Modeling Agreement Problems in the Universal Composability Framework

Masayuki Terada[1,2], Kazuki Yoneyama[2], Sadayuki Hongo[1], and Kazuo Ohta[2]

[1] NTT DoCoMo, Inc.,
3–5 Hikari-no-oka, Yokosuka, Kanagawa, Japan
[2] University of Electro-Communications,
1–5 Chofu-ga-oka, Chofu, Tokyo, Japan

Abstract. Agreement problems are one of the keys to distributed computing. In this paper, we propose a construction of the ideal-model functionality of one of the most important agreement problems, non-blocking atomic commitment (NBAC), in the universally-composability (UC) framework. NBAC is not only important in realizing dependable transactions in distributed computing environments but also useful in constructing security protocols that require the fairness property, such as fair exchange protocols. Our construction of NBAC functionality (namely $\mathcal{F}_{\text{NBAC}}$) is exactly equivalent to the NBAC definition; it is formally proved that a protocol UC-securely realizes $\mathcal{F}_{\text{NBAC}}$ *if and only if* the protocol is an NBAC protocol. The proposed functionality and its proof of equivalence to NBAC enables the NBAC protocols to be used as a provably secure building block, and thus makes it much easier to feasibly and securely create higher-level protocols.

1 Introduction

Motivation. Agreement problems, which represent consistent decision making processes among independent subjects interacting with each other, are the most essential problem in the distributed computing literature. Their resolution is indispensable to realize trustworthy electronic commerce; e.g., consistent money transfer among banks is guaranteed by transaction processing systems that solve one type of agreement problem, namely the atomic commitment problem[1].

Protocols that can solve agreement problems, namely agreement protocols, are not only useful in themselves but also needed as fundamental building-blocks for realizing security protocols since multi-party protocols that require fairness explicitly or implicitly must include a process to resolve agreement problems. For example, broadcast channels, commonly used in multi-party protocols[2], are realized by solving the Byzantine agreement problem[3]. Another example is fair exchange[4,5], which is known to be unconditionally reducible into non-blocking atomic commitment (NBAC)[6] among trusted processes.[7,8]

Since the agreement problems are well-studied and a number of agreement protocols for diverse environments and assumptions have been developed so far[6,3,9,8], modeling agreement problems as *securely composable* functionalities enables us to dispense with reinventing the wheel when creating multi-party protocols and makes it easier to implement those protocols by introducing the concept of reusability, which is quite common in software programming, to security protocol design.

To realize this scenario, it is important to design functionalities that are "exactly equivalent" to the agreement protocols; a too strong functionality may lack protocols that realize it, while a too weak functionality may make it difficult to prove the security of the protocols that uses the functionality. The aim of this paper is to introduce functionalities that are provably equivalent to the agreement protocols so that we can use them as securely composable and substitutable building-blocks and so permit the design of high-level and complex security protocols.

Contributions. In this paper, we propose a construction of the functionality of NBAC, which is a fundamental agreement problem[6,10,11] known to be capable of becoming a building block in the construction of fair exchange protocols[7,8], in the universal composability (UC) framework[12]. To ensure that the functionality, namely $\mathcal{F}_{\mathrm{NBAC}}$, captures the NBAC properties exactly, we introduce an oracle called *failure detection oracle* ($\mathcal{O}^{FD}$), which detects if any failure has occurred during protocol execution. This oracle is a sort of *failure detector*[13] commonly used to abstract the error detection abilities of distributed processes (e.g. timeout detection) in the distributed computing literature[14]. To be exact, the introduced oracle has abilities equivalent to those of the *anonymous perfect failure detector* (usually denoted as $?\mathcal{P}$)[11], which can correctly detect any failure but does not provide any information about where the failure has occurred.

A proof of the exactness of the proposed functionality $\mathcal{F}_{\mathrm{NBAC}}$ is also provided in this paper. Designing a functionality that exactly represents the properties of the agreement protocols is, however, not be so straightforward that everyone can intuitively understand its correctness; it must be proved that the designed functionality exactly satisfies the properties in order to utilize the functionality as a securely composable building block. We provide proofs that the proposed functionality $\mathcal{F}_{\mathrm{NBAC}}$ is exactly equivalent to NBAC; i.e., some protocol π securely realizes $\mathcal{F}_{\mathrm{NBAC}}$ *if and only if* π holds all of the NBAC properties.

Related works. The concept of universal protocol composition has been attracted much attention, however, few studies have focused upon the agreement properties of protocols.

