Alternating-time Temporal Logic in the Calculus of (Co)Inductive Constructions

Dante Zanarini[1], Carlos Luna[2], and Luis Sierra[2]

[1] CIFASIS, Blvd. 27 de Febrero 210 bis, Rosario, Argentina,
`dante@fceia.unr.edu.ar`

[2] InCo, Facultad de Ingeniería, Universidad de la República, Uruguay,
{`cluna,sierra`}`@fing.edu.uy`

Abstract. This work presents a complete formalization of Alternating-time Temporal Logic (ATL) and its semantic model, Concurrent Game Structures (CGS), in the Calculus of (Co)Inductive Constructions, using the logical framework Coq. Unlike standard ATL semantics, temporal operators are formalized in terms of inductive and coinductive types, employing a fixpoint characterization of these operators. The formalization is used to model a concurrent system with an unbounded number of players and states, and to verify some properties expressed as ATL formulas. Unlike automatic techniques, our formal model has no restrictions in the size of the CGS, and arbitrary state predicates can be used as atomic propositions of ATL.

1 Introduction

Linear-time and branching-time temporal logics are natural specification languages for reactive systems [8, 16]. Alternating-time Temporal Logic (ATL), introduced by Alur, Henzinger and Kupferman [1, 2], is a temporal logic suitable for open systems specificiations, where an open system is a system that interacts with its environment and whose behavior depends on the state of the system as well as the behavior of the environment [2].

The logic ATL offers selective quantification over those paths that are possible outcomes of games. For instance, by preceding the temporal operator "eventually" with a selective path quantifier, it is possible to specify that in a game between a reactive system and the environment, the system has a strategy to reach a certain state.

An ATL formula is interpreted over Concurrent Game Structures (CGS) [2]. Every state transition of a CGS results from a simultaneous choice of moves, one for each player. The players represent individual components and the environment of an open system. CGS can capture various forms of synchronous composition for open systems.

In this work we formalize the CGS semantics of ATL in the Calculus of (Co)Inductive Constructions (CIC) [6, 15, 9], using the logical framework Coq [19, 4]. This formalization is divided in two parts: the logic ATL and the CGS semantics for a given game structure S. We show that the proof of the Coq proposition φ q guarantees that the CGS S satisfies the ATL formula φ in the state q of S (i.e. $q \models \varphi$). This work uses a general approach to deal with CGS where the number of states is unbounded; this generality is scarcely obtained using standard model checking techniques [3].

There exists previous work in formalizing temporal logic in systems other than Coq. We can mention the axiomatic encoding of Lamport's Temporal Logic of Actions in Isabelle [14]; and formalizations of Linear Temporal Logic (LTL) [16] in PVS [17] and HOL [18].

The choice of the CIC is dictated by its considerable expressive power as well as by the fact that it is supported by a tool of industrial strength, namely the Coq proof assistant. As one example of its applicability, Coq has been used for the development and formal verification of a compiler of a large subset of the C programming language [12]. Furthermore, there are works that formalize temporal logics in the CIC. We can mention the formalization of LTL [7] and Computation Tree Logic (CTL) [13]. LTL assumes implicit universal quantification over all paths that are generated by system moves. CTL [21] allows explicit existential and universal quantification over all paths. ATL introduces a more general variety of temporal logic; offers selective quantification over those paths that are possible outcomes of games. As compared to previous work by the authors [13], the present formalization of ATL is more general and complex.

A detailed description of the formalization is presented in Spanish in [22]. This document, along with the full formalization in Coq may be obtained from `http://www.fceia.unr.edu.ar/~dante/`.

The rest of the paper is organized as follows. In Section 2 we introduce CGS as well as the syntax and semantics of ATL. In Section 3 are formalized both the logic ATL and CGS including the notions of coalition and strategies. Unlike standard ATL semantics, temporal operators are formalized in terms of inductive and coinductive types, employing a fixpoint characterization of these operators. Then, Section 4 shows a complete list of axioms, theorems and inference rules for ATL according to [10] that have been proved in Coq with our proposal [22, 23]. In Section 5 we present the usual train example [2] as a simple (due to space restrictions) case study for the bounded and unbounded cases. Finally, Section 6 concludes with a summary of our contributions and directions for future work.

2 Alternating-time Temporal Logic

In this section we introduce CGS (Section 2.1) as well as the syntax and the semantics of ATL (Section 2.2) as found in [2].

2.1 Concurrent Game Structures

Definition 1 (CGS). *A CGS is a tuple $S = \langle \Sigma, Q, \Pi, \pi, d, \delta \rangle$ with:*

- *A set $\Sigma = \{1, \ldots, k\}$ of players or agents.*
- *A set Q of states.*

- *A finite set Π of atomic propositions.*
- *For each $q \in Q$, a set $\pi(q) \subseteq \Pi$ of propositions true at q.*
- *For each player $a \in \Sigma$ and each state $q \in Q$, a natural number $d_a(q) \geq 1$ of moves available at state q to player a. We identify the moves of a at state q with the numbers $1, \ldots, d_a(q)$. For $q \in Q$, a* move vector *at q is a tuple $\langle j_1, \ldots, j_k \rangle$ such that $1 \leq j_a \leq d_a(q)$ for each player a. We define $D(q)$ as the set of move vectors available at q; function D is called the move function.*
- *For each state $q \in Q$ and each move vector $\langle j_1, \ldots, j_k \rangle \in D(q)$, a state $\delta(q, j_1, \ldots, j_k) \in Q$ that results from state q if each player $a \in \Sigma$ chooses move j_a. The function δ is called transition function.*

For two states q and q', we say that q' is a *successor* of q if there exists a move vector $\langle j_1, \ldots, j_k \rangle$ such that $q' = \delta(q, j_1, \ldots, j_k)$. A *computation* of S is an infinite sequence $\omega = q_0, q_1, q_2, \ldots$ of states such that for all $i \geq 0$, the state q_{i+1} is a successor of q_i. We refer to a computation starting at state q as a *q-computation.* For a computation ω and a position $i \geq 0$, we use $\omega[i]$ and $\omega[0, i]$ to denote the i-th state and the finite prefix $q_0, \ldots, q_i$, respectively.

2.2 ATL Syntax and Semantics

Definition 2 (ATL). *Let Π be a set of atomic propositions, and Σ a set of k players. The set of ATL formulas is inductively defined as follows:*

- *p, for each $p \in \Pi$.*
- *$\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \rightarrow \psi$, where φ, ψ are ATL formulas.*
- *$\langle\langle A \rangle\rangle \bigcirc \varphi$, $\langle\langle A \rangle\rangle \Box \varphi$, $\langle\langle A \rangle\rangle \varphi \,\mathcal{U}\, \psi$, where φ, ψ are ATL formulas and $A \subseteq \Sigma$.*

The operator $\langle\langle\ \rangle\rangle$ is a path quantifier*; $\bigcirc$ (next), $\Box$ (box) and $\mathcal{U}$ (until) are* temporal operators*.*

ATL can be viewed as a generalization of the branching-time temporal logic CTL where path quantifiers can be parametrized by sets of players. In particular, we obtain a CTL-equivalent logic restricting A to $\emptyset$ or Σ in Def. 2.

