D_{tk}$ for $k = 1. \ldots, n$, $t_{x_1,\ldots,x_n}(\upsilon)$ is reachable with respect to $\mathcal{S}$ if $\upsilon \in \Upsilon$ is reachable with respect to $\mathcal{S}$.

Let $\mathcal{R}_{\mathcal{S}}$ be the set of all reachable states with respect to $\mathcal{S}$.

Predicates whose types are $\Upsilon \rightarrow$ Bool are called state predicates. All properties considered in this paper are *invariants*.

Definition 3. (Invariants) Any state predicate $p : \Upsilon \rightarrow$ *Bool* is called *invariant* with respect to $\mathcal{S}$ if p holds in all reachable states with respect to $\mathcal{S}$, i.e. $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$. $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$ may be expressed as $\mathsf{invariant}_{\mathcal{S}}\ p$. $\mathcal{S}$ may be omitted from $\mathsf{invariant}_{\mathcal{S}}\ p$ if it is clear from the context.

We suppose that each state predicate p considered in this paper has the form $\forall z_1 : D_{p1} \ldots \forall z_a : D_{pa}.\, P(\upsilon, z_1, \ldots, z_a)$, where $\upsilon, z_1, \ldots, z_a$ are all variables in p and $P(\upsilon, z_1, \ldots, z_a)$ does not contain any quantifiers. All universal quantifiers may be omitted from $\forall z_1 : D_{p1} \ldots \forall z_a : D_{pa}.\, P(\upsilon, z_1, \ldots, z_a)$ when the state predicate is written.

2.3. *Specifying OTSs in CafeOBJ*

We suppose that a visible sort V_* corresponding to each data type D_* used in OTSs and the related operators are provided. X_k and Y_k are CafeOBJ variables corresponding to indexes x_k and y_k of observers and transitions, respectively.

The universal state space Υ is represented by a hidden sort, say H declared as `*[H]*` by enclosing it with `*[` and `]*`. Given an OTS $\mathcal{S}$, an arbitrary initial state is represented by a hidden constant, say *init*, each observer $o_{x_1,\ldots,x_m}$ is represented by an observation operator, say o, and each transition $t_{y_1,\ldots,y_n}$ is represented by an action operator, say t. The hidden constant *init*, the observation operator o and the action operator t are declared as follows:

`op` *init* : `->` H
`bop` o : H V_{o1} $\ldots$ V_{om} `->` V_o
`bop` t : H V_{t1} $\ldots$ V_{tn} `->`H

We suppose that the value returned by $o_{x_1,\ldots,x_m}$ in an arbitrary initial state can be expressed as $f(x_1,\ldots,x_m)$. This is expressed by the following equation:

`eq` $o(\mathit{init}, X_1,\ldots,X_m) = f(X_1,\ldots,X_m)$.

$f(X_1,\ldots,X_m)$ is the CafeOBJ term corresponding to $f(x_1,\ldots,x_m)$.

Each transition $t_{y_1,\ldots,y_n}$ is defined by describing what the value returned by each observer $o_{x_1,\ldots,x_m}$ in the successor state becomes when $t_{y_1,\ldots,y_n}$ is applied in a state υ. When $c\text{-}t_{y_1,\ldots,y_n}(\upsilon)$ holds, this is expressed generally by a conditional equation that has the form

`ceq` $o(t(S,Y_1,\ldots,Y_n),X_1,\ldots,X_m) = e\text{-}t(S,Y_1,\ldots,Y_n,X_1,\ldots,X_m)$
`if` $c\text{-}t(S,Y_1,\ldots,Y_n)$.

S is a CafeOBJ variable of H, corresponding to υ. $e\text{-}t(S,Y_1,\ldots,Y_n,X_1,\ldots,X_m)$, which does not contain any action operators, is the CafeOBJ term corresponding to the value returned by $o_{x_1,\ldots,x_m}$ in the successor state denoted by $t(S,Y_1,\ldots,Y_n)$. $c\text{-}t(S,Y_1,\ldots,Y_n)$ is the CafeOBJ term corresponding to $c\text{-}t_{y_1,\ldots,y_n}(\upsilon)$.

If $c\text{-}t_{y_1,\ldots,y_n}(\upsilon)$ always holds in any state υ or the value returned by $o_{x_1,\ldots,x_m}$ is not affected by applying $t_{y_1,\ldots,y_n}$ in any state υ (i.e. regardless of the truth value of $c\text{-}t_{y_1,\ldots,y_n}(\upsilon)$), then a usual equation is used instead of a conditional equation. The usual equation has the form

`eq` $o(t(S,Y_1,\ldots,Y_n),X_1,\ldots,X_m) = e\text{-}t(S,Y_1,\ldots,Y_n,X_1,\ldots,X_m)$.

$e\text{-}t(S,Y_1,\ldots,Y_n,X_1,\ldots,X_m)$ is $o(S,X_1,\ldots,X_m)$ if the value returned by $o_{x_1,\ldots,x_m}$ is not affected by applying $t_{y_1,\ldots,y_n}$ in any state.

When $c\text{-}t_{y_1,\ldots,y_n}(\upsilon)$ does not hold, $t_{y_1,\ldots,y_n}$ essentially changes nothing, which is expressed by a conditional equation that has the form

`bceq` $t(S,Y_1,\ldots,Y_n) = S$ `if not` $c\text{-}t(S,Y_1,\ldots,Y_n)$.

3. AM3KP and Payment Agreement

AM3KP[46] is a payment protocol based on the existing credit-card payment system and can be described as follows:

Initiate:	$B \longrightarrow S:$	ID_B
Invoice:	$S \longrightarrow B:$	$\mathrm{Clear}, \mathrm{Sig}_S$
Payment:	$B \longrightarrow S:$	$\mathrm{EncSlip}, \mathrm{Sig}_B$
Auth-Request:	$S \longrightarrow A:$	$\mathrm{Clear}, \mathrm{EncSlip}, \mathrm{Sig2}_S, \mathrm{Sig}_B$
Auth-Response:	$A \longrightarrow S:$	$\mathrm{RCODE}, \mathrm{Sig}_A$

The protocol involves three parties called B (Buyer), S (Seller) and A (Acquirer). We suppose that each principal X has a private key K_X that enables signing and decryption, and its public counterpart K_X^{-1} that enables signature verification and encryption is securely conveyed to every other principal; K_X is known to only X.

Cryptographic primitives used are a one-way hash function $\mathcal{H}(\cdot)$, a keyed one-way hash function $\mathcal{H}_k(K, \cdot)$, a public-key encryption function $\mathcal{E}_X(\cdot)$ with K_X^{-1} and a digitally signing function $\mathcal{S}_X(\cdot)$ with K_X. Basic values used are: R_B (a random number generated by B to form B's pseudo-ID ID_B), BAN (B's account number), and RCODE (a response from the credit card authorization system). Composite values used are:

- $\mathrm{ID}_B : \mathcal{H}_k(\mathrm{R}_B, \mathrm{BAN})$,
- $\mathrm{Common} : S, B, \mathrm{ID}_B$,
- $\mathrm{Clear} : \mathcal{H}(\mathrm{Common})$,
- $\mathrm{SLIP} : \mathcal{H}(\mathrm{Common}), \mathrm{BAN}, \mathrm{R}_B$,
- $\mathrm{EncSlip} : \mathcal{E}_A(\mathrm{SLIP})$,
- $\mathrm{Sig}_A : \mathcal{S}_A(\mathrm{RCODE}, \mathcal{H}(\mathrm{Common}))$,
- $\mathrm{Sig}_S : \mathcal{S}_S(\mathcal{H}(\mathrm{Common}))$,
- $\mathrm{Sig2}_S : \mathcal{S}_S(\mathcal{H}(\mathrm{Common}), \mathrm{EncSlip})$, and
- $\mathrm{Sig}_B : \mathcal{S}_B(\mathrm{EncSlip}, \mathcal{H}(\mathrm{Common}))$

To initiate a protocol run, B generates R_B, computes ID_B and sends ID_B to S as an Initiate message. On receipt of the Initiate message, S makes Clear and Sig_S, and sends them to B as an Invoice message. On receipt of the Invoice message, B retrieves a hash value and a digital signature from the message. B checks if the hash value equals $\mathcal{H}(\mathrm{Common})$ and verifies the digital signature with K_S^{-1}. When successful, B constructs EncSlip and Sig_B, and sends them to S as a Payment message. On receipt of the Payment message, S retrieves a ciphertext and a digital signature, and verifies the digital signature with K_B^{-1}. When successful, S computes $\mathrm{Sig2}_S$ and sends Clear that has been sent as part of the Invoice message in this protocol run, the ciphertext, $\mathrm{Sig2}_S$ and the digital signature to A as an Auth-Request message. On receipt of the Auth-Request message, A retrieves a hash value h_1, a ciphertext and two digital signatures. A tries to decrypt the ciphertext with K_A. When successful, A obtains a hash value h_2, an account number and a random number, and builds $\mathcal{H}(\mathrm{Common})$

using the account number, the random number, S and B. A checks if both h_1 and h_2 equal $\mathcal{H}$(Common) and verifies the two digital signatures with K_S^{-1} and K_B^{-1}. When successful, A forwards the account number and B to the credit card authorization system so as to obtain on-line authorization of this payment. On receipt of RCODE from the authorization system, A computes Sig_A, and sends RCODE and Sig_A to S as an Auth-Response message. On receipt of the Auth-Response message, S retrieves a value and a digital signature, and verifies the digital signature with K_A^{-1}. When successful, the value lets S know whether the payment has been authorized. S may forward the value and the digital signature to B.

For AM3KP, we discuss the property called *Payment Agreement*[46]:

> If an acquirer authorizes a payment, then both the buyer and seller concerned always agree on it.