Lindell et al.[15] focus upon composability of the Byzantine agreement (or generals) problem[3], which are known to be reducible from the problem to establish a broadcast channel in a point-to-point network. Their notable impossibility results show that a Byzantine agreement protocol requires more than 2/3 honest parties to be secure under parallel composition even if the messages are authenticated, although it was originally believed that the Byzantine agreement with authenticated messages (authenticated Byzantine agreement) could be solved under any number of dishonest (corrupted) parties.[3]

Garay et al.[16] specified a composable ideal functionality that gives the *resource-fair* property, which states that if one party learns the output of the protocol, then so can all other parties, as long as they expend roughly the same amount of resources. The functionality is defined in a similar way to the UC framework, but works in the framework where the environment compensate the additional resources for honest parties prematurely aborted by the adversary. This functionality and framework provide the notion of fairness (which is a similar notion to agreement) as a composable building block in secure multi-party computation protocols, however, this model assumes synchronous channels and the functionality is constructed upon a fair delivery mechanism that provides *complete fairness*[2], where if one party receives the output, all parties receive the output; these assumptions are too ideal and hard to realize in real systems.

In the distributed computing literature, on the other hand, NBAC[6] is considered to be a fundamental agreement problem to ensure the atomicity of a distributed transaction, and its characteristics including solvability have been well-studied[10,1,17,11,9,14,8].

NBAC can be solved in the asynchronous model if (a certain class of) failure detectors can be assumed. For example, every following protocol solves NBAC in certain environment: the 3-phase commitment (3PC) protocol[6], the 2-phase commitment (2PC) protocol[1][1] and the optimistic NBAC protocol[8]. A different application or situation would need a different NBAC protocol since every protocol has its own strengths and weaknesses; e.g., the 3PC protocol is the most versatile but is extremely complicated[9], 2PC is easy to implement but requires special assumptions on a process (i.e. coordinator), and the optimistic NBAC is simple and efficient but applicable only to the 2-party setting where each party has a trusted module such as a smartcard.

The functionality proposed in this paper makes all protocols that correctly hold the NBAC properties under parallel composition usable as securely composable building blocks in the UC framework.

2 Preliminaries

2.1 Non-Blocking Atomic Commitment (NBAC)

An NBAC agreement guarantees that the participating processes, each of which votes yes or no, eventually agree upon a common outcome, commit or abort. In this agreement, commit can be decided only if all participants vote yes[2]. If any process votes no, abort must be decided. Every correct process is guaranteed to receive the decided value.

Precisely, the NBAC problem is defined as follows.[6,17]

[1] The 2PC protocol is usually known to solve the atomic commitment problem, which is similar to NBAC but does not guarantee the Termination property, but it solves NBAC if no misbehavior (including crash) of the coordinator process and resilient channels between the coordinator and the other processes can be assumed.

[2] Note that "if" direction is not guaranteed.

Definition 1 (NBAC). *The NBAC problem consists of a set of independent processes that reach a unanimous decision,* commit *or* abort*, according to initial votes of the processes,* yes *or* no*, such that the following properties*[3] *are satisfied:*

Agreement. *No two processes decide differently;*
Termination. *Every correct process eventually decides;*
Commit-Validity (C-Validity). *If all processes propose* yes *and there is no failure, then the decision value must be* commit*; and*
Abort-Validity (A-Validity). *If at least one process proposes* no*, then the decision value must be* abort*.*

While NBAC is an essential problem in the distributed computing literature, it is also valuable in constructing fair exchange protocols as mentioned in Sect. 1. Fair exchange is reducible into NBAC between trusted processes by the following reduction algorithm[7]:

```
FairExchange(item $i$, description $d$) {
  ⟨send $i$ to exchange partners over secure channel⟩
  timed wait for ⟨expected item $i_e$ from exchange partners⟩
    ⟨check $d$ on $i_e$⟩
  if (check succeeds and no timeout)
    then $vote$ := yes else $vote$ := no endif
  $result$ := NBAC($vote$)
  if ($result$ = commit)
    then return $i_e$ else return ⟨abort⟩ endif
}
```

The fair exchange protocol based on this algorithm enables parties to fairly exchange arbitrary items. When using the optimistic NBAC protocol[8] as the NBAC part of this algorithm, this exchange realizes optimistic (strong) fair exchange of arbitrary items while other known optimistic protocols can guarantee strong fairness only when at least one exchanged item has a special property called strong generatability[5][4].

