DTL: Translation, SMT Verification, Separation and Interpolation

Miguel de Lacerda e Costa Serra do Nascimento
Instituto Superior Técnico, Lisboa, Portugal
miguel.s.nascimento@tecnico.pt

January 2021

Abstract

The purpose of this work is to contribute to better understand Distributed Temporal Logic (DTL), namely by investigating whether it enjoys some important logical properties. We start by presenting a translation from DTL formulas into first-order logic (FOL) formulas that preserves entailment, and afterwards we resort to the theorem prover and satisfiability modulo theories (SMT) solver CVC4 in order to check the validity of DTL formulas, capitalizing on the translation from DTL to FOL previously mentioned. Furthermore, we propose an extension of the separation property to DTL, and we present a result stating that our extension of this property holds for the distributed temporal logic whose local languages contain both the Until and Since operators. We also adapt the Craig interpolation property to DTL, and we present a result stating that the property holds for a fragment of this logic.

1 Introduction

Distributed temporal logic (DTL) is a temporal logic introduced with the purpose of reasoning about temporal properties of discrete distributed systems from the local point of view of its agents [6; 5]. Having been first proposed near the end of the twentieth century in [8], DTL is a relatively recent logic, when comparing to, for instance, first-order logic (FOL) or linear temporal logic (LTL). Because of this, there are still some important ideas and logical properties left to be studied for DTL, some of which may improve the applicability of this logic in certain areas. The purpose of this document is to investigate whether DTL enjoys some of these logical properties.

In this work, we will start by presenting the syntax and semantics of FOL, LTL and DTL, and we will then proceed to approach the topic of translation. More specifically, we will present a translation from DTL formulas into FOL formulas that preserves entailment. In fact, some problems can be made easier to solve by considering the DTL formula translated into FOL, rather than the DTL formula itself. We will make use of this translation when studying satisfiability modulo theories (SMT) verification, since the question of whether a DTL formula is valid or not can be tied to the SMT problem.

The SMT problem is a variant of the SAT problem for which the non-logical symbols are interpreted in the context of some background theory. We will study CVC4, a theorem prover for SMT problems. We will also attempt to check, using CVC4, the validity of DTL formulas, by initially translating them into FOL.

The topic of separation will be addressed as well. A logic is said to have the property of separation if every formula is equivalent to a Boolean combination of formulas that each refer only to the present, past or future [15; 17; 20; 14]. Dov Gabbay was the first to show that the temporal logic with the operators Until and Since has the separation property over the integers [9]. It turns out that, for temporal logic, the notion of separation is tied to the expressiveness of the logic, that is, the variety and quantity of ideas that the logic can be used to represent [16; 11]. In this work, a proposal for an extension of the separation property to distributed temporal logic is presented.

We will finish this document with the topic of Craig interpolation. A logic having the Craig interpolation property is such that if a formula ϕ entails a formula ψ, then there exists a formula θ (called the interpolant) such that ϕ entails θ, θ entails ψ, and every propositional symbol in θ occurs both in ϕ and ψ. When it comes to applications in computer science, interpolation is often a desired property to have in a temporal logic. For example, interpolation has played a role in building efficient model checkers. Uniform interpolation, which we will talk about in this work, has been particularly useful in this regard.

The Craig interpolation property is proven to hold for both FOL and a fragment of LTL [10; 12]. We will study the Craig interpolation property in the context of distributed temporal logic, and reach the conclusion that this property holds for the fragment of DTL whose local languages contain X as the only temporal operator.

2 A Translation from DTL into FOL

In this section, we present the syntax and semantics of first-order logic (FOL), linear temporal logic (LTL) and distributed temporal logic (DTL). Furthermore, we aim to show how to translate from DTL into FOL. With this intention, we will define a translation function that translates DTL formulas into

FOL formulas, and reach the conclusion that our translation function preserves entailment in DTL. This translation function will be necessary for section 3, where we attempt to check the validity of DTL formulas by resorting to a Mathematica function that, based on the translation function we will define, translates DTL formulas into FOL formulas written in CVC4's native language. The code of this Mathematica function, along with some guidance on how to use this function, can be found in [13].

2.1 First-order Logic

We will introduce formulas in the context of FOL. First, we start by explaining what a first-order signature is.

Definition 1. *A first-order signature is a tuple $\Sigma = \langle \mathcal{F}, \mathcal{P}, \tau \rangle$ such that*

- *$\mathcal{F}$ and $\mathcal{P}$ are disjoint sets, with $\mathcal{P} \neq \emptyset$;*
- *$\tau : \mathcal{F} \cup \mathcal{P} \to \mathbb{N}$ is a map.*

The elements of $\mathcal{F}$ are said to be the *function symbols*, while the elements of $\mathcal{P}$ are said to be the *predicate symbols* (or predicate letters). The map τ returns the arity of its argument. Additionally, let

- $\mathcal{F}_n$ denote the subset of function symbols with arity n;
- $\mathcal{P}_n$ denote the subset of predicate symbols with arity n.

Before introducing formulas in first-order logic, we first have to say what is a term. Let $\mathcal{X} = \{x_0, x_1, \dots\}$ denote the set of variables.

Definition 2. *The set $\mathcal{T}_\Sigma$ of terms over Σ is inductively defined as follows:*

- *$\mathcal{F}_0 \cup \mathcal{X} \subseteq \mathcal{T}_\Sigma$;*
- *$f(t_1, \dots, t_n) \in \mathcal{T}_\Sigma$ provided that $f \in \mathcal{F}_n$ and $t_1, \dots, t_n \in \mathcal{T}_\Sigma$.*

Definition 3. *The set $\mathcal{L}_\Sigma$ of formulas over Σ is inductively defined as follows:*

- *$\bot \in \mathcal{L}_\Sigma$ - "bottom", denotes a proposition that is always false;*
- *$p(t_1, \dots, t_n) \in \mathcal{L}_\Sigma$ provided that $p \in \mathcal{P}_n$ and $t_1, \dots t_n \in \mathcal{T}_\Sigma$;*
- *$\varphi_1 \Rightarrow \varphi_2 \in \mathcal{L}_\Sigma$ provided that $\varphi_1, \varphi_2 \in \mathcal{L}_\Sigma$ - implication;*
- *$\forall x\, \varphi \in \mathcal{L}_\Sigma$ provided that $x \in \mathcal{X}$ and $\varphi \in \mathcal{L}_\Sigma$ - universal quantification.*

For the sake of simplicity, it may be useful to consider certain formula abbreviations. As such, we define the following abbreviations:

- $\neg\, \varphi \equiv (\varphi \Rightarrow \bot)$ - negation;
- $\top \equiv (\bot \Rightarrow \varphi)$ - denotes a proposition which is unconditionally true;
- $\varphi \vee \psi \equiv \neg\, \varphi \Rightarrow \psi$ - disjunction;
- $\varphi \wedge \psi \equiv \neg\, (\varphi \Rightarrow \neg \psi)$ - conjunction;
- $\exists x\, \varphi \equiv \neg\, \forall x\, \neg\, \varphi$ - existential quantification.

In a formula, it may also be important to distinguish between variables that occur free and variables that are bounded to a quantifier.

We inductively define the map var_Σ that assigns to each term the set of variables occurring in it the following way:

- $var_\Sigma(x) = \{x\}$;
- $var_\Sigma(c) = \emptyset$;
- $var_\Sigma(f(t_1, \dots, t_n)) = var_\Sigma(t_1) \cup \dots \cup var_\Sigma(t_n)$.

Also, we inductively define the map fv_Σ that assigns to each formula the set of *variables occurring free* in it in the following way:

- $fv_\Sigma(p(t_1, \dots, t_n)) = var_\Sigma(t_1) \cup \dots \cup var_\Sigma(t_n)$.
- $fv_\Sigma(\bot) = \emptyset$;
- $fv_\Sigma(\varphi \Rightarrow \psi) = fv_\Sigma(\varphi) \cup fv_\Sigma(\psi)$;
- $fv_\Sigma(\forall x\, \varphi) = fv_\Sigma(\varphi) \backslash \{x\}$.

