Trace Anonymity in the OTS/CafeOBJ Method

Weiqiang Kong, Kazuhiro Ogata, Jian Cheng, and Kokichi Futatsugi
Japan Advanced Institute of Science and Technology (JAIST)
1-1, Asahidai, Nomi, Ishikawa 923-1292, Japan
{weiqiang, ogata, kokichi}@jaist.ac.jp

Abstract

We report on a case study in which the OTS/CafeOBJ method is used to formalize and verify trace anonymity property of distributed systems. In this case study, the property of trace anonymity is formalized with the trace notations of Observational Transition Systems (OTSs), and CafeOBJ language/system is used as an interactive theorem prover to verify that systems satisfy such property. The work presented in the paper follows the approach proposed in [3], in which I/O automaton and Larch prover are employed for handling trace anonymity.

1. Introduction

The use of formal methods for safety property analysis has become standard practice. However, although there is an increasing concern about people's privacy, the use of formal methods for analysis of privacy related properties such as anonymity, is still in its elementary stage and only a few studies exist in the literature.

In an early study by Schneider *et al.* [7], a formal definition to anonymity, which is called *strong anonymity*, is proposed based on the trace notations of CSP. Basic principle behind the definition is that: an event that could have originated from one agent could equally have originated from any other (from a given set of agents). A CSP model checker FDR is then employed to analyze the satisfaction of finite-state systems to such anonymity.

To analyze anonymity property of more general infinite-state systems, Kawabe *et al.* extend in [3] the concept of strong anonymity to *trace anonymity*, which is defined in terms of the trace notations of I/O automaton while keeping the basic principle of viewing anonymity used in [7]. An inductive verification technique based on a notion of *anonymous simulation* is then proposed. It is shown that the existence of an anonymous simulation leads to trace anonymity. The formal verification that an infinite-state system satisfies trace anonymity is carried out using Larch prover.

In this paper, following the definition of trace anonymity and its inductive proof technique proposed in [3], we demonstrate how the OTS/CafeOBJ method [6] could be used for trace anonymity analysis. More specifically, the trace anonymity is formalized in terms of trace notations of Observational Transition Systems (OTSs), which are a kind of state transition systems that can be straightforwardly written in terms of equations. We then detail the definition of anonymous simulation that leads to trace anonymity of an OTS. At last, the satisfaction of infinite-state systems (modeled as OTSs) to trace anonymity is verified by using the CafeOBJ language/system as an interactive theorem prover.

The rest of the paper is organized as follows: Section 2 introduces the OTS/CafeOBJ method. Section 3 describes how to formalize trace anonymity with the trace notations of OTSs and the proof technique based on anonymous simulation. Section 4 demonstrates modeling and verification of trace anonymity for a simple example used in [3]. And Section 5 concludes the paper and mentions future work.

2. The OTS/CafeOBJ Method

The notion of Observational Transition Systems (OTSs) to be introduced in this section is an extended version of the original one described in [6]. We assume that there exists a universal state space denoted by Υ, and each data type (including Bool for Boolean values) used is provided. A data type is denoted by D with a subscript such as D_{o1}.

Definition 1 *An OTS $\mathcal{S}$ is $\langle \mathcal{O}, \mathcal{I}, \mathcal{A}, \mathcal{T} \rangle$, where:*

- *$\mathcal{O}$: A finite set of observers. Each observer is an indexed function $o_{x_1:D_{o1},\ldots,x_m:D_{om}} : \Upsilon \rightarrow D_o$. Given an OTS $\mathcal{S}$ and two states $\upsilon_1, \upsilon_2 \in \Upsilon$, the equivalence (denoted by $\upsilon_1 =_{\mathcal{S}} \upsilon_2$) between them wrt $\mathcal{S}$ is defined as $\forall o_{x_1:D_{o1},\ldots,x_m:D_{om}} : \mathcal{O}.\ \forall x_1 : D_{o1} \ldots \forall x_m : D_{om}.\ (o_{x_1,\ldots,x_m}(\upsilon_1) = o_{x_1,\ldots,x_m}(\upsilon_2))$.*
- *$\mathcal{I}$: A set of initial states such that $\mathcal{I} \subseteq \Upsilon$.*
- *$\mathcal{A}$: A finite set of actions (indexed names). Actions are classified into a set of external actions $\mathcal{A}_E$ and a set of internal actions $\mathcal{A}_I$, and $\mathcal{A}_E \cap \mathcal{A}_I = \emptyset$.*

