Alternating-Time Temporal Logic

Fabienne Sophie Eigner

fabieigner@aol.com

Abstract. In this paper we will introduce Alternating-Temporal Time Logic as defined by Rajeev Alur, Thomas Henzinger, and Orna Kupferman. We will give an intuition why this logic is more expressive than other known temporal logics, like LTL or CTL, for open structures, e. g., concurrent game structures. We will then describe a symbolic model checking algorithm for the logic and briefly discuss its runtime.

1 Introduction

Imagine the following situation: A train arrives at a crossing. It needs to drive through a gate in order to continue its journey. Whether or not the train is allowed to enter the gate is decided by a controller.

This or other (real world) situations are often modelled using abstractions. A classical approach is to use *Kripke structures*, that model the computations of a closed system. Known logics that are classically interpreted over such structures are *Linear-Time Temporal Logic* (LTL) [3] and *Computational Tree Logic* (CTL) [2]. Whereas LTL assumes implicit quantification over all branches that result from an execution of the system, CTL allows us to explicitly quantify (universally and existentially) over the paths.

But looking at the train-crossing example, we see that it would be more suitable to view it as an open system, where one (or multiple) player(s), here the train, interact with the environment, in this case the controller. A model for this kind of open system would be a *concurrent game structure*, which we will introduce more formally later.

Similar to closed systems, we want a logic for open systems to describe properties such a model should fulfill. For the above mentioned train-crossing example such properties could be as follows:

1. "Sooner or later the train will drive past the gate."
2. "It is possible that the train will never enter the gate."
3. "Can the controller prevent the train from entering the gate?"

The first two properties can be stated using CTL, where the formulas would be of the following form: $\forall\Diamond in\_gate$ and $\exists\Box out\_of\_gate$. The third property, however, refers to the individual powers an agent in the game possesses and is not describable in CTL (much less in LTL).

A logic that can indeed be used to describe properties like the third one is *Alternating-Time Temporal Logic* (ATL), which we will introduce in the following. With this logic, we can give a specification for a model. Checking whether a model fulfills a specification is called *model-checking*. We will later give a symbolic algorithm that can be used to solve the model-checking problem for ATL. Note that there are some variants of ATL, such as ATL* (which corresponds to an extension of CTL*) and Fair ATL (which deals with additional fairness constraints), which we will not discuss in this paper. For further reference, have a look at [1].

2 Concurrent Game Structures

We need a computational model to describe compositions of open systems, where the system components and an environment interact. We choose concurrent game structures as such a model.

We define a concurrent game structure $S = (k, Q, \Pi, \pi, d, \delta)$ as follows:

- $k \in \mathbb{N}$ denotes the number of *players* (named from 1 to k).
- Q denotes the finite set of *states*.
- Π denotes the finite set of *propositions*.
- $\pi : Q \to 2^{\Pi}$ is called the *labeling function*, where $\pi(q)$ returns the set of propositions that hold at state q.
- $d : \{1, \ldots, k\} \times Q \to \mathbb{N}^{+}$, where $d_a(q)$ describes the number of possible moves of player a at state q. The possible moves of this player a at state q are identified with the numbers $1, \ldots, d_a(q)$. For each state q in Q, we define a *move vector* to be a tuple $\langle j_1, \ldots, j_k \rangle$, where $1 \leq j_a \leq d_a(q)$ for each a. Here j_a denotes a possible move of player a. For a state q we write $D(q)$, called the *move function*, for the set of move vectors $\{1, \ldots, d_1(q)\} \times \ldots \times \{1, \ldots, d_k(q)\}$.
- δ describes the *transition function*, which assigns to each state q and to each move vector $\langle j_1, \ldots, j_k \rangle$ a state $\delta(q, j_1, \ldots, j_k)$ that results from q if each player a executes the corresponding move j_a.

As an example, we will now define the train-crossing problem as a somewhat simplified (turn-based) game between the train and the controller. In turn-based games, each state is assigned to one player a, the other player(s) can only do one possible move at this state, that simply follows whatever a chooses.

For an intuitive definition of the train-crossing game, see Figure 1. Here we have $k = 2$ players, the train and the controller. The set of states Q is defined as $\{q0, q1, q2, q3\}$, where the states $q0$ and $q2$ are assigned to the train, and $q1$ and $q3$ to the controller. The set of propositions consiststs of four elements: *out_of_gate*, *in_gate*, *request*, and *grant*. The labeling function can be easily read off the graph, e. g., $\pi(q1) = \{$*out_of_gate*, *request*$\}$. For simplicity, we will omit a formal definition of d and δ, as this is implicitely defined by the transition-arcs of the graph.