Formulas in ATL are interpreted over states of a CGS with the same players and atomic propositions. The concept of *strategy* is introduced in [2] to formalize the semantics.

Definition 3 (Strategy). *Let $S = \langle \Sigma, Q, \Pi, \pi, d, \delta \rangle$ be a CGS and $a \in \Sigma$. A strategy for a is a function $f_a : Q^+ \rightarrow \mathbb{N}$ that maps every nonempty finite state sequence $\alpha \in Q^+$ to a natural number such that if q is the last state of α, then $1 \leq f_a(\alpha) \leq d_a(q)$.*

Given a state $q \in Q$, and $A \subseteq \Sigma$, an A-strategy $F_A = \{f_a \mid a \in A\}$ is a set of strategies, one for each player in A. The *outcomes* of F_A from a state q is the set of traces that players in A can enforce when they follow the strategies in F_A. A computation $\omega = q_0, q_1, \ldots$ belongs to $out(q, F_A)$ if $q_0 = q$ and for all positions i, there is a move vector $\langle j_1, \ldots, j_k \rangle$ such that (1) if $a \in A$, $j_a = f_a(\omega[0, i])$, and (2) $\delta(q_i, j_1, \ldots, j_k) = q_{i+1}$.

Definition 4 (Standard ATL Semantics). *Let S be a CGS and q a state of S. We write $q \models \varphi$ to indicate that the ATL formula φ holds at q. The relation $\models$ is defined inductively as follows:*

- $q \models p$ *, for atomic propositions* $p \in \Pi$ *iff* $p \in \pi(q)$.
- $q \models \neg\varphi$ *iff* $q \not\models \varphi$.
- $q \models \varphi_1 \vee \varphi_2$ *iff* $q \models \varphi_1$ *or* $q \models \varphi_2$.
- $q \models \varphi_1 \wedge \varphi_2$ *iff* $q \models \varphi_1$ *and* $q \models \varphi_2$.
- $q \models \varphi_1 \Rightarrow \varphi_2$ *iff* $q \models \varphi_2$ *given that* $q \models \varphi_1$.
- $q \models \langle\!\langle A \rangle\!\rangle \bigcirc \varphi$ *iff there exists an A-strategy* $F_A = \{f_a \mid a \in A\}$*, such that for all* $\omega \in out(q, F_A)$*, we have* $\omega[1] \models \varphi$.
- $q \models \langle\!\langle A \rangle\!\rangle \Box \varphi$ *iff there exists an A-strategy* $F_A = \{f_a \mid a \in A\}$ *such that for all* $\omega \in out(q, F_A)$ *and all positions* $i \geq 0$ *we have* $\omega[i] \models \varphi$.
- $q \models \langle\!\langle A \rangle\!\rangle \varphi_1 \,\mathcal{U}\, \varphi_2$ *iff there exists an A-strategy* $F_A = \{f_a \mid a \in A\}$*, such that for all* $\omega \in out(q, F_A)$ *there exists a position* $i \geq 0$ *such that* $\omega[i] \models \varphi_2$ *and for all positions* $0 \leq j < i$ *we have* $\omega[j] \models \varphi_1$.

3 Formalizing CGS and ATL

Our formalization is divided in two main parts. Section 3.2 provides a way to represent CGS, coalitions and strategies. In Section 3.3 we proceed to formalize the logic ATL. The formalization of temporal operators follows the axiomatization presented in [10], using fixpoints characterizations for $\langle\!\langle A \rangle\!\rangle \Box \varphi$ and $\langle\!\langle A \rangle\!\rangle \varphi \,\mathcal{U}\, \psi$.

We believe that giving semantics to temporal operators using fixpoint definitions by means of inductive and coinductive types has some advantages over the standard semantics from def. 4. The inductive and coinductive principles associated to our definition of temporal operators can be used to construct more elegant and concise proofs for ATL theorems (sect. 4) and for specific propierties of reactive systems (sect. 5).

3.1 The CIC and Coq

The CIC is a type theory, in brief, a higher order logic in which the individuals are classified into a hierarchy of types. The types work very much as in strongly typed functional programming languages which means that there are basic elementary types, types defined by induction, like sequences and trees, and function types. An inductive type is defined by its constructors and its elements are obtained as finite combinations of these constructors. Data types are called "Sets" in the CIC (in Coq). When the requirement of finiteness is removed we obtain the possibility of defining infinite structures, called coinductive types, like infinite sequences. On top of this, a higher-order logic is available which serves to predicate on the various data types. The interpretation of the propositions is constructive, i.e. a proposition is defined by specifying what a proof of it is and a proposition is true if and only if a proof of it has been constructed. The type of propositions is called `Prop`. We use the usual notation for logical connectives and quantifiers ($\rightarrow$,

$\vee$, $\wedge$, $\neg$, $\forall$, $\exists$). For anonymous functions and predicates, we utilize a notation similar to the Coq specification language. For instance, predicate $pos : \mathbb{N} \to Prop$ is written as $(\lambda\, n : \mathbb{N} \Rightarrow n > 0)$.

We define a (co)inductive predicate I by giving introduction rules of the form:

$$\frac{P_1 \dots P_m}{I\ x_1 \dots x_n}\ (\text{intro}_\text{i})$$

where free ocurrences of variables are implicitly universally quantified.

In this work we use some inductive types defined in the Coq Standard Library [20]. We employ notation $\{\ \}$ for the empty type, $\{1\}$ for unit type, $A + B$ for disjoint union (sum type). Type (seq A) denotes the set of finite sequences of type A. Empty sequence is noted as $\langle\rangle$, and the infix notation $s \frown e$ is used to denote the sequence resulting by appending element e to sequence s. The *Stream* type is used to represent infinite sequences of objects from a fixed type A. Constructor *Cons* adds an element $e : A$ to an infinite sequence ω. Infix notation $e \triangleleft \omega$ is used for $(Cons\ e\ \omega)$. We refer to [19, 4] for further details on the CIC and Coq.

3.2 Formalizing CGS

We assume three basic types in sort *Set*: *State*, the set of states; *Player*, the players in the system; and *Move*, the set of moves (or *actions*). These types are specification parameters, and must be instantiated when specifyng a concrete CGS. Observe that we do not imposse any finiteness requirement to these types.

