Formal Verification of Non-Repudiation Protocols
A Game Approach

Steve Kremer Jean-François Raskin
Département d'Informatique
Faculté des Sciences
Université Libre de Bruxelles, Belgium
skremer@ulb.ac.be Jean-Francois.Raskin@ulb.ac.be

Abstract

In this paper, we report on a recent work for the verification of non-repudiation protocols. We propose a verification method based on the idea that non-repudiation protocols are best modeled as games. To formalize this idea, we use alternating transition systems, a game based model, to model protocols and alternating temporal logic, a game based logic, to express requirements that the protocols must ensure. This method is automated by using the model-checker MOCHA, a model-checker that supports the alternating transition systems and the alternating temporal logic. An optimistic non-repudiation protocol is analyzed using MOCHA.

1 Introduction

Non-repudiation protocols. During the last decade open networks, above all the Internet, have known an impressive growth. As a consequence new security issues, like non-repudiation have to be considered. Repudiation is defined as the *denial* of an entity of having participated in all or part of a communication. Consider for instance the following scenario: Alice want to send a message to Bob; after having sent the message, Alice may deny having sent it (repudiation of origin), or Bob may deny having received it (repudiation of receipt). Therefore specific protocols have been designed, generating both non-repudiation of origin (NRO) evidences, destined to Bob, and non-repudiation of receipt (NRR) evidences, intended to Alice. In case of a dispute Alice or Bob can present their evidences to an adjudicator, who can take a decision in favor of one of the two entities without ambiguity.

Different solutions have been proposed: they are generally classified in two classes. The first class contains the probabilistic protocols. A second one uses a trusted third party (TTP). The probabilistic approach is often based on a gradual release of knowledge and generally requires that all involved parties have the same computational power. In [BOGMR90] a protocol for digital contract signing[1] without TTP is proposed: the probability that the contract has been signed, is increased each round until reaching one. The assumption of same computational power is not needed. Another recently presented probabilistic non-repudiation protocol [MR99], also succeeded in relaxing the condition on the computational power. Here the idea is that the recipient does not know a priori which transmission will contain the message. The probability of guessing the transmission including the message is arbitrarily small. However to decrease the probability the number of messages has to be increased. The second approach is the one using a TTP. Thus, the message is first sent to the TTP, who acts as an intermediary to assure delivery. The major problem of this approach is the network bottleneck, created at the TTP. To avoid the performance decrease created by this bottleneck, Asokan et al. introduced the optimistic

[1]Digital contract signing is a problem closely related to non-repudiation.

approach for fair exchange [ASW97]. In an optimistic protocol one supposes that in general the involved entities are honest and the network is well functioning. The rational is that the TTP only intervenes in case of a problem (a cheating entity or a network failing a delivery at a critical moment). Afterwards Zhou et al. applied the optimistic approach to the non-repudiation protocols [ZG97]. These are the protocols that received most of the attention in literature.

A non-repudiation protocol has to respect several properties. The most important one is *fairness*: fairness must ensure that if at least one entity is honest, either both entities receive the expected non-repudiation evidence or none of them receives it. This property can also be split in the following two properties. *Fairness for Alice* ensures that, if Alice is honest, Bob receives the non-repudiation of origin evidence only if Alice receives her non-repudiation of receipt evidence. *Fairness for Bob* ensures that, if Bob is honest, Alice receives her non-repudiation of receipt evidence only if Bob receives his non-repudiation of origin evidence. When we take the conjunction of these two properties, we refind the fairness requirement. Another property we require is *timeliness*: we want that the protocol finishes for each honest player after a finite amount of time. A third property that is desirable but not necessary is viability. A protocol is *viable* if two honest players always succeed in exchanging the expected evidences. In general viability can only be realized by strengthening the requirements we make on the communication channels. We define three classes of channels: unreliable channels, resilient channels and operational channels. No assumptions have to be made about *unreliable channels*: data may be lost. A *resilient channel* delivers data after a finite, but unknown amount of time. Data may be delayed, but will eventually arrive. When using an *operational channel* data arrive before a known, constant amount of time. Operational channels are however rather unrealistic in heterogenous networks. A formal definition of both these properties and the channels, will be found later.

In comparison to other security issues, such as privacy or authenticity of communications, non-repudiation has not been studied intensively. However, the more studied authentication protocols have shown that the design of security protocols is an error prone process. Some flaws have only been found after years (e.g. the Needham-Schroeder public-key authentication protocol [Low95]). Hence, a need for formal verification methods has been identified. These methods include an adequate specification, more precise than the traditional, informal and sometimes ambiguous one often used in literature. Another aim is the automatic verification. Several works have been engaged: believe logics, such as BAN [BAN96], general purpose model checkers, e.g. FDR [Low96], theorem provers as Isabelle for instance [Pau97] and also special purpose verification tools as the NRL protocol analyzer [Mea96] have successfully been applied to authentication protocols. First works have been done on non-repudiation protocols using CSP [Sch98] and on fair exchange protocols using the the model-checker Murφ [SM99].

Non-repudiation protocols as games. There are some fundamental differences between authentication protocols and exchange protocols, such as non-repudiation protocols. Generally one of the most difficult problems in authentication protocols is to deal with the presence of an intruder. In non-repudiation protocols we do not need to model an intruder, but we have to consider that either Alice or Bob, the two entities taking part in the protocol, may *cheat* (cf also [Sch98]). Another difference is that exchange protocols, above all optimistic ones, are not linear. Generally authentication protocols are rather linear and only allow very few different traces. On the other hand, non-repudation protocols are divided in several subprotocols (e.g. a main and a recovery protocol), making branching possible, although they are intended to be executed in a given order by a honest entity. Changing the order of execution could result in subtle errors. This is the reason why we propose a new method for the specification and the verification of exchange protocols. First, we want to model the actions that are possible in the course of the protocol and not stick to a given predefined order of execution. In that way, we give a malicious entity the potential not to follow the protocol, but to construct an attack against the honest entity. Second, we consider the execution of the protocol as a game: each entity (Alice, Bob, TTP) and each communication channel is a player. This point of view allows us to express formally the required properties as *strategies*. For instance, a property such as fairness for Alice can be expressed as follows: "a coalition of Bob and all the communication channels does not have a strategy to obtain a non-repudiation of origin evidence without Alice having a strategy to obtain a non-repudiation of receipt evidence". Here, we have rephrased the property as the existence of a strategy. The main advantage of modeling such protocols as games is that we *directly* and *formally* take into account the possibility of *adversarial* behaviors. The other participating entities can try to fool Alice when she tries to obtain her non-repudiation of receipt evidence. As the communication channels are also modelized as players, they can cooperate with protocol entities. This means that either the channels are not well-functioning or that they are controlled by a player of the coalition. For instance, when a message is lost on an unreliable channel, this can be due to a network failure, or a dishonest player who removed the message by cooperating with the communication channels. As a second example, consider the game view of the viability property: Alice, in cooperation with Bob must have a strategy against the coalition of the communication channels, such that both Alice and Bob possess their expected evidences at the end of the protocol. This example illustrates the facility using the strategies not only to express adversial behaviors, but also *cooperative* behaviors between several players. The trust in the TTP is modeled by giving the TTP a unique strategy: even if he makes part of one coalition or the other, his behavior is deterministic and thus *can not choose* to help anyone. By using alternating transition systems and alternating-time temporal logic of Alur et al. [AHK97], we are able to formalize the non-repudiation protocols and their requirements in a direct way.