2.2 UC Framework

In the following, we briefly introduce the concept of the UC framework; its comprehensive and rigorous definition is described in [12].

The UC framework is a sort of simulation-based security framework, where the "formal specification" of the security requirements of a task is represented as a set of instructions of a trusted process, namely ideal-model functionality.

[3] C-Validity and A-Validity are often called "non-triviality" and "uniform-validity", respectively.

[4] However, note that fair exchange by this method requires both parties to run trusted processes; i.e., each party has to have access to a trusted device such as a smartcard. This requirement would be easily satisfied by recent mobile phones (equipped with (U)SIM cards), but might be rather difficult in other environments.

A protocol is said to *UC-realize* the functionality (for the task) if running the protocol "emulates" the functionality, in sense that observable outputs from the parties and an adversary running the protocol and those from the parties and an ideal adversary (i.e. a simulator) interacting with the functionality cannot be distinguished by the *environment* machine with non-negligible probability. The environment machine is a probabilistic interactive Turing machine (PITM) that can hand arbitrary inputs to the parties and the adversary and can collect (observe) outputs from them.

The characteristic merit of this framework is to guarantee secure *universal composition* of the protocol (i.e. universal composition theorem); when a protocol UC-realize a functionality, the functionality that is "called" within another protocol (like a subroutine) can be securely substituted by an execution of the protocol UC-realizing the functionality[5].

This merit makes security protocols modular as building-blocks and makes it much easier to design and analyze complicated security protocols, however, the other side of the coin is the difficulty of designing functionalities. Since universal composability is founded upon tight simulatability between a protocol and a functionality, a functionality that is either stronger or weaker than the desired properties of the task obstructs secure composition of the protocol; a too strong functionality often lacks protocols that realize it, while a too weak functionality often makes it impossible to prove the security of the hybrid protocol that use the functionality even if the protocol is actually secure.

Designing an adequate functionality exactly equivalent to the desired properties of a task, therefore, is quite important in realizing modularized designs and simplified implementation of security protocols in the UC framework.

3 Ideal Functionality of NBAC

The ideal functionality in the UC framework is represented as a trusted party that captures the desired specification of the task by way of specifying a set of instructions.

Since the desired specification of NBAC is defined as the four NBAC properties (i.e., Agreement, Termination, C-Validity and A-Validity), designing the ideal functionality of NBAC is, accordingly, basically similar to finding a constraint satisfaction algorithm that *exactly* satisfies these properties.

As mentioned in Sect. 2, it is important to be careful not to make the functionality "too ideal" — it should exactly satisfy the properties to make the functionality useful as a secure building block. For example, a functionality, which outputs commit if all input values are yes or outputs abort otherwise, obviously satisfies all NBAC properties, but is almost useless as the ideal functionality of NBAC. This functionality is too ideal to be securely realized by any practical

[5] To be more exact, provided that protocol ρ UC-realizes some functionality $\mathcal{F}$ and protocol π in which parties make calls to $\mathcal{F}$ UC-realizes some functionality $\mathcal{G}$, protocol π^ρ in which parties run ρ instead of calling $\mathcal{F}$ also UC-realizes $\mathcal{G}$.

NBAC protocol in the real world, so any hybrid protocol constructed with this functionality would also be unrealizable.

In order for the functionality to exactly capture the desired properties, the functionality interacts with a party called *simulator*, which simulates attacks by an adversary in the real-world model. Most agreement protocols including NBAC, however, have a property that cannot be represented even by interacting with the simulator; i.e., the condition "there is no failure" in the C-Validity properties. A simulator naturally knows whether a failure occurs internally, but there is no means for a functionality to *rightly* know that from the simulator; even if the simulator sends a message that indicates that a failure has occurred, the functionality may not trust it because the UC framework does not guarantee that the simulator sends a message based on its internal states correctly . Accordingly, another means by which the functionality can accurately know whether a failure has occured is needed for the functionality to correctly represent the C-Validity property.

3.1 Failure Detection Oracle

To adequately represent the C-Validity property, we introduce the failure detection oracle, denoted by $\mathcal{O}^{FD}$. This oracle is an ideal failure detector that detects if any failure (including network failure, prematurely abortion or any other misbehaviors by corrupted party) is caused by the adversary.

