Regular Paper

How to Theorem-Prove Trace-Based Safety Properties

Toshinori Fukunaga[1], Hideki Goromaru[1], Tadanori Mizuno[2], Kazuhiko Ohkubo[1], and Yoshinobu Kawabe[2]

[1]NTT Secure Platform Laboratories, NTT Corporation
3-9-11 Midori-cho, Musashino-shi, Tokyo 180-0012, Japan
E-mail : { toshinori.fukunaga.vf, hideki.goroumaru.mx }@hco.ntt.co.jp,
ohkubo.kazuhiko@lab.ntt.co.jp
[2]Department of Information Science, Aichi Institute of Technology
Yachigusa 1247, Yakusa-cho, Toyota, Aichi 470-0392, Japan
E-mail : { tmizuno, kawabe }@aitech.ac.jp

Abstract - Recently, it is getting more important to analyze the trust of information and the trust of information sources. This is because information exchanged in social media may not be trustable; for example, even though you receive a message which is consistent with other messages, you might be skeptical if the message is sent from an unknown sender. Thus, we need to analyze trust values and their transitions. In this paper, we discuss how to analyze the transitions of two-dimensional trust values. Specifically, after formalizing a time-related trust safety property, this paper introduces an efficient verification method for the property. By conducting a case study with a theorem-proving tool, we show the applicability of our proof method.

Keywords: On-Line Trust, I/O-automaton, Safety Properties, Theorem-Proving

1 INTRODUCTION

In recent years, social media is actively used in large-scale disasters such as earthquakes and typhoons, and safety information and relief information are actively exchanged in social media. However, such information is not always trustable. Some information may be incorrect, and the incorrect information may be deliberately distributed in the wake of disasters. Moreover, the correctness of information may change as time passes. Even if the information "A person is seriously injured but currently alive," is true in a disaster site, it may become false one hour later. When dealing with such information in social networks, it is important to evaluate the trust of messages and the trust of information sources.

Marsh and Dibben introduced an evaluation on trust [1], which is a value between -1 and 1. In addition, the trust values are classified with the notions of trust, distrust and untrust; also, another notion called mistrust is introduced. As for these properties, there are studies [2] (on distrust and mistrust) and [3] (on trust and mistrust) by Primiero et al. This classification is based on the one-dimensional definition of trust values, where the point of total trust and the point of total distrust are at the extremities. However, Lewicki has indicated that trust and distrust should be treated as independent dimensions [4]. Trust is a concept closely related to human's impressions; hence, in evaluating the trust values "contradictions/confusions" and "ignorance" should be considered. From this point of view, we introduced a two-dimensional trust representation with a pair of trust value and distrust value [5]-[6]. Specifically, we employed Oda's Fuzzy-set Concurrent Rating method [7] (hereinafter referred to as FCR method), which is a fuzzy-logic-based psychological theory for impression formation, as a basis for the trust representation, and by applying [8]-[9] we explored the correspondence between our trust representation and Marsh and Dibben's representation.

The two-dimensional trust value of [5]-[6] represents a trust state at a certain moment. However, to analyze trust-related properties, it is necessary to handle the changing nature of trust values. Thus, in this paper, we model the property of ever-changing trust value, and we conduct a computer-assisted verification. Specifically, we define a safety property with the transition sequences of trust values. Furthermore, based on the results in I/O-automaton theory [10]-[11], we conduct an efficient computer-assisted proof for the trust safety property.

This paper is organized as follows. Section 2 shows an overview of [5]'s FCR-based two-dimensional trust representation. Then, in Section 3 we describe how to deal with ever-changing trust values. Finally, Section 4 shows a case study, and we describe how to theorem-prove a trust safety property.

2 TWO-DIMENSIONAL TRUST REPRESENTATION

The trust classification by Marsh and Dibben is as follows, where a trust value ranges over $[-1, 1)$:

- *Trust*: is a state where a trust value of a trustee is more than a threshold value. We can see that this is a state enough to cooperate, and this is a measure of how much an agent believes a trustee;
- *Distrust*: is a state where the trustee's trust value is negative. This is a measure of how much an agent believes that the trustee will actively work against the agent in a given situation;
- *Untrust*: is a state where the trustee's trust value is positive but not enough to cooperate. This is a measure of

The second and fourth authors are currently at Chiba Institute of Technology and KYOWA EXEO Corporation, respectively.

how little the trustee is actually trusted; and

- *Mistrust*: is a state in which the initial trust has been betrayed; more precisely, the notion of mistrust can be considered as "Either a former trust destroyed, or former distrust healed," since the trustee may not have had bad intentions and it is not always "betrayed".

Suppose that you received a message, and you calculated its trust value. If the trust value is 0.9 and the cooperation threshold is 0.85, then from the definition of the trust notion, the message should be trusted. However, can we say that there is no distrust on this message? The maximum of the trust value is 1, hence we can see that there is a deficit of 0.1 points on the trust value. In this sense, the message might not be trusted enough. In [5]-[6], we considered that this was due to the limitation on the expressive power of one-dimensional trust representation, and we introduced a two-dimensional trust value to be an element of $Trust \times DisTrust$, where $Trust$ and $DisTrust$ are respectively degrees of trust and distrust, and we have $Trust = DisTrust = \{\, v \,|\, 0 \leq v \leq 1\}$. Following the manner in the FCR method, a two-dimensional trust value is also called an observation.

We focus on some observations to consider the meaning of the two-dimensional trust values. We can see that an observation around $(1, 0)$ has a high trust value and a low distrust value. Thus, we can see that the observation represents a state of "trust". Similarly, observations around $(0, 1)$ are the states of "distrust" since they have a low trust value and a high distrust value. For any observation (t, d) on the line between $(1, 0)$ and $(0, 1)$, we can see that (t, d) is ideal in the sense that the trust value and the distrust value satisfy the consistency condition $t + d = 1$. We consider that Marsh and Dibben's trust values are on this line. That is, the conventional trust value is defined with a limitation with regard to the consistency condition. Finally, the observation which corresponds to the conventional trust value of 0 is $(0.5, 0.5)$.

