TATL: Implementation of ATL Tableau-Based Decision Procedure

Amélie David

Laboratoire IBISC - Université d'Évry Val d'Essonne - EA 4526
23 bd de France - 91037 Évry Cedex - France
adavid@ibisc.univ-evry.fr

Abstract. This paper describes the implementation of a tableau-based decision procedure for the Alternating-time Temporal Logic proposed by Goranko and Shkatov in 2009, as well as a set of representative formulas used for testing.

Keywords: Alternating-time Temporal Logic, tableaux, theorem prover.

1 Introduction

The Alternating-time Temporal Logic (ATL) was introduced by Alur, Henzinger and Kupferman in 2002 [1] in order to formally specify and verify reactive multi-agent systems. Such systems are represented by a concurrent game structure $\mathcal{C}$, in short CGS, which are state-transition graphs. Transitions enable to go from one state of the system to another depending on the choices made by every agent of the system at a given state. At a given state, the different possible transitions (which may lead to the same successor state) are represented by move vectors.

In ATL, properties of such systems are expressed with formulas following the grammar:

$F := p \mid \neg F \mid (F_1 \wedge F_2) \mid (F_1 \vee F_2) \mid (F_1 \rightarrow F_2) \mid \langle\langle A\rangle\rangle \bigcirc F \mid \langle\langle A\rangle\rangle \Box F \mid \langle\langle A\rangle\rangle \Diamond F \mid \langle\langle A\rangle\rangle F_1 \mathcal{U} F_2,$

where p is a proposition and A is a coalition, that is, a set of agents. ATL-formulas represent objectives for the agents of the system $\mathcal{C}$ or the possibility of achieving objectives. The connectors $\neg$, $\wedge$, $\vee$ and $\rightarrow$ have the same meanings as in classical propositional logic. The operators $\bigcirc$, $\Box$, $\Diamond$ and $\mathcal{U}$ are those of temporal logics and mean *next time*, *always*, *eventually* and *until*, respectively. The novelty of ATL in comparison to other temporal logics is the use of agents' coalitions and strategies for these coalitions by using the path quantifier $\langle\langle A\rangle\rangle$. $\langle\langle A\rangle\rangle F$ means: *the coalition A has a strategy to achieve F*. So, for example, the formula $\langle\langle 1,2\rangle\rangle \Box(p \wedge q) \wedge \neg\langle\langle 2\rangle\rangle \Diamond p$ means that the coalition of agents 1 and 2 has a strategy to always achieve p and q, and the "coalition" with only agent 2 does not have a strategy to eventually achieve p.

Checking whether a formula is satisfiable, that is, checking whether a model exists for the formula or not, is a common question in logic. Methods to respond to this question were introduced in 2006 by Goranko and van Drimmelen using

D. Galmiche and D. Larchey-Wendling (Eds.): TABLEAUX 2013, LNCS 8123, pp. 97–103, 2013.

automata [2], and then in 2009 by Goranko and Shkatov using tableaux [3]. To our knowledge, none of them has been implemented. Therefore, in this paper, we present TATL, a prototype constituting the first implementation of Goranko and Shkatov's decision procedure. The core of TATL is implemented in Ocaml. TATL is available at `http://atila.ibisc.univ-evry.fr/tableau_ATL/`.

At first, we recall the tableau-based decision procedure of Goranko and Shkatov, then we present, in Section 3, the general principles of TATL's algorithm. In Section 4 we explain how to get and use TATL, and in Section 5 how TATL had been tested.

2 Tableau-Based Decision Procedure

Goranko and Shkatov's tableau method decides the satisfiability of an ATL formula θ by constructing a representation of models for θ, if a model exists. Construction of a tableau for an ATL formula consists of two phases: first, construct a pretableau and then obtain the tableau itself. The pretableau and the tableau are state-transition graphs. The pretableau contains two kinds of vertexes (states and prestates) and two kinds of edges (unmarked and marked transitions). Prestates and states are sets of ATL-formulas and marked transitions are labeled by sets of move vectors. Prestates and unmarked transitions are technical items ensuring the termination of the procedure and will not remain in the tableau. Intuitively, a prestate is an embryo of states that, when properly saturated, generates one or more states.

