An Introduction to Alternating-time Temporal Logic

Domenico Bloisi
bloisi@dis.uniroma1.it

Dipartimento di Informatica e Sistemistica
Sapienza University of Rome
The aim of this talk is to provide an introduction to Temporal Logics and in particular Alternating-time Temporal Logics. We will discuss:

- why to use temporal logics.
- ATL syntax and semantics.
- ATL model checking.
- ATL model checking complexity.
Temporal logic is used to describe any system of rules and symbolism for representing, and reasoning about, propositions qualified in terms of time.

In a temporal logic, statements can have a truth value which can vary in time.

Temporal logic has in addition to common logical operators (\(\neg\), \(\land\), \(\lor\), \(\rightarrow\)), special modal operators:
- \(\Diamond \varphi\) means “eventually \(\varphi\) has to hold”.
- \(\Box \varphi\) means “\(\varphi\) always holds”.

Temporal logic has found an important application in formal verification, where it is used to state requirements of hardware or software systems.
Model checking was originally developed as a technique for verifying that finite state systems satisfy specifications expressed in the language of temporal logics.
A closed system is a system whose behavior is completely determined by the state of the system.

- A **computation** is an infinite sequence of states.
In 1977, linear-time temporal logic (LTL) was proposed to specify requirements for reactive systems.

A formula of LTL is interpreted over a computation.

A reactive system satisfies an LTL formula if all its computations do.

LTL express properties on the possible executions of the model.

Due to the implicit use of universal quantification over the set of computations, LTL cannot express existential properties.
In 1981 Branching-time temporal logics were proposed to overcome LTL limitations.

Branching-time temporal logics can express requirements on states (which may have several possible futures) of the model.

Branching-time temporal logics such as **Computation Tree Logic** (CTL) do provide explicit quantification over the set of computations.

**Example.** For a state predicate \( \varphi \), the CTL formula \( \forall \diamond \varphi \) requires that a state satisfying \( \varphi \) is visited in all computations, and the CTL formula \( \exists \diamond \varphi \) requires that there exists a computation that visits a state satisfying \( \varphi \).
The logics LTL and CTL have their natural interpretation over the computations of closed systems.

the compositional modeling and design of reactive systems requires each component to be viewed as a system interacting with its environment.

The model of a reactive system cannot consider only its internal state but also the state of the environment the system reacts to.

We need a logic for modeling open systems.
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.
Satisfaction Problem

- **Universal satisfaction**  
  LTL  
  do all computations satisfy a property?

- **Existential satisfaction**  
  CTL  
  does some computations satisfy a property?

- **Alternating satisfaction**  
  ATL  
  can the system resolve its internal choices so that the satisfaction of a property is guaranteed no matter how the environment resolves the external choices?
Alternating satisfaction can be viewed as a winning condition in a two-player game between the system and the environment.
In order to capture compositions of open systems, we consider, instead of two-player games between system and environment, the more general setting of multi-player games, with a set $\Sigma$ of players that represent different components of the system and the environment.

- a player can be viewed as an agent acting in a multi-agent scenario and a system as a coalition of agents.
The natural “common-denominator” model for compositions of open systems is the *concurrent game structure* (CGS).

A semantics in terms of concurrent game structures maintains the differentiation of a design into system components and environment.

A concurrent game is played on a state space.

In each step of the game, every player chooses a move.

The combination of choices determines a transition from the current state to a successor state.
Concurrent Games

Special cases of a concurrent game are:

- **turn-based synchronous**
  in each step, only one player has a choice of moves, and that player is determined by the current state.

- **Moore synchronous**
  the state is partitioned according to the players, and in each step, every player updates its own component of the state independently of the other players.

- **turn-based asynchronous**
  in each step, only one player has a choice of moves, and that player is chosen by a fair scheduler.
The Game

For a set $A \subseteq \Sigma$ of players, a set $\Gamma$ of computations and a state $q$ of the system consider the following game between a \textbf{protagonist} and an \textbf{antagonist}:

- The game starts at the state $q$.
- At each step, to determine the next state, the protagonist resolves the choices controlled by the players in the set $A$, while the antagonist resolves the remaining choices.
- If the resulting infinite computation belongs to the set $\Gamma$, then the protagonist wins.
- Otherwise, the antagonist wins.
If the protagonist has a winning strategy, we say that the alternating-time formula $\langle\langle A \rangle \rangle \Gamma$ is satisfied in the state $q$.

