# Strategic Planning through Model Checking of ATL Formulae

Wojciech Jamroga

Parlevink Group, University of Twente, Netherlands
jamroga@cs.utwente.nl
http://www.cs.utwente.nl/~jamroga

**Abstract.** Model checking of temporal logic has already been proposed for automatic planning. In this paper, we introduce a simple adaptation of the ATL model checking algorithm that returns a strategy to achieve given goal. We point out that the algorithm generalizes minimaxing, and that ATL models generalize traditional game trees. The paper ends with suggestions about other game theory concepts that can be transfered to ATL-based planning.

Keywords: multi-agent systems, multi-agent planning, model checking, minimaxing.

## 1 Introduction

Logic-based approaches to Artificial Intelligence seem to be presently undervalued by most AI practitioners. This owes much to the fact that logic was believed to deliver the ultimate solution for all basic AI problems for a long time, and the disappointment which came after that. Indeed, it is hard to claim now that we can use logic (the way neural networks or genetic algorithms are used, for instance) to obtain agents that behave in a satisfying way. Despite recent development of logic-based tools for multi-agent systems, their applications restrict mainly to artificial "toy worlds", as opposed to the real world which is usually fuzzy, noisy and, most of all, hard to characterize with a simple mathematical model. However, we believe that mathematical logic – while probably not the best tool for engineering – should still be important in AI research for at least two reasons.

First, it provides us with a vocabulary for talking about systems, and gives the vocabulary precise meaning via models and semantic rules.[1] More importantly, mathematical models provide a conceptual apparatus for *thinking* about systems, that can be as well used outside mathematical logic. The second reason is that creating a formal model of a problem makes one realize many (otherwise

[1] We do not mean here the crispness of predicates themselves (which might be seen as a weakness as well, and certainly a reason why logic is less successful than soft computing methods in many areas), but rather the precision on the meta-level formulation of the language.

implicit) assumptions underlying his or her approach to this problem. The assumptions are often given a simplistic treatment in the model, (otherwise the models get too complex to be dealt with), yet their implications are usually worth investigating even in this form. Moreover, having made them explicit, one can strive to relax some of them and still use a part of the formal and conceptual machinery – instead of designing solutions completely ad hoc.

### 1.1 Model Checking

*Model checking* is an interesting idea that emerged from the research on logic in computer science. The model checking problem asks whether a particular formula $\varphi$ holds in a particular model $M$, which is often more interesting than *satisfiability checking* (i.e. looking for a model $M$ in which $\varphi$ holds) or *theorem proving* (i.e. proving that $\varphi$ follows from some set of axioms), simply because in many cases the designer can come up with a precise model of the system behavior (e.g. a graph with all the actions that may be effected). only the model is too large to check on the fly whether it fulfills the design objectives. This seems especially useful in the case of dynamic or temporal logics, whose models can be interpreted as game models, transition systems, control flow charts, data flow charts etc. Moreover, model checking turns out to be relatively cheap in computational terms. while satisfiability checking and theorem proving is often intractable or even undecidable.

It has been already proposed that the model checking of computation tree logic (CTL) formulae can be used for generating plans in deterministic as well as non-deterministic domains [6, 7]. Alternating-time temporal logic ATL is an extension of CTL that includes notions of agents, their abilities and strategies (conditional plans) explicitly in its models. Thus, ATL seems even better suited for planning, especially in multi-agent systems, which was already suggested in [8]. In this paper, we introduce a simple adaptation of the ATL model checking algorithm from [1] that – besides checking if a goal can be achieved – returns also an appropriate strategy to achieve it. We point out that this algorithm generalizes the well-known search algorithm of minimaxing, and that ATL models generalize turn-based transition trees from game theory. The paper ends with some suggestions that the contribution can be bilateral, and that more game theory concepts can contribute to modal logic-based models and algorithms for multi-agent systems.