Move Vectors and Transitions. A move vector is a function that assigns a move to each player, $\langle Move\rangle \overset{\text{def}}{=} Player \to Move$. The transition function is introduced as a relation $\delta : State \to \langle Move\rangle \to State \to Prop$. We say that the move m is enabled at state q for player a if there exists a move vector mv and a state q' such that mv assigns m to player a and q' is the successor of q when players in Σ chooses the movements in mv. Formally, the relation $enabled : State \to Player \to Move \to Prop$ has one constructor:

$$\frac{mv : \langle Move\rangle \qquad q' : State \qquad mv\ a = m \qquad \delta\ q\ mv\ q'}{enabled\ q\ a\ m}\ (enabled\_intro) \qquad (1)$$

A proof of type $(enabled\ q\ a\ m)$ is interpreted as *"player a can choose move m at state q"*. Two expected properties are assumed over δ; the property δ_f guarantees that the relation is indeed a function, while the property δ_d guarantees that for every state q, if you choose a move vector mv such that $(mv\ a)$ is enabled at q for every player a, then you will found an outgoing transition from q labeled with mv.

$$\begin{aligned}
&\delta\_f : \forall (q, q', q'' : State)(mv : \langle Move\rangle), \delta\ q\ mv\ q' \to \delta\ q\ mv\ q'' \to q' = q'' \\
&\delta\_d : \forall (q : State)(mv : \langle Move\rangle), \\
&\qquad (\forall a : Player, enabled\ q\ a\ (mv\ a)) \to \exists (q' : State), \delta\ q\ mv\ q'
\end{aligned} \qquad (2)$$

Coalitions. A coalition is a set of players $A \subseteq \Sigma$. The Coq Standard Library [20] defines a set over a universe U as an inhabitant of type $U \to Prop$. We say that element x belongs to set X if we can exhibit a proof of proposition $(X\ x)$. In particular, the union of sets X, Y is defined as $Union\ X\ Y \stackrel{\text{def}}{=} (\lambda\, x : U \Rightarrow X\ x \vee Y\ x)$. However, this formalization of sets is not satisfactory for our purposes due to its lack of computational content. This computational content is required, for instance, to prove the valid formula $\langle\langle A\rangle\rangle \bigcirc \varphi \to \langle\langle B\rangle\rangle \bigcirc \psi \to \langle\langle A \cup B\rangle\rangle \bigcirc (\varphi \wedge \psi)$, when A and B are disjoint sets. The proof "joins" the strategies for A and B given in the premises to construct a new strategy for the coalition $A \cup B$. For a player $a \in A \cup B$, the new strategy chooses the strategy given by the first premise when $a \in A$, and the strategy given by the second premise when $a \in B$.

As we will introduce strategies as an object with computational content, i.e. an inhabitant of sort *Set*, the election of a strategy cannot be made eliminating an inhabitant in *Prop* [19]. We conclude that proofs of set membership must live in sort *Set*. Therefore, we define a coalition as a term of type *Player* $\to$ *Set*. We say that player a *belongs* to coalition C if we can construct an element in type $(C\ a)$. Coalitions Σ and $\emptyset$, and the union of two coalitions are defined as:

$$\Sigma \stackrel{\text{def}}{=} \lambda\, a \Rightarrow \{1\} \qquad \emptyset \stackrel{\text{def}}{=} \lambda\, a \Rightarrow \{\ \} \qquad A \uplus B \stackrel{\text{def}}{=} \lambda\, a \Rightarrow A\ a + B\ a \tag{3}$$

Other operators, like coalition complement, can be defined easily. We refer the interested reader to [23].

Strategies. A strategy decides the next move taking into account the complete history of the game:

$$Strategy \stackrel{\text{def}}{=} \text{seq}\ State \to State \to Move \tag{4}$$

where the first argument is the past sequence of states, and the second the current state of the game. Let A be a coalition. A *strategy for coalition A* is a term of type $(StrategySet\ A)$, where:

$$StrategySet(A : Coalition) \stackrel{\text{def}}{=} \forall a : Player, A\ a \to Strategy \tag{5}$$

A term $F_A : (StrategySet\ A)$ gives a strategy for each player a, provided that $a \in A$. We define the notion of F_A-successor state for a coalition strategy F_A. Let q be the current state, and qs the game history. We say that q' is an F_A-successor of $qs \frown q$ if there exists a move vector mv such that: (1) a transition from q to q' labelled with mv exists; and (2) strategy $f_a \in F_A$ for player $a \in A$ is such that $f_a(qs \frown q) = mv(a)$. Formally, relation suc is introduced by means of the following definition:

$$suc : \forall A : Coalition, StrategySet\ A \to \text{seq}\ State \to State \to State \to Prop$$

$$\frac{\begin{array}{c} mv : \langle Move\rangle \qquad \delta\ q\ mv\ q' \\ \forall (a : Player)(H : A\ a), F_A\ a\ H\ qs\ q = mv\ a \end{array}}{suc\ A\ F_A\ qs\ q\ q'}\ (suc\_intro) \tag{6}$$

In the sequel, we will omit the first argument, since it can be inferred from the second. Also, we write $q' \in \mathit{suc}(qs, q, F_A)$ for a proof of $(\mathit{suc}\ F_A\ qs\ q\ q')$.

Now, we define coinductively the set of traces that a coalition A can enforce by following the strategy F_A. The relation *isOut* determines if the trace $(q \triangleleft q' \triangleleft \omega)$ is a possible result of the game when players in A follows strategies in F_A and game history is qs:

$$\mathit{isOut} : \forall A : \mathit{Coalition}, \mathit{StrategySet}\ A \to \mathrm{seq}\ \mathit{State} \to \mathit{Trace} \to \mathit{Prop}$$

$$\frac{q' \in \mathit{suc}(qs, q, F_A) \qquad \mathit{isOut}\ A\ F_A\ (qs \frown q)\ (q' \triangleleft \omega)}{\mathit{isOut}\ A\ F_A\ qs\ (q \triangleleft q' \triangleleft \omega)}\ (\mathit{isOut\_intro}) \tag{7}$$

where $\mathit{Trace} \stackrel{\text{def}}{=} (\mathit{Stream\ State})$. The set $\mathit{out}(q, F_A)$ of traces a coalition A can enforce if follows strategies in F_A is defined as:

$$\omega \in \mathit{out}(q, F_A) \stackrel{\text{def}}{=} \mathit{isOut}\ A\ F_A\ \langle\rangle\ (q \triangleleft \omega) \tag{8}$$

3.3 Formalizing ATL

In this section we present a formalization of the syntax and semantics of ATL. Let S be a CGS, an ATL state formula is a term of type $\mathit{StateForm} \stackrel{\text{def}}{=} \mathit{State} \to \mathit{Prop}$. If $q : \mathit{State}$ and $\varphi : \mathit{StateForm}$, a proof (term) of $(\varphi\ q)$ is interpreted as $q \models \varphi$.

Constants and Boolean Connectives. The $\top$ and $\bot$ formulas are easily defined as $\top \stackrel{\text{def}}{=} (\lambda\, q : \mathit{State} \Rightarrow \mathit{True})$, and $\bot \stackrel{\text{def}}{=} (\lambda\, q : \mathit{State} \Rightarrow \mathit{False})$. We use a standard point-free use of boolean connectives. For example, for state formulas φ, ψ, disjunction is defined as $\varphi \vee \psi \stackrel{\text{def}}{=} (\lambda\, q : \mathit{State} \Rightarrow \varphi\ q \vee \psi\ q)$.

Temporal Operators. The standard ATL semantics presented in Def. 4 for $\langle\!\langle A \rangle\!\rangle \bigcirc \varphi$ uses the notion of execution traces. We present here an alternative (and equivalent) semantics using only the notion of successor state. Let q be the current state of a game. To guarantee that the property φ holds in the next state a coalition A should follow a strategy F_A such that for every possible F_A-successor state q' we have $q' \models \varphi$.