In AM3KP, that an acquirer authorizes a payment implies that the acquirer receives the valid Auth-Request message with respect to the payment. That the buyer and seller concerned agree on the payment is that they have sent the Initiate and Payment messages, and the Invoice and Auth-Request messages corresponding to the valid Auth-Request message, respectively. Therefore Payment Agreement can be rephrased as follows:

> If an acquirer receives a valid Auth-Request message stating that a buyer pays a seller some amount, no matter who has sent the valid Auth-Request message, then the buyer has always sent the Initiate and Payment messages corresponding to the valid Auth-Request message to the seller, and the seller has always sent the Invoice and Auth-Request messages corresponding to the valid Auth-Request message to the buyer and the acquirer, respectively.

4. Modeling and Specification of E-Commerce Protocols

4.1. *Assumptions*

We suppose that the cryptosystem used is perfect. We also suppose that there not only exist trustable principals but also malicious (untrustable) principals. Trustable principals exactly follow the protocol, while malicious principals may do something against the protocol as well, namely eavesdropping on and/or faking messages so as to attack and/or confuse the protocol. Instead of describing each of the malicious principals, however, the combination and cooperation of the malicious principals is modeled as the most general intruder à la Dolev and Yao[14]. The intruder can do the following:

- Eavesdrop on every message flowing in the network,
- Glean as much information as possible from the message, and
- Fake and send messages based on the gleaned information.

But, the intruder cannot break the perfect cryptosystem such that he/she cannot decrypt a ciphertext unless he/she knows the key to decrypt, cannot make a digital signature unless he/she knows the key to sign, and cannot predict unknown values such as random numbers. Since properties discussed are only invariants that are a subclass of safety properties, we do not take interception of messages into account. The intruder also acts as a trustable principal.

For AM3KP, we also suppose that there exists one and only one legitimate (trustable) acquirer and the legitimate acquirer is known to every other principal.

4.2. *Formalization of Basic Data Types*

Basic data types used in protocols such as random numbers and ciphertexts are formalized in terms of order-sorted algebras. Basic data types are denoted by visible sorts; data constructors and functions of such types are denoted by operators; values of such types are denoted by terms whose sorts correspond to the types.

AM3KP uses the 17 basic data types that correspond to BANs, RCODEs, public keys, private keys, buyers, sellers, acquirers, random numbers, keyed hashes of BANs, hashes of Commons, Commons, Clears, SLIPs, instances of Sig_A, instances of Sig_S, instances of $\mathrm{Sig2}_S$, and instances of Sig_B, which are denoted by the visible sorts `Ban`, `Rcode`, `Pkey`, `Skey`, `Buyer`, `Seller`, `Acquirer`, `Rand`, `Hban`, `Hcom`, `Common`, `Clear`, `Slip`, `Eslp`, `Siga`, `Sigs`, `Sigs2`, and `Sigb`, respectively. We use operators to denote data constructors and functions of those data types; the operators are as follows:

- Given a BAN n, the term $\mathtt{b}(n)$ denotes the buyer identified by n, and given a buyer b, $\mathtt{ban}(b)$ denotes his/her BAN. Given a buyer, a seller or an acquirer x, $\mathtt{pk}(x)$ and $\mathtt{sk}(x)$ denote his/her public key and private key. The constants `ib`, `is` and `ia` whose sorts are `Buyer`, `Seller` and `Acquirer` denote the intruder that acts as a buyer, a seller and an acquirer, respectively. Although we suppose that there exists one and only one legitimate acquirer, the constant `ia` is used to fake messages seemingly sent by the legitimate acquirer. The constant `la` of `Acquirer` denotes the legitimate acquirer; `la` does not equal `ia`.
- Given a buyer b and a random number r, $\mathtt{nxt}(b, r)$ denotes a random number generated by the buyer b, which is different from those (including r) that have been generated so far, and $\mathtt{gtr}(r)$ returns the buyer who has generated r.
- Given a Common c, $\mathtt{h}(c)$ denotes the hash of c. Given a random number r and a BAN n, $\mathtt{h}(r, n)$ denotes the keyed hash of n with r, and given a hashed BAN hn, $\mathtt{r}(hn)$ returns the random number used to compute hn.
- Given a seller s, a buyer b and a hashed BAN hn, $\mathtt{com}(s, b, hn)$ denotes the Common consisting of s, b and hn. Given a Common c, $\mathtt{s}(c)$, $\mathtt{b}(c)$ and $\mathtt{hban}(c)$ return the seller, the buyer and the hashed BAN that constitute c.
- Given a hashed Common hc, $\mathtt{cl}(hc)$ denotes the Clear that consists of hc,

and given a Clear *cl*, `hcom`(*cl*) returns the hashed Common that constitutes *cl*.

- Given a hashed Common *hc*, a BAN *n* and a random number *r*, `slp`(*hc*, *n*, *r*) denotes the SLIP that consists of *hc*, *n* and *r*. Given a SLIP *slp*, `hcom`(*slp*), `ban`(*slp*) and `rand`(*slp*) return the hashed Common, the BAN and the random number that constitute *slp*.
- Given a public key *pk* and a SLIP *slp*, `enc`(*pk*, *slp*) denotes the EncSlip that is *slp* encrypted with *pk*. Given an EncSlip *e*, `pk`(*e*) and `slip`(*e*) returns the public key and the SLIP used to compute *e*.
- Given a private key *sk*, an RCODE *rc* and a hashed Common *hc*, `sig`(*sk*, *rc*, *hc*) denotes the instance of Sig_A that is the digital signature of *rc* and *hc* computed with *sk*. Given an instance *ga* of Sig_A, `sk`(*ga*), `rc`(*ga*) and `hc`(*ga*) return the private key, the RCODE and the hashed Common used to compute *ga*.
- Given a private key *sk* and a hashed Common *hc*, `sig`(*sk*, *hc*) denotes the instance of Sig_S that is the digital signature of *hc* computed with *sk*. Given an instance *gs* of Sig_S, `sk`(*gs*) and `hc`(*gs*) return the private key and the hashed Common used to compute *gs*.
- Given a private key *sk*, a hashed Common *hc* and an EncSlip *e*, `sig`(*sk*, *hc*, *e*) denotes the instance of $\mathrm{Sig2}_S$ that is the digital signature of *hc* and *e* computed with *sk*. Given an instance *gs2* of $\mathrm{Sig2}_S$, `sk`(*gs2*), `hc`(*gs2*) and `es`(*gs2*) return the private key, the hashed Common and the EncSlip used to compute *gs2*.
- Given a private key *sk*, an EncSlip *e* and a hashed Common *hc*, `sig`(*sk*, *e*, *hc*) denotes the instance of Sig_B that is the digital signature of *e* and *hc* computed with *sk*. Given an instance *gb* of Sig_B, `sk`(*gb*), `es`(*gb*) and `hc`(*gb*) return the private key, the EncSlip and the hashed Common used to compute *gb*.

For each of the 17 visible sorts, the binary operator `_=_` is used, which checks if two terms of the visible sort denote the same value.

4.3. *Formalization of Messages*

Messages are denoted by a visible sort, say `Msg`. Each kind of message is denoted by an operator *msg*, which has at least three arguments. The declaration of the operator `msg` looks like:

`op` *msg* `:` $P_1\ P_1\ P_2\ D_1\ \ldots D_n$ `->` `Msg`

where P_i for $i = 1, 2$ is a visible sort denoting a class of principals, and D_i for $i = 1, \ldots, n$ is a visible sort denoting a data type. The fourth and the later arguments denote the body of the corresponding message. The first, second and third arguments mean the actual sender, the seeming source and the destination 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. The first argument cannot be forged by the intruder, while the remaining arguments may be forged by the intruder. Therefore, assuming that there exists a message denoted by the term $msg(p_1', p_1, p_2, d_1, \ldots, d_n)$, we can deduce the following: (1) It is true that the principal p_1' has sent the message; (2) If p_1' is trustable, p_1 equals p_1'; (3) If p_1' is the intruder, p_1 may not equal p_1', which means that the intruder has faked the message; (4) If p_1 is the intruder, p_1' is also the intruder; (5) If p_1' does not equal p_1, p_1' is the intruder.

For AM3KP, the five operators are used to denote the five kinds of messages. The operators are declared as follows:

```
op im : Buyer    Buyer    Seller   Hban       -> Msg
op vm : Seller   Seller   Buyer    Clear Sigs -> Msg
op pm : Buyer    Buyer    Seller   Eslp  Sigb -> Msg
op qm : Seller   Seller   Acquirer Clear Eslp
                                   Sigs2 Sigb -> Msg
op sm : Acquirer Acquirer Seller   Rcode Siga -> Msg
```

`im` stands for Initiate messages, `vm` for Invoice messages, `pm` for Payment messages, `qm` for Auth-Request messages and `sm` for Auth-Response messages. For each x`m` of the five operators, we have the operators x`c`, x`s` and x`d` that return the first, second and third arguments of a given term whose top is x`m`, respectively, where $x = $ `i`, `v`, `p`, `q`, `s`. We have the operators `hban`, `clear`, `eslip`, `rcode`, `siga`, `sigs`, `sigs2` and `sigb` that return the hashed BAN, the Clear, the EncSlip, the RCODE, the instance of Sig_A, the instance of Sig_S, the instance of $\mathrm{Sig2}_S$ and the instance of Sig_B in a given message, respectively, if any. We also have the operator x`m`? that checks if a given message is x`m` message, namely a message denoted by a term whose top is x`m`, where $x = $ `i`, `v`, `p`, `q`, `s`. Moreover, we have the binary operator `_=_` that checks if two terms of `Msg` denote the same message.

4.4. *Formalization of the Network*

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, whose first arguments denote 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 as much information as possible from the network. For each D of the data types whose values are gleaned by the intruder from the network, an operator, say *vals*, is used to denote the collection of values whose

types are D. Let *Val* be the visible sort denoting D. The operator *vals* is declared as follows:

`op` *vals* `: Network ->` *ColVals*

where `Network` is the visible sort denoting networks and *ColVals* is the visible sort denoting collections of values whose types are D. Given a snapshot *nw* of the network, the term *vals*(*nw*) denotes the collection of the values (whose types are D) gleaned by the intruder from *nw*. What values are in the collection denoted by *vals*(*nw*) is defined in terms of equations.

