Model Checking Cooperation, Knowledge, and Time — A Case Study

Wiebe van der Hoek and Michael Wooldridge

Department of Computer Science
University of Liverpool,
Liverpool L69 7ZF, United Kingdom.
{wiebe,m.j.wooldridge}@csc.liv.ac.uk

Abstract

Alternating-time Temporal Logic (ATL) is a logic developed by Alur, Henzinger, and Kupferman for reasoning about coalitional powers in multi-agent systems. Since what agents can achieve in a specific state will in general depend on the knowledge they have in that state, in an earlier paper we proposed ATEL, an epistemic extension to ATL which is interpreted over Alternating-time Temporal Epistemic Transition Systems (AETS). The logic ATEL shares with its ancestor ATL the property that its model checking problem is tractable. However, while model checkers have been implemented for ATL, no model checker yet exists for ATEL, or indeed for epistemic logics in general. The aim of this paper is to demonstrate, using the *alternating bit protocol* as a detailed case study, how existing model checkers for ATL can be deployed to verify epistemic properties.

1 Introduction

We begin with an informal overview of this paper, intended for a non-computing audience; we then give a detailed formal introduction.

This paper is about the use of computers for automatically proving properties of game-like multiagent encounters. Specifically, we describe how a technique called *model checking* can be used for this purpose. Model checking is an approach that was developed by computer scientists for verifying that computer programs are *correct* with respect to a specification of what they should do. Model checking is the subject of considerable interest within the computer science community because it is a fully automatic verification technique: that is, it can be fully automated by computer, and a computer program (a *model checker*) can be used to verify that another computer program is correct with respect to its specification.

Model checking systems take as input two entities: a system, whose correctness we wish to verify, and a statement from the requirements that we wish to check to hold in the system. For example, suppose we had written a computer program that was supposed to control a lift (elevator). Some typical requirements for such a system might be that 'two lifts will never react on the same call', and 'eventually, every call will be answered by a lift'. A model checking program would take as input the lift control program, these requirements (and potentially others), and would check whether or not the claims were correct with respect to the program; if not, then model checking programs give a *counter example*, to illustrate why not (see Figure 1). The latter property is of course especially useful in debugging programs.

Most existing model checking systems allow claims about the system's behaviour to be written in the language of *temporal logic*, which is well-suited to capturing the kind of requirement expressed

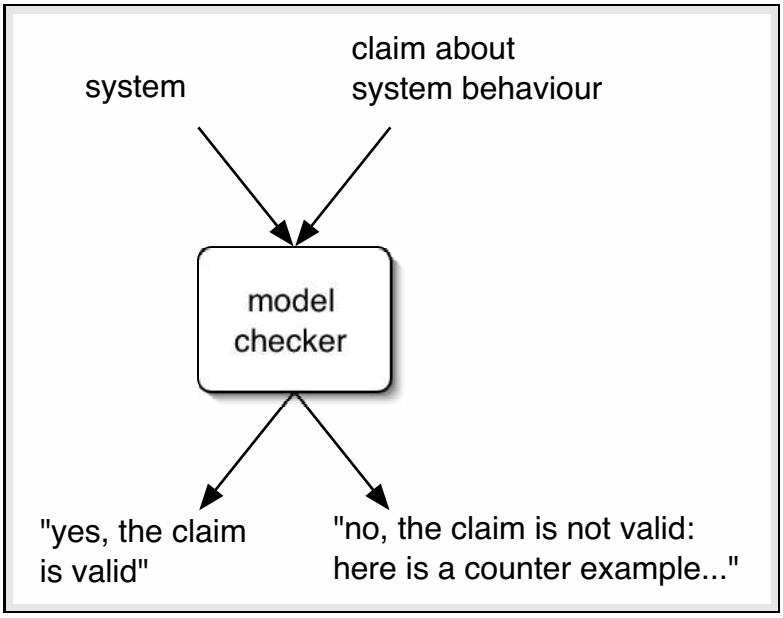

Figure 1: A model checking system takes as input a system to be checked, and logical formulae representing claims about the behaviour of the system, and verifies whether or not the system is correct with respect to these claims; if not, a counter example is produced.

above. However, the language Alternating-time Temporal Logic (ATL) is an extension to the temporal logic, which allows a system designer to express claims about the *cooperation* properties of a system [2]. Returning to the above example, we might want to verify that 'the lifts 1, 2 and 3 can establish that there is always a lift below the n-th floor'. ATL is thus especially suitable for *strategic* reasoning in many kinds of games: it allows to express properties concerning what certain coalitions of players can or cannot achieve. An even richer verification language is obtained when also *knowledge* properties of the agents involved are at stake: the recently introduced language ATEL adds an epistemic dimension to ATL, allowing one to express for instance 'if one lift knows that the other lifts are all below the n-th floor, it will immediately respond to a call from the $n + k$-th floor, and let the others know that it doing so'. Although no model checking systems exist for ATEL, a model checker for ATL has been implemented [3, 1].

Thus, our aim in this paper is to show how model checking, a technique developed by computer scientists for showing that computer programs are correct, can be applied to the problem of proving that game-like multiagent encounters have certain properties. We do this by means of a case study: the *alternating bit protocol*. This is a communications protocol that is designed to allow two agents to reliably communicate a string of bits between each other over a potentially faulty communications channel. The language ATEL is a natural choice to express properties of the protocol, since a property like 'the sender and the receiver have a way to achieve that, no matter how the channel behaves, eventually they know that a particular bit has been received' is nothing less than a statement about cooperation, knowledge and time. Thus, in this paper, we describe the alternating bit protocol as a finite state system, then specify ATEL properties that we want to check on this, and explain how we can check such properties using the MOCHA model checking system for ATL.

In more detail, then, Alternating-time Temporal Logic (ATL) was introduced by Alur, Henzinger, and Kupferman as a language for reasoning about *concurrent game structures* [2]. In a concurrent game structure, nodes represent states of the system, and edges denote possible choices of agents. Thus ATL generalizes *branching time logics*, a successful approach to reasoning about distributed computer systems, of which Computation Tree Logic (CTL) is the best known example [5]. In CTL, however, edges

between states represent transitions between time points. In distributed systems applications, the set of all paths through a tree structure is assumed to correspond to the set of all possible computations of a system. CTL combines *path quantifiers* "A" and "E" for expressing that a certain series of events will happen on all paths and on some path respectively, with *tense modalities* for expressing that something will happen eventually on some path ($\Diamond$), always on some path ($\Box$) and so on. Thus, for example, by using CTL-like logics, one may express properties such as "on all possible computations, the system never enters a fail state", which is represented by the CTL formula $\mathsf{A}\Box\neg fail$.

ATL is a generalisation of CTL, in the sense that it replaces the flow of time by the outcome of possible strategies. Thus, we might express a property such as "agent 1 has a *winning strategy* for ensuring that the system eventually terminates". More generally, the path quantifiers of CTL are replaced in ATL by modalities for *cooperative powers*, as in: "agents 1 and 2 can cooperate to ensure that the system never enters a fail state", which would be denoted by $\langle\langle\{1,2\}\rangle\rangle\,\Box\neg fail$. It is not possible to capture such statements using CTL-like logics. The best one can do is either state that something will inevitably happen, or else that it may possibly happen; CTL-like logics have *no notion of agency*. Incorporating such cooperative powers in the language makes ATL a suitable framework to model situations where strategic reasoning is crucial, as in multiagent systems in general [16], and in particular, games [14]. Note that ATL is, in a precise sense, a generalisation of CTL: the path quantifiers A ("on all paths…") and E ("on some paths…") can be simulated in ATL by the cooperation modalities $\langle\langle\emptyset\rangle\rangle$ ("the empty set of agents can cooperate to…") and $\langle\langle\Sigma\rangle\rangle$ ("the grand coalition of all agents can cooperate to…"). But between these extremes, we can express much more about what certain agents and coalitions can achieve.

In [11], we enriched ATL with epistemic operators [7, 13], and named the resulting system Alternating-time Temporal Epistemic Logic (ATEL). This seems a natural way to proceed: given the fact that ATL is tailored to reason about strategic decision problems, one soon wants to reason about the information that the agents have available upon which to base their decisions. ATEL is a succinct and very powerful language for expressing complex properties of multiagent systems. For example, the following ATEL formula expresses the fact that common knowledge in the group of agents Γ about φ is sufficient to enable Γ to cooperate and ensure ψ: $C_\Gamma\varphi \rightarrow \langle\langle\Gamma\rangle\rangle\Diamond\psi$. As another example, involving both a knowledge pre- and post-condition, the following ATEL formula says that, if a knows that φ, then a has a strategy to ensure that b knows φ — in other words, a can communicate what it knows to other agents: $K_a\varphi \rightarrow \langle\langle a\rangle\rangle\Diamond K_b\varphi$. For (many) other examples, we refer the reader to [11].

Although the computational complexity of the deductive proof problem for CTL is prohibitively expensive (it is EXPTIME-complete [5, p.1037]), the *model checking* problem for CTL — the problem of determining whether a given formula of CTL is satisfied in a particular CTL model — is comparatively easy: it can be solved in time $O(|M| \times |\varphi|)$, where $|M|$ is the size of the model and $|\varphi|$ is the size of the formula to be checked [5, p.1044]. Results reported in [2] and [11] show that the model checking problem remains PTIME-complete when moving to ATL and its epistemic variant ATEL, respectively. The tractability of CTL model checking has led to the development of a range of CTL model checking tools, which have been widely used in the verification of hardware and software systems [4]. Using the techniques pioneered in these systems, the MOCHA model checking system for ATL was implemented [3, 1]. However, this tool does not straightforwardly allow one to deal with epistemic extensions, as in ATEL. The aim of this paper is to demonstrate, by means of detailed case study, how one might put MOCHA to work in order to check properties in a system where reasoning about knowledge is essential. We verify properties of a communication protocol known as the *alternating bit protocol* [9]. In this protocol, the knowledge that one agent has about the knowledge of the other party is essential.

The remainder of the paper is structured as follows. In Section 2, we briefly re-introduce ATEL, beginning with a review of the semantic structures that underpin it. In section 3, we give an outline of the alternating bit protocol, and we show how it is possible to verify ATEL properties of this protocol MOCHA [3, 1].

2 Alternating-time Temporal Epistemic Logic (ATEL)

We begin by introducing the semantic structures used to represent our domains. These structures are a straightforward extension of the alternating transition systems used by Alur and colleagues to give a semantics to ATL. Formally, an *alternating epistemic transition system* (AETS) is a tuple

$$\langle \Pi, \Sigma, Q, \sim_{a_1}, \ldots, \sim_{a_n}, \pi, \delta \rangle,$$

where:

- Π is a finite, non-empty set of *atomic propositions*;
- $\Sigma = \{a_1, \ldots, a_n\}$ is a finite, non-empty set of *agents* ;
- Q is a finite, non-empty set of *states*;
- $\sim_a \subseteq Q \times Q$ is an *epistemic accessibility relation* for each agent $a \in \Sigma$ — we usually require that each $\sim_a$ is an equivalence relation;
- $\pi : Q \to 2^{\Pi}$ gives the set of primitive propositions satisfied in each state;
- $\delta : Q \times \Sigma \to 2^{2^Q}$ is the system transition function, which maps states and agents to the choices available to these agents. Thus $\delta(q, a) = \{Q_1, \ldots Q_k\}$ with $Q_i \subseteq Q$ $(i \leq k)$ is the set of choices available to agent a when the system is in state q. If a makes choice Q_i, this means that the next state will be some $q' \in Q_i$: which state it exactly will be, depends on the other agents. We require that this function satisfy the requirement that the system is completely controlled by its component agents: for every state $q \in Q$, if each agent a_i chooses Q_{a_i}, then the combined choice $Q_{a_1} \cap \cdots \cap Q_{a_n}$ is a singleton. This means that if every agent has made his choice, the system is completely determined: there is exactly one next state. Note that this requirement implies that every agent has at least one choice to make in every state: for every q and every a, there is a $Q \neq \emptyset$, such that $Q \in \delta(q, a)$.

We denote the set of sequences over Q by Q^*, and the set of non-empty sequences over Q by Q^+. Moreover, a and a_i will be used as variables ranging over the set of agents Σ.

Epistemic Relations

To appreciate the next example, we already stipulate the truth condition for knowledge formulas $K_a\varphi$, which intuitively is read as 'agent a knows φ'. If S is an AETS with Q its set of states, and $\sim_a$ an equivalence on it, then

$$S, q \models K_a\varphi \text{ iff } \forall q'(q \sim_a q' \Rightarrow S, q' \models \varphi)$$

We sometimes omit S, when it is clear from context. The definition says that what an agent knows in a state q is exactly what is true in all the states q' that cannot be distinguished from q, by that agent.