The first phase of the tableau construction uses two rules, **SR** and **Next** recursively applied on new sets of prestates or states. The construction phase starts with a set containing only the input formula as a prestate. The rule **SR** allows one to obtain states from prestates and the rule **Next** to obtain prestates from states. The rule **SR** decomposes each formula of a prestate Γ into primitive literal formulas, that is, formulas of the form $\top$,p,$\neg p$,$\langle\langle A\rangle\rangle\bigcirc F$ or $\neg\langle\langle B\rangle\rangle\bigcirc F$ ($B \neq \Sigma$, where Σ is the coalition of all agents). With the primitive formulas, we get all the information needed to verify the properties of the current state of a CGS and with the primitive next-time formulas $\langle\langle A\rangle\rangle\bigcirc F$ and $\neg\langle\langle B\rangle\rangle\bigcirc F$, all the information to verify the properties of next states in the CGS. From this decomposition, we obtain the states generated by Γ. Then, from each of these states, say Δ, the rule **Next** treats all Δ's next-time formulas in order to obtain a new set of prestates, the successors of Δ, which respects all the possible strategies of agents and their objectives. So the rule **SR** creates states and unmarked transitions, whereas the rule **Next** creates prestates and marked transitions. When a state or a prestate already exists, the rules **SR** and **Next** do not generate copies and only create a transition.

When the rules **SR** and **Next** cannot be applied any further, the obtained structure is the complete pretableau of the input formula and the elimination phase can start. The first elimination rule, **PR**, eliminates all prestates after having properly interconnected states. Then we apply the elimination rules **E1**, **E2**, and **E3**. The rule **E1** eliminates all states that contain an explicit inconsistency,

that is, any state containing formulas F and also $\neg F$, where F is a primitive formula. Then we apply rules **E3** and **E2** until no more states can be eliminated. The rule **E2** eliminates all states which have lost all their successors linked to the same move vector. Eventualities are formulas of the form a) $\langle\langle A\rangle\rangle\Diamond F_2$, b) $\langle\langle A\rangle\rangle F_1 \mathcal{U} F_2$ or c) $\neg\langle\langle A\rangle\rangle\Box F_3$. The rule **E3** eliminates all states not satisfying their eventualities, that is always postponing the realization of F_2 for a) and b), and the realization of F_3 for c).

At the end of the elimination phase, we obtain the tableau of the input formula. The tableau is open and the input formula is satisfiable if there remains at least one state containing the input formula; otherwise, the tableau is closed and the formula is unsatisfiable.

3 General Principles of TATL

The algorithm of TATL follows the steps described in the previous section. To represent states and prestates of the tableau and pretableau, that is, vertexes, we use a structure which allows us to stock information about the name and type of the vertex, the associated set of formulas, all the possible move vectors linked to that vertex, its successors, as well as an indicator about its consistency.

During the construction phase, we use four sets of vertexes: partial states, complete states, partial prestates and complete prestates. Partial prestates and partial states are waiting for application of the rule **SR** and the rule **Next**, respectively, to become complete prestates and complete states. The rule **SR** generates partial states and the rule **Next** generates partial prestates.

The most difficult rules to implement were the rules **SR** and **E3**, so we describe their implementation in more detail. Indeed, for the rule **SR**, we need to deal with decompositions where the operator *or* occurs. This decomposition generates several choices and therefore several successors for the same prestate. So we use a decomposition tree where all interior nodes still contain non-decomposed formulas and each leaf node contains a set of fully decomposed formulas. Each node of the tree is composed of two sets: one with decomposed formulas and primitives, and one with formulas that still need to be decomposed. For each interior node, we process one of the non-decomposed formulas to obtain successors. If a formula resulting from the decomposition is not primitive, it joins the set of non-decomposed formulas.