For AM3KP, the intruder tries to glean 10 kinds of values from the network. The 10 kinds of values are hashed Commons, hashed BANs, BANs, random numbers, EncSlips, RCODEs, instances of Sig_A, instances of Sig_S, instances of $\mathrm{Sig2}_S$ and instances of Sig_B. The collections of those values gleaned by the intruder from the network are denoted by the following operators respectively:

```
op hcoms  : Network -> ColHcoms
op hbans  : Network -> ColHbans
op bans   : Network -> ColBans
op rands  : Network -> ColRands
op eslps  : Network -> ColEslps
op rcodes : Network -> ColRcodes
op sigas  : Network -> ColSigas
op sigss  : Network -> ColSigss
op sigs2s : Network -> ColSigs2s
op sigbs  : Network -> ColSigbs
```

The set of equations that define `hcoms` is as follows:

```
eq  HC \in hcoms(void) = false .
ceq HC \in hcoms(M,NW) = true
    if vm?(M) and HC = hcom(clear(M)) .
ceq HC \in hcoms(M,NW) = true
    if pm?(M) and pk(ia) = pk(eslip(M))
              and HC = hcom(slip(eslip(M))) .
ceq HC \in hcoms(M,NW) = true
    if qm?(M) and HC = hcom(clear(M)) .
ceq HC \in hcoms(M,NW) = true
    if qm?(M) and pk(ia) = pk(eslip(M))
              and HC = hcom(slip(eslip(M))) .
ceq HC \in hcoms(M,NW) = HC \in hcoms(NW)
    if not(vm?(M) and HC = hcom(clear(M))) and
       not(pm?(M) and pk(ia) = pk(eslip(M))
                  and HC = hcom(slip(eslip(M))))
       and not(qm?(M) and HC = hcom(clear(M))) and
```

```
       not(qm?(M) and pk(ia) = pk(eslip(M))
                  and HC = hcom(slip(eslip(M)))) .
```

`M` and `NW` are CafeOBJ variables whose sorts are `Msg` and `Network`. The constant `void` denotes the empty bag, and the operator `_,_` in `M,NW` is the data constructor of nonempty bags. The operator `_\in_` is the membership predicate of collections. If the network is empty, there are no hashed Commons available to the intruder, which is denoted by the first equation. If there exists a message that includes a hashed Common in the network, the hashed Common could be gleaned by the intruder, provided that if the hashed Common is part of a ciphertext, the intruder must know the key to decrypt the ciphertext. Such messages are Invoice, Payment and Auth-Request messages. If there exists an Invoice message in the network, the hashed Common in the message is available to the intruder, which is denoted by the second equation. If there exists a Payment message in the network and the EncSlip in the message is encrypted with the intruder's public key, the hashed Common in the message is available to the intruder, which is denoted by the third equation. If there exists an Auth-Request message in the network, the hashed Common appearing in clear in the message is available to the intruder, which is denoted by the fourth equation. Besides, if the EncSlip in the message is encrypted with the intruder's public key, the other hashed Common in the message is also available to the intruder, which is denoted by the fifth equation. Whether there exists a hashed Common available to the intruder in a message does not depend on other messages because there never exists a message from which a private key is available, which is denoted by the final equation.

The set of equations that define `bans` is as follows:

```
eq  ban(ib) \in bans(void) = true .
ceq N \in bans(void) = false if not(N = ban(ib)) .
ceq N \in bans(M,NW) = true
    if pm?(M) and pk(ia) = pk(eslip(M))
              and N = ban(slip(eslip(M))) .
ceq N \in bans(M,NW) = true
    if qm?(M) and pk(ia) = pk(eslip(M))
              and N = ban(slip(eslip(M))) .
ceq N \in bans(M,NW) = N \in bans(NW)
    if not(pm?(M) and pk(ia) = pk(eslip(M))
                  and N = ban(slip(eslip(M)))) and
       not(qm?(M) and pk(ia) = pk(eslip(M))
                  and N = ban(slip(eslip(M)))) .
```

The intruder can use his/her own BAN denoted by `ban(ib)` at any time, which is denoted by the first equation. Any other BANs are not available to the intruder if the network is empty, which is denoted by the second equation. If there exists a message that includes a BAN in the network, the BAN could be gleaned by the intruder,

provided that if the BAN is part of a ciphertext, the intruder must know the key to decrypt the ciphertext. Such messages are Payment and Auth-Request messages. The remaining equations describe how to obtain BANs from such messages like the equations that define `hcoms`.

The remaining operators are defined likewise.

4.5. *Formalization of Behavior of E-Commerce Protocols*

The behavior of an e-commerce protocol is formalized as an OTS. We first select observers. One indispensable observer, say $nw : \Upsilon \rightarrow Network$, is used to observe the network with which messages are transmitted, where *Network* is the data type corresponding to the visible sort `Network`. Given a state υ of the OTS, $nw(\upsilon)$ denotes the snapshot of the network in the state υ. We next define the initial value returned by each observer. Let *init* denote an arbitrary initial state of the OTS. Since the network is initially empty, $nw(init)$ is defined as the empty bag. We then find transitions, which change the value returned by each observer in conformity with the behavior of the protocol. Transitions are classified into two classes: (1) transitions in one class represent the behavior of trustable principals, and (2) transitions in the other class represent the behavior of the intruder. The former send messages exactly following the protocol, and the latter fake and send messages based on the gleaned information by the intruder from the network.

The behavior of an e-commerce protocol is basically modeled by sending messages via a network, which are denoted by transitions. Messages are asynchronously sent. Receipt of messages is implicitly expressed as the effective conditions of transitions.

Let $\mathcal{S}_{\text{AM3KP}}$ be an OTS formalizing AM3KP. $\mathcal{S}_{\text{AM3KP}}$ has two observers. In addition to *nw*, we use one more observer to observe a random number, which has been generated by a perfect random number generator. The two observers are represented by the two observation operators that are declared as follows:

```
bop nw   : Protocol -> Network
bop rand : Protocol -> Rand
```

`Protocol` is the hidden sort denoting the state space Υ. Given a state p, `nw`(p) denotes the snapshot of the network in p and `rand`(p) denotes the random number that has been generated most recently by a perfect random number generator in p.

We have the constant `init`, whose sort is `Protocol`, that denotes an arbitrary initial state of $\mathcal{S}_{\text{AM3KP}}$. The network is initially empty, and the random number is initially arbitrary. Therefore, we have one equation defining `init`:

```
eq nw(init) = void .
```

Formalization of Trustable Principals

Since AM3KP uses five kinds of messages, we use five transitions to formalize the behavior of trustable principals. The five transitions are represented by the five action operators `sdim`, `sdvm`, `sdpm`, `sdqm` and `sdsm`. `sdim` stands for sending Initiate messages, `sdvm` for sending Invoice messages, `sdpm` for sending Payment messages, `sdqm` for sending Auth-Request messages, and `sdsm` for sending Auth-Response messages. The operators are declared as follows:

```
bop sdim : Protocol Buyer  Seller       -> Protocol
bop sdvm : Protocol Seller Msg          -> Protocol
bop sdpm : Protocol Buyer  Rand Msg Msg -> Protocol
bop sdqm : Protocol Seller Hban Msg Msg -> Protocol
bop sdsm : Protocol Msg                 -> Protocol
```

Those operators are defined with equations. Let `P`, `B`, `S`, `R`, `M1` and `M2` be CafeOBJ variables whose sorts are `Protocol`, `Buyer`, `Seller`, `Rand`, `Msg` and `Msg`, respectively, in the rest of this section.

The effective condition of the transition denoted by `sdim` is always true. The set of equations that define `sdim` is as follows:

```
eq nw(sdim(P,B,S))
   = im(B,B,S,h(nxt(B,rand(P)),ban(B))) , nw(P) .
eq rand(sdim(P,B,S)) = nxt(B,rand(P)) .
```

The equations mean that a buyer `B` generates a newly fresh random number denoted by `nxt(B,rand(P))`, creates an Invoice message denoted by `im(B,B,S,h(nxt(B,rand(P)),ban(B)))` and sends it to a seller `S` by putting it into the network denoted by `nw(P)`. Note that the comma between `im(...)` and `nw(P)` is the data constructor of non-empty bags.

The effective condition of the transition denoted by `sdpm` is that there exists a valid Invoice message in the network, which seems to have been sent by a seller S to a buyer B and looks like the response to the Initiate message that has been sent by B to S. If there exists such an Invoice message in the network, B can receive the message and respond to it by sending a Payment message to S, which is formalized by the transition.

The effective condition is denoted by the operator `c-sdpm` that is declared and defined as follows:

```
op c-sdpm : Protocol Buyer Rand Msg Msg -> Bool
eq c-sdpm(P,B,R,M1,M2)
   = (M1 \in nw(P) and im?(M1) and B = ic(M1) and
      B = is(M1) and hban(M1) = h(R,ban(B)) and
      M2 \in nw(P) and vm?(M2) and B = vd(M2) and
      id(M1) = vs(M2) and
      sig(sk(vs(M2)),hcom(clear(M2))) = sigs(M2) and
```

```
      hcom(clear(M2)) = h(com(id(M1),B,hban(M1))) .
```

The term `c-sdpm(P,B,R,M1,M2)` means that there exists a valid Invoice message `M2` in the network, which seems to have been sent by a seller `vs(M2)` to a buyer `B` and looks like the response to the Initiate message `M1` that has been sent by `B` to `vs(M2)`, and a random number `R` is used in the Initiate message. The term `M1 \in nw(P) and im?(M1) and B = ic(M1) and B = is(M1) and hban(M1) = h(R,ban(B))` means that there exists an Initiate message `M1`, in which `R` is used, in the network, which implies that `B` has sent `M1` to `id(M1)`. Note that `B` uses the network as his/her private memory that reminds him/her to send `M1` as described in Subsection 4.4. The term `M2 \in nw(P) and vm?(M2) and B = vd(M2) and id(M1) = vs(M2)` means that there exists an Invoice message `M2` in the network, which implies that `vs(M2)`, which equals `id(M1)`, seems to have sent `M2` to `B`. The term `sig(sk(vs(M2)),hcom(clear(M2))) = sigs(M2)` means that the signature retrieved from `M2` is valid, and the term `hcom(clear(M2)) = h(com(id(M1),B,hban(M1)))` means that the hashed Common retrieved from `M2` is proper with respect to `M1`.