## 2 Multi-Agent Planning through ATL Model Checking

*Alternating-time Temporal Logic* ATL [1] is an extension of CTL [3], and inherits from the latter several operators for describing temporal properties of systems: $\mathsf{A}$ (*for all paths*), $\mathsf{E}$ (*there is a path*), $\bigcirc$ (*at the next moment*), $\Diamond$ (*sometime*), $\Box$ (*always*) and $\mathcal{U}$ (*until*). Typically, paths are interpreted as sequences of successive states of computations. An example CTL model (transition system), together with the tree of possible computations, is displayed in Figure 1. A CTL formula

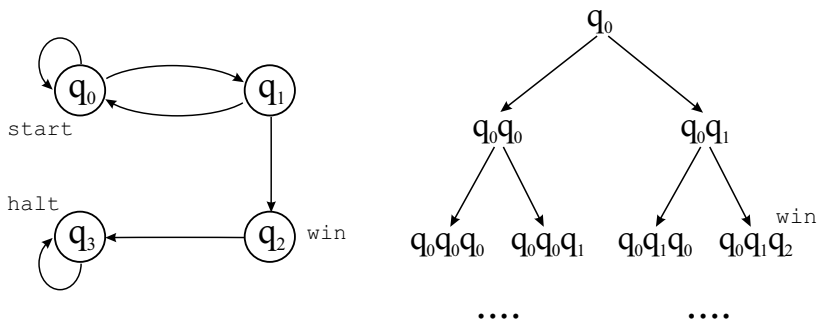


**Fig. 1.** Transition system and the tree of possible computations.

$\mathsf{A}\Diamond\mathtt{halt}$, for instance, expresses the property that the the system is bound to terminate (which is true if the initial state is either $q_2$ or $q_3$, but false for $q_1$ and $q_2$). Another formula, $\mathsf{E}(\neg\mathtt{halt})\mathcal{U}\mathtt{win}$, means that it is possible achieve a winning position before the system halts (which is true for all states except $q_4$).

ATL replaces $\mathsf{E}$ and $\mathsf{A}$ with a class of *cooperation modalities* $\langle\langle A\rangle\rangle$ (where $A$ is a group of agents). The common-sense reading of $\langle\langle A\rangle\rangle\Phi$ is: "*The group of agents $A$ have a collective strategy to enforce $\Phi$ regardless of what all the other agents do*". ATL models include a set of players $\Sigma$, a set of (global) system states $Q$, valuation of propositions $\pi$ (specifying which propositions are true in which states), and decisions available to every player at each particular state; finally, a complete tuple of decisions and a state imply a deterministic transition according to the transition function $\delta$. We will be writing $\delta(q, \sigma_A, \sigma_{\Sigma\setminus A})$ to denote the system transition from state $q$ when the agents from $A$ decide to proceed with (collective) action $\sigma_A$, and $\sigma_{\Sigma\setminus A}$ is the collective choice from their opponents.[2]

It is worth noting that the complexity of ATL model checking is linear ($O(nml)$, where $n$ is the number of states in the model, $m$ the number of transitions, and $l$ – length of the tested formula), so the checking should terminate in a sensible time even for *huge* models and formulae.

### 2.1 Planning Algorithm

In this section, a simple modification of the ATL model checking algorithm [1] is proposed, as shown in Figure 2. Function *pre* is used here to go "one step back" while constructing a plan for some coalition of agents. Thus, $pre(A, Q_1)$ takes as input a coalition $A$ and a set of states $Q_1 \subseteq Q$ and returns as output the set $Q_2$ of all states such that when the system is in one of the states from $Q_2$, the agents $A$ can cooperate and force the next state to be one of $Q_1$ (together with

[2] Note that – although the transitions must be deterministic in a concurrent game system – modeling nondeterminism is in fact easy. It suffices to add another player (*nature* or *environment*) and attribute our uncertainty about the outcome of agents' actions to decisions of this new player.

**function** $plan(\varphi)$**.**
Returns a subset of $Q$ for which formula $\varphi$ holds, together with a (conditional) plan to achieve $\varphi$. The plan is sought within the context of concurrent game structure $S = \langle \Sigma, Q, \Pi, \pi, \delta \rangle$.