Definition 2 (Failure detection oracle). *Failure detection oracle $\mathcal{O}^{FD}$ is an oracle that outputs $f \leftarrow \{0,1\}$ upon receiving sid, where f takes the following value according to failure events in the protocol execution identified by sid:*

$$f = \begin{cases} 1 & (\textit{if}\ \text{no failure}\ \textit{occurs in the protocol}), \textit{or} \\ 0 & (\textit{otherwise}). \end{cases} \tag{1}$$

No failure *in the above definition means that every participant behaved correctly and every message is transfered as expected by the protocol definition.*

Since $\mathcal{O}^{FD}$ is a virtual function to define a functionality in the ideal model, protocols realizing a functionality with this oracle do not have to assume the existence of $\mathcal{O}^{FD}$ in the real-world model. However, interestingly, this oracle has equivalent abilities to the *anonymous perfect failure detector* ($?\mathcal{P}$), which can detect any failure *completely* (i.e. any failure is detected within some time) and *accurately* (i.e. no failure is detected unless a failure occurs) but does not provide any information about where the failure happened; this failure detector is known to be sufficient to transform *Consensus*[6] into NBAC.[18]

[6] The Consensus problem is another agreement problem where the following *validity* property holds instead of C-Validity and A-Validity properties in NBAC: *a value decided must be a value proposed by some process.*

3.2 Definition of $\mathcal{F}_{\mathbf{NBAC}}$

The ideal functionality of NBAC, namely $\mathcal{F}_{\text{NBAC}}$, is defined as follows:

Definition 3 (Ideal functionality of NBAC ($\mathcal{F}_{\text{NBAC}}$)). *This functionality proceeds as follows, running with participants* $\mathbf{P} = \{P_1, P_2, ..., P_n\}$*, simulator* $\mathcal{S}$*, and failure detection oracle* $\mathcal{O}^{FD}$*:*

1. *Upon receiving a vote* $(\mathsf{Vote}, sid, \mathbf{P}, vote_i)$ *from* P_i*, where* $vote_i \leftarrow \{0,1\}$ *is the proposal value of* P_i *(*yes*: 1,* no*: 0), check if a vote from* P_i *is recorded. If it is already recorded, ignore the received vote. If not, record the vote.*
2. *When all votes from* $\mathbf{P}$ *are recorded, send* $(\mathsf{Exec}, sid, \mathbf{P})$ *to* $\mathcal{S}$*.*
3. *Upon receiving* $(\mathsf{Notice}, sid, \phi, \chi)$ *from* $\mathcal{S}$*, where* $\phi \leftarrow \{0,1\}$ *indicates if abort was forcibly caused by the adversary (forcibly aborted: 1, otherwise: 0) and* $\chi = \chi_1||\chi_2||...||\chi_n \leftarrow \{0,1\}^n$ *indicates if termination of a participant* P_i *was interrupted by corrupt acts of the adversary (*P_i *can terminate:* $\chi_i = 1$*,* P_i *is corrupted and cannot terminate:* $\chi_i = 0$*), send sid to* $\mathcal{O}^{FD}$*.*
4. *Upon receiving* f *from* $\mathcal{O}^{FD}$*, send* $(\mathsf{Result}, sid, result_i)$ *to* P_i*, where* $result_i$ *takes the following value:*

$$result_i = \begin{cases} \perp & (\textit{if } \chi_i = 0 \textit{ and } P_i \textit{ is corrupted}), \\ \prod_{j=1}^{n} vote_j & (\textit{else if } f = 1), \\ \prod_{j=1}^{n} vote_j \cdot \phi & (\textit{otherwise}). \end{cases} \tag{2}$$

$result_i$ is the decision value that participant P_i receives. 0 and 1 represent abort *and* commit*, respectively.* $\perp$ *indicates that* P_i *cannot decide (i.e., does not terminate).*

4 Proving the Equivalence of $\mathcal{F}_{\mathbf{NBAC}}$ to NBAC

In this section, we prove that $\mathcal{F}_{\text{NBAC}}$ defined in Sect. 3 is the equivalent functionality of NBAC; i.e., a protocol is an NBAC protocol if and only if it UC-securely realizes $\mathcal{F}_{\text{NBAC}}$.

Before proving it, we firstly provide a formal definition of an NBAC protocol, namely π_{NBAC}, by using symbolic logic in order to rigorously discuss the equivalence of the functionality to NBAC. Next, we prove two lemmas: 1) π_{NBAC} UC-realizes $\mathcal{F}_{\text{NBAC}}$ (i.e., $\mathcal{F}_{\text{NBAC}}$ is not stronger than π_{NBAC}), and 2) any protocol realizing $\mathcal{F}_{\text{NBAC}}$ is π_{NBAC} (i.e., $\mathcal{F}_{\text{NBAC}}$ is not weaker than π_{NBAC}). The equivalence can be stated as a corollary of these two lemmas.