$\langle\langle A \rangle \rangle$ can be viewed as a path quantifier, parameterized with the set $A$ of players, which ranges over all computations that the players in $A$ can force the game into, irrespective of how the players in $\Sigma \setminus A$ proceed.

The parameterized path quantifier $\langle\langle A \rangle \rangle$ is a generalization of the path quantifiers of branching-time temporal logics:
- the existential path quantifier $\exists$ corresponds to $\langle\langle \Sigma \rangle \rangle$
- the universal path quantifier $\forall$ corresponds to $\langle\langle \emptyset \rangle \rangle$. 
In a multi-processor system, one of the desired properties is the absence of *deadlocks*, where a deadlock state is one in which a processor, say *a*, is permanently blocked from accessing a memory cell.

- This requirement can be specified using the CTL formula
  \[ \forall \Box (\exists \Diamond \text{readable} \land \exists \Diamond \text{writable}) \]

- The ATL formula
  \[ \langle\langle \emptyset\rangle\rangle \Box (\langle\langle a\rangle\rangle \Diamond \text{readable} \land \langle\langle a\rangle\rangle \Diamond \text{writable}) \]
captures the informal requirement more precisely.

- While the CTL formula only asserts that it is always possible for all processors to cooperate so that *a* can eventually read and write (“collaborative possibility”), the ATL formula is stronger: it guarantees a memory access for processor *a*, *no matter what the other processors in the system do* (“adversarial possibility”).
Example Legged

- $a$ and $b$ robots of the team $A$.
- $⟨⟨A⟩⟩◊(⟨⟨a⟩⟩◊in\_possesso \land ⟨⟨b⟩⟩◊in\_attesa\_di\_ricevere)$
Compositional Reasoning

- ATL is better suited for compositional reasoning than CTL.
- If a component $a$ satisfies the CTL formula $\exists \Diamond \varphi$, we cannot conclude that a composite system $a \parallel b$ that contains $a$ as a component, also satisfies $\exists \Diamond \varphi$.
- If $a$ satisfies the ATL formula $\langle a \rangle \Diamond \varphi$ system, then so does $a \parallel b$. 
The temporal logic ATL is defined with respect to a finite set $\Pi$ of propositions and a finite set $\Sigma = \{1, \ldots, k\}$ of players. An ATL formula is one of the following:

- $p$, for propositions $p \in \Pi$.
- $\neg \varphi$ or $\varphi_1 \lor \varphi_2$, where $\varphi$, $\varphi_1$ and $\varphi_2$ are ATL formulas.
- $\langle\langle A \rangle\rangle \circ \varphi$, $\langle\langle A \rangle\rangle \Box \varphi$ or $\langle\langle A \rangle\rangle \varphi_1 U \varphi_2$, where $A \subseteq \Sigma$ is a set of players and $\varphi$, $\varphi_1$ and $\varphi_2$ are ATL formulas.

The operator $\langle\langle \cdot \rangle\rangle$ is a path quantifier, and $\circ$ ("next"), $\Box$ ("always"), and $U$ ("until") are temporal operators. We write $\langle\langle A \rangle\rangle \diamond \varphi$ for $\langle\langle A \rangle\rangle \text{true} U \varphi$. 
To evaluate a formula of the form $\langle\langle A \rangle\rangle \psi$ at a state $q$ of a concurrent game structure $S$, consider the following game between a protagonist and an antagonist.

- The game proceeds in an infinite sequence of rounds.
- After each round, the position of the game is a state of $S$.
- The initial position is $q$.
- Now consider the game in some position $u$. To update the position, first the protagonist chooses for every player $a \in A$ a move $j_a$.
- Then, the antagonist chooses for every player $b \in \Sigma \setminus A$ a move $j_b$ and the position of the game is updated to $\delta(u, j_1, \ldots, j_k)$.
- In this way, the game continues forever and produces a computation.
The protagonist wins the game if the resulting computation satisfies the subformula $\psi$, read as a linear temporal formula whose outermost operator is $\bigcirc$, $\Box$, or $U$;

otherwise, the antagonist wins.