978-1-4244-2358-3/08/$20.00 © 2008 IEEE

- $\mathcal{T}$ *: A finite set of conditional transitions. Each transition is a nondeterministic function*[1] $t_{y_1:D_{t1},\ldots,y_n:D_{tn}} : \Upsilon \rightarrow \Upsilon$ *with an indexed action in* $\mathcal{A}$ *as its name.* $t_{y_1,\ldots,y_n}(\upsilon)$ *for each* $\upsilon \in \Upsilon$ *is called a successor state of* υ *wrt* $t_{y_1,\ldots,y_n}$*. The condition* $c_{t_{y_1,\ldots,y_n}}$ *for a transition associated with action* $t_{y_1,\ldots,y_n} \in \mathcal{A}$*, which is a predicate on states, is called the effective condition. If* $c_{t_{y_1,\ldots,y_n}}$ *does not hold in* υ*, then* $t_{y_1,\ldots,y_n}(\upsilon) =_{\mathcal{S}} \upsilon$*.*

For brevity, we may omit the indexes of observers and actions by assuming that their names (without indexes) are distinct from each other. A transition in $\mathcal{T}$, in which application of an action t moves state υ_1 to υ_2 (or other states nondeterministically, say υ_3), can be written as $\upsilon_1 \rightarrow^t_{\mathcal{S}} \upsilon_2$ (or $\upsilon_1 \rightarrow^t \upsilon_3$) and t could be omitted if $t \in \mathcal{A}_I$. The subscript $\mathcal{S}$ could also be omitted if it is clear from the context. $\rightsquigarrow_{\mathcal{S}}$ is a reflexive transitive closure of $\rightarrow_{\mathcal{S}}$. We write $\upsilon_1 \Rightarrow^t_{\mathcal{S}} \upsilon_2$ for multiple-time applications of actions moving υ_1 to υ_2 and among the applied actions there is one $t \in \mathcal{A}_E$, or for $\upsilon_1 \rightsquigarrow_{\mathcal{S}} \upsilon_2$ if no external actions are applied.

Definition 2 *Given an OTS* $\mathcal{S}$*, reachable states wrt* $\mathcal{S}$ *are defined as: (1) Each* $\upsilon_0 \in \mathcal{I}$ *is reachable wrt* $\mathcal{S}$*. (2) For each* $\upsilon, \upsilon' \in \Upsilon$*, and some* $t \in \mathcal{A}$ *such that* $\upsilon \rightarrow^t_{\mathcal{S}} \upsilon'$ *is a transition in* $\mathcal{T}$*, if* υ *is reachable wrt* $\mathcal{S}$*, so is* υ'*. The set of all reachable states of* $\mathcal{S}$ *is denoted by* $\mathcal{R}_{\mathcal{S}}$*.*

Definition 3 *Given an OTS* $\mathcal{S}$*, transition sequences wrt* $\mathcal{S}$ *are defined as: (1) Each* $\upsilon_0 \in \mathcal{I}$ *is a transition sequence. (2) For an arbitrary transition sequence* α*, if there is a transition* $last(\alpha) \rightarrow^t_{\mathcal{S}} \upsilon$*, then* $\alpha \rightarrow^t_{\mathcal{S}} \upsilon$ *is a transition sequence. The function* ***last*** *over transition sequences is defined as: (1)* $last(\upsilon_0) = \upsilon_0$*, and (2)* $last(\alpha \rightarrow^t_{\mathcal{S}} \upsilon) = \upsilon$*. The set of all transition sequences of* $\mathcal{S}$ *is denoted by* $\mathcal{TS}_{\mathcal{S}}$*.*

Definition 4 *Given an OTS* $\mathcal{S}$*, traces wrt* $\mathcal{S}$ *are defined by a function* ***trace*** *over transition sequences as:* $\{trace(ts) \mid ts \in \mathcal{TS}_{\mathcal{S}}\}$*. The function* ***trace*** *is defined as: (1)* $trace(\upsilon_0) = \epsilon$*, where* ϵ *denotes an empty trace, and* $\upsilon_0 \in \mathcal{I}$*, and (2)* $trace(\alpha \rightarrow^t_{\mathcal{S}} \upsilon)$ = (**if** $t \in \mathcal{A}_E$ **then** $trace(\alpha), t$ **else** $trace(\alpha)$*. The set of all traces of* $\mathcal{S}$ *is denoted by* $Traces(\mathcal{S})$*.*

In the OTS/CafeOBJ method, an OTS is described in CafeOBJ [1]. CafeOBJ is an algebraic specification language and system mainly based on order-sorted algebras and hidden algebras. Data types can be specified in terms of order-sorted algebras, and state machines such as OTSs can be specified in terms of hidden algebras. A CafeOBJ visible sort denotes an abstract data type, and a hidden sort denotes the state space of an abstract state machine. There are two kinds of operators in hidden sorts: action and observation operators. An action operator can change a state of an abstract state machine; only observation operators can be used to observe the inside of an abstract state machine. Declarations of observation and action operators start with `bop`, and those of other operators with `op`. Operators are defined in equations. Declarations of equations start with `eq`, and those of conditional equations with `ceq`. The CafeOBJ system rewrites a given term by regarding equations as left-to-right rewrite rules.