Example 1 Consider the following AETS $\langle \Pi, \Sigma, Q, \sim_1, \ldots, \sim_n, \pi, \delta \rangle$ (see also Figure 2):

- Π, the set of atoms, comprises just the singleton $\{p\}$;
- $\Sigma = \{1, 2, e\}$ is the set of agents consisting of the environment e and two agents that want to communicate;
- $Q = \{q_1, \ldots q_8\}$;

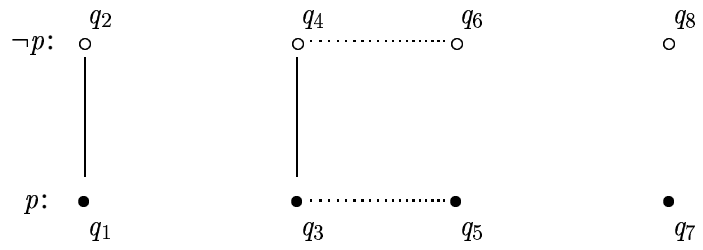

Figure 2: Closed bullets denote p-states, in open bullets, $\neg p$ is true. The epistemic accessibility relation of agent 1 is the reflexive closure of the straight lines, that of agent 2 is the reflexive closure of the dotted lines.

- We suppose that e is omniscient: $\sim_e = \{(q_i, q_i) \mid i \leq 8\}$. Agent 1 cannot distinguish q_1 from q_2, and also not q_3 from q_4, i.e., $\sim_1 = \sim_e \cup \{(q_1, q_2), (q_2, q_1), (q_3, q_4), (q_4, q_3)\}$. Accessibility for 2 is defined as $\sim_2 = \sim_e \cup \{(q_3, q_5), (q_5, q_3), (q_4, q_6), (q_6, q_4)\}$;
- π is such that $\pi(q_i)(p) = true$ iff i is odd ($i \leq 8$);
- $\delta : Q \times \Sigma \to 2^{2^Q}$ is given in Figure 3 and explained below.

To understand the example further, we anticipate it may be helpful to understand the model of Figure 2 in fact as three components: the first comprising q_1 and q_1, and the last consisting of q_7 and q_8. This is how knowledge evolves along the system:

1. Agent 2 has information whether p, in the first component. For the sake of the example, let us suppose that p is true, so the initial state is q_1. Note that we have:

$$q_1 \models p \wedge K_2 p \wedge \neg K_1 p \wedge K_1(K_2 p \vee K_2 \neg p) \wedge K_2(\neg K_1 p \wedge \neg K_1 \neg p)$$

   Thus: in q_1, agent 2 knows that p, agent 1 does not know p, but 1 knows that 2 *knows whether* p (i.e., 1 knows that 2 knows whether p is true or false); moreover, 2 knows that 1 does not know whether p.

2. Agent 2 then can choose to either send information about the status of p to agent 1 (in which case there is a transition to either q_3 or q_5), or to do nothing (leaving the system in q_1). In state q_5, the communication was apparently successful, although 2 does not know that:

$$q_5 \models K_1 p \wedge K_2 p \wedge \neg K_2 K_1 p \wedge \neg K_2 \neg K_1 p$$

   The environment e can choose the communication to fail, in which case agent 2's action leads us to q_3, where 2 knows p but 1 does not, and moreover agent 2 does not know whether 1 knows p:

$$q_3 \models K_2 p \neg K_1 p \wedge \neg K_2 K_1 p \wedge \neg K_2 \neg K_1 p$$

   Finally, q_7 models a state in which agent 2 successfully acknowledged receipt of the message p:

$$q_7 \models K_1 p \wedge K_2 p \wedge K_1 K_2 p \wedge K_2 K_1 p \wedge K_1 K_2 K_1 p$$

We have given part of the transition function δ in Figure 3. The description is not complete: we have for instance not stipulated what the choices are in the $\neg p$-states, neither did we constrain the options in the 'final' state q_7. Let us explain δ for the initial state q_1. In that state, the environment e cannot control

whether agent 2 will send the message p (explaining the fact that q_1 is a member of all of e's options $\{q_1, q_3\}$ and $\{q_1, q_5\}$, leaving the choice for q_1 so to speak up to agent 1), but *if* the message is sent, e can control whether the resulting state will be q_3 (message not arrived) or q_5 (message arrived). Agent 1, on the other hand, has the possibility in state q_1 either to not send the message p at all (in which case he chooses $\{q_1\}$), or to send the message, in which case he cannot control delivery (explaining why 1's second option in q_1 is $\{q_3, q_5\}$). Finally, agent 2 has no control over 1's choice at all, it leaves it to the other agents to choose among $\{q_1, q_3, q_5\}$.

$$
\begin{array}{llcl}
e1 & \delta(q_1, e) & = & \{\{q_1, q_3\}, \{q_1, q_5\}\} \\
e2 & \delta(q_3, e) & = & \{\{q_3\}, \{q_5\}\} \\
e3 & \delta(q_5, e) & = & \{\{q_5, q_7\}\} \\
 & & & \\
1.1 & \delta(q_1, 1) & = & \{\{q_1\}, \{q_3, q_5\}\} \\
1.2 & \delta(q_3, 1) & = & \{\{q_3, q_5, q_7\}\} \\
1.3 & \delta(q_5, 1) & = & \{\{q_3, q_5, q_7\}\} \\
 & & & \\
2.1 & \delta(q_1, 2) & = & \{\{q_1, q_3, q_5\}\} \\
2.2 & \delta(q_3, 2) & = & \{\{q_3, q_5\}\} \\
2.2 & \delta(q_5, 2) & = & \{\{q_5\}, \{q_7\}\}
\end{array}
$$

Figure 3: Partial description of the transition function e, 1 and 2.

Note that indeed, the system as a whole is deterministic: once all the agents have made their choices in q_1, the resulting state will either be q_1 (agent 1 choosing $\{q_1\}$, no matter what the others choose), or q_3 (e choosing $\{q_1, q_3\}$ and 1 choosing $\{q_3, q_5\}$) or q_5 (e choosing $\{q_1, q_5\}$ and 1 choosing $\{q_3, q_5\}$). Further note that no agent has the possibility to alter the truth of p, only the knowledge concerning p is updated.

Computations

For two states $q, q' \in Q$ and an agent $a \in \Sigma$, we say that state q' is an *a-successor* of q if there exists a set $Q' \in \delta(q, a)$ such that $q' \in Q'$. Intuitively, if q' is an a-successor of q, then q' is a possible outcome of one of the choices available to a when the system is in state q. We denote by $succ(q, a)$ the set of a successors to state q. We say that q' is simply a *successor* of q if for all agents $a \in \Sigma$, we have $q' \in succ(q, a)$; intuitively, if q' is a successor to q, then when the system is in state q, the agents Σ can cooperate to ensure that q' is the next state the system enters.

A *computation* of an AETS $\langle \Pi, \Sigma, Q, \sim_1, \ldots, \sim_n, \pi, \delta \rangle$ is an infinite sequence of states $\lambda = q_0, q_1, \ldots$ such that for all $u > 0$, the state q_u is a successor of q_{u-1}. A computation λ starting in state q is referred to as a *q-computation*; if $u \in \mathbb{N} = \{0, 1, \ldots\}$, then we denote by $\lambda[u]$ the u'th state in λ; similarly, we denote by $\lambda[0, u]$ and $\lambda[u, \infty]$ the finite prefix $q_0, \ldots, q_u$ and the infinite suffix $q_u, q_{u+1}, \ldots$ of λ respectively. We use $\cdot$ to concatenate a sequence of states λ with a state q to a new sequence $\lambda \cdot q$.

Example 1 (ctd). Possible computations starting in q_1 in our example are $q_1, q_1, q_1, \ldots$ (1 decides not to sent the message); $q_1, q_1, q_3, q_3, q_3, q_3, \ldots$ (1 sends the message at the second step, but it never arrives), and for instance $q_1, q_1, q_3, q_3, q_3, q_5, q_7, \ldots$ (after two steps, 1 sends the message five times in a row, after the fourth time it is delivered, and immediately after that 1 obtains an acknowledgement). Note that these computations do not alter the truth of p, but only knowledge about the status of p: in this sense, the dynamics can semantically be interpreted as changing accessibility relations, and not the states themselves.

Strategies and Their Outcomes

Intuitively, a *strategy* is an abstract model of an agent's decision-making process; a strategy may be thought of as a kind of plan for an agent. By *following* a strategy, an agent can bring about certain states of affairs. Formally, a strategy f_a for an agent $a \in \Sigma$ is a total function $f_a : Q^+ \to 2^Q$, which must satisfy the constraint that $f_a(\lambda \cdot q) \in \delta(q, a)$ for all $\lambda \in Q^*$ and $q \in Q$. Given a set $\Gamma \subseteq \Sigma$ of agents, and an indexed set of strategies $F_\Gamma = \{f_a \mid a \in \Gamma\}$, one for each agent $a \in \Gamma$, we define $out(q, F_\Gamma)$ to be the set of possible outcomes that may occur if every agent $a \in \Gamma$ follows the corresponding strategy f_a, starting when the system is in state $q \in Q$. That is, the set $out(q, F_\Gamma)$ will contain all possible q-computations that the agents Γ can "enforce" by cooperating and following the strategies in F_Γ. Note that the "grand coalition" of all agents in the system can cooperate to uniquely determine the future state of the system, and so $out(q, F_\Sigma)$ is a singleton. Similarly, the set $out(q, F_\emptyset)$ is the set of all possible q-computations of the system.

Alternating Temporal Epistemic Logic: Syntax

Alternating epistemic transition systems are the structures we use to model the systems of interest to us. Before presenting the detailed syntax of ATEL, we give an overview of the intuition behind its key constructs. ATEL is an extension of classical propositional logic, and so it contains all the conventional connectives that one would expect to find: $\wedge$ ("and"), $\vee$ ("or"), $\neg$ ("not"), $\rightarrow$ ("implies"), and so on. In addition, ATEL contains the temporal cooperation modalities of ATL, as follows. The formula $\langle\langle\Gamma\rangle\rangle \Box\varphi$, where Γ is a group of agents, and φ is a formula of ATEL, means that the agents Γ can work together (cooperate) to ensure that φ is always true. Similarly, $\langle\langle\Gamma\rangle\rangle \bigcirc \varphi$ means that Γ can cooperate to ensure that φ is true in the *next* state. The formula $\langle\langle\Gamma\rangle\rangle \varphi \mathcal{U} \psi$ means that Γ can cooperate to ensure that φ remains true *until* such time as ψ is true — and moreover, ψ *will* be true at some time in the future.

An ATEL formula, formed with respect to an alternating epistemic transition system $S = \langle \Pi, \Sigma, Q, \sim_1, \ldots, \sim_n, \pi, \delta \rangle$, is one of the following:

(S0) $\top$

(S1) p, where $p \in \Pi$ is a primitive proposition;

(S2) $\neg\varphi$ or $\varphi \vee \psi$, where φ and ψ are formulae of ATEL;

(S3) $\langle\langle\Gamma\rangle\rangle \bigcirc \varphi$, $\langle\langle\Gamma\rangle\rangle \Box\varphi$, or $\langle\langle\Gamma\rangle\rangle \varphi \mathcal{U} \psi$, where $\Gamma \subseteq \Sigma$ is a set of agents, and φ and ψ are formulae of ATEL;

(S4) $K_a\varphi$, where $a \in \Sigma$ is an agent, and φ is a formula of ATEL.

Alternating Temporal Epistemic Logic: Semantics

We interpret the formulae of ATEL with respect to AETS, as introduced in the preceding section. Formally, if S is an AETS, q is a state in S, and φ is a formula of ATEL over S, then we write $S, q \models \varphi$ to mean that φ is satisfied (equivalently, true) at state q in system S. The rules defining the satisfaction relation $\models$ are as follows:

- $S, q \models \top$
- $S, q \models p$ iff $p \in \pi(q)$ (where $p \in \Pi$);
- $S, q \models \neg\varphi$ iff $S, q \not\models \varphi$;
- $S, q \models \varphi \vee \psi$ iff $S, q \models \varphi$ or $S, q \models \psi$;

- $S, q \models \langle\langle\Gamma\rangle\rangle \bigcirc \varphi$ iff there exists a set of strategies F_Γ, one for each $a \in \Gamma$, such that for all $\lambda \in out(q, F_\Gamma)$, we have $S, \lambda[1] \models \varphi$;
- $S, q \models \langle\langle\Gamma\rangle\rangle \Box \varphi$ iff there exists a set of strategies F_Γ, one for each $a \in \Gamma$, such that for all $\lambda \in out(q, F_\Gamma)$, we have $S, \lambda[u] \models \varphi$ for all $u \in \mathbb{N}$;
- $S, q \models \langle\langle\Gamma\rangle\rangle \varphi \mathcal{U} \psi$ iff there exists a set of strategies F_Γ, one for each $a \in \Gamma$, such that for all $\lambda \in out(q, F_\Gamma)$, there exists some $u \in \mathbb{N}$ such that $S, \lambda[u] \models \psi$, and for all $0 \leq v < u$, we have $S, \lambda[v] \models \varphi$;
- $S, q \models K_a \varphi$ iff for all q' such that $q \sim_a q'$: $S, q' \models \varphi$.