The set of equations that define `sdpm` is as follows:

```
ceq nw(sdpm(P,B,R,M1,M2))
    = pm(B,B,vs(M2),
         enc(pk(la),slp(hcom(clear(M2)),ban(B),R)),
         sig(sk(B),enc(pk(la),
                       slp(hcom(clear(M2)),ban(B),R)),
                   hcom(clear(M2)))) , nw(P)
    if c-sdpm(P,B,R,M1,M2) .
eq  rand(sdpm(P,B,R,M1,M2)) = rand(P) .
bceq sdpm(P,B,R,M1,M2)      = P
    if not c-sdpm(P,B,R,M1,M2) .
```

The equations mean that if the effective condition denoted by `c-sdpm(P,B,R,M1,M2)` holds, `B` responds to `M2` by sending a Payment message denoted by `pm(...)` to the seeming source `vs(M2)` of `M2`, and otherwise nothing changes.

The remaining action operators can be defined in terms of equations likewise.

Formalization of the Intruder

Part of the intruder has been modeled as the network. We have defined what information the intruder can glean from the network. We next describe what messages the intruder fakes based on the gleaned information, which are formalized by transitions.

The transitions corresponding to the intruder's faking messages are divided into five classes. Transitions in each class fake messages corresponding to one of the five kinds. The effective conditions of these transitions are that the intruder can use the

necessary information to fake messages.

We basically suppose that the intruder can fake any message if the message can be made of the values gleaned by the intruder. But, we do not have the intruder fake meaningless messages that do not clearly conform to the protocol and do not clearly work for the attack of the protocol. For example, the intruder does not fake a Payment message denoted by `pm(ib,b,s,ep,sig(sk(ib),ep,hc))`, where b is a trustable buyer. If the message is faked and sent, a seller s just rejects it because the signature denoted by `sig(sk(ib),ep,hc)` is not computed with the private key of the seeming sender b. Therefore, the message does not attack the protocol.

We have two transitions that fake Initiate messages, six transitions that fake Invoice messages, five transitions that fake Payment messages, 12 transitions that fake Auth-Request messages, and four transitions that fake Auth-Response messages. These transitions are represented by action operators.

We describe how to obtain these transitions. Let us take Payment messages, for example. Since a Payment message consists of an EncSlip and an instance of Sig_B, the intruder can fake a Payment message if an EncSlip is available to the intruder or able to be computed by the intruder, and so is an instance of Sig_B. Note that the intruder chooses an arbitrary seller ID and/or an arbitrary buyer ID. An EncSlip is computed from a hashed Common, a BAN and a random number with the legitimate acquirer's public key, a hashed Common is computed from a seller ID, a buyer ID and a keyed hash of a BAN, and a keyed hash of a BAN is computed from a BAN and a random number. All needed to compute an EncSlip is then a BAN and a random number. Therefore, there are two possibilities to obtain an EncSlip computed in accordance with the protocol: (1) the intruder has an EncSlip itself and (2) the intruder has a BAN and a random number. This means that the intruder may compute an EncSlip denoted by `enc(pk(la),slp(h(com(s,b(n),h(r,n))),n,r))` from a BAN n and a random number r, but does not compute an EncSlip denoted by `enc(pk(la),slp(hc,n,r))` from a hashed Common hc, a BAN n and a random number r because the EncSlip may not follow the protocol. As an EncSlip, there are four possibilities to obtain an instance of Sig_B computed in accordance with the protocol: (1) the intruder has an instance of Sig_B itself, (2) the intruder has an EncSlip and a hashed Common, (3) the intruder has an EncSlip and a keyed hash of a BAN and (4) the intruder has a BAN and a random number. Although the number of the naive combinations is eight, the necessary combinations among them are (1,1), (1,2), (1,3), (2,1) and (2,4). Note that the combination (2,4) uses one BAN and one random number. The reason why the combination (1,4) is redundant is that a BAN and a random number can be used to compute an EncSlip, and the reasons why the other combinations are redundant are similar. Consequently, we have five transitions to fake Payment messages. The remaining transitions to fake messages of the other four kinds can be obtained likewise.

In this paper, we show the five action operators corresponding to the transitions faking Payment messages, which are declared as follows:

```
bop fkpm1 : Protocol Buyer  Seller Eslp Sigb -> Protocol
bop fkpm2 : Protocol Seller Ban    Rand Sigb -> Protocol
bop fkpm3 : Protocol Buyer  Seller Eslp Hcom -> Protocol
bop fkpm4 : Protocol Buyer  Seller Eslp Hban -> Protocol
bop fkpm5 : Protocol Seller Ban    Rand      -> Protocol
```

`fkpm` stands for faking Payment messages. Given an EncSlip *es* and an instance *gb* of Sig_B that are available to the intruder, `fkpm1` fakes a Payment message denoted by `pm(ib,b,s,es,gb)`, where *b* and *s* are an arbitrary buyer and an arbitrary seller. Given a BAN *n*, a random number *r* and an instance *gb* of Sig_B that are available to the intruder, `fkpm2` fakes a Payment message denoted by `pm(ib,b(n),s,enc(pk(la),slp(h(com(s,b(n),h(r,n))),n,r)),gb)`, where *s* is an arbitrary seller. The remaining `fkpm` action operators fake Payment messages likewise.

The effective condition of the transition denoted by `fkpm2` is denoted by the operator `c-fkpm2` that is declared and defined as follows:

```
op c-fkpm2 : Protocol Seller Ban Rand Sigb -> Bool
op c-fkpm2(P,S,N,R,GB)
   = (N \in bans(nw(P)) and R \in rands(nw(P))
                        and GB \in sigbs(nw(P))) .
```

The set of equations that define `fkpm2` is as follows:

```
ceq nw(fkpm2(P,S,N,R,GB))
    = pm(ib,b(N),S,
         enc(pk(la),slp(h(com(S,b(N),h(R,N))),N,R)),GB)
      , nw(P) if c-fkpm2(P,S,N,R,GB) .
eq  rand(fkpm2(P,S,N,R,GB)) = rand(P) .
bceq fkpm2(P,S,N,R,GB)      = P
    if not c-fkpm2(P,S,N,R,GB) .
```

The remaining `fkpm` operators are defined with equations likewise. The action operators faking other kinds of messages are also declared and defined likewise.

5. Formalization of Properties

In our way of modeling e-commerce protocols, their properties are basically expressed in terms of the existence of messages in the network and the existence of values in the collections gleaned by the intruder from the network. Note that all properties considered are invariants.

Given a state *p*, let $nw(p)$ denote the network in the state, let $vals(nw(p))$, $vals_1(nw(p))$ and $vals_2(nw(p))$ be collections of values gleaned by the intruder from the network, and let *pred* be a predicate. Moreover let *msg* be the data constructor of some kind of message, let p_1, p_2 and q denote principals, let *i* denote the intruder,

and let b be a message body. Properties of e-commerce protocols are then expressed using the following nine kinds of invariants:

(1) $\mathsf{invariant}\,(x \in \mathit{vals}(\mathit{nw}(p)) \Rightarrow \mathit{pred}(y))$.

Secrecy properties can be expressed using this kind of invariant. For example, assuming that $y = x$ and $\mathit{pred}(x)$ means that x is generated by the intruder, this invariant is to claim that values related to *vals* cannot be obtained illegally by the intruder.

(2) $\mathsf{invariant}\,(x_1 \in \mathit{vals}_1(\mathit{nw}(p)) \Rightarrow x_2 \in \mathit{vals}_2(\mathit{nw}(p)))$.

This is a special case of the first kind of invariant. This kind of invariant makes it possible to express properties of $\mathit{vals}_1(\mathit{nw}(p))$ in terms of those of $\mathit{vals}_2(\mathit{nw}(p))$. If we know that values related to val_2 cannot be obtained illegally by the intruder, then we can deduce that those related to val_1 cannot be obtained illegally by the intruder either from this invariant.

(3) $\mathsf{invariant}\,(x \in \mathit{vals}(\mathit{nw}(p)) \Rightarrow m \in \mathit{nw}(p))$.

This is also a special case of the first kind of invariant. This kind of invariant means that x is never obtained by the intruder unless m is sent.

(4) $\mathsf{invariant}\,(m \in \mathit{nw}(p) \Rightarrow \mathit{pred}(x))$.

This kind of invariant means that $\mathit{pred}(x)$ is closely related to the existence of the message m in the network. Let m be $\mathit{msg}(p_1, i, q, b)$ and $\mathit{pred}(x)$ be $p_1 = i$, and then this invariant holds for all OTSs modeling e-commerce protocols (see Subsection 4.3).

(5) $\mathsf{invariant}\,(m \in \mathit{nw}(p) \Rightarrow x \in \mathit{vals}(\mathit{nw}(p)))$.

This is a special case of the fourth kind of invariant. This kind of invariant means that x can be gleaned by the intruder from m.

(6) $\mathsf{invariant}\,(m_1 \in \mathit{nw}(p) \Rightarrow m_2 \in \mathit{nw}(p))$.