4.1 Formal Definition of NBAC

Although the definition of NBAC in Sect. 2 (1) is commonly used and easy to understand, it isn't rigorous enough to discuss the equivalence of $\mathcal{F}_{\text{NBAC}}$ to NBAC. An NBAC protocol can be formally defined by using symbolic logic as follows:

Definition 4 (NBAC protocol). *A protocol running with a set of independent processes* $\mathbf{P} = \{P_1, P_2, ..., P_n\}$ *is an NBAC protocol* π_{NBAC} *if it satisfies the* *following, where P_i inputs a vote value $v_i (\in \{0,1\}; \mathsf{yes}:1, \mathsf{no}:0)$ and receives a decision value $r_i (\in \{0,1,\perp\}; \mathsf{commit}:1, \mathsf{abort}:0, \textit{no decision}:\perp)$, $\mathbf{P}_C (\subseteq \mathbf{P})$ is a set of corrupted (incorrect) processes, and f is a value defined as Eqn. 1:*

Agreement. *The requirement of Agreement property, No two processes decide differently, means that the results of any two different processes (r_i and $r_j (i \neq j)$) are equal if both processes receive the result ($r_i \neq \perp \cap r_j \neq \perp$). Hence, this property can be formalized as follows:*

$$\forall ((r_i, r_j) | r_i \neq \perp, r_j \neq \perp) r_i = r_j . \tag{3}$$

(Termination). *This property requires that any honest process ($P_i (\notin \mathbf{P}_C)$) can terminate and get a result ($r_i \neq \perp$). Hence,*

$$\forall (r_i | P_i \notin \mathbf{P}_C) r_i \neq \perp . \tag{4}$$

(C-Validity). *This property requires that the result of any process is* commit *($\forall (r_i) r_i = 1$) if all votes are* yes *($(\forall (v_j) v_j = 1)$) and there is no failure (i.e., the failure detection oracle $\mathcal{O}^{FD}$ outputs $f = 1$). Hence,*

$$\forall (r_i) r_i = 1 \text{ if } (\forall (v_j) v_j = 1) \cap (f = 1). \tag{5}$$

(A-Validity). *This property requires that the result of any terminated process is* abort *($\forall (r_i | r_i \neq \perp) r_i = 0$) if at least one vote is* no *($\exists (v_j) v_j = 0$). Hence,*

$$\forall (r_i | r_i \neq \perp) r_i = 0 \text{ if } \exists (v_j) v_j = 0. \tag{6}$$

4.2 Proving That π_{NBAC} UC-Realizes $\mathcal{F}_{\text{NBAC}}$

Lemma 1. *An NBAC protocol (π_{NBAC}) UC-securely realizes $\mathcal{F}_{\text{NBAC}}$.*

Proof. To prove this lemma, it suffices to show that there exists a simulator $\mathcal{S}$ that can simulate any adversary by internal execution of π_{NBAC}.

Suppose simulator $\mathcal{S}$ runs as follows:

1. Upon receiving message ($\mathsf{Exec}, sid, \mathbf{P}$), internally run π_{NBAC} with vote values $v_i = vote_i$ and obtain decision values r_i $(i = 1, ..., n)$.
2. Equate ϕ and χ_i as follows:

$$\phi = \begin{cases} 0 & (\exists (r_i | r_i \neq \perp) r_i = 0), \\ 1 & (\text{otherwise}); \end{cases} \tag{7}$$

$$\chi_i = \begin{cases} 0 & (r_i = \perp), \\ 1 & (\text{otherwise}). \end{cases} \tag{8}$$

3. Send ($\mathsf{Notice}, sid, \phi, \chi$) to the functionality in terms of the above ϕ and χ_i.

In the following, we prove that such a simulator can simulate any adversary so that environment $\mathcal{Z}$ cannot distinguish $\mathcal{F}_{\text{NBAC}}$ and π_{NBAC}; i.e., the observable output of $\mathcal{F}_{\text{NBAC}}$ ($result_i$) and that of π_{NBAC} (r_i) are always consistent ($result_i = r_i$).