We introduced a classification of trust for two-dimensional trust values for observations with the consistency condition $t + d = 1$. Let CT be a cooperation threshold. The observations of the *trust* region should satisfy $t - d \geq CT$; that is, they are between $(\frac{1+CT}{2}, \frac{1-CT}{2})$ and $(1, 0)$. The observations between $(0.5, 0.5)$ and $(0, 1)$ correspond to a negative trust value; that is, they are in the *distrust* region. Actually, for any observation (t, d) in the distrust region we have $t < d$ where the degree of distrust is greater than the degree of trust. We can see that the other observations are in the *untrust* region.

For any observations (t, d) which may not satisfy $t+d = 1$, the above definition for trust notions is generalized as follows. To explain this, we employ a transformation:

$$\left[\begin{pmatrix} \cos\frac{\pi}{4} & -\sin\frac{\pi}{4} \\ \sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{pmatrix} \left\{ \begin{pmatrix} t \\ d \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right\} + \begin{pmatrix} \frac{\sqrt{2}}{2} \\ 0 \end{pmatrix}\right] \times \frac{1}{\frac{\sqrt{2}}{2}} = \begin{pmatrix} t-d \\ t+d-1 \end{pmatrix}$$

and the resulting point $(t - d,\ t + d - 1)$ is called (i, c).

First, we consider the first element $i = t - d$ of (i, c). This is a value with $-1 \leq i \leq 1$, and the value corresponds to the conventional trust value of Marsh and Dibben. In fact, the value i indicates that the result of subtracting the degree of trust by the degree of distrust is actually the net trust value, and we see that this matches the intuition.

The value of i is calculated graphically. Actually, we first draw a perpendicular line from (t, d) to the diagonal line between $(1, 0)$ and $(0, 1)$, and let (p, q) be the resulting point. The value of p is in $[0, 1]$, and the value of i is calculated as the result of normalizing p to be in $[-1, 1]$. We can see that this is a calculation of an integration value $I_2(t, d) = \dfrac{t + (1 - d)}{2}$ by the reverse-item averaging method; actually, the integration value should be normalized with $i = 2(I_2(t, d) - 0.5)$ to be a value in $[-1, 1]$.

We find that two observations in the same perpendicular line have the same integration value. For example, observation $A = (t, d)$ and its nearest point on the diagonal line

$$A' = (\frac{t + (1 - d)}{2},\ 1 - \frac{t + (1 - d)}{2})$$

have the same integration value. However, for observations A and A', the distance from the diagonal line is different. The distance between the observation (t, d) and the diagonal line is given by $|t + d - 1|$, which is the absolute value of the second element $c = t + d - 1$ of (i, c). We can easily see that the formula of c is equivalent to the degree of contradiction-irrelevance $C(t, d)$ of the FCR method. The degree of contradiction represents how much it deviates from the consistent condition ($t + d = 1$), and its value is between -1 and 1. The degree of contradiction is close to 1 if we deal with a trustee of a contradictory evaluation; for example, "I trust him but at the same time I feel some distrust on his behaviour". Also, if the degree is around -1, then a truster is ignorant on a trustee; that is, this is a situation like "I do not care for him at all." If we have $t + d = 1$, then the degree of contradiction is 0.

With the notion of the degree of contradiction, we can introduce two types of new untrust notions:

- *Untrust confusional*: This is a case where a trustee is both trusted and distrusted. Formally, this is a case with $0 < i < CT$ and $c \geq 0$; and
- *Untrust ignorant*: This is a case where the trustee is ignored; in other words, the trustee is both little trusted and little distrusted. Formally, this is a case with $0 < i < CT$ and $c < 0$.

The original untrust notion in [1] is considered as the notion of untrust ignorant.

3 TRANSITION OF OBSERVATIONS

Mistrust is a trust property with regard to a misplaced trust, and this is related to a change of trust values over time. In order to build a trust relationship among victims and volunteers in a large-scale disaster [12]-[14], it is important to analyze a chronological change of trust values. Also, a trust concept called *swift trust* [15]-[16] attracts an attention, which is a

trust property for building trust relations in a short period of time. In this section, we deal with the observations in the previous section as states, and we formalize time-related trust properties with state machines.

I/O-automaton [10]-[11] by Lynch et al. is a well-known mathematical model for distributed algorithms. In the theory of I/O-automata, a system is regarded as a collection of state machines which interacts with each other; the interactions are formalized with events. Some of events are observable from the outside of the system; for example, keyboard input, screen output, communication through the Internet are typical observable events. Some events cannot be observed; computer's internal processing and communications using a private line are such examples. A sequence of observable events from the initial state is called a trace, and a set of traces represents the behavior of the system. In general, systems have multiple (possibly infinite) traces, and the properties of automata are characterized with the set of traces.

Safety and liveness properties are well-known properties of distributed algorithms and they are defined with traces. A safety property guarantees that there is no occurrence of (specified) bad event. For example, if a computer program has no "division by 0" errors, then we regard that the program satisfies a safety property. On the other hand, a liveness property represents that finally some good behaviour will happen. For example, if a computer program always terminates, we can see that the program has a liveness property. If a communication system can reach an initial state from any state of the system, then we can see that the communication system satisfies another liveness property.

If we regard observations as states, the property "The trustee never goes to the region of distrust," is regarded as a safety property on trust transitions. Also, "A trustee will finally reach the region of trust," can be considered an liveness property on trust. In the following, let CT be a cooperation threshold with $0 < CT \leq 1$. We define the trust region $T(CT)$, the distrust region D, and the untrust region $U(CT)$ with:

$$\begin{cases} T(CT) &= \{\,(t,d)\,|\,t \in Trust \wedge d \in DisTrust \\ & \quad \wedge t - d \geq CT\,\}, \\ D &= \{\,(t,d)\,|\,t \in Trust \wedge d \in DisTrust \\ & \quad \wedge t < d\,\}, \text{ and} \\ U(CT) &= Trust \times DisTrust \setminus (T(CT) \cup D) \end{cases}$$

and we formalize trust safety properties. Note that operator "$\setminus$" is for the set subtraction.