This is also a special case of the fourth kind of invariant. One-to-many correspondences can be expressed using this kind of invariant. This invariant claims that if there exists m_1 in the network, then there also exists m_2 in the network, although it does not claim that there exists one and only one m_2 in the network. Let m_1 and m_2 be $\mathit{msg}_1(p_2, p_1, q_1, b_1)$ and $\mathit{msg}_2(p_3, p_3, q_2, b_2)$. This invariant then claims that if q_1 receives m_1, no matter who m_1 originates from, then p_3 has always sent m_2 to q_2.

(7) $\mathsf{invariant}\,(\mathit{msg}(p_2, p_1, q, b) \in \mathit{nw}(p) \Rightarrow \mathit{msg}(p_1, p_1, q, b) \in \mathit{nw}(p))$.

This is a special case of the sixth kind of invariant. This kind of invariant assures that the message originates from the right principal, namely that the intruder cannot fake this message unless the right principal sends it.

(8) $\mathsf{invariant}\,(\mathit{pred}(x) \Rightarrow y \in \mathit{vals}(\mathit{nw}(p)))$.

This is a general case of the second and fifth kinds of invariant and the inverse of the first kind of invariant.

(9) $\mathsf{invariant}\,(\mathit{pred}(x) \Rightarrow m \in \mathit{nw}(p))$.

This is a general case of the third and sixth kinds of invariant and the inverse of the fourth kind of invariant.

For AM3KP, let us consider formalizing Payment Agreement. In our way of modeling e-commerce protocols, the receipt of a message, which seems have been sent by a principal p_1, by a principal p_2 implies the existence of the message whose second argument is p_1 and third argument is p_2 in the network, and the existence of a message in the network implies the transmission of the message by the principal denoted by the first argument of the message. Therefore, Payment Agreement can be expressed in the combination of the sixth, seventh and ninth kinds of invariant.

Before showing the invariant expressing Payment Agreement, the following are defined:

- $\mathrm{mkhc}(s, b, r) \triangleq \mathtt{h}(\mathtt{com}(s, b, \mathtt{h}(r, \mathtt{ban}(b))))$
- $\mathrm{mkcl}(s, b, r) \triangleq \mathtt{cl}(\mathrm{mkhc}(s, b, r))$
- $\mathrm{mkeslp}(s, b, r) \triangleq \mathtt{enc}(\mathtt{pk}(\mathtt{la}), \mathtt{slp}(\mathrm{mkhc}(s, b, r), \mathtt{ban}(b), r))$
- $\mathrm{mkgs2}(s, b, r) \triangleq \mathtt{sig}(\mathtt{sk}(s), \mathrm{mkhc}(s, b, r), \mathrm{mkeslp}(s, b, r))$
- $\mathrm{mkgb}(s, b, r) \triangleq \mathtt{sig}(\mathtt{sk}(b), \mathrm{mkeslp}(s, b, r), \mathrm{mkhc}(s, b, r))$
- $\mathrm{mkgs}(s, b, r) \triangleq \mathtt{sig}(\mathtt{sk}(s), \mathrm{mkhc}(s, b, r))$

Those terms denote a hashed Common, a Clear, an EncSlip, an instance of $\mathrm{Sig2}_S$, an instance of Sig_B and an instance of Sig_S, respectively, that are built in accordance with the protocol.

Payment Agreement is then formally expressed as the following invariant:

$$
\begin{aligned}
&\mathsf{invariant}\ (\neg(\mathit{s1} = \mathtt{is}\ \mathtt{and}\ \mathit{b1} = \mathtt{ib}) \wedge \\
&\quad \mathtt{qm}(\mathit{s2}, \mathit{s1}, \mathtt{la}, \mathrm{mkcl}(\mathit{s1}, \mathit{b1}, \mathit{r1}), \mathrm{mkeslp}(\mathit{s1}, \mathit{b1}, \mathit{r1}), \\
&\qquad\qquad \mathrm{mkgs2}(\mathit{s1}, \mathit{b1}, \mathit{r1}), \mathrm{mkgb}(\mathit{s1}, \mathit{b1}, \mathit{r1})) \in \mathtt{nw}(p) \\
&\quad \Rightarrow \\
&\quad \mathtt{im}(\mathit{b1}, \mathit{b1}, \mathit{s1}, \mathtt{h}(\mathit{r1}, \mathtt{ban}(\mathit{b1}))) \in \mathtt{nw}(p) \wedge \\
&\quad \mathtt{vm}(\mathit{s1}, \mathit{s1}, \mathit{b1}, \mathrm{mkcl}(\mathit{s1}, \mathit{b1}, \mathit{r1}), \mathrm{mkgs}(\mathit{s1}, \mathit{b1}, \mathit{r1})) \in \mathtt{nw}(p) \wedge \\
&\quad \mathtt{pm}(\mathit{b1}, \mathit{b1}, \mathit{s1}, \mathrm{mkeslp}(\mathit{s1}, \mathit{b1}, \mathit{r1}), \mathrm{mkgb}(\mathit{s1}, \mathit{b1}, \mathit{r1})) \in \mathtt{nw}(p) \wedge \\
&\quad \mathtt{qm}(\mathit{s1}, \mathit{s1}, \mathtt{la}, \mathrm{mkcl}(\mathit{s1}, \mathit{b1}, \mathit{r1}), \mathrm{mkeslp}(\mathit{s1}, \mathit{b1}, \mathit{r1}), \\
&\qquad\qquad \mathrm{mkgs2}(\mathit{s1}, \mathit{b1}, \mathit{r1}), \mathrm{mkgb}(\mathit{s1}, \mathit{b1}, \mathit{r1})) \in \mathtt{nw}(p))\,.
\end{aligned}
$$

Let this invariant be called Inv0. If a protocol run is performed by `la`, `is` and `ib`, namely the legitimate acquirer and the intruder, it is clear that this setting breaks the property because the intruder fakes any Auth-Request message stating that `ib` pays any amount to `is`. That is why the first conjunct of the premise of the property is added. *s2* might be the intruder, and if *s1* does not equal *s2*, the Auth-Request message whose first argument is *s2* has been faked by the intruder.

6. Verification of Properties

6.1. *Proof Scores of Invariants*

We describe how to write proof scores of invariants. Although some invariants may be proved by rewriting and/or case analyses only, we often need to use induction, especially simultaneous induction on the structure of reachable states with respect

to an OTS $\mathcal{S}$[49,53]. We then describe how to verify $\mathsf{invariant}_{\mathcal{S}}\ p_1$ by simultaneous induction on the structure of reachable states with respect to $\mathcal{S}$ by writing proof scores in CafeOBJ based on the CafeOBJ specification of $\mathcal{S}$.

It is often impossible to prove $\mathsf{invariant}_{\mathcal{S}}\ p_1$ alone. We then suppose that it is possible to prove $\mathsf{invariant}_{\mathcal{S}}\ p_1$ together with $n-1$ other state predicates[c]. Let the $n-1$ other state predicates be $p_2, \ldots, p_n$. That is, we prove $\mathsf{invariant}_{\mathcal{S}}\ (p_1 \wedge \ldots \wedge p_n)$. Let $x_{i1}, \ldots, x_{im_i}$ whose types are $D_{i1}, \ldots, D_{im_i}$ be all free variables in p_i except for υ whose type is Υ, where $i = 1, \ldots, n$. p_i may be written as $p_i(\upsilon, x_{i1}, \ldots, x_{im_i})$, where $i = 1, \ldots, n$, and $p_1 \wedge \ldots \wedge p_n$ may be written as p or $p(\upsilon, x_{11}, \ldots, x_{nm_n})$.

Let *init* denote an arbitrary initial state of $\mathcal{S}$. For the base case, all we have to do is to prove

$$p_i(init, x_{i1}, \ldots, x_{im_i}) \tag{1}$$

for $i = 1, \ldots, n$. We suppose that the free variables $x_{i1}, \ldots, x_{im_i}$ are universally quantified. We also suppose that all free variables in every formula are universally quantified in this subsection unless otherwise stated. (1) is logically equivalent to $p(init, x_{11}, \ldots, x_{nm_n})$.

For the inductive cases, for each $t_{j_1,\ldots,j_{m_j}} \in \mathcal{T}$ all we have to do is to prove

$$(SIH_i \wedge p_i(\upsilon, x_{i1}, \ldots, x_{im_i})) \Rightarrow p_i(t_{j_1,\ldots,j_{m_j}}(\upsilon), x_{i1}, \ldots, x_{im_i}) \tag{2}$$

for $i = 1, \ldots, n$. SIH_i is used to strengthen the basic inductive hypothesis $p_i(\upsilon, x_{i1}, \ldots, x_{im_i})$ and can be in the form $p_\alpha(\upsilon, e_{\alpha 1}, \ldots, e_{\alpha m_\alpha}) \wedge p_\beta(\upsilon, e_{\beta 1}, \ldots, e_{\beta m_\beta}) \wedge \ldots$, where $\alpha, \beta, \ldots \in \{1, \ldots, n\}$ and each e_ι is an expression whose type is D_ι. From (2), we can deduce

$$\begin{aligned}&(SIH_1 \wedge \ldots \wedge SIH_n \wedge p(\upsilon, x_{11}, \ldots, x_{nm_n}))\\ &\Rightarrow\ p(t_{j_1,\ldots,j_{m_j}}(\upsilon), x_{11}, \ldots, x_{nm_n})\,.\end{aligned}$$

The formula says that p holds in $t_{j_1,\ldots,j_{m_j}}(\upsilon)$ if p holds in υ, which corresponds to the inductive case where we show that $t_{j_1,\ldots,j_{m_j}}$ preserves p for the proof of $\mathsf{invariant}_{\mathcal{S}}\ p$ by induction on the structure of reachable states with respect to $\mathcal{S}$.