The universal state space Υ is denoted by a hidden sort, say *H*. An observer $o_{x_1,\ldots,x_m}$ is denoted by a CafeOBJ observation operator. We assume that there exist visible sorts V_k and *V* denoting D_k and *D*, where $k = o1, \ldots, om$. The CafeOBJ observation operator is declared as `bop` o : $H\ V_{o1} \ldots V_{om}$ `->` V .

An action $t_{y_1,\ldots,y_n}$ is denoted by a CafeOBJ action operator. We assume that there exists a visible sort V_k denoting D_k, where $k = t1, \ldots, tn$. The CafeOBJ action operator is declared as `bop` t : $H\ V_{t1} \ldots V_{tn}$ `->` H. A transition associated with the action $t_{y_1,\ldots,y_n}$ is described by a conditional equation in a form of changing the value returned by $o_{x_1,\ldots,x_m}$ when $c_{t_{y_1,\ldots,y_n}}(\upsilon)$ holds (note that in the following equation we essentially only consider the transition as a deterministic one):

$$
\begin{aligned}
&\texttt{ceq}\ o(t(S, X_{y_1}, \ldots, X_{y_n}), X_{x_1}, \ldots, X_{x_m}) \\
&\qquad = \text{e-}t(S, X_{y_1}, \ldots, X_{y_n}, X_{x_1}, \ldots, X_{x_m}) \\
&\qquad\quad \texttt{if}\ c\text{-}t(S, X_{y_1}, \ldots, X_{y_n})\ .
\end{aligned}
$$

S is a CafeOBJ variable of sort *H* and all the *X*s being as parameters of *o* and t are CafeOBJ variables of corresponding visible sorts. $t(S, X_{y_1}, \ldots, X_{y_n})$ denotes the successor state of *S* wrt $t_{y_1,\ldots,y_n}$. e-$t(S, X_{y_1}, \ldots, X_{y_n}, X_{x_1}, \ldots, X_{x_m})$ denotes the value returned by $o_{x_1,\ldots,x_m}$ in the successor state. c-$t(S, X_{y_1}, \ldots, X_{y_n})$ denotes the effective condition $c_{t_{y_1,\ldots,y_n}}(\upsilon_1)$. The value returned by $o_{x_1,\ldots,x_m}$ is not changed if $\neg c_{\tau_{y_1,\ldots,y_n}}(\upsilon_1)$, and this is described as $t(S, X_{y_1}, \ldots, X_{y_n}) = S$.

We will illustrate the verification technique of the OTS/CafeOBJ method in Section 4 with an example.

3. Trace Anonymity

In this section, following the approach described in [3], we demonstrate how to use the trace notations of OTSs to formalize trace anonymity and the proof technique based on a notion of anonymous simulation.

3.1 Formalization of Trace Anonymity

According to the IT security functional requirements formulated by ISO/IEC [2], the notion of anonymity ensures that a user may use a resource or service without disclosing the user's identity. Although this informal requirement/definition is enough for us to understand the meaning

[1] This nondeterminism is only for explanation purpose. The transition function that we essentially specified in CafeOBJ is a deterministic one.

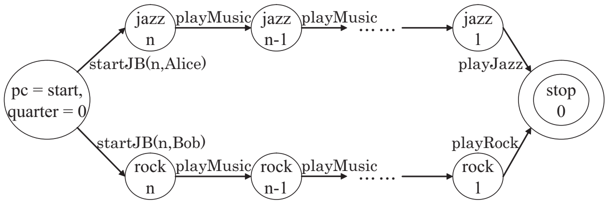

Figure 1. *Jukebox*

of anonymity, it is difficult, due to the non-functional essentials, to propose a formal definition of anonymity that could serve as the basis for formal analysis.

The definition of trace anonymity described in [3] circumvents the problem by following a principle called "principle of confusion" [7, 5]: a system is anonymous if one user can cause an observable trace, then it is possible for the other users (from a given set of users) to cause the same trace (modulo special actions with regard to a user's identity). Therefore an intruder (or outside observer) of a system could not distinguish the difference between the given set of users' behaviors. This given set of users is also called anonymity group in [5], which is to be determined and introduced by system analyzers. To formalize trace anonymity based on this principle, we first define the concept of *family of actor actions* (corresponds to anonymity group) with which the trace anonymity is discussed.

Definition 5 *Given an OTS $\mathcal{S}$, $\mathbb{A}$ is called a family of $\mathcal{S}$'s actor actions if the following conditions hold: (1) $\bigcup_{A'\in\mathbb{A}} A' \subset \mathcal{T}_E$, (2) $A' \cap A'' = \emptyset$ for any $A', A'' \in \mathbb{A}$. An element of $\mathbb{A}$ is called a set of actor actions.*

