Deciding Distributed Temporal Logic

Catarina Inácio Marques Neves

Instituto Superior Técnico, Lisbon, Portugal

December 2015

Abstract

The purpose of this dissertation is to implement a tableaux system for Distributed Temporal Logic [3] (DTL) and an algorithm to check global invariants in a system described with this version of DTL.

The tableaux system and the algorithm implemented had already been proposed in [2], but they were not implemented to experiment with, and this was the main motivation of this work.

Herein we include all the necessary definitions to understand the DTL version and the tableaux system that we use. We also present all the necessary results and theorems to understand the algorithm that we have implemented and how it works.

Keywords: Distributed Temporal Logic, tableaux system, invariants.

1. Introduction

The Distributed Temporal Logic referred in the Abstract is a type of Temporal Logic and was first proposed in [3]. One of the advantages of these logics is their characterisation of the progress of time. When dealing with a Temporal Logic one can describe a system dynamically. In order to describe the evolution of a system one uses the temporal operators of the logic to do it, describing the evolution of the system as the time progresses.

In particular, one can use a Temporal Logic that involves events and agents. In general, the occurrence of an event changes the state of the system. An event usually occurs between two entities, often called agents. Events can happen simultaneously and can involve only one, or more agents.

Temporal Logic can reflect the future and the past events. Herein we consider a logic that only reflects the future events.

In Section 2 we present the DTL and in Section 3 we present a tableaux system for DTL. In section 4 we present the algorithm for checking invariants, and in Section 5 we present the implementation of the tableaux system, and the details of implementation of the algorithm presented previously. Finally in section 6 we give our conclusions on this work.

2. Distributed Temporal Logic

Distributed Temporal Logic (DTL) is a Temporal Logic created with the purpose of reasoning about temporal properties of distributed systems, from the local point of view of its agents.

Herein we study its syntax and semantics. The version of DTL here used is heavily based on the technical report [2].

DTL is also used to describe concurrent systems, representing sequences of events for each agent, not needing to have global sequences of actions. The actions of an agent and the way they affect other agents are characterised as the dependencies between agents.

The agents interact with each other by sharing their events, which is called synchronous event sharing.

2.1. Sintax

In order to present DTL we first need to define its signature.

Definition 1 (DTL-signature)**.** A DTL-signature is a pair

$$\Sigma = \langle Id, \{\mathrm{Prop}_i\}_{i \in Id} \rangle,$$

where Id is a finite, non-empty set and Prop_i is a set of propositional symbols.

Id is the set of agent identifiers, each element of Id represents an agent of the system. For each $i \in Id$, Prop_i is the set of local state propositional symbols, used to describe the state of each agent.

The language of DTL is divided in a global and a local one. Herein, we will define the global language $\mathcal{L}$ and the local languages $\mathcal{L}_i$ of each agent $i \in Id$.

Definition 2 (Global Language)**.** The global DTL-language is denoted by $\mathcal{L}$ and can be defined as:

$$\begin{aligned}\mathcal{L} ::= {} & \neg\mathcal{L} \mid \mathcal{L} \wedge \mathcal{L} \mid \mathcal{L} \vee \mathcal{L} \mid \\ & \mathcal{L} \Rightarrow \mathcal{L} \mid \mathcal{L} \Leftrightarrow \mathcal{L} \mid @_i[\mathcal{L}_i] \mid False \mid True,\end{aligned}$$

with $i \in Id$.

Informally, one can say that the meaning of the symbols presented in this definition are the usual ones. However, a new symbol is here invoked, the symbol @. This symbol is a DTL operator that indicates to each agent the following property refers to, for example $@_i[\alpha]$ means that the agent i has the property α.

Note that there are no temporal operators in the global language. All the temporal operators are defined in the local language. As mentioned in the Introduction, herein the temporal operators reflect only future events. These operators are F, G, X, U and have their usual meaning.

In the following definition there are also some common propositional connectives, such as $\neg$, $\wedge$, $\vee$, $\Rightarrow$ and $\Leftrightarrow$, also with their usual meaning. As for the new symbol ©, it is the *communication* operator and it represents the communication between two agents. For example, $@_i[©_j[\varphi]]$ means that "agent i started an interaction with agent j and said φ to him".

Definition 3 (Local Language)**.** The local language $\mathcal{L}_i$ of each agent $i \in Id$ is defined as follows:

$$\mathcal{L}_i ::= \neg\mathcal{L}_i \mid \mathcal{L}_i \wedge \mathcal{L}_i \mid \mathcal{L}_i \vee \mathcal{L}_i \mid \mathcal{L}_i \Rightarrow \mathcal{L}_i \mid \mathcal{L}_i \Leftrightarrow \mathcal{L}_i \mid \mathrm{X}[\mathcal{L}_i] \mid \mathrm{G}[\mathcal{L}_i] \mid \mathrm{F}[\mathcal{L}_i] \mid \mathcal{L}_i\mathrm{U}\mathcal{L}_i \mid ©_j[\mathcal{L}_i] \mid \mathrm{Prop}_i \mid False \mid True,$$

with $j \in Id$ and $j \neq i$.

In order to more easily identify if we are referring to a local formula or a global one, we have decided to uniform its notation. Global formulas, will always be denoted by α, β or δ and φ, ψ will denote local formulas.

2.2. Semantics

The interpretation structures of DTL are labelled distributed life-cycles: $\mu = \langle \lambda, \vartheta \rangle$, where λ consists of a distributed life-cycle and ϑ a family of local labelling functions.

Definition 4 (Local life-cycle)**.** A local life-cycle of an agent $i \in Id$ is a countable infinite, discrete and well-founded total order λ_i.

$\lambda_i = \langle Ev_i, \leq_i \rangle$, where Ev_i is the set of local events and $\leq_i$ the local order of causality.

Definition 5 (Local successor relation)**.** The relation $\rightarrow_i \subseteq Ev_i \times Ev_i$ is the relation where $e \rightarrow_i e'$ if $e <_i e'$ and there is no e'' such that $e <_i e'' <_i e'$.

As a consequence, $\leq_i$ is the reflexive and transitive closure of $\rightarrow_i$, that is $\leq_i = \rightarrow_i^*$.

A local life-cycle (of an agent $i \in Id$) is therefore a countable infinite and well-founded total order.

Definition 6 (Distributed life-cycle)**.** A distributed life-cycle is a family of local life cycles, $\lambda = \{\lambda_i\}_{i \in Id}$.

The communication is modelled by synchronous event sharing, and thus for some event e we may have $e \in Ev_i \cap Ev_j$, with $i \neq j$. $\leq = (\cup_{i \in Id} \leq_i)^*$ defines a partial order of global causality on the set of all events $Ev = \cup_{i \in Id} Ev_i$.