From what has been described, all we have to do is to prove (1) and (2) in order to prove $\mathsf{invariant}_{\mathcal{S}}\ p$. This means that it is possible to write the proof of each $\mathsf{invariant}_{\mathcal{S}}\ p_i$ separately, where $i = 1, \ldots, n$. Since we prove multiple state predicates $p_1, \ldots, p_n$ invariant with respect to $\mathcal{S}$ (virtually) simultaneously by induction, we call the proof method *simultaneous induction*.

We next describe how to write proof plans of (1) and (2) in CafeOBJ. We suppose that $\mathcal{S}$ is written in CafeOBJ.

We first declare the operators denoting $p_1, \ldots, p_n$ and the equations defining the operators. The operators and equations are declared in a module, say `INV` (which imports the module where $\mathcal{S}$ is written), as follows:

[c]Generally such $n-1$ state predicates should be found while $\mathsf{invariant}_{\mathcal{S}}\ p_1$ is being proved.

`op` inv_i : H $V_{i1} \ldots V_{im_i}$ `-> Bool`
`eq` $inv_i(S, X_{i1}, \ldots, X_{im_i}) = p_i(S, X_{i1}, \ldots, X_{im_i})$.

for $i = 1, \ldots, n$. $p_i(S, X_{i1}, \ldots, X_{im_i})$ is a CafeOBJ term denoting p_i. In the module `INV`, we also declare a constant x_k denoting an arbitrary value of V_k, where $k = 11, \ldots, nm_n$.

We then declare the operators denoting basic formulas to prove in the inductive cases and the equations defining the operators. The operators and equations are declared in a module, say `ISTEP` (which imports `INV`), as follows:

`op` $istep_i$: `-> Bool`
`eq` $istep_i = inv_i(s, x_{i1}, \ldots, x_{im_i})$ `implies` $inv_i(s', x_{i1}, \ldots, x_{im_i})$.

for $i = 1, \ldots, n$. s and s' are constants of H, which are declared in `INV` and `ISTEP`, respectively. s denotes an arbitrary state and s' denotes an arbitrary successor state of s. $inv_i(s', x_{i1}, \ldots, x_{im_i})$ is the formula to prove in each inductive case and $inv_i(s, x_{i1}, \ldots, x_{im_i})$ is an instance of the induction hypothesis. Since the instance is often used, $istep_i$ is defined as above.

The proof plan of (1), written in CafeOBJ, is like

```
open INV
 red inv_i(init, x_i1, ..., x_im_i) .
close
```

for $i = 1, \ldots, n$. CafeOBJ scripts like this constitute *proof scores*. Such fragments of proof scores are called *proof passages*. Feeding this proof passage into the CafeOBJ system, if the system returns `true`, then the proof is successful in the base case. Otherwise, you need to do case analyses and/or use some lemmas.

The proof of (2) often needs case analysis. We suppose that the state space is split into l sub-spaces[d] in order to prove (2) and that each sub-space is characterized by a state predicate $case_{i_k}$, where $k = 1, \ldots, l$. The state predicates should satisfy $(case_{i_1} \vee \ldots \vee case_{i_l}) \Leftrightarrow$ true. Then the proof of (2) can be replaced with

$$(SIH_i \wedge case_{i_k} \wedge p_i(\upsilon, x_{i1}, \ldots, x_{im_i})) \Rightarrow p_i(t_{j_1,\ldots,j_{m_j}}(\upsilon), x_{i1}, \ldots, x_{im_i}) \qquad (4)$$

where $i = 1, \ldots, n$ and $k = 1, \ldots, l$.

We suppose that $t_{j_1,\ldots,j_{m_j}}$ is denoted by a CafeOBJ action operator t and the CafeOBJ term that denotes SIH_i is $\mathtt{SIH}_i$. Then the proof passage of (4) is like

```
open ISTEP
 -- arbitrary objects
 op y_1m_1 : -> V_1m_1 .
 ...
 op y_j_m_j : -> V_j_m_j .
```

[d]Generally such case analysis should be done while $\mathsf{invariant}_{\mathcal{S}}\ p_1$ is being proved.

`-- assumptions`
Declaration of equations denoting $case_{i_k}$.
`-- successor state`
`eq` $s' = t(s, y_{j_1}, \ldots, y_{j_{m_j}})$.
`-- check`
`red` $\mathtt{SIH}_i$ `implies` $istep_i$.
`close`

for $i = 1, \ldots, n$ and $k = 1, \ldots, l$. A comment starts with `--` and terminates at the end of the line. The constants $y_{1_{m_1}}$, ..., $y_{j_{m_j}}$ denote arbitrary values of the corresponding visible sorts. The state predicate $case_{i_k}$ representing the subcase is written in equations in the proof passage. $\mathtt{SIH}_i$ `implies` $istep_i$ is logically equivalent to $(\mathtt{SIH}_i$ `and` $inv_i(s, x_{i1}, \ldots, x_{im_i}))$ `implies` $inv_i(s', x_{i1}, \ldots, x_{im_i})$. Feeding this proof passage into the CafeOBJ system, if the system returns `true`, then the proof that $t_{j_1,\ldots,j_{m_j}}$ preserves p_i is successful in the case denoted by $case_{i_k}$. Otherwise, you need to do further case analyses and/or use other instances of state predicates to strengthen the induction hypothesis furthermore.

6.2. *Verification of Payment Agreement*

We need 17 more invariants in order to prove that AM3KP enjoys Payment Agreement, namely that Inv0 holds for $\mathcal{S}_{\mathrm{AM3KP}}$. Five of the invariants, including Inv0, are proved by rewriting and/or case analyses only, and the remaining are proved by simultaneous induction on the structure of reachable states with respect to $\mathcal{S}_{\mathrm{AM3KP}}$. We write proof scores in CafeOBJ for all invariant properties.

First we describe the proof of Inv0, which needs to prove four more state predicates invariant. We declare the five operators denoting the five state predicates in a module `INV` (which imports the module where $\mathcal{S}_{\mathrm{AM3KP}}$ is written) as follows:

```
op inv0  : Protocol Buyer Seller Seller Rand -> Bool
op inv5  : Protocol Buyer Seller Seller Rand -> Bool
op inv9  : Protocol Buyer Seller Seller Rand -> Bool
op inv13 : Protocol Buyer Seller Seller Rand -> Bool
op inv17 : Protocol Buyer Seller Seller Rand -> Bool
```

The five operators are defined in equations as follows:

```
eq inv0(P,B1,S1,S2,R1)
   = (not(S1 = is and B1 = ib) and
      qm(S2,S1,la,mkcl(S1,B1,R1),mkeslp(S1,B1,R1),
         mkgs2(S1,B1,R1),mkgb(S1,B1,R1)) \in nw(P)
      implies
      im(B1,B1,S1,h(R1,ban(B1))) \in nw(P) and
      vm(S1,S1,B1,mkcl(S1,B1,R1),mkgs(S1,B1,R1))
                                          \in nw(P) and
```

```
      pm(B1,B1,S1,mkeslp(S1,B1,R1),mkgb(S1,B1,R1))
                                          \in nw(P) and
      qm(S1,S1,la,mkcl(S1,B1,R1),mkeslp(S1,B1,R1),
         mkgs2(S1,B1,R1),mkgb(S1,B1,R1)) \in nw(P)) .

eq inv5(P,B1,S1,S2,R1)
   = (qm(S2,S1,la,mkcl(S1,B1,R1),mkeslp(S1,B1,R1),
         mkgs2(S1,B1,R1),mkgb(S1,B1,R1)) \in nw(P)
      implies
      qm(S1,S1,la,mkcl(S1,B1,R1),mkeslp(S1,B1,R1),
         mkgs2(S1,B1,R1),mkgb(S1,B1,R1)) \in nw(P)) .

eq inv9(P,B1,S1,S2,R1)
   = (not(S1 = is and B1 = ib) and
      qm(S2,S1,la,mkcl(S1,B1,R1),mkeslp(S1,B1,R1),
         mkgs2(S1,B1,R1),mkgb(S1,B1,R1)) \in nw(P)
      implies
      vm(S1,S1,B1,mkcl(S1,B1,R1),mkgs(S1,B1,R1))
                                          \in nw(P)) .

eq inv13(P,B1,S1,S2,R1)
   = (not(S1 = is and B1 = ib) and
      qm(S2,S1,la,mkcl(S1,B1,R1),mkeslp(S1,B1,R1),
         mkgs2(S1,B1,R1),mkgb(S1,B1,R1)) \in nw(P)
      implies
      pm(B1,B1,S1,mkeslp(S1,B1,R1),mkgb(S1,B1,R1))
                                          \in nw(P)) .

eq inv17(P,B1,S1,S2,R1)
   = (not(S1 = is and B1 = ib) and
      qm(S2,S1,la,mkcl(S1,B1,R1),mkeslp(S1,B1,R1),
         mkgs2(S1,B1,R1),mkgb(S1,B1,R1)) \in nw(P)
      implies
      im(B1,B1,S1,h(R1,ban(B1))) \in nw(P)) .
```

`P`, `B1`, `S1`, `S2` and `R1` are CafeOBJ variables whose sorts are `Protocol`, `Buyer`, `Seller`, `Seller` and `Rand`, respectively. `mkcl(S1,B1,R1)`, `mkeslp(S1,B1,R1)`, `mkgs2(S1,B1,R1)`, `mkgb(S1,B1,R1)` and `mkgs(S1, B1,R1)` are terms corresponding to mkcl(s, b, r), mkeslp(s, b, r), mkgs2$(\ s, b, r)$, mkgb(s, b, r) and mkgs(s, b, r), respectively. In the module `INV`, we declare constants `p` of `Protocol`, `b1` of `Buyer`, `s1` and `s2` of `Seller`, and `r1` of `Rand`.

