# Tableau-Based Automata Construction for Dynamic Linear Time Temporal Logic*

Laura Giordano
Università del Piemonte Orientale
Alessandria, Italy
laura@mfn.unipmn.it

Alberto Martelli
Università di Torino
Torino, Italy
mrt@di.unito.it

**Abstract**

We present a tableau-based algorithm for obtaining a Büchi automaton from a formula in Dynamic Linear Time Temporal Logic (DLTL), a logic which extends LTL by indexing the until operator with regular programs. The construction of the states of the automaton is similar to the standard construction for LTL, but a different technique must be used to verify the fulfillment of until formulas. The resulting automaton is a Büchi automaton rather than a generalized one. The construction can be done on-the-fly, while checking for the emptiness of the automaton. We also extend the construction to the Product Version of DLTL.

**Keywords**: Temporal logic, model checking.
**AMS subject classification**: 03B44, 68N30

## 1 Introduction

The problem of constructing automata from Linear-Time Temporal (LTL) formulas has been deeply studied [24]. The interest on this problem comes from the wide use of temporal logic for the verification of properties of concurrent systems. The standard approach to LTL model checking consists of translating the negation of a given LTL formula (property) into a Büchi automaton, and checking the product of the property automaton and the model for language emptiness. Therefore it is essential to keep the size of the automaton as small as possible. A tableau-based algorithm for efficiently constructing a Büchi automaton is presented in [4]. This algorithm allows to build the graph "on the fly" and in most cases builds quite small automata, although the problem is intrinsically exponential. Further improvements have been presented in [3, 19].

Temporal logics are widely used in the specification and verification of distributed systems and they have recently gained attention in the area of reasoning

*This research has been partially supported by the project MIUR PRIN 2003 "Logic-based development and verification of multi-agent systems".

about actions and planning [1, 8, 14, 2], as well as in the specification and in the modeling and verification of multi-agent systems [22, 13, 20].

Dynamic Linear Time Temporal Logic (DLTL) [10] extends LTL by indexing the until operator with programs in Propositional Dynamic Logic, and has been shown to be strictly more expressive than LTL [10]. In [5, 6, 7] we have developed an action theory based on DLTL and of its product version [9], and we have shown how to use it in the specification and verification of multi-agent systems. This motivates the interest in developing efficient techniques for translating formulas into automata.

In [10] it is shown that the satisfiability problem for DLTL can be solved in exponential time, by reducing it to the emptiness problem for Büchi automata. The construction presented there requires to build an automaton with an exponential number of states, most of which will not be reachable from the initial state, and thus is not suitable for practical implementation.

In this paper we present a tableau-based algorithm for constructing a Büchi automaton from a DLTL formula, which adapts the approach of [4] to DLTL. Although the worst case remains exponential, this construction usually achieves a substantial reduction in the number of generated nodes. Furthermore it allows to build the automaton *on-the-fly* while checking for emptiness. The construction of the states of the automaton is similar to the standard construction for LTL [4], but the possibility of indexing until formulas with regular programs puts stronger constraints on the fulfillment of until formulas than in LTL, requiring more complex acceptance conditions. Thus we extend the structure of graph nodes and the acceptance conditions by adapting a technique proposed in [10]. The resulting automaton will be a Büchi automaton instead of a generalized Büchi automaton as in [4].

The schedule of the paper is the following. Section 2 introduces the logic DLTL. Section 3 describes the tableau-based method for constructing a Büchi automaton for a DLTL formula, proves the correctness of the method and shows that the satisfiability problem for DLTL is PSPACE complete. Section 4 introduce the product version for DLTL, $\mathrm{DLTL}^{\otimes}(\tilde{\Sigma})$, and Section 5 describes the automaton construction for $\mathrm{DLTL}^{\otimes}(\tilde{\Sigma})$. Section 6 shortly describes how the logic can be used in reasoning about actions and in the verification of systems of communicating agents. Section 7 contains related work and Section 8 concludes the paper.

## 2 Dynamic Linear Time Temporal Logic

In this section we define the syntax and semantics of DLTL as introduced in [10]. In such a linear time temporal logic the next state modality is indexed by actions. Moreover, (and this is the extension to LTL) the until operator is indexed by programs in Propositional Dynamic Logic (PDL).

Let $\Sigma$ be a finite non-empty alphabet. The members of $\Sigma$ are actions. Let $\Sigma^*$ and $\Sigma^\omega$ be the set of finite and infinite words on $\Sigma$, where $\omega = \{0, 1, 2, \ldots\}$. Let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. We denote by $\sigma, \sigma'$ the words over $\Sigma^\omega$ and by $\tau, \tau'$ the words

over $\Sigma^*$. Moreover, we denote by $\leq$ the usual prefix ordering over $\Sigma^*$ and, for $u \in \Sigma^\infty$, we denote by *prf(u)* the set of finite prefixes of $u$.

We define the set of programs (regular expressions) $Prg(\Sigma)$ generated by $\Sigma$ as follows:

$$Prg(\Sigma) ::= a \mid \pi_1 + \pi_2 \mid \pi_1; \pi_2 \mid \pi^*$$

where $a \in \Sigma$ and $\pi_1, \pi_2, \pi$ range over $Prg(\Sigma)$. A set of finite words is associated with each program by the mapping $[[]] : Prg(\Sigma) \to 2^{\Sigma^*}$, which is defined as follows:

- $[[a]] = \{a\}$;
- $[[\pi_1 + \pi_2]] = [[\pi_1]] \cup [[\pi_2]]$;
- $[[\pi_1; \pi_2]] = \{\tau_1\tau_2 \mid \tau_1 \in [[\pi_1]] \text{ and } \tau_2 \in [[\pi_2]]\}$;
- $[[\pi^*]] = \bigcup [[\pi^i]]$, where
  - $[[\pi^0]] = \{\varepsilon\}$
  - $[[\pi^{i+1}]] = \{\tau_1\tau_2 \mid \tau_1 \in [[\pi]] \text{ and } \tau_2 \in [[\pi^i]]\}$, for every $i \in \omega$.

Let $\mathcal{P} = \{p_1, p_2, \ldots\}$ be a countable set of atomic propositions. The set of formulas of DLTL($\Sigma$) is defined as follows:

$$\text{DLTL}(\Sigma) ::= p \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha \mathcal{U}^\pi \beta$$

where $p \in \mathcal{P}$, $\pi \in Prg(\Sigma)$ and $\alpha, \beta$ range over DLTL($\Sigma$).

A model of DLTL($\Sigma$) is a pair $M = (\sigma, V)$ where $\sigma \in \Sigma^\omega$ and $V : prf(\sigma) \to 2^{\mathcal{P}}$ is a valuation function. Given a model $M = (\sigma, V)$, a finite word $\tau \in prf(\sigma)$ and a formula $\alpha$, the satisfiability of a formula $\alpha$ at $\tau$ in $M$, written $M, \tau \models \alpha$, is defined as follows:

- $M, \tau \models p$ iff $p \in V(\tau)$;
- $M, \tau \models \neg\alpha$ iff $M, \tau \not\models \alpha$;
- $M, \tau \models \alpha \vee \beta$ iff $M, \tau \models \alpha$ or $M, \tau \models \beta$;
- $M, \tau \models \alpha \mathcal{U}^\pi \beta$ iff there exists $\tau' \in [[\pi]]$ such that $\tau\tau' \in prf(\sigma)$ and $M, \tau\tau' \models \beta$. Moreover, for every $\tau''$ such that $\varepsilon \leq \tau'' < \tau'$[1], $M, \tau\tau'' \models \alpha$.

A formula $\alpha$ is satisfiable iff there is a model $M = (\sigma, V)$ and a finite word $\tau \in prf(\sigma)$ such that $M, \tau \models \alpha$.

The formula $\alpha \mathcal{U}^\pi \beta$ is true at $\tau$ if "$\alpha$ until $\beta$" is true on a finite stretch of behavior which is in the linear time behavior of the program $\pi$.

The symbols $\top$ and $\bot$ can be defined as: $\top \equiv p \vee \neg p$ and $\bot \equiv \neg\top$. The derived modalities $\langle\pi\rangle$ and $[\pi]$ can be defined as follows: $\langle\pi\rangle\alpha \equiv \top\mathcal{U}^\pi\alpha$ and $[\pi]\alpha \equiv \neg\langle\pi\rangle\neg\alpha$. Furthermore, if we let $\Sigma = \{a_1, \ldots, a_n\}$, the $\mathcal{U}$, $\bigcirc$ (next), $\Diamond$ and $\Box$ of LTL can be defined as follows:

[1] We define $\tau \leq \tau'$ iff $\exists\tau''$ such that $\tau\tau'' = \tau'$. Moreover, $\tau < \tau'$ iff $\tau \leq \tau'$ and $\tau \neq \tau'$.

$\bigcirc\alpha \equiv \bigvee_{a\in\Sigma}\langle a\rangle\alpha$,
$\alpha\mathcal{U}\beta \equiv \alpha\mathcal{U}^{\Sigma^*}\beta$,
$\Diamond\alpha \equiv \top\mathcal{U}\alpha$,
$\Box\alpha \equiv \neg\Diamond\neg\alpha$,
where, in $\mathcal{U}^{\Sigma^*}$, $\Sigma$ is taken to be a shorthand for the program $a_1 + \ldots + a_n$. Hence both LTL and PDL are fragments of DLTL. As shown in [10], DLTL is strictly more expressive than LTL. In fact, as the logic ETL [23] to which DLTL is inspired, DLTL has the full expressive power of the monadic second order theory of $\omega$-sequences.

# 3 Automaton Construction

In this section we show how to build a Büchi automaton for a given DLTL formula $\phi$ using a tableau-like procedure. First we recall the definition of Büchi automaton.

A *Büchi automaton* over an alphabet $\Sigma$ is a tuple $\mathcal{B} = (Q, \rightarrow, Q_{in}, F)$ where:

- $Q$ is a finite nonempty set of states;
- $\rightarrow \subseteq Q \times \Sigma \times Q$ is a transition relation;
- $Q_{in} \subseteq Q$ is the set of initial states;
- $F \subseteq Q$ is a set of accepting states.

Let $\sigma \in \Sigma^\omega$. Then a *run* of $\mathcal{B}$ over $\sigma$ is a map $\rho : prf(\sigma) \rightarrow Q$ such that:

- $\rho(\varepsilon) \in Q_{in}$
- $\rho(\tau) \xrightarrow{a} \rho(\tau a)$ for each $\tau a \in prf(\sigma)$ with $a \in \Sigma$.

The run $\rho$ is *accepting* iff $inf(\rho) \cap F \neq \emptyset$, where $inf(\rho) \subseteq Q$ is given by: $q \in inf(\rho)$ iff $\rho(\tau) = q$ for infinitely many $\tau \in prf(\sigma)$. Finally $\mathcal{L}(\mathcal{B})$, the language of $\omega$-words accepted by $\mathcal{B}$, is: $\mathcal{L}(\mathcal{B}) = \{\sigma | \exists$ an accepting run of $\mathcal{B}$ over $\sigma\}$.

Our aim is now to construct a Büchi automaton for a given DLTL formula $\phi$.