3 Alternating-Time Temporal Logic

Alternating-Time Temporal Logic was introduced by Rajeev Alur, Thomas Henzinger, and Orna Kupferman in [1]. It uses operators known from LTL and CTL,

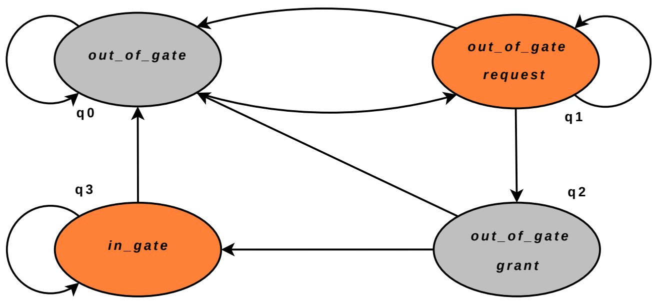

Fig. 1. Turn-based concurrent game structure for the train-crossing problem

whose intuitive meanings are given in Figure 2. Additionally, it introduces a selective form of quantors. We will now formally define the syntax and semantics of ATL.

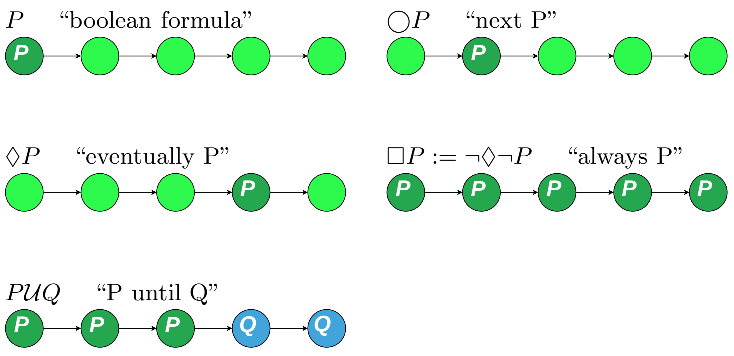
P "boolean formula"
P
$\bigcirc P$ "next P"
P
$\Diamond P$ "eventually P"
P
$\Box P := \neg\Diamond\neg P$ "always P"
P P P P P
$P\mathcal{U}Q$ "P until Q"
P P P Q Q

Fig. 2. Intuitive semantics of LTL operators

3.1 Syntax of ATL

An ATL formula is defined w. r. t. a set $\Sigma = \{1, \ldots, k\}$ of players and a finite set Π of propositions. It has one of the following forms:

- **(S1)**: p, where $p \in \Pi$
- **(S2)**: $\neg\phi$ or $\phi \vee \psi$, where ϕ, ψ ATL formulas
- **(S3)**: $\langle\langle A\rangle\rangle \bigcirc \phi$, or $\langle\langle A\rangle\rangle\Box\phi$, or $\langle\langle A\rangle\rangle\Diamond\phi$, or $\langle\langle A\rangle\rangle\phi\ \mathcal{U}\ \psi$, where $A \subseteq \Sigma$ is a set of players, and ϕ, ψ are ATL formulas

We will also define the dual of the quantifier as $[[A]]\phi := \neg\langle\langle A\rangle\rangle\neg\phi$. Other boolean operators like $\wedge, \rightarrow, \leftrightarrow$ can be used for notational convenience, since they can be easiy constructed from the given $\neg, \vee$.

3.2 Semantics of ATL

Above, we have seen the intuitive meanings of formulas only constisting of type **S1** and **S2**. The intuitive meaning of formulas of type **S3**, containing the quantifier $\langle\langle A\rangle\rangle\phi$ or its dual $[[A]]\phi$ is as follows:

- $\langle\langle A\rangle\rangle\phi$: Players in A can cooperate to make ϕ true
- $[[A]]\phi$: Players in A cannot cooperate to make ϕ false

Let us now define the semantics of ATL formally. Note that we will not define the $\Diamond$ - operator, as it can be replaced by the other operators.

We say that a state $q \in Q$ satifies the ATL formula ϕ in a structure S ($S, q \models \phi$), iff:

- $q \models p$ for propositions $p \in \Pi$, iff $p \in \pi(q)$
- $q \models \neg\phi$, iff $q \not\models \phi$
- $q \models \phi \vee \psi$, iff $q \models \phi$ or $q \models \psi$
- $q \models \langle\langle A\rangle\rangle \bigcirc \phi$, iff there exists a strategy for each player in A, such that for all computations λ starting fom q and following these strategies, it holds that: $\lambda[1] \models \phi$
- $q \models \langle\langle A\rangle\rangle\Box\phi$, iff for λ as defined above, it holds that $\forall i \geq 1$: $\lambda[i] \models \phi$
- $q \models \langle\langle A\rangle\rangle\phi \;\mathcal{U}\; \psi$, iff for λ as defined above, there is a position $i \geq 1$, such that: $\forall\ 1 \leq j < i :\ \lambda[j] \models \phi$ and $\lambda[i] \models \psi$

3.3 Examples for ATL Formulas

We will now show what kind of properties one can state using ATL. We will again use the train-crossing problem as an example, giving an informal meaning of the property we want to define and the corresponding ATL formula. Here $\langle\langle\rangle\rangle$ is an abbreviation for $\langle\langle\emptyset\rangle\rangle$. Note that all of these properties do indeed hold for our model from Figure 1.