Structure of the paper. We organize the paper as follows. In section 2, we introduce the *alternating transition systems* and the *alternating-time temporal logic* of Alur et al. [AHK97]. The alternating transition systems are used to formalize exchange protocols as games and alternating-time temporal logic is used to formalize the requirements that exchange protocols must ensure. Section 3 shows how exchange protocols, and more specifically non-repudiation protocols, can be modeled *naturally* and *accurately* as games. Viewing those protocols as games allows us to define the requirements as existence of strategy for players to reach a given goal. The players are the different entities participating to the protocol. In that way, we *directly* and *formally* take into account the possible existence of adversarial (cheating) behaviors of participating entities to the protocol. The main properties that must ensure a non-repudiation protocol are given as alternating-time temporal logic formulas. Here, we also show that the expressive power of the logic CTL is not strong enough to model those properties. In section 4, we report on primarily results about the automatic verification of a non-repudiation protocol using the model-checker MOCHA [AHM+98]. MOCHA is a model-checker that supports the alternating transition systems and the alternating-time temporal logic. We show that the use of this tool allows us to formally analyze in details the optimistic Zhou-Gollman non-repudiation protocol [ZG97]. We were also able to explain formally in term of strategies a subtle bug, discovered only recently [KM00] in this protocol. Finally, we report on plans for future works in section 5 and draw some conclusions.

2 A Formal Model of Games and its Logic

Alternating Transition System. An *alternating transition system* [AHK97] (ATS, for short) is a 5-tuple $S = \langle \Pi, \Sigma, Q, \pi, \delta \rangle$ with the following components:

- Π is a set of propositions.
- Σ is a set of players.
- Q is a set of states.
- $\pi : Q \to 2^{\Pi}$ maps each state to the set of propositions that are true in the state.

- $\delta : Q \times \Sigma \to 2^{2^Q}$ is a transition function that maps a state and a player to a nonempty set of choices, where each choice is a set of possible next states. Whenever the system is in state q, each player a chooses a set $Q_a \in \delta(q,a)$. In this way, a player a ensures that the next state of the system will be in its choice Q_a. However, which state in Q_a will be next depends on the choices made by the other players, because the successor of q must lie in the intersection $\bigcap_{a\in\Sigma} Q_a$ of the choices made by all players. The transition function is non-blocking and the players together choose a unique next state. Thus, we require that this intersection always contains a unique state: assuming $\Sigma = \{a_1,\ldots,a_n\}$, then for every state $q \in Q$ and every set $Q_1,\ldots,Q_n$ of choices $Q_i \in \delta(q,a_i)$, the intersection $Q_1 \cap \ldots \cap Q_n$ is a singleton.

For two states q and q' and a player a, we say that q' is an *a-successor* of q if there exists a set $Q' \in \delta(q,a)$ such that $q' \in Q'$. We denote by $succ(q,a)$ the set of a-successors of q. For two states q and q', we say that q' is a *successor* of q if for all players $a \in \Sigma$, we have $q' \in succ(q,a)$. Thus, q' is a successor of q iff whenever the system S is in state q, the players in Σ can cooperate so that q' will be the next state. A *computation* of S is an infinite sequence $\lambda = q_0,q_1,q_2,\ldots$ of states such that for all positions $i \geq 0$, the state q_{i+1} is a successor of the state q_i. We refer to a computation starting at state q as a *q-computation*. For a computation λ and a position $i \geq 0$, we use $\lambda[i]$, $\lambda[0,i]$, and $\lambda[i,\infty]$ to denote the i-th state in λ, the finite prefix $q_0,q_1,\ldots,q_i$ of λ, and the infinite suffix $q_i,q_{i+1},\ldots$ of λ, respectively.

Alternating-Time Temporal Logic. The *Alternating-time Temporal Logic* [AHK97] (ATL for short) is defined with respect to a finite set Π of *propositions* and a finite set Σ of *players*. An ATL formula is one of the following:

- p, for propositions $p \in \Pi$.
- $\neg\varphi$ or $\varphi_1 \vee \varphi_2$, where φ, φ_1, and φ_2 are ATL formulas.
- $\langle\langle A\rangle\rangle \bigcirc \varphi$, $\langle\langle A\rangle\rangle \Box\varphi$, or $\langle\langle A\rangle\rangle \varphi_1 \,\mathcal{U} \varphi_2$, where $A \subseteq \Sigma$ is a set of players, and φ, φ_1, and φ_2 are ATL formulas.

The operator $\langle\langle\ \rangle\rangle$ is a *path quantifier*, and $\bigcirc$ ("next"), $\Box$ ("always"), and $\mathcal{U}$ ("until") are *temporal operators*. The logic ATL is similar to the branching-time temporal logic CTL, only that path quantifiers are parameterized by sets of players. In fact, the parameterized path quantifier $\langle\langle A\rangle\rangle$ is a generalization of the path quantifiers of CTL: the existential path quantifier $\exists$ corresponds to $\langle\langle\Sigma\rangle\rangle$, and the universal path quantifier $\forall$ corresponds to $\langle\langle\emptyset\rangle\rangle$. Sometimes we write $\langle\langle a_1,\ldots,a_n\rangle\rangle$ instead of $\langle\langle\{a_1,\ldots,a_n\}\rangle\rangle$. Additional boolean connectives are defined from $\neg$ and $\vee$ in the usual manner. As in CTL, we write $\langle\langle A\rangle\rangle\Diamond\varphi$ for $\langle\langle A\rangle\rangle\, true\, \mathcal{U}\varphi$. Extensions of ATL as ATL* and the alternating μ-calculus are defined in [AHK97].