Assuming that the four state predicates denoted by `inv5`, `inv9`, `inv13` and `inv17` are invariants with respect to $\mathcal{S}_{\mathrm{AM3KP}}$, we can write the proof score of Inv0

as follows:

```
open INV
-- check if the predicate is true
  red inv5(p,b1,s1,s2,r1) and inv9(p,b1,s1,s2,r1) and
      inv13(p,b1,s1,s2,r1) and inv17(p,b1,s1,s2,r1)
      implies
      inv0(p,b1,s1,s2,r1) .
close
```

Feeding this proof score into the CafeOBJ system, the system returns `true` as expected, which means that the proof is successful, provided that the four invariants hold for $\mathcal{S}_{\text{AM3KP}}$.

Next we describe the plan of the proof that the state predicate denoted by `inv9` is invariant, which needs case analysis. The case is split into four subcases characterized by the following four state predicates:

(1) $(\texttt{s1} = \texttt{is}) \wedge (\texttt{m10 \in nw(p)})$
(2) $(\texttt{s1} = \texttt{is}) \wedge \neg(\texttt{m10 \in nw(p)})$
(3) $\neg(\texttt{s1} = \texttt{is}) \wedge (\texttt{m10 \in nw(p)})$
(4) $\neg(\texttt{s1} = \texttt{is}) \wedge \neg(\texttt{m10 \in nw(p)})$

where `m10` is `qm(s2,s1,la,mkcl(s1,b1,r1),mkeslp(s1,b1,r1),mkgs2(s1,b1,r1),mkgb(s1,b1,r1))`. The proof passage of subcase 1 needs another state predicate, that of subcase 3 needs two other state predicates, and those of subcases 2 and 4 do not.

We show the proof passage of subcase 1. We declare and define the operator denoting the state predicate needed for the subcase in `INV` as follows:

```
op inv8 : Protocol Buyer Rand -> Bool
eq inv8(P,B1,R1)
   = (not(B1 = ib) and mkeslp(S1,B1,R1) \in eslps(nw(P))
      implies
      vm(S1,S1,B1,mkcl(S1,B1,R1),mkgs(S1,B1,R1))
                                          \in nw(P)) .
```

This state predicate means that if an encrypted SLIP including a BAN whose owner is different from the intruder is available to the intruder, then the seller included in the encrypted SLIP has always sent the owner the Invoice message corresponding to the encrypted SLIP.

Assuming that the state predicate denoted by `inv8` is invariant with respect to $\mathcal{S}_{\text{AM3KP}}$, we can write the proof passage of subcase 1 as follows:

```
open INV
-- arbitrary chosen objects
  op m10 : -> Msg .
```

```
  op nw10 : -> Network .
-- assumptions
  eq s1 = is .
  eq m10 = qm(s2,s1,la,mkcl(s1,b1,r1),mkeslp(s1,b1,r1),
                       mkgs2(s1,b1,r1),mkgb(s1,b1,r1)) .
  eq nw(p) = m10 , nw10 .
-- check if the predicate is true
  red inv8(p,b1,r1) implies inv9(p,b1,s1,s2,r1) .
close
```

The assumption `m10 \in nw(p)` is expressed by the equation `nw(p) = m10 , nw10`. Feeding this proof passage into the CafeOBJ system, the system returns `true` as expected, which means that the proof is successful in subcase 1. The proof passages of the remaining subcases can be written likewise.

We also describe the proof that the state predicate denoted by `inv8` is invariant, which needs simultaneous induction on the structure of reachable states with respect to $\mathcal{S}_{\text{AM3KP}}$. The proof score of the invariant is written in the way described in Subsection 6.1.

We declare and define the operator denoting the basic formula to prove in each inductive case in a module `ISTEP` (which imports `INV`) as follows:

```
op istep8 : -> Bool
eq istep8 = inv8(p,b1,r1) implies inv8(p',b1,r1) .
```

`p'` is a constant of `Protocol` declared in `ISTEP`. `p'` is used to denote an arbitrary successor state of `p`, which is used to denoted an arbitrary state.

For the base case, we can write the following proof passage:

```
open INV
  red inv8(init,b1,r1) .
close
```

The CafeOBJ system returns `true` for this proof passage.

Let us consider the inductive case where the transition denoted by `fkpm2` preserves the state predicate denoted by `inv8`. The case is split into four subcases characterized by the following four state predicates:

(1) `c-fkpm2(p,s10,n10,r10,sb10)` $\wedge$ (`b1` $=$ `ib`)
(2) `c-fkpm2(p,s10,n10,r10,sb10)` $\wedge \neg$(`b1` $=$ `ib`) $\wedge$ (`n10` $=$ `ban`(`ib`))
(3) `c-fkpm2(p,s10,n10,r10,sb10)` $\wedge \neg$(`b1` $=$ `ib`) $\wedge \neg$(`n10` $=$ `ban`(`ib`))
(4) $\neg$`c-fkpm2(p,s10,n10,r10,sb10)`

`s10`, `n10`, `r10` and `sb10` are constants of `Seller`, `Ban`, `Rand` and `Sigb`, respectively. The constants are used as the arguments of `fkpm2`, and declared in each proof passage of the four subcases. The proof passage of subcase 3 needs another state predicate and those of the remaining do not.

We show the proof passage of subcase 3. We declare and define the operator denoting the state predicate needed for the subcase in `INV` as follows:

```
op inv2 : Protocol Ban -> Bool
eq inv2(P,N1)
   = (N1 \in bans(nw(P)) implies N1 = ban(ib)) .
```

`N1` is a CafeOBJ variable whose sort is `Ban`. This state predicate means that BANs available to the intruder are his/her own BAN only, namely that BANs are really secret in the protocol. Then, we can write the proof passage of subcase 3 as follows:

```
open ISTEP
-- arbitrary objects
  op s10 : -> Seller .  op n10 : -> Ban .
  op r10 : -> Rand .    op sb10 : -> Sigb .
-- assumptions
  -- eq c-fkpm2(p,s10,n10,r10,sb10) = true .
  eq n10 \in bans(nw(p)) = true .
  eq r10 \in rands(nw(p)) = true .
  eq sb10 \in sigbs(nw(p)) = true .
  --
  eq (b1 = ib) = false .
  eq (n10 = ban(ib)) = false .
-- successor state
  eq p' = fkpm2(p,s10,n10,r10,sb10) .
-- check
  red inv2(p,n10) implies istep8 .
close
```

Instead of declaring the equation `c-fkpm2(p,s10,n10,r10,sb10) = true`, the three equations are declared to assume that the effective condition holds in `p`. One reason is that `c-fkpm2(p,s10,n10,r10,sb10)` is not in normal form in the sense of term rewriting. Generally the left-hand side of an equation should be in normal form to make effective use of the equation as a rewrite rule. The other reason is that the equation `c-fkpm2(p,s10,n10,r10,sb10) = true` can be deduced from the three equations, while the three equations cannot be deduced from the equation by means of rewriting. The CafeOBJ system returns `true` for this proof passage.

The proof passages of the remaining subcases can be written likewise. The proof passages of the remaining inductive cases can also be written likewise, but yet another invariant is needed. The other invariants can be proved in the ways shown in this subsection.

The size of all the proof scores is approximately of 20,000 lines. It took about 4 minutes to have the CafeOBJ system load the CafeOBJ specification and execute all the proof scores on a laptop with 850MHz Pentium III processor and 512MB memory.

7. Concluding Remarks

We have described the proof score approach to analysis of e-commerce protocols, which has been developed and refined through several case studies conducted. As described in Section 1, compared to systems analysis using other existing interactive theorem provers such as Isabelle/HOL and Coq, the proof score approach to systems analysis has some advantages: (1) balanced human-computer interaction and (2) flexible but clear structure of proof scores. Moreover, thanks to our way of modeling messages and networks, we can express various properties such that some events always precede others in terms of invariants only. It is not necessary to introduce other constructs such as finite automata and filtering functions, which may make verification complicated.

As described in Section 1, the proof score approach to systems analysis is less rigorous than systems analysis using other existing interactive theorem provers such as Isabelle/HOL and Coq. To overcome this disadvantage, we have been developing some tools: Gateau[60] and Crème[41,42,43]. Given state predicates for case splitting and necessary lemmas, Gateau generates proof scores. Gateau has successfully generated the proof scores to verify that some security properties hold for the NSLPK authentication protocol[33,34] and the Otway-Rees authentication protocol[56]. Crème is an automatic invariant verification tool for algebraic specifications of OTSs. We have verified fully automatically that some security properties hold for the NSLPK authentication protocol and the STS authentication protocol[13]. One piece of our future work is to apply those tools to analyses of e-commerce protocols.

To take advantage of model checkers, which are complementary to interactive theorem provers, we have also been developing some tools, which takes CafeOBJ specifications of OTSs and generates specifications that can be model-checked: Chocolat/SMV[55] and Cafe2Maude[31]. Chocolat/SMV translates CafeOBJ specifications of OTSs into SMV specifications, which are model-checked with SMV[38]. Cafe2Maude translates CafeOBJ specifications of OTSs into Maude specifications that can be model-checked with the Maude model checker[16]. Another piece of our future work is to apply those tools to analyses of e-commerce protocols.

This paper only focuses on invariant properties. There must be, however, some security properties that cannot be expressed as invariant properties. The OTS/CafeOBJ method can deal with some class of liveness properties[54]. Therefore, it would be worth pursuing what kind of security properties should be expressed as liveness properties and extending the method described in this paper to deal with liveness properties.

There have been proposed some dedicated security analysis tools. It would be worth integrating the method described in this paper with such a tool.

Acknowledgement

The authors wish to thank anonymous referees who commented on drafts of this paper.