Definition 6 *Given an OTS $\mathcal{S} = \langle \mathcal{O}_\mathcal{S}, \mathcal{I}_\mathcal{S}, \mathcal{A}_\mathcal{S}, \mathcal{T}_\mathcal{S} \rangle$, and a family of actor actions $\mathbb{A}$, an "anonymized" OTS $\mathcal{S}_\mathbb{A} = \langle \mathcal{O}_{\mathcal{S}_\mathbb{A}}, \mathcal{I}_{\mathcal{S}_\mathbb{A}}, \mathcal{A}_{\mathcal{S}_\mathbb{A}}, \mathcal{T}_{\mathcal{S}_\mathbb{A}} \rangle$ wrt $\mathbb{A}$ is defined as follows:*

(1) $\mathcal{O}_{\mathcal{S}_\mathbb{A}} = \mathcal{O}_\mathcal{S}$, $\mathcal{I}_{\mathcal{S}_\mathbb{A}} = \mathcal{I}_\mathcal{S}$, and $\mathcal{A}_{\mathcal{S}_\mathbb{A}} = \mathcal{A}_\mathcal{S}$,

(2) For any transition $v_1 \to_{\mathcal{S}}^{t} v_2$ in $\mathcal{T}_\mathcal{S}$, (2.1) if $t \in A$ for some $A \in \mathbb{A}$, then $v_1 \to_{\mathcal{S}_\mathbb{A}}^{t'} v_2$ for any $t' \in A$ is a transition of $\mathcal{T}_{\mathcal{S}_\mathbb{A}}$, (2.2) otherwise if $t \notin \bigcup_{A\in\mathbb{A}} A$, then $v_1 \to_{\mathcal{S}_\mathbb{A}}^{t} v_2$ is a transition of $\mathcal{T}_{\mathcal{S}_\mathbb{A}}$.

Note that the (external and internal) attribution of $\mathcal{S}_\mathbb{A}$'s actions remains the same as the one of $\mathcal{S}$. And besides, we can deduce that $\mathcal{R}_{\mathcal{S}_\mathbb{A}} = \mathcal{R}_\mathcal{S}$.

Definition 7 *Given an OTS $\mathcal{S}$, and a family of actor actions $\mathbb{A}$, $\mathcal{S}$ is called trace anonymous wrt $\mathbb{A}$ if $Traces(\mathcal{S}_\mathbb{A}) = Trace(\mathcal{S})$.*

We use the same example *Jukebox* system described in [3] to explain the idea of trace anonymity. Assume that there is an electronic jukebox placed in a building that plays n music to the public if someone inserts n quarters. There are two persons – Alice and Bob, who are going to insert coins anonymously, namely that they do not want others to know who inserted the coins. If Alice inserts the coins, she would choose songs randomly, but choose the title `jazz` for the last song for her favorite genre; and if Bob inserts the coins, he would also choose songs randomly, but choose the title `rock` for the last song. The behavior of $Jukebox$ is shown in Figure 1, in which the values of the state variable `pc` denote the status of the jukebox wrt the category of songs it is playing, and the values of the variable `quarter` denote the coins currently remained. A formula attached to an arrow denotes an external action applied between two states.

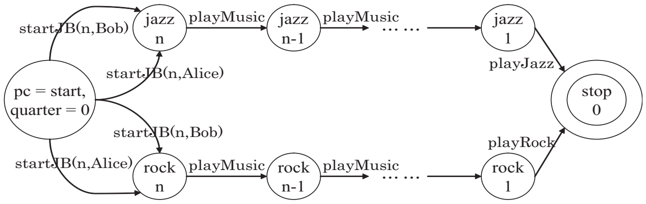

Figure 2. $Jukebox_\mathbb{A}$

The family of actor actions $\mathbb{A} = \{\{startJB(n, Alice), startJB(n, Bob)\} \mid n \in \{1, 2, \ldots\}\}$ is introduced to analyze trace anonymity of the jukebox system. The behavior of the "anonymized" jukebox $Jukebox_\mathbb{A}$ wrt $\mathbb{A}$ is shown in Figure 2. We can see that the jukebox system is not trace anonymous wrt $\mathbb{A}$ because the trace "$startJB(n, Bob), playMusic, \ldots, playJazz$" of $Jukebox_\mathbb{A}$ is not a trace of $Jukebox$, and therefore $Traces(Jukebox_\mathbb{A}) \neq Traces(Jukebox)$.

An observation obtained from the unsatisfiability of trace anonymity of $Jukebox$ is that: the occurrence of the actions $playJazz$ and $playRock$ makes it possible for the intruder to deduce the identity of the person who inserts the coins. We now hide these two actions by considering them as internal actions that could not be observed by the intruder. This jukebox system is named as $\overline{Jukebox}$. It can be easily seen that $\overline{Jukebox}$ is trace anonymous wrt $\mathbb{A}$ because $Traces(\overline{Jukebox}_\mathbb{A}) = Traces(\overline{Jukebox})$.

3.2 Proof Technique for Trace Anonymity

We now demonstrate how to use OTSs' notations to formalize the proof technique for trace anonymity described in [3], which is based on a notion of anonymous simulation.