We interpret ATL formulas over the states of a given ATS S that has the same propositions and players. The labeling of the states of S with propositions is used to evaluate the atomic formulas of ATL. The logical connectives $\neg$ and $\vee$ have the standard interpretation. To evaluate a formula of the form $\langle\langle A\rangle\rangle\psi$ at a state q of S, consider the following two-player game. The game proceeds in an infinite sequence of rounds, and after each round, the position of the game is a state of S. The initial position is q. Now consider the game in some position u. To update the position, first the protagonist chooses for every player $a \in A$, a set $Q_a \in \delta(u,a)$. Then, the antagonist chooses a successor v of u such that $v \in Q_a$ for all $a \in A$, and the position of the game is updated to v. In this way, the game continues forever and produces a computation. The protagonist wins the game if the resulting computation satisfies the subformula ψ, read as a linear temporal formula whose outermost operator is $\bigcirc$, $\Box$, or $\mathcal{U}$. The ATL formula $\langle\langle A\rangle\rangle\psi$ holds at the state q if the protagonist has a winning strategy in this game.

In order to define the semantics of ATL formally, we first define the notion of strategies. Consider an ATS $S = \langle \Pi, \Sigma, Q, \pi, \delta\rangle$. A *strategy* for a player $a \in \Sigma$ is a mapping $f_a : Q^+ \to 2^Q$ such that for all $\lambda \in Q^*$ and all $q \in Q$, we have $f_a(\lambda\cdot q) \in \delta(q,a)$. Thus, the strategy f_a maps each finite prefix $\lambda\cdot q$ of a computation to a set in $\delta(q,a)$. This set contains possible extensions of the computation as suggested to player a by the strategy. Each strategy f_a induces a set of computations that player a can enforce. Given a state q, a set A of players, and a set $F_A = \{f_a \mid a \in A\}$ of strategies, one for each player in A, we define the *outcomes* of F_A from q to be the set $out(q, F_A)$ of all q-computations that the players in A can enforce when they cooperate and follow the strategies in F_A; that is, a computation $\lambda = q_0, q_1, q_2, \ldots$ is in $out(q, F_A)$ if $q_0 = q$ and for all positions $i \geq 0$, the state q_{i+1} is a successor of q_i satisfying $q_{i+1} \in \bigcap_{a \in A} f_a(\lambda[0, i])$.

We can now turn to a formal definition of the semantics of ATL. We write $S, q \models \varphi$ ("state q satisfies formula φ in the structure S") to indicate that the formula φ holds at state q of S. When S is clear from the context we omit it and write $q \models \varphi$. The relation $\models$ is defined, for all states q of S, inductively as follows:

- For $p \in \Pi$, we have $q \models p$ iff $p \in \pi(q)$.
- $q \models \neg\varphi$ iff $q \not\models \varphi$.
- $q \models \varphi_1 \vee \varphi_2$ iff $q \models \varphi_1$ or $q \models \varphi_2$.
- $q \models \langle\!\langle A \rangle\!\rangle \bigcirc \varphi$ iff there exists a set F_A of strategies, one for each player in A, such that for all computations $\lambda \in out(q, F_A)$, we have $\lambda[1] \models \varphi$.
- $q \models \langle\!\langle A \rangle\!\rangle \Box \varphi$ iff there exists a set F_A of strategies, one for each player in A, such that for all computations $\lambda \in out(q, F_A)$ and all positions $i \geq 0$, we have $\lambda[i] \models \varphi$.
- $q \models \langle\!\langle A \rangle\!\rangle \varphi_1 \mathcal{U} \varphi_2$ iff there exists a set F_A of strategies, one for each player in A, such that for all computations $\lambda \in out(q, F_A)$ there exists a position $i \geq 0$ such that $\lambda[i] \models \varphi_2$ and for all positions $0 \leq j < i$, we have $\lambda[j] \models \varphi_1$.

Note that the next operator $\bigcirc$ is local: $q \models \langle\!\langle A \rangle\!\rangle \bigcirc \varphi$ iff for each player $a \in A$ there exists a set $Q_a \in \delta(q, a)$ such that for each state $q' \in \bigcap_{a \in A} Q_a$, if q' is a successor of q, then $q' \models \varphi$.

Multiplayer Modeling Language. Modeling a protocol directly with an alternating transition system would be cumbersome. Instead we use a simple modeling formalism based on Dijkstra's guarded command language, previously introduced in [HMMR00]. This simple modeling language is a subset of the reactive modules used by the verification tool MOCHA. Let X be a set of variables interpreted over some possibly infinite domains. We denote by $X' = \{x' \mid x \in X\}$ the set of variables obtained by priming each variable in X. Those variables are used to refer to the value of the variable in the next state. A valuation v of the variables X is a function which maps each of the variables in X into a value in its domain. Denote by $\mathcal{V}_X$ the set of all possible valuations of the variables X. For any valuation v, we write v' the valuation obtained by priming each domain variable of v. For any predicate φ over the variables X and a valuation $v \in \mathcal{V}_X$, denote by $\varphi[\![v]\!]$ the truth value of the predicate with all the free variables interpreted according to the valuation v.

Given a set of program variables X, a Y-*action* ξ (guarded command) has the form $[\!]$ *guard* $\rightarrow$ *update*, where $Y \subseteq X$, the guard *guard* is a boolean predicate over X, and the update relation *update* is a boolean predicate over $X \cup Y'$. We also write $guard_\xi$ and $update_\xi$ to represent the guard and update relation of ξ respectively. Given a valuation $v \in \mathcal{V}_X$, the action ξ is said to be *enabled* if the guard $guard_\xi[\![v]\!]$ evaluates to true. We assume that for every Y-action, the update relation is functional, that is, for all valuation functions $v \in \mathcal{V}_X$, there is exactly one valuation function $u \in \mathcal{V}_Y$ such that $update_\xi[\![v \cup u']\!]$ holds. A Y-*process* Φ is a finite set of Y-actions. We say that the set of variables Y is *the controlled variables* of Φ. We require that for any valuation of the program variables, at least one action in the process is enabled.