Before proceeding, we introduce some derived connectives: these include the remaining connectives of classical propositional logic ($\bot \mathrel{\hat{=}} \neg\top$, $\varphi \to \psi \mathrel{\hat{=}} \neg\varphi \vee \psi$ and $\varphi \leftrightarrow \psi \mathrel{\hat{=}} (\varphi \to \psi) \wedge (\psi \to \varphi)$).

As well as asserting that some collection of agents is able to bring about some state of affairs, we can use the dual "$[\![\ldots]\!]$" to express the fact that a group of agents cannot *avoid* some state of affairs. Thus $[\![\Gamma]\!] \Diamond \varphi$ expresses the fact that the group Γ cannot cooperate to ensure that φ will never be true; the remaining agents in the system have a collection of strategies such that, if they follow them, φ may eventually be achieved. Formally, $[\![\Gamma]\!] \bigcirc \varphi$ is defined as an abbreviation for $\neg\langle\langle\Gamma\rangle\rangle \bigcirc \neg\varphi$, while $[\![\Gamma]\!] \Diamond \varphi$ is defined as an abbreviation for $\neg\langle\langle\Gamma\rangle\rangle \Box \neg\varphi$, and so on. Finally, we generally omit set brackets inside cooperation modalities, (i.e., writing $\langle\langle a, b\rangle\rangle$ instead of $\langle\langle\{a, b\}\rangle\rangle$), and we will usually write $\langle\langle\rangle\rangle$ rather than $\langle\langle\emptyset\rangle\rangle$.

Example 1 (ctd). In Example 1, the grand coalition can achieve that the agents have mutual knowledge about the status of p, but this cannot be achieved without one of its members. We can capture this property in ATEL as follows, letting φ denote $K_1 p \wedge K_2 p \wedge K_1 K_2 p \wedge K_2 K_1 p$:

$$q_1 \models \langle\langle e, 1, 2\rangle\rangle \Diamond \varphi \wedge \neg\langle\langle e, 1\rangle\rangle \Diamond \varphi \wedge \neg\langle\langle e, 2\rangle\rangle \Diamond \varphi \wedge \neg\langle\langle 1, 2\rangle\rangle \Diamond \varphi$$

First of all, note that φ is only true in q_7. Now, to explain the first conjunct in the displayed formula, it is easy to see that the computation $q_1, q_3, q_5, q_7, \ldots$ can be obtained by letting every agent play a suitable strategy. To see that, for instance, $\neg\langle\langle e, 1\rangle\rangle \Diamond \varphi$ is true in q_1, note that agent 2 has the power to always choose option $\{q_5\}$, preventing the other agents from having a strategy to reach q_7.

3 The Alternating Bit Protocol

We now present a case study of model checking an ATEL scenario, in the form of the *bit transmission problem*. We adapt our discussion of this problem from [13, pp.39–44]. The bit transmission protocol was first studied in the context of epistemic logic by Halpern and Zuck [9]. In game theory, similar problems are studied as the 'Electronic Mail Game' ([15]). The basic idea is that there are two agents, a sender S and a receiver R, who can communicate with one another through an unreliable communication environment. This environment may delete messages, but if a message does arrive at the recipient, then the message is correct. It is also assumed that the environment satisfies a kind of fairness property, namely that if a message is sent infinitely often, then it eventually arrives. The sender has a sequence of bits $X = \langle x_0, x_1, \ldots, \rangle$ that it desires to communicate to the receiver; when the receiver receives the bits, it prints them on a tape Y, which, when R has received k number of x_i's, is $Y = \langle y_0, y_2, \ldots y_k\rangle$. The goal is to derive a protocol that satisfies the safety requirement that the receiver never prints incorrect bits, and the liveness requirement that every bit will eventually be printed by the receiver.

More formally, given an input tape $X = \langle x_0, x_1, \ldots, \rangle$, with the x_i from some alphabet $Alph$, and a network in which deletion errors may occur but which is also fair, the aim is to find a protocol P for S and R that satisfies

- *safety* — at any moment the sequence Y written by R is a prefix of X; and
- *liveness* — every $x_i \in X$ will eventually be written by R.

Here, the alphabet for the tape is arbitrary: we may choose a tape X over any symbols. In our case study, we will choose $Alph = \{0, 1\}$. The obvious solution to this problem involves sending acknowledgement messages, to indicate when a message was received. Halpern and Zuck's key insight was to recognise that an acknowledgement message in fact carries information about the knowledge state of the sender of the message. This motivated the development of the following knowledge-based protocol. After obtaining the first bit (say $x_0 = 1$), the sender transmits it to the receiver. However, it cannot stop at this point, because for all it knows, the message may have been deleted by the environment. It thus continues to transmit the bit *until it knows the bit has been received*. At this point, the receiver knows the value of the bit that was transmitted, and the sender knows that the receiver knows the value of the bit — but the receiver does not know whether or not its acknowledgement was received. And it is important that the receiver knows this, since, if it keeps on receiving the bit 1, it does not know how to interpret this: either the acknowledgement of the receiver has not arrived yet (and the received 1 should be interpreted as a repetition of the sender's sending x_0), or the receiver's acknowledgement had indeed arrived, and the 1 that it is now receiving is the next entry x_1 on the tape, which just happens to be the same as x_0. So the sender repeatedly sends a second acknowledgement, until it receives back a third acknowledgement from the receiver; when it receives this acknowledgement, it starts to transmit the next bit. When the receiver receives this bit, this indicates that its final (third) acknowledgement was received.

A pseudo-code version of the protocol is presented in Figure 4 (from [13, pp.39–44]). Note that we write x_i as a shorthand for "the value of bit x_i". Thus $K_R(x_i)$ means that the receiver (R) knows the value of bit x_i. The variables $scount$ and $rcount$ range over the natural numbers. A command such as "send x until b" for a boolean b must be understood as: the process is supposed to repeatedly send x, until the boolean becomes true. The boolean is typically an epistemic formula: in line $S_{1.2}$ for instance, S keeps on sending x_{scount}, until it knows that R has received x_{scount}.

The protocol of Figure 4 is sometimes called a *knowledge-based program* [8], since it uses constructs (boolean tests and messages) from the epistemic language. Of course, conventional programming languages do not provide such constructs. So, let us consider how the protocol of Figure 4 can be translated into a normal program, on the fly also explaining the name of the protocol.

The first insight toward an implementation tells us that messages of the form $K_i\varphi$ ($i \in \{S, R\}$) can be seen as acknowledgements. Since there are three such messages, on lines $R_{1.2}, S_{1.3}$ and $R_{1.3}$, respectively, we assume we have three messages: $\mathtt{ack}^1, \mathtt{ack}^2$ and $\mathtt{ack}^3$. So, for instance, in $S_{1.3}$, 'send "$K_S K_R(x_{scount})$" ' is replaced by 'send $\mathtt{ack}^2$'. Next, we can replace the Boolean conditions of type $K_i\varphi$ by the condition that φ is received. Thus, $S_{1.3}$ gets replaced by "send $\mathtt{ack}^2$ until $\mathtt{ack}^3$ is received". Thus, the lines $S_{1.2}, S_{1.3}$, $R_{1.2}$ and $R_{1.3}$ of Figure 4 become, respectively:

$S_{1.2}$ send x_{scount} until $\mathtt{ack}^1$ is received
$S_{1.3}$ send $\mathtt{ack}^2$ until $\mathtt{ack}^3$ is received

$R_{1.2}$ send $\mathtt{ack}^1$ until $\mathtt{ack}^2$ is received
$R_{1.3}$ send $\mathtt{ack}^3$ until $x_{(rcount+1)}$ is received

Now, the *alternating bit protocol* is obtained by the following last step. Sender can put information on which bit he sends by adding a "colour" to it: say a 0 to every x_i for even i, and a 1 for every odd i. So the new alphabet $Alph'$ becomes $\{x \cdot 0 \mid x \in Alph\} \cup \{x \cdot 1 | x \in Alph\}$ We also now need only two acknowledgements: $\mathtt{ack}0$ and $\mathtt{ack}1$[1]. Thus, the idea in the alternating bit protocol is that S first

[1]Note that the specifi cation for this protocol in terms of epistemic operators would not require knowledge of depth four anymore, since some of the knowledge now goes in the colouring.

Sender

S_0 $scount := 0$

S_1 while **true** do

$S_{1.1}$ read x_{scount}

$S_{1.2}$ send x_{scount} until $K_S K_R(x_{scount})$

$S_{1.3}$ send "$K_S K_R(x_{scount})$" until $K_S K_R K_S K_R(x_{scount})$

$S_{1.4}$ $scount := scount + 1$

end-while

end-Sender

Receiver

R_0 when $K_R(x_0)$ set $rcount := 0$

R_1 while **true** do

$R_{1.1}$ write x_{rcount}

$R_{1.2}$ send "$K_R(x_{rcount})$" until $K_R K_S K_R(x_{rcount})$

$R_{1.3}$ send "$K_R K_S K_R(x_{rcount})$" until $K_R(x_{rcount+1})$

$R_{1.4}$ $rcount := rcount + 1$

end-while

end-Receiver

Figure 4: The bit transmission protocol (Protocol A).

sends $x_0 \cdot 0$. When it receives $\mathtt{ack}0$, it goes on to the next state on the tape and sends $x_1 \cdot 1$. R of course sees that this is a new message (since the colour is different), and acknowledges the receipt of it by $\mathtt{ack}1$, which S can also distinguish from the previous acknowledgement, allowing it to send $x_2 \cdot 0$ next. The only function we need is thus a colouring function $col : Alph \cup \{ack\} \to Alph' \cup \{ack0, ack1\}$, defined by

$$col(s, i) = \begin{cases} s \cdot 0 & \text{if } i \text{ is even} \\ s \cdot 1 & \text{otherwise.} \end{cases}$$

The protocol thus obtained is displayed in Figure 5.

3.1 Modelling the Protocol

In the Alternating Epistemic Transition System that we build for the protocol, we consider three agents: Sender S, Receiver R and an environment e. We are only interested in the knowledge of R and S. Naturally, the local state of S would be comprised of the following information: a scounter i and atoms $\mathtt{sent}x_i \cdot 0$, $\mathtt{sent}x_i \cdot 1$ for every $x \in Alph$, and, finally, two atoms $\mathtt{rcvd}a \cdot 0$ and $\mathtt{rcvd}a \cdot 1$, where x_i is the i-th bit of the input tape, and $a \cdot 0$ and $a \cdot 1$ are the two acknowledgement messages. This however would lead to an infinite system, because of the counter used. Hence, we don't consider this counter as part of the local state. Moreover, we also abstract from the value x. In other words, in the local state for S, we will identify any x_0 and x_0' ($x, x' \in Alph$). We do the same for R.

There is another assumption that helps us reduce the number of states: given the intended meaning of the atoms $\mathtt{sent}x \cdot 0$ (the last bit that has been sent by S is of the form $x \cdot 0$), we can impose as a general constraint that the property $\mathtt{sent}x \cdot 0 \leftrightarrow \neg\mathtt{sent}x \cdot 1$ should hold: agent S can only have sent one kind of message as the last one. The same applies to the atom-pairs ($\mathtt{sent}a \cdot 0$,$\mathtt{sent}a \cdot 1$), ($\mathtt{rcvd}x \cdot 0$,$\mathtt{rcvd}x \cdot 1$) and ($\mathtt{rcvd}a \cdot 0$,$\mathtt{rcvd}a \cdot 1$).

There are in general two ways one could deal with such a constraint. One would be not to impose it on the model, but instead verify that the protocol adheres to it, and prove, for instance, that there is

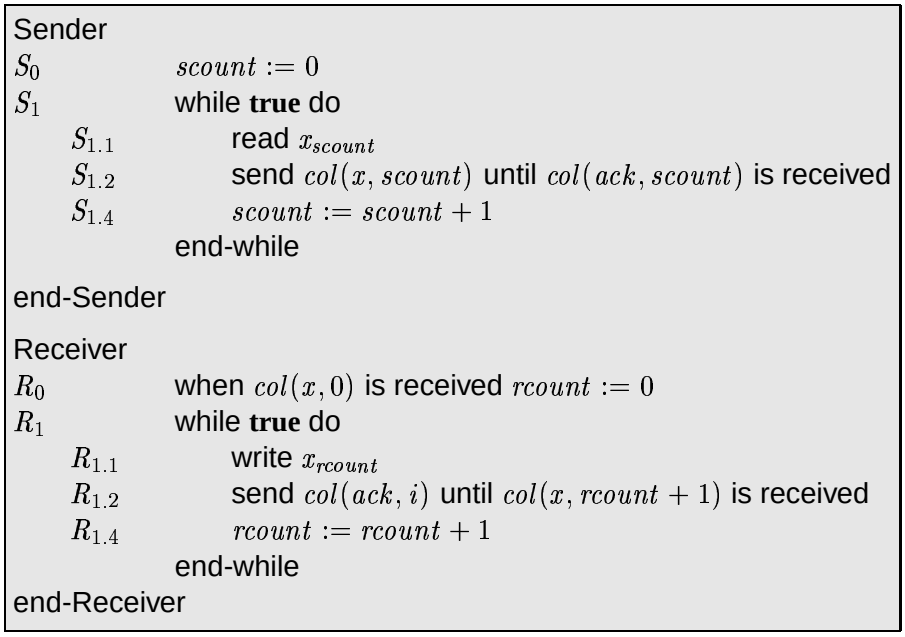

Figure 5: The alternating bit protocol.

no run in which one of the agents has just sent (or received) two messages of a pair in a particular state (such an approach is undertaken in [12], where the model is divided in "good" and "bad" states). Since here we are more interested in epistemic properties of the protocol, rather than proving correctness properties of the channels (e.g., *no two messages arrive at the same time*) we choose the second option, and define a model that incorporates the constraint on the pairs already.