Formally, automaton X has a set of actions $sig(X)$, a set of states $states(X)$, a set of initial states $start(X) \subset states(X)$ and a set of transitions $trans(X) \subset states(X) \times sig(X) \times states(X)$. Transition $(s, a, s') \in trans(X)$ is written as $s \xrightarrow{a}_X s'$. In this paper, a state is a tuple of values. Each element of the tuple has a corresponding distinct variable name. The name of a variable is used as an access function to the value. This kind of modeling is standard in I/O-automaton theory and its extensions such as [17]. In this paper, we use variables tr and dis for trust value and distrust value, respectively. The degrees of trust and distrust in state $s \in states(X)$ are referred as $s.tr$ and $s.dis$, respectively.

For any state $s \in states(X)$, a property "If s is not in the distrust region then the next state of s is not in the distrust region," is defined with:

$$\begin{array}{l} stepTrustSafe(s) \\ \quad \Longleftrightarrow \\ \quad\quad (s.tr,\, s.dis) \notin D \\ \quad\quad\quad \Longrightarrow\ \forall a \in sig(X)\, \forall s' \in states(X) \\ \quad\quad\quad\quad\quad [\, s \xrightarrow{a}_X s' \ \Longrightarrow\ (s'.tr,\, s'.dis) \notin D\,]. \end{array}$$

Hence, if we prove

$$\begin{array}{rl} & \forall s \in start(X)[\,(s.tr, s'.tr) \notin D\,] \\ \wedge & \forall s \in state(X)[\,stepTrustSafe(s)\,] \end{array} \quad (1)$$

then we have "The system X never reaches the distrust region." This formula consists of two conditions. The first condition represents that an initial state is not in the distrust region. The second condition means that every state s should satisfy $stepTrustSafe(s)$; that is, for any transition from s the system X never goes to the distrust region. If we use the predicate $reachable(s, s')$ for the reachability from state s to state s', the second condition can be:

$$\begin{array}{l} \forall s_{init} \in start(X),\, \forall s \in state(X) \\ \quad\quad [\,reachable(s_{init}, s) \ \Longrightarrow\ stepTrustSafe(s)\,] \end{array}$$

In this case, we consider the safety property only for reachable states. In any cases, this is to prove a trust safety property by induction on the length of execution sequences.

With the predicate $reachable$, another safety property "If a user exits the region of distrust, then the user never goes back to the distrust region," is formalized with:

$$\begin{array}{l} \forall s, s' \in states(X) \\ \quad [(reachable(s, s') \wedge (s'.tr, s'.dis) \notin D) \\ \quad\quad \Longrightarrow\ \forall s'' \in states(X) \\ \quad\quad\quad\quad [reachable(s', s'') \\ \quad\quad\quad\quad\quad \Longrightarrow\ (s''.tr, s''.dis) \notin D]\,]. \end{array}$$

We believe trust liveness properties can be formalized similarly.

4 PROVING TRUST SAFETY PROPERTY — A CASE STUDY

Let X be an automaton which specifies a communication system. In order to prove a trust safety property of X, it suffices to prove the condition (1) in the previous section with a theorem-proving tool directly. However, this proof approach is not efficient.

In this section, we apply an efficient proof method for trace inclusion of I/O-automaton to the verification of trust safety properties. Specifically, we describe two automata. The first automaton is what we can easily check that the automaton satisfies a trust safety property. The second automaton is a specification of the target communication system. If we can prove the trace inclusion between the two automata, the trust safety property of the first automaton leads to the trust safety property of the second automaton.

4.1 Preliminary Definition

First, we introduce the definition for sort `VL` with the Larch language. Specifically, the definition is:

```
VL: trait

  introduces
    vl0, vl1, otherValues: -> VL
    -__, abs: VL           -> VL
    __+__, __-__: VL, VL   -> VL
    __<__, __<=__,
    __>__, __>=__: VL, VL  -> Bool
    CT:                    -> VL
    D: VL, VL              -> Bool
    T: VL, VL, VL          -> Bool
    U: VL, VL, VL          -> Bool

  asserts with x, y, z, t, d, ct: VL
    x + vl0 = x;
    vl0 + x = x;
    x - vl0 = x;
    vl0 - x = -x;
    -(-x) = x;
    (x >= vl0) => abs(x)  = x;
    (x < vl0) => abs(x)  = -x;
    abs(-x) = abs(x);
    (-vl1) <= x;
    x <= vl1;
    (x >= y) <=> ((x = y) \/ (x > y));
    (x <= y) <=> ((x = y) \/ (x < y));
    (x > y) <=> ~(x <= y);
    (x < y) <=> ~(x >= y);
    CT > vl0;
    CT < vl1;
    D(t, d) <=> ((t-d) < vl0);
    T(t, d, ct) <=> ((t-d) >= ct);
    U(t, d, ct)
        <=> (~D(t, d) /\ ~(T(t, d, ct)))
```

and this kind of description is called a trait. The trait `VL` is for the set of real numbers whose domain is $[-1, 1]$. In this trait, several constants and operators are introduced; for example, constants `vl0` and `vl1` are respectively for 0 and 1 in sort `VL`. `CT` is a term for the cooperation threshold. Addition `+`, subtraction `-`, an unary operator for negative numbers `-`, comparison operators `<`, `>`, `<=` and `>=`, and the function for absolute value `abs` are defined as usual.

Predicates `D(t, d)`, `T(t, d, ct)` and `U(t, d, ct)` are true if observation `(t, d)` is in the distrust region, the trust region and the untrust region, respectively; note that `ct` is a parameter for the cooperation threshold.