Definition 7 (Local state)**.** The local state of an agent i is a finite set $\xi_i \subseteq \mathrm{Ev}_i$ such that if $e \leq e'$ and $e' \in \xi_i$ then also $e \in \xi_i$.

Ξ_i is the set of all local states of an agent i. This set is totally ordered by inclusion and has $\emptyset$ as the minimal element.

The first local state of each agent is always the $\emptyset$, and each other state is reached by the occurrence of an event, let us call this event the $last(\xi_i)$, as it is the last event in which agent i took part in order to reach the present state ξ_i. The k-th state of agent i is denoted by ξ_i^k. The initial state ξ_i^0 is $\emptyset$ and ξ_i^k is reached from the initial state after the occurrence of k events.

Definition 8 (Local satisfaction relation)**.** Given a local interpretation structure μ_i and a local state ξ_i then:

- $\mu_i, \xi_i \Vdash_i True$;
- $\mu_i, \xi_i \nVdash_i False$;
- $\mu_i, \xi_i \Vdash_i \mathrm{p}$ if $\mathrm{p} \in \vartheta_i(\xi_i)$;
- $\mu_i, \xi_i \Vdash_i \neg\varphi$ if $\mu_i, \xi_i \nVdash_i \varphi$;
- $\mu_i, \xi_i \Vdash_i \varphi \Rightarrow \psi$ if $\mu_i, \xi_i \nVdash_i \varphi$ or $\mu_i, \xi_i \Vdash_i \psi$;
- $\mu_i, \xi_i \Vdash_i \varphi \Leftrightarrow \psi$ if $\mu_i, \xi_i \Vdash_i \varphi \Rightarrow \psi$ and $\mu_i, \xi_i \Vdash_i \psi \Rightarrow \varphi$;
- $\mu_i, \xi_i \Vdash_i \varphi \wedge \psi$ if $\mu_i, \xi_i \Vdash_i \varphi$ and $\mu_i, \xi_i \Vdash_i \psi$;
- $\mu_i, \xi_i \Vdash_i \varphi \vee \psi$ if $\mu_i, \xi_i \Vdash_i \varphi$ or $\mu_i, \xi_i \Vdash_i \psi$;
- $\mu_i, \xi_i \Vdash_i \mathrm{X}[\varphi]$ if there is $e \in \mathrm{Ev}_i$ such that $\xi_i \cup \{e\} \in \Xi_i$ and $\mu_i, \xi_i \cup \{e\} \Vdash_i \varphi$;
- $\mu_i, \xi_i \Vdash_i \mathrm{G}[\varphi]$ if $\mu_i, \xi_i' \Vdash_i \varphi$, for every $\xi_i' \in \Xi_i$ such that $\xi_i \subseteq \xi_i'$;
- $\mu_i, \xi_i \Vdash_i \mathrm{F}[\varphi]$ if $\mu_i, \xi_i' \Vdash_i \varphi$, for a $\xi_i' \in \Xi_i$ such that $\xi_i \subseteq \xi_i'$;
- $\mu_i, \xi_i \Vdash_i ©_j[\varphi]$ if $\xi_i \neq \emptyset$, $\mathrm{last}(\xi_i) \in \mathrm{Ev}_j$ and μ_j, $\mathrm{last}(\xi_i){\downarrow_j} \Vdash_j \varphi$;
- $\mu_i, \xi_i \Vdash_i \varphi\mathrm{U}\psi$ if there is $\xi_i' \in \Xi_i$, such that $\xi_i \subseteq \xi_i'$ and $\mu_i, \xi_i' \Vdash_i \psi$, and $\mu_i, \xi_i'' \Vdash_i \varphi$ for every $\xi_i'' \in \Xi_i$ such that $\xi_i \subseteq \xi_i'' \subset \xi_i'$.

It is said that μ_i *(locally) satisfies* φ, written $\mu_i \Vdash_i \varphi$, if $\mu_i, \emptyset \Vdash_i \varphi$.

Now that the local satisfaction is defined, we need to establish the definition of global satisfaction of DTL-formulas. In order to do it, we first need to introduce the notion of global state.

Definition 9 (Global state)**.** A global state is the set $\xi \subseteq \mathrm{E}v$, such that $e \leq e'$ and $e' \in \xi$ then $e \in \xi$. Every global state ξ includes the local states $\xi|_i = \xi \cap \mathrm{E}v_i$.

Definition 10 (Global satisfaction relation)**.** Given a global interpretation structure μ and a global state ξ the global satisfaction relation is defined as:

- $\mu, \xi \Vdash True$;
- $\mu, \xi \nVdash False$;
- $\mu_i, \xi_i \Vdash_i \neg\alpha$ if $\mu_i, \xi_i \nVdash_i \alpha$;
- $\mu, \xi \Vdash \alpha \Rightarrow \beta$ if $\mu, \xi \Vdash \alpha$ or $\mu, \xi \Vdash \beta$;
- $\mu, \xi \Vdash \alpha \Leftrightarrow \beta$ if $\mu, \xi \Vdash \alpha \Rightarrow \beta$ and $\mu, \xi \Vdash \beta \Rightarrow \alpha$;
- $\mu, \xi \Vdash \alpha \wedge \beta$ if $\mu, \xi \Vdash \alpha$ and $\mu, \xi \Vdash \beta$;
- $\mu, \xi \Vdash \alpha \vee \beta$ if $\mu, \xi \Vdash \alpha$ or $\mu, \xi \Vdash \beta$;
- $\mu, \xi \Vdash @_i[\varphi]$ if $\mu_i, \xi|_i \Vdash_i \varphi$.

$\mu \Vdash \alpha$ read μ *(globally) satisfies* α, if $\mu, \emptyset \Vdash \alpha$. From another perspective, α is said to be *satisfiable*, whenever there is a global interpretation structure μ such that $\mu \Vdash \alpha$.

α is said to be *valid* and written $\vDash \alpha$, if $\mu \Vdash \alpha$ for every global interpretation structure μ.

The following definitions and propositions are necessary to understand the tableaux system proposed in [2] and here used.

Definition 11 (Atomic global formula)**.** A formula α is said to be atomic global if it has no strict global subformulas. That is, if it is either $False$, $True$ or $@_i[\varphi]$, for some $\varphi \in \mathcal{L}_i$ and $i \in Id$.

A formula is called atomic if the $\varphi \in \mathcal{L}_i$ is of the form $\mathrm{p} \in \mathrm{Prop}_i$.

Definition 12 (State Formula)**.** A state formula is either an atomic formula or a formula $@_i[\mathrm{X}[\varphi]]$, for some $\varphi \in \mathcal{L}_i$ and $i \in Id$.

The set of subformulas of a formula α is obtained with the application of either *Subf* or *GlobSubf*, depending on which type of formula we are dealing with.