In our construction we will make use of a reformulation of the following axioms of DLTL in [10][2]:

$(A1)$ $\bigvee_{a\in\Sigma}\langle a\rangle\top$

$(A2)$ $\alpha\mathcal{U}^\pi\beta \equiv (\beta \vee (\alpha \wedge \bigvee_{a\in\Sigma}\langle a\rangle \bigvee_{\pi'\in\delta_a(\pi)} \alpha\mathcal{U}^{\pi'}\beta))$, for $\varepsilon \in [[\pi]]$,

$(A3)$ $\alpha\mathcal{U}^\pi\beta \equiv \alpha \wedge \bigvee_{a\in\Sigma}\langle a\rangle \bigvee_{\pi'\in\delta_a(\pi)} \alpha\mathcal{U}^{\pi'}\beta$, for $\varepsilon \notin [[\pi]]$,

where $\delta_a(\pi) = \{\pi' | \pi \xrightarrow{a} \pi'\}$ and $\xrightarrow{a}$ is a transition relation (defined in [10]) such that the program $\pi'$ is obtained from the program $\pi$ by executing action $a$.

In our construction, we exploit the equivalence results between regular expressions and finite automata and we make use of an equivalent formulation

[2]Remember that $\langle a\rangle\alpha \equiv \top\mathcal{U}^a\alpha$.

of DLTL formulas in which "until" formulas are indexed with finite automata rather than regular expressions. Thus we have $\alpha \mathcal{U}^{\mathcal{A}} \beta$ instead of $\alpha \mathcal{U}^{\pi} \beta$, where $\mathcal{L}(\mathcal{A}) = [[\pi]]$.

Satisfiability of until formulas $\alpha \mathcal{U}^{\mathcal{A}} \beta$ must be modified accordingly by replacing $[[\pi]]$ with $\mathcal{L}(\mathcal{A})$ in the definition above[3].

More precisely, in the construction we will make use of the following notation for automata. Let $\mathcal{A} = (Q, \delta, Q_F)$ be an $\epsilon$-free nondeterministic finite automaton over the alphabet $\Sigma$ without an initial state, where $Q$ is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, and $Q_F$ is the set of final states. Given a state $q \in Q$, we denote with $\mathcal{A}(q)$ an automaton $\mathcal{A}$ with initial state $q$.

The two axioms A2 and A3 above will thus be restated as follows:

$(A2')$ $\alpha \mathcal{U}^{\mathcal{A}(q)} \beta \equiv (\beta \vee (\alpha \wedge \bigvee_{a \in \Sigma} \langle a \rangle \bigvee_{q' \in \delta(q,a)} \alpha \mathcal{U}^{\mathcal{A}(q')} \beta))$ ($q$ is a final state)

$(A3')$ $\alpha \mathcal{U}^{\mathcal{A}(q)} \beta \equiv \alpha \wedge \bigvee_{a \in \Sigma} \langle a \rangle \bigvee_{q' \in \delta(q,a)} \alpha \mathcal{U}^{\mathcal{A}(q')} \beta$ ($q$ is not a final state)

These formulas can be easily proved to be valid. Observe that the disjunction $\bigvee_{q' \in \delta(q,a)} \alpha \mathcal{U}^{\mathcal{A}(q')} \beta$ is a finite disjunction, as the set of states $q'$ in $\delta(q, a)$ is finite.

The main procedure to construct the Büchi automaton for a formula $\phi$ builds a graph $\mathcal{G}(\phi)$ whose nodes are labelled by sets of formulas, and whose edges are labelled by symbols from the alphabet $\Sigma$. States and transitions of the Büchi automaton are obtained directly form the nodes and edges of the graph. The procedure makes use of an auxiliary tableau-based function which is described in the next section.

## 3.1 Tableau computation

The tableau procedure makes use of *signed formulas*, i.e. formulas prefixed with the symbol **T** or **F**. This procedure takes as input a set of formulas[4] and returns a set of sets of formulas, obtained by expanding the input set according to a set of tableau rules, formulated as follows:

> $\phi \Rightarrow \psi_1, \psi_2$, if $\phi$ belongs to the set of formulas, then add $\psi_1$ and $\psi_2$ to the set
>
> $\phi \Rightarrow \psi_1 | \psi_2$, if $\phi$ belongs to the set of formulas, then make two copies of the set and add $\psi_1$ to one of them and $\psi_2$ to the other one.

The rules are the following:

| | |
|---|---|
| Tand: | $\mathbf{T}(\alpha \wedge \beta) \Rightarrow \mathbf{T}\alpha, \mathbf{T}\beta$ |
| Fand: | $\mathbf{F}(\alpha \wedge \beta) \Rightarrow \mathbf{F}\alpha \vert \mathbf{F}\beta$ |
| Tor: | $\mathbf{T}(\alpha \vee \beta) \Rightarrow \mathbf{T}\alpha \vert \mathbf{T}\beta$ |
| For: | $\mathbf{F}(\alpha \vee \beta) \Rightarrow \mathbf{F}\alpha, \mathbf{F}\beta$ |
| Tneg: | $\mathbf{T}\neg\alpha \Rightarrow \mathbf{F}\alpha$ |

[3] The idea of using finite state automata to label "until" formulas is inspired both by the automata construction for DLTL in [10] and by the automata construction for ETL in [21].

[4] In this section we will always refer to signed formulas

| | |
|---|---|
| Fneg: | $\mathbf{F}\neg\alpha \Rightarrow \mathbf{T}\alpha$ |
| TuntilFS: | $\mathbf{T}\alpha\mathcal{U}^{\mathcal{A}(q)}\beta \Rightarrow \mathbf{T}(\beta \vee (\alpha \wedge \bigvee_{a\in\Sigma}\langle a\rangle \bigvee_{q'\in\delta(q,a)} \alpha\mathcal{U}^{\mathcal{A}(q')}\beta))$ ($q$ is a final state) |
| TuntilNFS: | $\mathbf{T}\alpha\mathcal{U}^{\mathcal{A}(q)}\beta \Rightarrow \mathbf{T}(\alpha \wedge \bigvee_{a\in\Sigma}\langle a\rangle \bigvee_{q'\in\delta(q,a)} \alpha\mathcal{U}^{\mathcal{A}(q')}\beta)$ ($q$ is not a final state) |
| FuntilFS: | $\mathbf{F}\alpha\mathcal{U}^{\mathcal{A}(q)}\beta \Rightarrow \mathbf{F}(\beta \vee (\alpha \wedge \bigvee_{a\in\Sigma}\langle a\rangle \bigvee_{q'\in\delta(q,a)} \alpha\mathcal{U}^{\mathcal{A}(q')}\beta))$ ($q$ is a final state) |
| FuntilNFS: | $\mathbf{F}\alpha\mathcal{U}^{\mathcal{A}(q)}\beta \Rightarrow \mathbf{F}(\alpha \wedge \bigvee_{a\in\Sigma}\langle a\rangle \bigvee_{q'\in\delta(q,a)} \alpha\mathcal{U}^{\mathcal{A}(q')}\beta)$ ($q$ is not a final state) |

Here we use the $\wedge$ boolean operator and the modality $\langle a\rangle$ as primitive operators, although they can be derived from the operators given in the definition of DLTL.

Let us call *elementary formulas* the formulas to which the above rules cannot be applied, that is signed formulas $\mathbf{T}\phi$ or $\mathbf{F}\phi$ where $\phi$ is either $\top$, or $\bot$, or a proposition or $\langle a\rangle\alpha$.

Given a set of formulas $s$, function *tableau(s)* works as follows:

- add $\mathbf{T}\bigvee_{a\in\Sigma}\langle a\rangle\top$ to $s$,
- repeatedly apply the above rules to the formulas of $s$ (by possibly creating new sets) until all non-elementary formulas in all sets have been expanded,
- return the resulting set of sets.

Formula $\mathbf{T}\bigvee_{a\in\Sigma}\langle a\rangle\top$ makes explicit that in DLTL each state must be followed by a next state (as stated by axiom (A1)).

If the expansion of a set of formulas produces an inconsistent set, then this set is deleted (*consistency constraint*). A set of formulas s is inconsistent in the following cases:

- $\mathbf{T}\bot \in s$
- $\mathbf{F}\top \in s$
- $\mathbf{T}\alpha \in s$ and $\mathbf{F}\alpha \in s$
- $\mathbf{T}\langle a\rangle\alpha \in s$ and $\mathbf{T}\langle b\rangle\beta \in s$ with $a \neq b$".

The last constraint comes from the fact that DLTL is a linear time logic and thus two different actions cannot be executed in the same state.

**Example 1** Let us consider for instance the formula $\Box\langle\mathcal{A}(s_1)\rangle p$, with $\Sigma = \{a\}$, where the automaton $\mathcal{A}$ is given in Figure 1(a). By eliminating the derived modalities, this formula can be rewritten as the formula $\neg(\top\mathcal{U}\neg(\top\mathcal{U}^{\mathcal{A}(s_1)}p))$ and, by the definition of the until $\mathcal{U}$ as $\mathcal{U}^{\Sigma^*}$ and the fact that $\Sigma^*$ is the program $a^*$, it can be rewritten as $\neg(\top\mathcal{U}^{\mathcal{A}_1(s_0)}\neg(\top\mathcal{U}^{\mathcal{A}(s_1)}p))$, where the automaton $\mathcal{A}_1$ has only one (final) state $s_0$ connected to itself through a transition labelled $a$. Let us apply the function tableau to the initial set of signed formulas $\{\mathbf{T}\neg(\top\mathcal{U}^{\mathcal{A}_1(s_0)}\neg(\top\mathcal{U}^{\mathcal{A}(s_1)}p))\}$. It introduces the following formulas:

F1: $\mathbf{T}\langle a\rangle\top$ (initial step)

F2: $\mathbf{F}(\top\mathcal{U}^{\mathcal{A}_1(s_0)}\neg(\top\mathcal{U}^{\mathcal{A}(s_1)}p))$,

by applying the rule (Tneg) to the initial signed formula. Then, by applying rule (FuntilFS) to F2, we get

F3: $\mathbf{F}(\neg(\top\mathcal{U}^{\mathcal{A}(s_1)}p)\vee(\top\wedge\langle a\rangle(\top\mathcal{U}^{\mathcal{A}_1(s_0)}\neg(\top\mathcal{U}^{\mathcal{A}(s_1)}p)))$

By applying rule (For) to F3, we get

F4: $\mathbf{F}(\neg(\top\mathcal{U}^{\mathcal{A}(s_1)}p))$

and

F5: $\mathbf{F}(\top\wedge\langle a\rangle(\top\mathcal{U}^{\mathcal{A}_1(s_0)}\neg(\top\mathcal{U}^{\mathcal{A}(s_1)}p))$

By applying rule (Fand) we get two copies of the set of formulas. One of them contains formula $\mathbf{F}\top$, and thus it is inconsistent. The other one contains

F6: $\mathbf{F}(\langle a\rangle(\top\mathcal{U}^{\mathcal{A}_1(s_0)}\neg(\top\mathcal{U}^{\mathcal{A}(s_1)}p))$

F4 can be expanded with rule (Fneg), obtaining

F7: $\mathbf{T}(\top\mathcal{U}^{\mathcal{A}(s_1)}p)$

By applying rule (TuntilNFS) to F7, we get

F8: $\mathbf{T}(\top\wedge\langle a\rangle(\top\mathcal{U}^{\mathcal{A}(s_2)}p))$

and finally, by applying (Tand) to F8, we get

F9: $\mathbf{T}\top$

and

F10: $\mathbf{T}(\langle a\rangle(\top\mathcal{U}^{\mathcal{A}(s_2)}p))$

It is easy to see that for each set of formulas returned by *tableau* there is exactly one symbol $a\in\Sigma$ such that the set contains formulas of the form $\mathbf{T}\langle a\rangle\alpha$. In fact, because of $\mathbf{T}\bigvee_{a\in\Sigma}\langle a\rangle\top$, there must be at least one formula of that kind, whereas the consistency constraint prevents from having more than one formula of the form $\mathbf{T}\langle a\rangle\alpha$ for different symbols $a\in\Sigma$.