Definition 5 (Next). *Let A : Coalition, q : State and φ : StateForm. The relation Next : Coalition $\to$ StateForm $\to$ StateForm is defined with one constructor as follows:*

$$\frac{F : \mathit{StrategySet}\ A \qquad \forall q', q' \in \mathit{suc}(\langle\rangle, q, F) \to \varphi\ q'}{\mathit{Next}\ A\ \varphi\ q}\ (\text{next}) \tag{9}$$

The ATL axiomatization found in [10] establishes that $\langle\!\langle A \rangle\!\rangle \Box \varphi$ is the greatest fixed point of equation $X \leftrightarrow \varphi \wedge \langle\!\langle A \rangle\!\rangle \bigcirc X$. Following this approach, we introduce a coinductive predicate to model this semantics for formulas of the form $\langle\!\langle A \rangle\!\rangle \Box \varphi$.

Definition 6 (Box). *Let A : Coalition, φ : StateForm and q : State. The coinductive predicate Box : Coalition → StateForm → StateForm is defined as:*

$$\frac{\begin{array}{c}\varphi\ q \qquad F : \mathit{StrategySet}\ A \\ \forall q', q' \in \mathit{suc}(\langle\rangle, q, F) \to \mathit{Box}\ A\ \varphi\ q'\end{array}}{\mathit{Box}\ A\ \varphi\ q}\ (\text{box}) \tag{10}$$

To construct a proof of $q \models \langle\!\langle A \rangle\!\rangle \Box\varphi$ two conditions must hold: (1) φ must be valid at state q; and (2) we need to find an A-strategy F such that, for all F-successor state q' of q we have $q' \models \langle\!\langle A \rangle\!\rangle \Box\varphi$.

Using the fact that $\langle\!\langle A \rangle\!\rangle \varphi \mathcal{U} \psi$ is the least fixed point of $X \leftrightarrow \psi \vee (\varphi \wedge \langle\!\langle A \rangle\!\rangle \bigcirc X)$ we introduce the semantics of $\langle\!\langle A \rangle\!\rangle \varphi \mathcal{U} \psi$ by an inductive relation.

Definition 7 (Until). *Let A : Coalition, φ, ψ : StateForm and q : State. The inductive relation Until : Coalition → StateForm → StateForm → StateForm is defined with two constructors as follows:*

$$\frac{\psi\ q}{\mathit{Until}\ A\ \varphi\ \psi\ q}\ (\mathcal{U}_1) \qquad \frac{\begin{array}{c}F : \mathit{StrategySet}\ A \qquad \varphi\ q \\ \forall q', q' \in \mathit{suc}(\langle\rangle, q, F) \to \mathit{Until}\ A\ \varphi\ \psi\ q'\end{array}}{\mathit{Until}\ A\ \varphi\ \psi\ q}\ (\mathcal{U}_2) \tag{11}$$

If $q \models \psi$, then $q \models \langle\!\langle A \rangle\!\rangle \varphi \mathcal{U} \psi$ (constructor $\mathcal{U}_1$). To prove $q \models \langle\!\langle A \rangle\!\rangle \varphi \mathcal{U} \psi$ using constructor $\mathcal{U}_2$, we need to prove that $q \models \varphi$ and there exists an A-strategy F such that, if players in A follow this strategy, in all F_A-successor state q' of q we have $q' \models \langle\!\langle A \rangle\!\rangle \varphi \mathcal{U} \psi$.

Derived operators like $\langle\!\langle A \rangle\!\rangle \Diamond\varphi$ (eventually), and $\langle\!\langle A \rangle\!\rangle \overset{\infty}{F} \varphi$ (infinitely often) have been defined. For example, $\langle\!\langle A \rangle\!\rangle \overset{\infty}{F} \varphi \overset{\text{def}}{=} \langle\!\langle A \rangle\!\rangle \Box \langle\!\langle \emptyset \rangle\!\rangle \Diamond\varphi$. For details see [23].

4 A Deductive System for ATL

The formalization presented in Section 3 can be used to reason about properties of ATL and CGS. To prove ATL theorems we often use general properties involving coalitions and strategies.

A complete set of axioms and inference rules for ATL is presented in [10]. We have proved all these results in our formalization. Due to space constraints, proofs are merely outlined; however, all proofs have been formalized in Coq and are available as part of the full specification [23].

Theorem 1. *The following formulas are valid in all states of all CGS:*

($\bot$) $\neg\langle\!\langle A \rangle\!\rangle \bigcirc \bot$.
($\top$) $\langle\!\langle A \rangle\!\rangle \bigcirc \top$.
(Σ) $\neg\langle\!\langle \varnothing \rangle\!\rangle \bigcirc \neg\varphi \to \langle\!\langle \Sigma \rangle\!\rangle \bigcirc \varphi$.
(S) $\langle\!\langle A_1 \rangle\!\rangle \bigcirc \varphi_1 \wedge \langle\!\langle A_2 \rangle\!\rangle \bigcirc \varphi_2 \to \langle\!\langle A_1 \cup A_2 \rangle\!\rangle \bigcirc (\varphi_1 \wedge \varphi_2)$, if $A_1 \cap A_2 = \varnothing$.
(FP$_\Box$) $\langle\!\langle A \rangle\!\rangle \Box\varphi \leftrightarrow \varphi \wedge \langle\!\langle A \rangle\!\rangle \bigcirc \langle\!\langle A \rangle\!\rangle \Box\varphi$.
(GFP$_\Box$) $\langle\!\langle \varnothing \rangle\!\rangle \Box (\theta \to (\varphi \wedge \langle\!\langle A \rangle\!\rangle \bigcirc \theta)) \to \langle\!\langle \varnothing \rangle\!\rangle \Box (\theta \to \langle\!\langle A \rangle\!\rangle \Box\varphi)$.

($\mathbf{FP}_{\mathcal{U}}$) $\langle\!\langle A\rangle\!\rangle\varphi_1\,\mathcal{U}\,\varphi_2 \leftrightarrow \varphi_2 \vee (\varphi_1 \wedge \langle\!\langle A\rangle\!\rangle\bigcirc\langle\!\langle A\rangle\!\rangle\varphi_1\,\mathcal{U}\,\varphi_2)$.
($\mathbf{LFP}_{\mathcal{U}}$) $\langle\!\langle\varnothing\rangle\!\rangle\Box\,((\varphi_2 \vee (\varphi_1 \wedge \langle\!\langle A\rangle\!\rangle\bigcirc\theta)) \to \theta) \to \langle\!\langle\varnothing\rangle\!\rangle\Box\,(\langle\!\langle A\rangle\!\rangle\varphi_1\,\mathcal{U}\,\varphi_2 \to \theta)$.