Given an OTS $\mathcal{S}$ and a family of actor actions $\mathbb{A}$, to prove that $\mathcal{S}$ is trace anonymous wrt $\mathbb{A}$, we could show that (1) $Traces(\mathcal{S}_\mathbb{A}) \subseteq Traces(\mathcal{S})$ and then (2) $Traces(\mathcal{S}) \subseteq Traces(\mathcal{S}_\mathbb{A})$, and thus $Traces(\mathcal{S}_\mathbb{A}) = Traces(\mathcal{S})$. Item (2) holds trivially since $\mathcal{S}_\mathbb{A}$ contains all the transitions of $\mathcal{S}$. As to the proof of item (1), Lynch *et al.* has proved a theorem in [4] that (1) holds if there exists a forward simulation from $\mathcal{S}_\mathbb{A}$ to $\mathcal{S}$. To directly prove the existence of such forward simulation using some formal verification tools, $\mathcal{S}_\mathbb{A}$ and $\mathcal{S}$ should be firstly specified. However, reasoning on $\mathcal{S}_\mathbb{A}$ may be problematic due to the in-confluent problem caused by $\mathcal{S}_\mathbb{A}$'s non-deterministic feature (different successor states may be reached from a given state by applying one same action). This problem is circumvented in [3] by introducing the notion of *anonymous simulation*.

Definition 8 *Given an OTS $\mathcal{S}$ and a family of actor actions $\mathbb{A}$, $r : \mathcal{R}_{\mathcal{S}}\ \mathcal{R}_{\mathcal{S}} \rightarrow Bool$ is called an anonymous simulation of $\mathcal{S}$ on $\mathbb{A}$ if it satisfies the following conditions: (1) $r(\upsilon_0, \upsilon_0)$ holds for any $\upsilon_0 \in \mathcal{I}$. (2) For each $\upsilon_1, \upsilon_2, \upsilon_1' \in \mathcal{R}_{\mathcal{S}}$ such that $r(\upsilon_1, \upsilon_1')$ and $\upsilon_1 \rightarrow_{\mathcal{S}}^{t} \upsilon_2$:*

- *(i) If $t \in A$ for some $A \in \mathbb{A}$, then for all $t' \in A$ there exists $\upsilon_2' \in \mathcal{R}_{\mathcal{S}}$ such that $r(\upsilon_2, \upsilon_2')$ and $\upsilon_1' \Rightarrow_{\mathcal{S}}^{t'} \upsilon_2'$;*
- *(ii) If $t \notin A$ for any $A \in \mathbb{A}$: (a) if $t \in \mathcal{T}_E$, then there exists $\upsilon_2' \in \mathcal{R}_{\mathcal{S}}$ such that $r(\upsilon_2, \upsilon_2')$ and $\upsilon_1' \Rightarrow_{\mathcal{S}}^{t} \upsilon_2'$; (b) If $t \in \mathcal{T}_I$, then there exists $\upsilon_2' \in \mathcal{R}_{\mathcal{S}}$ for some $t' \in \mathcal{T}_I$ such that $r(\upsilon_2, \upsilon_2')$ and $\upsilon_1' \Rightarrow_{\mathcal{S}}^{t'} \upsilon_2'$ (or $\upsilon_1' \rightsquigarrow_{\mathcal{S}} \upsilon_2'$ in this situation where no external transitions).*

A theorem has been proved in [3] that an anonymous simulation $r : \mathcal{R}_{\mathcal{S}_{\mathbb{A}}} \mathcal{R}_{\mathcal{S}} \rightarrow Bool$ (note that $\mathcal{R}_{\mathcal{S}_{\mathbb{A}}} = \mathcal{R}_{\mathcal{S}}$) is a forward simulation from $\mathcal{S}_{\mathbb{A}}$ to $\mathcal{S}$. Therefore, if there exists an anonymous simulation of $\mathcal{S}$ on $\mathbb{A}$, then $Traces(\mathcal{S}_{\mathbb{A}}) \subseteq Traces(\mathcal{S})$ holds, and thus $\mathcal{S}$ is trace anonymous on $\mathbb{A}$. In this paper, we omit the proof for this theorem, and the idea of the proof follows the one in [3]. By introducing the notion of anonymous simulation, we do not need to explicitly construct $\mathcal{S}_{\mathbb{A}}$ when reasoning about trace anonymity of an OTS $\mathcal{S}$ wrt $\mathbb{A}$, but only need to work on $\mathcal{S}$.

4. Formal Analysis of the Jukebox System

In this section, we demonstrate how to analyze trace anonymity of $\overline{Jukebox}$ using the OTS/CafeOBJ method.

4.1 Modeling and Specification

$\overline{Jukebox}$ is firstly modeled as an OTS $\mathcal{S}_{jb}$. The data types used in $\mathcal{S}_{jb}$ are: (1) `Bool` for Boolean values, (2) `Nat` for Natural numbers representing the value of coins inserted, (3) `Label` for the values of program counters (pc). `start`, `jazz`, `rock` and `stop` are declared as constants of `Label`, (4) `Pid` for users' identities of the jukebox system, and `Alice` and `Bob` are declared as constants of `Pid`.

