Linear Temporal Logic: Separation and Translation

Daniel Rebelo de Oliveira

Thesis to obtain the Master of Science Degree in

Mathematics and Applications

Supervisor: Prof. João Filipe Quintas dos Santos Rasga

November 2017

Acknowledgments

I would like to thank my advisor, Professor João Rasga, for sticking with me and helping me through this process. And my family, for everything they have done for me.

Resumo

Lógica temporal linear com as modalidades Since e Until tem o mesmo poder expressivo, sobre ordens lineares completas, que um fragmento de primeira-ordem conhecido como FOMLO. Também se sabe que uma lógica temporal linear, assumindo algumas propriedades básicas, tem a mesma expressividade que FOMLO se e só se tem uma propriedade, chamada separação, que qualquer fórmula é equivalente a uma combinação Booleana de fórmulas tal que cada uma delas apenas considera o passado, presente ou futuro.

Neste texto apresenta-se algoritmos simples e as suas implementações para fazer esta separação da lógica temporal linear com Since e Until, sobre ordens discretas e completas, e tradução de FOMLO para esta mesma lógica.

Palavras-chave: Lógica Temporal, LTL, FOMLO, Separação, Tradução, Completude Expressiva

Abstract

Linear temporal logic with Since and Until modalities is expressively equivalent, over the class of complete linear orders, to a fragment of first-order logic known as FOMLO. It turns out that a linear temporal logic, under some basic assumptions, is expressively complete if and only if it has the property, called separation, that every formula is equivalent to a Boolean combination of formulas that each refer only to the past, present or future.

Here we present simple algorithms and their implementations to perform separation of this linear temporal logic with Since and Until, over discrete and complete linear orders, and translation from FOMLO formulas into equivalent temporal logic formulas.

Keywords: Temporal Logic, LTL, FOMLO, Separation, Translation, Expressive Completeness

Contents

1 Introduction

In 1957 in [1], Arthur Prior introduced Tense Logic, by extending propositional logic with two operators P and F, with the intended semantics being such that $P\varphi$ is true if φ was true at some time in the past, and F meaning the same but in the future direction.

In [2], Amir Pnueli proposed the use of this logic in the analysis of properties of computer programs, and while Prior's Tense Logic is strong enough to express many useful properties, there are still properties that are not expressible. Examples of properties of practical interest that cannot be expressed can be found in [3].

The question of how expressive these logics are naturally arises. A way to measure their expressiveness is by considering a fragment of first-order logic which seems to appropriately describe everything that we ought to be able to say, and then checking if all these properties can also be expressed in temporal logic. Or, more precisely, 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 from temporal logic to the first-order fragment).

In 1968 in [4], Hans Kamp introduced two temporal operators called Since and Until. With semantics such that $A\,\text{Until}\,B$ means that in the future B will be true and until then A is always true, and similarly for Since but in the direction of the past. He also defined a fragment of first-order now known as FOMLO (First-Order Monadic Logic of Order) which appears appropriate in the sense described above, and showed that if we consider a Dedekind-complete linear model of time, then a temporal logic with just these two operators is expressively complete. A shorter proof of this result can be found in [5].

Still in [3], it is shown that a logic with only the temporal operator Until is expressively complete over the naturals, when considering satisfaction at the initial point of time. Or alternatively over complete and discrete linear orders when considering a fragment of FOMLO that restricts quantifiers such that they can only do bounded quantification and never into the past. This result also implies that, at the initial point, having the additional Since operator does not add expressive power. Although it does not add expressive power, it still gives us something, namely succinctness. In [6] it is shown that there are formulas in the language with both operators such that an equivalent formula using only Until is necessarily at least exponentially larger. Even so, nowadays, this language only with Until is quite popular and has found many practical applications, for example in the analysis of the correctness of computer programs and systems with concurrent processes. This language is now known as LTL (Linear Temporal Logic).

Dov Gabbay showed that this logic with both Since and Until has the property, called separation, that for every formula there is an equivalent formula consisting of a Boolean combination of formulas that each talk only about the past, present or future. This is mentioned already in [3] and a proof of this over the integers can be found in [7], it was later extended to Dedekind-complete linear time in [8]. This property not only leads to proof a Kamp's theorem, but it turns out that, over any class of linear orders, a temporal logic has the separation property if and only if it is expressively complete, provided that the temporal logic can at least express Prior's P and F operators. This is shown in [8] and [9].

In this text we provide simple algorithms to perform separation and translation. We show that they are sound and complete in the sense given any formula of the temporal language with Since and Until, the separation algorithm constructs a separated formula equivalent to the first over complete and discrete linear time, and given any FOMLO formula, the translation algorithm constructs an equivalent temporal logic formula. In particular we show that these algorithms always terminate.

The translation algorithm is a simplified version of the algorithm that can be extracted from theorem 2.3 in [7]. The same translation algorithm can in fact be used to translate from FOMLO to any expressively complete temporal logic, when provided with a separation algorithm for the logic. We also give an algorithm to translate FOMLO to LTL, which only requires a small addition to the main translation algorithm.