Herein, the subformula meaning is not only to be understood as the traditional one, but also as a state formula. This happens when we are dealing with a formula with G, F or U operators:

- $\mathit{Subf}(@_i[\mathrm{G}[\varphi]]) = \{@_i[\mathrm{G}[\varphi]], @_i[\mathrm{X}[\mathrm{G}[\varphi]]]\} \cup \mathit{Subf}(@_i[\varphi])$;
- $\mathit{Subf}(@_i[\mathrm{F}[\varphi]]) = \{@_i[\mathrm{F}[\varphi]], @_i[\mathrm{X}[\mathrm{F}[\varphi]]]\} \cup \mathit{Subf}(@_i[\varphi])$;
- $\mathit{Subf}(@_i[\varphi \mathrm{U} \psi]) = \{@_i[\varphi \mathrm{U} \psi], @_i[\mathrm{X}[\varphi \mathrm{U} \psi]]\} \cup \mathit{Subf}(@_i[\varphi]) \cup \mathit{Subf}(@_i[\psi])$.

The other cases are similar to the usual ones. We present next the definition for the © operator that is the only "new" case.

$$\mathit{Subf}(@_i[©_j[\varphi]]) = \{@_i[©_j[\varphi]]\} \cup \mathit{Subf}(@_j[\varphi]).$$

Remark 1. The sets obtained by applying *Subf* and *GlobSubf* are always finite.

Remark 2. *Subf* and *GlobSubf* on the right hand side are always applied to a simpler formula.

3. Tableaux system

The tableaux system for DTL we present here is based on the one proposed in [2], with minor differences in the form.

A tableau $\mathcal{T}$ is a directed graph whose nodes are state tuples, and the edges are defined according to specific rules, with the proviso that exactly one rule is applied to each node, and no rule is applied to a contradictory state tuple.

The set of all state tuples (over a DTL-signature Σ) is denoted by $\mathcal{S}$.

A tableau for $\mathrm{Q}_0 \in \mathcal{S}$ is a tableau whose construction starts with the state tuple Q_0 and proceeds with the application of rules from one state tuple to another, until no more rules can be applied.

A state tuple characterises all the relevant information of a global state of the system.

Definition 13 (State tuple)**.** A state tuple is a quintuple

$$\mathrm{Q} = \langle \mathrm{Q}^+, \mathrm{Q}^-, \mathrm{Q}^{\bowtie}, \mathrm{Q}^{\#}, \mathrm{Q}^{\mathrm{X}} \rangle,$$

where Q^+, $\mathrm{Q}^- \subseteq \mathcal{L}$; $\mathrm{Q}^{\bowtie}$, $\mathrm{Q}^{\#} \subseteq Id \times Id$ and $\mathrm{Q}^{\mathrm{X}} \subseteq Id$.

As defined above, Q^+ and Q^- are sets of formulas, the first set is composed by the formulas that are intended to be true and the second by formulas that are intended to be false.

The set of forced synchronizations, $\mathrm{Q}^{\bowtie}$, indicates which agents need to be synchronized in order to reach the current state.

On the other hand, the elements of $\mathrm{Q}^{\#}$, the set of conflicts, indicate which agents cannot synchronize when advancing to the current state.

The identifiers in Q^{X} indicate which agents synchronized in order to reach the current state.

The sets $\mathrm{Q}^{\bowtie}$, $\mathrm{Q}^{\#}$ and Q^{X} are referred to as the synchronization requirements of Q. Q is a state tuple with no synchronization requirements when: $\mathrm{Q}^{\bowtie} = \mathrm{Q}^{\#} = \mathrm{Q}^{\mathrm{X}} = \emptyset$.

Definition 14 ($\hat{Q}$)**.** Given a state tuple Q, we denote by $\hat{Q}$ the formula

$$\bigwedge_{\alpha \in \mathrm{Q}^+} \alpha \wedge \bigwedge_{\beta \in \mathrm{Q}^-} \neg\beta,$$

where the empty conjunctions are defined to be *True*.

Definition 15 (State)**.** A state tuple is called state if all its formulas are state formulas. A state tuple rises to the statute of global state if all its formulas are global state formulas.

As the sets Q^+ and Q^- have a semantic meaning it can be the case that we have created a state tuple that has a contradictory meaning, an easy example is if we have a formula in both Q^+ and Q^-, this leads to an obvious contradictory state tuple. This is not the only scenario where we get a contradictory state tuple.

Definition 16 (Contradictory state tuple)**.** A state tuple is said to be contradictory if and only if any of the following conditions holds:

($\perp$1) $False \in \mathrm{Q}^+$ or $@_i[False] \in \mathrm{Q}^+$, for some $i \in Id$;

($\perp$2) $True \in \mathrm{Q}^-$ or $@_i[True] \in \mathrm{Q}^-$, for some $i \in Id$;

($\perp$3) $\mathrm{Q}^+ \cap \mathrm{Q}^- \neq \emptyset$;

($\perp$4) $i \notin \mathrm{Q}^{\mathrm{X}}$ or $j \notin \mathrm{Q}^{\mathrm{X}}$, for some $\langle i, j\rangle \in \mathrm{Q}^{\bowtie}$;

($\perp$5) $i \in \mathrm{Q}^{\mathrm{X}}$ and $j \in \mathrm{Q}^{\mathrm{X}}$, for some $\langle i, j\rangle \in \mathrm{Q}^{\#}$.

As introduced previously, a rule provides a way of connecting two state tuples.

Thinking in the tableau as a directed graph, the rule is represented by the edge that connects the nodes. Each rule is composed by its premises and conclusion. The source node of the edge corresponds to the conclusion, and is connected by the rule to one or more premisses, which are the end nodes of the edges.

The rules preserve the notion of subformula in the sense that if Q is the conclusion of a rule and Q′ is a premiss, then $\mathrm{Q}'^+ \cup \mathrm{Q}'^- \subseteq subf(\mathrm{Q}^+ \cup \mathrm{Q}^-)$. This means that the formulas in the premiss are always subformulas of the formulas in the conclusion. Recalling Remark 2, we can state that after applying a rule a simpler state tuple is obtained.

In general, for each operator there are two rules defined, the one applied if the formula is in Q^+ and the one applied if the formula is in Q^-. The only exception is the rule for the X operator, which is a special case.

The existence of a global and a local language forces the existence of global and local rules.