The ATL formula $\langle\langle A \rangle\rangle \psi$ is satisfied at the state $q$ iff the protagonist has a winning strategy in this game.
We write $S, q \models \psi$ to indicate that the state $q$ satisfies the formula $\psi$ in the structure $S$. The satisfaction relation $\models$ is defined, for all states $q$ of $S$, inductively as follows:

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

- It is often useful to express an ATL formula in a dual form.
- For this purpose, we use the path quantifier $[[A]]$, for a set $A$ of players.
- While the ATL formula $\langle\langle A\rangle\rangle \psi$ intuitively means that the players in $A$ can cooperate to make $\psi$ true (they can “enforce” $\psi$), the dual formula $[[A]] \psi$ means that the players in $A$ cannot cooperate to make $\psi$ false (they cannot “avoid” $\psi$).
- Example. We can write $[[A]] \diamond \varphi$ for $\neg\langle\langle A\rangle\rangle \Box \neg \varphi$. 
Example: Train Protocol

A protocol for a train entering a railroad crossing.
Example: Train Protocol

- $k = 2$. Player 1 represents the train, and player 2 the gate controller.
- $Q = \{ q_0, q_1, q_2, q_3 \}$.
- $\Pi = \{ \text{out\_of\_gate, in\_gate, request, grant} \}$.
  - $\pi(q_0) = \{ \text{out\_of\_gate} \}$. The train is outside the gate.
  - $\pi(q_1) = \{ \text{out\_of\_gate, request} \}$. The train is still outside the gate, but has requested to enter.
  - $\pi(q_2) = \{ \text{out\_of\_gate, grant} \}$. The controller has given the train permission to enter the gate.
  - $\pi(q_3) = \{ \text{in\_gate} \}$. The train is in the gate.
Example: Train Protocol

- $q_1$ and $q_3$ are controlled when a computation is in one of these states, the controller chooses the next state.
- $q_0$ and $q_2$ are uncontrolled the train chooses successor states.
Whenever the train is out of the gate and does not have a grant to enter the gate, the controller can prevent it from entering the gate: 
\[\langle\langle\emptyset\rangle\rangle\Box((\text{out\_of\_gate} \land \neg \text{grant}) \rightarrow \langle\langle\text{ctr}\rangle\rangle\Box\text{out\_of\_gate}).\]

Whenever the train is out of the gate, the controller cannot force it to enter the gate: 
\[\langle\langle\emptyset\rangle\rangle\Box(\text{out\_of\_gate} \rightarrow [[\text{ctr}]]\Box\text{out\_of\_gate}).\]

Whenever the train is out of the gate, the train and the controller can cooperate so that the train will enter the gate: 
\[\langle\langle\emptyset\rangle\rangle\Box(\text{out\_of\_gate} \rightarrow \langle\langle\text{ctr}, \text{train}\rangle\rangle\Diamond\text{in\_gate}).\]
Example: Train Protocol

Whenever the train is out of the gate, it can eventually request a grant for entering the gate, in which case the controller decides whether the grant is given or not:

\[
\langle\langle\emptyset\rangle\rangle\Box (out\_of\_gate \rightarrow \\
\langle\langle train\rangle\rangle\Diamond (request \land (\langle\langle ctr\rangle\rangle\Diamond grant) \land (\langle\langle ctr\rangle\rangle\Box \neg grant))).
\]

Whenever the train is in the gate, the controller can force it out in the next step:

\[
\langle\langle\emptyset\rangle\rangle\Box (in\_gate \rightarrow \langle\langle ctr\rangle\rangle \bigcirc out\_of\_gate).
\]
Consider the ATL formulas
\[
\langle\langle\emptyset\rangle\rangle\Box((\text{out\_of\_gate} \land \neg \text{grant}) \rightarrow \langle\langle\text{ctr}\rangle\rangle\Box\text{out\_of\_gate}) \\
\langle\langle\emptyset\rangle\rangle\Box(\text{out\_of\_gate} \rightarrow [\langle\langle\text{ctr}\rangle\rangle]\Box\text{out\_of\_gate})
\]
They provide more information than the CTL formula
\[
\forall\Box(\text{out\_of\_gate} \rightarrow \exists\Box\text{out\_of\_gate}).
\]
While the CTL formula only requires the existence of a computation in which the train is always out of the gate, the two ATL formulas guarantee that no matter how the train behaves, the controller can prevent it from entering the gate, and no matter how the controller behaves, the train can decide to stay out of the gate.
ATL Symbolic Model Checking