In summary, the states in our model will be represented as four-tuples

$$q_i = \langle \mathtt{ssx}, \mathtt{sra} \mid \mathtt{rrx}, \mathtt{rsa} \rangle$$

where:

- $\mathtt{ssx}, \mathtt{sra}, \mathtt{rrx}, \mathtt{rsa} \in \{\mathtt{0}, \mathtt{1}\} = \mathbb{B}$; and
- q_i, where $1 \leq i \leq 16$, is the name of the state, which is just the decimal representation of the binary number the state corresponds to (so for instance, $q_5 = \langle \mathtt{0}, \mathtt{1} \mid \mathtt{0}, \mathtt{1} \rangle$).

We furthermore use variables $s, t, q, q_1, q_2 \ldots$ to range over states. The interpretation of the atoms are as follows:

- `ssx` being true means that the last bit that S sent was of type $x \cdot 1$;
- `sra` says that the last bit he has received is $a \cdot 1$;
- `rrx` means that R has recently received a bit of type $x \cdot 1$; and
- `rsa` indicates that he just sent $a \cdot 1$.

Obviously, given the constraint mentioned above, an atom like `sra` being false indicates that the last bit that S received is $a \cdot 0$. Thus, for instance, the state $q_9 = \langle \mathtt{1}, \mathtt{0} \mid \mathtt{0}, \mathtt{1} \rangle$ encodes the situation in which the last bit that S has sent is of type $x \cdot 1$, the last bit he received was $a \cdot 0$, the last bit R received was $x \cdot 0$, and he has recently sent $a \cdot 1$.

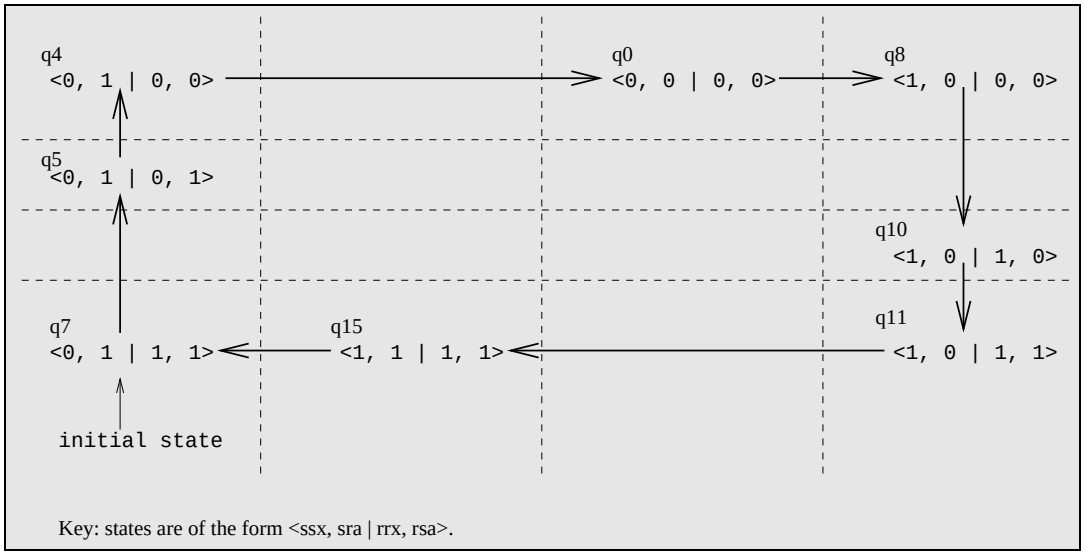

Figure 6: Runs of the alternating bit protocol: arrows indicate successor states. Note that there are states that are not reachable, such as $\langle 1, 0 \mid 0, 1\rangle$. The Sender cannot distinguish states that are placed on top of each other; similarly, for the Receiver, horizontally placed states are indistinguishable. Thus columns represent equivalence classes of S-indistinguishable states, while rows represent equivalence classes of R-indistinguishable states.

There is a natural way to model the epistemics in a system like that just described, since it is a special case of an *interpreted system* (cf. [7]), where the assumption rules that agents (exactly) know what their own state is, in other words, an agent a cannot distinguish the global state s from t (i.e., $s \sim_a t$) if his local part in those states, $s[a]$ and $t[a]$ are the same. In our example it would mean for S that two states $q_1 = \langle \mathtt{ssx}_1, \mathtt{sra}_1 \mid \mathtt{rrx}_1, \mathtt{rsa}_1\rangle$ and $q_2 = \langle \mathtt{ssx}_2, \mathtt{sra}_2 \mid \mathtt{rrx}_2, \mathtt{rsa}_2\rangle$ are epistemically indistinguishable for him if and only if $q_1[S] = \langle \mathtt{ssx}_1, \mathtt{sra}_1\rangle = \langle \mathtt{ssx}_2, \mathtt{sra}_2\rangle = q_2[S]$. Thus S knows what the last bits are that he has sent and received, but has no clue about the last bits that were sent or received by R. The same reasoning determines, mutatis mutandis, the epistemic accessibility for agent R.

In [2], Alur et al. identify several kinds of alternating systems, depending on whether the system is synchronous or not, and on how many agents "really" determine the next state. They also model a version of the alternating bit protocol (recall that in ATL no epistemics is involved) for a sender S and an environment e. Their design-choice leads them to a *turn-based asynchronous ATS* where there is a *scheduler* who, for every state, decides which other agent is to proceed, and the other agents do not influence the successor state if not selected. As the name suggests, this in in fact an asynchronous modelling of the communication, which we are not aiming for here. In our modelling, if one of S and R (or both) sends a message, it does not wait and then find out whether it has been delivered: instead, it keeps on sending the same message until it receives an acknowledgement.

The way we think about the model is probably closest to [2]'s *lock-step synchronous ATS*. In such an ATS, the state space is the Cartesian product of the local spaces of the agents. Another constraint that [2] imposes on lock-step synchronous ATSs is that every agent uniquely determines its next local state (not the next global state, of course), mirroring systems where agents just update their own variables. However, this assumption about determinacy of their own local state is not true in our example: the communicating agents cannot control the messages they receive.

Modelling the Protocol as an AETS

Perhaps a more fundamental departure from [2] however, is the way the epistemics influences the transition relation. We adopt the following assumption, for both agents $a = S, R$ (recall that $q \sim_a t$ iff $q[a] = t[a]$).

Constraint 1 *$q \sim_a t$ implies $\delta(q, a) = \delta(t, a)$*

Constraint 1 says that agent have the same options in indistinguishable states. That means that agents have some awareness of their options. If we aim at modelling rational agents, then we generally assume that they make rational decisions, which usually boils down to optimising some utility function. This would imply that a rational agent always makes the same decision when he is in the same (or, for that matter, indistinguishable) situation. This would entail a further constraint on runs, rather than on the transition function. However, given the fact that we are dealing with autonomous agents, this requirement is not that natural anymore: why shouldn't an agent be allowed to choose once for Q, and another time for Q'? Even if we would not have epistemic notions, we would allow the agent to make different choices when encountering *exactly the same state*.

There is a second epistemic property that we wish to impose on our model. It says that every choice for an agent is closed under epistemic alternatives. If an agent is unable to distinguish two states, then he should also not discriminate between them when making choices. We first state the constraint, and then explain it.

Constraint 2 *If $Q' \in \delta(q, a), q_1 \in Q'$ and $q_1 \sim_a q_2$, then $q_2 \in Q'$*

This constraint is very natural in an interpreted system, where $q_1 \sim_a q_2$ iff a's local states in q_1 and q_2 are the same. In such a system, we often want to establish that agents can *only* alter their own local state. Suppose we have two processes, agent 1 controlling variable x_1 and 2 controling x_2 (both variables ranging over the natural numbers). If δ has to model that 1 can only control his own variables, he could for instance have an option to make x_1 equal to 7, and an option to make it 13 ($\delta(q, 1) = \{ \{(x, y) \mid x = 7\}, \{(x, y) \mid x = 13\} \}$), but he should not be allowed to determine by himself that the next state be one in which x_1 is 7 and at the same time x_2 being at most 20 ($\{(x, y) \mid x = 7 \wedge x_2 \leq 20\}$): the latter constraint should be left to process 2.

In the following, let $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{a}', \mathtt{b}', \ldots$ be variables over the booleans. Moreover, let $\neg\mathtt{a}, \neg\mathtt{b}$ receive their natural interpretation. Remember that $\langle \mathtt{ssx}, \mathtt{sra} \mid \mathtt{rrx}, \mathtt{rsa} \rangle = \langle 1, 1 \mid 1, 1 \rangle$ iff the last bit sent by S is of the form $x \cdot 1$, the last acknowledgement received is $a \cdot 1$, the last bit received by R is $x \cdot 1$ and the last acknowledgement sent by R is $a \cdot 1$.

A next assumption is that sending a message takes some time: it is never the case that a message is sent and received in the same state.

Constraint 3 *Let q' be a successor of q (that means: there are Q_S, Q_R and Q_e such that $Q_S \in \delta(q, S)$, $Q_R \in \delta(q, R)$ and $Q_e \in \delta(q, e)$ with $Q_S \cap Q_R \cap Q_e = \{q'\}$. Then:*

- *if $q = \langle \mathtt{a}, \mathtt{b} \mid \mathtt{a}, \mathtt{d} \rangle$ and $q' = \langle \neg\mathtt{a}, \mathtt{b}' \mid \mathtt{c}', \mathtt{d}' \rangle$ then $\mathtt{c}' = \mathtt{a}$.*
- *if $q = \langle \mathtt{a}, \mathtt{b} \mid \mathtt{c}, \mathtt{b} \rangle$ and $q' = \langle \mathtt{a}', \mathtt{b}' \mid \mathtt{c}', \neg\mathtt{b} \rangle$ then $\mathtt{b}' = \mathtt{b}$.*

Definition 1 *We say that an agent a does not have control over the bit $\mathtt{b}$ if for all q, we have: if $Q_a \in \delta(q, a)$, then for any $q' \in Q_a$, if $q' = \langle \mathtt{a}, \mathtt{b} \mid \mathtt{c}, \mathtt{d} \rangle$, then also $\langle \mathtt{a}, \neg\mathtt{b} \mid \mathtt{c}, \mathtt{d} \rangle \in Q_a$.*

Constraint 4 *Given our terminology for states $\langle \mathtt{ssx}, \mathtt{sra} \mid \mathtt{rrx}, \mathtt{rsa} \rangle$, agent S has no control over $\mathtt{sra}, \mathtt{rrx}, \mathtt{rsa}$, agent R has no control over $\mathtt{ssx}, \mathtt{sra}, \mathtt{rrx}$ and the environment e has no control over $\mathtt{ssx}, \mathtt{rsa}$.*

$$
\begin{array}{lllll}
S1 & \delta(\langle 0,0 \mid \mathtt{c},\mathtt{d}\rangle, S) & = & \{\,\{\langle 1,\mathtt{b}' \mid \mathtt{c}',\mathtt{d}'\rangle | \mathtt{b}',\mathtt{c}',\mathtt{d}' \in \mathbb{B}\}\,\} \\
S2 & \delta(\langle 0,1 \mid \mathtt{c},\mathtt{d}\rangle, S) & = & \{\,\{\langle 0,\mathtt{b}' \mid \mathtt{c}',\mathtt{d}'\rangle | \mathtt{b}',\mathtt{c}',\mathtt{d}' \in \mathbb{B}\}\,\} \\
S3 & \delta(\langle 1,0 \mid \mathtt{c},\mathtt{d}\rangle, S) & = & \{\,\{\langle 1,\mathtt{b}' \mid \mathtt{c}',\mathtt{d}'\rangle | \mathtt{b}',\mathtt{c}',\mathtt{d}' \in \mathbb{B}\}\,\} \\
S4 & \delta(\langle 1,1 \mid \mathtt{c},\mathtt{d}\rangle, S) & = & \{\,\{\langle 0,\mathtt{b}' \mid \mathtt{c}',\mathtt{d}'\rangle | \mathtt{b}',\mathtt{c}',\mathtt{d}' \in \mathbb{B}\}\,\}
\end{array}
$$