Also, the following inference rules preserves validity [1]*:*

$$\frac{\varphi \to \psi}{\langle\!\langle A\rangle\!\rangle\bigcirc\varphi \to \langle\!\langle A\rangle\!\rangle\bigcirc\psi}\ \text{(monotonicity)} \qquad \frac{\varphi}{\langle\!\langle\emptyset\rangle\!\rangle\Box\varphi}\ \text{(necessitation)}$$

Proof. The proof of ($\mathbf{FP}_{\Box}$) *in our system is trivial, because we have used this formula as a definition for* $\langle\!\langle A\rangle\!\rangle\Box$*. Formula* ($\mathbf{GFP}_{\Box}$) *is a consequence of the use of a coinductive type for this operator. A similar consideration can be done about formulas* ($\mathbf{FP}_{\mathcal{U}}$)*, used to define formulas involving* $\mathcal{U}$*; and* ($\mathbf{LFP}_{\mathcal{U}}$)*, consequence of the inductive definition. Formula* (Σ) *is valid only in classical logic. In constructive logic we can prove* (Σ')*:* $\neg\langle\!\langle\varnothing\rangle\!\rangle\bigcirc\neg\varphi \to \neg\neg\langle\!\langle\Sigma\rangle\!\rangle\bigcirc\varphi$*. To demonstrate the equivalence* (Σ) $\leftrightarrow$ (Σ') *from classical logic in our system, we must add the excluded middle law explicitly. Proof of* (**S**) *involves reasoning about union of coalitions and strategies, as well as relating the "join" of coalition strategies (collaborative game) and the traces in which each coalition plays regardless the other one (competitive game). These results are properties about game structures, and we have proved them in [23] using definitions introduced in Section 2.1. Rule monotonicity is proved by showing that strategy* F_A *given by premise* $\langle\!\langle A\rangle\!\rangle\bigcirc\varphi$ *is an A strategy ensuring* ψ *in all states* $q' \in suc(\langle\rangle, q, F_A)$*. We prove necessitation by coinduction, unfolding Def. 6 and using the fact that* φ *is valid in all states.* □

To show that our formalization can be used as a suitable proof system for ATL, we have proved in [23] an extensive list of ATL theorems taken from [10]. Lemma 1 shows a list with a subset of such formulas.

Lemma 1 (Derived formulas). *The following judges can be proved valid in our formalization:*

(1) Regularity: $\vdash \langle\!\langle A\rangle\!\rangle\bigcirc\varphi \to \neg\langle\!\langle\Sigma\backslash A\rangle\!\rangle\bigcirc\neg\varphi$.
(2) And monotonicity: $\vdash \langle\!\langle A\rangle\!\rangle\bigcirc(\varphi \wedge \psi) \to \langle\!\langle A\rangle\!\rangle\bigcirc\varphi$.
(3) Coalition monotonicity: $\vdash \langle\!\langle A\rangle\!\rangle\bigcirc\varphi \to \langle\!\langle A \uplus B\rangle\!\rangle\bigcirc\varphi$.
(4) Monotonicity of $\langle\!\langle\ \rangle\!\rangle\Box$*:* $(\varphi \to \psi) \vdash \langle\!\langle A\rangle\!\rangle\Box\varphi \to \langle\!\langle A\rangle\!\rangle\Box\psi$.
(5) Monotonicity of $\langle\!\langle\ \rangle\!\rangle\,\mathcal{U}$*:* $(\varphi \to \varphi'), (\psi \to \psi') \vdash \langle\!\langle A\rangle\!\rangle\varphi\,\mathcal{U}\,\psi \to \langle\!\langle A\rangle\!\rangle\varphi'\,\mathcal{U}\,\psi'$.
(6) Necessitation of $\langle\!\langle\ \rangle\!\rangle\Box$*:* $\varphi \vdash \langle\!\langle A\rangle\!\rangle\Box\varphi$.
(7) Induction for $\langle\!\langle\ \rangle\!\rangle\Box$*:* $(\varphi \to (\psi \wedge \langle\!\langle A\rangle\!\rangle\bigcirc\varphi)) \vdash \varphi \to \langle\!\langle A\rangle\!\rangle\Box\psi$.
(8) Induction for $\langle\!\langle\ \rangle\!\rangle\,\mathcal{U}$*:* $(\psi \vee (\varphi \wedge \langle\!\langle A\rangle\!\rangle\bigcirc\chi) \to \chi) \vdash \langle\!\langle A\rangle\!\rangle\varphi\,\mathcal{U}\,\psi \to \chi$.

5 A Case Study

The formalization presented in Section 3 has been used in Section 4 to prove general properties over CGS and the logic ATL. In this section, we specify and

[1] We omit the modus ponens rule from [10], since this rule is already valid in our meta-logic via the shallow embedding.

verify a simple concrete system which is a good guide to model and analyze many systems. Section 5.1 presents an example taken from [2], describing a control protocol for a train entering a railroad crossing. Section 5.2 presents a generalization of this model where an unknown number of trains compete to enter a gate, and the gate controller must ensure some safety and liveness properties. This example can not be directly analyzed using model checking techniques because it involves an unbounded space of states.

5.1 Controlling a Railroad Crossing

We formalize a protocol for a train entering a railroad crossing with a finite CGS. All components for this CGS are instantiated using definitions presented in Section 3.2, and some properties for the system are specified using ATL formulas as described in Section 3.3.

Example 1. The CGS $S_T = \langle k, Q, \mathit{\Pi}, \pi, d, \delta \rangle$ has the following components:

- $k = 2$. Player 1 represents the train, and player 2 the gate controller.
- $Q = \{q_{out}, q_{req}, q_{gran}, q_{in}\}$.
- $\mathit{\Pi} = \{\mathit{Out}, \mathit{Request}, \mathit{In\_gate}, \mathit{Grant}\}$.
- $\pi(q_{out}) = \{\mathit{Out}\}$, the train is outside the gate; $\pi(q_{req}) = \{\mathit{Out}, \mathit{Request}\}$, the train is still outside the gate, but has requested to enter; $\pi(q_{gran}) = \{\mathit{Out}, \mathit{Grant}\}$, the controller has given the train permission to enter the gate; $\pi(q_{in}) = \{\mathit{In\_gate}\}$, the train is in the gate.
-
    - $d_1(q_{out}) = 2$ and $d_2(q_{out}) = 1$.
      At q_{out}, the train can choose to either stay outside the gate, or request to enter the gate.
    - $d_1(q_{req}) = 1$ and $d_2(q_{req}) = 3$.
      At q_{req}, the controller can choose to either grant the train permission to enter the gate, or deny the train's request, or delay the handling of the request.
    - $d_1(q_{gran}) = 2$ and $d_2(q_{gran}) = 1$.
      At q_{gran}, the train can choose to either enter the gate, or relinquish its permission to enter the gate.
    - $d_1(q_{in}) = 1$ and $d_2(q_{in}) = 2$.
      At q_{in}, the controller can choose to either keep the gate closed, or reopen the gate to new requests.
- The transition function δ is depicted in Figure 1.

A Model Based on CGS. In order to prove properties of the protocol described in Example 1, we proceed to model all the components of S_T following definitions presented in Section 3.2.

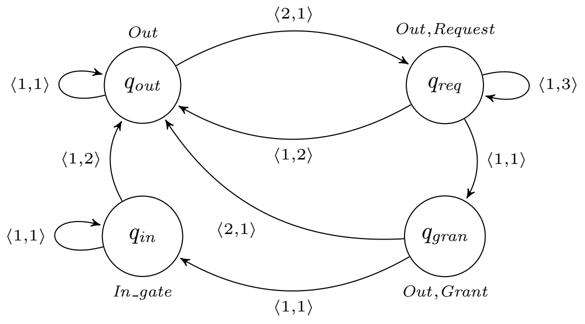

Fig. 1. Graphical representation of Example 1.