## 3.2 Building the graph

To build the graph we will consider each set of formulas obtained through the tableau construction as a node of the graph. The above tableau rules do not expand formulas of the kind $\mathbf{T}\langle a\rangle\alpha$. Since the operator $\langle a\rangle$ is a *next state* operator, expanding this kind of formulas from a node $n$ means to create a new node containing $\alpha$ connected to $n$ through an edge labelled with $a$. Thus an obvious procedure for building the graph would be to repeatedly apply to all nodes the following construction: if node $n$ contains a formula $\mathbf{T}\langle a\rangle\alpha$, then build the set of the nodes connected to $n$ through an edge labelled $a$ as $tableau(\{\mathbf{T}\alpha|\mathbf{T}\langle a\rangle\alpha\in n\}\cup\{\mathbf{F}\alpha|\mathbf{F}\langle a\rangle\alpha\in n\})$. States and transitions of the Büchi automaton correspond directly to the nodes and edges of the graph.

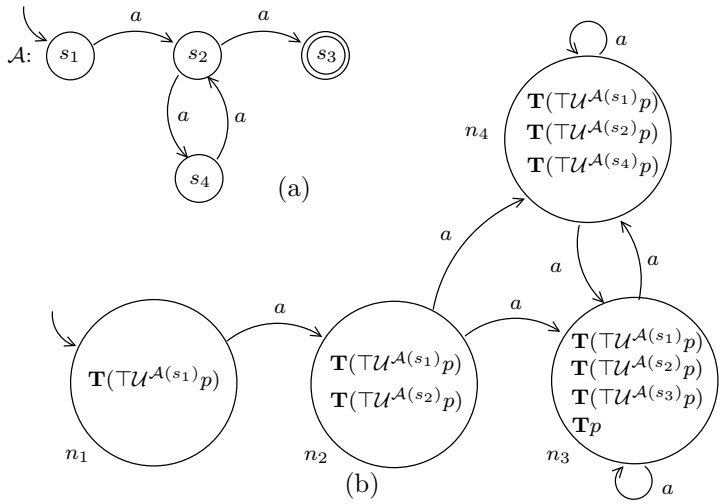


Figure 1: (a) automaton $\mathcal{A}$; (b) graph for $\Box\langle\mathcal{A}(S_1)\rangle p$.

**Example 2** Let us consider again formula $\Box\langle\mathcal{A}(s_1)\rangle p$. The initial node of the graph will contain formulas F1-F10 of Example 1. The adjacent node will be obtained first by moving formulas F6 and F10 through the next state $\langle a\rangle$ operator, obtaining:

F2: $\mathbf{F}(\top\mathcal{U}^{\mathcal{A}_1(s_0)}\neg(\top\mathcal{U}^{\mathcal{A}(s_1)}p))$, and

F11: $\mathbf{T}(\top\mathcal{U}^{\mathcal{A}(s_2)}p)$

Since this state contains again the initial formula of the previous state, i.e. formula F2, by applying function *tableau* to F2 and F11, we get formulas F1-F10 of Example 1, plus the following formulas derived from F11:

F12: $\mathbf{T}(\top\wedge\langle a\rangle((\top\mathcal{U}^{\mathcal{A}(s_3)}p)\vee(\top\mathcal{U}^{\mathcal{A}(s_4)}p)))$, and

F13: $\mathbf{T}\langle a\rangle((\top\mathcal{U}^{\mathcal{A}(s_3)}p)\vee(\top\mathcal{U}^{\mathcal{A}(s_4)}p))$

By continuing with this construction, we obtain the graph in Figure 1(b), where for simplicity we have kept only some significant formulas.

Unfortunately this construction does not allow to define correctly the accepting states of the automaton. Intuitively, we would like to define as accepting those runs in which all the until formulas of the form $\mathbf{T}\alpha\mathcal{U}^{\mathcal{A}(q)}\beta$ are fulfilled. If a node $n$ of the graph contains the formula $\mathbf{T}\alpha\mathcal{U}^{\mathcal{A}(q)}\beta$, then we can accept an infinite path containing $n$, if node $n$ is followed in the path by a node $n'$ containing $\mathbf{T}\beta$ and $\mathbf{T}\alpha\mathcal{U}^{\mathcal{A}(q_F)}\beta$, where $q_F$ is a final state of $\mathcal{A}$. Furthermore if $\tau$ is the sequence of labels in the path from $n$ to $n'$, then $\tau$ must belong to $\mathcal{L}(\mathcal{A}(q))$, and all nodes between $n$ and $n'$ must contain $\mathbf{T}\alpha$.

This problem has been solved in LTL by imposing generalized Büchi acceptance conditions [4]. Such condition can be stated as follows: For each subformula $\alpha \mathcal{U} \beta$ of the initial formula there is a set $F_i$ of accepting states including all the nodes $q \in Q$ such that either $\mathbf{T}\alpha \mathcal{U} \beta \notin q$ or $\mathbf{T}\beta \in q$ holds. A run $\rho$ is accepting iff $inf(\rho) \cap F_i \neq \emptyset$ for each $F_i$.

However a similar solution does not apply in the case of DLTL, because acceptance of until formulas is more constrained than in LTL.

**Example 3** Let us consider the graph in Figure 1(b). Every node of this graph contains a formula $\mathbf{T}(\top \mathcal{U}^{\mathcal{A}(s_1)} p)$, and the only node which might fulfill the until formulas is node $n_3$, since it contains $\mathbf{T}(\top \mathcal{U}^{\mathcal{A}(s_3)} p)$, with $s_3$ final, and $\mathbf{T}p$. However it is easy to see that not all infinite paths through $n_3$ will be accepting. For instance, in the path $n_1 n_2 (n_3 n_4)^\omega$, i.e. $n_1 n_2 n_3 n_4 n_3 n_4 n_3 n_4 \ldots$, no occurrence of $n_3$ fulfills the formula $\mathbf{T}(\top \mathcal{U}^{\mathcal{A}(s_1)} p)$ in $n_2$, since the distance in this path between node $n_2$ and any occurrence of $n_3$ is odd, while all strings in $\mathcal{L}(\mathcal{A}(s_1))$ have even length.

To overcome the above problem we present now a different solution, derived from [10], where the nodes containing until formulas will be marked to keep track of the fulfillment of those formulas. Before describing the construction of the graph, we make the following observation. Let us assume that a node $n$ contains the until formula $\mathbf{T}\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$, such that $q$ is not a final state. Since this formula has been expanded with (TuntilNFS), node $n$ will also contain $\mathbf{T}\langle a \rangle \bigvee_{q' \in \delta(q,a)} \alpha \mathcal{U}^{\mathcal{A}(q')} \beta$ for some $a$. Therefore, according to the construction of the successor nodes, each successor node will contain a formula $\mathbf{T}\alpha \mathcal{U}^{\mathcal{A}(q')} \beta$, where $q' \in \delta(q, a)$[5]. We say that this until formula is *derived* from formula $\mathbf{T}\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ in node $n$. On the other hand, if $q$ is a final state, then $\mathbf{T}\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ has been expanded with (TuntilFS), and two cases are possible: either $n$ contains $\mathbf{T}\beta$ or all successor nodes contain a derived until formula as described above.

**Definition 1** *Given a signed formula $\boldsymbol{T}\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ to which the (TuntilFS) rule (or the (TuntilNFS) rule) has been applied in a node $n$, the signed formulas $\boldsymbol{T}\alpha \mathcal{U}^{\mathcal{A}(q')} \beta$, with $q' \in \delta(q, a)$ and $a \in \Sigma$, obtained from the consequent of the rule by applying the propositional tableau rules and by expanding the next state operator $\langle a \rangle$ to create a new node $n'$, are said to be* derived *from $\boldsymbol{T}\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$.*

If a node contains an until formula which is not derived from a predecessor node, we will say that the formula is *new*. New until formulas are obtained during the expansion of the *tableau* procedure. It is easy to see that if $\mathbf{T}\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ is a new formula, then $\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ must be a subformula of the initial formula. For instance, the formula $\mathbf{T}(\top \mathcal{U}^{\mathcal{A}(s_1)} p)$ is new in each of the nodes in Figure 1. Note that an until formula in a node might be both a derived and a new formula. In that case we will consider it as a derived formula.

[5]Note that, if $\delta(q, a) = \emptyset$, then node $n$ contains $\mathbf{T}\langle a \rangle \bot$, and thus it has no outgoing edge labelled with $a$.

We can now show how the graph can be built and how the accepting conditions are formulated. Each node of the graph is a triple $(\mathcal{F}, x, f)$, where $\mathcal{F}$ is an expanded set of formulas, $x \in \{0, 1\}$, and $f \in \{\downarrow, \checkmark\}$.

In order to formulate the accepting condition, we must be able to trace the until formulas along the paths of the graph to make sure that they are fulfilled. To do this we extend signed formulas so that all until formulas have a label 0 or 1, i.e. they have the form $\mathbf{T}^l \alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ where $l \in \{0, 1\}$. Note that two formulas $\mathbf{T}^0 \alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ and $\mathbf{T}^1 \alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ are considered to be different.

For each node $(\mathcal{F}, x, f)$, the label of an until formula in $\mathcal{F}$ will be assigned as follows:

- if it is a derived until formula, then its label is the same as that of the until formula in the predecessor node it derives from,
- otherwise, if the formula is new, it is given the label $1 - x$.

Of course function *tableau* must be suitably modified in order to propagate the label from an until formula to its derived formulas in the successor nodes, and to give the right label to new formulas. To do this we assume that it has two parameters: a set of formulas and the value of $x$.

Function *create_graph* in Figure 2 builds a graph $\mathcal{G}(\phi)$, given an initial formula $\phi$, by returning the triple $\langle Q, I, \Delta \rangle$, where $Q$ is the set of nodes, $I$ the set of initial nodes and $\Delta : Q \times \Sigma \times Q$ the set of labelled edges. $U$ is the set of nodes whose successor nodes have still to be determined.