Figure 7: The transition function for S

$$
\begin{array}{lllll}
R1 & \delta(\langle \mathtt{a},\mathtt{b} \mid 0,0\rangle, R) & = & \{\,\{\langle \mathtt{a}',\mathtt{b}' \mid \mathtt{c}',0\rangle | \mathtt{a}',\mathtt{b}',\mathtt{c}' \in \mathbb{B}\}\,\} \\
R2 & \delta(\langle \mathtt{a},\mathtt{b} \mid 0,1\rangle, R) & = & \{\,\{\langle \mathtt{a}',\mathtt{b}' \mid \mathtt{c}',0\rangle | \mathtt{a}',\mathtt{b}',\mathtt{c}' \in \mathbb{B}\}\,\} \\
R3 & \delta(\langle \mathtt{a},\mathtt{b} \mid 1,0\rangle, R) & = & \{\,\{\langle \mathtt{a}',\mathtt{b}' \mid \mathtt{c}',1\rangle | \mathtt{a}',\mathtt{b}',\mathtt{c}' \in \mathbb{B}\}\,\} \\
R4 & \delta(\langle \mathtt{a},\mathtt{b} \mid 1,1\rangle, R) & = & \{\,\{\langle \mathtt{a}',\mathtt{b}' \mid \mathtt{c}',1\rangle | \mathtt{a}',\mathtt{b}',\mathtt{c}' \in \mathbb{B}\}\,\}
\end{array}
$$

Figure 8: The transition function for R

The transition function of S establishes that:

($S1$) if S has recently sent $x \cdot 0$ and received its acknowledgement $a \cdot 0$, it moves on to a state where it is sending $x \cdot 1$, being unable to determine what kind of messages it will receive, and likewise unable to influence what happens to R's state;

($S2$) if S is in the process of sending $x \cdot 0$, but still receiving $a \cdot 1$, it continues to send $x \cdot 0$, being unable to determine what kind of messages it will receive, and likewise unable to influence what happens to R's state.

The motivation for the clauses $S3$ and $S4$ are similar, as is the transition function for R, which we give in Figure 8.

It is tempting to attempt to combine $S1$ and $S3$ into one clause

$$S1/3 \quad \delta(\langle \mathtt{a},0 \mid \mathtt{c},\mathtt{d}\rangle, S) = \{\{\langle 1,\mathtt{b}' \mid \mathtt{c}',\mathtt{d}'\rangle | \mathtt{b}',\mathtt{c}',\mathtt{d}' \in \mathbb{B}\}\}$$

However, there still should be a difference between the two. The transition in $S1$ can be described as "I have received acknowledgement $a \cdot 0$ now, so I proceed to send a *new* $x \cdot 1$ from the tape", where as the transition at $S3$ should be labelled "My message $x \cdot 1$ is not acknowledged yet, so I have to repeat to send this *old* version of $x \cdot 1$ once more". In other words, the transition at $S1$ has as a side effect that S advances on the tape X, whereas the transition at $S3$ keeps S at the same position of the tape.

We finally concentrate on the transition function of the environment e, as displayed in Figure 9. As a first approach to this transition function, consider the following clause:

$$
\begin{array}{llll}
e & \delta(\langle \mathtt{a},\mathtt{b} \mid \mathtt{c},\mathtt{d}\rangle, e) = & \{ & \{\langle \mathtt{a}',0 \mid 0,\mathtt{d}'\rangle | \mathtt{a}',\mathtt{d}' \in \mathbb{B}\}, \\
 & & & \{\langle \mathtt{a}',0 \mid 1,\mathtt{d}'\rangle | \mathtt{a}',\mathtt{d}' \in \mathbb{B}\}, \\
 & & & \{\langle \mathtt{a}',1 \mid 0,\mathtt{d}'\rangle | \mathtt{a}',\mathtt{d}' \in \mathbb{B}\}, \\
 & & & \{\langle \mathtt{a}',1 \mid 1,\mathtt{d}'\rangle | \mathtt{a}',\mathtt{d}' \in \mathbb{B}\} \quad \}
\end{array}
$$

This transition rule expresses the following. The environment leaves the control of the first state variable `ssx` to the other agents, as well as that of the variable `rsa`: it cannot control what either of the other agents has sent as a latest message. On the other hand, e completely controls whether messages arrive, so it is in charge of the bits `sra` and `rrx`. If, using this transition function, e would for instance choose the first alternative, $\{\langle \mathtt{a}',0 \mid 0,\mathtt{d}'\rangle | \mathtt{a}',\mathtt{d}' \in \mathbb{B}\}$, it would decide that $a \cdot 0$ and $x \cdot 0$ get delivered. However, this is in a sense too liberal: we do not, for instance, want the environment to

establish a transition from a state $\langle \mathtt{a}, 0 \mid \mathtt{c}, 0\rangle$ to any state $\langle \mathtt{a}', 1 \mid \mathtt{c}', 0\rangle$, since it would mean that in the first state R has recently sent $a \cdot 0$ and S has received it, and in the next state R is still sending $a \cdot 0$ but "spontaneously", S receives a different message $a \cdot 1$. It can of course be the case that an agent is receiving another message than that most recently sent to him, but once any agent has received the most recent message sent to him, he cannot from then on receive another message if the other agent does not change its message.

So, the condition that we put on the environment is that it may loose messages, but it will never alter a message. This property is formalised as follows.

Constraint 5 *Suppose state q is of type $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{a}, \mathtt{d}\rangle$. Then for any $Q \in \delta(q, e)$, any $q' \in Q$ is of the form $\langle \mathtt{a}', \mathtt{b}' \mid \mathtt{a}, \mathtt{d}'\rangle$. Similarly, if q is of type $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{c}, \mathtt{b}\rangle$, then any q' in Q should be of the form $\langle \mathtt{a}', \mathtt{b} \mid \mathtt{c}', \mathtt{d}'\rangle$.*

The transition function for e basically distinguishes three kinds of transitions. First of all, $e0$ and $e15$ are on states of type $\langle \mathtt{b}, \mathtt{b} \mid \mathtt{b}, \mathtt{b}\rangle$. It is obvious that Constraint 5 only allows transitions to states of type $\langle \mathtt{a}', \mathtt{b} \mid \mathtt{b}, \mathtt{d}'\rangle$, for which e is not in the power of determining $\mathtt{a}'$ or $\mathtt{d}'$. Secondly, there are transitions from states of type $\langle \mathtt{b}, \mathtt{c} \mid \mathtt{b}, \mathtt{c}'\rangle$ with $\mathtt{c} \neq \mathtt{c}'$ (clauses $e1$, $e4$, $e11$ and $e14$) or of type $\langle \mathtt{c}, \mathtt{b} \mid \mathtt{c}', \mathtt{b}\rangle$ ($e2$, $e7$, $e8$ and $e13$), with $\mathtt{c} \neq \mathtt{c}'$. Consider the first case, i.e., $q = \langle \mathtt{b}, \mathtt{c} \mid \mathtt{b}, \mathtt{c}'\rangle$ (the other case is analogous). This is a state where Receiver has received the right x, but Sender still waits for the right acknowledgement. Environment can now choose whether this acknowledgement will arrive: so it can choose between the states of type $\langle \mathtt{d}, \mathtt{c} \mid \mathtt{b}, \mathtt{d}''\rangle$ (acknowledgement a still not arrived) and those of type $\langle \mathtt{d}, \mathtt{c}' \mid \mathtt{b}, \mathtt{d}''\rangle$ (acknowledgement a arrived), which is equivalent to saying that the resulting states or either of type $\langle \mathtt{d}, 0 \mid \mathtt{b}, \mathtt{d}''\rangle$ or of type $\langle \mathtt{d}, 1 \mid \mathtt{b}, \mathtt{d}''\rangle$.

Finally, we have the cases $e3$, $e6$, $e9$ and $e12$. Here, the argument q is of type $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{c}, \mathtt{d}\rangle$, where $\mathtt{a} \neq c$ and $\mathtt{b} \neq d$. This means that both Sender and Receiver are waiting for the "right" acknowledgement, and the receiver can choose to have none, one, or both of them to be delivered. This explains the four sets of alternatives for these cases.

Theorem 1 *The transition function displayed in Figures 7, 8 and 9 together constitute a transition function for an ATS, in particular, we have for any state q, that $\delta(q, S) \cap \delta(q, R) \cap \delta(q, e)$ is a singleton.*

Besides the constraints we have put on to our model and on the transition function, we also have to make the requirements of fairness (by the environment) on the set of all computation runs.

Constraint 6 *Let $\gamma : Q \times \Sigma \to 2^{2^Q}$ be defined as follows:*

1. $\gamma(q, S) = \emptyset$
2. $\gamma(q, R) = \emptyset$
3. $\gamma(q, e)$ *is defined as follows:*
   (a) *for the states q of which the binary encoding is not 4, 7, 8 or 11, $\gamma(q, e) = \delta(q, e)$*
   (b) *for the remaining four states, γ is as defined in Figure 10.*

Let us explain the fairness constraint $\gamma 4$, the others are similar. When comparing $\gamma 4$ with $e4$, we see that $\gamma 4$ removes $\{\langle \mathtt{d}, 1 \mid 0, \mathtt{d}'\rangle | \mathtt{d}, \mathtt{d}' \in \mathbb{B}\}$ from $\delta(\langle 0, 1 \mid 0, 0\rangle, e)$: which is exactly the choice that would not take us any further in a computation step: given the state $\langle 0, 1 \mid 0, 0\rangle$, we know that S is waiting to receive $a \cdot 0$ from R, who, in his turn, awaits to receive $x \cdot 1$ from R. Until these messages are not received, S and R will keep on doing what they do in the given state: send $x \cdot 0$ and $a \cdot 0$, respectively. If the environment will not eventually deliver $a \cdot 0$, the process will not really make progress any more. We do not have to impose such a constraint on for instance $e1$, defined on $\langle 0, 0 \mid 0, 1\rangle$: here, the Sender has received an acknowledgement for his message $x \cdot 0$, and hence he will continue to send an $x \cdot 1$, leading us to a state of type $\langle 1, \mathtt{c} \mid \mathtt{c}', \mathtt{c}''\rangle$.

$$
\begin{array}{lllll}
e0 & \delta(\langle 0,0 \mid 0,0\rangle, e) & = & \{ & \{\langle \mathtt{d},0 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} \quad \} \\
e1 & \delta(\langle 0,0 \mid 0,1\rangle, e) & = & \{ & \{\langle \mathtt{d},0 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},1 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} \quad \} \\
e2 & \delta(\langle 0,0 \mid 1,0\rangle, e) & = & \{ & \{\langle \mathtt{d},0 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},0 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} \quad \} \\
e3 & \delta(\langle 0,0 \mid 1,1\rangle, e) & = & \{ & \{\langle \mathtt{d},0 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},0 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},1 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},1 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} \quad \} \\
 & & & & \\
e4 & \delta(\langle 0,1 \mid 0,0\rangle, e) & = & \{ & \{\langle \mathtt{d},0 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},1 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} \quad \} \\
e5 & \delta(\langle 0,1 \mid 0,1\rangle, e) & = & \{ & \{\langle \mathtt{d},1 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} \quad \} \\
e6 & \delta(\langle 0,1 \mid 1,0\rangle, e) & = & \{ & \{\langle \mathtt{d},0 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},0 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},1 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},1 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} \quad \} \\
e7 & \delta(\langle 0,1 \mid 1,1\rangle, e) & = & \{ & \{\langle \mathtt{d},1 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},1 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} \quad \} \\
 & & & & \\
e8 & \delta(\langle 1,0 \mid 0,0\rangle, e) & = & \{ & \{\langle \mathtt{d},0 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},1 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} \quad \} \\
e9 & \delta(\langle 1,0 \mid 0,1\rangle, e) & = & \{ & \{\langle \mathtt{d},0 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},0 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},1 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},1 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} \quad \} \\
e10 & \delta(\langle 1,0 \mid 1,0\rangle, e) & = & \{ & \{\langle \mathtt{d},0 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} \quad \} \\
e11 & \delta(\langle 1,0 \mid 1,1\rangle, e) & = & \{ & \{\langle \mathtt{d},0 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},1 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} \quad \} \\
 & & & & \\
e12 & \delta(\langle 1,1 \mid 0,0\rangle, e) & = & \{ & \{\langle \mathtt{d},0 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},0 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},1 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},1 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} \quad \} \\
e13 & \delta(\langle 1,1 \mid 0,1\rangle, e) & = & \{ & \{\langle \mathtt{d},1 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},1 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} \quad \} \\
e14 & \delta(\langle 1,1 \mid 1,0\rangle, e) & = & \{ & \{\langle \mathtt{d},0 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\}, \\
 & & & & \{\langle \mathtt{d},1 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} \quad \} \\
e15 & \delta(\langle 1,1 \mid 1,1\rangle, e) & = & \{ & \{\langle \mathtt{d},1 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{ds}' \in \mathbb{B}\} \quad \}
\end{array}
$$