1. Whenever the train is outside the gate and has not been granted permission to enter, then the controller can prevent it from entering:

$$\langle\langle\rangle\rangle\Box((out\_of\_gate \wedge \neg grant) \rightarrow \langle\langle ctr\rangle\rangle\Box out\_of\_gate)$$

2. Whenever the train is outside the gate, the controller cannot force it to enter:

$$\langle\langle\rangle\rangle\Box(out\_of\_gate \rightarrow [[ctr]]\Box out\_of\_gate)$$

3. Whenever the train is outside the gate, the train and the controller can cooperate so that the train will enter the gate:

$$\langle\langle\rangle\rangle\Box(out\_of\_gate \rightarrow \langle\langle train, ctr\rangle\rangle\Diamond in\_gate)$$

As one can easily see, the only way of describing the first two properties in CTL is

$$\forall\Box(out\_of\_gate \rightarrow \exists\Box out\_of\_gate)$$

From this formula we cannot deduce whether the train, or the controller, or the (possibly faulty) overall system keep the train from entering the gate. This gives an intuition as to why ATL is more expressive than CTL. With ATL we can quantify over the individual powers of one player or a cooperating team of players.

4 ATL Symbolic Model Checking

The *model-checking problem* for ATL is defined as follows:

– For a given game structure $S = (k, Q, \Pi, \pi, d, \delta)$, the set of players $\Sigma = \{1, \ldots, k\}$, and an ATL formula ϕ, compute the set $[\phi]_S$ of all states in S that satisfy ϕ.

We will introduce a symbolic algorithm $MC(S, \phi)$ as defined in [1] to solve the model-checking problem for structure S and formula ϕ. This algorithm makes use of the following functions:

– $Reg : \Pi \rightarrow 2^Q$, where $q \in Reg(p) \Leftrightarrow p \in \pi(q)$
– $Pre(2^\Sigma \times 2^Q)$, where $q \in Pre(A, \rho)$, iff from q the set A of players can cooperate and enforce the next state do be in ρ.

The algorithm itself is defined as follows:

```
– MC(S, φ):
  **foreach** subformula $\phi'$ of $\phi$ (in ascending order) **do**
    • **case** $\phi' = p$: **return** $Reg(\phi')$
    • **case** $\phi' = \neg\psi$: **return** $Q \setminus MC(S, \psi)$
    • **case** $\phi' = \psi_1 \vee \psi_2$: **return** $MC(S, \psi_1) \cup MC(S, \psi_2)$
    • **case** $\phi' = \langle\langle A\rangle\rangle \bigcirc \psi$: **return** $Pre(A, MC(S, \psi))$
    • **case** $\phi' = \langle\langle A\rangle\rangle\Box\psi$:
        ∗ $\rho := Q; \tau := MC(S, \psi)$
        ∗ **while** $\rho \not\subseteq \tau$ **do** $\rho := \tau; \tau := Pre(A, \rho) \cap MC(\psi)$;
        ∗ **return** $\rho$
    • **case** $\phi' = \langle\langle A\rangle\rangle\psi_1 \, \mathcal{U} \, \psi_2$:
        ∗ $\rho := \emptyset; \tau := MC(S, \psi_2)$
        ∗ **while** $\tau \not\subseteq \rho$ **do** $\rho := \rho \cup \tau; \tau := Pre(A, \rho) \cap MC(S, \psi_1)$;
        ∗ **return** $\rho$
```

A correctness proof of the algorithm and a detailed runtime analysis can be found in [1]. As can be seen there, the runtime of the algorithm is in $\mathcal{O}(m \cdot l)$, were l is defined as the length of the formula and m as the size of the structure ($m = |\delta|$). The outer loop of the algorithm clearly depends on l, and it can be shown that each of the cases lies in $\mathcal{O}(m)$.

5 Conclusion

We have shown how concurrent game structures can be used to model open systems. We have then specified a logic ATL over such models, which as we have seen in an example is more expressive than CTL (or LTL), because it can capture the powers of individual players or cooperating groups of players in a game. Furthermore we have shown how the model-checking problem for ATL can be solved, using a symbolic algorithm that was first defined in [1].

References

1. Rajeev Alur, Thomas Henzinger, and Orna Kupferman. Alternating-time temporal logic. Journal of the ACM 49: pages 672-713, 2002.
2. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. Proceedings of the International Workshop on Logic of Program, volume 131 of Lecture Notes in Computer Science: pages 52-71, 1981.
3. A. Pnuelli. The temporal logic of programs. Proceedings of the 18th International Symposium on Foundations of Computer Science: pages 46-57, 1977.