To model multi-player games, we partition the program variables X into disjoint sets $X_1, X_2, \ldots, X_n$, i.e., $X = X_1 \uplus X_2 \uplus \ldots \uplus X_n$. Intuitively, X_i contains the variables updated by player i. A *program* $P = (\Phi_1, \Phi_2, \ldots, \Phi_n)$ over X is a tuple of processes such that Φ_i is an X_i-process. Each process of the program can be defined by composing smaller processes, each of which controls a subset of the variables. For two processes Φ_1 and Φ_2 with disjoint controlled variables, their *composition* $\Phi_1 \| \Phi_2$ is the process $\Phi = \{[\!]\ guard_\xi \wedge guard_\eta \rightarrow update_\xi \wedge update_\eta \mid \xi \in \Phi_1 \wedge \eta \in \Phi_2\}$.

The *semantics* of a program $P = \{\Phi_1, \Phi_2, \ldots \Phi_n\}$ over the program variables X is the alternating transition system S_p

- $\Sigma = \{1, 2, \ldots, n\}$ (the n players of the program P).
- $Q = \mathcal{V}_X$, i.e., a state of S_P is a valuation of the program variables.
- $\Delta = (\delta_1, \delta_2, \ldots, \delta_n)$ with $\delta_i : \mathcal{V}_X \to 2^{\mathcal{V}_X}$ is the transition function of player i and is defined as $\delta_i(v) = \{u \mid \xi \in \Phi_i \wedge guard_\xi[\![v]\!] \wedge update_\xi[\![v \cup u']\!]\}$.
- Π is a set of boolean predicates over the program variables X.
- π maps a state q, which is a valuation, to the set $\pi(q)$ of propositions that the valuation satisfies.

Intuitively, for $i \in \{1, 2, \ldots, n\}$, player i controls the actions in Φ_i.

A program is run in steps. At each step, player 1 chooses an enabled action $\xi \in \Phi_1$, and updates the values of the variables in X_1 according to the predicate $update_\xi$. Independently and simultaneously, player 2 chooses an enabled action $\eta \in \Phi_2$ and updates the values of the variables in X_2 according to $update_\eta$. And so on for the $n-2$ other players.

3 Formal Modelization of Non-Repudiation Protocols

Modeling of the protocols. The notation, classically used in literature to describe cryptographic protocols ($A \to B : m$, to denote that Alice sends a message m to Bob), has several drawbacks. The protocols are presented as a linear sequence of message exchanges, with a predefined order. In the case of optimistic exchange protocols often subprotocols can be invoked at different moments. Unfortunately, running a subprotocol at a time not foreseen by the designer, may have unexpected side-effects, threatening the security of a protocol if one of the participating entities tries to cheat the other entities. Therefore, we use the modeling language described above to model the exchange protocols. To execute a given action ξ, $guard_\xi$ must evaluate to true. The guard $guard_\xi$ is used to represent the elements (such as keys, messages, ...) necessary for a participating entity, that is a player in our modelization, to execute the action ξ. The variables controlled by a player represent his current knowledge and thus determine which actions he/she can execute. This specification takes into account all possible executions of subprotocols corresponding to a given initial knowledge, as we do not give any predefined order to these guarded commands. Thus, the specification enables a malicious entity to choose a different order of execution and construct a possible attack. We model the two players Alice and Bob with this approach.

The TTP is a special player and has to be modeled in a particular way. The TTP must be *impartial*, it may not help one or the other player. To make sure that the TTP does not have a strategy to help one of the players to cheat, we model the TTP in a deterministic way: at each stage of the execution of the protocol, the TTP executes the action requested by the protocol. This can easily be realized. Consider a protocol where the TTP has to execute the actions $\xi_1, \xi_2, ...\xi_n$ in this given order. We only have to use a n valued variable initialized to 1, each time that the TTP take an action this counter is incremented and the guards of the actions of the TTP ensure that only this sequence is possible.

The communication channels can also be modeled as players. Each transmission is modeled as a guarded command. We consider three different kinds of communication channels: unreliable, resilient, and operational. To specify *unreliable channels* we add an action that the channel can always take and has no effect, i.e. this is simply an idle action. As a consequence, a message send on a unreliable channel may never be delivered. A *resilient channel* can be specified in the same way, but in the presence of resilient channel, we only consider protocol runs where all the messages sent on the channel have been delivered. *Operational channels* do not have any additional action and thus immediately transmit the messages that have been sent on it. By immediately transmitting messages we alter the channel definition, which requires messages to be transmitted before a known constant time. This is however acceptable as the sender can always wait before sending the message to obtain the desired delay.

As an illustration, let us consider the following guarded commands:

$\vdots$

$[]\ \mathtt{La} \wedge \mathtt{Ta} \wedge \mathtt{Ca} \wedge \mathtt{EOOa} \wedge \neg\mathtt{STOPa} \wedge \neg\mathtt{SENDm1} \to \mathtt{SENDm1}' := \mathtt{true};$

$\vdots$

$[]\ \mathtt{SENDm1} \wedge \neg\mathtt{M1} \to \mathtt{M1}' := \mathtt{true};$

$\vdots$

$[]\ \mathtt{M1} \wedge \neg\mathtt{EOOb} \wedge \neg\mathtt{STOPb} \to \mathtt{Lb}' := \mathtt{true}; \mathtt{Tb}' := \mathtt{true}; \mathtt{Cb}' := \mathtt{true}; \mathtt{EOOb} := \mathtt{true};$

$\vdots$

We suppose that the first line belongs to Alice's specification, the second to a communication channel and the third one to Bob. The first guarded command expresses that "if Alice has knowledge of `L`, `T`, `C` and `EOO`, she can send message 1". The second guarded command models that the communication channel deliver the message 1 (`M1` becomes true). And the last guarded command expresses that Bob increases his knowledge when he receives the message `M1`.

Modeling of requirements. We show here how the main requirements that an exchange protocol must fulfill, can be *naturally* rephrased as the existence of strategies for the participating entities to reach their goal. The logic ATL is used to formalize those requirements. We will concentrate here on properties of non-repudiation protocols.