Figure 9: The transition function for e

Modelling the Protocol with MOCHA

We now consider an implementation of the protocol and the automatic verification of its properties via model checking. We make use of the MOCHA model checking system for ATL [3, 1]. MOCHA takes as input an alternating transition system described using a (relatively) high level language called REACTIVEMODULES. MOCHA is then capable of either randomly simulating the execution of this system, or else of taking formulae of ATL and automatically checking their truth or falsity in the transition system.

$$
\begin{array}{llcll}
\gamma 4 & \delta(\langle 0,1 \mid 0,0\rangle,\gamma) & = & \{ & \{\langle \mathtt{d},0 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} & \} \\
\gamma 7 & \delta(\langle 0,1 \mid 1,1\rangle,\gamma) & = & \{ & \{\langle \mathtt{d},1 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} & \} \\
\gamma 8 & \delta(\langle 1,0 \mid 0,0\rangle,\gamma) & = & \{ & \{\langle \mathtt{d},1 \mid 0,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} & \} \\
\gamma 11 & \delta(\langle 1,0 \mid 1,1\rangle,\gamma) & = & \{ & \{\langle \mathtt{d},1 \mid 1,\mathtt{d}'\rangle | \mathtt{d},\mathtt{d}' \in \mathbb{B}\} & \}
\end{array}
$$

Figure 10: The transition function for γ

A key issue when model checking any system, which we alluded to above, is that of what to encode within the model, and what to check as its properties. In general, the model that is implemented and subsequently checked will represent an *abstraction* of the actual system at hand: the model designer must make reasoned choices about which features of the system are critical to the fidelity of the model, and which are merely irrelevant detail. Our first implementation, given in Figure 11, is a good illustration of this issue. In this model, we do not even attempt to model the actual communication of information between the sender and receiver; instead, we focus on the way in which the acknowledgement process works, via the "colouring" of messages with alternating binary digits, as described above. This leaves us with a simple and elegant model, which we argue captures the key features of the protocol.

Before proceeding, we give a brief description of the MOCHA constructs used in the model (see [1] for more details). The greater part of the model consists of three `module`s, which correspond to the agents `Sender`, `Receiver`, and `Environment`. Each module consist of an *interface* and a number of `atom`s, which define the behaviour of the corresponding agent. The interface part of a module defines two key classes of variables[2]:

- `interface` variables are those variables which the module "owns" and makes use of, but which may be observed by other modules;
- `external` variables are those which the module sees, but which are "owned" by another module (thus a variable that is declared as `external` in one module must be declared as an `interface` variable in another module).

Thus the variable `ssx` is "owned" by the sender, but other modules may observe the value of this variable if they declare it to be `external`. The `Environment` module in fact declares `ssx` to be `external`, and so observes the value of `ssx`.

Within each module are a number of `atom`s, which may be thought of as *update threads*. Atoms are rather like "micro modules" — in much the same way to regular modules, they have an interface part, and an action component, which defines their behaviour. The interface of an atom defines the variables it `reads` (i.e., the variables it observes and makes use of in deciding what to do), and those it `controls`. If a variable is controlled by an atom, then no other atom is permitted to make any changes to this variable[3]. The behaviour of an atom is defined by a number of *guarded commands*, which have the following syntactic form.

```
[] guard -> command
```

Here, `guard` is a predicate over the variables accessible to the atom (those it `reads` and `controls`), and `command` is a list of assignment statements. The idea is that if the guard part evaluates to true, then the corresponding command part is "enabled for execution". At any given time, a number of

[2] MOCHA also allows an additional class of `private` (local) variables, which are internal to a module, and which may not be seen by other modules. However, we make no use of these in our model.

[3] Again, there is actually a further class of variables that an atom may access: those it `awaits`. However, again, we make no use of these in our model, and so we will say no more about them.

```
module Sender is
   interface ssx :  bool
   external sra :  bool
   atom controls ssx reads ssx, sra
   init
        [] true -> ssx' := false
   update
        [] (ssx = false) & (sra = false) -> ssx' := true
        [] (ssx = false) & (sra = true) -> ssx' := false
        [] (ssx = true) & (sra = false) -> ssx' := true
        [] (ssx = true) & (sra = true) -> ssx' := false
   endatom
endmodule -- Sender

module Receiver is
   interface rsa :  bool
   external rrx :  bool
   atom controls rsa reads rrx
   init
        [] true -> rsa' := true
   update
        [] true -> rsa' := rrx
   endatom
endmodule -- Receiver

module Environment is
   interface rrx, sra :  bool
   external ssx, rsa :  bool
   atom controls rrx reads ssx
   init
        [] true -> rrx' := true
   update
        [] true -> rrx' := ssx
   endatom
   atom controls sra reads rsa
   init
        [] true -> sra' := true
   update
        [] true -> sra' := rsa
   endatom
endmodule -- Environment

System := (Sender || Receiver || Environment)
```

Figure 11: Modelling the alternating bit protocol for MOCHA

guards in an atom may be true, and so a number of commands within the atom may be enabled for execution. However, only one command within an atom may actually be selected for execution: the various alternatives represent the *choices* available to the agent.

Assignment statements have the following syntactic form.

```
var' := expression
```

On the right-hand side of the assignment statement, `expression` is an expression over the variables that the atom observes. On the left-hand side is a variable which must be controlled by the atom. The prime symbol on this variable name means "the value of the variable *after* this set of updates are carried out"; an un-primed variable means "the value of this variable *before* updating is carried out". Thus the prime symbol is used to distinguish between the value of variables before and after updating has been carried out.

There are two classes of guarded commands that may be declared in an atom: `init` and `update`. An `init` guarded command is only used in the first round of updating, and as the name suggests, these commands are thus used to initialise the values of variables. In our model, the one atom within the `Sender` module has a single `init` guarded command, as follows.

```
[] true -> ssx' := false
```

The guard part of this command is simply `true`, and so this guard is always satisfied. As this is the only guarded command in the `init` section of this atom, it will always be selected for execution at the start of execution. Thus variable `ssx` will always be assigned the value `false` when the system begins execution.

The `update` guarded commands in the `Sender` atom define how the `ssx` variable should be updated depending on the values of the `ssx` and `sra` variables: essentially, these update commands say that when the same bit that has been sent has been acknowledged, the next bit (with a different "colour") can be sent. It is instructive to compare the four guarded commands that define the behaviour of this atom with the transition function for the sender, as defined in Figure 7.

The behaviour of the `Receiver` module is rather similar to that of `Sender`; however, in the `Environment` module there are two atoms. These atoms are responsible for updating the `rrx` and `sra` variables, respectively. Thus, for example, the first atom simply passes on the value of the `ssx` variable directly to `rrx`.

Finally, following the definition of the `Environment` module is the definition of the `System` itself. The one-line declaration says that `System` is obtained from the parallel composition of the `Sender`, `Receiver`, and `Environment` modules. The properties that we establish through model checking are thus properties of `System`. We will write `System` $\models \varphi$ to mean that the property φ is true in all the states that may be generated by `System`, where φ is an ATEL formula over the propositional variables $\Pi = \{\texttt{ssx}, \texttt{sra}, \texttt{rrx}, \texttt{rsa}\}$, which have the obvious interpretation.

Notice that in our first version of the protocol, illustrated in Figure 11, we have designed the environment so that communication of the colouring bits and the corresponding acknowledgement messages is both *safe*, (in that bits are never transmitted incorrectly), and *immediate* (the environment always immediately passes on bits it has received: the communication delay between, for example, an acknowledgement being sent by the receiver and the acknowledgement being received by the sender is a single round). Of course, this does not correspond exactly with the definition of the AETS given earlier, because the AETS permitted the environment to *lose* messages, and hence to allow states to be *stuttered*. In fact, the system in Figure 11 implements the protocol with the *weak fairness* assumptions described earlier.

We can easily amend the MOCHA model to allow for stuttering, by simply making the two atoms that define the environment's behaviour to be *lazy*. This is achieved by simply prefixing the atom definitions with the keyword `lazy`. The effect is that in any given round, a `lazy` atom which has a number of commands enabled for execution need not actually choose any for execution: it can "skip" a round, leaving the values of the variables it `controls` unchanged. In particular, a `lazy` atom may stutter states infinitely often, never performing an update. In the case of the environment, this means that messages may never be delivered. We refer to the system containing the standard `Sender` and `Receiver` definitions, together with the `lazy` version of the `Environment`, as `Lazy`.

```
module Environment
   interface rrx, sra :  bool
   external ssx, rsa :  bool
   atom controls rrx reads rrx, ssx
   init
        [] true -> rrx' := true
   update
        [] true -> rrx' := ssx
        [] true -> rrx' := true
        [] true -> rrx' := false
   endatom
   atom controls sra reads sra, rsa
   init
        [] true -> sra' := true
   update
        [] true -> sra' := rsa
        [] true -> sra' := true
        [] true -> sra' := false
   endatom
endmodule -- Environment
```

Figure 12: A faulty environment.

Finally, the two variations of the system that we have considered thus far are both *safe*, in the terminology we introduced above, in the sense that if a message is delivered by the `Environment`, then it is delivered correctly. We define a third version of the protocol, which we refer to as `Faulty`, in which the environment has the choice of either delivering a message correctly, or else delivering it incorrectly. The definition of the `Environment` module for the `Faulty` system is given in Figure 12.

3.2 Verifying Properties of the Model with MOCHA

Before giving specific formulae φ that we establish of `System`, we will list some axiom schemas that correspond to properties of the *class* of systems we study. First, we have a property that corresponds to Constraint 1:

Proposition 1 *For all $a \in \{S, R\}$, and for all φ,* `System` $\models \langle\langle a \rangle\rangle \bigcirc \varphi \rightarrow K_a \langle\langle a \rangle\rangle \bigcirc \varphi$

Recall that $\langle\langle a \rangle\rangle \bigcirc \varphi$ means that a can achieve that φ is true in the next state — no matter what the other agents do. This just means that the agent a has some choice $Q' \in \delta(q, a)$ such that φ is true in every member of Q'. Constraint 1 guarantees that this Q' will be an option for a in every state similar to q, for a. Note that this does not mean that the agent will achieve the same in all accessible states: consider four states $q_1, \ldots, q_4$, with the atom x true in just q_1 and q_2, and y in q_1 and q_4. Suppose $\delta(q, a) = \{\{q_1, q_2\}, \{q_3, q_4\}\}$ and $q \sim_a s$, so that $\delta(s, a) = \delta(q, a)$. This transition function guarantees that $q \models \langle\langle a \rangle\rangle \bigcirc x$. Moreover suppose that for agent b $\delta(q, b) = \{\{q_2, q_3\}\}$ and $\delta(s, b) = \{\{q_1, q_4\}\}$. Now it holds that if a chooses the option $\{q_1, q_2\}$ in q, due to b's options, $x \wedge \neg y$ will hold in the next state, while if a makes the same choice in state s, $x \wedge y$ will hold in the next state. This is not in contradiction with Proposition 1, however: all that a can guarantee in q is x, not $x \wedge \neg y$.

Next, we have a property that corresponds to Constraint 2:

Proposition 2 *For all $a \in \{S, R\}$, and for all φ,* `System` $\models \langle\langle a \rangle\rangle \bigcirc \varphi \rightarrow \langle\langle a \rangle\rangle \bigcirc K_a \varphi$

Proposition 2 reflects our assumption that agents can only modify their own local state: if an agent a can achieve some state of affairs in the next state, it must be because he makes some decisions on his local variables, and these are known by agent a.

Proofs of these properties are very straightforward, and we leave them to the reader. We now move on to the specific properties we wish to model check. The first proposition mirrors Constraint 3:

Proposition 3

1. $\mathtt{System} \models (\mathtt{ssx} \wedge \mathtt{rrx}) \rightarrow [\![\Sigma]\!] \bigcirc (\neg \mathtt{ssx} \rightarrow \mathtt{rrx})$
2. $\mathtt{System} \models (\neg \mathtt{ssx} \wedge \neg \mathtt{rrx}) \rightarrow [\![\Sigma]\!] \bigcirc (\mathtt{ssx} \rightarrow \neg \mathtt{rrx})$
3. $\mathtt{System} \models (\mathtt{rsa} \wedge \mathtt{sra}) \rightarrow [\![\Sigma]\!] \bigcirc (\neg \mathtt{rsa} \rightarrow \mathtt{sra})$
4. $\mathtt{System} \models (\neg \mathtt{rsa} \wedge \neg \mathtt{sra}) \rightarrow [\![\Sigma]\!] \bigcirc (\mathtt{rsa} \rightarrow \neg \mathtt{sra})$

The first two of these formula, when translated into the textual form of ATL formulae required by MOCHA, are as follows.

```
atl "f01" <<>> G (ssx & rrx) =>
          [[Sender,Receiver,Environment]] X (~ssx => rrx) ;

atl "f02" <<>> G (~ssx & ~rrx) =>
          [[Sender,Receiver,Environment]] X (ssx => ~rrx) ;
```

Here, the `atl` label indicates to MOCHA that this is an ATL formula to check, while "`f01`" and "`f02`" are the names given to these formulae for reference purposes. Following the label, the formula is given: "`G`" is the MOCHA form of "$\Box$"; "`F`" is the MOCHA form of "$\Diamond$"; "`X`" is the MOCHA form of "$\bigcirc$", while "`U`" is the MOCHA form of "$\mathcal{U}$". Note that the prefix `<<>>G` to a formula indicates that the property should be checked in all states of the system.