**Example 4** Let us apply some steps of *create_graph* to the formula of Figure 1. As before we only put the most significant formulas in the set of formulas of a node. The initial node will be:

$n1 = (\{\mathbf{T}^1(\top \mathcal{U}^{\mathcal{A}(s_1)} p)\}, 0, \checkmark)$

From this node we get the node

$n2 = (\{\mathbf{T}^0(\top \mathcal{U}^{\mathcal{A}(s_1)} p), \mathbf{T}^1(\top \mathcal{U}^{\mathcal{A}(s_2)} p)\}, 1, \downarrow)$

connected to $n1$ through an edge $(n1, a, n2)$. From $n2$ we get the nodes

$n3 = (\{\mathbf{T}^0(\top \mathcal{U}^{\mathcal{A}(s_1)} p), \mathbf{T}^0(\top \mathcal{U}^{\mathcal{A}(s_2)} p), \mathbf{T}^1(\top \mathcal{U}^{\mathcal{A}(s_3)} p), \mathbf{T} p\}, 1, \downarrow)$

and

$n4 = (\{\mathbf{T}^0(\top \mathcal{U}^{\mathcal{A}(s_1)} p), \mathbf{T}^0(\top \mathcal{U}^{\mathcal{A}(s_2)} p), \mathbf{T}^1(\top \mathcal{U}^{\mathcal{A}(s_4)} p)\}, 1, \downarrow)$

connected to $n2$ through the edge $(n2, a, n3)$ and $(n2, a, n4)$. These nodes correspond to the nodes with the same name in Figure 1. However, if we continue the construction of the graph from node $n3$, we get the nodes

$n5 = (\{\mathbf{T}^0(\top \mathcal{U}^{\mathcal{A}(s_1)} p), \mathbf{T}^0(\top \mathcal{U}^{\mathcal{A}(s_2)} p), \mathbf{T}^0(\top \mathcal{U}^{\mathcal{A}(s_3)} p), \mathbf{T} p\}, 1, \checkmark)$

and

$n6 = (\{\mathbf{T}^0(\top \mathcal{U}^{\mathcal{A}(s_1)} p), \mathbf{T}^0(\top \mathcal{U}^{\mathcal{A}(s_2)} p), \mathbf{T}^0(\top \mathcal{U}^{\mathcal{A}(s_4)} p)\}, 1, \checkmark)$

which are different from nodes $n3$ and $n4$ because the until formulas have different labels, and $f$ has different values.

**Theorem 1** *The algorithm in Figure 2 terminates.*

**function** *create_graph*($\phi$)
$I := \emptyset$
**for all** $\mathcal{F} \in tableau(\{\mathbf{T}\phi\}, 0)$
    $I := I \cup \{(\mathcal{F}, 0, \checkmark)\}$
**end-for**
$U := Q := I$
$\Delta := \emptyset$
**while** $U \neq \emptyset$ **do**
    remove $n = (\mathcal{F}, x, f)$ from $U$
    **if** $f = \checkmark$ **then**
        $x' := 1 - x$
    **else**
        $x' := x$
    **end-if**
    **for all** $\mathcal{F}' \in tableau(\{\mathbf{T}\alpha | \mathbf{T}\langle a\rangle\alpha \in \mathcal{F}\} \cup$
                $\{\mathbf{F}\alpha | \mathbf{F}\langle a\rangle\alpha \in \mathcal{F}\}, x')$
        **if** $f = \checkmark$ **then**
            $f' :=\downarrow$
        **else if** there exists no $\mathbf{T}^{x'}\alpha\mathcal{U}^{\mathcal{A}(q)}\beta \in \mathcal{F}'$ **then**
            $f' := \checkmark$
        **else**
            $f' :=\downarrow$
        **end-if**
        **end-if**
        $n' := (\mathcal{F}', x', f')$
        **if** $\exists n'' \in Q$ such that $n'' = n'$ **then**
            $\Delta := \Delta \cup \{(n, a, n'')\}$
        **else**
            $Q := Q \cup \{n'\}$
            $\Delta := \Delta \cup \{(n, a, n')\}$
            $U := U \cup \{n'\}$
        **end-if**
    **end-for**
**end-while**
**return** $\langle Q, I, \Delta\rangle$

Figure 2: Function *create_graph*

**Proof.** First of all observe that the number of possible different nodes is finite. In fact, a node can only contain formulas which occur in the initial formula or which can be built from the subformulas occurring in the initial one by (finitely many) changes of the initial state of the automata in the until operator. All the (finitely many) nodes that can be added to the set $U$ will be eventually removed, so that $U$ will eventually become empty. □

Given a graph $\mathcal{G}(\phi)$, a Büchi automaton $\mathcal{B}(\phi)$ can be obtained as follows:

- States and transitions of $\mathcal{B}(\phi)$ correspond directly to the nodes and edges of $\mathcal{G}(\phi)$.
- The set of accepting states of $\mathcal{B}(\phi)$ consists of all states whose corresponding node contains $f = \checkmark$.

Let $\rho$ be a run of $\mathcal{B}(\phi)$. Since we identify states of the automaton with nodes of the graph, $\rho$ can also be considered as an infinite path of $\mathcal{G}(\phi)$, and $\rho(\tau)$ will denote a node of such a graph. According to the construction of the graph, the $x$ and $f$ values of the nodes of $\rho$ have the following properties:

- if a node contains $(0, \checkmark)$ then its successor node contains $(1, \downarrow)$
- if a node contains $(1, \checkmark)$ then its successor node contains $(0, \downarrow)$
- if a node contains $(0, \downarrow)$ then its successor node contains either $(0, \downarrow)$ or $(0, \checkmark)$
- if a node contains $(1, \downarrow)$ then its successor node contains either $(1, \downarrow)$ or $(1, \checkmark)$

Therefore the sequence of the $x$ and $f$ values in $\rho$ will be as follows:
$(0, \checkmark), (1, \downarrow), \ldots, (1, \downarrow), (1, \checkmark), (0, \downarrow), \ldots, (0, \downarrow), (0, \checkmark), \cdots$

Let us call *0-sequences* or *1-sequences* the sequences of nodes of $\rho$ with $x = 0$ or $x = 1$ respectively. If $\rho$ is an *accepting run*, then it must contain infinitely many nodes containing $\checkmark$, and thus all 0-sequences and 1-sequences must be finite.

Intuitively, every until formula contained in a node of a 0-sequence must be fulfilled within the end of the next 1-sequence, and vice versa. In fact, assuming that the formula has label 1, the label will be propagated to all derived formulas in the following nodes until a node is found fulfilling the until formula. But, on the other hand, the 1-sequence terminates only when there are no more until formulas with label 1, and thus that node must be met before the end of the next 1-sequence.

**Example 5** Let us consider some further nodes for our example, besides the nodes in Example 4:
$n7 = (\{\mathbf{T}^1(\top \mathcal{U}^{\mathcal{A}(s_1)} p), \mathbf{T}^0(\top \mathcal{U}^{\mathcal{A}(s_2)} p), \mathbf{T}^0(\top \mathcal{U}^{\mathcal{A}(s_3)} p), \mathbf{T}p\}, 0, \downarrow)$
$n8 = (\{\mathbf{T}^1(\top \mathcal{U}^{\mathcal{A}(s_1)} p), \mathbf{T}^1(\top \mathcal{U}^{\mathcal{A}(s_2)} p), \mathbf{T}^0(\top \mathcal{U}^{\mathcal{A}(s_4)} p)\}, 0, \downarrow)$
$n9 = (\{\mathbf{T}^1(\top \mathcal{U}^{\mathcal{A}(s_1)} p), \mathbf{T}^0(\top \mathcal{U}^{\mathcal{A}(s_2)} p), \mathbf{T}^1(\top \mathcal{U}^{\mathcal{A}(s_2)} p), \mathbf{T}^1(\top \mathcal{U}^{\mathcal{A}(s_3)} p), \mathbf{T}p\}, 0, \downarrow)$

$n10 = (\{\mathbf{T}^1(\top\mathcal{U}^{\mathcal{A}(s_1)}p), \mathbf{T}^1(\top\mathcal{U}^{\mathcal{A}(s_2)}p), \mathbf{T}^0(\top\mathcal{U}^{\mathcal{A}(s_4)}p), \mathbf{T}^1(\top\mathcal{U}^{\mathcal{A}(s_4)}p)\}, 0, \downarrow)$

These nodes are connected through the edges: $(n_6, a, n_7)$, $(n_7, a, n_8)$, $(n_8, a, n_9)$, $(n_9, a, n_{10})$, $(n_{10}, a, n_9)$.

The infinite path $n_1n_2n_3n_6n_7n_8(n_9n_{10})^\omega$ does not represent an accepting run, since the loop $(n_9n_{10})^\omega$ does not contain any accepting state. It is easy to see that this is not an accepting run because, as in Example 3, there are some until formulas which are never fulfilled.

On the other hand, let us consider the following nodes:

$n11 = (\{\mathbf{T}^1(\top\mathcal{U}^{\mathcal{A}(s_1)}p), \mathbf{T}^1(\top\mathcal{U}^{\mathcal{A}(s_2)}p), \mathbf{T}^0(\top\mathcal{U}^{\mathcal{A}(s_3)}p), \mathbf{T}p\}, 0, \downarrow)$

$n12 = (\{\mathbf{T}^1(\top\mathcal{U}^{\mathcal{A}(s_1)}p), \mathbf{T}^1(\top\mathcal{U}^{\mathcal{A}(s_2)}p), \mathbf{T}^1(\top\mathcal{U}^{\mathcal{A}(s_3)}p), \mathbf{T}p\}, 0, \checkmark)$

$n13 = (\{\mathbf{T}^0(\top\mathcal{U}^{\mathcal{A}(s_1)}p), \mathbf{T}^1(\top\mathcal{U}^{\mathcal{A}(s_2)}p), \mathbf{T}^1(\top\mathcal{U}^{\mathcal{A}(s_3)}p), \mathbf{T}p\}, 1, \downarrow)$

connected through the edges: $(n_5, a, n_7)$, $(n_7, a, n_{11})$, $(n_{11}, a, n_{12})$, $(n_{12}, a, n_{13})$, $(n_{13}, a, n_3)$.

The infinite path $n_1n_2(n_3n_5n_7n_{11}n_{12}n_{13})^\omega$ represents an accepting run, since the loop $(n_3n_5n_7n_{11}n_{12}n_{13})^\omega$ contains two accepting states. It is easy to see that each until formula with label $l$ will be fulfilled by the end of the next $l-sequence$.

### 3.3 Correctness of the procedure

In this section we show that the satisfiability of a DLTL formula can be decided by building the Büchi automaton accepting all the models of the formula and then by determining if the language recognized by the automaton is nonempty.

The next proposition summarizes some properties that we have already pointed out in the previous section.

**Proposition 1** *Assume that a node $n$ of the graph contains $\boldsymbol{T}^l\alpha\mathcal{U}^{\mathcal{A}(q)}\beta$, and let $a$ be the label of the outgoing edges (remember that all outgoing edges from a node have the same label). Then the following holds:*
*if $q$ is not a final state of $\mathcal{A}$*

> *node $n$ contains $\boldsymbol{T}\alpha$ and each outgoing edge leads to a node containing an until formula $\boldsymbol{T}^l\alpha\mathcal{U}^{\mathcal{A}(q')}\beta$ derived from $\boldsymbol{T}^l\alpha\mathcal{U}^{\mathcal{A}(q)}\beta$ in $n$, such that $q' \in \delta(q, a)$*

*else, if $q$ is a final state of $\mathcal{A}$, either*

> *(a) node $n$ contains $\boldsymbol{T}\beta$ and no successor node contains a formula derived from $\boldsymbol{T}^l\alpha\mathcal{U}^{\mathcal{A}(q)}\beta$, or*
> *(b) node $n$ contains $\boldsymbol{T}\alpha$ and each outgoing edge leads to a node containing a derived until formula $\boldsymbol{T}^l\alpha\mathcal{U}^{\mathcal{A}(q')}\beta$, such that $q' \in \delta(q, a)$*

The following proposition can be easily proved from the tableau construction.