The algorithm that can be extracted from Gabbay's proof of separation over integer time is somewhat complicated. It works by using a family of algorithms that can separate a restricted class of formulas, with the last one being able to perform separation on every formula. Each algorithm considers the whole formula, identifying the most "unseparated" subformulas and substituing certain subformulas with atoms so that the formula as a whole is of a form that can be processed with the previous algorithm. Then it substitutes the previously introduced atoms back with the subformulas they had replaced, and continues recursively. We instead opt for a simple unified recursive algorithm to perform separation, and offload most of the complexity to the proof of correctness. We also simplify some of basic steps in the separation process and provide detailed proofs after introducing some concepts and notation that make it easier to do so.

All of the algorithms we describe have been implemented in the Haskell programming language, and can be found in the main text.

2 Temporal Logic and FOMLO

In this chapter we introduce the logics and some notation. For the remainder of this text, unless explicitly mentioned, when we say "temporal logic" or "TL", we mean the temporal logic defined in this chapter.

2.1 Syntax

Fix a countable set Pred that will serve as both the propositional variables in temporal logic and monadic predicates in FOMLO.

Definition 1. The language of temporal logic $\mathcal{L}_{\text{TL}}$ is defined inductively by:

- $\bot$
- $\top$
- $\neg A \quad (A \in \mathcal{L}_{TL})$
- $A \vee B \quad (A, B \in \mathcal{L}_{TL})$
- $A \wedge B \quad (A, B \in \mathcal{L}_{TL})$
- $p \quad (p \in \text{Pred})$
- $A\mathcal{S}B \quad (A, B \in \mathcal{L}_{TL})$
- $A\mathcal{U}B \quad (A, B \in \mathcal{L}_{TL})$

Definition 2. The set $\mathcal{L}_{\text{LTL}}$ of LTL formulas is defined just like the one of TL, but omitting the $\mathcal{S}$ case.

Definition 3. Some additional temporal operators:

- $\newmoon A := \bot \mathcal{S} A$ (Previous)
- $\blacklozenge A := \top \mathcal{S} A$ (Eventually in the past)
- $\blacksquare A := \neg \blacklozenge \neg A$ (Forever in the past)
- $\bigcirc A := \bot \mathcal{U} A$ (Next)
- $\lozenge A := \top \mathcal{U} A$ (Eventually)
- $\square A := \neg \lozenge \neg A$ (Forever)

Definition 4. We say a formula A is *simple* if it has no outer Boolean structure, that is, if it is of the form p, $B\mathcal{S}C$, or $B\mathcal{U}C$ (for some $p \in$ Pred and $B, C \in \mathcal{L}_{\text{TL}}$).

Definition 5. A formula is called *non-future* if it has no occurrences of $\mathcal{U}$ and *non-past* if it has no occurrences of $\mathcal{S}$.

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

A formula is *pure present* if it is a Boolean combination of variables.

Definition 6. A formula is *separated* if it is a Boolean combination of pure formulas.

For FOMLO, we also need a set of variables, we call it Var and assume it is countable and infinite.

Definition 7. The set $\mathcal{L}_{\text{FOMLO}}$ of FOMLO formulas is defined inductively by:

- $\bot$
- $\top$
- $\neg\alpha$ $(\alpha \in \mathcal{L}_{\text{FOMLO}})$
- $\alpha \vee \beta$ $(\alpha, \beta \in \mathcal{L}_{\text{FOMLO}})$
- $\alpha \wedge \beta$ $(\alpha, \beta \in \mathcal{L}_{\text{FOMLO}})$
- $P(x)$ $(P \in \text{Pred}, x \in \text{Var})$
- $x = y$ $(x, y \in \text{Var})$
- $x < y$ $(x, y \in \text{Var})$
- $\exists_x \alpha$ $(x \in \text{Var}, \alpha \in \mathcal{L}_{\text{FOMLO}})$
- $\forall_x \alpha$ $(x \in \text{Var}, \alpha \in \mathcal{L}_{\text{FOMLO}})$

In both TL and FOMLO, we additionally define $\varphi \to \psi$ as an abbreviation for $\neg\varphi \vee \psi$.

We use lower case letters from the latin alphabet to refer to elements of Pred in the context of temporal logic, and upper case latin latters to refer to these same elements in the context of FOMLO. Additionally, each letter refers to the same predicate symbol in both the lower case and upper case versions, for example p and P refer to the same predicate symbol.

2.2 Semantics

We will consider interpretation structures over a signature with a binary predicate $<$ and countable unary predicates $P \in$ Pred, with the interpretation of $<$ a *complete* and *discrete* linear order. Where *complete* means that every non-empty bounded above subset has a supremum and every non-empty bounded below subset has an infimum, and *discrete* means that every non-maximal element has a successor and every non-minimal element has a predecessor. These conditions also imply that every infimum is in fact a minimum element of the subset, as if the infimum was not a minimum element, then its successor would be a larger lower bound for the subset. A similar reasoning also allows us to conclude that every non-empty bounded above subset has a greatest element.

Definition 8. An interpretation structure I is a tuple $I = \left\langle D, <^I, \{P^I\}_{P \in \text{Pred}} \right\rangle$ where D is a non-empty set called the domain of I, $<^I$ is a complete and discrete linear order over D, and each $P^I \subseteq D$.