Now, we study the semantics of FOL, that is, the study of the logical system in the point of view of their interpretation. With this in mind, we introduce the concepts of interpretation structure, assignment and satisfaction.

Definition 4. *An interpretation structure over Σ is a tuple $\mathcal{I} = \langle \mathcal{D}, \{f^\mathcal{I}\}_{f \in \mathcal{F}}, \{p^\mathcal{I}\}_{p \in \mathcal{P}} \rangle$ such that*

- *$\mathcal{D}$ is a non-empty set which we call the domain;*
- *$f^\mathcal{I} : \mathcal{D}^n \to \mathcal{D}$ is a map providing that $f \in \mathcal{F}_n$;*
- *$p^\mathcal{I} : \mathcal{D}^n \to \{0, 1\}$ is a map providing that $p \in \mathcal{P}_n$;*

Essentially, $f^\mathcal{I}$ and $p^\mathcal{I}$ represent the interpretation or denotation of, respectively, f and p in $\mathcal{I}$.

An assignment is a map $\rho : \mathcal{X} \to \mathcal{D}$. We say that an assignment σ is x-equivalent to another assignnment ρ, which we write $\sigma \equiv_x \rho$, if $\sigma(y) = \rho(y)$ for every $y \in \mathcal{X} \backslash \{x\}$.

Definition 5. *We inductively define contextual satisfaction as follows:*

- *$\mathcal{I}\rho \nVdash_\Sigma \bot$;*
- *$\mathcal{I}\rho \Vdash_\Sigma p(t_1, \dots, t_n)$ whenever $p^\mathcal{I}([\![t_1^{\mathcal{I}\rho}]\!], \dots, [\![t_n^{\mathcal{I}\rho}]\!]) = 1$;*
- *$\mathcal{I}\rho \Vdash_\Sigma \varphi_1 \Rightarrow \varphi_2$ provided that either $\mathcal{I}\rho \nVdash_\Sigma \varphi_1$ or $\mathcal{I}\rho \Vdash_\Sigma \varphi_2$;*
- *$\mathcal{I}\rho \Vdash_\Sigma \forall x\, \varphi$ providing that $\mathcal{I}\sigma \Vdash_\Sigma \varphi$ for every σ that is x-equivalent to ρ.*

Given an interpretation structure $\mathcal{I}$ and an assignment ρ, if $\mathcal{I}\rho \Vdash \varphi$, we say that $\mathcal{I}$ and ρ contextually satisfy $\varphi \in \mathcal{L}_\Sigma$.

Definition 6. *We say that $\mathcal{I}$ satisfies φ (we can also say that φ is true in $\mathcal{I}$, or that $\mathcal{I}$ is a model of φ), which we write $\mathcal{I} \Vdash_\Sigma \varphi$, whenever $\mathcal{I}\rho \Vdash_\Sigma \varphi$ for every ρ.*

Definition 7. *A formula is valid, written $\vDash_\Sigma \varphi$, if $\mathcal{I} \Vdash \varphi$ for every interpretation structure $\mathcal{I}$ over Σ.*

We will also need the definition of entailment and the definition of theory for some of the following sections. We introduce these concepts in the next definitions.

Definition 8. *A formula φ over Σ is entailed by a set Γ of formulas over the same signature, which we write $\Gamma \models_\Sigma \varphi$, if, for every interpretation structure $\mathcal{I}$ over Σ, $\mathcal{I} \Vdash_\Sigma \varphi$ whenever $\mathcal{I} \Vdash_\Sigma \gamma$ for each $\gamma \in \Gamma$.*

Definition 9. *The semantic closure of a set $\Gamma \subseteq \mathcal{L}_\Sigma$ is the set*

$$\Gamma^{\models_\Sigma} = \{\varphi \in \mathcal{L}_\Sigma : \Gamma \models_\Sigma \varphi\}$$

of its entailed formulas.

Definition 10. *A set of formulas $\Theta \subseteq L_\Sigma$ is said to be a theory if $\Theta^{\models_\Sigma} = \Theta$.*

2.2 Linear Temporal Logic

While first-order logic describes a static situation, temporal logic is able to depict that situation as time progresses. In particular, linear temporal logic (LTL) is a propositional temporal logic with modalities that describe events along a single time path.

We introduce LTL formulas in the following definition.

Definition 11. *Given a set of propositional symbols $Prop$, the language of linear temporal logic $\mathcal{L}_{LTL}$ is defined as follows:*

$\mathcal{L}_{LTL} ::= Prop \mid \bot \mid \mathcal{L}_{LTL} \Rightarrow \mathcal{L}_{LTL} \mid \mathsf{X}[\mathcal{L}_{LTL}] \mid \mathcal{L}_{LTL} \,\mathsf{U}\, \mathcal{L}_{LTL} \mid \mathcal{L}_{LTL} \,\mathsf{S}\, \mathcal{L}_{LTL}$.

The $\bot$ and $\Rightarrow$ have the usual meanings. Intuitively, the formula $\mathsf{X}\varphi$, which we call a next formula, stands for "φ will hold in the next instant". The formula $\varphi_1 \,\mathsf{U}\, \varphi_2$, which we call a until formula, stands for "there is an instant in the future where φ_2 will hold and until then formula φ_1 must hold". Furthermore, the formula $\varphi_1 \,\mathsf{S}\, \varphi_2$, which we call a since formula, is similar to an until formula but in the past direction. Intuitively, it stands for "since the last instant in the past for which φ_2 was true, φ_1 has always been true".

We introduce the following abbreviations:

- $\mathsf{F}\,\varphi \equiv \top \,\mathsf{U}\, \varphi$ - which stands for "sometime in the future, φ must hold";
- $\mathsf{G}\,\varphi \equiv \neg\, \mathsf{F}\, \neg\, \varphi$ - which stands for "always in the future, φ must hold".

Note that all the temporal operators we have shown talk about the future, except for the since operator, which talks about the past. Additionally, temporal operators can be combined to express more complex properties.

Regarding the semantics, we introduce the definition of finite and infinite words. We assume a nonempty and finite set Σ, called the alphabet, whose elements are called symbols or letters.

Definition 12. *Let Σ be an alphabet. A finite word w over Σ is a finite, possibly empty, sequence $\nu_1\nu_2 \ldots \nu_n$ where $n \in \mathbb{N}$ and each $\nu_i \in \Sigma$, for $i = 1, \ldots, n$. The set of all finite words over Σ is denoted by Σ^*.*

The finite word corresponding to the empty sequence, which we call the empty word, is denoted by ϵ. The length of a word, which we denote by $|w|$, is the number of symbols that appear in the sequence. For instance, the length of the word $w = \nu_1\nu_2 \ldots \nu_n$ in n. The length of the empty word is 0. Also, we write $w|_i$ to denote the prefix of w of length i, that is, $w|_i = \nu_1 \ldots \nu_i$, provided that $0 \leq i \leq |w|$.

Definition 13. *An infinite word σ over Σ is an infinite sequence $\sigma = \nu_1\nu_2 \ldots$ where each $\nu_i \in \Sigma$, for $i \in \mathbb{N}$. The set of all infinite words over Σ is denoted by Σ^ω.*

The length of an infinite word is always ω.

Definition 14. *An interpretation for LTL is an infinite word over 2^{Prop}.*

Intuitively, an interpretation structure for LTL is an infinite sequence of valuations, such that each element of the sequence in an LTL interpretation structure will determine the boolean values of the propositional symbols at a given moment in time. Given an interpretation $\sigma = \nu_0\nu_1\nu_2 \ldots$, we can see ν_i as the set of propositional symbols that hold at an instant i. We can now define satisfaction in the context of LTL.