**Proposition 2** *Assume that a node $n$ of the graph contains $\boldsymbol{F}\alpha\mathcal{U}^{\mathcal{A}(q)}\beta$. Then either:*

> *(a) node $n$ contains $\boldsymbol{F}\alpha$, or*
> *(b) for each $a$ such that $\delta(q, a) \neq \emptyset$, each outgoing edge from the node $n$ labelled with $a$ leads to a node $n'$ containing a formula $\boldsymbol{F}\alpha\mathcal{U}^{\mathcal{A}(q')}\beta$ such that $q' \in \delta(q, a)$*

*Furthermore, if $q$ is a final state of $\mathcal{A}$, node $n$ contains $\boldsymbol{F}\beta$.*

Given a run $\rho$, we will denote with $\rho(\tau).\mathcal{F}$ the $\mathcal{F}$ field of the node $\rho(\tau)$, and similarly for the $x$ and $f$ fields.

**Proposition 3** *Let $\sigma \in \Sigma^\omega$ and $\rho : prf(\sigma) \to Q$ be a (non necessarily accepting) run. For each $\tau \in prf(\sigma)$ and for each $\boldsymbol{T}^l \alpha \mathcal{U}^{\mathcal{A}(q)} \beta \in \rho(\tau).\mathcal{F}$ one of the following holds:*

1. *$\forall \tau'$ s.t. $\tau\tau' \in prf(\sigma) : \boldsymbol{T}^l \alpha \mathcal{U}^{\mathcal{A}(q')} \beta \in \rho(\tau\tau').\mathcal{F}$ and $q' \in \delta^*_{\mathcal{A}}(q, \tau')$* [6]
2. *$\exists \tau'$ s.t. $\tau\tau' \in prf(\sigma) : \boldsymbol{T}^l \alpha \mathcal{U}^{\mathcal{A}(q')} \beta \in \rho(\tau\tau').\mathcal{F}$, $q'$ is a final state, $q' \in \delta^*_{\mathcal{A}}(q, \tau')$, $\boldsymbol{T}\beta \in \rho(\tau\tau').\mathcal{F}$ and no successor node of $\rho(\tau\tau')$ contains an until formula derived from $\boldsymbol{T}^l \alpha \mathcal{U}^{\mathcal{A}(q')} \beta$. Moreover, for every $\tau''$ such that $\varepsilon \leq \tau'' < \tau'$, $\boldsymbol{T}\alpha \in \rho(\tau\tau'').\mathcal{F}$.*

*Furthermore for each $\boldsymbol{F}\alpha \mathcal{U}^{\mathcal{A}(q)} \beta \in \rho(\tau).\mathcal{F}$ the following holds:*

3. *$\forall \tau'$ s.t. $\tau\tau' \in prf(\sigma)$: if $\tau' \in \mathcal{L}(\mathcal{A}(q))$ then either $\boldsymbol{F}\beta \in \rho(\tau\tau').\mathcal{F}$ or there is $\tau''$ such that $\varepsilon \leq \tau'' < \tau'$, $\boldsymbol{F}\alpha \in \rho(\tau\tau'').\mathcal{F}$.*

**Proof.** It follows from Propositions 1 and 2, and procedure *create_graph*. □

In an accepting run, case (2) must hold for all until formulas and all nodes. This is proved in the following theorem, together with its converse.

**Theorem 2** *Let $\sigma \in \Sigma^\omega$ and $\rho : prf(\sigma) \to Q$ be a run. Then, for each $\tau \in prf(\sigma)$ and for each $\boldsymbol{T}^l \alpha \mathcal{U}^{\mathcal{A}(q)} \beta \in \rho(\tau).\mathcal{F}$, condition (2) of Proposition 3 holds if and only if $\rho$ is an accepting run.*

**Proof.** If part: $\rho$ is an accepting run. As pointed out before the nodes of $\rho$ are arranged in alternating 0-sequences and 1-sequences of finite length. Then we can have the following cases:

a) $l = 0$ and $\rho(\tau).x = 0$. Let us assume that condition (1) of Proposition 3 holds. Then each node $\rho(\tau\tau')$ following $\rho(\tau)$ in the same 0-sequence, will contain a derived formula $\mathbf{T}^0 \alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ (remember that the label of a derived formula cannot change). On the other hand, the 0-sequence containing $\rho(\tau)$ is finite, and, by construction, the last node of this sequence does not contain any until formula labelled with 0. Therefore the assumption is wrong, and condition (2) must hold.

b) $l = 1$ and $\rho(\tau).x = 1$. Same as case (a).

[6] $\delta^*_{\mathcal{A}}$ is the obvious extension of $\delta_{\mathcal{A}}$ to sequences

c) $l = 1$ and $\rho(\tau).x = 0$. Let us assume again that condition (1) of Proposition 3 holds. Then each node $\rho(\tau\tau')$ following $\rho(\tau)$ will contain an until formula derived from $\mathbf{T}^1\alpha\mathcal{U}^{\mathcal{A}(q)}\beta$ in $\rho(\tau)$. All derived formulas will be labelled 1 up to the last node of the 0-sequence. This label will necessarily propagate to the first node of the following 1-sequence, and we are reduced to case (b).

d) $l = 0$ and $\rho(\tau).x = 1$. Same as case (c).

Only if: condition (2) holds. We show that all 0 and 1-sequences of $\rho$ are finite. This is true for the initial 0-sequence, which consists only of the first node. Let us assume now that a 0-sequence is finite. We show that the following 1-sequence is also finite. According to the construction, the last node of the 0-sequence can contain only until formulas with label 1. The following 1-sequence goes on until its nodes contain some until formula with label 1. Since condition (2) holds, for each of these until formulas there is a $\tau'$ such that the successor node of $\rho(\tau\tau')$ does not contain an until formula derived from it. On the other hand all new until formulas created in this 1-sequence will have label 0. Therefore, if $\tau^{max}$ is the longest among all $\tau'$, after node $\rho(\tau\tau^{max})$ there will be no until formula labelled with 1, and the 1-sequence will terminate. The same holds by replacing 0 with 1 and vice versa. □

Given a set $s$ of signed formulas, in the following we will denote by $\bigwedge s$ the conjunction of the formulas in $s$, where the prefixes $\mathbf{T}$ are removed and the prefixes $\mathbf{F}$ are replaced by $\neg$.

**Lemma 1** *Let $s$ be a set of formulas and tableau$(s) = \{s_1, \ldots, s_n\}$. Then $\bigwedge s \leftrightarrow \bigvee_{1 \leq i \leq n} \bigwedge s_i$.*

**Proof.** All rules used by the function *tableau* correspond to equivalence formulas. □

**Lemma 2** *Let $M = (\sigma, V)$ be a model, $\tau \in prf(\sigma)$, and let $n = (\mathcal{F}, x, f)$ be a node of the graph such that $M, \tau \models \bigwedge \mathcal{F}$. Then there exists a successor $n' = (\mathcal{F}', x', f')$ of $n$ such that $M, \tau a \models \bigwedge \mathcal{F}'$, where $\tau a \in prf(\sigma)$. Moreover, if $\boldsymbol{T}\alpha\mathcal{U}^{\mathcal{A}(q)}\beta \in \mathcal{F}'$ where $q$ is a final state and $M, \tau a \models \beta$, then $\boldsymbol{T}\beta \in \mathcal{F}'$.*

**Proof.** The proof comes from the construction and the previous lemma. In particular the last part holds if, when expanding $\mathbf{T}\alpha\mathcal{U}^{\mathcal{A}(q)}\beta$ in $\mathcal{F}'$ with rule (TuntilFS), we choose the set containing $\mathbf{T}\beta$. □

**Theorem 3** *Let $M = (\sigma, V)$ and $M, \varepsilon \models \phi$. Then $\sigma \in \mathcal{L}(\mathcal{B}(\phi))$.*

**Proof.** We show how to build an accepting run $\rho$ of $\mathcal{B}(\phi)$ over $\sigma$. The first node of $\rho$ is chosen by taking an initial node $n = (\mathcal{F}, x, f)$ of the graph such that $M, \varepsilon \models \bigwedge \mathcal{F}$. The following nodes of $\rho$ are chosen by repeatedly applying Lemma 2. To prove that the run is an accepting run, we have to show that all the until formulas are fulfilled. Assume that $\mathbf{T}\alpha\mathcal{U}^{\mathcal{A}(q)}\beta$ occurs on the run at

$\rho(\tau)$. Then, for the choice of the run $\rho$, it must be the case that $M, \tau \models \alpha \mathcal{U}^{\mathcal{A}(q)} \beta$. By definition of satisfiability we have that there exists $\tau' \in \mathcal{L}(\mathcal{A}(q))$ such that $\tau\tau' \in prf(\sigma)$ and $M, \tau\tau' \models \beta$. Moreover, for every $\tau''$ such that $\varepsilon \leq \tau'' < \tau'$, $M, \tau\tau'' \models \alpha$. As $\tau' \in \mathcal{L}(\mathcal{A}(q))$, by the choice of run $\rho$ and the construction of the automaton, there must be a final state $q' \in \delta^*_{\mathcal{A}}(q, \tau')$ such that $\mathbf{T}\alpha\mathcal{U}^{\mathcal{A}(q')}\beta$ belongs to $\rho(\tau\tau').\mathcal{F}$. Moreover for all $\tau''$ such that $\varepsilon \leq \tau'' < \tau'$, $\mathbf{T}\alpha$ belongs to $\rho(\tau\tau'').\mathcal{F}$. By Lemma 2, $\mathbf{T}\beta$ also belongs to $\rho(\tau\tau').\mathcal{F}$. Hence, condition (2) of Proposition 3 holds and we can conclude, by Theorem 2, that $\rho$ is an accepting run. □

Given a set $\mathcal{F}$ of signed formulas, we define the sets $Pos(\mathcal{F})$ and $Neg(\mathcal{F})$ respectively as the sets of positive and negative propositions in $\mathcal{F}$, i.e. $Pos(\mathcal{F}) = \{p \in \mathcal{P} | \mathbf{T}p \in \mathcal{F}\}$, and $Neg(\mathcal{F}) = \{p \in \mathcal{P} | \mathbf{F}p \in \mathcal{F}\}$.

**Theorem 4** *Let $\sigma \in \mathcal{L}(\mathcal{B}(\phi))$. Then there is a model $M = (\sigma, V)$ such that $M, \varepsilon \models \phi$.*

**Proof.** Let $\rho$ be an accepting run. For each $\tau \in prf(\sigma)$ let $\rho(\tau) = (\mathcal{F}_\tau, x_\tau, f_\tau)$. The model $M = (\sigma, V)$ can be obtained by defining $V(\tau) \in 2^{\mathcal{P}}$ such that $V(\tau) \supseteq Pos(\mathcal{F}_\tau)$ and $V(\tau) \cap Neg(\mathcal{F}_\tau) = \emptyset$.