$\mathcal{S}_{jb}$ is $\langle \mathcal{O}_{jb}, \mathcal{I}_{jb}, \mathcal{A}_{jb}, \mathcal{T}_{jb} \rangle$ such that:

$$\mathcal{O}_{jb} \triangleq \{pc : \Upsilon \rightarrow Label, quarter : \Upsilon \rightarrow Nat\}$$

$$\mathcal{I}_{jb} \triangleq \{\upsilon_{init} \in \Upsilon \mid pc(\upsilon_{init}) = start \wedge quarter(\upsilon_{init}) = 0\}$$

$$\mathcal{A}_{jb} \triangleq \mathcal{A}_{E_{jb}} \cup \mathcal{A}_{I_{jb}} \text{ where,}$$

$$\mathcal{A}_{E_{jb}} \triangleq \{startJB_{n:Nat,p:Pid},\ playMusic\}$$

$$\mathcal{A}_{I_{jb}} \triangleq \{playJazz,\ playRock\}$$

$$\mathcal{T}_{jb} \triangleq \{startJB_{n:Nat,p:Pid} : \Upsilon \rightarrow \Upsilon,$$
$$playMusic : \Upsilon \rightarrow \Upsilon,$$
$$playJazz : \Upsilon \rightarrow \Upsilon,\ playRock : \Upsilon \rightarrow \Upsilon\}$$

The four transitions modeling the behavior of the jukebox system are defined as follows:

- $c_{startJB_{n,p}}(\upsilon) \triangleq pc(\upsilon) = start \wedge quarter(\upsilon) = 0$.
  If $c_{startJB_{n,p}}(\upsilon)$, then $startJB_{n,p}(\upsilon) \triangleq \upsilon'$ such that $pc(\upsilon') \triangleq$ **if** $p = Alice$ **then** $jazz$ **else** $rock$, $quarter(\upsilon') \triangleq n$.
- $c_{playMusic}(\upsilon) \triangleq (pc(\upsilon) = jazz \vee pc(\upsilon) = rock) \wedge quarter(\upsilon) > 1$.
  If $c_{playMusic}(\upsilon)$, then $playMusic(\upsilon) \triangleq \upsilon'$ such that $pc(\upsilon') \triangleq pc(\upsilon),\ \ quarter(\upsilon') \triangleq quarter(\upsilon) - 1$.
- $c_{playJazz}(\upsilon) \triangleq (pc(\upsilon) = jazz \wedge quarter(\upsilon) = 1$.
  If $c_{playJazz}(\upsilon)$, then $playJazz(\upsilon) \triangleq \upsilon'$ such that $pc(\upsilon') \triangleq\ stop,\ \ quarter(\upsilon') \triangleq 0$.
- $c_{playRock}(\upsilon) \triangleq (pc(\upsilon) = rock \wedge quarter(\upsilon) = 1$.
  If $c_{playRock}(\upsilon)$, then $playRock(\upsilon) \triangleq \upsilon'$ such that $pc(\upsilon') \triangleq\ stop,\ \ quarter(\upsilon') \triangleq 0$.

$\mathcal{S}_{jb}$ is specified in CafeOBJ. Basic building blocks of CafeOBJ specifications are modules. We assume that the data types corresponding to the sorts `Bool`, `Nat`, `Label` and `Pid` have been defined. $\mathcal{S}_{jb}$ is specified as a module `JUKEBOX`. The signature of the module is as follows:

```
 op init      : -> Sys
bop pc        : Sys -> Label
bop quarter   : Sys -> Nat
bop startJB   : Sys Nat Pid -> Sys
bop playMusic : Sys -> Sys
bop playJazz  : Sys -> Sys
bop playRock  : Sys -> Sys
```

`Sys` is the hidden sort denoting the state space Υ. Constant `init` denotes an arbitrary initial state of $\mathcal{S}_{jb}$. The two observation operators correspond to the observers, and the four action operators correspond to the actions. In this paper, we show the CafeOBJ specifications for `init` and transitions associated with `startJB` as demonstration examples, where "`--`" denotes a line of comments:

```
-- for any initial state init
eq pc(init) = start .
eq quarter(init) = 0 .

-- for transitions associated with startJB
op c-startJB : Sys Nat Pid -> Bool
eq c-startJB(S,N,P)
   = (pc(S) = start and quarter(S) = 0) .
--
ceq pc(startJB(S,N,P))
    = (if P = Alice then jazz else rock fi)
    if c-startJB(S,N,P) .
ceq quarter(startJB(S,N,P))
    = N if c-startJB(S,N,P) .
ceq startJB(S,N,P) = S if not c-startJB(S,N,P) .
```

4.2 Verification of Trace Anonymity

Basic steps for verifying trace anonymity wrt the family of actor actions $\mathbb{A}$ is first to define a candidate anonymous simulation r of $\mathcal{S}_{jb}$, and then to prove that r satisfies the conditions defined in Definition 8.