Definition 15. *Let σ be an interpretation and $i \in \mathbb{N}$. The local satisfaction relation for LTL is inductively defined as follows:*

- $\sigma, i \nVdash \bot$;
- $\sigma, i \Vdash p$ *if* $p \in \sigma[i]$ *if* $\sigma_i(p) = 1$;
- $\sigma, i \Vdash \varphi_1 \Rightarrow \varphi_2$ *if* $\sigma, i \nVdash \varphi_1$ *or* $\sigma, i \Vdash \varphi_2$;
- $\sigma, i \Vdash \mathsf{X}\,\varphi$ *if* $\sigma, i+1 \Vdash \varphi$;
- $\sigma, i \Vdash \varphi_1 \,\mathsf{U}\, \varphi_2$ *if there is* $j > i$ *such that* $\sigma, j \Vdash \varphi_2$ *and* $\sigma, k \Vdash \varphi_1$, *for every* $i < k < j$.

The interpretation σ satisfies the formula φ, which we write $\sigma \Vdash \varphi$, if $\sigma, i \Vdash \varphi$ for every i. Similarly to FOL, we say that a formula is valid if it is satisfied by all interpretations.

We also define entailment in the context of LTL.

Definition 16. *Let $\Gamma \cup \{\varphi\} \subseteq \mathcal{L}$. We say that Γ entails φ, written $\Gamma \models \varphi$, when $\sigma \Vdash \varphi$ for every interpretation σ such that $\sigma \Vdash \Gamma$.*

2.3 Distributed Temporal Logic

The syntax of distributed temporal logic (DTL) is defined over a distributed signature.

Definition 17. *A distributed signature is defined as a tuple $\Sigma = \langle Id, \{Prop_i\}_{i \in Id} \rangle$, where Id is a non-empty finite set of agents and, for each $i \in Id$, $Prop_i$ is a set of local state propositions.*

The global language $\mathcal{L}_{DTL}$ is defined by

$$\mathcal{L}_{DTL} ::= @_{i_1}[\mathcal{L}_{i_1}] \mid \cdots \mid @_{i_n}[\mathcal{L}_{i_n}] \mid \bot \mid \mathcal{L}_{DTL} \Rightarrow \mathcal{L}_{DTL},$$

for $Id = \{i_1, \ldots, i_n\}$, where the local languages $\mathcal{L}_i$ for each $i \in Id$ are defined by

$$\mathcal{L}_i ::= Prop_i \mid \bot \mid \mathcal{L}_i \Rightarrow \mathcal{L}_i \mid \mathcal{L}_i \,\mathsf{U}\, \mathcal{L}_i \mid \mathcal{L}_i \,\mathsf{S}\, \mathcal{L}_i \mid \copyright_j[\mathcal{L}_j],$$

with $j \in Id$.

The $\bot$ and $\Rightarrow$ have the usual meaning, while U and S were introduced in the previous section. In the global language, we introduce a new kind of formula. We say that the formula $@_i[\varphi]$, called a global formula, means that φ holds for agent i. On the other hand, local formulas hold locally for each agent.

Note that temporal operators only occur in local formulas, meaning that we talk about these formulas in the context of a given agent. Also locally for an agent i, the formula $\copyright_j[\psi]$,

called a communication formula, means that agent i has just communicated (or synchronized) with agent j, for whom ψ holds.

Before presenting the definition of a DTL interpretation structure, we will introduce the concepts of local and distributed life-cycles, and of local and global states.

Definition 18. *A local life-cycle of an agent $i \in Id$ is a countable infinite, discrete and well-founded total order $\lambda_i = \langle Ev_i, \leq_i \rangle$, where Ev_i is the set of local events and $\leq_i$ the local order of causality.*

Definition 19. *The relation $\rightarrow_i \subseteq Ev_i \times Ev_i$, called the local successor relation, is the relation such that $e \rightarrow_i e'$ if $e <_i e'$ and there is no e'' such that $e <_i e'' <_i e'$.*

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

Definition 20. *A distributed life-cycle is a family $\lambda = \{\lambda_i\}_{i \in Id}$ of local life-cycles. This family is such that $\leq = (\bigcup_{i \in Id} \leq_i)^*$ defines a partial order of global causality on the set of all events $E = \bigcup_{i \in Id} E_i$.*

Note that, due to the fact that communication between agents involves event sharing, we may have, for some event e, $e \in E_i \cap E_j$, for $i \neq j$.

Definition 21. *The local state of agent i is a finite set $\xi_i \subseteq Ev_i$ down-closed for local causality, that is, if $e \leq_i e'$ and $e' \in \xi_i$, then also $e \in \xi_i$.*

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

Due to the total order on local events, the local states of each agent are totally ordered. The 0^{th} state of each agent is $\emptyset$, and the next local state is reached by the occurrence of an event which we call $last(\xi_i)$, since it is the last event in which agent i took part in order to reach the present state ξ_i. We denote by ξ_i^k the k^{th} state of agent i, meaning that $\xi_i^0 = \emptyset$ is the initial state and ξ_i^k is the state reached after the occurrence of the first k events. Note that ξ_i^k is the only state of agent i that contains exactly k elements, that is, where $|\xi_i^k| = k$. Moreover, given $e \in Ev_i$, $(e \downarrow i) = \{e' \in Ev_i | e' \leq_i e\}$ is always a local state. Furthermore, we have that $(last(\xi_i) \downarrow i) = \xi_i$, assuming that ξ_i is non-empty.

Definition 22. *A global state is a finite set $\xi \subseteq Ev$ closed for global causality, that is, if $e \leq e'$ and $e' \in \xi$, then also $e \in \xi$.*

The set of all global states, written Ξ, has $\emptyset$ as the minimal element. Also, we can see that every global state ξ includes the local state of agent i.

After these definitions, we can finally introduce DTL interpretation structures.

Definition 23. *An interpretation structure μ for DTL is a labelled distributed life-cycle of the form $\mu = \langle \lambda, \sigma \rangle$, where λ consists of a distributed life-cycle and $\sigma = \{\sigma_i\}_{i \in Id}$ is a family of local labelling functions.*

For each $i \in Id$, the local labelling functions σ_i associate a set of local state propositions to each local state. Furthermore, we also denote the tuple $\langle \lambda_i, \sigma_i \rangle$ by μ_i.

Now, we can define the global satisfaction relation by

- $\mu \Vdash \gamma$ if $\mu, \xi \Vdash \gamma$ for every $\xi \in \Xi$,

where the global satisfaction relation at a global state is defined by

- $\mu, \xi \nVdash \bot$;
- $\mu, \xi \Vdash \gamma \Rightarrow \delta$ if $\mu, \xi \nVdash \gamma$ or $\mu, \xi \Vdash \delta$;
- $\mu, \xi \Vdash @_i[\varphi]$ if $\mu_i \Vdash_i \varphi$ if $\mu_i, \xi \Vdash_i \varphi$ for every $\xi \in \Xi_i$,

and where the local satisfaction relations at local states are defined by

- $\mu_i, \xi \Vdash_i p$ if $p \in \sigma_i(\xi)$;
- $\mu_i, \xi \Vdash_i \neg \varphi$ if $\mu_i, \xi \nVdash_i \varphi$;
- $\mu_i, \xi \Vdash_i \varphi \Rightarrow \psi$ if $\mu_i, \xi \nVdash_i \varphi$ or $\mu_i, \xi \Vdash_i \psi$;
- $\mu_i, \xi \Vdash_i \varphi \,\mathsf{U}\, \psi$ if $|\xi| = k$ and there exists $\xi_i^n \in \Xi_i$ such that $k < n$ with $\mu_i, \xi_i^n \Vdash_i \psi$, and $\mu_i, \xi_i^m \Vdash_i \varphi$ for every $k < m < n$;
- $\mu_i, \xi \Vdash_i \varphi \,\mathsf{S}\, \psi$ if $|\xi| = k$ and there exists $\xi_i^n \in \Xi_i$ such that $n < k$ with $\mu_i, \xi_i^n \Vdash_i \psi$, and $\mu_i, \xi_i^m \Vdash_i \varphi$ for every $n < m < k$;
- $\mu_i, \xi \Vdash_i \copyright_j[\varphi]$ if $|\xi| > 0, last_i(\xi) \in E_j$, and $\mu_j, (last_i(\xi) \downarrow j) \Vdash_j \varphi$.