Constraint 4 specifies which atoms cannot be controlled by which agents. The fact that agent a does not control atom p is easily translated into the ATEL formula $\neg \langle\!\langle a \rangle\!\rangle \Diamond \mathtt{p} \wedge \neg \langle\!\langle a \rangle\!\rangle \Diamond \neg \mathtt{p}$. Hence, the part for agent S of Constraint 4 reads as follows:

$$\neg \langle\!\langle S \rangle\!\rangle \Diamond \mathtt{sra} \wedge \neg \langle\!\langle S \rangle\!\rangle \Diamond \neg \mathtt{sra} \wedge \neg \langle\!\langle S \rangle\!\rangle \Diamond \mathtt{rrx} \wedge \neg \langle\!\langle S \rangle\!\rangle \Diamond \neg \mathtt{rrx} \wedge \neg \langle\!\langle S \rangle\!\rangle \Diamond \mathtt{rsa} \wedge \neg \langle\!\langle S \rangle\!\rangle \Diamond \neg \mathtt{rsa}$$

This is equivalent to the first item of the next proposition.

Proposition 4

1. $\mathtt{System} \models [\![S]\!] \Diamond \neg \mathtt{sra} \wedge [\![S]\!] \Diamond \mathtt{sra} \wedge [\![S]\!] \Diamond \neg \mathtt{rrx} \wedge [\![S]\!] \Diamond \mathtt{rrx} \wedge [\![S]\!] \Diamond \neg \mathtt{rsa} \wedge [\![S]\!] \Diamond \mathtt{rsa}$
2. $\mathtt{System} \models [\![R]\!] \Diamond \neg \mathtt{ssx} \wedge [\![R]\!] \Diamond \mathtt{ssx} \wedge [\![R]\!] \Diamond \neg \mathtt{sra} \wedge [\![R]\!] \Diamond \mathtt{sra} \wedge [\![R]\!] \Diamond \neg \mathtt{rrx} \wedge [\![R]\!] \Diamond \mathtt{rrx}$
3. $\mathtt{System} \models [\![e]\!] \Diamond \neg \mathtt{ssx} \wedge [\![e]\!] \Diamond \mathtt{ssx} \wedge [\![e]\!] \Diamond \neg \mathtt{rsa} \wedge [\![e]\!] \Diamond \mathtt{rsa}$

Again, these properties may easily be translated into the MOCHA form of ATL and model checked, as follows.

```
atl "f03" <<>> G [[Sender]] F ~sra ;

atl "f04" <<>> G [[Sender]] F sra ;
```

We now turn to Constraint 5, which states that communication is not instantaneous. This constraint corresponds to the following proposition.

Proposition 5

1. $\texttt{System} \models (\texttt{ssx} \wedge \texttt{rrx}) \rightarrow [\![\Sigma]\!] \bigcirc \texttt{rrx}$
2. $\texttt{System} \models (\neg\texttt{ssx} \wedge \neg\texttt{rrx}) \rightarrow [\![\Sigma]\!] \bigcirc \neg\texttt{rrx}$
3. $\texttt{System} \models (\texttt{sra} \wedge \texttt{rsa}) \rightarrow [\![\Sigma]\!] \bigcirc \texttt{sra}$
4. $\texttt{System} \models (\neg\texttt{sra} \wedge \neg\texttt{rsa}) \rightarrow [\![\Sigma]\!] \bigcirc \neg\texttt{sra}$

The corresponding MOCHA properties are as follows; all of these properties succeed when checked against `System`.

```
atl "f09" <<>> G (ssx & rrx) =>
          [[Sender,Receiver,Environment]] X rrx ;

atl "f10" <<>> G (~ssx & ~rrx) =>
          [[Sender,Receiver,Environment]] X ~rrx ;

atl "f11" <<>> G (sra & rsa) =>
          [[Sender,Receiver,Environment]] X ~sra ;

atl "f12" <<>> G (~sra & ~rsa) =>
          [[Sender,Receiver,Environment]] X ~sra ;
```

Verifying Knowledge Properties of the Model

Thus far, the properties we have checked of our system do not make full use of ATEL: in particular, none of the properties we have checked employ *knowledge modalities*. In this section, we will show how formulae containing such properties can be checked. The approach we adopt is based on the *model checking knowledge via local propositions* approach, developed by the present authors and first described in [10, 11]. This approach combines ideas from the *interpreted systems* semantics for knowledge [7] with concepts from the *logic of local propositions* developed by Engelhardt et al [6]. We will skip over the formal details, as these are rather involved, and give instead a high-level description of the technique.

The main component of ATEL missing from MOCHA is the accessibility relations used to give a semantics to knowledge. Where do these relations come from? Recall that we are making use of the *interpreted systems* approach of [7] to generate these relations. Given a state $q \in Q$ and agent $a \in \Sigma$, we write $state_a(q)$ to denote the *local* state of agent a when the system is in state q. The agent's local state includes all its local variables (in MOCHA terminology, the local variables of a `module` are all those that it declares to be `external`, `interface`, or `private`). We then define the accessibility relation $\sim_a$ as follows:

$$q \sim_a q' \quad \text{iff } state_a(q) = state_a(q'). \tag{1}$$

So, suppose we want to check whether, when the system is in some state q, agent a knows φ, i.e., whether $S, q \models K_a\varphi$. Then by (1), this amounts to showing that

$$\forall q' \in Q \text{ s.t. } state_a(q) = state_a(q') \text{ we have } S, q' \models \varphi. \tag{2}$$

We can represent such properties directly as formulae of ATL, which can be automatically checked using MOCHA. In order to do this, we need some additional notation. We denote by $\chi(a, q)$ a formula which is satisfied in exactly those states q' of the system where a's local state in q' is the same as a's local state in q, i.e., those states q' such that $state_a(q) = state_a(q')$. Note that we can "read off" the formula $\chi(a, q)$ directly from a's local state in q: it is the conjunction of a's local variables in q.

Then we can express (2) as the following ATL invariant formula:

$$\langle\langle\rangle\rangle \Box(\chi(a, q) \rightarrow \varphi) \tag{3}$$

We can directly encode this as a property to be checked using MOCHA:

```
<< >>G(chi(a,q) -> phi)
```

If this check succeeds, then we will have verified that $S, q \models K_a\varphi$. Checking the absence of knowledge can be done in a similar way. Suppose we want to check whether $S, q \models \neg K_a\varphi$. From the semantics of knowledge modalities, this amounts to showing that there is some state q' such that $q \sim_a q'$ and $S, q' \models \neg\varphi$. This can be done by checking the ATL formula

$$\langle\langle\Sigma\rangle\rangle \Diamond(\chi(a, q) \wedge \neg\varphi)$$

Note that there is actually a good deal more to this methodology for checking knowledge properties than we have suggested here; see [10] for details.

We begin our investigation of knowledge properties with some *static* properties.

Proposition 6

1. $\mathtt{System}, q15 \models K_S(\mathtt{rrx} \wedge \mathtt{rsa})$
   $\mathtt{Lazy}, q15 \models K_S(\mathtt{rrx} \wedge \mathtt{rsa})$
   $\mathtt{Faulty}, q15 \not\models K_S(\mathtt{rrx} \wedge \mathtt{rsa})$

2. $\mathtt{System}, q15 \models \neg K_R(\mathtt{ssx}) \wedge \neg K_R(\mathtt{sra})$
   $\mathtt{Lazy}, q15 \models \neg K_R(\mathtt{ssx}) \wedge \neg K_R(\mathtt{sra})$
   $\mathtt{Faulty}, q15 \models \neg K_R(\mathtt{ssx}) \wedge \neg K_R(\mathtt{sra})$

3. $\mathtt{System}, q0 \models K_S(\neg\mathtt{rsx} \wedge \neg\mathtt{rsa})$
   $\mathtt{Lazy}, q0 \models K_S(\neg\mathtt{rsx} \wedge \neg\mathtt{rsa})$
   $\mathtt{Faulty}, q0 \not\models K_S(\neg\mathtt{rsx} \wedge \neg\mathtt{rsa})$

4. $\mathtt{System}, q0 \models \neg K_R(\neg\mathtt{ssx}) \wedge \neg K_R(\neg\mathtt{sra})$
   $\mathtt{Lazy}, q0 \models \neg K_R(\neg\mathtt{ssx}) \wedge \neg K_R(\neg\mathtt{sra})$
   $\mathtt{Faulty}, q0 \models \neg K_R(\neg\mathtt{ssx}) \wedge \neg K_R(\neg\mathtt{sra})$

We can code these properties to check using MOCHA as follows.

```
atl "f50" <<>> G ((ssx & sra) => rrx) ;
atl "f51" <<>> G ((ssx & sra) => rsa) ;
atl "f52" <<Sender,Receiver,Environment>> F ((rrx & rsa) & ~ssx) ;
atl "f53" <<Sender,Receiver,Environment>> F ((rrx & rsa) & ~sra) ;
atl "f54" <<>> G ((~ssx & ~sra) => ~rrx) ;
atl "f55" <<>> G ((~ssx & ~sra) => ~rsa) ;
atl "f56" <<Sender,Receiver,Environment>> F ((rrx & rsa) & ~ssx) ;
atl "f57" <<Sender,Receiver,Environment>> F ((rrx & rsa) & ~sra) ;
```

Notice that the positive knowledge tests, which correspond to the safety requirement we mentioned earlier, fail in the Faulty system; however, they succeed in the Lazy system, indicating that this system, while it may cause states to be stuttered, is nevertheless sound in that bits are correctly transmitted.

Next, we prove some static properties of *nested* knowledge. To deal with nested knowledge, we generalise the formula $\chi(a, q)$ a bit. Let us, for any state q, with $\mathtt{q}$ denote the conjunction of the four literals that completely characterise q. For instance,

$$\mathtt{q}_7 = \neg\mathtt{ssx} \wedge \mathtt{sra} \wedge \mathtt{rrx} \wedge \mathtt{rsa}$$

We will now write $\chi(a, q)$ as a disjunction for all the states that a cannot distinguish, given q. So, for instance, $\chi(S, q_7)$ is written as $\mathtt{q}_7 \vee \mathtt{q}_5 \vee \mathtt{q}_4$. Now, a property like

$$\mathtt{System}, q \models K_a K_b \varphi$$

is verified in the following way.

First, note that this property is the same as $\mathtt{System} \models \mathtt{q} \rightarrow K_a K_b \varphi$. Let $\chi(a, q)$, or, for that matter, $\chi(a, \mathtt{q})$, be $\mathtt{s}_1 \vee \mathtt{s}_2 \vee \ldots \vee \mathtt{s}_n$. Then a first step in the translation yields

$$\mathtt{System} \models \langle\langle\rangle\rangle \Box(\mathtt{s}_1 \vee \mathtt{s}_2 \vee \ldots \vee \mathtt{s}_n) \rightarrow K_b \varphi \tag{4}$$

Verifying (4) is equivalent to verifying the following:

$$\begin{array}{l}
\mathtt{System} \models \langle\langle\rangle\rangle \Box \mathtt{s}_1 \rightarrow K_b \varphi \\
\mathtt{System} \models \langle\langle\rangle\rangle \Box \mathtt{s}_2 \rightarrow K_b \varphi \\
\quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\
\mathtt{System} \models \langle\langle\rangle\rangle \Box \mathtt{s}_n \rightarrow K_b \varphi
\end{array}$$

It will now be clear how this can be iteratively continued: let $\chi(b, \mathtt{s_i})$ be $(\mathtt{s}_{i_1} \vee \mathtt{s}_{i_2} \ldots \vee \mathtt{s}_{i_{k(i)}})$, and replace every check of type $\mathtt{System} \models \langle\langle\rangle\rangle \Box \mathtt{s_i} \rightarrow K_b \varphi$ by $k(i)$ checks $\mathtt{System} \models \langle\langle\rangle\rangle \Box \mathtt{s}_{i_1} \rightarrow \varphi$ ... $\mathtt{System} \models \langle\langle\rangle\rangle \Box \mathtt{s}_{i_{k(i)}} \rightarrow \varphi$.