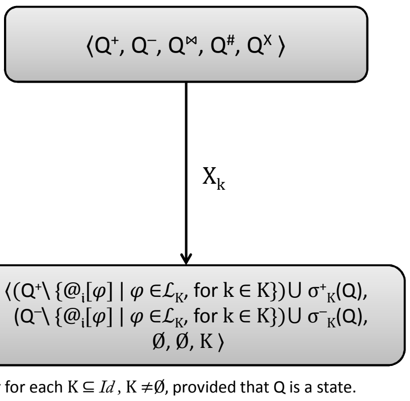

$\langle \mathrm{Q}^+, \mathrm{Q}^-, \mathrm{Q}^{\bowtie}, \mathrm{Q}^{\#}, \mathrm{Q}^{\mathrm{X}} \rangle$
X_k
$\langle (\mathrm{Q}^+ \setminus \{@_i[\varphi] \mid \varphi \in \mathcal{L}_K, \text{for } k \in K\}) \cup \sigma^+_K(\mathrm{Q}),$
$(\mathrm{Q}^- \setminus \{@_i[\varphi] \mid \varphi \in \mathcal{L}_K, \text{for } k \in K\}) \cup \sigma^-_K(\mathrm{Q}),$
$\emptyset, \emptyset, K \rangle$
one sucessor for each $K \subseteq Id$, $K \neq \emptyset$, provided that Q is a state.

Figure 1: Rules for the X operator.

The global rules, are the usual ones, the ones used in propositional tableaux systems. As for the local ones, we will present the rules for the temporal operators G and X and for the ⓒ operator.

For each operator there are two rules defined, the one applied if the formula is in Q^+ and the one applied if the formula is in Q^-. The only exception is the rule for the X operator, presented in Figure 1, which is applied to do a state transition.

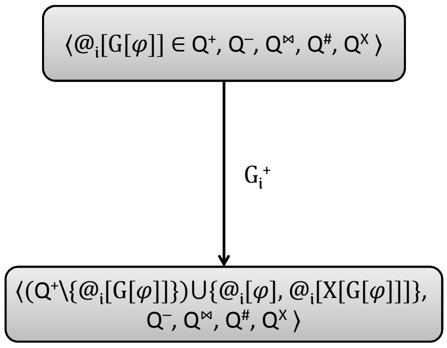
In this rule we use the auxiliary functions σ defined bellow.

Definition 17 (Auxiliary functions σ)**.** For each set of identifiers $\mathrm{I} \subseteq Id$, let $\sigma^+_{\mathrm{I}}, \sigma^-_{\mathrm{I}} : \mathcal{S} \to 2^{\mathcal{L}}$ be functions such that, for each $\mathrm{Q} \in \mathcal{S}$:

$$\sigma^+_{\mathrm{I}} = \{@_i[\varphi] \mid @_i[\mathrm{X}[\varphi]] \in \mathrm{Q}^+ \text{ and } i \in \mathrm{I}\};$$
$$\sigma^-_{\mathrm{I}} = \{@_i[\varphi] \mid @_i[\mathrm{X}[\varphi]] \in \mathrm{Q}^- \text{ and } i \in \mathrm{I}\}.$$

These functions return the set of formulas free from the *next* operator, given that the formula is in the correspondent Q.

The rules for the G operator, presented in Figure 2, serve as example of the rules for a temporal operator, the rules for U and F operators are similar, and can be checked in the original work [2].


$\langle @_i[\mathrm{G}[\varphi]] \in \mathrm{Q}^+, \mathrm{Q}^-, \mathrm{Q}^{\bowtie}, \mathrm{Q}^{\#}, \mathrm{Q}^{\mathrm{X}} \rangle$
G_i^+
$\langle (\mathrm{Q}^+ \setminus \{@_i[\mathrm{G}[\varphi]]\}) \cup \{@_i[\varphi], @_i[\mathrm{X}[\mathrm{G}[\varphi]]]\},$
$\mathrm{Q}^-, \mathrm{Q}^{\bowtie}, \mathrm{Q}^{\#}, \mathrm{Q}^{\mathrm{X}} \rangle$

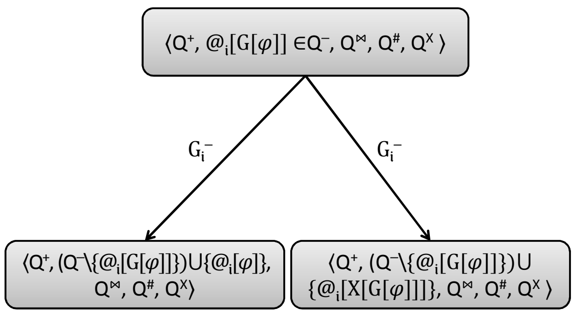

⟨Q+, @i[G[φ]] ∈Q−, Q⋈, Q#, QX ⟩
Gi−
Gi−
⟨Q+, (Q−\{@i[G[φ]]})∪{@i[φ]}, Q⋈, Q#, QX⟩
⟨Q+, (Q−\{@i[G[φ]]})∪ {@i[X[G[φ]]]}, Q⋈, Q#, QX ⟩

Figure 2: Rules for the G operator.

In Figure 3 we present the rules for the introduced © operator.

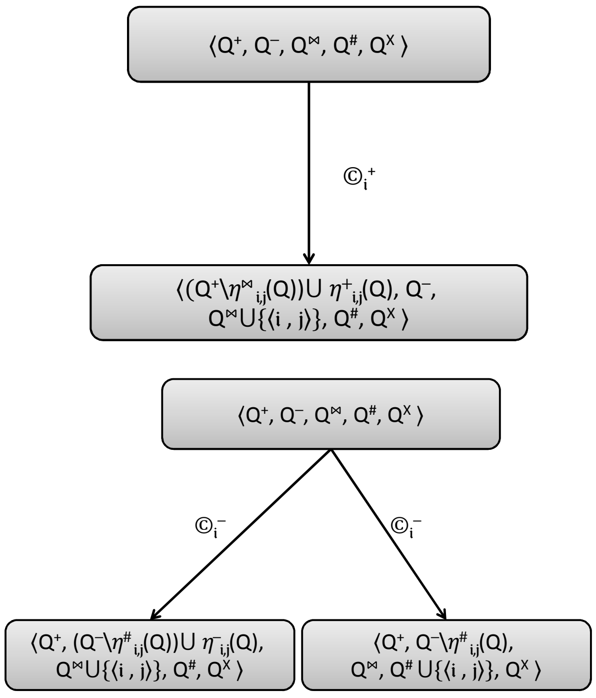

⟨Q+, Q−, Q⋈, Q#, QX ⟩
©i+
⟨(Q+\η⋈i,j(Q))∪ η+i,j(Q), Q−, Q⋈∪{⟨i , j⟩}, Q#, QX ⟩
⟨Q+, Q−, Q⋈, Q#, QX ⟩
©i−
©i−
⟨Q+, (Q−\η#i,j(Q))∪ η−i,j(Q), Q⋈∪{⟨i , j⟩}, Q#, QX ⟩
⟨Q+, Q−\η#i,j(Q), Q⋈, Q# ∪{⟨i , j⟩}, QX ⟩

Figure 3: Rules for the © operator.