The rule **E3** needs, for each eventuality occurring in the tableau, to find all states containing that eventuality. This results in finding a path from a given state to a state satisfying the eventuality, avoiding looping indefinitely in a cycle. Let us call ξ the set of all states containing the eventuality. Then we separate the states of ξ which realize the eventuality from the others, using three sets: *to_be_treated*, *satisfied* and *current*. The procedure is iterative with the following halt conditions: the set *current* is empty or stable at the end of the iteration. At the beginning, all states ξ are placed in the set *to_be_treated*. For each state s in the set *to_be_treated*, we first check if the eventuality is immediately satisfied, that is, if s contains F_2 for an eventuality of the form $\langle\langle A\rangle\rangle\Diamond F_2$ or $\langle\langle A\rangle\rangle F_1 \mathcal{U} F_2$,

and $\neg F_3$ for an eventuality of the form $\neg\langle\langle A\rangle\rangle\Box F_3$. In that case, we move s to the set *satisfied*, otherwise, we check for each move vector leading to successors of s in ξ whether one of these successors is also in the set *satisfied*. If the eventuality is of the form $\langle\langle A\rangle\rangle F_1 \mathcal{U} F_2$, we also check that s contains F_1. If these conditions are satisfied then the state is moved to the set *satisfied*, otherwise it is moved to the set *current*. When the set *to_be_treated* is empty, if the set *current* is also empty or contains all the states to be treated at the beginning of the iteration, we return the set *current*, which contains all the states not satisfying the eventuality, otherwise we move the states of the set *current* to the set *to_be_treated* and the procedure is repeated.

4 Description of TATL

TATL was conceived as a web application in order to be multi-platform and easy to use, but binaries are also available. TATL is an Ocaml program and

Fig. 1. Screenshot of TATL's result page for options 1 and 2

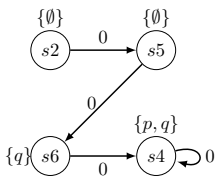
we use PHP for the graphical user interface of the web application. The web application and binaries are both available at `http://atila.ibisc.univ-evry.fr/tableau_ATL/`. In this paper, we will focus on the web application as the functionalities of binaries are similar. However, you can find details about the use of the binaries on the web page.

At the top of the page, a menu gives access to three options:

1. enter a formula as specified above;
2. choose among a set of preselected formulas corresponding to test cases (see Section 5);
3. use the random generator of formulas.

Option 1 (*One formula*) allows one to enter an ATL-formula and get in return the information about the satisfiability of the formula as well as the pretableau and the tableau, as shown in the screenshot in Fig. 1.

On the result area, each state (or prestate) is displayed with 4 elements: a name, a set of formulas, a mark and a set of successors. Prestates are referred to as Px and states as Sx where x is a number and the pretableau always begins with prestate P1. The check mark indicates that the state is consistent whereas the cross indicates that it is inconsistent. Successors of states are also given with their associated move vectors. An example of a move vector could be, for instance, $(0, 1, 0)$ for a formula with 3 agents. As agents are automatically sorted by their number, the first element of the move vector corresponds to the choice of the player with the smallest number. When a formula is satisfiable, this means that there exists at least one model for that formula. It is possible to manually construct a model from a tableau via the explanations in the completeness proof of [3]. For the tableau of Fig. 1, a model can be:



Option 2 (*Preselected formula*) allows one to select a formula among a set of 42 formulas and get in return the pretableau, the tableau, and the satisfiability of the formula, in the same way as in option 1. This set of formulas has been used to test the application (see Section 5).

Options 3 (*Random formulas*) allows one to randomly generate a set of ATL-formulas and get the answer on their satisfiability. This option needs some additional information to run: a set of propositions, a maximum number of agents from which TATL creates effective agents, a number of formulas to generate, a maximal depth of formulas and a time-out in seconds to stop the computation when it takes too long. The screenshot in Fig. 2 shows the output for this option. A check mark indicates that the formula is satisfiable, a cross that the formula is unsatisfiable and a question mark that the computation has timed-out. Clicking on the box in front of each generated formula transforms the syntax of the

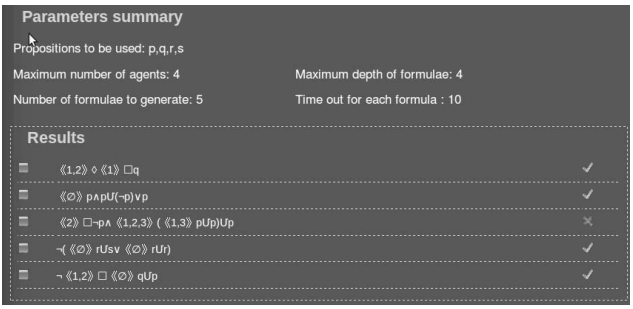


Fig. 2. Example of results obtained with the option "random formulas"

formula to make it compatible with option "One formula", thereby getting the pretableau and tableau of the formula.