Using this approach, we can establish the following.

Proposition 7

1. $\mathtt{System}, q15 \models K_S K_R(\mathtt{ssx} \vee \mathtt{sra})$
   $\mathtt{Lazy}, q15 \models K_S K_R(\mathtt{ssx} \vee \mathtt{sra})$
   $\mathtt{Faulty}, q15 \not\models K_S K_R(\mathtt{ssx} \vee \mathtt{sra})$

2. $\mathtt{System}, q15 \models K_S(\neg K_R(\mathtt{ssx})) \wedge K_S(\neg K_R(\mathtt{sra}))$
   $\mathtt{Lazy}, q15 \models K_S(\neg K_R(\mathtt{ssx})) \wedge K_S(\neg K_R(\mathtt{sra}))$
   $\mathtt{Faulty}, q15 \models K_S(\neg K_R(\mathtt{ssx})) \wedge K_S(\neg K_R(\mathtt{sra}))$

3. $\mathtt{System}, q0 \models K_S(\neg\mathtt{rsx} \wedge \neg\mathtt{rsa})$
   $\mathtt{Lazy}, q0 \models K_S(\neg\mathtt{rsx} \wedge \neg\mathtt{rsa})$
   $\mathtt{Faulty}, q0 \not\models K_S(\neg\mathtt{rsx} \wedge \neg\mathtt{rsa})$

4. $\mathtt{System}, q0 \models \neg K_R(\neg\mathtt{ssx}) \wedge \neg K_R(\neg\mathtt{sra})$
   $\mathtt{Lazy}, q0 \models \neg K_R(\neg\mathtt{ssx}) \wedge \neg K_R(\neg\mathtt{sra})$
   $\mathtt{Faulty}, q0 \models \neg K_R(\neg\mathtt{ssx}) \wedge \neg K_R(\neg\mathtt{sra})$

For the first property, we first observe that $\chi(S, \mathtt{q_{15}})$ is $\mathtt{q_{15}}$. So the first translation step yields the following.

$$\mathtt{System} \models \mathtt{q_{15}} \rightarrow K_R(\mathtt{ssx} \vee \mathtt{sra}) \tag{5}$$

Noting that $\chi(R, \mathtt{q_{15}}) = \mathtt{q_7} \vee \mathtt{q_{15}} \vee \mathtt{q_{11}}$, one more iteration then gives the following three checks.

$$\begin{array}{rcl} \mathtt{System} & \models & \mathtt{q_7} \rightarrow (\mathtt{ssx} \vee \mathtt{sra}) \\ \mathtt{System} & \models & \mathtt{q_{15}} \rightarrow (\mathtt{ssx} \vee \mathtt{sra}) \\ \mathtt{System} & \models & \mathtt{q_{11}} \rightarrow (\mathtt{ssx} \vee \mathtt{sra}) \end{array}$$

In the following, we prove some properties involving knowledge and cooperation. The first property expresses that although both S and R will at a certain point know $\mathtt{ssx}$, it is never the case that S knows that R knows $\mathtt{ssx}$. The second property shows when S can know the state of R: if he enters the first state in which S does not know $(\mathtt{ssx} \wedge \neg\mathtt{sra})$ anymore. Three is a bit complicated, and may be best understood with reference to Figure 6. The property can then be read as $\mathtt{ssx} \rightarrow \langle\langle\rangle\rangle(K_S\mathtt{ssx})\,\mathcal{U}\,\neg\mathtt{q10}$: if $\mathtt{ssx}$ is true, it keeps on being true until we enter q_7. And from q_7, state q_{10} is the only state that can not be reached by an $S-R-$ step. A more informal reading goes as follows: suppose $\mathtt{ssx}$ is the last bit sent by S. Then this will keep on being true (and, since $\mathtt{ssx}$ is local for S, it will be known by S), until a state is reached where S sends $\neg\mathtt{ssx}$ for the first time. Here, S knows that R knows that if $\mathtt{ssx}$ is true, it is either the previous $\mathtt{ssx}$ (in states q_7, q_{15} and q_{11}), which is both received and acknowledged, or a new version of $\mathtt{ssx}$, sent for the first time (q_8), which cannot be received and acknowledged yet. Finally, the last item says that if S's last bit that has been sent is $\mathtt{ssx}$ for which has not been received an acknowledgement, then S will not know whether R has received $\mathtt{ssx}$ (that is, S does not know whether $\mathtt{rrx}$), until S has received the acknowledgement for $\mathtt{sra}$.

Proposition 8 *We have:*

1. $\mathtt{System} \models \langle\langle\rangle\rangle \Diamond K_S\mathtt{ssx} \wedge \langle\langle\rangle\rangle \Diamond K_R\mathtt{ssx} \wedge \langle\langle\rangle\rangle \Box\neg K_S K_R\mathtt{ssx}$
2. $\mathtt{System} \models \mathtt{ssx} \wedge \neg\mathtt{sra} \rightarrow \langle\langle\rangle\rangle(K_S(\mathtt{ssx} \wedge \neg\mathtt{sra})\,\mathcal{U}\,K_S(\mathtt{rrx} \wedge \mathtt{rsa}))$
3. $\mathtt{System} \models \mathtt{ssx} \rightarrow \langle\langle\rangle\rangle(K_S\mathtt{ssx}\,\mathcal{U}\,K_S K_R(\mathtt{ssx} \rightarrow (\mathtt{rrx} \leftrightarrow \mathtt{rsa})))$
4. $\mathtt{System} \models \mathtt{ssx} \wedge \neg\mathtt{sra} \rightarrow \langle\langle\rangle\rangle((\neg K_S\mathtt{rrx} \wedge \neg K_S\neg\mathtt{rrx})\,\mathcal{U}\,\mathtt{sra})$

We will describe how the second and third of these properties are established; the remainder are straightforward using the techniques described above. So, consider the second property. First, note that since $\mathtt{ssx}$ and $\mathtt{sra}$ are local to S, we have the following equivalence.

$$K_S(\mathtt{ssx} \wedge \neg\mathtt{sra}) \leftrightarrow (\mathtt{ssx} \wedge \neg\mathtt{sra})$$

Second, recall that we established the following result earlier.

$$\mathtt{System}, q15 \models K_S(\mathtt{rrx} \wedge \mathtt{rsa})$$

Finally, we establish the following ATL property.

$$\mathtt{System} \models \mathtt{ssx} \wedge \neg\mathtt{sra} \rightarrow \langle\langle\rangle\rangle(\mathtt{ssx} \wedge \neg\mathtt{sra})\,\mathcal{U}\,(\mathtt{ssx} \wedge \mathtt{sra} \wedge \mathtt{rrx} \wedge \mathtt{rsa})$$

This is straightforwardly encoded for checking with MOCHA. Together, these are sufficient to establish the second property.

The third property may be established as follows. Observe that it is sufficient to prove the following two properties:

- $\texttt{System} \models \texttt{ssx} \rightarrow \langle\langle\rangle\rangle K_S \texttt{ssx} \mathcal{U}\, \texttt{q7}$
- $\texttt{System} \models \texttt{q7} \rightarrow K_S K_R(\texttt{ssx} \rightarrow (\texttt{rrx} \leftrightarrow \texttt{rsa}))$

Now, since ssx is local to the sender, we have the following equivalence.

$$\texttt{ssx} \leftrightarrow K_S \texttt{ssx}$$

This means that the first of these properties may be directly translated to an ATL formula. For the second property, one can use the approach described earlier for dealing with nested operators.

Note that none of these four properties holds for the Faulty or Lazy. This is essentially because, by using the empty coalition modality, we are quantifying over all possible computations of the system. The component that is essential to the correct functioning of the system turns out to be the environment: in the Faulty version, the environment can choose to corrupt the messages being sent. In both the Faulty and Lazy versions, the Environment can also choose to delay (indefinitely) the delivery of messages. However, the Environment can also choose to deliver messages correctly, with no delay. Thus, we obtain the following.

Proposition 9

1. $\texttt{Faulty} \models \langle\langle e\rangle\rangle \Diamond K_S \texttt{ssx} \wedge \langle\langle e\rangle\rangle \Diamond K_R \texttt{ssx} \wedge \langle\langle\rangle\rangle \Box \neg K_S K_R \texttt{ssx}$
2. $\texttt{Faulty} \models \texttt{ssx} \wedge \neg\texttt{sra} \rightarrow \langle\langle e\rangle\rangle (K_S(\texttt{ssx} \wedge \neg\texttt{sra})\, \mathcal{U}\, K_S(\texttt{rrx} \wedge \texttt{rsa}))$
3. $\texttt{Faulty} \models \texttt{ssx} \rightarrow \langle\langle e\rangle\rangle (K_S \texttt{ssx} \mathcal{U}\, K_S K_R(\texttt{ssx} \rightarrow (\texttt{rrx} \leftrightarrow \texttt{rsa})))$
4. $\texttt{Faulty} \models \texttt{ssx} \wedge \neg\texttt{sra} \rightarrow \langle\langle e\rangle\rangle ((\neg K_S \texttt{rrx} \wedge \neg K_S \neg \texttt{rrx})\, \mathcal{U}\, \texttt{sra})$

4 Conclusion

We have employed Alternating-time Temporal Epistemic Logic to reason about the alternating bit protocol. More specifically, we have built a transition system AETS for this protocol, specifying the behaviour of a Sender, a Receiver, and an environment. In fact we looked at three different behaviours of the environment: in System, the environment cannot but immediately transmit messages that have been sent. In Faulty, the environment may alter messages at will. Finally, in Lazy, the environment does not alter messages, but can wait arbitrary long to transmit them.

These different behaviours mirror a trade off between on the one hand precisely constraining the behaviour of the agents, and then proving that they cannot but act as specified in the protocol, and on the other hand, taking the paradigm of autonomy of agents serious, and then prove that the agents *can* behave in a way that is consistent with the protocol. We gave several examples in which System cannot but achieve a desired property, where for Faulty, we could only prove that the environment can cooperate in such a way that this property is achieved. We could have gone even further in leaving the behaviour of the agents open. When defining the transition functions for Sender for example, we could have loosened the constraint that he immediately sends a next fresh bit when receiving an acknowledgement for the old one: in such a system we would then have to prove that the environment and Sender have the cooperative power to ensure certain properties.

With respect to future work, an obvious avenue for further investigation is to generalise the techniques developed, and apply them to proving properties of games of incomplete information. We also want to further investigate natural constraints on the interaction between the temporal and epistemic aspects of ATEL.

Acknowledgements

We thank Giacomo Bonanno and two anonymous referees for their helpful comments. The framework on which this case study is based was presented at the fifth conference on Logic and the Foundations of Game and Decision Theory (LOFT5).

References

[1] R. Alur, L. de Alfaro, T. A. Henzinger, S. C. Krishnan, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Taşiran. MOCHA user manual. University of Berkeley Report, 2000.

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

[3] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Taşiran. Mocha: Modularity in model checking. In *CAV 1998: Tenth International Conference on Computer-aided Verification, (LNCS Volume 1427)*, pages 521–525. Springer-Verlag: Berlin, Germany, 1998.

[4] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press: Cambridge, MA, 2000.

[5] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science Volume B: Formal Models and Semantics*, pages 996–1072. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1990.

[6] K. Engelhardt, R. van der Meyden, and Y. Moses. Knowledge and the logic of local propositions. In *Proceedings of the 1998 Conference on Theoretical Aspects of Reasoning about Knowledge (TARK98)*, pages 29–41, Evanston, IL, July 1998.

[7] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. The MIT Press: Cambridge, MA, 1995.

[8] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. Knowledge-based programs. *Distributed Computing*, 10(4):199–225, 1997.

[9] J. Y. Halpern and L. D. Zuck. A little knowledge goes a long way: knowledge-based derivations and correctness proofs for a family of protocols. *Journal of the ACM*, 39(3):449–478, 1992.

[10] W. van der Hoek and M. Wooldridge. Model checking knowledge and time. In D. Bošnački and S. Leue, editors, *Model Checking Software, Proceedings of SPIN 2002 (LNCS Volume 2318)*, pages 95–111. Springer-Verlag: Berlin, Germany, 2002.

[11] W. van der Hoek and M. Wooldridge. Tractable multiagent planning for epistemic goals. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002)*, pages 1167–1174, Bologna, Italy, 2002.

[12] A. Lomuscio and M. Sergot. Deontic interpreted systems. To appear in *Studia Logica*, volume 75, 2003.

[13] J.-J. Ch. Meyer and W. van der Hoek. *Epistemic Logic for AI and Computer Science*. Cambridge University Press: Cambridge, England, 1995.

[14] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press: Cambridge, MA, 1994.

[15] Ariel Rubinstein. The electronic mail game: Strategic behavior under almost common knowledge. *American Economic Review*, 79(3):85–391, 1989.

[16] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2002.