4.2 Specifying and Proving Trust Safety Property

We introduce Fig. 1's I/O-automaton `testerSafety` to define a trust safety property. The automaton has three actions:

- `move(ev, pt, pd, dt, dd)`: enabled if event `ev` occurs and the two-dimensional trust value changes from `(pt, pd)` to `(pt+dt, pd+dd)`;
- `inDistr(t, d)`: enabled if trust value `(t, d)` is in the distrust region; and
- `notInDistr(t, d)`: enabled if trust value `(t, d)` is not in the distrust region.

```
uses testerSafetyDT

automaton testerSafety
  signature
    internal move(ev:Event, pt: VL,
                  pd: VL, dt: VL, dd: VL)
    output inDistr(t:VL, d:VL)
    output notInDistr(t:VL, d:VL)

  states
    tr: VL := vl0,
    dis: VL := vl0,
    stateOfAgent: agtState := InitState

  transitions
    internal move(ev, pt, pd, dt, dd)
      pre    pt = tr
          /\ pd = dis
          /\ (   vl0 <= (pt + dt)
              /\ (pt + dt) <= vl1)
          /\ (   vl0 <= (pd + dd)
              /\ (pd + dd) <= vl1)
          /\ condition(stateOfAgent, ev,
                       pt, pd, dt, dd)
      eff tr  := tr + dt;
          dis := dis + dd;
          stateOfAgent
              := change(stateOfAgent, ev)

    output inDistr(t, d)
      pre tr < dis /\ t = tr /\ d = dis
      eff tr := tr

    output notInDistr(t, d)
      pre ~(tr < dis) /\ t = tr /\ d = dis
      eff tr := tr
```

Figure 1: `testerSafety`: An Abstract System

Note that actions `inDistr` and `notInDistr` are special actions for analyzing trust transitions. If action `inDistr` does not appear on any trace, the automaton will not go to the distrusted region.

The transition of trust values is determined only by action `move`, and the action is enabled if predicate `condition` in the `pre`-part is true. The predication `condition` is introduced in Fig. 2's `testerSafetyDT`, and it is defined as

```
condition(st, ev, pt, pd, dt, dd)
    <=> ~D(pt, pd) /\ ~D(pt+dt, pd+dd)
```

in this study. This condition means that "The observation `(pt, pd)` of the current state and the observation `(pt + dt, pd + dd)` of the next state are neither in the distrust region." It is important for verifiers that the correctness of automaton `testerSafety` can be checked easily. Actually, `testerSafety` is small and simple. Moreover, with the `condition` predicate above, we can easily see that there is no occurrence of `inDistr` in `testerSafety`'s traces; this leads to the correctness of `testerSafety`. The correctness may look straightforward, but we should formally verify the correctness. This is because the result with regard to `testerSafety` is required when we conduct a simulation-based proof for trace inclusion in Sec. 4.3.2. The correctness for `testerSafety` is shown in the end of this section.

An example of event sequence from `testerSafety` is:

```
testerSafetyDT: trait

includes VL

introduces
  condition: agtState, Event,
             VL, VL, VL, VL -> Bool,
  InitState: -> agtState,
  AnotherState: -> agtState,
  change: agtState, Event -> agtState

asserts with st: agtState, ev: Event,
             pt, pd, dt, dd: VL
  condition(st, ev, pt, pd, dt, dd)
       <=> ~D(pt, pd) /\ ~D(pt+dt, pd+dd)
```

Figure 2: Datatype for `testerSafety`

```
notInDistr(vl0, vl0).
  move(get_mes(usrA, "hello"),
               vl0, vl0, vl0.3, vl0.2).
    notInDistr(vl0.3, vl0.2).
```

In the initial state, both of the trust and distrust degrees are 0; that is, we have `(tr, dis) = (vl0, vl0)`. In this case, `tr < dis` holds, and this allows an occurrence of event `notInDistr(vl0, vl0)`. Then, after the occurrence of event `get_mes`, the pair of trust and distrust degrees becomes `(vl0.3, vl0.2)`, where constants `vl0.3` and `vl0.2` in sort `VL` represent 0.3 and 0.2 respectively. In the resulting state, the distrust degree still does not exceed the trust degree, thus event `notInDistr(vl0.3, vl0.2)` can occur.

In the above example, it looks that "random" real numbers 0.2 and 0.3 are used in the event sequence. However, "arbitrary" numbers are actually assumed for the parameters. That is, at a level of abstraction, theorem-proving is exhaustive with regard to the event sequences, and all the event sequences are logically checked in a verification.

In the specification of Fig. 1, we have variables `dt` and `dd` in action `move`. They are variables of sort `VL`, and the definitions in trait `VL` restrict their values to be in $[-1, 1]$. Additionally, `dt` and `dd` should be the values which satisfy both of values `pt + dt` and `pd + dd` are in $[0, 1]$. This condition is written in the precondition part of `move`.

The I/O-automaton `testerSafety` can be translated into first-order predicate logic formulae, and a trust safety property can be proven with Larch Prover (LP) [18]. In the following we prove the trust safety property that "For any state `s` of automaton `testerSafety`, if `s` is reachable then the two-dimensional trust value at `s` is not in the distrust region." This is formally described as:

```
(\A st:States[testerSafety]
    (reachableAbst(st)
     => ~D(st.tr, st.dis))).
```

Proving this formula is equivalent to proving two conditions

```
   (\A st:States[testerSafety]
       (start(st)
        => ~D(st.tr, st.dis)))
```

and

```
(\A st:States[testerSafety]
(\A at:Actions[testerSafety]
    (reachableAbst(st)
     => ((   enabled(st, at)
          /\ ~D(st.tr, st.dis))
         => ~D(effect(st, at).tr,
               effect(st, at).dis))))).
```

We can see that this is to prove a trust safety property by induction on the length of execution sequences starting from initial states; the first condition represents a base case, and the second condition is an induction step.

Below, we show a proof script for a trust safety property.

```
prove
 (\A st:States[testerSafety]
     (reachableAbst(st)
      => ~D(st.tr, st.dis)))
..
%
 prove
  (\A st:States[testerSafety]
      (start(st) => ~D(st.tr, st.dis)))
 ..
 res by =>
 %
 prove
  (\A st:States[testerSafety]
      (\A at:Actions[testerSafety]
          (reachableAbst(st)
           => ((enabled(st, at)
                /\ ~D(st.tr, st.dis))
               => ~D(effect(st, at).tr,
                     effect(st, at).dis)))))
 ..
 res by =>
 %
 res by ind on at
 res by =>
 res by =>
 res by =>
qed
```

From this, `~D(st.tr, st.dis)` holds for any reachable state `st` of `testerSafety`. This means action `inDistr` is not enabled at `st`. Therefore, we obtain the following lemma; a screenshot of computer-assisted theorem-proving for this lemma is shown in Fig. 3.