In this rule we use the auxiliary functions η defined bellow.

Definition 18 (Auxiliary functions η)**.** For each $i, j \in Id$, let $\eta^{+}_{i,j}$, $\eta^{-}_{i,j}$, $\eta^{\bowtie}_{i,j}$, $\eta^{\#}_{i,j} : \mathcal{S} \to 2^{\mathcal{L}}$ be functions such that, for each $\mathrm{Q} \in \mathcal{S}$:

$$\begin{aligned}
\eta^{+}_{i,j} &= \{@_j[\varphi] \mid @_i[©_j[\varphi]] \in \mathrm{Q}^{+}\};\\
\eta^{-}_{i,j} &= \{@_j[\varphi] \mid @_i[©_j[\varphi]] \in \mathrm{Q}^{-}\};\\
\eta^{\bowtie}_{i,j} &= \{@_i[©_j[\varphi]] \mid @_i[©_j[\varphi]] \in \mathrm{Q}^{+}\};\\
\eta^{\#}_{i,j} &= \{@_i[©_j[\varphi]] \mid @_i[©_j[\varphi]] \in \mathrm{Q}^{-}\}.
\end{aligned}$$

The rules for all the operators, including the ones not referenced here (the local rules for the propositional connectives) can be checked in [5].

There is also a very important set, when dealing with the construction of the tableau, the set of closed nodes.

Definition 19 (Closed nodes)**.** The set of closed nodes of a tableau $\mathcal{T}$ is represented by $Cl_{\mathcal{T}}$ and is defined inductively:

(C1) $\mathrm{Q} \in Cl_{\mathcal{T}}$ for every contradictory node $\mathrm{Q} \in \mathcal{T}$;

(C2) $\mathrm{Q} \in Cl_{\mathcal{T}}$ whenever $\mathrm{Q}' \in Cl_{\mathcal{T}}$ for every successor node Q' of Q;

(C3) $\mathrm{Q} \in Cl_{\mathcal{T}}$ whenever $@_i[\mathrm{G}[\varphi]] \in \mathrm{Q}^{-}$ for some $i \in Id$ and $\varphi \in \mathcal{L}_i$, and every path from Q to nodes Q' with $@_i[\varphi] \in \mathrm{Q}'^{-}$ has a node in $Cl_{\mathcal{T}}$;

(C4) $\mathrm{Q} \in Cl_{\mathcal{T}}$ whenever there is $i \in \mathrm{Id}$ such that every path from Q to nodes Q' with $i \in \mathrm{Q}'^{\mathrm{X}}$ has a node in $Cl_{\mathcal{T}}$;

(C5) $\mathrm{Q} \in Cl_{\mathcal{T}}$ whenever $@_i[\mathrm{F}[\varphi]] \in \mathrm{Q}^{+}$, for some $i \in Id$ and $\varphi \in \mathcal{L}_i$, and every path from Q to a node Q' with $@_i[\varphi] \in \mathrm{Q}^{+}$ has a node in $Cl_{\mathcal{T}}$;

(C6) $\mathrm{Q} \in Cl_{\mathcal{T}}$ whenever $@_i[\varphi \mathrm{U} \psi] \in \mathrm{Q}^{+}$, for some $i \in Id$ and $\varphi, \psi \in \mathcal{L}_i$, and every path from Q to a node Q' with $@_i[\psi] \in \mathrm{Q}^{+}$ has a node in $Cl_{\mathcal{T}}$.

When the construction of the tableau is finished, no more rules can be applied, one may want to analyse the successfulness of the tableau.

Definition 20 (Successful tableau)**.** A tableau for Q_0 is unsuccessful if the node Q_0 is closed, and it is successful otherwise.

Reaching the end of the section, let us analyse the construction of tableaux and their finiteness:

- Global rules are always applied first - reducing the complexity of the formulas;
- Local rules (also reduce the complexity of the formulas and) do not cause the application of more global rules - the number of applications of global rules is always finite;
- When applying a local rule, we obtain subformulas of the original formulas, in a finite number, recall Remarks 1 and 2;
- The set of subformulas that can be generated from the application of local rules is finite - the number of applications of rules is finite;
- The state tuples created with the application of the rules, are always state tuples with lower complexity;

- The set of agent identifiers Id is finite, by definition, which means that the set of possible elements in $\mathrm{Q}^{\bowtie}, \mathrm{Q}^{\#}$ and Q^{X} is also finite.

With that stated, for a finite state tuple Q, the set of nodes of the tableau constructed for Q is always finite. We can derive one more remark:

Remark 3. A tableau for a finite state tuple is always finite.

Proposition 1 (Soundness). *Let Q be a state tuple with no synchronization requirements and $\mathcal{T}$ the tableau for Q. If $\hat{Q}$ is satisfiable then $\mathcal{T}$ is successful.*

Proposition 2 (Completeness). *Let Q be a state tuple with no synchronization requirements. If $\mathcal{T}$ is a successful tableau for Q then $\hat{Q}$ is satisfiable.*

The proofs can be found in [2].

From the previous results, Propositions 1 and 2, we can extract that our tableaux system is sound and complete, if we consider tableaux for state tuples with no synchronization requirements.

Piecing together these results with the finiteness of the tableaux obtained for any finite state tuple with no synchronization requirements (Remark 3), we achieve decidability. Therefore, the following propositions can be extracted.

Proposition 3 (Satisfiability). *The satisfiability problem for DTL is decidable.*

The decidability of DTL was already established in [1] but it involved a translation to Linear Temporal Logic. Proposition 3 establishes this result without translations.

Corollary 4 (Validity). *The validity problem for DTL is decidable.*

4. Checking global invariants

In this work we also want to verify if a set of properties of a system, described using DTL, is always satisfied. In order to do it we need to verify global invariants of the system.

Let Γ be a finite set of DTL-formulas, and δ a global formula (without temporal operators).

Our goal is to verify if for every interpretation structure μ that satisfies Γ, denoted by $\mu \Vdash \Gamma$, δ is a semantic consequence of Γ, that is if $\Gamma \vDash \delta$. δ is a semantic consequence of Γ if $\mu, \xi \Vdash \delta$, for every global state ξ of μ.

Our tableaux system does not directly capture this reasoning and one needs to implement some auxiliary procedures, as described in [2]. Three procedures are described therein: Ent, SetGlobal and LOCSAT.

The procedure Ent is used to check whether $\Gamma \vDash \delta$. This procedure outputs *yes* if $\Gamma \vDash \delta$ and outputs *no* and a counterexample if $\Gamma \nvDash \delta$.

The negative output is given when we can find an interpretation structure μ such that $\mu \Vdash \Gamma$, but $\mu,\ \xi \nVdash \delta$ for some global state ξ.

This is the procedure that we have implemented.