In a first approximation, we introduced fairness as the property that either both entities receive all their desired evidences or none of them receives any valuable evidence. We can now formulate this requirement as the existence of strategies. We say that the protocol is fair for Alice if "Bob and the communication channels do not have a strategy to reach a state where Bob has his proof of origin and Alice has no more a strategy to obtain her proof of receipt". This can be formally expressed by the following ATL formula:

$$\neg\langle\!\langle B, \mathsf{Com}\rangle\!\rangle\Diamond(\mathsf{NRO} \wedge \neg\langle\!\langle A\rangle\!\rangle\Diamond\mathsf{NRR}) \qquad (1)$$

Here, Com denotes the set of all communication channels. NRR and NRO respectively denote the non-repudiation of receipt and origin evidences. This is a generic notation as the form of those evidences depends on a particular protocol. The expressive power of the logic ATL is well illustrated by this example. Cooperation and adversarity are naturally expressed. To better understand the advantages of using a game logic to formalize this requirement, let us consider the following CTL formula:

$$\neg\exists\Diamond(\mathsf{NRO} \wedge \neg\exists\Diamond\mathsf{NRR}) \qquad (2)$$

that may look as an appropriate CTL candidate for the formalization of fairness for Bob. Note that we have obtained this formula by replacing the two teams in (1) by the entire set of players Σ. Note also that the formula (2) can be rewritten in the equivalent and more readable positive form:

$$\forall\Box(\mathsf{NRO} \to \exists\Diamond\mathsf{NRR}) \qquad (3)$$

That says "on every state of every run of the protocol, if Bob has his evidence of origin then there should exist a continuation of the protocol on which Alice eventually receives her evidence of receipt". This way, we have lost however the ability of distinguishing between cooperative and adversarial behaviors. Let us show what are some consequences of this, so to say, "loss of precision". For example, it may be the case that the formula (3) is verified because in the course of the protocol a *resilient* channel helps Alice to obtain her proof by delivering a message within a given bound (which is not generally the case for such a channel). As this assumption of cooperative behavior of the resilient channels can not be made in general, in order to be fair the protocol should guarantee Alice to obtain her proof even if the resilient channel does not deliver the message within certain bound. This fact is not ensured by the CTL formalization but is ensured by our ATL formalization. In fact, in formula (1), it is explicitly stated that the communication channels play against Alice. A resilient channel has only the obligation to deliver the message after a finite amount of time and not within a given bound, even if that helps Alice to obtain her proof. This non-cooperative aspect is precisely formalized in ATL. Also formula (2) may be verified because there exists some paths, where Bob is honest and allows Alice to obtain her proof. Again, the protocol should not make the hypothesis that Bob is honest, and should ensure fairness for Alice even if Bob is trying to cheat her. This adversarial behavior is formally modeled by our ATL formula and can not be formalized in CTL. Now let us take the other possibility: the CTL requirement can be false even if the ATL formula is verified. This can occur in the following situation: as we consider all reachable states where NRO is true, we may consider states where Alice, by its own fault, has reached a situation where she is not able to get her evidence anymore. This may happen if, for example, Alice has sent the origin evidence to Bob and decides to quit the protocol. This problem does not occur in our ATL formalization as by using the notion of strategy, we "automatically" exclude behaviors of Alice that does not serve her objective. Note that this last comment also rules out the following CTL formula as a candidate for fairness:

$$\forall\Box(\mathsf{NRO} \rightarrow \forall\Diamond\mathsf{NRR}) \tag{4}$$

Similar problems as those related to the CTL specification have been encountered in [SM99]. To complete the fairness requirement, we express fairness for Bob as follows:

$$\neg\langle\langle A, \mathsf{Com}\rangle\rangle\Diamond(\mathsf{NRR} \wedge \neg\langle\langle B\rangle\rangle\Diamond\mathsf{NRO}) \tag{5}$$

and take the conjunction of (1) and (5). We are also interested in first verifying a basic (weaker) notion of fairness: "Alice and Bob cooperate to achieve fairness". This would be translated by the following ATL formula:

$$\langle\langle A, B\rangle\rangle\Diamond((\mathsf{NRO} \wedge \mathsf{NRR}) \vee (\neg\mathsf{NRO} \wedge \neg\mathsf{NRR})) \tag{6}$$

Viability, which can be seen as a variation of the property above, is expressed by the following formula:

$$\langle\langle A, B\rangle\rangle\Diamond(\mathsf{NRO} \wedge \mathsf{NRR}) \tag{7}$$

that says "Alice and Bob can cooperate, so they are honest (i.e. they are following the protocol), and in that case the protocol should allow them, even in presence of non-cooperating channels, to obtain their respective evidences". And finally timeliness is formalized by :

$$\langle\langle A\rangle\rangle\Diamond(\mathsf{stop}_A \wedge (\neg\mathsf{NRR} \rightarrow \neg\langle\langle B\rangle\rangle\Diamond\mathsf{NRO})) \tag{8}$$

which expresses that "Alice has a strategy to finish the protocol and if she does not have her evidence at that point, Bob will not be able to obtain his evidence neither". The same requirement can be expressed for Bob.

4 Verification with Mocha

We have applied the proposed verification technique, to the optimistic Zhou-Gollman protocol [ZG97]. We have used the model-checker Mocha [AHM+98] that supports the logic ATL. The interested reader can find the complete system specification in appendix. First we will present the protocol using the classic, arrow based, notation.

We use the following notation to describe the protocol:

- $X \rightarrow Y$: transmission from entity X to entity Y
- $X \leftrightarrow Y$: ftp get operation performed by X at Y
- $h()$: a collision resistant one-way hash function
- $E_k()$: a symmetric-key encryption function under key k

- $D_k()$: a symmetric-key decryption function under key k
- $S_X()$: the signature function of entity X
- m: the message sent from A to B
- k: the message key A uses to cipher m
- $c = E_k(m)$: the cipher of m under the key k
- $l = h(m, k)$: a label to identify a protocol run
- f: a flag indicating the purpose of a message
- t: the time-out chosen by A
- $\mathsf{EOO} = S_A(f_{\mathsf{EOO}}, B, l, t, c)$: the evidence of origin of c
- $\mathsf{EOR} = S_B(f_{\mathsf{EOR}}, A, l, t, c)$: the evidence of receipt of c
- $\mathsf{EOO}_k = S_A(f_{\mathsf{EOO}_k}, B, l, t, k)$: the evidence of origin of k
- $\mathsf{EOR}_k = S_B(f_{\mathsf{EOR}_k}, A, l, t, k)$: the evidence of receipt of k
- $\mathsf{Sub}_k = S_A(f_{\mathsf{Sub}_k}, B, l, k)$: the evidence of submission of k
- $\mathsf{Con}_k = S_{\mathsf{TTP}}(f_{\mathsf{Con}_k}, A, B, l, t, k)$: the evidence of confirmation of k issued by the TTP

The protocol is divided into two subprotocols: the main protocol and a recovery protocol. As the protocol is based on the optimistic approach, the trusted third party (TTP) does only intervene in the recovery protocol. We start by describing the main protocol.