A DTL formula γ is said to be valid, written $\vDash \gamma$, if $\mu \Vdash \gamma$ for every global interpretation structure μ.

Finally, we define entailment in the context of DTL.

Definition 24. *Let $\Gamma \cup \{\varphi\} \subseteq \mathcal{L}_{DTL}$. We say that Γ entails φ, written $\Gamma \vDash \varphi$, when $\sigma \Vdash \varphi$ for every interpretation σ such that $\sigma \Vdash \Gamma$.*

2.4 The Translation Function

In this subsection, we present a translation function from DTL formulas into FOL formulas that preserves entailment in DTL.

First, we need to introduce some definitions. To start, note that, excluding communication formulas, local DTL formulas coincide with LTL formulas. This fact is used in [5] to prove that DTL is decidable by a translation into LTL. We make use of this idea, along with the fact that there is a known translation from LTL into FOL [7], in our own translation.

Given a DTL signature $\Sigma = \langle Id, Prop \rangle$, we define the corresponding FOL signature as having $\mathcal{P} = \{@(i,n) \mid i \in Id,\ n \in \mathbb{N}_0\} \cup \biguplus_{i \in Id} Prop_i \cup \{<\}$, where $<$ has the usual meaning. We assume that the symbol $p \in Prop_i$ is represented in $\mathcal{P}$ by the unary predicate p_i. The additional predicate $@(i,n)$, with $i \in Id$ and $n \in \mathbb{N}_0$, is meant to express whether the n-th event in the global order of events of the DTL signature belongs to agent i or not.

Thus, the translation of global formulas is given by the function $f : \mathcal{L}_{DTL} \rightarrow \mathcal{L}_{FOL}$ such that

- $f(@_i[\varphi]) = \forall x\ (@(i,x) \Rightarrow f_i(\varphi, x))$,

and for each $i \in Id$, the function $f_i : \mathcal{L}_i \rightarrow \mathcal{L}_{FOL}$ translates local formulas to FOL formulas the following way:

- $f_i(p, x) = p_i(x)$;
- $f_i(\neg\varphi, x) = \neg f_i(\varphi, x)$;
- $f_i(\varphi \Rightarrow \psi, x) = f_i(\varphi, x) \Rightarrow f_i(\psi, x)$;

- $f_i(\varphi \,\mathsf{U}\, \psi, x) = \exists y\ x < y \wedge @(i,y) \wedge f_i(\psi, y) \wedge \forall z\ (x < z < y \Rightarrow (@(i,z) \Rightarrow f_i(\varphi, z)))$;
- $f_i(\varphi \,\mathsf{S}\, \psi, x) = \exists y\ y < x \wedge @(i,y) \wedge f_i(\psi, y) \wedge \forall z\ (y < z < x \Rightarrow (@(i,z) \Rightarrow f_i(\varphi, z)))$;
- $f_i(\copyright_j[\varphi], x) = @(j,x) \wedge f_j(\varphi, x)$.

Let us also consider the map β from FOL interpretation structures to DTL interpretation structures such that $\beta(\mathcal{I}) = \langle \lambda, \sigma \rangle$, with $\lambda_i = \langle E_i, \leq_i \rangle$, where:

- $E_i = \{n \in \mathbb{N} \mid @(i,n) \in \mathcal{P}^{\mathcal{I}}\}$;
- $\leq_i$ is the restriction of the usual order on $\mathbb{N}$, with $n \rightarrow_i m$ if $n, m \in E_i$ and there is no $k \in E_i$, such that $n < k < m$;
- $\sigma_i(\emptyset) = \{p \in Prop_i \mid p_i(0) = 1\}$ and $\sigma_i(\{m \in E_i \mid m \leq n\}) = \{p \in Prop_i \mid p_i(n) = 1\}$, for each $n \in E_i$.

Now, we introduce Proposition 1, which is necessary for reaching the conclusion that our functions f_i, for $i \in Id$, are well-defined. Note that we must only consider the FOL interpretation structures that satisfy $\{\bigwedge_{i \in Id} @(i,0)\}$. We need to add this restriction due to the fact that, at the initial DTL state $\emptyset$, no events have yet occurred.

Proposition 1. *[13] Given a FOL interpretation structure $\mathcal{I}$ that satisfies $\{\bigwedge_{i \in Id} @(i,0)\}$, we have that, for every $\varphi \in \mathcal{L}_{DTL}$, $\beta(\mathcal{I})_i, \xi_i^k \Vdash_i \varphi$ if and only if $\mathcal{I}, [x/last_i(\xi_i^k)] \Vdash_{FOL} f_i(\varphi, x)$, for every $\xi_i^k \in \Xi_i$, where $[x/last_i(\xi_i^k)]$ stands for a variable assignment that assigns the free variable x of $f_i(\varphi, x)$ the value $last_i(\xi_i^k)$.*

We are finally ready to present two propositions that allow us to reach the most important result in this chapter: the fact that our translation function f, that translates DTL formulas into FOL formulas, preserves entailment. Again, we only regard in our translation FOL interpretation structures that satisfy $\{\bigwedge_{i \in Id} @(i,0)\}$.

Proposition 2. *[13] Let $\Gamma \cup \{\delta\} \subseteq \mathcal{L}_{DTL}$. We have that if $\Gamma \models_{DTL} \delta$ then $f(\Gamma) \cup \{\bigwedge_{i \in Id} @(i,0)\} \models_{FOL} f(\delta)$.*

Proposition 3. *[13] Let $\Gamma \cup \{\delta\} \subseteq \mathcal{L}_{DTL}$. We have that if $f(\Gamma) \cup \{\bigwedge_{i \in Id} @(i,0)\} \models_{FOL} f(\delta)$ then $\Gamma \models_{DTL} \delta$.*

The two previous propositions allow us to conclude the following result.

Corollary 1. *[13] Let $\Gamma \cup \{\delta\} \subseteq \mathcal{L}_{DTL}$. We have that $\Gamma \models_{DTL} \delta$ if and only if $f(\Gamma) \cup \{\bigwedge_{i \in Id} @(i,0)\} \models_{FOL} f(\delta)$.*

3 SMT Verification using CVC4

The SMT problem is a variant of the SAT problem for first-order logic, where the difference is in the fact that, for an SMT instance, the non-logical symbols are interpreted in the context of some background theory. Thus, the SMT problem consists of determining whether the SMT instance is satisfiable with respect to the background theory. Examples of theories typically used in computer science include the theory of integers, the theory of real numbers, the theory of lists or arrays and so on. Additionally, a given SMT instance might combine a set of theories (for instance, combining the theory of real numbers and the theory of integers).

CVC4 is an open-source theorem prover for SMT problems that can be used to prove the validity or the satisfiability of first-order formulas in several logical theories and their combination [1]. It was released in 2012 as the fourth in the CVC family of tools. It supports 4 different input languages, namely the CVC4 Native Input Language, SMT-LIB v2, SyGuS-IF and TPTP.

SMT problems have important applications in some areas of computer science, such as software verification, model checking and automated test generation [2]. Currently, CVC4 is considered to be one of the most efficient and up-to-date tools for solving these problems.

In this section, a summary of CVC4's functionalities is presented, along with its operability. Additionally, we try to prove or refute the validity of LTL and DTL formulas using two different Mathematica functions we implemented, which translate LTL or DTL formulas into FOL formulas written in CVC4's native language. The code for these two Mathematica functions, and some indications on how to use them, can be found on Appendix A. Throughout this section, we use version 1.7 of CVC4.

3.1 The Functionalities and Operability of CVC4

In this subsection, we focus on providing a summary of the CVC4 Native Input Language documentation, highlighting the most important features and how to operate with them. With this in mind, we start by describing the SAT and SMT problems, by first explaining what a literal is.