5 Tests for TATL

A common way to test an implementation is to compare the outputs against another implementation. But, to our knowledge, there are no available tools to decide the satisfiability of an ATL-formula, either by using tableaux or by using automata. So we decided to check that TATL works correctly by creating a set of ATL-formulas that enables us to test each part of the algorithm. It should be noted that, to our knowledge, a benchmark set does not exist for ATL-satisfiability, thus our set of formulas might be used as a starting set for more refined future benchmarks. Our set consists of 42 formulas and allows us to test 50 points distributed in 15 categories, which are:

Coalition screen output	Next-time formulas
recognition of formulas	Cartesian product for move vectors
Treatment of agents	formula decomposition
Primitives	Rule E2
Inconsistency(Rule E1)	Rule E3
Eventualities	Creation of state/prestates sets
Move vectors	Several eventualities
sorting of Next-time formulas	

The set of formulas is provided in the appendix. Details of the 50 points and of the 15 categories can be found at `http://atila.ibisc.univ-evry.fr/tableau_ATL/test_cases.ods`. For instance, the formulas 23, 24 and 25 have been conceived to test eventualities, whereas formulas 16 and 17 allowed us to test the rule **E2**.

In order to test our implementation, we manually calculated a tableau for each of the 42 formulas and compared our results with TATL results to ensure that both satisfiability outputs and tableau's descriptions comply with the specification for these test cases.

6 Conclusion and Perspectives

TATL is, to our knowledge, the first implementation for testing the satisfiability of an ATL-formula. TATL is multi-platform and easy to use thanks to its web interface. TATL is also available as a command line application. As no reference tools were available for testing, we also provide a set of ATL-formulas. Such a set can be reused to develop more sophisticated benchmarks for ATL. TATL is a prototype so we need to improve it, for example, using better data structures to save computation time. We also worked on automata's construction based on tableaux, so we plan to add this functionality to TATL, by adding automata construction.

Acknowledgment. The author would like to thank for their help and support Serenella Cerrito, Marta Cialdea Mayer, Valentin Goranko and Laurent Poligny.

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM 49(5), 672–713 (2002)
2. Goranko, V., van Drimmelen, G.: Complete axiomatization and decidability of Alternating-time temporal logic. Theor. Comput. Sci. 353(1-3), 93–117 (2006)
3. Goranko, V., Shkatov, D.: Tableau-based decision procedures for logics of strategic ability in multi-agent systems. ACM Trans. Comput. Log. 11(1) (2009)

List of the 42 Formulas

1 p	5 $\langle\langle 1\rangle\rangle \bigcirc p$	9 $\neg\langle\langle 1\rangle\rangle pUq$	11 $\langle\langle 1,2\rangle\rangle p\mathcal{U}q \wedge \langle\langle 1,2\rangle\rangle \bigcirc r$
2 $p \wedge q$	6 $\langle\langle 1\rangle\rangle \Diamond p$	10 $\neg\langle\langle 1\rangle\rangle \Diamond p$	12 $\langle\langle 1,2\rangle\rangle p\mathcal{U}q \wedge \langle\langle 3,4\rangle\rangle \bigcirc r$
3 $p \vee q$	7 $\langle\langle 1\rangle\rangle \Box p$	18 $\neg p \vee \langle\langle 1\rangle\rangle \Diamond p$	13 $\langle\langle 1,2\rangle\rangle p\mathcal{U}q \wedge \langle\langle 2,3\rangle\rangle \bigcirc r$
4 $p \rightarrow q$	8 $\langle\langle 1\rangle\rangle p\mathcal{U}q$	19 $p \wedge \neg p$	14 $\langle\langle 2,1\rangle\rangle p\mathcal{U}q \wedge \langle\langle 3,2\rangle\rangle \bigcirc r$