1. $A \to B : \ f_{\mathsf{EOO}}, B, l, t, c, \mathsf{EOO}$
2. $B \to A : \ f_{\mathsf{EOR}}, A, l, \mathsf{EOR}$
3. $A \to B : \ f_{\mathsf{EOO}_k}, B, l, k, \mathsf{EOO}_k$
4. $B \to A : \ f_{\mathsf{EOR}_k}, A, l, \mathsf{EOR}_k$

First Alice sends the digitally signed cipher $c = E_k(m)$ to Bob. In the second message Bob responds with the evidence of receipt for this cipher (EOR). If Alice does not receive the second transmission she stops the protocol, otherwise she sends the signed decryption key k to Bob. Bob answers by sending the receipt EOR_k for the key. The label l, present in each transmission, identifies the protocol run. The time-out t specified in message 1 is used in the recovery protocol. Alice may initiate the recovery protocol if Bob does not send the receipt for the key.

The steps of the recovery protocol are the following:

1. $A \to \mathsf{TTP} : \ f_{\mathsf{Sub}_k}, l, t, k, \mathsf{Sub}_k$
2. $B \leftrightarrow \mathsf{TTP} : \ f_{\mathsf{Con}_k}, A, B, l, t, k, \mathsf{Con}_k$
3. $A \leftrightarrow \mathsf{TTP} : \ f_{\mathsf{Con}_k}, A, B, l, t, k, \mathsf{Con}_k$

Alice sends the signed key, together with the deadline t to the TTP. If the key arrives after t, the TTP does not accept the recovery. Otherwise the TTP publishes the key together with a confirmation Con_k for the key in a read-only accessible directory, where both Alice and Bob can fetch the key as well as Con_k. Con_k serves to Bob as the evidence of origin of the key, and to Alice as the evidence of receipt of the key as it is accessible to Bob. The deadline t is necessary for Bob to know the moment when either the key is published or will not be published anymore. In this protocol, the non-repudiation of origin evidence NRO is composed of EOO and EOO_k or of EOO and Con_k. The non-repudiation of receipt evidence NRR is composed of EOR and EOR_k or of EOR and Con_k.

Zhou et al. suppose that the channels between Alice and Bob are unreliable and the channels between the TTP and both Alice and Bob are resilient. In our model we do not modelize the ftp get operation, but assume that the TTP takes the initiative to send messages 2 and 3 of the recovery protocol to Alice respectively Bob, as soon as these actions are possible.

To verify that the protocol satisfies a requirement, we proceed as follows. Assume that we want to verify the fairness requirement (1). So we have to establish if the protocol actually proposes a strategy to Alice in order to protect her against all possible cheating behaviors of

Bob and the communication channels. We construct the ATS that contains all possible behaviors of all players. Note that this ATS contains all possible behaviors of Bob and the communication channels, and so all the potential attacks that these entities can construct with the set of actions they possess. The model-checking algorithm delivers three informations:

- for each state q we know wether formula (1) is satisfied or not;
- for each state q where formula (1) is satisfied the model-checking algorithm also computes the subset of actions that Alice can take in order to ensure her fairness requirement, i.e. the truth of formula (1);
- for each state q where formula (1) is false, the algorithm computes the subset of choices that Bob and the communication channels can take in q in order to ensure that formula (1) is false, and so to cheat Alice.

We can exploit those informations as follows:

- If the formula (1) is false in the state that models the initial state of the protocol, it means that Alice does not have a strategy to maintain her fairness requirement, no matter what she does, wether she follows the protocol or not.
- Instead, if formula (1) is true in the state that models the initial state of the protocol, we must verify that the actions proposed to Alice by the protocol are the right ones. Formally this amounts to verifying that in each state the choices dictated by the protocol to Alice are a subset of the choices computed by the model-checking algorithm. If this is the case, then the protocol is safe against all the attacks that Bob and the communication channels can construct (with respect to the actions that they have at their disposal).

Unfortunately, the current version of Mocha does not yet provide the above described facilities. Therefore we have to use a slightly different approach for the verification. We have to modelize that a honest entity follows the protocol. Therefore we require the execution trace to include the transmission of several given messages. This is done by including the correct protocol messages in the requirement specification. In that way we can force a *honest* player to respect the predefined protocol execution, while a *cheating* player still has the possibility to execute all possible actions. We advance stepwise in the protocol, verifying that after each transmission of the honest player, fairness is still provided. Using this systematic investigation, we succeeded in finding a flaw in the protocol. Remember that the communication channels between the TTP and both Alice and Bob are resilient and that a finite time-out must be choosen by Alice at the begin of the protocol. In our model, the time-out is a boolean variable controlled by a player *Clock*. We restrict the time-out to become true after a finite amount of time, using a similar technique as used for transmission on resilient channels. In fact, when the main protocol has been executed until step 3, Bob has received all of his evidences and may decide to stop the protocol in order to try to cheat Alice. At this point we check wether Alice in cooperation with the clock does have a strategy against Bob who is cooperating with the communication channels, to receive her non-repudiation of receipt evidences. As Alice cooperates with the clock, she entirely controls the triggering of the time-out. Intuitively, it means that Alice can choose the time-out to occur as late as she wants, which represents the fact, that she chooses the time-out at the first step of the protocol. As Alice does not receive her evidence, she has to initiate a recovery. For the recovery protocol to be successful, the request needs to arrive before the time-out. However as communication channels can help Bob, this message can be delayed until the time-out occurs. Intuitively, a resilient channel has the capacity to delay messages: thus for each possible finite time-out value choosen by Alice, Bob (with the help of the communication channels) can delay long enough the delivery of the message in order for the TTP to reject the recovery request. However when we change the communication channel to be operational (messages are transmitted immediately) the protocol becomes fair. The two properties are given in the appendix. The first one was established true by Mocha, showing an unfair situation in presence of resilient channels and the second was established false by Mocha showing a fair situation in presence of operational channels. Unfortunately, operational channels are rather unrealistic in practice.

If one of the properties is not respected, we may try to add players to the cooperation to detect the player(s) that make that the property is not verified. As an example, take a look at the flaw we described in the previous paragraph. We asked if Bob, together with all the communication channels, has a strategy to obtain the NRO evidence, such that Alice, in cooperation with the clock, does not have any strategy to receive the NRR evidence. Using Mocha we showed the existence of a flaw, by validating this property. However it may be difficult to determine the failure's origin. Thus we try to change the coalitions to find the player responsible for te flaw. By adding the communication channel between Alice and the TTP to Alice's coalition we succeeded in avoiding the property to be validated. Once we know that the communication channel is at the orign of the error it may be interesting to alter the channel's quality. In that way we see which modifications are needed to recover the fairness property. In this example we concluded that replacing the resilient channel by an operational channel was enough to prevent protocol failure. Hence, we have a rather easy and quick investigation method to locate and possibly prevent flaws.