**case** $\varphi \in \Pi$ **:** return $\{\langle q, - \rangle \mid \varphi \in \pi(q)\}$
**case** $\varphi = \neg\psi$ **:** $P_1 := plan(\psi)$; return $\{\langle q, - \rangle \mid q \notin states(P_1)\}$
**case** $\varphi = \psi_1 \vee \psi_2$ **:**
$P_1 := plan(\psi_1)$; $P_2 := plan(\psi_2)$;
return $\{\langle q, - \rangle \mid q \in states(P_1) \cup states(P_2)\}$
**case** $\varphi = \langle\langle A \rangle\rangle \bigcirc \psi$ **:** return $pre(A, states(plan(\psi)))$
**case** $\varphi = \langle\langle A \rangle\rangle \Box \psi$ **:**
$P_1 := plan(true)$; $P_2 := plan(\psi)$; $Q_3 := states(P_2)$;
**while** $states(P_1) \not\subseteq states(P_2)$
**do** $P_1 := P_2|_{states(P_1)}$; $P_2 := pre(A, states(P_1))|_{Q_3}$ **od**;
return $P_2|_{states(P_1)}$
**case** $\varphi = \langle\langle A \rangle\rangle \psi_1 \mathcal{U} \psi_2$ **:**
$P_1 := \emptyset$; $Q_3 := states(plan(\psi_1))$; $P_2 := plan(true)|_{states(plan(\psi_2))}$;
**while** $states(P_2) \not\subseteq states(P_1)$ **do** $P_1 := P_1 \oplus P_2$; $P_2 := pre(A, states(P_1))|_{Q_3}$ **od**;
return $P_1$
**end case**

**Fig. 2.** Adapted model checking algorithm for ATL formulae. Cases of $\psi_1 \vee \psi_2$ and $\langle\langle A \rangle\rangle \Diamond \psi$ are omitted, because the first can be re-written as $\neg(\neg\psi_1 \vee \neg\psi_2)$, and the latter as $\langle\langle A \rangle\rangle true \mathcal{U} \psi$.

$A$'s collective choices that accomplish this). Function $states(P)$ returns all the states for which plan $P$ is defined. $P_1 \oplus P_2$ refers to augmenting plan $P_1$ with all new subplans that can be found in $P_2$; finally $P|_{Q_1}$ denotes plan $P$ restricted to the states from $Q_1$ only. More formally:

- $pre(A, Q_1) = \{\langle q, \sigma_A \rangle \mid \forall \sigma_{\Sigma \setminus A} \delta(q, \sigma_A, \sigma_{\Sigma \setminus A}) \in Q_1\}$;
- $states(P) = \{q \in Q \mid \exists_\sigma \langle q, \sigma \rangle \in P\}$;
- $P_1 \oplus P_2 = P_1 \cup \{\langle q, \sigma \rangle \in P_2 \mid q \notin states(P_1)\}$;
- $P|_{Q_1} = \{\langle q, \sigma \rangle \in P \mid q \notin Q_1\}$.

Note that the algorithm returns a (non-empty) plan only if the outmost operator of the checked formula is a cooperation modality (i.e. it specifies explicitly *who* is to execute the plan and what is the objective). In consequence, our approach to negation is *not* constructive: for $\neg\langle\langle A \rangle\rangle \Phi$, the algorithm will not return a strategy for the rest of agents to actually avoid $\Phi$ (although $\neg\langle\langle A \rangle\rangle \Phi$ implies that such a strategy exists). Similar remark applies to alternative, conjunction, and nesting of strategic formulae. This approach is more natural than it seems at the first glance – even if the subformulae refer to the same set of agents for whom plans are needed. Consider, for instance, the transition system from Figure 1, and suppose that there is only one agent $a$ in the system, who executes the transitions. Formula $\langle\langle a \rangle\rangle \Box \texttt{start} \wedge \langle\langle a \rangle\rangle \Diamond \texttt{halt}$ is obviously true in $q_1$; however, it is hard to see what plan should be generated in this case. True, $a$ has a plan to