It is easy to prove by induction on the structure of formulas that, for each $\tau$ and for each formula $\alpha$, if $\mathbf{T}\alpha \in \mathcal{F}_\tau$ then $M, \tau \models \alpha$, and if $\mathbf{F}\alpha \in \mathcal{F}_\tau$ then $M, \tau \not\models \alpha$. In particular, for until formulas labelled $\mathbf{T}$ we make use of Theorem 2 and of Proposition 3, case 2, while for until formulas labelled $\mathbf{F}$ we make use of Proposition 3, case 3. From $\mathbf{T}\phi \in \mathcal{F}_\epsilon$, it follows that $M, \epsilon \models \phi$. □

From Theorem 3 and Theorem 4 we can conclude that:

**Theorem 5** *A DLTL formula $\phi$ is satisfiable if and only if $\mathcal{L}(\mathcal{B}(\phi)) \neq \emptyset$.*

## 3.4 Complexity

As described in Section 3, the satisfiability of a DLTL formula $\phi$ can be decided by building the Büchi automaton accepting all the models of the formula. In this section we show that, though in the worst case, the number of states of the Büchi automaton is exponential in the size of $|\phi|$, the satisfiability problem for DLTL is in PSPACE.

To show this we can make use of the same argument used in [21] to show that the satisfiability problem for ETL is in PSPACE. The argument is that, though the automaton is exponential in size, it is not necessary to build the whole automaton before applying the algorithm for nonemptiness. In fact, the nonemptiness problem for Büchi automata is in NLOGSPACE (see Theorem 2.4 in [21]).

More precisely, to show that there is a polynomial space algorithm for determining if a formula $\phi$ of DLTL($\Sigma$) is satisfiable, by making use of Lemma 2.5

in [21], it is enough to show that it is possible to associate to $\phi$ the encoding of a Büchi automaton

$$\mathcal{B}_\phi = (Q_\phi, \rightarrow_\phi, Q^\phi_{in}, F_\phi)$$

over $\Sigma$ in such a way that $\phi$ is satisfiable iff $\mathcal{B}_\phi$ is nonempty and, furthermore: the encoding of the states in $Q_\phi$ and of the symbol in $\Sigma$ can be done in polynomial space; the membership in $\Sigma$, $Q_\phi$, $Q^\phi_{in}$, $F_\phi$ can be decided in polynomial space; finally, verifying whether a state can be obtained by another one by a transition of the automata can be decided in polynomial space.

It is not difficult to see that the automaton construction in section 3.2 satisfies the conditions mentioned above. In particular, let us show that the size of the states of the automaton is polynomial in the size of $\phi$.

Let $\alpha_1 \mathcal{U}^{\pi_1} \beta_1$,....,$\alpha_n \mathcal{U}^{\pi_n} \beta_n$ be all the until formulas occurring in the initial formula $\phi$. It is known that for $\pi_i \in Prg(\Sigma)$, we can construct in polynomial time a non-deterministic finite state automaton $\mathcal{A}_i$ with $\mathcal{L}(\mathcal{A}_i) = [[\pi_i]]$ such that the number of states of $\mathcal{A}_i$ is linear in the size of $\pi_i$ [12]. Therefore, each until formula $\alpha_i \mathcal{U}^{\mathcal{A}_i(q)} \beta_i$, replacing $\alpha_i \mathcal{U}^{\pi_i} \beta_i$ in the initial formula $\phi$, has size linear in $|\phi|$. Moreover, its expansion introduces at most a number of formulas which is polynomial in the size of $\mathcal{A}_i$ and, hence, polynomial in the size of $\phi$. In fact, observe that the expansion of the until formula $\alpha_i \mathcal{U}^{\mathcal{A}_i(q)} \beta_i$ (and its descendants) introduces: at most $|Q_{\mathcal{A}_i}|$ subformulas of the form $\alpha_i \mathcal{U}^{\mathcal{A}_i(q')} \beta_i$, with $q' \in Q_{\mathcal{A}_i}$; at most, $|Q_{\mathcal{A}_i}|$ subformulas of the form $\bigvee_{a \in \Sigma} \langle a \rangle \bigvee_{q'' \in \delta(q',a)} \alpha_i \mathcal{U}^{\mathcal{A}_i(q'')} \beta_i$ (one for each $q' \in Q_{\mathcal{A}_i}$); at most, $|Q_{\mathcal{A}_i}| \times |\Sigma|$ subformulas $\langle a \rangle \bigvee_{q'' \in \delta(q',a)} \alpha_i \mathcal{U}^{\mathcal{A}_i(q'')} \beta_i$ (one for each $q' \in Q_{\mathcal{A}_i}$ and $a \in \Sigma$ ); at most, $|Q_{\mathcal{A}_i}| \times |\Sigma|$ subformulas $\bigvee_{q'' \in \delta(q',a)} \alpha_i \mathcal{U}^{\mathcal{A}_i(q'')} \beta_i$ (one for each $q' \in Q_{\mathcal{A}_i}$ and $a \in \Sigma$ ). Moreover, $|Q_{\mathcal{A}_i}|$ is linear in $|\pi_i|$, and and hence linear in the size of $\phi$. The alphabet $\Sigma$ contains only symbols occurring in the initial formula $\phi$, so its size is linear in in the size of $\phi$. Hence, $|Q_{\mathcal{A}_i}| \times |\Sigma|$ is quadratic in the size of $\phi$. Finally, the size of each formula added by the expansion rules is polynomial in size with respect to $|\phi|$. For instance, the size of the formula $\bigvee_{a \in \Sigma} \langle a \rangle \bigvee_{q'' \in \delta(q',a)} \alpha_i \mathcal{U}^{\mathcal{A}_i(q'')} \beta_i$ is at most $|\Sigma| \times |Q_{\mathcal{A}_i}| \times |\phi|$, that is, cubic in $|\phi|$. In essence, the expansion rules introduce at most a polynomial number of formulas of size polynomial in $\phi$.

The automaton $\mathcal{B}_\phi$ can be checked for nonemptiness in polynomial space in the size of $\phi$ by making use of the algorithm described in [21] (see the proof of Theorem 2.4). To find a model satisfying $\phi$, the algorithm nondeterministically guesses an initial state $s$ and a final state $r$ and then nondeterministically tries to construct a path from $s$ to $r$ and a path from $r$ to itself. To construct a path from $x$ to $y$ the algorithm chooses a transition from the current state (initially $x$) to a target state, which in turn will become the current state. At each point of the algorithm only three states are remembered (the current state, the next state and the state $y$), which are of size polynomial in the size of $\phi$, and all the steps of the algorithm can be executed in polynomial space in the size of $\phi$. Therefore, we can conclude that:

**Theorem 6** *The problem of verifying the satisfiability of a DLTL formula is in*

*PSPACE.*

Given the fact that satisfiability for an LTL formula is PSPACE-hard [17], we can conclude that the satisfiability problem for DLTL is PSPACE-complete.

# 4 The Product Version of DLTL

Let us now recall the definition of DLTL$^{\otimes}$ from [9]. Let $Loc = \{1, \ldots, K\}$ be a set of *locations*, the names of the agents synchronizing on common actions. A *distributed alphabet* $\tilde{\Sigma} = \{\Sigma_i\}_{i=1}^{K}$ is a family of (possibly non-disjoint) alphabets, with each $\Sigma_i$ a non-empty, finite set of actions ($\Sigma_i$ is the set of actions which require the participation of agent $i$). Let $\Sigma = \bigcup_{i=1}^{K} \Sigma_i$. For $\sigma \in \Sigma^{\infty}$, we denote by $\sigma \restriction i$ the projection of $\sigma$ unto $\Sigma_i$, that is the sequence obtained by erasing from $\sigma$ all occurrences of symbols that are not in $\Sigma_i$. Furthermore $Loc(a)$ denotes the set of all locations which share action $a$, i.e. $Loc(a) = \{i \mid a \in \Sigma_i\}$.

Atomic propositions are introduced in a local fashion, by introducing a non-empty set of atomic propositions $\mathcal{P}$. For each proposition $p \in \mathcal{P}$ and agent $i \in Loc$, $p_i$ represents the "local" view of the proposition $p$ at $i$, and is evaluated in the local state of agent $i$.

Let us define the set of formulas of DLTL$^{\otimes}(\tilde{\Sigma})$ and their locations (if $\alpha$ is a formula, then $loc(\alpha)$, which is a subset of $Loc$, denotes its location):

(a) if $p \in \mathcal{P}$ and $i \in Loc$, $p_i$ is a formula and $loc(p_i) = \{i\}$;
(b) if $\alpha$ and $\beta$ are formulas, then $\neg\alpha$ and $\alpha \vee \beta$ are formulas and $loc(\neg\alpha) = loc(\alpha)$ and $loc(\alpha \vee \beta) = loc(\alpha) \cup loc(\beta)$;
(c) if $\alpha$ and $\beta$ are formulas and $loc(\alpha) = loc(\beta) = \{i\}$ and $\pi \in Prg(\Sigma_i)$, then $\alpha \mathcal{U}_i^{\pi} \beta$ is a formula and $loc(\alpha \mathcal{U}_i^{\pi} \beta) = \{i\}$.

Notice that no nesting of modalities $\mathcal{U}_i$ and $\mathcal{U}_j$ (for $i \neq j$) is allowed, and the formulas in DLTL$^{\otimes}(\tilde{\Sigma})$ are boolean combinations of formulas from the set $\bigcup_i DLTL_i^{\otimes}(\tilde{\Sigma})$, where

$$DLTL_i^{\otimes}(\tilde{\Sigma}) = \{\alpha \mid \alpha \in DLTL^{\otimes}(\tilde{\Sigma}) \textit{ and } loc(\alpha) = \{i\}\}.$$

A model of DLTL$^{\otimes}(\tilde{\Sigma})$ is a pair $M = (\sigma, V)$, where $\sigma \in \Sigma^{\infty}$ and $V = \{V_i\}_{i=1}^{K}$ is a family of functions $V_i$, where $V_i : prf(\sigma \restriction i) \to 2^{\mathcal{P}}$ is the valuation function for location $i$.

The satisfiability of formulas in a model is defined as above, except that propositions are evaluated locally. In particular, for all $\tau \in prf(\sigma)$:

- $M, \tau \models p_i$ iff $p \in V_i(\tau \restriction i)$;
- $M, \tau \models \alpha \mathcal{U}_i^{\pi} \beta$ iff there exists a $\tau'$ such that $\tau' \restriction i \in [[\pi]]$, $\tau\tau' \in prf(\sigma)$ and $M, \tau\tau' \models \beta$. Moreover, for all $\tau'' \in prf(\tau')$, if $\varepsilon \leq \tau'' \restriction i < \tau' \restriction i$, then $M, \tau\tau'' \models \alpha$.

Satisfiability in DLTL$^{\otimes}$ is defined as in DLTL. Moreover, the derived modalities $\langle\pi\rangle_i$, $[\pi]_i$, $\bigcirc_i$, $\Diamond_i$ and $\Box_i$ are defined as in Section 2, but only considering the actions in $\Sigma_i$.

In the product version of DLTL the global state of the system can be regarded as a set of local states, one for each agent $i$. The execution of an action $a$ may cause the modification of the local state of agent $i$ only when $a \in \Sigma_i$.

# 5 Automaton construction for DLTL$^{\otimes}$

In [9] it is shown how to check the satisfiability of a formula $\phi \in$ DLTL$^{\otimes}$, by constructing a product Büchi automaton $\mathcal{B}(\phi)$, such that the language accepted by $\mathcal{B}(\phi)$ is non-empty iff $\phi$ is satisfiable. Here we show how the on-the-fly construction of Section 3 can be adapted to DLTL$^{\otimes}$, by outlining the main features of the construction.