The proof is divided into the following two cases: P_i is not corrupted (case 1) and P_i is corrupted (case 2).

Case 1: P_i is not corrupted. If P_i is assumed not to be corrupted (P_i is correct), i.e., $P_i \notin \mathbf{P}_C$, it holds $r_i \neq \perp$ according to Termination property and $\chi_i = 1$ is given by Eqn. (8).

Under this assumption, r_i must have the following values: $r_i = 1$ if $(\forall(v_j)v_j = 1) \cap (f = 1)$ according to C-Validity property, $r_i = 0$ if $\exists(v_j)v_j = 0$ according to A-Validity property; r_i is undefined in other cases, i.e., if $(\forall(v_j)v_j = 1) \cap (f = 0)$, however, it must be 0 or 1 according to the assumption (cf. Termination property).

Consequently, r_i holds the following value in terms of $v_j (j = 1, ..., n)$ and f.

$$r_i = \begin{cases} 1 & (\forall(v_j)v_j = 1 \cap f = 1), \\ 0 \text{ or } 1 & (\forall(v_j)v_j = 1 \cap f = 0), \\ 0 & (\exists(v_j)v_j = 0). \end{cases} \tag{9}$$

In the first case, $result_i$ has the following value according to Eqn. (2):

$$result_i = \prod_{j=1}^{n} vote_j = 1. \tag{10}$$

It holds $result_i = r_i$.

In the second case, Agreement property requires that $\forall(r_j | r_j \neq \perp) r_j = r_i$. Eqn. (7) gives $\phi = 0$ if $\exists(r_j)r_j = 0$, or $\phi = 1$ otherwise. Hence, it holds that $r_i = \phi$. Since $\forall(j)vote_j = v_j = 1$, $result_i$ becomes:

$$result_i = \prod_{j=1}^{n} vote_j \cdot \phi = 1 \cdot \phi = \phi. \tag{11}$$

It also holds that $result_i = r_i$ in this case.

In the last case, it obviously holds that $result_i = 0 (= r_i)$ since $\prod_{j=1}^{n} vote_j = 0$.

Therefore, it always holds that $result_i = r_i$ if P_i is not corrupted.

Case 2: P_i is corrupted. If P_i is assumed to be corrupted, Termination property does not restrict r_i not to become $\perp$ (no decision) so that r_i may take any value of $\{0, 1, \perp\}$.

If $r_i \neq \perp$, it holds that $result_i = r_i$ as shown in case 1. If $r_i = \perp$, Eqn. (8) gives $\chi_i = 0$. Since $\chi_i = 0$ and P_i is corrupted in this case, it becomes $result_i = \perp$ according to Eqn. (2); it holds that $result_i = r_i (= \perp)$.

Hence, simulator $\mathcal{S}$ can simulate any adversary so that environment $\mathcal{Z}$ cannot distinguish $\mathcal{F}_{\text{NBAC}}$ and π_{NBAC}, and therefore π_{NBAC} UC-securely realizes $\mathcal{F}_{\text{NBAC}}$. □

4.3 Proving That Any Protocol UC-Realizing $\mathcal{F}_{\text{NBAC}}$ Is π_{NBAC}

Next, we prove that any protocol that UC-realizes $\mathcal{F}_{\text{NBAC}}$ is π_{NBAC} by proving the following contrapositive lemma.

Lemma 2. *A protocol that is* not π_{NBAC} *cannot UC-realize* $\mathcal{F}_{\text{NBAC}}$.

Proof. In the following, we show that no simulator can simulate adversaries such that the environment cannot distinguish $\mathcal{F}_{\text{NBAC}}$ and protocol π that does not hold at least one of the NBAC properties.

If π does not hold Agreement property. Assume that π does not hold Agreement property and thus outputs decision values r_i, r_j where $r_i \neq r_j (r_i \neq \perp, r_j \neq \perp)$.

In order that π UC-securely realize $\mathcal{F}_{\text{NBAC}}$, there must exist a simulator such that $\mathcal{F}_{\text{NBAC}}$ outputs $result_i, result_j$ where $result_i \neq result_j (i \neq j)$.

However, $\mathcal{F}_{\text{NBAC}}$ always outputs $result_i$ and $result_j$ such that $result_i = result_j$ for any pair of i and j when $result_i \neq \perp$ and $result_j \neq \perp$, regardless to the behavior of the simulator; i.e., $\mathcal{F}_{\text{NBAC}}$ outputs either $result_i = result_j = \prod_{k=1}^{n} vote_k$ or $result_i = result_j = \prod_{k=1}^{n} vote_k \cdot \phi$. Hence, under this assumption, the outcome of π and $\mathcal{F}_{\text{NBAC}}$ become inconsistent so the environment can easily distinguish them.