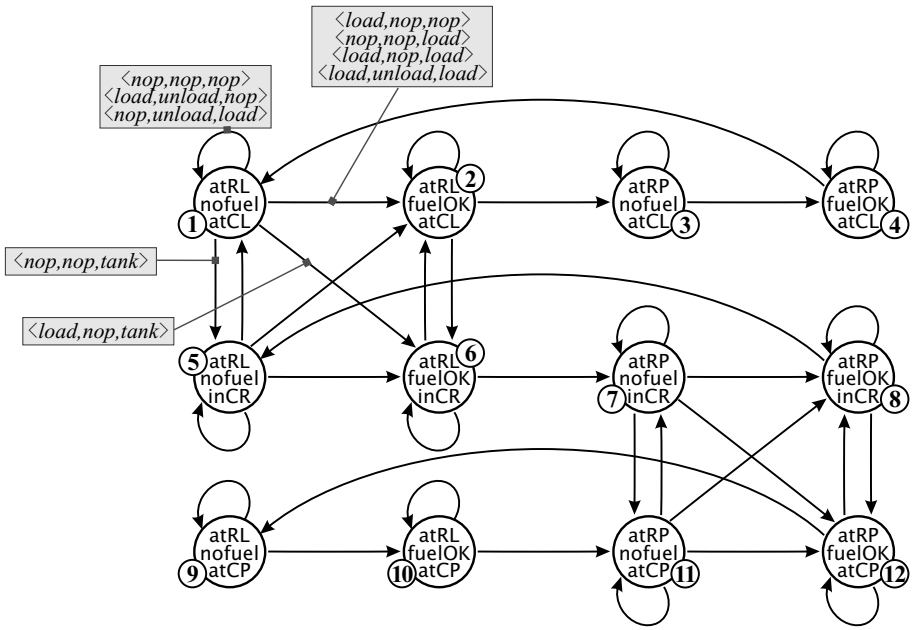


**Fig. 3.** A version of the Simple Rocket Domain. States of the system are labeled with natural numbers. All the transitions for state 1 (the cargo and the rocket are in London, no fuel in the rocket) are labeled. Output of agents' choices for other states is analogous.

remain in $q_1$ for ever, and he has a plan to halt the system eventually, but these are *different* plans and cannot be combined. Similarly, $\langle\langle a\rangle\rangle\Box\langle\langle a\rangle\rangle\Diamond\texttt{win}$ holds in $q_0$, but it does not mean that $a$ has a plan to win infinitely many times. He can always see a way to win; however, if he chooses that way, he will be unable to win again!

### 2.2 Rocket Example

As an example, consider a modified version of the Simple Rocket Domain from [2]. The task is to ensure that a cargo eventually arrives in Paris (proposition `atCP`); there are three agents with different capabilities who can be involved, and a single rocket that can be used to accomplish the task. Initially, the cargo may be in Paris, at the London airport (`atCL`) or it may lie inside the rocket (`inCR`). Accordingly, the rocket can be moved between London (`atRL`) and Paris (`atRP`).

There are three agents: $x$ who can load the cargo, unload it, or move the rocket; $y$ who can unload the cargo or move the rocket, and $z$ who can load the cargo or supply the rocket with fuel (action *fuel*). Every agent can also decide to do nothing at a particular moment (the *nop* – "no-operation" action). The agents act simultaneously. The "moving" action has the highest priority (so, if one agent tries to move the rocket and another one wants to, say, load the cargo,

then only the moving is executed). "Loading" is effected when the rocket does not move and more agents try to load than to unload. "Unloading" works in a similar way (in a sense, the agents "vote" whether the cargo should be loaded or unloaded). If the number of agents trying to load and unload is the same, then the cargo remains where it was. Finally, "fueling" can be accomplished alone or in parallel with loading or unloading. The rocket can move only if it has some fuel (`fuelOK`), and the fuel must be refilled after each flight. We assume that all the agents move with the rocket when it flies to another place. The concurrent game structure for the domain is shown in Figure 3.

$$plan(\langle\langle x\rangle\rangle\Diamond\texttt{atCP}) = \{\ \langle 9,-\rangle, \langle 10,-\rangle, \langle 11,-\rangle, \langle 12,-\rangle\ \} \quad (1)$$

$$plan(\langle\langle x,y\rangle\rangle\Diamond\texttt{atCP}) = \{\ \langle 2, x{:}load{\cdot}y{:}nop\rangle, \langle 6, x{:}move{\cdot}y{:}nop\rangle, \quad (2)$$
$$\langle 7, x{:}unload{\cdot}y{:}unload\rangle, \langle 8, x{:}unload{\cdot}y{:}unload\rangle,$$
$$\langle 9,-\rangle, \langle 10,-\rangle, \langle 11,-\rangle, \langle 12,-\rangle\ \}$$

$$plan(\langle\langle x,z\rangle\rangle\Diamond\texttt{atCP}) = \{\ \langle 1, x{:}load{\cdot}z{:}load\rangle, \langle 2, x{:}load{\cdot}z{:}load\rangle, \quad (3)$$
$$\langle 3, x{:}nop{\cdot}z{:}fuel\rangle, \langle 4, x{:}move{\cdot}z{:}nop\rangle,$$
$$\langle 5, x{:}load{\cdot}z{:}fuel\rangle, \langle 6, x{:}move{\cdot}z{:}nop\rangle,$$
$$\langle 7, x{:}unload{\cdot}z{:}nop\rangle, \langle 8, x{:}unload{\cdot}z{:}nop\rangle,$$
$$\langle 9,-\rangle, \langle 10,-\rangle, \langle 11,-\rangle, \langle 12,-\rangle\ \}$$