15 $\langle\langle\rangle\rangle p\mathcal{U}q \wedge \langle\langle 1,2\rangle\rangle \bigcirc r$	23 $\langle\langle 1\rangle\rangle p\mathcal{U}q \vee \neg\langle\langle 1\rangle\rangle \Box q$
16 $\neg\langle\langle 1,2\rangle\rangle \bigcirc p \wedge \langle\langle 1\rangle\rangle \Box p$	24 $\langle\langle 1,2\rangle\rangle p\mathcal{U}(\neg\langle\langle 1\rangle\rangle \Box p)$
17 $\neg\langle\langle 1,2\rangle\rangle \bigcirc p \wedge \langle\langle 1,2,3\rangle\rangle \Box p$	25 $\langle\langle 1\rangle\rangle(\neg\langle\langle 1,2\rangle\rangle \Box p)Uq$
20 $(p \wedge q) \wedge \langle\langle 1\rangle\rangle \Box\neg(p \wedge q)$	26 $\langle\langle\rangle\rangle \Box\langle\langle\rangle\rangle p\mathcal{U}q$
21 $\langle\langle 1\rangle\rangle \Box p \wedge \neg\langle\langle 2\rangle\rangle \Diamond\langle\langle 1\rangle\rangle \Box p$	27 $\neg\langle\langle 1\rangle\rangle \Box p \wedge \langle\langle 1,2\rangle\rangle \bigcirc p \wedge \neg\langle\langle 2\rangle\rangle \bigcirc\neg p$
22 $\langle\langle 1\rangle\rangle \bigcirc p \wedge \neg\langle\langle 1\rangle\rangle \bigcirc p$	31 $\langle\langle 1,2,3\rangle\rangle \Box\langle\langle 2,3,4\rangle\rangle \Box(p \wedge q)$

33 $\neg\neg\langle\langle 1\rangle\rangle pUq$	38 $\langle\langle 1\rangle\rangle \Box p \wedge \neg\langle\langle 1,2\rangle\rangle \Box p$
34 $\neg(\langle\langle 1\rangle\rangle \Box p \vee \langle\langle 1\rangle\rangle \Box\neg p)$	39 $\neg\langle\langle 1\rangle\rangle \bigcirc p \wedge \langle\langle 2\rangle\rangle \bigcirc\neg p$
35 $\neg(\langle\langle 1\rangle\rangle \Box p \wedge \langle\langle 1\rangle\rangle \Box\neg p)$	40 $\langle\langle 1\rangle\rangle \bigcirc p \wedge \langle\langle 2\rangle\rangle \bigcirc\neg p$
36 $\neg\langle\langle 1\rangle\rangle p\mathcal{U}\neg\langle\langle 2\rangle\rangle q\mathcal{U}r$	41 $\langle\langle 1\rangle\rangle pUq \wedge \langle\langle 2\rangle\rangle q\mathcal{U}r \wedge \langle\langle 2\rangle\rangle \Box\neg r$
37 $\langle\langle 1\rangle\rangle \Box\neg q \wedge \langle\langle 2\rangle\rangle pUq$	42 $\langle\langle 1\rangle\rangle pUq \wedge \langle\langle 2\rangle\rangle q\mathcal{U}r \wedge \langle\langle 1\rangle\rangle \Box\neg r$

28 $\langle\langle 1\rangle\rangle \bigcirc p \wedge \langle\langle 2\rangle\rangle \bigcirc q \wedge \langle\langle 1,2\rangle\rangle \bigcirc r \wedge \neg\langle\langle 1\rangle\rangle \bigcirc r \wedge \neg\langle\langle 3\rangle\rangle \bigcirc q$
29 $\neg\langle\langle 1\rangle\rangle \bigcirc r \wedge \neg\langle\langle 3\rangle\rangle \bigcirc q \wedge \langle\langle 1\rangle\rangle \bigcirc p \wedge \langle\langle 2\rangle\rangle \bigcirc q \wedge \langle\langle 1,2\rangle\rangle \bigcirc r$
30 $\neg\langle\langle 1\rangle\rangle \bigcirc r \wedge \langle\langle 1\rangle\rangle \bigcirc p \wedge \langle\langle 2\rangle\rangle \bigcirc q \wedge \neg\langle\langle 3\rangle\rangle \bigcirc q \wedge \langle\langle 1,2\rangle\rangle \bigcirc r$
32 $\langle\langle 1,2,3\rangle\rangle \Box\langle\langle 2,3\rangle\rangle \Box(p \wedge q) \wedge \langle\langle 4\rangle\rangle \bigcirc\neg p$