Definition 25. *A literal is a either a propositional variable or the negation of a propositional variable. We say that x and y are positive literals, while $\neg x$ and $\neg y$ are said to be negative literals.*

Definition 26. *A clause is a disjunction of one or more literals.*

Definition 27. *A clausal formula is a conjunction of one or more clauses.*

For instance, $(\neg x \vee y) \wedge (\neg y \vee w \vee \neg z)$ is a clausal formula.

The SAT problem is, then, the problem of determining whether a given clausal formula is satisfiable, i.e., if there exists an interpretation that satisfied a given clausal formula.

On the other hand, the SMT problem, also known as the satisfiability modulo theory problem, is a variant of the SAT problem for first-order logic such that, for an SMT instance, the variables are interpreted in the context of some background theory. Some theories that can be considered are, for example, the theory of integer numbers, real numbers and also theories of data structures, such as lists and arrays. Being an SMT-solver, CVC4 supports many of these theories.

As mentioned in the beginning of the section, the SMT-solver CVC4 supports 4 different input languages, which differ in their syntax and in the names of certain commands. The language we used in this document is the CVC4 Native Input Language, whose documentation can be seen in [3]. It should also be mentioned that, even though the examples provided in this section were run in the (Windows) command line, CVC4 can also be run online [4].

```
CVC4> x,y:REAL;
CVC4> ASSERT FORALL (a,b,i,j,k:REAL): i=j AND i/=k => EXISTS (z:REAL): x/=z OR z/=y;
```

Figure 1: Example of the declaration of variables and the use of some symbols and connectives for the ASSERT command in CVC4 (run on the terminal).

CVC4 supports variables of different types, such as REAL, INT, BOOLEAN, STRING, array and tuple, to name a few. Variables can be declared in a similar way to some programming languages. Moreover, resorting to user-created variables or quantified variables, the ASSERT command can be used to add formulas to our current logical context. The current logical context Γ is a collection of the assertions the user has made so far, although CVC4 may also add formulas to the current context (this will be explained further ahead). CVC4 will always take into account the current logical context when executing queries.

As for the real and integer arithmetic theories, CVC4 currently supports numerals, along with the symbols $-$ (both unary and binary), $+$, $*$, $/$, $<$, $>$, $<=$ and $>=$. This excludes certain mathematical operations such as roots, powers, logarithms and factorials, for instance. Specific operations for other data types, like strings, arrays, sets or bit vectors, are also included in the tool.

In addition to these symbols, CVC4 allows the use of connectives such as $\forall$, $\Rightarrow$ and $\vee$, all of which have their own syntax. Figure 1 shows an example, run on the terminal, of the syntax in the declaration of variables, along with the ASSERT command, where some of the previously mentioned symbols and connectives are used.

CVC4's main functionalities come from the QUERY and CHECKSAT commands. The QUERY command gets a formula as input and checks if the formula is valid in the current logical context ($\Gamma \models F$, where Γ is our current logical context and F is the formula). This means that the QUERY command can be used to verify whether a given formula is a theorem or not. The QUERY command can produce 3 different answers:

- If the query returns "valid", it means that $\Gamma \models_T F$. After this, the logical context stays exactly as it was before the query.
- If the query returns "invalid", it means that $\Gamma \not\models_T F$. This implies that there is a model of the theory T that satisfies $\Gamma \cup \{\neg F\}$. After an invalid answer, the current logical context is augmented with a set Δ of variable-free literals such that $\Gamma \cup \Delta$ is satisfiable in T, but $\Gamma \cup \Delta \models_T \neg F$ (which in fact means that Δ entails $\neg F$). We call the new context $\Gamma \cup \Delta$ a counterexample for the formula F.
- If the query returns "unknown", a set Δ of literals which entail $\neg F$ is added to the logical context, similarly to what occurs if the query returns "invalid". However, the tool is not able to guarantee that $\Gamma \cup \Delta$ is satisfiable in T.

On the other hand, the CHECKSAT command also takes a formula as input and checks if the formula is satisfiable in the current logical context ($\Gamma \not\models \neg F$). Thus, this command behaves in the same way as making a query to $\neg F$, returning "sat" if $\neg F$ is invalid, "unsat" if $\neg F$ is valid, and "unknown" in the remaining cases. This command can be used to solve SAT and SMT instances.

```
CVC4> QUERY FORALL(x:REAL): EXISTS(y:REAL): y>x;
valid
CVC4> a,b:INT;
CVC4> ASSERT a+b=2;
CVC4> CHECKSAT a<2 AND 1>b;
unsat
```

Figure 2: Example of the QUERY command being used in CVC4 to prove a theorem and the CHECKSAT command being used to solve a SMT instance (run on the terminal).

Furthermore, if CVC4's produce-models option is turned on (this can only be done on start-up, using the OPTION command), the user gains access to the COUNTERMODEL command. This command can be used to, after an invalid QUERY or a satisfiable CHECKSAT, print a model that makes the input formula invalid or satisfiable, respectively. An example of the application of these commands can be seen in Figure 3.

```
CVC4> OPTION "produce-models";
CVC4> x,y:REAL;
CVC4> QUERY x+y=2;
invalid
CVC4> COUNTERMODEL;
x : REAL = 0;
y : REAL = 0;
CVC4> CHECKSAT x+y=2;
sat
CVC4> COUNTERMODEL;
x : REAL = 2;
y : REAL = 0;
```

Figure 3: Example of the COUNTERMODEL command being applied in CVC4 to get a model that makes the formula $x + y = 2$ invalid after an invalid QUERY, and a model that makes the same formula satisfiable after a satisfiable CHECKSAT (run on the terminal).

CVC4 also contains other commands that can be advantageous in certain situations:

- The PUSH command saves the current state of the system, while POP restores the system to the state it was in right before the last PUSH. It can be useful if we want to return to the logical context we had before a possibly invalid QUERY or satisfiable CHECKSAT, since these results lead to a change in the context.
- The WHERE command prints all the formulas belonging to the current logical context.
- After an invalid QUERY or a satisfiable CHECKSAT, RESTART may be used to repeat the QUERY or CHECKSAT with an additional formula in the logical context. The formula needs to be introduced as input when calling the RESTART command.
- The TRANSFORM command takes a term as input, simplifies it using the current logical context and prints the result.

- PRINT and ECHO, which are common commands in programming languages, are also available.

Many modern SMT solvers, including CVC4, use a specific algorithm, called DPLL(T), to solve the SMT problem for quantifier-free SMT instances in an arbitrary theory T.

The DPLL(T) algorithm (also referred to as the lazy approach) transforms an SMT formula into an SAT one, by replacing every atom in the formula with Boolean variables [18]. Using the regular SAT-solving DPLL algorithm, a satisfying valuation for the new formula is found, if it exists (if not, the algorithm returns unsat). Afterwards, a theory solver is used to check if the assignments found are satisfiable in the theory T. We say that the theory solver checks if the assignments are T-satisfiable. If a contradiction is found by the theory solver, which means that the assignments are not T-satisfiable, then the algorithm refines the SAT formula with this information, and the regular DPLL algorithm is used once more on the SAT formula.

3.2 Validity Checking of LTL and DTL Formulas using CVC4

CVC4 does not directly support LTL, DTL, nor its operators. However, we have seen in section 2 that there is a translation from LTL into FOL, and we proved that a translation from DTL into FOL also exists. That is, any LTL or DTL formula can be transformed into a FOL formula through translation functions that preserve entailment.

Note that a FOL formula obtained using these translation functions will be such that every variable in it is of integer sort, and it can be a quantified formula (in the case of a translation from DTL into FOL, the translated formula will certainly be quantified). This means that, in truth, a translated LTL or DTL formula belongs to the LIA theory. Thus, we should be able to use CVC4 to check the validity of LTL and DTL formulas, if we first translate them into FOL formulas written in an input language that CVC4 supports. Our aim in this section is to understand if the theorem prover is able to correctly check the validity of these formulas.