Given a game structure $S$ and an ATL formula $\varphi$, the model-checking problem for ATL asks for the set of states that satisfy $\varphi$.

```
foreach $\varphi'$ in $\text{Sub}(\varphi)$ do
  case $\varphi' = p$:  [$\varphi'$] := $\text{Reg}(p)$
  case $\varphi' = \neg \theta$:  [$\varphi'$] := $[true] \setminus [\theta]$
  case $\varphi' = \theta_1 \lor \theta_2$:  [$\varphi'$] := $[\theta_1] \cup [\theta_2]$
  case $\varphi' = \langle A \rangle \circ \theta$:  [$\varphi'$] := $\text{Pre}(A, [\theta])$
  case $\varphi' = \langle A \rangle \Box \theta$:
    $\rho := [true]; \tau := [\theta];$
    while $\rho \not\subseteq \tau$ do $\rho := \tau; \tau := \text{Pre}(A, \rho) \cap [\theta]$ od;
    [$\varphi'$] := $\rho$
  case $\varphi' = \langle A \rangle \theta_1 \cup \theta_2$:
    $\rho := [false]; \tau := [\theta_2];$
    while $\tau \not\subseteq \rho$ do $\rho := \rho \cup \tau; \tau := \text{Pre}(A, \rho) \cap [\theta_1]$ od;
    [$\varphi'$] := $\rho$
end case
od;
return [$\varphi$].
```
[\varphi] denotes the desired set of states, we write [true] for the set of all states, and [false] for the empty set of states.

The function \textit{Sub}, when given an ATL formula \varphi, returns a queue of syntactic subformulas of \varphi such that if \varphi_1 is a subformula of \varphi and \varphi_2 is a subformula of \varphi_1, then \varphi_2 precedes \varphi_1 in the queue \textit{Sub}(\varphi).

The function \textit{Reg}, when given a proposition \( p \in \Pi \), returns the set of states that satisfy \( p \).

The function \textit{Pre}, when given a set \( A \subseteq \Sigma \) of players and a set \( \rho \) of states, returns the set of states \( q \) such that from \( q \), the players in \( A \) can cooperate and enforce the next state to lie in \( \rho \). Formally, \( \textit{Pre}(A, \rho) \) contains state \( q \) if for every player \( a \in A \), there exists a move \( j_a \) such that for all players \( b \in \Sigma \setminus A \) and moves \( j_b \), we have \( \delta(q, j_1, \ldots, j_k) \in \rho \).

Termination is guaranteed, because the state space is finite.
The model-checking problem for ATL is PTIME-complete, and can be solved in time $O(m \cdot \ell)$ for a game structure with $m$ transitions and an ATL formula of length $\ell$. The problem is PTIME-hard even for a fixed formula, and even in the special case of turn-based synchronous game structures.

That result was claimed by Alur, Henzinguer and Kupferman (September 2002).

Idea: The control structure of the algorithm is identical to symbolic algorithms for CTL model checking that is PTIME-complete.
The model-checking problem for ATL has the same complexity as the corresponding satisfiability problem that is EXPTIME-complete.

That result was claimed by van der Hoek, Lomuscio and Wooldridge (May 2006).

Idea: Alur, Henzingerer and Kupferman assumes that the CGS $S$ is explicitly enumerated in the input to the problem, and in particular that all the states of $S$ are listed in the input.
Model-checking problem for ATL on implicit CGSs is in $\Delta_3^P$.

That result was claimed by Laroussinie, Markey and Oreiby (May 2008).

**ATL Expressiveness.** Duality is a fundamental concept in temporal logic: for instance, the dual of modality $U$, often denoted by $R$, is defined by $pRq = \neg((\neg p)U(\neg q))$.

Laroussinie, Markey and Oreiby proved the following theorem: There is no ATL formula equivalent to $\Phi = \langle \langle A \rangle \rangle (pRq)$. 
Conclusions

- ATL have attracted much interest from the multi-agents system community due to its implicit modeling of multiple agents predicate.
- ATL may be deeply studied for understanding its expressiveness power.
- We can consider ATL model checking simply as a CTL model checking extension.