Let us consider first function *tableau(s)*, where $s$ is a set of formulas of DLTL$^{\otimes}$. This function works as the one in Section 3, with the only difference that we do not add to $s$ the formula $\mathbf{T}\bigvee_{a\in\Sigma_i}\langle a\rangle_i\top$, but we add the formula $\mathbf{T}\bigvee_{a\in\Sigma}\bigwedge_{i\in Loc(a)}\langle a\rangle_i\top$. The reason is that the semantics of DLTL$^{\otimes}$ does not require $\sigma \restriction i$ to be infinite, but it only requires $\sigma$ to be infinite. Moreover, when an action $a$ is executed, it must be executed simultaneously by all the locations which share the action $a$.

Given a set of formulas returned by *tableau(s)*, we can eliminate from it without loss of information all disjunctions and conjunctions. All remaining formulas have exactly one location, and thus can be partitioned into $K$ sets $\mathcal{F}_1, \ldots, \mathcal{F}_K$, such that each formula $\phi \in \mathcal{F}_i$ has location $\{i\}$.

Each node of the graph is a tuple $(\mathcal{F}_1, \ldots, \mathcal{F}_K, x, f)$, where each $\mathcal{F}_i$ is an expanded set of formulas, such that each formula $\phi \in \mathcal{F}_i$ has location $\{i\}$. Furthermore, as before, $x \in \{0, 1\}$, and $f \in \{\downarrow, \checkmark\}$. Edges of the graph are labeled with symbols of $\Sigma$.

Given a node $n = (\mathcal{F}_1, \ldots, \mathcal{F}_K, x, f)$, there is a node $n' = (\mathcal{F}'_1, \ldots, \mathcal{F}'_K, x', f')$ adjacent to it through an edge labeled with $a \in \Sigma$ if, for all $\mathcal{F}_i$ such that $a \in \Sigma_i$, $\mathcal{F}_i$ contains a formula $\mathbf{T}\langle a\rangle_i\alpha$. Under such conditions, for all $i$ with $a \in \Sigma_i$, the set $\mathcal{F}'_i$ is obtained from $\mathcal{F}_i$ by proceeding as in Section 3. Instead, for all $i$ with $a \notin \Sigma_i$, $\mathcal{F}'_i = \mathcal{F}_i$. The values of $x'$ and $f'$ are obtained as in function *create_graph*.

# 6 Reasoning about Actions

DLTL is well suited for reasoning about actions because it combines the features of PDL for reasoning about complex actions, with those of LTL for expressing temporal properties. In this section we describe an action theory based on DLTL we have developed in [5], and used to reason about interaction protocols in multi-agent systems.

In our theory, a *domain description* $D$ is defined as a tuple $(\Pi, \mathcal{C})$, where $\Pi$ is a set of *action laws* and *causal laws*, and $\mathcal{C}$ is a set of *constraints*.

*Action laws* in $\Pi$ have the form: $\Box(\alpha \rightarrow [a]\beta)$, with $a \in \Sigma$ and $\alpha, \beta$ arbitrary formulas, meaning that executing action $a$ in a state where precondition $\alpha$ holds causes the effect $\beta$ to hold.

*Causal laws* in $\Pi$ have the form: $\Box((\alpha \wedge \bigcirc\beta) \rightarrow \bigcirc\gamma)$, meaning that if $\alpha$ holds in a state and $\beta$ holds in the next state, then $\gamma$ also holds in the next state. Such laws are intended to expresses "causal" dependencies among fluents, where fluents are atomic propositions which can change their truth value from one state to another.

*Constraints* in $\mathcal{C}$ are arbitrary temporal formulas of DLTL. In particular, the set of constraints $\mathcal{C}$ includes the precondition laws.

*Precondition laws* have the form: $\Box(\alpha \rightarrow [a]\bot)$, meaning that the execution of an action $a$ is not possible if $\alpha$ holds (i.e. there is no resulting state following the execution of $a$ when $\alpha$ holds). Observe that, when there is no precondition law for an action, the action is executable in all states.

Action laws and causal laws describe the changes to the state. All *fluents* which are not changed by the actions or causal laws are assumed to persist unaltered to the next state. To cope with the *frame problem*, the laws in $\Pi$, describing the (immediate and ramification) effects of actions, have to be distinguished from the constraints in $\mathcal{C}$ and given a special treatment. In [5], to deal with the frame problem, a completion construction is defined which, given a domain description, introduces frame axioms in the style of the successor state axioms introduced by Reiter [15] in the context of the situation calculus. The completion construction is applied only to the action laws and causal laws in $\Pi$ and not to the constraints. In the following we call $Comp(\Pi)$ the completion of a set of laws $\Pi$ and we refer to [5] for the details on the completion construction.

In [6, 7] we have used this theory for specifying interaction protocols among agents by adopting a social approach to agent communication [16], in which communicative actions affect the "social state" of the system, rather than the internal (mental) states of the agents. The social state records social facts, like the *permissions* and the *commitments* of the agents. The dynamics of the system emerges from the interactions of the agents, which must respect these permissions and commitments (if they are compliant with the protocol). The social approach allows a high level specification of the protocol, and does not require the rigid specification of the allowed action sequences. It is well suited for dealing with "open" multi-agent systems, where the history of communications is observable, but the internal states of the single agents may not be observable.

Commitments are represented by special fluents: a *base-level commitments* of the form $C(i, j, \alpha)$, means that agent $i$ is committed to agent $j$ to bring about $\alpha$, where $\alpha$ is an arbitrary formula, and a *conditional commitments* of the form $CC(i, j, \beta, \alpha)$ means that agent $i$ is committed to agent $j$ to bring about $\alpha$, if the condition $\beta$ is brought about.

Some reasoning rules have been defined for canceling commitments when they have been fulfilled and for dealing with conditional commitments, by means of *causal laws*:

$\Box(\bigcirc\alpha \rightarrow \bigcirc\neg C(i,j,\alpha))$
$\Box(\bigcirc\alpha \rightarrow \bigcirc\neg CC(i,j,\beta,\alpha))$
$\Box((CC(i,j,\beta,\alpha) \wedge \bigcirc\beta) \rightarrow \bigcirc(C(i,j,\alpha) \wedge \neg CC(i,j,\beta,\alpha)))$

A commitment (or a conditional commitment) to bring about $\alpha$ is cancelled when $\alpha$ holds, and a conditional commitment $CC(i,j,\beta,\alpha)$ becomes a base-level commitment $C(i,j,\alpha)$ when $\beta$ has been brought about.

For instance, let us consider the *Contract Net* protocol[18]. The Contract Net protocol begins with an agent (the manager) broadcasting a task announcement (call for proposals) to other agents viewed as potential contractors (the participants). Each participant can reply by sending either a proposal or a refusal. The manager must send an accept or reject message to all those who sent a proposal. When a contractor receives an acceptance it is committed to perform the task.

The effects of communicative actions are described by *action laws*, such as:

$\Box[cfp(T)]CC(M,P,proposal,acc\_rej)$
$\Box[propose](replied \wedge proposal)$

The law for action $cfp(T)$ adds to the state the information that the manager is committed to reply to the task announcement, by accepting or rejecting a proposal, if it receives a proposal.

The permissions to execute communicative actions are represented by precondition laws. For instance, preconditions on the execution of action *accept* can be expressed as:

$\Box(\neg proposal \vee acc\_rej \rightarrow [accept]\bot)$

meaning that action *accept* cannot be executed if a proposal has not been done, or if the manager has already replied.

We will denote $Perm_i$ (permissions of agent i) the set of all the precondition laws of the protocol pertaining to the actions of which agent $i$ is the sender.

We are interested in those execution of the protocol in which all commitments have been fulfilled. We can express the condition that the commitment $C(i,j,\alpha)$ has been fulfilled by the following constraint:

$\Box(C(i,j,\alpha) \rightarrow \Diamond\alpha)$

We will call $Com_i$ the set of constraints of this kind for all commitments of agent $i$. $Com_i$ states that agent $i$ will fulfill all the commitments of which it is the debtor.

The domain description $D = (\Pi, \mathcal{C})$ of a protocol is defined as follows: $\Pi$ is the set of the action and causal laws, and $\mathcal{C} = Init \wedge \bigwedge_i (Perm_i \wedge Com_i)$ is the set containing the the initial state, the permissions $Perm_i$ and the commitments $Com_i$ of all the agents.

Given a domain description $D$, the completed domain description $Comp(D)$ is the set of formulas $(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i))$. The runs of the system according the protocol are the linear models of $Comp(D)$. Observe

that in these protocol runs all permissions and commitments have been fulfilled. By applying the automaton construction described in this paper to the formula $Comp(D)$ we obtain a Büchi automaton whose accepting runs are all correct executions of the protocol.

Given the DLTL specification of a protocol by a domain description, we can address different kinds of verification problems.

For instance, given an execution history describing the interactions of the agents, we want to verify the compliance of that execution to the protocol. We are given a history $\tau = a_1, \ldots, a_n$ of the communicative actions executed by the agents, and we want to verify that the history $\tau$ is the prefix of a run of the protocol, that is, it respects the permissions and commitments of the protocol. This problem can be formalized by requiring that the formula

$$(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i)) \wedge < a_1; a_2; \ldots; a_n > \top$$

(where $i$ ranges on all the agents involved in the protocol) is satisfiable. In fact, the above formula is satisfiable if it is possible to find a run of the protocol starting with the action sequence $a_1, \ldots, a_n$.

Note that this verification is carried out at runtime, and we can take advantage of the incremental nature of the construction. In fact we can carry out the construction on-the-fly whenever a new action is executed by some agent, until either the protocol is completed or the construction cannot go on, in case of violation of the protocol.

A further problem is to verify that an agent is compliant with the protocol, given the program executed by the agent itself. In DLTL we can specify the behavior of an agent by making use of complex actions (regular programs). Consider for instance the following program $\pi_P$ for a participant in the *Contract Net* protocol:

$$[\neg done?; ((cfp(T); eval\_task; (\neg ok?; refuse; exit + ok?; propose)) + (reject; exit) + (accept; do\_task; exit))]^*; done?$$

The participant cycles and reacts to the messages received by the manager: for instance, if the manager has issued a call for proposal, the participant can either refuse or make a proposal according to his evaluation of the task; if the manager has accepted the proposal, the participant performs the task; and so on. The program $\pi_P$ contain private actions of the participant, besides the public actions of the protocol.

The verification that the participant is compliant with the protocol can be formalized as a validity check. Let $D = (\Pi, \mathcal{C})$ be the domain description describing the protocol, and $(\Pi_P, \mathcal{C}_P)$ be the domain description of the participant, where $\Pi_P$ is a set of action laws describing the effects of its private actions, and $\mathcal{C}_P$ contains the formula $\langle \pi_P \rangle \top$ stating that the program is executable in the initial state. The formula

$$(Comp(\Pi) \wedge Init \wedge Perm_M \wedge Com_M \wedge Comp(\Pi_P) \wedge \mathcal{C}_P) \rightarrow (Perm_P \wedge Com_P)$$

is valid if in all the behaviors of the system, in which the participant executes its program $\pi_P$ and the manager (whose internal program is unknown) respects the protocol specification (in particular, its permissions and commitments), the permissions and commitment of the participant are also satisfied.