5 Conclusions and Future Works

We have shown that exchange protocols, due to their particularities, are best modeled as games. First, the adversarial and cooperative behaviors that occur during the execution of those protocols are naturally and precisely expressed in term of strategies. Second, the main requirements that an exchange protocol must ensure are easily phrased as existence of strategies for the participating entities to reach well defined goals. We have proposed to use the framework of alternating transition systems and alternating-time temporal logic to formalize this view of exchange protocols as games. The great expressive power of alternating temporal logic in this context has been illustrated. The practical interest of the method has been demonstrated by showing how we can analyze very precisely the non-repudiation protocol due to Zhou-Gollman. For the analysis, we have used the tool Mocha which is able to automatically verify alternating temporal formula on alternating transition systems.

In the future, we will try to show that this approach is applicable to the analysis and verification of other, more complex, exchange protocols (fair exchange, digital contract signing, certified mail, ...). We will also investigate more theoretical implication related to the fact that we see those protocols as games. In particular, we will investigate the relevance of the notion of partial knowledge [Rei84] in the modeling of exchange protocols and its implication on the automatic verification of those protocols.

Acknowledgement. The authors would like to thank Thierry Massart for carefully reading a previous version of this paper and the members of the group "Vérification" of ULB, FUNDP and UMH for their comments on that work.

References

[AHK97] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 100–109. IEEE Computer Society Press, 1997.

[AHM+98] R. Alur, T.A. Henzinger, F.Y.C. Mang, S. Qadeer, S.K. Rajamani, and S. Tasiran. Mocha: modularity in model checking. In A.J. Hu and M.Y. Vardi, editors, *CAV 98: Computer-aided Verification*, Lecture Notes in Computer Science 1427, pages 521–525. Springer-Verlag, 1998.

[ASW97] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In Tsutomu Matsumoto, editor, *4th ACM Conference on Computer and Communications Security*, pages 6, 8–17, Zurich, Switzerland, April 1997. ACM Press.

[BAN96] M. Burrows, M. Abadi, and R. Needham. A logic of authentication, from proceedings of the royal society, volume 426, number 1871, 1989. In *William Stallings, Practical Cryptography for Data Internetworks, IEEE Computer Society Press, 1996*. 1996.

[BOGMR90] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.

[HMMR00] T.A. Henzinger, R. Manjumdar, F.Y.C. Mang, and J.-F. Raskin. Abstract interpretation of game properties. In *SAS 2000: Intertional Symposium on Static Analysis*, Lecture Notes in Computer Science. Springer-Verlag, (to appear) 2000.

[KM00] S. Kremer and O. Markowitch. Optimistic non-repudiable information exchange. In *21th Symp. on Information Theory in the Benelux*. Werkgemeenschap Informatie- en Communicatietheorie, Enschede, may 2000. to appear.

[Low95] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, November 1995.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. *Lecture Notes in Computer Science*, 1055:147–157, 1996.

[Mea96] C. A. Meadows. Analyzing the Needham-Schroeder public-key protocol: A comparison of two approaches. *Lecture Notes in Computer Science*, 1146:351–365, 1996.

[MR99] O. Markowitch and Y. Roggeman. Probabilistic non-repudiation without trusted third party. In *Second Conference on Security in Communication Networks'99*, Amalfi, Italy, September 1999.

[Pau97] L. C. Paulson. Proving properties of security protocols by induction. In *Proceedings of the 10th Computer Security Foundations Workshop*, pages 70–83. IEEE Computer Society Press, June 1997.

[Rei84] J.H. Reif. The complexity of two player games of incomplete information. *Computer and System Science*, 29(2):274–301, 1984.

[Sch98] S. Schneider. Formal analysis of a non-repudiation protocol. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop (CSFW '98)*, pages 54–65, Washington - Brussels - Tokyo, June 1998. IEEE.

[SM99] Vitaly Shmatikov and John C. Mitchell. Analysis of a fair exchange protocol. In *Proceedings of the 1999 FLoC Workshop on Formal Methods and Security Protocols*, Trento, Italy, July 1999.

[ZG97] Y. Zhou and D. Gollmann. An efficient non-repudiation protocol. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