States, Players and Moves. These sets are introduced as types with one constructor for each element in the set, excepting the sets of moves, where a unique constructor is used to represent an *idle* move.

$$\begin{aligned}
&\mathit{State} \;\; : \mathit{Set} \overset{\text{def}}{=} \mid q_{out} \mid q_{req} \mid q_{gran} \mid q_{in} \\
&\mathit{Player} : \mathit{Set} \overset{\text{def}}{=} \mid \mathit{Train} \mid \mathit{Controller} \\
&\mathit{Move} \;\; : \mathit{Set} \overset{\text{def}}{=} \mid \mathit{stayOut} \mid \mathit{request} \mid \mathit{grant} \mid \mathit{delay} \mid \mathit{deny} \mid \mathit{enter} \\
&\qquad\qquad\quad\;\; \mid \mathit{relinquish} \mid \mathit{keepClosed} \mid \mathit{reopen} \mid \mathit{idle}
\end{aligned}$$

We use the tuple notation $\langle m_t, m_c\rangle$ to denote the move vector: $\lambda\, p : \mathit{Player} \Rightarrow (\mathit{match}\ p\ \mathit{with}\, \mathit{Train} \Rightarrow m_t \mid \mathit{Controller} \Rightarrow m_c)$.

Transitions. Transitions are introduced with the following predicate [2]:

$$\begin{array}{llllllll}
\multicolumn{8}{l}{\delta : \mathit{State} \to \langle \mathit{Move}\rangle \to \mathit{State} \to \mathit{Prop} \overset{\text{def}}{=}} \\
\mid \delta & q_{out} & \langle \mathit{stayOut}, \mathit{idle}\rangle & q_{out} & \mid \delta\ q_{out} & \langle \mathit{request}, \mathit{idle}\rangle & q_{req} \\
\mid \delta & q_{req} & \langle \mathit{idle}, \mathit{grant}\rangle & q_{gran} & \mid \delta\ q_{req} & \langle \mathit{idle}, \mathit{delay}\rangle & q_{req} \\
\mid \delta & q_{req} & \langle \mathit{idle}, \mathit{deny}\rangle & q_{out} & \mid \delta\ q_{gran} & \langle \mathit{enter}, \mathit{idle}\rangle & q_{in} \\
\mid \delta & q_{gran} & \langle \mathit{relinquish}, \mathit{idle}\rangle & q_{out} & \mid \delta\ q_{in} & \langle \mathit{idle}, \mathit{keepClosed}\rangle & q_{in} \\
\mid \delta & q_{in} & \langle \mathit{idle}, \mathit{reopen}\rangle & q_{out} & & &
\end{array}$$

Coalitions. Singleton sets of players $T = \{\mathit{Train}\}$ and $C = \{\mathit{Controller}\}$ are defined as:

$$\begin{aligned}
T &\overset{\text{def}}{=} \lambda\, p \Rightarrow \mathit{match}\ p\ \mathit{with}\, \mathit{Train} \Rightarrow \{1\} \mid \mathit{Controller} \Rightarrow \{\ \} \\
C &\overset{\text{def}}{=} \lambda\, p \Rightarrow \mathit{match}\ p\ \mathit{with}\, \mathit{Train} \Rightarrow \{\ \} \mid \mathit{Controller} \Rightarrow \{1\}
\end{aligned}$$

Atomic State Formulas. The atomic state formulas are easily introduced using case analysis over the current state. For example, a state formula representing the fact that the train is not in the gate is:

$$\mathit{OutGate} \overset{\text{def}}{=} \lambda\, q \Rightarrow \mathit{match}\ q\ \mathit{with}\ q_{in} \Rightarrow \mathit{False} \mid \_ \Rightarrow \mathit{True}$$

[2] For the sake of readability, we omit here the name of contructors.

. In a similar way, we have defined formulas *Requested*, *Granted* and *InGate* according to Example 1.

Proving Properties. The following properties, taken from [2], are provable in our system:

1. Whenever the train is out of the gate, the controller cannot force it to enter the gate:
$$\langle\langle \varnothing \rangle\rangle \Box\, (\mathit{OutGate} \rightarrow \neg \langle\langle C \rangle\rangle \Diamond \mathit{InGate})$$
2. Whenever the train is out of the gate, the train and the controller can cooperate so that the train will enter the gate:
$$\langle\langle \varnothing \rangle\rangle \Box\, (\mathit{OutGate} \rightarrow \langle\langle \Sigma \rangle\rangle \Diamond \mathit{InGate})$$

For space constraints, we omit proofs here, and we refer the interested reader to [23].

5.2 Controlling an Unbounded Number of Trains

Suppose there is an unknown number of trains to cross a single gate. The gate controller must ensure some safety (for instance, at most one train is in the gate) and liveness (for instance, a request must be processed) properties.

Formalizing the System Using CGS. We propose an extendend CGS S_∞ as a model of the system described above.

Players. The system components are the controller and the set of trains:

$$\mathit{Player} : \mathsf{Set} \overset{\text{def}}{=} \mathit{Train} : \mathit{Id} \rightarrow \mathit{Player} \mid \mathit{Controller} : \mathit{Player}$$

where $\mathit{Id} \overset{\text{def}}{=} \mathbb{N}$. We abbreviate t_n the term $\mathit{Train}\ n$, denoting the n-th train.

States. In each state of the system, we should have information about the trains that have made a request to enter the gate, and which train has obtained such permission. To represent the set of trains that want to enter to the gate, we introduce the type $\mathit{Petition} \overset{\text{def}}{=} \mathit{Id} \rightarrow \mathit{Bool}$. For a function $f : \mathit{Petition}$, we say that t_n wants to enter the gate if $f\ t_n = \mathit{true}$. The set of states is defined as:

$$\begin{array}{lll} \mathit{State} : \mathsf{Set} \overset{\text{def}}{=} & \mid q_{out} : \mathit{State} & \mid q_{req} : \mathit{Petition} \rightarrow \mathit{State} \\ & \mid q_{gran} : \mathit{Petition} \rightarrow \mathit{Id} \rightarrow \mathit{State} & \mid q_{in} : \mathit{Petition} \rightarrow \mathit{Id} \rightarrow \mathit{State} \end{array}$$

The first argument of states q_{req}, q_{gran} and q_{in} is used to represent the set of trains that have made a request. The second argument of state q_{gran} (q_{in}) is the *id* of the train having permission to enter (has entered) the gate.