As we have seen, an agent participating in a protocol makes use of a set of actions different from that of the protocol or of other agents, since this set can contain also private actions. Thus, in the general case, the verification of properties of multi-agent systems may require the use of DLTL$^{\otimes}$ [6].

# 7 Related Work

In this paper we have presented a tableau-based algorithm for constructing a Büchi automaton from a DLTL formula. DLTL, and its product version DLTL$^{\otimes}$, were proposed by Henriksen and Thiagarajan [9, 10]. DLTL combines in an orthogonal way two well-known formalisms, LTL and PDL, by indexing the until operator of LTL with the programs of PDL.

As described in [10], the satisfiability problem for DLTL can be solved in deterministic exponential time, as for LTL, by constructing for each formula $\alpha \in$ DLTL($\Sigma$) a Büchi automaton $\mathcal{B}_\alpha$ such that the language of $\omega$-words accepted by $\mathcal{B}_\alpha$ is non-empty if and only if $\alpha$ is satisfiable. The construction presented in [10] is highly inefficient since it requires to build an automaton with an exponential number of states, most of which will not be reachable from the initial state.

A tableau-based algorithm for constructing a Büchi automaton has been proposed in [4] for LTL. The construction of the automaton can be done on-the-fly, while checking for the emptiness of the language accepted by the automaton. Given a formula $\varphi$, the algorithm builds a graph $\mathcal{G}(\varphi)$ whose nodes are labelled by sets of formulas. States and transitions of the Büchi automaton correspond to nodes and arcs of the graph. The algorithm makes use of a tableau-based procedure for expanding the set of formulas at each node. The algorithm builds a *generalized Büchi automaton*, which has a set of sets of accepting states and can easily be translated into a standard Büchi automaton. The number of states of the automaton is, in the worst case, exponential in the size if the input formula, but in practice it can be much smaller.

Algorithms of this kind are the kernel of model checkers like SPIN [11]. Due to its practical importance, various improvements have been presented [3, 19] to keep the automaton as small as possible.

The construction presented in this paper for DLTL is based on the standard construction for LTL [4], but the possibility of indexing until formulas with regular programs puts stronger constraints on the fulfillment of until formulas than in LTL, requiring more complex acceptance conditions. The solution adopted in this paper was inspired by [10], and allows to obtain directly a standard Büchi automaton.

DLTL has some similarities with ETL, an extension of LTL proposed by Vardi and Wolper [21, 23]. This logic does not provide any standard temporal connective, but extends propositional logic with formulas $A(\phi_1, \ldots, \phi_n)$, where

the $\phi_i$ are formulas, and $A = (\Sigma, S, \delta, S_0, F)$ is a finite automaton called *automaton connective*. Each formula $\phi_i$ corresponds to a symbol of the alphabet $\Sigma$, ($|\Sigma| = n$). A model is an infinite sequence of truth assignments, i.e. a function $\pi : \omega \rightarrow 2^{Prop}$. Roughly speaking, a formula $A(\phi_1, \ldots, \phi_n)$ is satisfied by $\pi$ at position $i$, iff there is a word $a_1, \ldots, a_m$ accepted by $A$ such that formula $\phi_j$, corresponding to the symbol $a_j$ is satisfied at position $i + j$, $1 \leq j \leq m$.

Various versions of ETL have been defined, depending on the acceptance condition of the automaton $A$. Standard temporal operators can be defined with suitable automata. Furthermore this logic is more expressive than PTL. For instance it is possible to express in ETL the property that a formula is true in every even state of the sequence $\pi$, while this property is not expressible in PTL.

Vardi and Wolper present in [21] a procedure for building a Büchi automaton from a given ETL formula. The automaton is obtained as the intersection of two automata: The first one, called the *local* automaton, is a Büchi automaton, while the other one, called the *eventuality* automaton, is a set-subword automaton. The authors give also a construction for obtaining a Büchi automaton from a set-subword automaton.

Wolper [23] presents a different version of ETL which uses grammars instead of automata, and gives a tableau decision procedure for it. This procedure is rather different from the one for LTL [4], due to the different features of the logics, in particular in the condition of eventuality fulfillment.

Although there are similarities between DLTL and ETL, and DLTL could be translated to ETL, we would like to quote from [10]: "ETL as formulated in [21] has an uncontrolled amount of *external* elements, in the sense that the states and the alphabets of the automata which are used to write down the automaton formulas have little to do with the logic under consideration", while "DLTL adds to the familiar mechanism of LTL an *orthogonal* and well-understood component, namely the language of regular expressions".

# 8 Conclusions

In this paper we have presented a tableau-based algorithm for constructing a Büchi automaton from a DLTL formula. The formula is satisfiable if the language recognized by the automaton is nonempty. The construction of the states of the automaton can be done on-the-fly during the search that checks for emptiness. As in [10] we make use of finite automata to verify the fulfillment of until formulas. However, the construction of the automaton given in [10] is based on the idea of generating all the (maximally consistent) sets of the subformulas of the initial formula. Moreover, rather then introducing the states of the finite automata in the global states of the Büchi automaton, we stay closer to the standard construction for LTL [4] and we detect the point of fulfillment of the until formulas by associating a finite automaton with each until formula (rather than a regular expression) and by keeping track of the evolution of the state of these (finite) automata during the expansion of temporal formulas.

This construction could be improved in various ways, in particular by adapting the techniques presented in [3]. That paper presents a general form of the algorithm, and gives optimized variants of it. The main idea is to reduce the number of formulas contained in a node of the graph, by keeping only elementary formulas. This allows to reduce the number of nodes of the graph. The same techniques can also be applied to our algorithm.

Further optimizations can be devised specifically for our algorithm. For instance, we can avoid the situation where a node contains two copies of the same until formula with different labels 0 and 1. In fact, let $\mathbf{T}^0 \alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ and $\mathbf{T}^1 \alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ be two until formulas contained in a node $n$ of the graph. If there is a path in the graph from $n$ to $n'$ which fulfills one formula, it will fulfill also the other one, since the two formulas are the same, apart from the label. Thus we can keep only one of the two formulas, and precisely the one whose $l$ label is equal to the value of $x$, since we know that it will be fulfilled within the end of the current l-sequence.

# 9 Acknowledgements

We want to thank the unknown referees for their helpful comments.

# References

[1] F. Bacchus and F. Kabanza, Planning for temporally extended goals, Annals of Mathematics and Artificial Intelligence 22 (1998) 5–27.

[2] D. Calvanese, G. De Giacomo and M.Y.Vardi, Reasoning about Actions and Planning in LTL Action Theories, in: *Proc. Principles of Knowledge Representation and Reasoning, KR'02*, (Morgan Kaufmann 2002) pp. 593–602.

[3] M. Daniele, F. Giunchiglia and M.Y. Vardi, Improved automata generation for linear temporal logic, in: *Proc. Computer Aided Verification, 11th International Conference, CAV'99*, Lecture Notes in Computer Science, Vol. 1633 (Springer 1999) pp. 249–260.

[4] R. Gerth, D. Peled, M.Y. Vardi and P. Wolper, Simple on-the-fly automatic verification of linear temporal logic, in: *Proc. 15th International Symposium on Protocol Specification, Testing and Verification XV, PSTV 1995* (IFIP Conference Proceedings 38 Chapman & Hall 1996) pp. 3–18.

[5] L. Giordano, A. Martelli, and C. Schwind, Reasoning about actions in dynamic linear time temporal logic, Logic Journal of the IGPL 9(2) (2001) 289–303.

[6] L. Giordano, A. Martelli, and C. Schwind, Specifying and Verifying Systems of Communicating Agents in a Temporal Action Logic, in: *Proc. AI*IA 2003: Advances in Artificial Intelligence, 8th Congress of the Italian Association for Artificial Intelligence*, Lecture Notes in Computer Science, Vol. 2829 (Springer 2003) pp. 262–274.

[7] L. Giordano, A. Martelli, and C. Schwind, Verifying Communicating Agents by Model Checking in a Temporal Action Logic, in: *Proc. Logics in Artificial Intelligence, 9th European Conference, JELIA 2004*, Lecture Notes in Computer Science, Vol. 3229 (Springer 2004) pp. 57-69.

[8] F. Giunchiglia and P. Traverso, Planning as Model Checking, in: *Proc. The 5th European Conference on Planning, ECP'99*, Lecture Notes in Computer Science, Vol. 1809 (Springer 2000) pp. 1–20.

[9] J.G. Henriksen and P.S. Thiagarajan, A Product Version of Dynamic Linear Time Temporal Logic, in: *Proc. CONCUR '97: Concurrency Theory, 8th International Conference*, Lecture Notes in Computer Science, Vol. 1243 (Springer 1997) pp. 45–58.

[10] J.G. Henriksen and P.S. Thiagarajan, Dynamic Linear Time Temporal Logic, Annals of Pure and Applied Logic 96 (1-3) (1999) 187–207.

[11] G.J. Holzmann, The model checker SPIN, IEEE Trans. on Software Engineering 23 (5) (1997) 279–295.

[12] J. Hromkovic, S. Seibert and T. Wilke, Translating Regular Expressions into Small $\varepsilon$-Free Nondeterministic Finite Automata, in: *Proc. STACS 97, 14th Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, Vol. 1200 (Springer 1997) pp. 55–66.

[13] W. Penczek and A. Lomuscio, Verifying Epistemic Properties of Multi-agent Systems via Bounded Model Checking, Fundamenta Informaticae 55(2) (2003) 167–185.

[14] M.Pistore and P.Traverso, Planning as Model Checking for Extended Goals in Non-deterministic Domains, in: *Proc. of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001* (Morgan Kaufmann 2001) pp.479-484.

[15] R. Reiter, The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, V. Lifschitz, ed., (Academic Press 1991) 359–380.

[16] M. P. Singh, Agent communication languages: Rethinking the principles, IEEE Computer 31(12) (1998) 40–47.

[17] A. P. Sistla and E. M. Clarke, The complexity of propositional linear temporal logic, Journal of the ACM 32 (1985) 733–749.

[18] R. G. Smith, The contract net protocol: High level communication and control in a distributed problem solver, IEEE Transactions on Computers C-29(12) (1980) 1104–1113.

[19] F. Somenzi and R. Bloem, Efficient Büchi automata from LTL formulae, in: *Proc. Computer Aided Verification, 12th International Conference, CAV 2000*, Lecture Notes in Computer Science, Vol. 1855 (Springer 2000) pp. 247–263.

[20] W. van der Hoek and M.J.W. Wooldridge, Cooperation, Knowledge, and Time: Alternating-time Temporal Epistemic Logic and its Applications, Studia Logica 75:1 (2003) 125–157.

[21] M. Vardi and P. Wolper, Reasoning about infinite computations, Information and Computation 115 (1994) 1–37.

[22] M. Wooldridge, M. Fisher, M.P. Huget and S. Parsons, Model Checking Multi-Agent Systems with MABLE, in: *Proc. First International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2002* (ACM 2002) pp. 952–959.

[23] P. Wolper, Temporal logic can be more expressive, Information and Control 56 (1983) 72–99.

[24] P. Wolper, Constructing Automata from Temporal Logic Formulas: A Tutorial, in: *Lectures on Formal Methods and Performance Analysis FMPA 2000*, Lecture Notes in Computer Science, Vol. 2090 (Springer 2001) pp. 261–277.