Lemma 1 *Every trace of I/O-automaton* `testerSafety` *does not have any occurrence of action* `inDistr`. □

In this sense, we can see that `testerSafety` never goes to the region of distrust; this leads to a trust safety property.

We have shown that `testerSafety` satisfies a trust safety property. We can see that the property is, informally, "The transition of trust value does not reach the region of distrust." Note that this is a trust safety property but is not the only trust safety property. In general, and more formally, if the following conditions are satisfied for the set $traces(P)$ of all traces of automaton P, we say P is called a safety property [10]:

(i) $traces(P)$ is non-empty;

(ii) $traces(P)$ is prefix-closed, that is, if $\beta \in traces(P)$ and β' is a finite prefix of β, then $\beta' \in traces(P)$; and

(iii) $traces(P)$ is limit-closed, that is, if $\beta_1, \beta_2, \ldots$ is an infinite sequence of finite sequences of $traces(P)$, and

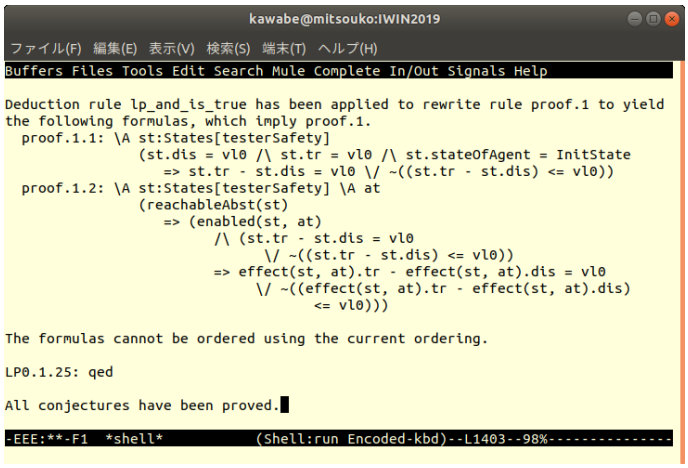


Figure 3: Theorem-Proving Trust Safety of `testerSafety`

> for each i, β_i is a prefix of β_{i+1}, then the unique sequence β that is the limit of the β_i under the successive extension ordering is also in $traces(P)$.

Automaton `testerSafety` is not general in this sense, however, from a practical viewpoint, it is too strong to require a general automaton satisfying the above conditions. We consider `testerSafety` is powerful enough for analysis.

4.3 Specifying Communication System and Safety Proof by Trace Inclusion

4.3.1 Specification of Communication System

We consider a specification `bbdSystem` of a communication system; see Fig. 4. This communication system sends a message to an online bulletin board after evaluating the user's trust value. The system receives a message from a user by action `get_mes` and evaluates the trust value of the message with actions `discard_mes` and `approve_mes`. If the trust value in the next state reaches the distrust region when writing the message to the bulletin board, the message is not sent and discarded by `discard_mes`. Otherwise, the message is written to the bulletin board by actions `approve_mes` and `say`. Actions `inDistrC` and `notInDistrC` are special actions to discuss the trust values, and they correspond to actions `inDistr` and `notInDistr` of `testerSafety`.

Automaton `bbdSystem` uses a datatype defined in the trait `bbdSystemDT` shown in Fig. 5. This trait employs `Sequence(MES)`, which defines a message queue.

We introduce functions `evalTr` and `evalDis` for calculating the degree of trust and the degree of distrust, respectively, though the concrete definitions of these functions are not given in this paper; giving a definition for these functions is a future work. In this paper, only the constraints on these functions are defined as follows. For function `evalTr`, we employ a constraint

```
(\A tr:VL
      ((vl0 <= tr /\ tr <= vl1)
       => (\A m:MES
              (   vl0 <=
                  (tr + evalTr(tr, m))
               /\ (tr + evalTr(tr, m))
                  <= vl1))))
```

```
uses bbdSystemDT

automaton bbdSystem
  signature
    input get_mes(i:ID, m:MES)
    internal discard_mes(i:ID, m:MES)
    internal approve_mes(i:ID, m:MES)
    output say(i:ID, m:MES)
    output inDistrC(t:VL, d:VL)
    output notInDistrC(t:VL, d:VL)

  states
    tr: VL := vl0,
    dis: VL := vl0,
    flg: Bool := false,
    mesQ: Seq[MES] := empty

  transitions
    input get_mes(i, m)
      eff mesQ := mesQ ||
                  (packet(i, m) -| empty)

    internal discard_mes(i, m)
      pre    ~flg
          /\ mesQ ~= empty
          /\ packet(i, m) = head(mesQ)
          /\ ((tr + evalTr(tr, m))
              -(dis + evalDis(dis, m)))
             < vl0
      eff mesQ := tail(mesQ)

    internal approve_mes(i, m)
      pre    ~flg
          /\ mesQ ~= empty
          /\ packet(i, m) = head(mesQ)
          /\ ((tr + evalTr(tr, m))
              -(dis + evalDis(dis, m)))
             >= vl0
      eff flg := true

    output say(i, m)
      pre flg /\ mesQ ~= empty
              /\ packet(i, m) = head(mesQ)
      eff tr := tr + evalTr(tr, m);
          dis := dis + evalDis(dis, m);
          mesQ := tail(mesQ);
          flg := false

    output inDistrC(t, d)
      pre tr < dis /\ t = tr /\ d = dis
      eff tr := tr

    output notInDistrC(t, d)
      pre ~(tr < dis) /\ t = tr /\ d = dis
      eff tr := tr
```

Figure 4: `bbdSystem`: A Concrete System

for the degree of trust to be in $[0, 1]$. For `evalDis`, a similar condition is introduced.

We do not introduce the definition of `change`, which is a function in the effect part of action `move`. This is because in our case study the value of `stateOfAgent` is not required in evaluating the precondition part of `move`. However, if we modify the definition of predicate `condition`, a concrete definition might be required.