With this in mind, we implemented a Mathematica function that takes a LTL formula and the current moment in time as input, and returns as output the corresponding FOL formula, written in CVC4's native language. For this, we resorted to the translation function g, which is known to translate LTL formulas into FOL formulas, preserving entailment. We inductively define g the following way:

- $g(p, x) = p(x)$;
- $g(\neg\varphi, x) = \neg g(\varphi, x)$;
- $g(\varphi \Rightarrow \psi, x) = g(\varphi, x) \Rightarrow g(\psi, x)$;
- $g(\mathsf{X}\,\varphi, x) = g(\varphi, x + 1)$;
- $g(\mathsf{F}\,\varphi, x) = \exists y\; x < y \wedge g(\varphi, y)$;
- $g(\mathsf{G}\,\varphi, x) = \forall y\; x < y \Rightarrow g(\varphi, y)$;
- $g(\varphi\,\mathsf{U}\,\psi, x) = \exists y\; x < y \wedge g(\psi, y) \wedge \forall z\; (x < z < y \Rightarrow g(\varphi, z))$.

The code for this Mathematica function can be seen in [13].

Formula	Expected output	CVC4's output	Counterexample
$\mathsf{X}\,(p \wedge q) \rightarrow \mathsf{X}\,p \wedge \mathsf{X}\,q$	valid	valid	
$\mathsf{X}\,p \wedge \mathsf{X}\,q \rightarrow \mathsf{X}\,(p \wedge q)$	valid	valid	
$\mathsf{X}\,(p \vee q) \rightarrow \mathsf{X}\,p \vee \mathsf{X}\,q$	valid	valid	
$\mathsf{X}\,p \vee \mathsf{X}\,q \rightarrow \mathsf{X}\,(p \vee q)$	valid	valid	
$\mathsf{F}\,(p \wedge q) \rightarrow \mathsf{F}\,p \wedge \mathsf{F}\,q$	valid	valid	
$\mathsf{F}\,p \wedge \mathsf{F}\,q \rightarrow \mathsf{F}\,(p \wedge q)$	invalid	unknown	
$\mathsf{G}\,(p \wedge q) \rightarrow \mathsf{G}\,p \wedge \mathsf{G}\,q$	valid	valid	
$\mathsf{G}\,p \wedge \mathsf{G}\,q \rightarrow \mathsf{G}\,(p \wedge q)$	valid	valid	
$\mathsf{F}\,(p \vee q) \rightarrow \mathsf{F}\,p \vee \mathsf{F}\,q$	valid	valid	
$\mathsf{F}\,p \vee \mathsf{F}\,q \rightarrow \mathsf{F}\,(p \vee q)$	valid	valid	
$\mathsf{G}\,(p \vee q) \rightarrow \mathsf{G}\,p \vee \mathsf{G}\,q$	invalid	unknown	
$\mathsf{G}\,p \vee \mathsf{G}\,q \rightarrow \mathsf{G}\,(p \vee q)$	valid	valid	
$\mathsf{F}\,p \rightarrow p$	invalid	invalid	$p = \lambda x.x == 1$ [1]
$\mathsf{F}\,\mathsf{F}\,p \rightarrow \mathsf{F}\,p$	valid	valid	
$\mathsf{G}\,\mathsf{G}\,p \rightarrow \mathsf{G}\,p$	valid	unknown	
$\mathsf{X}\,\mathsf{X}\,p \rightarrow \mathsf{X}\,p$	invalid	invalid	$p = \lambda x.x == 2$ [1]
$\mathsf{G}\,(p \rightarrow \mathsf{X}\,(p)) \rightarrow (p \rightarrow \mathsf{G}\,p)$	valid	–	
$\mathsf{G}\,(p \rightarrow \neg q \wedge \mathsf{X}\,p) \rightarrow (p \rightarrow \neg(r\,\mathsf{U}\,q)$	valid	–	

Table 1: Output of the QUERY command for LTL formulas.

After its implementation, we used the Mathematica function to convert various LTL formulas into FOL formulas written in CVC4's native language, and used the QUERY command to check the validity of each formula.

Table 1 shows the output of the QUERY command for these formulas. For the last two formulas in the table, the queries for both formulas did not halt, and therefore no output was given by CVC4.

It is possible to see that CVC4 was able to verify and refute the validity of many of the formulas. Overall, the results were positive, since although it has its flaws, CVC4 can be used to verify or refute the validity of LTL formulas. However, for some of the formulas, it was not able to reach a conclusion when it comes to their validity, therefore returning "unknown". Furthermore, for the more complex formulas (the last two formulas), the queries did not halt, potentially because CVC4 entered some sort of loop during both queries.

Similarly, a Mathematica function that takes a global DTL formula as input, and returns as output the corresponding FOL formula written in CVC4's native language, was implemented. For this, we resorted to the translation function defined in section 2. Again, the code for this function can be found in [13]. The Mathematica function was also applied on certain formulas, and then the QUERY command was used to check their validity.

First, the formula $@_i[\mathsf{G}\,(p \wedge q) \Rightarrow \mathsf{G}\,p \wedge \mathsf{G}\,q]$ was analyzed, among other DTL formulas consisting of a single global formula containing a local formula equal to a valid LTL formula shown on the table we have previously shown in this section. The results were similar to the ones we had obtained for the LTL formulas, meaning that most of the DTL formulas were correctly determined to be valid.

Considering a different formula, CVC4 correctly established $@_i[\mathsf{F}\,p \Rightarrow p]$ as invalid, for which the command COUNTERMODEL provided the following counterexample:

$$@(id, k) = True \text{ and } p(id, k) = \begin{cases} k == 1 & \text{if } id = i \\ False & \text{otherwise.} \end{cases}$$

As for the formula $(@_i[\mathsf{G}\,(p \Rightarrow \copyright_j[q])] \wedge @_i[\mathsf{F}\,p]) \Rightarrow @_j[\mathsf{F}\,q]$, which should be valid, CVC4 returned "unknown",

[1] For this formula, the current moment in time was considered to be the instant 0.

meaning that it was not able to reach a conclusion regarding the validity of the formula. We had hoped that CVC4 would be able to reach a successful output in this case. However, based on experiments we made with other formulas, CVC4 seems to have difficulty proving that DTL formulas containing communication formulas are valid, most likely because these formulas involve multiple agents and, therefore, more quantifiers, making them more complex than most of the other formulas we have shown.

Due to this, the results obtained for DTL formulas were not as good as we had hoped. Communication formulas are an important part of DTL, since they allow communication between different agents. Unfortunately, the fact that CVC4 has difficulty proving the validity of DTL formulas with communication makes it quite limited when it comes to distributed temporal logic.

4 Separation Properties

In this section we introduce the separation property [15; 17; 20; 14]. Intuitively, a logic has the separation property if every formula can be written as a Boolean combination of formulas that each only talk about the past, present and future.

A temporal logic is expressively complete if for every first-order formula in this fragment, there is a temporal logic formula that has exactly the same models (and vice-versa). A temporal logic is expressively complete if and only if it has the separation property, provided that the temporal logic can express the F and P operators (although we have not yet defined the P operator, it is similar to F, standing for "sometime in the past"). This makes the separation property very useful when it comes to understanding how expressive a temporal logic is, i.e., the quantity and variety of ideas that the temporal logic is able to represent.

The separation property is proven to hold for the temporal logic with the Until and Since operators over the integers [9]. We propose an extension of this property to DTL, and we reach the conclusion that this extension of the separation property holds for the distributed temporal logic for which the local languages have both the Until and Since operators.

4.1 Separation Property for Temporal Logic

We start by introducing $\mathcal{L}_{TL}$, the language of the temporal logic with both Until and Since. Let us consider a countable set of propositional variables $Prop$.

Definition 28. *The language of temporal logic $\mathcal{L}_{TL}$ is defined as follows:*

$$\mathcal{L}_{TL} ::= Prop \mid \bot \mid \mathcal{L}_{TL} \Rightarrow \mathcal{L}_{TL} \mid \mathcal{L}_{TL} \,\mathsf{U}\, \mathcal{L}_{TL} \mid \mathcal{L}_{TL} \,\mathsf{S}\, \mathcal{L}_{TL}.$$

Note that the set $\mathcal{L}_{LTL}$ of LTL formulas can be defined just like the one of TL, but omitting the S case.