4.1. Description of the Algorithm

Ent starts by building a tableau $\mathcal{T}$ for $\mathrm{Q}_0 = \langle \Gamma,\ \emptyset,\ \emptyset,\ \emptyset,\ \emptyset \rangle$. Once constructed, it checks if the tableau is successful or not. If $\mathcal{T}$ is unsuccessful, by the Property 1 (soundness), $\hat{Q}$ is not satisfiable, so the output is obviously *yes*, as the antecedent of the implication is false.

If the tableau $\mathcal{T}$ is successful, we have to investigate further. We have to try to find an interpretation structure μ such that μ satisfies Γ but does not satisfy δ in some state of μ.

If we cannot find one, it means that $\Gamma \vDash \delta$, so the algorithm outputs *yes*. Otherwise, it is the case where $\Gamma \nvDash \delta$, this is the case where Ent outputs *no* together with the counterexample.

During the run of procedure Ent, it resorts to two other procedures also described in [2], SetGlobal and LocSat, both have as input a cut C and the formula δ. The definition of cut is not presented herein because of the space constraints, but can be consulted in [2].

The procedure SetGlobal constructs a stack of non contradictory extensions of the cut. This procedure is used, inside the procedure LocSat. LocSat is responsible for using the extensions obtained in SetGlobal and search for a global state that falsifies δ.

A cut is a set of states and agents' identifiers. An extension of a cut is the same form, but instead of states we have extensions, which are state tuples with extra formulas added.

4.2. Procedures Ent, SetGlobal, LocSat

Herein we present the pseudo-code for the three procedures, already referred, as they were presented in [2]. Firstly we present Ent in Figure 4. This procedure is the main one, responsible for the construction of the tableaux and the invocation of the procedure LocSat.

The procedure LocSat is presented in Figure 5. This procedure is invoked by Ent and invokes the procedure SetGlobal. From its result (a stack of elements from the input cut) it tries to build a counterexample, an interpretation structure that satisfies Γ but not δ.

The last procedure presented is SetGlobal and it can be checked in Figure 6. This procedure constructs a stack of non contradictory extensions of the cut inputed.

5. Implementation

We have implemented the tableaux system and the procedures presented above in *Mathematica* 10, so

```
ENT($\Gamma, \delta$)
Input: a set $\Gamma$ of global formulas and a global formula $\delta$.
Output: yes if $\Gamma \vDash \delta$, no and a counterexample, otherwise.

$Q_0 := \langle \Gamma, \emptyset, \emptyset, \emptyset, \emptyset \rangle$;
construct a tableau $\mathcal{T}$ for $Q_0$;
if $\mathcal{T}$ is unsuccessful then
    return yes;
else
    $O := \{Q$ in $\mathcal{T} \mid Q$ is not closed$\}$;
    $b :=$ false;
    while $O \neq \emptyset \wedge \neg b$ do
        let $Q \in O$;
        $O := O \setminus \{Q\}$;
        $T_Q := \mathcal{C}_Q$;
        while $T_Q \neq \emptyset \wedge \neg b$ do
            let $C \in T_Q$;
            $T_Q := T_Q \setminus \{C\}$;
            $\langle b_1, \mu, \xi \rangle :=$ LOCSAT($C, \delta$);
            $b := b \vee b_1$;
        end
    end
    if $\neg b$ then
        return yes;
    else
        return $\langle no, \mu, \xi \rangle$;
    end
end
```

Figure 4: Pseudo code of procedure ENT.

```
LOCSAT($C, \delta$)
Input: a cut $C$ and a global formula $\delta$.
Output: a triple $\langle b, \mu, \xi \rangle$.

$b :=$ false;
SETGLOBAL($C, \delta$);
while $S \neq$empty $\wedge \neg b$ do
    $D :=$ top($S$);
    if some $D_k$ with $k \in Id$ contains a non state formula then
        pop($S$);
        let $\delta'$ be a non state formula in $D_k$;
        case $\delta'$
            $@_i[\psi_1 \Rightarrow \psi_2] \in D_k^+$:
                $Q := \langle D_k^+ \setminus \{\delta'\}, D_k^- \cup \{@_i[\psi_1]\}, D_k^{\bowtie}, D_k^{\#}, D_k^{\mathsf{X}} \rangle$;
                if $\neg$ contradictory($Q$) then push($D[D_k := Q], S$) end
                $Q := \langle (D_k^+ \setminus \{\delta'\}) \cup \{@_i[\psi_2]\}, D_k^-, D_k^{\bowtie}, D_k^{\#}, D_k^{\mathsf{X}} \rangle$;
                if $\neg$ contradictory($Q$) then push($D[D_k := Q], S$) end
            $@_i[\psi_1 \Rightarrow \psi_2] \in D_k^-$:
                $Q := \langle D_k^+ \cup \{@_i[\psi_1]\}, (D_k^- \setminus \{\delta'\}) \cup \{@_i[\psi_2]\}, D_k^{\bowtie}, D_k^{\#}, D_k^{\mathsf{X}} \} \rangle$
                if $\neg$ contradictory($Q$) then push($D[D_k := Q], S$) end
            $@_i[\copyright_j[\psi]] \in D_k^+$:
                if $j \in D_k^{\mathsf{X}}$ then
                    $Q := \langle D_k^+ \setminus \{\delta'\} \cup \{@_j[\psi]\}, D_k^-, D_k^{\bowtie}, D_k^{\#}, D_k^{\mathsf{X}} \rangle$;
                    if $\neg$ contradictory($Q$) then push($D[D_k := Q], S$) end
                end
            $@_i[\copyright_j[\psi]] \in D_k^-$:
                if $j \in D_k^{\mathsf{X}}$ then
                    $Q := \langle D_k^+, D_k^- \setminus \{\delta'\} \cup \{@_j[\psi]\}, D_k^{\bowtie}, D_k^{\#}, D_k^{\mathsf{X}} \rangle$;
                    if $\neg$ contradictory($Q$) then push($D[D_k := Q], S$) end
                else
                    $Q := \langle D_k^+, D_k^- \setminus \{\delta'\}, D_k^{\bowtie}, D_k^{\#}, D_k^{\mathsf{X}} \rangle$;
                    push($D[D_k := Q], S$);
                end
    else
        $b :=$ true
    end
end;
if $b$ then
    return $\langle b, \mu^{\mathtt{top}(S)}, \xi^{n_{\mathtt{top}(S)}} \rangle$
else
    return $\langle b, \_, \_ \rangle$
end
```

Figure 5: Pseudo code of procedure LOCSAT.

in order to run the program one needs to have this version or higher.

Whenever possible we took advantage of the *Mathematica*'s built-in functions and structures. All the structures that we have implemented follow [2] as close as possible.