If π does not hold Termination property. Assume that π does not hold Termination property and thus outputs $r_i = \perp$ to $P_i (\notin \mathbf{P}_C)$.

$\mathcal{F}_{\text{NBAC}}$ always gives $result_i \neq \perp$ according to Eqn. (2) to P_i if P_i is not corrupted. $result_i$ and r_i contradict each other and are thus distinguishable.

If π does not hold C-Validity property. Assume that π does not hold C-Validity property and thus outputs $r_i = 0$ to P_i despite $\forall (v_j) v_j = 1$ and $f = 1$.

Under this assumption, $\mathcal{F}_{\text{NBAC}}$ outputs $result_i = \prod_{j=1}^{n} vote_j = 1$ to P_i, according to Eqn. (2); it becomes $result_i \neq r_i$.

If π does not hold A-Validity property. Assume that π does not hold A-Validity property and thus outputs $r_i = 1$ despite $\exists (v_j) v_j = 0$.

According to Eqn. (2), $\mathcal{F}_{\text{NBAC}}$ outputs $result_i = \prod_{j=1}^{n} vote_j = 0$ when $\exists (v_j) v_j = 0$ and therefore $result_i \neq r_i$ under this assumption.

Consequently, an execution of π, which does not hold at least one of the NBAC properties, is inevitably distinguishable from $\mathcal{F}_{\text{NBAC}}$ running with any simulator; a protocol that is not π_{NBAC} cannot UC-securely realize $\mathcal{F}_{\text{NBAC}}$ and therefore any protocol that can UC-securely realize $\mathcal{F}_{\text{NBAC}}$ is π_{NBAC}. □

4.4 Equivalence of $\mathcal{F}_{\text{NBAC}}$ to NBAC

As a corollary of Lemma 1 and Lemma 2, the following theorem can be stated:

Theorem 1. *Some protocol π realizes $\mathcal{F}_{\text{NBAC}}$ if and only if π is an NBAC protocol (π_{NBAC}).*

The equivalence of $\mathcal{F}_{\text{NBAC}}$ to NBAC is thus proved. □

5 Conclusion

We proposed a construction of ideal functionality of the non-blocking atomic commitment, namely $\mathcal{F}_{\text{NBAC}}$, in the universal composability framework. To exactly capture the NBAC properties by the functionality, we introduced a failure detection oracle $\mathcal{O}^{FD}$, which is an ideal failure detector notifying the functionality of the occurrence of failures caused by the adversary during protocol execution. We also confirmed that the proposed functionality is a proper functionality of NBAC by proving the equivalence of the $\mathcal{F}_{\mathrm{NBAC}}$ and the formalized NBAC protocol π_{NBAC}; i.e., a protocol UC-realizes $\mathcal{F}_{\mathrm{NBAC}}$ if and only if the protocol is π_{NBAC}.

Our construction of NBAC can be easily applied to design another agreement problems. For example, the atomic commitment (AC) problem, the most frequently-examined agreement problem in the distributed computing field, is equivalent to NBAC without the Termination property; the equivalent functionality can be defined as follows:

Definition 5 (Ideal functionality of AC ($\mathcal{F}_{\mathrm{AC}}$)). *This functionality proceeds as follows, running with participants* $\mathbf{P} = \{P_1, P_2, ..., P_n\}$*, simulator* $\mathcal{S}$*, and failure detection oracle* $\mathcal{O}^{FD}$*:*

1. *(Same as Step* $1 \sim 3$ *of* $\mathcal{F}_{\mathrm{NBAC}}$*, see Def. 3)*
2. *Upon receiving* f *from* $\mathcal{O}^{FD}$*, send* $(\mathsf{Result}, sid, result_i)$ *to* P_i*, where* $result_i$ *takes the following value:*

$$result_i = \begin{cases} \perp & (if\ \chi_i = 0), \\ \prod_{j=1}^{n} vote_j & (else\ if\ f = 1), \\ \prod_{j=1}^{n} vote_j \cdot \phi & (otherwise). \end{cases} \tag{12}$$

$result_i$ is the decision value that participant P_i receives. 0 and 1 represent abort *and* commit*, respectively.* $\perp$ *indicates that* P_i *cannot decide (i.e., does not terminate).*

The only difference of $\mathcal{F}_{\mathrm{AC}}$ from $\mathcal{F}_{\mathrm{NBAC}}$ is the condition that $result_i$ becomes $\perp$; this difference reflects the fact that only corrupted processes can become unable to terminate in NBAC but any process can become unable to terminate in AC. The proof of equivalence between an AC protocol and $\mathcal{F}_{\mathrm{AC}}$ is also similar to that of NBAC and is thus trivial.