References

1. Giampaolo Bella, Fabio Massacci, and Lawrence C. Paulson. The verification of an industrial payment protocol: The SET purchase phase. In *9th ACM CCS*, pages 12–20, 2002.
2. Giampaolo Bella, Fabio Massacci, and Lawrence C. Paulson. Verifying the SET registration protocols. *IEEE J-SAC*, 21:77–87, 2003.
3. M. Bellare, J. A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, E. Van Herreweghen, and M. Waidner. Design, implementation and deployment of the *i*KP secure electronic payment system. *IEEE J-SAC*, 18(4):611–627, 2000.
4. M. Bellare, J. A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner. *i*KP – a family of secure electronic payment protocols. In *1st USENIX EC*, pages 89–106, 1995.
5. Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development – Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.
6. Dominique Bolignano. Towards the formal verification of electronic commerce protocols. In *10th IEEE CSFW*, pages 133–146, 1997.
7. Michael Butler and Divakar Yadav. An incremental development of the Mondex system in Event-B. *Formal Asp. Comput.*, 20(1):61–77, 2008.
8. Benjamin Cox, J. D. Tygar, and Marvin Sirbu. NetBill security and transaction protocol. In *1st USENIX EC*, pages 77–88, 1995.
9. Răzvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report*, volume 6 of *AMAST Series in Computing*. World Scientific, 1998.
10. Răzvan Diaconescu and Kokichi Futatsugi. Behavioural coherence in object-oriented algebraic specification. *J. UCS*, 6:74–96, 2000.
11. Răzvan Diaconescu, Kokichi Futatsugi, and Kazuhiro Ogata. CafeOBJ: Logical foundations and methodologies. *Computing and Informatics*, 22:257–283, 2003.
12. T. Dierks and C. Allen. The TLS protocol version 1.0. Request for Commnets: 2246, `http://www.ietf.org/rfc/rfc2246.txt`, 1999.
13. Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.
14. Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Trans. Info. Theory*, IT-29:198–208, 1983.
15. Jr. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 2001.
16. Steven Eker, José Meseguer, and Ambarish Sridharanarayanan. The Maude LTL model checker. In *4th WRLA*, volume 71 of *ENTCS*, pages 162–187. Elsevier, 2002.
17. Leo Freitas and Jim Woodcock. Mechanising Mondex with Z/Eves. *Formal Asp. Comput.*, 20(1):117–139, 2008.
18. Kokichi Futatsugi, Joseph A. Goguen, and Kazuhiro Ogata. Verifying design with proof scores. In *VSTTE 2005*, volume 4171 of *LNCS*, pages 277–290. Springer, 2008.
19. Chris George and Anne Elisabeth Haxthausen. Specification, proof, and model checking of the Mondex electronic purse using RAISE. *Formal Asp. Comput.*, 20(1):101–116, 2008.
20. Joseph Goguen. *Theorem Proving and Algebra*. The MIT Press, (to appear).
21. Joseph Goguen and Grant Malcolm. A hidden agenda. *TCS*, 245:55–101, 2000.
22. Joseph Goguen and Grant Malcolm, editors. *Software Engineering with OBJ: Algebraic Specification in Action*. Kluwer, 2000.
23. Dominik Haneberg, Gerhard Schellhorn, Holger Grandy, and Wolfgang Reif. Verification of Mondex electronic purses with KIV: from transactions to a security protocol. *Formal Asp. Comput.*, 20(1):41–59, 2008.

24. Nevin Heintze, J.D. Tygar, Jeannette Wing, and H. Chi Wong. Model checking electronic commerce protocols. In *2nd USENIX EC*, pages 147–164, 1996.
25. Tony Hoare and Jayadev Misra. Verified software: Theories, tools, experiments vision of a grand challenge project. In *1st VSTTE*, volume 4171 of *LNCS*, pages 1–18. Springer, 2008.
26. Gunther Horn and Bart Preneel. Authentication and payment in future mobile systems. In *5th ESORICS*, volume 1485 of *LNCS*, pages 277–293. Springer, 1998.
27. J. Hsiang and N. Dershowitz. Rewrite methods for clausal and nonclausal theorem proving. In *10th ICALP*, volume 154 of *LNCS*, pages 331–346. Springer, 1983.
28. Florent Jacquemard, Michael Rusinowitch, and Laurent Vigneron. Compiling and verifying security protocols. In *7th LPAR*, volume 1955 of *LNCS*, pages 131–160. Springer, 2000.
29. Cliff B. Jones and Jim Woodcock, editors. *Formal Aspects of Computing*. Number 1 in 20. Springer, 2008.
30. Weiqiang Kong, Kazuhiro Ogata, and Kokichi Futatsugi. Algebraic approaches to formal analysis of the mondex electronic purse system. In *6th IFM*, volume 4591 of *LNCS*, pages 393–412. Springer, 2007.
31. Weiqiang Kong, Kazuhiro Ogata, Takahiro Seino, and Kokichi Futatsugi. A lightweight integration of theorem proving and model checking for system verification. In *12th APSEC*, pages 59–66. IEEE CS Press, 2005.
32. Mirco Kuhlmann and Martin Gogolla. Modeling and validating Mondex scenarios described in UML and OCL with USE. *Formal Asp. Comput.*, 20(1):79–100, 2008.
33. Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *IPL*, 56:131–133, 1995.
34. Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *2nd TACAS*, volume 1055 of *LNCS*, pages 147–166. Springer, 1996.
35. Gavin Lowe. Casper: A compiler for the analysis of security protocols. In *10th IEEE CSFW*, pages 18–30, 1997.
36. Shiyong Lu and Scott A. Smolka. Model checking the Secure Electronic Transaction (SET) protocol. In *6th MASCOTS*, pages 358–365, 1999.
37. MasterCard/Visa. SET secure electronic transactions protocol – book 1: Business specifications; book 2: Technical specification; book 3: Formal protocol definition. `http://www.setco.org/set_specifications.html`, May 1997.
38. Kenneth L. McMillan. *Symolic Model Checking: An Approach to the State Explosion Problem*. Kluwer, 1993.
39. Jonathan K. Millen and Grit Denker. CAPSL and MuCAPSL. *J. Telecomm. & Info. Tech.*, 4:16–27, 2002.
40. John C. Mitchell, Vitaly Shmatikov, and Ulrich Stern. Finite-state analysis of SSL 3.0 and related protocols. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.
41. Masahiro Nakano, Kazuhiro Ogata, Masaki Nakamura, and Kokichi Futatsugi. Automatic verification of the STS authentication protocol with Crème. In *20th ITC-CSCC*, pages 15–16, 2005.
42. Masahiro Nakano, Kazuhiro Ogata, Masaki Nakamura, and Kokichi Futatsugi. Automating invariant verification of behavioral specifications. In *6th QSIC*, pages 49–56. IEEE CS Press, 2006.
43. Masahiro Nakano, Kazuhiro Ogata, Masaki Nakamura, and Kokichi Futatsugi. Crème: An automatic invariant prover of behavioral specifications. *IJSEKE*, 17(6):783–804, 2007.
44. Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in

large networks of computers. *CACM*, 21(12):993–999, 1978.
45. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
46. Kazuhiro Ogata and Kokichi Futatsugi. Flaw and modification of the *i*KP electronic payment protocols. *IPL*, 86:57–62, 2003.
47. Kazuhiro Ogata and Kokichi Futatsugi. Formal analysis of the *i*KP electronic payment protocols. In *1st ISSS*, volume 2609 of *LNCS*, pages 441–460. Springer, 2003.
48. Kazuhiro Ogata and Kokichi Futatsugi. Formal verification of the Horn-Preneel micropayment protocol. In *4th VMCAI*, volume 2575 of *LNCS*, pages 238–252. Springer, 2003.
49. Kazuhiro Ogata and Kokichi Futatsugi. Proof scores in the OTS/CafeOBJ method. In *6th FMOODS*, volume 2884 of *LNCS*, pages 170–184. Springer, 2003.
50. Kazuhiro Ogata and Kokichi Futatsugi. Equational approach to formal verification of SET. In *4th QSIC*, pages 50–59. IEEE CS Press, 2004.
51. Kazuhiro Ogata and Kokichi Futatsugi. Formal analysis of the NetBill electronic commerce protocol. In *2nd ISSS*, volume 3233 of *LNCS*, pages 45–64. Springer, 2004.
52. Kazuhiro Ogata and Kokichi Futatsugi. Equational approach to formal analysis of TLS. In *25th ICDCS*, pages 795–804. IEEE CS Press, 2005.
53. Kazuhiro Ogata and Kokichi Futatsugi. Some tips on writing proof scores in the OTS/CafeOBJ method. In *Algebra, Meaning, and Computation: A Festschrift Symposium in Honor of Joseph Goguen*, volume 4060 of *LNCS*, pages 596–615. Springer, 2006.
54. Kazuhiro Ogata and Kokichi Futatsugi. Proof score approach to verification of liveness properties. *IEICE Trans. Inf. & Syst.*, E91-D(12):2804–2817, 2008.
55. Kazuhiro Ogata, Masahiro Nakano, Masaki Nakamura, and Kokichi Futatsugi. Chocolat/SMV: A translator from CafeOBJ into SMV. In *6th PDCAT*, pages 416–420. IEEE CS Press, 2005.
56. D. Otway and O. Rees. Efficient and timely mutual authentication. *ACM Operating Systems Review*, 21(1):8–10, 1987.
57. Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *J. Comp. Security*, 6:85–128, 1998.
58. Lawrence C. Paulson. Inductive analysis of the internet protocol TLS. *ACM TISSEC*, 2(3):332–351, 1999.
59. Tahina Ramananandro. Mondex , an electronic purse: specification and refinement checks with the Alloy model-finding method. *Formal Asp. Comput.*, 20(1):21–39, 2008.
60. Takahiro Seino, Kazuhiro Ogata, and Kokichi Futatsugi. A toolkit for generating and displaying proof scores in the OTS/CafeOBJ method. In *6th RULE*, ENTCS. Elsevier, 2005.
61. Luca Viganò. Automated security protocol analysis with the AVISPA tool. In *21st MFPS*, volume 155 of *ENTCS*. Elsevier, 2005.
62. Jim Woodcock, Susan Stepney, David Cooper, John A. Clark, and Jeremy Jacob. The certification of the Mondex electronic purse to ITSEC Level E6. *Formal Asp. Comput.*, 20(1), 2008.