The modularity of the program was one of our big concerns. All the different structures needed were implemented in packages, this way each part can be improved, without affecting the other modules. The packages can be used in different programs, or with different tableaux systems, or variants of DTL or even for different algorithms. If changed, they will not affect the global behaviour of the program, as long as the functions' inputs, outputs and names are preserved. These informations are accessible to the regular user, as they are described in the public part of the package.

5.1. Tableaux system

The DTL formulas were implemented in the package `Formulas.m`. This package allows the user to construct and handle the formulas without needing to worry with how they are constructed.

The syntax that the user sees is the one presented in Definitions 2 and 3. We have protected the symbols `X`, `F`, `G`, `U`, @ and ©, this way the user can write the formulas as expected and previously defined.

The DTL formulas are used to construct state tuples, which is also a non built-in structure in *Mathematica*. We have implemented them in a different package.

The state tuples were defined in the package `StateTuple.m`, following the Definition 13. All the sets and pairs were implemented as *Mathematica*'s `Lists`. The two first sets are set of formulas, here `Lists` of `Formulas`, and the others set of agent's identifiers, here `Lists` of `Strings`.

When implementing the structure of the tableaux itself we took advantage of a *Mathematica* built-in structure, the `Graph`. Each node of the `Graph` will be a StateTuple.

Each `vertex` of the `Graph` will be a `StateTuple`, and each `edge` is constructed by the application of a `Rule`.

This decision also allowed us to work with the extensions of tableaux that might not respect the definition of tableau, but are still directed graphs. See [6] for the full code.

The packages presented in Figure 7 are the ones used to implement our tableaux system.

On the bottom of the figure there is the package `Formulas.m` which is essential as DTL-formulas are the constituents of the language. On top of the formulas we have the package of the state tuples, where we have the functions that build and operate

```
SETGLOBAL($C, \delta$)
Input: a cut $C$ and a global formula $\delta$.
Output: a (possibly empty) stack $S$ such that each of its e
contradictory extension of $C$.

$R$:=empty;
$S$:=empty;
push($\langle \emptyset, \{\delta\} \rangle, R$);
repeat
    $P$ := top($R$);
    pop($R$);
    if $P$ contains a non atomic global formula then
        let $\delta'$ be a a non atomic global formula in $P$;
        case $\delta'$
            $\delta'_1 \Rightarrow \delta'_2 \in P^+$:
                push($\langle P^+ \setminus \{\delta'\}, P^- \cup \{\delta'_1\} \rangle, R$);
                push($\langle (P^+ \setminus \{\delta'\}) \cup \{\delta'_2\}, P^- \rangle, R$);
            $\delta'_1 \Rightarrow \delta'_2 \in P^-$:
                push($\langle P^+ \cup \{\delta'_1\}, (P^- \setminus \{\delta'\}) \cup \{\delta'_2\} \rangle, R$);
    else
        if $\perp \notin P^+$ then
            $D := C$;
            for each $@_i[\varphi] \in P^+$
                $D_i^+ := D_i^+ \cup \{@_i[\varphi]\}$;
            for each $@_i[\varphi] \in P^-$
                $D_i^- := D_i^- \cup \{@_i[\varphi]\}$;
            if $\neg$contradictory($D$) then push($D, S$) end;
        end
    end
```

Figure 6: Pseudo code of procedure SETGLOBAL.

on `StateTuples`.

As explained, the tableau will be reasoned about as a graph, in fact as directed graph. That is why we have represented the `Rules` package on the right and not on top of the `Tableau.m` package. The `Rules.m` package will have the functions that build the tableaux depending on which formulas the state tuples have.

The next examples present some formulas in the DTL syntax and how they should be written in our program:

- $True$ - `True`;
- $@_i$[p] - `@`$_i$`[‘‘p’’]`;
- $\neg\alpha$ - $\neg\alpha$, where α has to be a global DTL-formula;
- $@_i$[X[φ]] - `@`$_i$`[X `φ`]`, assuming φ a well-formed formula.

In our implementation the construction of the tableaux is done trough the use of two functions `Tableau` and `Iterate`. The first creates the structure of the tableau and the second makes sure that the rules are applied correctly and in order. These

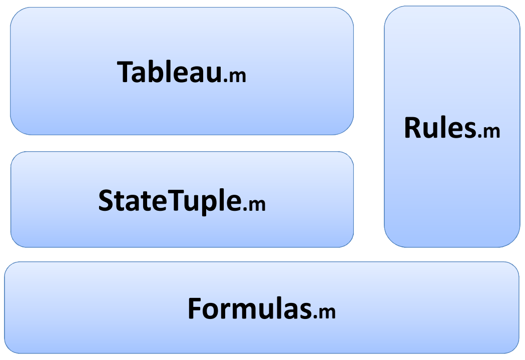


Figure 7: Sketch of the implementation of the tableaux system.

were the conditions explained in Section 3 and will be reflected in the function `Rules` from `Tableaux.m`.

The rules for all the operators are implemented in the package `Rules.m`, as defined in [5] and illustrated in Figures 1 to 3. The function `Rule` guarantees that the rules for the global formulas are applied first, only then the local ones, and when no other rule can be applied (and we are in the presence of a state) applies the rule for the X operator.

This package is almost independent from the tableau implementation, from the `Tableau` perspective the only thing that matters is that we give the right input to the `ApplyRule` function, the rest is up to the `Rules` package to handle.

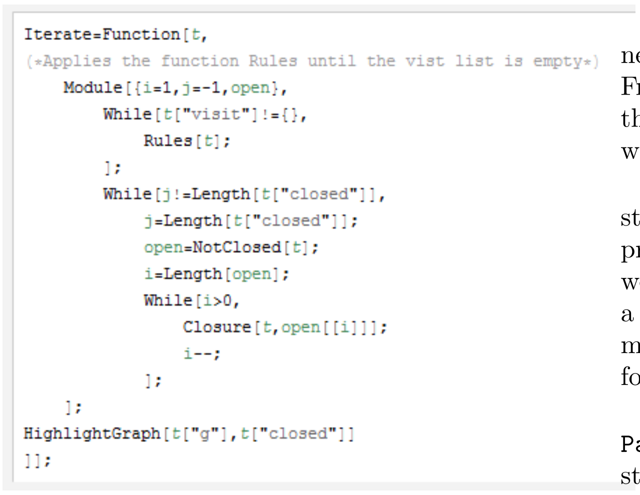
Herein we only present the implementation of the function `Iterate`, from the package `Tableau.m` in Figure 8. This function iterates the previously referred `Rules` function on the set of nodes that are "visitable", that is the nodes that are not closed, or to whom a rule was not applied yet.

The full code can be found in [6] along side with examples of the construction and manipulation of the formulas and tableaux.