4.3.2 Safety Proof by Trace Inclusion

We replace `bbdSystem`'s observable actions `get_mes` and `say` with internal actions of the same name; the resulting automaton is called `bbdSystem`$\backslash\{$`get_mes, say`$\}$. If we prove the existence of a binary relation called a forward simulation from automaton `bbdSystem`$\backslash\{$`get_mes, say`$\}$ to

```
bbdSystemDT: trait

includes VL, Sequence(MES)

introduces
  packet:  ID, MES -> MES
  evalTr:  VL, MES -> VL
  evalDis: VL, MES -> VL
  getMesFromPacket:  MES -> MES

asserts with tr, dis: VL, i:ID, m: MES
  (\A tr:VL
      ((vl0 <= tr /\ tr <= vl1)
       => (\A m:MES
              (   vl0 <=
                  (tr + evalTr(tr, m))
               /\ (tr + evalTr(tr, m))
                  <= vl1))));
  (\A dis:VL
      ((vl0 <= dis /\ dis <= vl1)
       => (\A m:MES
              (   vl0 <=
                  (dis + evalDis(dis, m))
               /\ (dis + evalDis(tr, m))
                  <= vl1))));
  getMesFromPacket(packet(i, m)) = m;
```

Figure 5: Datatype for bbdSystem

automaton testerSafety, then we have a trace inclusion

$$traces(\texttt{bbdSystem}\backslash\{\texttt{get\_mes}, \texttt{say}\}) \subseteq traces(\texttt{testerSafety})$$

from Theorem 3.10 of [11]. From Lemma 1, every trace in $traces(\texttt{testerSafety})$ does not have any occurrence of action inDistr. Hence, if we prove a trace inclusion, every trace in $traces(\texttt{bbdSystem}\backslash\{\texttt{get\_mes}, \texttt{say}\})$ does not contain inDistr; this leads to the absence of inDistr in $traces(\texttt{bbdSystem})$.

A candidate binary relation for a forward simulation is defined as follows:

```
fs(sb, st) <=> (   (sb.tr = st.tr)
                /\ (sb.dis = st.dis)
                /\ (sb.flg => X) )

where X =
 ( ((sb.tr
     + evalTr(sb.tr,
         getMesFromPacket(head(sb.mesQ))))
   - (sb.dis
      + evalDis(sb.dis,
         getMesFromPacket(head(sb.mesQ)))))
   >= vl0)
```

To prove that binary relation fs is a forward simulation, we should prove the initial state condition

```
(\A sb:States[bbdSystem]
    (start(sb)
     => (\E st:States[testerSafety]
            (start(st) /\ fs(sb, st)))))
```

and step correspondence condition

```
(\A sb:States[bbdSystem]
(\A sb':States[bbdSystem]
(\A st:States[testerSafety]
(\A ab:Actions[bbdSystem]
    (reachableAbst(st)
     => ((fs(sb, st) /\ step(sb, ab, sb'))
         => (\E st':States[testerSafety]
                (   steps(st, ab, st')
                 /\ fs(sb', st')))))))))
```

with a theorem proving tool. We show a proof script as follows.

```
% ------------------------------
% Initial states' correspondence
% ------------------------------
prove
 (\A sb:States[bbdSystem]
     (start(sb)
      => (\E st:States[testerSafety]
             (start(st) /\ fs(sb, st)))))
..
res by =>
res by spec st to [InitState, vl0, vl0]
qed

% ------------------------------
% Step's correspondence
% ------------------------------
prove
 (\A sb:States[bbdSystem]
 (\A sb':States[bbdSystem]
 (\A st:States[testerSafety]
 (\A ab:Actions[bbdSystem]
     (reachableAbst(st)
      => ((fs(sb, st) /\ step(sb, ab, sb'))
          => (\E st':States[testerSafety]
                 (   steps(st, ab, st')
                  /\ fs(sb', st')))))))))
..
res by =>
%
make passive c-o(steps)
res by ind on ab
%
res by =>
res by spec st' to stc
%
res by =>
res by spec st' to stc
%
res by =>
res by spec st' to stc
prove getMesFromPacket(head(sbc.mesQ)) = mc
crit c-o(mc) / c-o(packet) with *
%
res by =>
res by spec st' to
  effect(stc, move(ev, stc.tr, stc.dis,
  evalTr(sbc.tr, mc), evalDis(sbc.dis, mc)))
..
make passive c-o(intTrans)
rewrite con with reversed
  (c-o(intTrans) / c-o(enabled)) ~ c-o(steps)
..
res by spec at to
  move(ev, stc.tr, stc.dis,
       evalTr(stc.tr, mc),
       evalDis(stc.dis, mc))
..
make active c-o(intTrans)
prove getMesFromPacket(head(sbc.mesQ)) = mc
crit c-o(mc) / c-o(packet) with *
%
crit c-o(stc) with *
crit c-o(stc) with *
%
res by =>
res by spec st' to stc
res by spec st1 to stc
prove effect(stc, inDistr(t, d)) = stc
crit c-o(inDistr) with *
%
res by =>
res by spec st' to stc
res by spec st1 to stc
prove effect(stc, notInDistr(t, d)) = stc
crit c-o(notInDistr) with *
%
qed
```

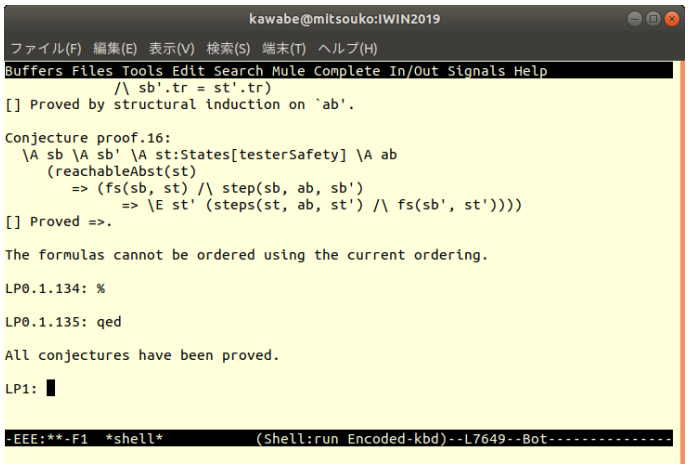


Figure 6: Theorem-Proving Trace Inclusion

This leads to the following lemma.