As mentioned in Sect. 1, these agreement protocols are useful in constructing other higher-level (and more complicated) protocols such as fair exchange protocols. The proposed functionality $\mathcal{F}_{\mathrm{NBAC}}$ and other functionalities derivable from our $\mathcal{F}_{\mathrm{NBAC}}$ construction (e.g. $\mathcal{F}_{\mathrm{AC}}$) will be beneficial in designing such complicated protocols and in making it easier to formally prove their security, without sacrificing any feasibility or realizability of the protocols in the real world.

References

1. Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco (1992)
2. Goldwasser, S., Lindell, Y.: Secure computation without agreement. In: Malkhi, D. (ed.) DISC 2002. LNCS, vol. 2508, pp. 17–32. Springer, Heidelberg (2002)
3. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Programming Language and Systems 4, 382–401 (1982)
4. Asokan, N.: Fairness in Electronic Commerce. PhD thesis, University of Waterloo (1998)

5. Pagnia, H., Vogt, H., Gärtner, F.C.: Fair exchange. The Computer Journal 46, 55–75 (2003)
6. Skeen, D.: Nonblocking commit protocols. In: Proc. 1981 ACM SIGMOD Intl. Conf. Management of Data, pp. 133–142 (1981)
7. Avoine, G., Gärtner, F.C., Guerraoui, R., Kursawe, K., Vaudenay, S., Vukolic, M.: Reducing fair exchange to atomic commit. Technical Report 200411, Swiss Federal Institute of Technology (EPFL), School of Computer and Communication Sciences, Lausanne, Switzerland (2004)
8. Terada, M., Mori, K., Hongo, S.: An optimistic NBAC-based fair exchange method for arbitrary items. In: Domingo-Ferrer, J., Posegga, J., Schreckling, D. (eds.) CARDIS 2006. LNCS, vol. 3928, pp. 105–118. Springer, Heidelberg (2006)
9. Gray, J., Lamport, L.: Consensus on transaction commit. Technical Report MSR-TR-2003-96, Microsoft Research (2004)
10. Gray, J.: A comparison of byzantine agreement problem and the transaction commit problem. In: Simons, B., Spector, A. (eds.) Fault-Tolerant Distributed Computing. LNCS, vol. 448, pp. 10–17. Springer, Heidelberg (1990)
11. Guerraoui, R., Kouznetsov, P.: On the weakest failure detector for non-blocking atomic commit. In: Proc. 2nd IFIP Intl. Conf. Theoretical Computer Science (TCS). IFIP Conference Proceedings, vol. 223, pp. 461–473. Kluwer, Dordrecht (2002)
12. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proc. 42nd Symp. Foundations of Computer Science (FOCS) Full version at Cryptology ePrint Archive 2000/067, pp. 136–145 (2001)
13. Chandra, T.D., Toueg, S.: The weakest failure detector for solving consensus. J. ACM 43, 225–267 (1996)
14. Freiling, F.C., Guerraoui, R., Kouznetsov, P.: The failure detector abstraction. Technical Report 2006-003, Faculty of Mathematics and Computer Science, University of Mannheim (2006)
15. Lindell, Y., Lysyanskaya, A., Rabin, T.: On the composition of authenticated byzantine agreement. In: Proc. 34th Annual ACM Symp. Theory of Computing (STOC), pp. 514–523 (2002)
16. Garay, J.A., MacKenzie, P.D., Prabhakaran, M., Yang, K.: Resource fairness and composability of cryptographic protocols. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 404–428. Springer, Heidelberg (2006)
17. Guerraoui, R.: Revisiting the relationship between non-blocking atomic commitment and consensus. In: Helary, J.-M., Raynal, M. (eds.) WDAG 1995. LNCS, vol. 972, pp. 87–100. Springer, Heidelberg (1995)
18. Guerraoui, R.: Non-blocking atomic commit in asynchronous distributed systems with failure detector. Distributed Computing 15, 17–25 (2002)