We now introduce certain concepts that are useful to understand the notion of separation.

Definition 29. *We say a formula A is $simple$ if it has no outer Boolean structure, that is, if it is of the form p, $B \,\mathsf{S}\, C$, or $B \,\mathsf{U}\, C$ (for some $p \in Prop$ and $B, C \in \mathcal{L}_{TL}$).*

Definition 30. *A formula is called $non\text{-}future$ if it has no occurrences of U and $non\text{-}past$ if it has no occurrences of S.*

A $pure\ past$ formula is then a Boolean combination of formulas of the form $A \,\mathsf{S}\, B$ where both A and B are non-future and similarly a formula is $pure\ future$ if it is a Boolean combination of formulas of the form $A \,\mathsf{U}\, B$ with A and B non-past.

A formula is $pure\ present$ if it is a Boolean combination of variables, $\bot$ and $\top$.

Definition 31. *A formula is $separated$ if it is a Boolean combination of pure past, pure future and pure present formulas.*

As mentioned before, TL has the separation property over the integers, a result that was proven by Gabbay in [9]. When we say "over the integers", we are referring to the domain over which the formulas in the logic are evaluated.

With these definitions, we can proceed to our adaptation of the separation property to distributed temporal logic.

4.2 Separation Property for DTL

To understand how we reached our adaptation of the separation property to distributed temporal logic, there are certain details we first need to take into account.

Note that our definition of separation property for DTL needs only to focus on the local languages of DTL. This is true seeing that any temporal operators in a DTL formula occur in the local languages. In fact, the global language in DTL only deals with global formulas and the relations of global formulas between different agents. Since the global language of a distributed temporal logic is not defined using temporal operators, we can direct our attention to the local languages.

Regarding the formulas in the local languages, we can see that these coincide with temporal formulas, when excluding communication formulas. In turn, communication formulas contain assertions inside them that belong to other local languages of the distributed temporal logic, meaning that they may contain temporal operators inside them or even other communication formulas.

Due to the fact that communication formulas are what differentiates formulas in the local DTL languages from TL formulas, it would be useful to find a way to look at communication formulas as propositional symbols when it comes to separation, which we might be able to achieve if we can separate the assertions inside communication formulas. By looking at communication formulas in local DTL formulas as propositional symbols, it might be possible to separate these local DTL formulas using the same techniques required to separate TL formulas.

Following this train of thought, we reached definition 32.

Definition 32. *A distributed temporal logic is said to have the separation property if its local formulas, for every agent, can be equivalently rewritten as a boolean combination of formulas, each of which depends only on the past, present or future.*

Now that we have a definition of separation property for DTL, it is of our interest to understand if this property holds for distributed temporal logic.

With this intention, let us introduce the concept of complexity of a local DTL formula, which we define the following way:

- $|\varphi| = 0$ for all $\varphi \in \mathcal{L}_i^{\emptyset}$, with $i \in Id$, where $\mathcal{L}_i^{\emptyset}$ denotes all purely temporal formulas of $\mathcal{L}_i$, that is, excluding communication formulas;
- $|\copyright_j[\varphi]| = 1 + |\varphi|$, with $\varphi \in \mathcal{L}_j$, $j \in Id$;
- $|\varphi| = \max(|\copyright^1|, |\copyright^2|, \ldots, |\copyright^m|)$, where $\copyright^1, \copyright^2, \ldots, \copyright^m$ represent the communication formulas in φ that are not inside other communication formulas, with $\varphi \in \mathcal{L}_i$, $i \in Id$.

Now, we are ready to introduce Proposition 4.

Proposition 4. *[13] Distributed temporal logic, as defined in subsection 2.3, has the separation property.*

5 Craig Interpolation

In this section, we address the Craig interpolation property in the context of distributed temporal logic and we reach the conclusion that a fragment of DTL has this property.

In classical logics, the Craig interpolation property states that if a formula ϕ entails a formula ψ, then there exists an interpolant formula θ such that ϕ entails θ, θ entails ψ, and every propositional symbol in θ occurs both in ϕ and ψ.

The Craig interpolation property is frequently a convenient property to have in a temporal logic. Temporal logics, in general, are widely used in the verification of systems and software, and interpolation has been useful for building efficient model-checkers. A stronger form of interpolation called uniform interpolation, which we will talk about in this section, has been particularly useful in this regard.

It has been proved that, in contrast to first-order logic, LTL does not have the Craig interpolation property. However, this property has been proved to hold for fragments of LTL, namely the fragment of LTL for which we consider X as the only temporal operator and exclude the rest of the temporal operators [10; 12]. Inspired by these proofs, we will present a result stating that the Craig interpolation property also holds for the fragment of DTL whose local languages contain X as the only temporal operator.

In this section, will often talk about fragments of DTL. We define fragments of the DTL by allowing in the syntax of their local languages only a subset of the temporal operators that the local languages of DTL are able to express.

5.1 Definition of Craig Interpolation in the Context of DTL

First, we need to introduce some concepts. Let us consider DTL (or a fragment of DTL) having global language $\mathcal{L}$ and local languages $\mathcal{L}_i$, for $i \in Id$. Additionally, let $\alpha = \{\alpha_i\}_{i \in Id}$ be a family of finite sets of propositional letters. We define $\mathcal{L}[\alpha]$ as the set of formulas in the global language $\mathcal{L}$ for which, for each $i \in Id$, the local formulas in the language $\mathcal{L}_i$ contain only propositional symbols from the set α_i.

Given an interpretation structure $\mu = \langle \lambda, \sigma \rangle = \langle \{\lambda_i\}_{i \in Id}, \{\sigma_i\}_{i \in Id} \rangle$ and a family of finite sets of propositional letters $\alpha = \{\alpha_i\}_{i \in Id}$, we define $\mu \restriction \alpha$ as the interpretation structure with valuations $\{\sigma_i \restriction \alpha_i\}_{i \in Id}$, where $\sigma_i \restriction \alpha_i$ represents the restriction of the valuation σ_i to the propositional letters in α_i, for $i \in Id$. We say that $\mu \restriction \alpha$ is the α-reduct of μ. We also say that μ is an expansion of $\mu \restriction \alpha$. Moreover, if $\mathcal{K}$ is a class of DTL interpretation structures, we write $\mathcal{K} \restriction \alpha$ for $\{\mu \restriction \alpha \mid \mu \in \mathcal{K}\}$.

Definition 33. *Let $\mathcal{K}$ be a class of DTL interpretation structures and $\mathcal{L}$ be the global language of DTL (or of a fragment of DTL). We say that $\mathcal{K}$ is definable by $\mathcal{L}$ if there is a $\phi \in \mathcal{L}$ such that, for every interpretation structure μ, $\mu \Vdash \phi$ iff $\mu \in \mathcal{K}$.*

Definition 34. *Let $\mathcal{K}$ be a class of DTL interpretation structures and $\mathcal{L}$ be the global language of DTL (or a fragment of DTL) with set of agents Id. Additionally, let $\alpha = \{\alpha_i\}_{i \in Id}$ be a family of finite sets of propositional letters. Then $\mathcal{K}$ is a projective class of the global language $\mathcal{L}$ if there is a $\phi \in \mathcal{L}[\beta]$, with $\{\beta_i \supseteq \alpha_i\}_{i \in Id}$, such that $\mathcal{K} = Mod(\phi) \restriction \alpha$.*

Now we define the Craig interpolation property for distributed temporal logic.