We declare a module SIM, which imports module JUKEBOX. In module SIM, the following operator and equation are declared for defining a candidate anonymous simulation r:

```
op r : Sys Sys -> Bool
eq r(A:Sys,B:Sys)
   = quarter(A) = quarter(B) and
     (pc(A) = pc(B)
      or (pc(A) = jazz and pc(B) = rock)
      or (pc(A) = rock and pc(B) = jazz)) .
```

In the module SIM, we also declare constants s1, s1', s2 and s2' of sort Sys to denote arbitrary states of $\mathcal{S}_{jb}$. We next declare a module STEP, which imports module SIM. In module STEP, the following operator and equation are declared:

```
op step-r : -> Bool
eq step-r = r(s1,s1') implies r(s2,s2') .
```

We start now to prepare proofs (or proof scores) to check the satisfaction of r to those conditions. A proof passage (basic fragments of a proof score) for checking the condition of (1) of Definition 8 (for simplicity, we write D8.1 and so on in the following description) is written as follows:

```
open SIM
 red r(init,init) .
close
```

The CafeOBJ command open constructs a temporary module that imports a given module (module SIM in the above proof passage), and CafeOBJ command close destroys such a temporary module. CafeOBJ command red reduces (via term rewriting) a term denoting a proposition to its truth value by considering equations available in this temporary module as left-to-right rewrite rules. CafeOBJ system returns true for this proof passage, which means that the condition D8.1 holds.

To check the satisfaction of r to condition D8.2, the constants s1, s1', s2 and s2' declared in SIM are used. By assuming the premise of D8.2, namely that r(s1,s1') and an action (named as t) moving s1 to s2, we conduct case-splitting on t to check if there exists state s2', which makes (i) and (ii) of D8.2 hold. A template for writing proof passages for such cases is as follows:

```
open STEP
   (1) Declare constants to be used as parameters of $t$ and $t'$.
   (2) Declare equations denoting that $c\text{-}t(s_1)$ holds or does not hold.
   (3) eq s2 = $t$(s1) .
   (4) eq s2' = $t'$(s1') .
   (5) red step-r .
close
```

Note that in the above template, we omit the parameters of actions t and t'. For each such case, we usually split it into multiple subcases with further basic predicates. We now start the case-splitting on t.

case 1: $t = startJB_{n,Alice}$.
Since $t \in \{startJB(n, Alice), startJB(n, Bob)\}$ of $\mathbb{A}$, we need to consider two situations where: (1) $t' = startJB_{n,Alice}$ and (2) $t' = startJB_{n,Bob}$. Situation (1) is split into three subcases based on two basic predicates:

bp1 $\triangleq$ pc(s1') = start, bp2 $\triangleq$ quarter(s1') = 0.

The subcases correspond to: ¬bp1, bp1 ∧ bp2, and bp1 ∧ ¬bp2, respectively. We show the proof passage for subcase ¬bp1 as an example in this paper:

```
open STEP
 op n : -> Nat .
-- eq c-startJB(s1,n,Alice) = true .
 eq pc(s1) = start .    eq quarter(s1) = 0 .
 --
 eq s2 = startJB(s1,n,Alice) .
-- basic predicate: not bp1
 eq (pc(s1') = start) = false .
-- check if there exists s2'
 eq s2' = startJB(s1',n,Alice) .
 red step-r .
close
```

CafeOBJ system returns true for all these three proof passages. As to situation (2), it is also split into three subcases based on the same basic predicates used for situation (1). We show the proof passage for subcase bp1 ∧¬bp2 as an example as follows:

```
open STEP
 op n : -> Nat .
-- eq c-startJB(s1,n,Alice) = true .
 eq pc(s1) = start .    eq quarter(s1) = 0 .
 --
 eq s2 = startJB(s1,n,Alice) .
-- basic predicate: bp1 and not bp2
 eq pc(s1') = start .
 eq (quarter(s1') = 0) = false .
-- check if there exists s2'
 eq s2' = startJB(s1',n,Bob) .
 red step-r .
close
```

CafeOBJ system returns true for all the three subcases of situation (2).

case 2: $t = startJB_{n,Bob}$.
We omit the description of this case since it is very similar to **case 1**.

case 3: $t = playMusic$.
Since t is not in any set of $\mathbb{A}$ and $t \in \mathcal{T}_E$, we only need to check the situation that $t' = playMusic$. This case is split into ten subcases based on three basic predicates:

bp1 $\triangleq$ `pc(s1') = jazz`, bp2 $\triangleq$ `pc(s1') = rock`,
bp3 $\triangleq$ `quarter(s1') = quarter(s1)`.