Lemma 2 *Binary relation* `fs(sb, st)` *is a forward simulation from automaton* `bbdSystem`$\backslash\{$`get_mes, say`$\}$ *to automaton* `testerSafety`. □

Summarizing, we have the following result.

Theorem 1 *Every trace of I/O-automaton* `bbdSystem` *does not have an occurrence of action* `inDistr`.

Proof: Proven by Lemmata 1 and 2. □

Consequently, a trust safety property has been shown for automaton `bbdSystem`; see Fig. 6 for a snapshot of proving with LP.

We can see that the specification of `bbdSystem` is small. Thus, it may look straightforward that automaton `bbdSystem` satisfies some safety-related trust property. This observation is correct in some sense. However, it is not easy to rigorously state what kind of safety property is satisfied by `bbdSystem`. Based on the correctness of `testerSafety`, we can define the correctness of `bbdSystem` formally. Moreover, the correctness can be proven with a semi-automatic theorem-proving approach.

Suppose that we write automaton `distSystem`, which is a specification of distributed or more concrete version of `bbdSystem`. If we can prove a trace inclusion

$$traces(\texttt{distSystem}) \subseteq traces(\texttt{bbdSystem})$$

at a level of abstraction, this paper's result with regard to the safety property of `bbdSystem` leads to the safety property of `distSystem`. In this sense, a stepwise verification for trust safety properties is possible when we deal with larger specifications. This is an advantage of this paper's verification method.

5 DISCUSSION AND CONCLUSION

In this paper, we discussed how to analyze transitions of two-dimensional trust values. Specifically, we employed a theory of distributed algorithms to formalize trust safety properties such as "Any transition does not lead to the distrust region," or "After reaching some region other than the distrust region, the system never goes to the distrust region."

We cannot say that it efficient to theorem-prove the condition for trust safety property of each automaton. In this paper, we employed I/O-automata to represent a trust safety property, and applied a proof method for trace inclusion. This enables us to verify trust safety properties efficiently. Furthermore, with simple examples, we empirically demonstrated that it is possible to verify trust safety properties with automatic theorem provers. In this study, we employed the Larch Prover (LP), which is theorem proving tool based on first-order predicate logic and equational theory. LP proves formulae with the techniques of term rewriting systems [19]-[21]. Although we need to translate IOA specifications into each theorem prover's formulae as in [18][22], we believe that a similar proof with another theorem proving tool (such as Coq [23], Isabelle [24], and Maude [25]) is possible. With regard to the functions `evalTr` and `evalDis`, we introduced the constraint on the range of the trust value only, and we did not discuss the concrete evaluation method of the trust value. In other words, this paper gives an analysis method for the transition of the evaluated trust value at each moment. It is an interesting future work to define the trust value of each moment.

The trust, distrust, and untrust notions were defined for two-dimensional trust values in [5]-[6]. This is defined with the value of $i = t - d$, and we do not use the value of c. We can see that this is formalized as a linear case; for example, the trust notion and the distrust notion are defined with linear restrictions $d \leq -t + CT$ and $d > t$, respectively. This is just for simplicity, and we believe it is possible to provide a finer definition for trust notions with both of i and c. Introducing such a non-linear definition is an interesting future work. In order to deal with the non-linear settings, it would be required to change the definitions for the regions of trust, distrust and untrust. However, we believe that the verification method in this paper is also applicable for the non-linear case.

Finally, conducting larger verification examples is an important work. Specifically, we are planning a trust verification for communication systems [26]-[27] based on social media.

REFERENCES

[1] S. Marsh and M. R. Dibben, "Trust, untrust, distrust and mistrust – an exploration of the dark(er) side," in *Proceedings of the Third International Conference on Trust Management*, iTrust'05, (Berlin, Heidelberg), pp. 17–33, Springer-Verlag (2005).

[2] G. Primiero, "A calculus for distrust and mistrust," in *Trust Management X* (S. M. Habib, J. Vassileva, S. Mauw, and M. Mühlhäuser, eds.), (Cham), pp. 183–190, Springer International Publishing (2016).

[3] G. Primiero, F. Raimondi, M. Bottone, and J. Tagliabue, "Trust and distrust in contradictory information transmission," *Applied Network Science*, vol. 2, p. 12 (2017).

[4] R. J. Lewicki, D. J. B. McAllister, and R. J. Bies, "Trust and distrust: New relationships and realities," *Academy of Management Review*, vol. 23, pp. 438–458 (1998).