Moves and Move Vectors. The set of moves is similar to the finite case. Additional moves are used for communication between components. The set of moves is extended in the following way:

$$\begin{array}{lllll} \mathit{Move} \stackrel{\mathrm{def}}{=} & \mid \mathit{stayOut} : \mathit{Move} & \mid \mathit{request} : \mathit{Move} & \mid \mathit{grant} : \mathit{Id} \to \mathit{Move} \\ & \mid \mathit{delay} : \mathit{Move} & \mid \mathit{deny} : \mathit{Id} \to \mathit{Move} & \mid \mathit{denyAll} : \mathit{Move} \\ & \mid \mathit{enter} : \mathit{Move} & \mid \mathit{relinquish} : \mathit{Move} & \mid \mathit{keepClosed} : \mathit{Move} \\ & \mid \mathit{reopen} : \mathit{Move} & \mid \mathit{idle} : \mathit{Move} & \end{array}$$

In the following moves appear the main difference with the finite example: $(\mathit{deny}\ n)$ represents a move where the controller rejects a request from train t_n, *denyAll* models a situation where controller can reject all requests, and $(\mathit{grant}\ n)$ represents a situation where controller gives permission to t_n.

Let $m_c : \mathit{Move}$ be a move of the controller and let $m_t : \mathit{Id} \to \mathit{Move}$ be a function assigning a move to each train, we use the notation $\langle m_t, m_c \rangle$ to represent the move vector defined as $\lambda\, p \Rightarrow (\mathit{match}\ p\ \mathit{with}\ t_n \Rightarrow m_t\ n \mid \mathit{Controller} \Rightarrow m_c)$.

Transitions. To model the transition relation we use the following auxiliary functions: $=_b$: $\mathit{Id} \to \mathit{Id} \to \mathit{Bool}$, that decides equality in type Id; and an overwrite operator $\oplus : \mathit{Petition} \to \mathit{Id} \to \mathit{Bool} \to \mathit{Petition}$, such that $(f \oplus \{n \leftarrow b\})$ applied to m returns b if $m = n$, and $f\ m$ otherwise. The transition relation is defined as follows [3]:

$$\begin{array}{ll} \delta \stackrel{\mathrm{def}}{=} & \mid \delta\ q_{out}\ \langle \lambda\, n \Rightarrow \mathit{stayOut}, \mathit{idle} \rangle\ q_{out} \\ & \mid \forall f, (\exists n : \mathit{Id}, f\ n = \mathit{true}) \to \\ & \quad \delta\ q_{out}\ \langle \lambda\, n \Rightarrow \mathit{if}\ \ f\, n\ \ \mathit{then}\ \ \mathit{request}\ \ \mathit{else}\ \ \mathit{stayOut}, \mathit{idle} \rangle\ (q_{req}\ f) \\ & \mid \forall f\, n, f\ n = \mathit{true} \to \\ & \quad \delta\ (q_{req}\ f)\ \langle \lambda\, n \Rightarrow \mathit{idle}, \mathit{grant}\ n \rangle\ (q_{gran}\ (f \oplus \{n \leftarrow \mathit{false}\})\ n) \\ & \mid \forall f, \delta\ (q_{req}\ f)\ \langle \lambda\, n \Rightarrow \mathit{idle}, \mathit{delay} \rangle\ (q_{req}\ f) \\ & \mid \forall f\, n, (\exists m : \mathit{Id}, m \neq n \wedge f\ m = \mathit{true}) \to \\ & \quad \delta\ (q_{req}\ f)\ \langle \lambda\, n \Rightarrow \mathit{idle}, \mathit{deny}\ n \rangle\ (q_{req}\ f \oplus \{n \leftarrow \mathit{false}\}) \\ & \mid \forall f\, n, (\forall m : \mathit{Id}, m \neq n \to f\ m = \mathit{false}) \to \\ & \quad \delta\ (q_{req}\ f)\ \langle \lambda\, n \Rightarrow \mathit{idle}, \mathit{deny}\ n \rangle\ q_{out} \\ & \mid \forall f, \delta\ (q_{req}\ f)\ \langle \lambda\, n \Rightarrow \mathit{idle}, \mathit{denyAll} \rangle\ q_{out} \\ & \mid \forall f\, n, \delta\ (q_{gran}\ f\ n)\ \langle \mathit{enter}_n, \mathit{idle} \rangle\ (q_{in}\ f\ n) \\ & \mid \forall f\, n, (\forall k : \mathit{Id}, k \neq n \to f\ k = \mathit{false}) \to \\ & \quad \delta\ (q_{gran}\ f\ n)\ \langle \mathit{relinquish}_n, \mathit{idle} \rangle\ q_{out} \\ & \mid \forall f\, n, (\exists k : \mathit{Id}, k \neq n \wedge f\ k = \mathit{true}) \to \\ & \quad \delta\ (q_{gran}\ f\ n)\ \langle \mathit{relinquish}_n, \mathit{idle} \rangle\ (q_{req}\ f) \\ & \mid \forall f\, n, \delta\ (q_{in}\ f\ n)\ \langle \lambda\, n \Rightarrow \mathit{idle}, \mathit{keepClosed} \rangle\ (q_{in}\ f\ n) \\ & \mid \forall f\, n, (\forall m, f\, m = \mathit{false}) \to \delta\ (q_{in}\, f\, n)\ \langle \lambda\, n \Rightarrow \mathit{idle}, \mathit{reopen} \rangle\ q_{out} \\ & \mid \forall f\, n, (\exists m, f\, m = \mathit{true}) \to \delta\ (q_{in}\, f\, n)\ \langle \lambda\, n \Rightarrow \mathit{idle}, \mathit{reopen} \rangle\ (q_{req}\, f) \end{array}$$

where $\mathit{enter}_n, \mathit{relinquish}_n : \mathit{Id} \to \mathit{Move}$ are defined as:

$$\begin{array}{ll} \mathit{enter}_n & \stackrel{\mathrm{def}}{=} \lambda\, m \Rightarrow \mathit{if}\ \ m =_b n\ \ \mathit{then}\ \ \mathit{enter}\ \ \mathit{else}\ \ \mathit{idle} \\ \mathit{relinquish}_n & \stackrel{\mathrm{def}}{=} \lambda\, m \Rightarrow \mathit{if}\ \ m =_b n\ \ \mathit{then}\ \ \mathit{relinquish}\ \ \mathit{else}\ \ \mathit{idle} \end{array}$$

[3] We have omitted constructors names.

The relation δ takes into account the existence of different train requests using the petition function. For instance, when the system is in state q_{out}, there are two possible transitions: (1) no train make a request, then the system stays in q_{out}; and (2) there exists a subset of trains making a request to enter the gate, represented with f; in this case, the system make a transition to state $(q_{req}\ f)$.