5.2. Algorithm for checking invariants

The procedure ENT has two inputs: the set Γ of global formulas and a global formula δ. A `StateTuple` $Q0$ is built with the formulas in Γ, represented in the way that we have been doing herein $Q0 = \langle \Gamma, \emptyset, \emptyset, \emptyset, \emptyset \rangle$. Note that $Q0$ does not have synchronization requirements.

Now that the input is explained let us look at the first step of the algorithm. The algorithm will construct a `Tableau t`, using the `StateTuple Q0` as a source node. This will be done as previously explained. Then, the algorithm will analyse the successfulness of `t` by running the function `SuccessfulQ` from package `Tableau.m`, which will



```
Iterate=Function[t,
(*Applies the function Rules until the vist list is empty*)
    Module[{i=1,j=-1,open},
        While[t["visit"]!={},
            Rules[t];
        ];
        While[j!=Length[t["closed"]],
            j=Length[t["closed"]];
            open=NotClosed[t];
            i=Length[open];
            While[i>0,
                Closure[t,open[[i]]];
                i--;
            ];
    ];
HighlightGraph[t["g"],t["closed"]]
]];
```

Figure 8: Implementation of function `Iterate` from package `Tableau.m`.

return a boolean to answer the question.

The algorithm now establishes the successfulness of t with function `SuccesfullQ`. As described in 4.1, if the tableau is unsuccessful we can output `yes` and exit the computation. However, if the tableau is successful we need to construct cuts and extensions, to try to find an interpretation structure that serves as a counterexample.

We identify all the state tuples in t that are not closed. For each we construct the set of all cuts induced by it. In order to do this we will need two new structures: the `Cuts` and the `Paths`. For the extraction of the counterexample we also need a different structure: `Entailment`.

In Figure 9 we represent the modularity of these three packages.

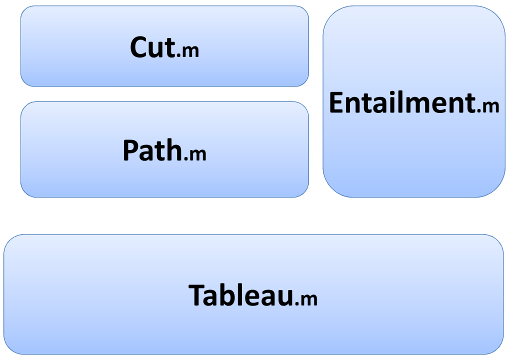


Figure 9: Sketch of some of the packages used by the algorithm.

These are the main packages and the structures necessary to do the following part of the algorithm. From the `Tableau` we will construct a path, with the function `Path` from the homonym package, and with it we will construct a `Cut`.

With the function `SetGlobal` we construct a stack with non contradictory extensions of the `Cut` provided as input. For the construction of the stack we need a structure and *Mathematica* does not have a built-in structure for stacks. We have implemented a `Stack` structure, adapted from the one found in [4].

The `Entailment.m` package works with the `Path.m` and `Cut.m` to construct an interpretation structure and a set of events that falsify (or not) the structure defined. Depending on the result the algorithm will either output `yes` or `no` with the counterexample created.

6. Conclusions

In this work we have studied the Distributed Temporal Logic [3] (DTL). We have implemented a tableaux system for DTL as well as an algorithm to check global invariants in systems specified with DTL, closely following the technical report [2].

We have presented all the relevant notions and results that were essential to understand the algorithm and the implementation.

We defined the logic that we were interested in the first place, the DTL. We presented its syntax and semantics. These definitions were essential to understand the algorithm that we have implemented, especially for those with no previous contact with DTL.

The tableaux deductive system was presented in Section 3. Besides the definition of tableaux, state tuple, and the some illustrative examples of the rules used to construct the tableaux, we also included some important results The tableaux system presented was slightly different from the one presented in [2]. We were not worried with the proofs of the results, but with the efficiency of the program, so we included all the operators and rules that we could, to facilitate the procedure and speed up the algorithm. We also stated, without proving the results on soundness, completeness and decidability of the system.

Finally in Section 5, we described the implementation, briefly analysing the procedures and the implemented structures that were used.

The algorithm was implemented in *Mathematica*. This decision was based on the authors experience with this programming language. Although *Mathematica* is a great environment to analyse or calculate numeric, symbolic or geometric data, we found that it was probably not the best choice for this problem.

Whenever possible, we took advantage of the built-in functions and structures, mostly for the `Graph` components of our algorithm. This allowed us to use functions like `FindPath`, which can return all the paths between two nodes of a `Graph`. As it is a built-in function of *Mathematica*, we do not have access to its implementation, but we believe that it is more efficient than any other implementation made by us.

This data structure, and its functions were a great advantage of using *Mathematica*. The other one was the graphical feedback provided, once again, by this component of *Mathematica*.

The disadvantages started when we tried to run the algorithm for large sets. Which was also the main disadvantage of having the procedure, but not its implementation. In fact, we could not run the program for the system proposed in the last example of [2]. Which is, in fact, a simple system, if compared with "real" systems.

The major problem is in the construction of the tableau, which takes too much RAM. We have tried to compute it in systems with 18Gb of RAM and in the Técnico's cluster system, and it crashed both for lack of memory.

We have tried to overcome this problem with several tricks: freeing *Mathematica*'s memory; force the parallel computation; writing the obtained tableau in the disc; restarting the program when high picks of memory were reached.

In the end, we conclude that the procedure works (as it was already proved). To overcame the problems encountered, we suggest:

- the implementation of the algorithm in a more efficient programming language - having in mind that we would probably lose the graphical visualization;

- a mix approach - were one would implement parts of the problem in another programming language, improving the efficiency of the algorithm and keeping the graph structure and output;

- a topological analysis of the graph - in order to close the nodes without needing to explore all the paths between two given points.

These suggestions were not carried out because the deadlines for the conclusion of this work were being met.

References

[1] D. A. Basin, C. Caleiro, J. Ramos, and L. Viganò. Labelled tableaux for distributed temporal logic. *J. Log. Comput.*, 19(6):1245–1279, 2009.

[2] C. Caleiro, P. Gouveia, and J. Ramos. Deciding distributed temporal logic. Technical report, Instituto Superior Técnico, Universidade de Lisboa, Portugal, 2014. Also available as `https://fenix.tecnico.ulisboa.pt/downloadFile/563568428719779/dtl_decidability.pdf`.

[3] H. Ehrich and C. Caleiro. Specifying communication in distributed information systems. *Acta Inf.*, 36(8):591–616, 2000.

[4] D. Lichtblau. Enhance your code performance: Effective data structures techniques in mathematica. `http://library.wolfram.com/infocenter/Conferences/388/`, 1999.

[5] C. Neves. Deciding distributed temporal logic. Master's thesis, Instituto Superior Técnico, Lisboa, December 2015.

[6] C. Neves. Implementation. `http://web.ist.utl.pt/ist167150/`, 2015.