Definition 35. *Let $\mathcal{L}$ be the global language of DTL (or a fragment of DTL) with set of agents Id, and $\alpha = \{\alpha_i\}_{i \in Id}$ and $\beta = \{\beta_i\}_{i \in Id}$ two families of finite sets of propositional letters. Then $\mathcal{L}$ has the Craig interpolation property whenever the following holds. Let $\phi \in \mathcal{L}[\alpha], \psi \in \mathcal{L}[\beta]$. Whenever $\phi \vDash \psi$, then there exists $\theta \in \mathcal{L}[\{\alpha_i \cap \beta_i\}_{i \in Id}]$ such that $\phi \vDash \theta$ and $\theta \vDash \psi$.*

We also define uniform interpolation, a stronger form of interpolation than Craig interpolation.

Definition 36. *Let $\mathcal{L}$ be the global language of DTL (or a fragment of DTL) with set of agents Id and let $\alpha = \{\alpha_i\}_{i \in Id}$ be a family of finite sets of propositional letters. Then $\mathcal{L}$ has uniform interpolation if, for all families of sets of propositional letters $\beta = \{\beta_i\}_{i \in Id}$ such that $\{\beta_i \subseteq \alpha_i\}$ for all $i \in Id$, and for each formula $\phi \in \mathcal{L}[\alpha]$, there is a formula $\theta \in \mathcal{L}[\beta]$ such that $\phi \vDash \theta$ and for each formula $\psi \in \mathcal{L}[\alpha']$ with $\{\alpha_i \cap \alpha'_i \subseteq \beta_i\}$ for all $i \in Id$, if $\phi \vDash \psi$ then $\theta \vDash \psi$.*

To give an intuition, a fragment of DTL that has uniform interpolation is such that the interpolant can be constructed so that it depends only on the family of sets of propositional symbols of the antecedent and its intersection, for each agent, with the family of sets of propositional symbols of the consequent.

5.2 Craig Interpolation for a Fragment of DTL

Now, let us consider the fragment of DTL for which the local languages contain X as the only temporal operator (although they still contain communication formulas). We call this fragment DTL(X). The global language $\mathcal{L}$ of DTL(X) is defined by

$$\mathcal{L} ::= @_{i_1}[\mathcal{L}_{i_1}] \mid \cdots \mid @_{i_n}[\mathcal{L}_{i_n}] \mid \bot \mid \mathcal{L} \Rightarrow \mathcal{L},$$

for $Id = \{i_i, \ldots, i_n\}$, where the local languages $\mathcal{L}_i$ for each $i \in Id$ are defined by

$$\mathcal{L}_i ::= Prop_i \mid \bot \mid \mathcal{L}_i \Rightarrow \mathcal{L}_i \mid \mathsf{X}\mathcal{L}_i \mid \copyright_j[\mathcal{L}_j].$$

Theorem 2. *[13] DTL(X) has uniform interpolation.*

Corollary 3. *DTL(X) has the Craig interpolation property.*

Proof. Considering that uniform interpolation is a stronger form of interpolation than Craig interpolation, the result comes directly from Theorem 2. □

6 Conclusion

One of the goals of this work was to present a translation from DTL into FOL. With this in mind, we have presented a translation function that translates DTL formulas into FOL formulas, and we have shown a result stating that it preserves entailment in DTL.

We have also studied the theorem-prover CVC4, and following the work done in [19], we have presented the algorithms used in the theorem-prover for proving (or refuting) the validity of quantified formulas in the LIA theory. Since LTL and DTL formulas translated into FOL belong to the LIA theory, and typically contain quantifiers, we created two Mathematica functions, which can be seen in Appendix A, that translate LTL and DTL formulas into FOL formulas written in CVC4's native language. With this, we wanted to understand if it was possible to use CVC4 to check the validity of LTL and DTL formulas. While this method shows some positive results for LTL formulas, it fares worse when it comes to DTL formulas, specifically the ones that contain communication formulas. This likely occurs because these formulas involve multiple agents, which lead to more quantifiers on the translated formula, often making them more complex. Unfortunately, we are not able to recommend our approach with CVC4 for checking the validity of DTL formulas. However, it would be interesting to study another theorem prover and to understand if it could provide more consistent results in our approach for checking the validity of DTL formulas.

We have proposed an extension of the separation property to DTL, based on the definition of this property for temporal logic [15; 17; 20; 9; 14]. We have presented a result stating that this extension of the separation property holds for the distributed temporal logic for which the local languages have both the Until and Since operators.

Finally, we have addressed the Craig interpolation property in the context of DTL and we have presented a result stating that this property holds for the fragment of DTL for which the local languages contain X as the only temporal operator.

References

[1] "About CVC4," 2020, last accessed 30 January 2020. [Online]. Available: https://cvc4.github.io/index.html

[2] "About SMT-LIB," 2020, last accessed 30 January 2020. [Online]. Available: http://smtlib.cs.uiowa.edu/about.shtml

[3] "CVC4 Native Input Language," 2020, last accessed 04 February 2020. [Online]. Available: https://cvc4.github.io/cvc4-native-input-language.html

[4] "CVC4 Online App," 2020, last accessed 04 February 2020. [Online]. Available: https://cvc4.github.io/app/

[5] D. Basin, C. Caleiro, J. Ramos, and L. Viganò, "Labelled tableaux for distributed temporal logic," *Journal of Logic and Computation*, vol. 19, no. 6, pp. 1245–1279, January 2009.

[6] D. Basin, C. Caleiro, J. Ramos, and L. Vigano, "Distributed temporal logic for the analysis of security protocol models," *Theor. Comput. Sci.*, vol. 412, pp. 4007–4043, July 2011. [Online]. Available: https://doi.org/10.1016/j.tcs.2011.04.006

[7] G. De Giacomo and M. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," *IJCAI International Joint Conference on Artificial Intelligence*, pp. 854–860, January 2013.

[8] H.-D. Ehrich and C. Caleiro, "Specifying communication in distributed information systems," *Acta Informatica*, vol. 36, pp. 591–616, 03 2000.

[9] D. Gabbay, "The declarative past and imperative future," in *Temporal Logic in Specification*, B. Banieqbal, H. Barringer, and A. Pnueli, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 409–448.

[10] A. Gheerbrant and B. ten Cate, "Craig interpolation for linear temporal languages," in *Computer Science Logic*, E. Grädel and R. Kahle, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 287–301.

[11] I. Hodkinson and M. Reynolds, "Separation - past, present, and future," in *We Will Show Them!*, 2005.

[12] N. Kamide, "Interpolation theorems for some variants of ltl," *Reports on Mathematical Logic*, vol. 2015, no. Number 50, 2015. [Online]. Available: https://www.ejournals.eu/rml/2015/Number-50/art/5718/

[13] M. Nascimento, "DTL: Translation, SMT Verification, Separation and Interpolation," 2021, Master's thesis, Instituto Superior Técnico.

[14] D. Oliveira, "Linear Temporal Logic: Separation and Translation," 2017, Master's thesis, Instituto Superior Técnico.

[15] D. Oliveira and J. Rasga, "Revisiting separation: algorithms and complexity," *Logic Journal of the IGPL*, 02 2020, jzz081. [Online]. Available: https://doi.org/10.1093/jigpal/jzz081

[16] D. Peled, "Temporal Logic: Mathematical Foundations and Computational Aspects, Volume 1," *The Computer Journal*, vol. 38, no. 3, pp. 260–261, 01 1995. [Online]. Available: https://doi.org/10.1093/comjnl/38.3.260

[17] A. Rabinovich, "A proof of kamp's theorem," *Log. Methods Comput. Sci.*, vol. 10, no. 1, 2014. [Online]. Available: https://doi.org/10.2168/LMCS-10(1:14)2014

[18] A. Reynolds, "Satisfiability modulo theories and DPLL(T)," 2015, last accessed 30 January 2020. [Online]. Available: http://homepage.divms.uiowa.edu/~ajreynol/pres-dpllt15.pdf

[19] A. Reynolds, T. King, and V. Kuncak, "Solving quantified linear arithmetic by counterexample-guided instantiation," *Formal Methods in System Design*, 2017.

[20] K. Schneider, *Verification of Reactive Systems: Formal Methods and Algorithms*. SpringerVerlag, 2004.