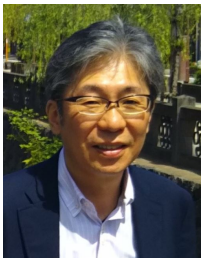
[5] K. Ohkubo, T. Oda, Y. Koizumi, T. Ohki, M. Nishigaki, T. Hasegawa, and Y. Kawabe, "Trust representation under confusion and ignorance," in *Proceedings of International Workshop on Informatics (IWIN 2018)*, pp. 191–198 (2018).
[6] Y. Kawabe, Y. Koizumi, T. Ohki, M. Nishigaki, T. Hasegawa, and T. Oda, "On trust confusional, trust ignorant, and trust transitions," in *Proceedings of IFIPTM 2019* (2019).
[7] T. Oda, "Fundamental characterestics of fuzzy-set concurrent rating method," *Journal of Japan Association for Management Systems*, vol. 12, no. 1, pp. 23–32 (1995). In Japanese.
[8] T. Oda, "Fuzzy set theoretical approach for improving the rating scale method : Proposing and introducing the FCR-method and the IR-method as novel rating methods," *Japanese Psychological Review*, vol. 56, no. 1, pp. 67–83 (2013). In Japanese.
[9] T. Oda, "Measurement technique for ergonomics, section 3: Psychological measurements and analyses (3) measurements and analyses by kansei evaluation," *The Japanese Journal of Ergonomics*, vol. 51, no. 5, pp. 293–303 (2015). In Japanese.
[10] N. A. Lynch, *Distributed Algorithms*. Morgan Kaufmann Publishers (1996).
[11] N. Lynch and F. Vaandrager, "Forward and backward simulations — part I: Untimed systems," *Information and Computation*, vol. 121, pp. 214–233, (1995).
[12] Y. Murayama, "Issues in disaster communications," *Journal of Information Processing*, vol. 22, no. 4, pp. 558–565 (2014).
[13] M. G. Busa, M. T. Musacchio, S. Finan, and C. Fennell, "Trust-building through social media communications in disaster management," in *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, (New York, NY, USA), pp. 1179–1184, ACM (2015).
[14] F. Lemieux, "The impact of a natural disaster on altruistic behaviour and crime," *Disasters*, vol. 38, pp. 483–499 (2014).
[15] D. Meyerson, K. E. Weick, and R. M. Kramer, *Swift Trust and Temporary Groups in Trust in Organizations: Frontiers of Theory and Research*. SAGE (1995).
[16] J. Wildman, M. Shuffler, E. Lazzara, S. Fiore, and S. Burke, "Trust development in swift starting action teams: A multilevel framework," *Group & organization management*, vol. 37, no. 2, pp. 137–170 (2012).
[17] D. Kaynar, N. Lynch, R. Segala, and F. Vaandrager, *The Theory of Timed I/O Automata, Second Edition*. Morgan & Claypool Publishers, 2nd ed. (2010).
[18] J. F. Soegaard-Andersen, S. J. Garland, J. V. Guttag, N. A. Lynch, and A. Pogosyants, "Computer-assisted simulation proofs," in *CAV '93*, vol. 697 of *Lecture Notes in Computer Science*, pp. 305–319, Springer-Verlag (1993).
[19] F. Baader and T. Nipkow, *Term Rewriting And All That*. Cambridge University Press (1998).
[20] J. W. Klop, "Term rewriting systems," in *Handbook of Logic in Computer Science* (D. G. S. Abramsky and T. S. E. Maibaum, eds.), vol. 2, pp. 1–112, Oxford University Press (1992).
[21] E. Ohlebusch, *Advanced topics in term rewriting*. Springer-Verlag (2002).
[22] A. Bogdanov, "Formal verification of simulations between I/O-automata," Master's thesis, Massachusetts Institute of Technology (2000).
[23] "The Coq Proof Assistant." `https://coq.inria.fr/`, (June 13, 2020).
[24] L. C. Paulson, "Isabelle: The next seven hundred theorem provers," in *Proceedings of the 9th Conference on Automated Deduction*, vol. 310 of *Lecture Notes in Computer Science*, pp. 772–773, Springer-Verlag (1989).
[25] N. Martí-Oliet and J. Meseguer, "Rewriting logic: Roadmap and bibliography," Theoretical Computer Science, Vol. 285, pp. 121–154 (2002).
[26] J. Chen, M. Arumaithurai, X. Fu, and K. K. Ramakrishnan, "CNS: Content-oriented notification service for managing disasters," in *Proceedings of ACM Conference on Information-Centric Networking*, pp. 122–131, ACM (2016).
[27] M. Jahanian, Y. Xing, J. Chen, K. K. Ramakrishnan, H. Seferoglu, and M. Yuksel, "The evolving nature of disaster management in the internet and social media era," in *2018 IEEE International Symposium on Local and Metropolitan Area Networks, LANMAN 2018, Washington, DC, USA, June 25-27, 2018*, pp. 79–84 (2018).

(Received October 30, 2019)
(Revised April 9, 2020)

Toshinori Fukunaga received the B.E., and M.E. degrees in electrical engineering from Chiba University in 1996 and 1998. He joined Nippon Telegraph and Telephone Corporation in 1998. His main research area is cryptography and its implementation. He is also involved in information security management system work and is currently engaged in human resource development for researchers.

Hideki Goromaru received the M.E. degree in Graduate School of Engineering from the Kagoshima University in 1995 and received the Ph.D. degree in Graduate School of System Engineering from the Wakayama University, Japan, in 2015. In 1995, he joined NTT Corp. and he had been a research and development engineer at NTT Laboratories until 2020. In 2020, he is an Associate professor at the Chiba Institute of Technology, Japan. His research interests include groupware, security and risk management. He is a member of Information Processing Society of Japan, the Institute of Electronics, Information and Communication Engineer, Japan Creativity Society and the Japan Psychological Association.

Tadanori Mizuno received the B.E. degree in Industrial Engineering from the Nagoya Institute of Technology in 1968 and received the Ph.D. degree in Engineering from Kyushu University, Japan, in 1987. In 1968, he joined Mitsubishi Electric Corp. From 1993 to 2011, he had been a Professor at Shizuoka University, Japan. From 2011 to 2016, he had been a Professor at the Aichi Institute of Technology, Japan. Since 2016, he is an Affiliate Professor at the Aichi Institute of Technology, Japan. His research interests include mobile computing, distributed computing, computer networks, broadcast communication and computing, and protocol engineering. He is a member of Information Processing Society of Japan, the Institute of Electronics, Information and Communication Engineers, the IEEE Computer Society and Consumer Electronics, and Informatics Society.



Kazuhiko Ohkubo received his B.E. degree in Informatics from the University of Tsukuba in 1987 and his M.E. in Electrical Engineering from the University of Tokyo in 1989. In 1989, he joined NTT (Nippon Telegraph and Telephone Corporation) Telecommunication Laboratories and earned his M.S. degree in Management of Technology from the MIT Sloan School of Management, USA in 2000. From 2016 to 2019, he had been the head of NTT Secure Platform Laboratories and also been a director of NTT Security Corporation, the specialized security company of NTT Group. In 2019, he received his Ph.D. in Business Administration and Computer Science from the Aichi Institute of Technology. Since 2019, he is the CISO of Kyowa Exeo Corporation. He is a member of IEEE.

Yoshinobu Kawabe received his B.E., M.E., and D.E. degrees in information engineering from Nagoya Institute of Technology in 1995, 1997, and 2003. He joined NTT Communication Science Laboratories, Nippon Telegraph and Telephone Corporation in 1997. In 2002, he was a visiting research scientist at MIT Laboratory for Computer Science. Since 2008, he has been at Aichi Institute of Technology, where he is a professor at the Department of Information Science. His research interests include term rewriting systems, process algebras, network programming languages, formal methods, security/privacy verification, and computational trust. He is a member of ACM, JSSST, IPSJ, and IEICE.