The first five subcases assume bp1 in `c-playMusic`, while the other five assume bp2. Each of the two five subcases correspond to: ¬bp1 ∧ ¬bp2, ¬bp1 ∧ bp2 ∧ ¬bp3, ¬bp1 ∧ bp2 ∧ bp3, bp1 ∧ ¬bp3, and bp1 ∧ bp3, respectively. We show the proof passage for subcase ¬bp1 ∧ ¬bp2 while assuming bp1 in `c-playMusic` as an example:

```
open STEP
-- eq c-playmusic(s1) = true .
 eq pc(s1) = jazz . eq (quarter(s1) > 1) = true .
 --
 eq s2 = playmusic(s1) .
-- basic predicate: not bp1 and not bp2
 eq (pc(s1') = jazz) = false .
 eq (pc(s1') = rock) = false .
-- check if there exists s2'
 eq s2' = playMusic(s1') .
 red step-r .
close
```

case 4: $t = playJazz$.
Since $t \in \mathcal{T}_I$, we need to check the situation that t' might be any transition in $\mathcal{T}_I$, namely $playJazz$ or $playRock$. This case is split into five subcases based on three basic predicates:

bp1 $\triangleq$ `pc(s1') = jazz`, bp2 $\triangleq$ `quarter(s1') = 1`,
bp3 $\triangleq$ `pc(s1') = rock`.

The five subcases correspond to: bp1 ∧ bp2, bp1 ∧¬ bp2, ¬ bp1 ∧ ¬bp2, ¬bp1 ∧ bp2 ∧ ¬bp3, ¬bp1 ∧ bp2 ∧ bp3, respectively. We show the proof passages for subcases bp1 ∧ bp2 and ¬bp1 ∧ bp2 ∧ bp3 as follows:

```
open STEP .
-- eq c-playJazz(s1) = true .
 eq pc(s1) = jazz .    eq quarter(s1) = 1 .
 --
 eq s2 = playJazz(s1) .
-- basic predicate: bp1 and bp2
 eq pc(s1') = jazz .    eq quarter(s1') = 1 .
-- check if there exists s2'
 eq s2' = playJazz(s1') .
 red step-r  .
close
```

```
open STEP .
-- eq c-playJazz(s1) = true .
 eq pc(s1) = jazz .    eq quarter(s1) = 1 .
 --
 eq s2 = playJazz(s1) .
-- basic predicate: not bp1 and bp2 and bp3
 eq (pc(s1') = jazz) = false .
 eq quarter(s1') = 1 .    eq pc(s1') = rock .
 -- check if there exists s2'
 eq s2' = playRock(s1') .
 red step-r  .
close
```

Note that in the above two proof passages, internal actions $playJazz$ and $playRock$ are used as t' respectively. This point is not detailed in [3]. CafeOBJ system returns `true` for all the above proof passages.

case 5: $t = playRock$.
We omit the description of this case since it is very similar to **case 4**.

5. Conclusion

In this paper, following the approach proposed in [3], we demonstrated how to use the OTS/CafeOBJ method to formalize and verify trace anonymity of infinite-state systems. This work is a starting point for our further research on formal analysis of anonymity, and more broadly, formal analysis of privacy related properties.

We have noticed that in the proof technique described above, coming up with the anonymous simulation relation r is a non-trivial task. If anonymity can be formalized as an invariant property, then the already well-studied proof technique of the OTS/CafeOBJ method for invariants could be directly used for the analysis of anonymity. This leads to our future work of proposing a way to formalize anonymity as invariant properties, which could be inductively verified by the OTS/CafeOBJ method.

Acknowledgements

This research is conducted as a program for the "21st Century COE Program" in Special Coordination Funds for promoting Science and Technology by Ministry of Education, Culture, Sports, Science and Technology. We would like to thank Yoshinobu Kawabe for the discussion in JAIST about his anonymity work.

References

[1] CafeOBJ web site. http://www.ldl.jaist.ac.jp/cafeobj/.
[2] ISO. Information technology – Security techniques – Evaluation criteria for IT security. ISO/IEC 15408-2, 2005.
[3] Y. Kawabe, K. Mano, H. Sakurada, and Y. Tsukada. Theorem-proving anonymity of infinite-state systems. *Information Processing Letterss*, 101(2007):46–51, 2007.
[4] N. Lynch and F. Vaandrager. Forward and backward simulations part i: Untimed system. *Information and Computation*, 121(2):214–233, 1995.
[5] S. Mauw, J. Verschuren, and E. P. de Vink. A formalization of anonymity and onion routing. In *ESORICS 2004*, volume 3193 of *LNCS*, pages 109–124. Springer, 2004.
[6] K. Ogata and K. Futatsugi. Proof scores in the OTS/CafeOBJ method. In *FMOODS 2003*, volume 2884 of *LNCS*, pages 170–184. Springer, 2003.
[7] S. Schneider and A. Sidiropoulos. CSP and anonymity. In *ESORICS 1996*, volume 1146 of *LNCS*, pages 198–218. Springer, 1996.