Coalitions. Different coalitions can be defined for this system, depending on the properties to be specified. For example:

$$\begin{array}{ll} \{t_n\} \overset{\text{def}}{=} \lambda\, p \Rightarrow \mathit{match}\ p\ \mathit{with} & \mid \mathit{Train}\ k \Rightarrow \mathit{if}\ n =_b k\ \mathit{then}\ \{1\}\ \mathit{else}\ \{\ \} \\ & \mid \mathit{Controller} \Rightarrow \{\ \} \end{array}$$

State Formulas. State formulas can be defined by pattern matching on states. For example, we define formula *Out*, valid if the current state is q_{out}, and $\mathit{In}(n)$, valid if train t_n is in the gate:

$$\begin{array}{lll} \mathit{Out} & \overset{\text{def}}{=} \lambda\, q \Rightarrow \mathit{match}\ q\ \mathit{with} & \mid q_{out} \qquad \Rightarrow \mathit{True} \mid \_ \Rightarrow \mathit{False} \\ \mathit{In}(n) & \overset{\text{def}}{=} \lambda\, q \Rightarrow \mathit{match}\ q\ \mathit{with} & \mid q_{in}\ f\ m \Rightarrow \mathit{if}\ n =_b m\ \mathit{then}\ \mathit{True}\ \mathit{else}\ \mathit{False} \\ & & \mid \_ \qquad\quad \Rightarrow \mathit{False} \end{array}$$

Properties. Some properties proved in S_∞ are:

– Controller and train t_n can cooperate so that this train will enter the gate:

$$\langle\!\langle \varnothing \rangle\!\rangle \Box\, (\mathit{Out} \to \langle\!\langle \{t_n\} \uplus \{\mathit{Controller}\} \rangle\!\rangle \Diamond \mathit{In}(n)) \tag{12}$$

– Cooperation is needed in order to ensure progress: Neither the set of trains nor the controller can enforce a trace where state $\mathit{In}(n)$ is reached, for some n:

$$\langle\!\langle \varnothing \rangle\!\rangle \Box\, (\mathit{Out} \to \neg\, (\langle\!\langle \{\mathit{Controller}\} \rangle\!\rangle \Diamond \mathit{In}(n) \vee \langle\!\langle \{t_1, t_2, \ldots\} \rangle\!\rangle \Diamond \mathit{In}(n))) \tag{13}$$

Formula (12) express a liveness property. To prove it, we construct a strategy F_A for coalition $A = \{t_n, \mathit{Controller}\}$; then, we proceed to show that, if player in A follows strategy F_A, a state where $\mathit{In}(n)$ is valid will be eventually reached, regardless the behaviour of the other components. To prove the safety property (13), we show that it is not the case that controller (the set of trains) can construct an strategy F such that, if controller (the set of trains) follows F, then state q_{in} will be eventually reached. A detailed proof of these properties can be found in [23], along with the analysis of other safety and liveness properties.

6 Conclusions and Future Work

ATL is a game-theoretic generalization of CTL with applications in the formal verification of multi-agent systems. In this paper we have presented a formalization of ATL and its semantic model CGS. Unlike standard ATL semantics, temporal operators have been interpreted in terms of inductive and coinductive types, using a fixpoint characterization of these operators in the CIC.

The formalization presented here was used to model a concurrent system with an unbounded number of players and states, and we have verified some safety and liveness properties expressed as ATL formulas. Unlike automatic techniques, our formal model has no restriction in the size of the CGS, and arbitrary state predicates can be used as atomic propositions of ATL. We conclude that in systems with an intractable size, our formal model, based on an existent type theory (the CIC) with the proof assistant Coq can be used as a specification and verification tool for open multi-agent systems.

A possible extension of our system would consist of formalizing fair-ATL [2], a logic extending ATL semantics with fairness constraints. These constraints rule out certain infinite computations that ignore enabled moves forever.

The logic ATL is a fragment of a more expressive logic, ATL* [2]. In ATL*, a path quantifier $\langle\!\langle A \rangle\!\rangle$ is followed by an arbitrary linear time formula, allowing boolean combination and nesting, over $\bigcirc$, $\Box$ and $\mathcal{U}$. Another interesting extension to our work is to formalize this logic in the CIC.

ATL has been used to specify properties in contract signing protocols where n agents exchange signatures [11, 5]. The model checker Mocha [3] has succeeded in verifying these protocols in the case where two agents are involved [5]. However, model checking algorithms fail in case of multi-party protocols ($n > 2$), since these algorithms can be used only with a fixed (and, in practice, small) value for n.

The formalization presented in this work can be used as basis for a formal verification of such protocols. Thus, a further extension of this work involves the verification of multi-party protocols following an approach similar to the one of Section 5.

References

1. R. Alur, T. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Revised Lectures from the International Symposium on Compositionality: The Significant Difference*, COMPOS'97, pages 23–60, London, UK, 1998. Springer-Verlag.
2. R. Alur, T. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49:672–713, 2002.
3. R. Alur, T. Henzinger, F. Mang, S. Qadeer, S. Rajamani, and S. Tasiran. Mocha: Modularity in model checking. In *CAV '98: Proceedings of the 10th International Conference on Computer Aided Verification*, pages 521–525, London, UK, 1998. Springer-Verlag.
4. Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science. Springer Verlag, 2004.
5. R. Chadha, S. Kremer, and A. Scedrov. Formal analysis of multiparty contract signing. *Journal of Automated Reasoning*, 36(1-2):39–83, 2006.
6. T. Coquand and G. Huet. The Calculus of Constructions. In *Information and Computation*, volume 76, pages 95–120. Academic Press, February/March 1988.

7. S. Coupet-Grimal. LTL in Coq. Contributions to the Coq system, Laboratoire d'Informatique Fondamentale de Marseille, 2002. Available at `http://coq.inria.fr/contribs/LTL.tar.gz`.
8. E. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, pages 995–1072. Elsevier, 1995.
9. E. Giménez. *A Calculus of Infinite Constructions and its application to the verification of communicating systems.* PhD thesis, Ecole Normale Supérieure de Lyon, 1996.
10. V. Goranko and G. van Drimmelen. Complete axiomatization and decidability of alternating-time temporal logic. *Theoretical Computer Science*, 353(1):93–117, 2006.
11. S. Kremer and J. Raskin. A game-based verification of non-repudiation and fair exchange protocols. *Journal of Computer Security*, 11(3):399–429, 2003.
12. X. Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.
13. C. Luna. Computation tree logic for reactive systems and timed computation tree logic for real time systems. Contributions to the Coq system, Universidad de la República, Uruguay, 2000.
14. S. Merz. An encoding of TLA in Isabelle. Technical report, Institut für Informatic, Universität München, Germany, 1999.
15. C. Paulin-Mohring. Inductive definitions in the system coq - rules and properties. In *TLCA '93: Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 328–345, London, UK, 1993. Springer-Verlag.
16. A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.
17. A. Pnueli and T. Arons. TLPVS: A PVS-based LTL verification system. In Nachum Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, pages 598–625. Springer, 2003.
18. K. Schneider and D.W. Hoffmann. A HOL conversion for translating linear time temporal logic to omega-automata. In *Theorem Proving in Higher Order Logics*, pages 255–272. Springer, 1999.
19. The Coq development team. *The Coq proof assistant reference manual, version 8.2.* LogiCal Project, 2010. Distributed electronically at `http://coq.inria.fr`.
20. The Coq development team. *The Coq Standard Library.* LogiCal Project, 2010. Available at `http://coq.inria.fr/stdlib/`.
21. J. van Leeuwen, editor. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics.* Elsevier and MIT Press, 1990.
22. D. Zanarini. Formalización de lógica temporal alternante en el cálculo de construcciones coinductivas. Master's thesis, FCEIA, Universidad Nacional de Rosario, Argentina, 2008. Available at `www.fceia.unr.edu.ar/~dante`.
23. D. Zanarini. Formalization of alternating time temporal logic in Coq, 2010. Available at `www.fceia.unr.edu.ar/~dante`.