Appendix: MOCHA model and properties for fairness

```
module Alice
  external M2,M4,R2,Initial : bool
  interface Ma,Ka,La,Ta,Ca,EOOa,EOOka,EORa,EORka,SUBka,CONka,SENDm1,SENDm3,
SENDr1,STOPa : bool
  atom controls
                Ma,Ka,La,Ta,Ca,EOOa,EOOka,EORa,EORka,SUBka,CONka,SENDm1,SENDm3,
                SENDr1,STOPa reads M2,M4,R2,Ma,Ka,La,Ta,Ca,EOOa,EOOka,EORa,
                                   EORka,SUBka,CONka,SENDm1,SENDm3,
                                   SENDr1,STOPa,Initial
        init
        [] true -> Ma':=true;Ka':=true;La':=true;Ta':=true;
                        Ca':=false;EOOa':=false;EORa':=false;EORka':=false;
                        SUBka':=false;CONka':=false;SENDm1':=false;
                        SENDm3':=false;SENDr1':=false;STOPa':=false;
                        EOOka':=false
        update
        [] ~Initial & Ma & Ka & ~Ca & ~ STOPa -> Ca':=true
        [] ~Initial & La & Ta & Ca & ~EOOa & ~STOPa -> EOOa':=true
        [] ~Initial & La & Ta & Ka & ~EOOka & ~STOPa -> EOOka':=true
        [] ~Initial & La & Ta & Ka & ~SUBka & ~STOPa -> SUBka':=true
        [] ~Initial & La & Ta & Ca & EOOa & ~STOPa & ~SENDm1 -> SENDm1':=true
        [] ~Initial & La & Ka & EOOka & ~SENDm3 & ~STOPa -> SENDm3':=true
        [] ~Initial & La & Ta & Ka & SUBka & ~SENDr1 & ~STOPa -> SENDr1':=true
        [] ~Initial & M2 & ~EORa -> EORa':=true; La':=true
        [] ~Initial & M4 & ~EORka -> EORka':=true; La':=true
        [] ~Initial & R2 & ~CONka & ~STOPa -> La':=true; Ta':=true;
                        Ka':=true; CONka':=true
        [] ~Initial & ~STOPa -> STOPa':=true
        [] true ->
  endatom
endmodule

module Bob
  external M1,M3,R3,Initial : bool
  interface Mb,Kb,Lb,Tb,Cb,EOOb,EOOkb,EORb,EORkb,SUBkb,CONkb,SENDm2,SENDm4,
                STOPb : bool
  atom controls Mb,Kb,Lb,Tb,Cb,EOOb,EOOkb,EORb,EORkb,SUBkb,CONkb,SENDm2,
                SENDm4,STOPb reads M1,M3,R3,Mb,Kb,Lb,Tb,Cb,EOOb,EOOkb,EORb,
                               EORkb,SUBkb,CONkb,SENDm2,SENDm4,STOPb,Initial
        init
        [] true -> Mb':=false;Kb':=false;Lb':=false;Tb':=false;Cb':=false;
                        EOOb':=false;EOOkb':=false;EORb':=false;
                        EORkb':=false;SUBkb':=false;CONkb':=false;
                        SENDm2':=false;SENDm4':=false;STOPb':=false
        update
        [] ~Initial & Cb & Kb & ~Mb & ~STOPb -> Mb':=true
        [] ~Initial & Lb & Cb & Tb & ~EORb & ~STOPb -> EORb':=true
        [] ~Initial & Lb & Tb & Kb & ~EORkb & ~STOPb -> EORkb':=true
        [] ~Initial & Lb & EORb & ~SENDm2 & ~STOPb -> SENDm2':=true
        [] ~Initial & Lb & EORkb & ~SENDm4 & ~STOPb -> SENDm4':=true
        [] ~Initial & M1 & ~EOOb & ~STOPb -> Lb':=true; Tb':=true; Cb':=true;
                                   EOOb':=true
        [] ~Initial & M3 & ~EOOkb & ~STOPb -> Lb':=true; Kb':=true;
                                   EOOkb':=true
        [] ~Initial & R3 & ~CONkb & ~STOPb -> Lb':=true; Tb':=true; Kb':=true;
                                   CONkb':=true
```

```
        [] ~Initial & ~STOPb -> STOPb':=true
        [] true ->
  endatom
endmodule

module TTP
  external TimeOut,R1,Initial : bool
  interface Kt,Lt,Tt,SUBkt,CONkt,SENDr23 : bool
  atom controls Kt,Lt,Tt,SUBkt,CONkt,SENDr23 reads TimeOut,R1,
                        Kt,Lt,Tt,SUBkt,CONkt,SENDr23,Initial
        init
        [] true -> Kt':=false;Lt':=false;Tt':=false;SUBkt':=false;
                        CONkt':=false;SENDr23':=false
        update
        [] ~Initial & R1 & ~TimeOut & ~SUBkt -> Lt':=true; Tt':=true;
                                        Kt':=true; SUBkt':=true
        [] ~Initial & Lt & Tt & Kt & SUBkt & ~CONkt -> CONkt':=true
        [] ~Initial & TimeOut & Lt & Tt & Kt & CONkt
                                & ~SENDr23 -> SENDr23':=true
  endatom
endmodule

module OperationalComTAB
  external SENDr1,SENDr23,Initial : bool
  interface R1,R2,R3 : bool
  atom controls R1,R2,R3 reads SENDr1,SENDr23,R1,R2,R3,Initial
        init
        [] true -> R1':=false;R2':=false;R3':=false
        update
        [] ~Initial & SENDr1 & ~R1 -> R1':=true
        [] ~Initial & SENDr23 & (~R2 | ~R3) -> R2':=true; R3':=true
  endatom
endmodule

module Clock
  interface TimeOut,Initial : bool
  atom controls TimeOut,Initial reads TimeOut,Initial
          init
        [] true -> TimeOut':=false; Initial':=true
        update
        [] Initial -> Initial':=false
        [] ~Initial & true -> TimeOut':=true
        [] ~Initial & true ->
  endatom
endmodule

module UnreliableComAB
  external SENDm1,SENDm2,SENDm3,SENDm4,Initial : bool
  interface M1,M2,M3,M4 : bool
  atom controls M1,M2,M3,M4 reads SENDm1,SENDm2,SENDm3,SENDm4,
                                        M1,M2,M3,M4,Initial
        init
        [] true -> M1':=false;M2':=false;M3':=false;M4':=false
        update
        [] ~Initial & SENDm1 & ~M1 -> M1':=true
        [] ~Initial & SENDm2 & ~M2 -> M2':=true
        [] ~Initial & SENDm3 & ~M3 -> M3':=true
        [] ~Initial & SENDm4 & ~M4 -> M4':=true
```

```
        [] true ->
                -- Idle action
  endatom
endmodule

module UnreliableComTAB
  external SENDr1,SENDr23,Initial : bool
  interface R1,R2,R3 : bool
  atom controls R1,R2,R3 reads SENDr1,SENDr23,R1,R2,R3,Initial
        init
        [] true -> R1':=false;R2':=false;R3':=false
        update
        [] ~Initial & SENDr1 & ~R1 -> R1':=true
        [] ~Initial & SENDr23 & (~R2 | ~R3) -> R2':=true; R3':=true
        [] true ->
                -- Idle action
  endatom
endmodule


SystemUnreliableCom:=Alice||Bob||TTP||UnreliableComAB||UnreliableComTAB||Clock
SystemHybrid:=Alice||Bob||TTP||UnreliableComAB||OperationalComTAB||Clock

atl "unfairness-bob-resilient-system"
                E F ( M1 & M2 & M3 & ~TimeOut & ~STOPa &
                    <<Bob,UnreliableComAB,UnreliableComTAB>> F
                        (
                            ( EOOb & (EOOkb | CONkb) )
                              & ~<<Alice,Clock>> F
                                   (  TimeOut &
                                       (
                                           ( (SENDr1 => R1) & (SENDr23 => R2) )
                                                =>(EORa & (EORka | CONka))
                                       )
                                   )
                        )
                    );
-- passed


atl "unfairness-bob-operational-system"
                E F ( M1 & M2 & M3 & ~TimeOut & ~STOPa &
                     <<Bob,UnreliableComAB,OperationalComTAB>> F
                              (
                                 ( EOOb & (EOOkb | CONkb) )
                                  & ~<<Alice,Clock>>F
                                     ( TimeOut &
                                       (EORa & (EORka | CONka))
                                     )
                              )
                    );

-- failed
```