Plans to eventually achieve `atCP` – for $x$ alone, $x$ with $y$, and $x$ with $z$, respectively – are shown above. In the first case, $x$ cannot guarantee to deliver the cargo to Paris (unless the cargo already *is* there), because $y$ and $z$ may prevent him from unloading the goods (1). The coalition of $x$ and $y$ is more competent: they can, for instance, deliver the cargo from London if only there is fuel in the rocket (2). However, they have no infallible plan for the most natural case when 1 is the initial state. Finally, $\{x, z\}$ have an effective plan for any initial situation (3).

### 2.3 Minimaxing as Model Checking

It is easy to see that the algorithm from Figure 2 can be used for emulating the well known search algorithm of minimaxing. To find the best plan for $A$, we should label the final positions with the payoff values $p_1, p_2, ...$, then check which $plan(\langle\langle A\rangle\rangle\Diamond p_i)$ returns a decision for the initial state, and pick the one for maximal $p_i$. The resulting procedure is still linear in the number of states, transitions and different payoff values. Note that the algorithm proposed here is more general than the original minimaxing: the latter can be applied only to finite turn-based game trees (i.e. systems in which the number of states is finite, there are no cycles, and players cannot act simultaneously), while the model checking-based approach deals also with models in which players act in parallel, and with infinite trees that can be generated by a finite transition system.

Let us also observe that the planning algorithm, proposed in this paper, looks for a plan that must be successful against every line of events – hence the

algorithm generalizes minimaxing in zero-sum (i.e. strictly competitive) games. It can be interesting to model the non-competitive case within the scope of ATL as well: while checking $\langle\langle A\rangle\rangle\varphi$, the opponents $\Sigma \setminus A$ may be assumed different goals than just to prevent $A$ from achieving $\varphi$. Then, assuming optimal play from $\Sigma \setminus A$, we can ask whether $A$ have a strategy to enforce $\varphi$ provided that $\Sigma \setminus A$ intend (or desire) to bring about $\psi$.

## 3 Conclusions and Future Work

In this paper, a simple adaptation of ATL model checking from [1] is proposed. The algorithm proposed here looks for infallible conditional plans to achieve objectives that can be defined via ATL formulae. The algorithm generalizes minimaxing in zero-sum games, extending its scope to (possibly infinite) games in which the agents can act in parallel.

It seems that the link between model checking and minimaxing can be exploited to enrich the framework of ATL, too. First (as already mentioned in the previous section), ATL might be extended so that it can be used to model non-competitive games. Next, efficient pruning techniques exist for classical minimaxing – it may be interesting to transfer them to ATL model checking. Moreover, game theory has developed more sophisticated frameworks, like games with incomplete information and games with probabilistic outcomes (including temporal models, best defense criteria etc.). Investigation of similar concepts in the context of ATL can prove worthwhile, and lead to new research questions, concerning phenomena like non-locality [4] and design of efficient suboptimal algorithms [5] in the scope of logics for multi-agent systems.

## References


1. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49:672–713, 2002.
2. A. L. Blum and M. L. Furst. Fast planning through graph analysis. *Artificial Intelligence*, 90:281–300, 1997.
3. E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier, 1990.
4. I. Frank and D. Basin. Search in games with incomplete information : A case study using bridge card play. *Artificial Intelligence*, 100(1-2):87–123, 1998.
5. I. Frank, D.A. Basin, and H. Matsubara. Finding optimal strategies for imperfect information games. In *Proceedings of AAAI/IAAI*, pages 500–507, 1998.
6. F. Giunchiglia and P. Traverso. Planning as model checking. *ECP*, pp. 1–20, 1999.
7. M. Pistore and P. Traverso. Planning as model checking for extended goals in non-deterministic domains. In *Proceedings of IJCAI*, pages 479–486, 2001.
8. W. van der Hoek and M. Wooldridge. Tractable multiagent planning for epistemic goals. In *Proceedings of AAMAS-02*, pages 1167–1174. ACM Press, 2002.