We write $\text{Domain}(I)$ to refer to this D.

In temporal logic, we use the notation $I, (t,s) \Vdash A$ to mean that all the points in the interval (t,s) satisfy A. That is, for all $r^{>t}_{<s}$: $I, r \Vdash A$. And similarly for $[t,s)$, $(t,s]$ and $[t,s]$, which mean $r^{\geq t}_{<s}$, $r^{>t}_{\leq s}$ and $r^{\geq t}_{\leq s}$, respectively. When we use $\nVdash$ with this interval notation, we mean that *no* point in the interval satisfies the formula, it does not mean that some point in the interval fails to satisfy.

This use of $<$ should in fact have been $<^I$. We will continue to write $<$ to refer to $<^I$ when there is no risk of confusion. We might also write $t \in I$ to mean $t \in \mathrm{Domain}(I)$.

Definition 9. Let I be an interpretation structure and $t \in I$. Then we define satisfaction of a temporal logic formula by the structure I at the point t by:

- $I, t \nVdash_{TL} \bot$
- $I, t \Vdash_{TL} \top$
- $I, t \Vdash_{TL} \neg A$ iff $I, t \nVdash_{TL} A$
- $I, t \Vdash_{TL} A \vee B$ iff $I, t \Vdash_{TL} A$ or $I, t \Vdash_{TL} B$
- $I, t \Vdash_{TL} A \wedge B$ iff $I, t \Vdash_{TL} A$ and $I, t \Vdash_{TL} B$
- $I, t \Vdash_{TL} p$ iff $t \in P^I$
- $I, t \Vdash_{TL} A\mathcal{S}B$ iff there is an $s <^I t$ such that $I, s \Vdash_{TL} B$ and $I, (s,t) \Vdash_{TL} A$
- $I, t \Vdash_{TL} A\mathcal{U}B$ iff there is an $s >^I t$ such that $I, s \Vdash_{TL} B$ and $I, (t,s) \Vdash_{TL} A$

Naturally, in the last two cases, s and (s,t) and (t,s) are over the domain of I.

We call the s in the definition of $\mathcal{S}$ ($\mathcal{U}$) satisfaction a *witness* for $A\mathcal{S}B$ ($A\mathcal{U}B$) at t.

Satisfaction for FOMLO is defined in the usual way.

Definition 10. An assignment ρ into an interpretation structure I is a function $\rho : \mathsf{Var} \to \mathrm{Domain}(I)$.

We use the notation $[x \mapsto x_0, y \mapsto y_0]$ to denote an assignment ρ such that $\rho(x) = x_0$ and $\rho(y) = y_0$. And the notation $\rho[x \mapsto x_0, y \mapsto y_0]$ for an assignment identical to ρ except possibly at x and y, to which it assigns x_0 and y_0, respectively.

We write $I, \rho \Vdash_{\mathsf{FOMLO}} \varphi$ to mean that the interpretation I satisfies φ under assignment ρ.

In both TL and FOMLO, we may simply write $\Vdash$ when it is clear which logic we mean. And in TL we may write $t \Vdash A$ instead of $I, t \Vdash A$ when it is clear from the context which interpretation we are considering.

We use $\equiv$ to denote semantic equivalence of formulas. That is, in the case of temporal logic, $A \equiv B$ means that for all interpretations I and points t of I: $I, t \Vdash A$ if and only if $I, t \Vdash B$. And in the case of FOMLO, $\varphi \equiv \psi$ means that for all interpretations I and assignments ρ: $I, \rho \Vdash \varphi$ if and only if $I, \rho \Vdash \psi$.

3 Separation

Definition 11. The *dual* of a formula is obtained by replacing every $\mathcal{S}$ with an $\mathcal{U}$, and vice-versa. We write $\mathrm{Dual}(A)$ for the dual of A.

Proposition 12 (Distribution)**.** *For all $A, B \in \mathcal{L}_{TL}$:*

- $\left(\bigwedge_{i=1}^{n} A_i\right)\mathcal{S}B \equiv \bigwedge_{i=1}^{n} (A_i\mathcal{S}B)$
- $\left(\bigwedge_{i=1}^{n} A_i\right)\mathcal{U}B \equiv \bigwedge_{i=1}^{n} (A_i\mathcal{U}B)$
- $A\mathcal{S}\left(\bigvee_{i=1}^{n} B_i\right) \equiv \bigvee_{i=1}^{n} (A\mathcal{S}B_i)$
- $A\mathcal{U}\left(\bigvee_{i=1}^{n} B_i\right) \equiv \bigvee_{i=1}^{n} (A\mathcal{U}B_i)$

The algorithm works by recursively separating the immediate subformulas, which is all that is required for a Boolean operator. Instead of worrying about both $\mathcal{S}$ and $\mathcal{U}$, the $\mathcal{U}$ case is handled by duality and the bulk of the algorithm is on how to separate a $\mathcal{S}$.

When trying to separate a $\mathcal{S}$, the distribution results help in the following way: Consider a formula $A\mathcal{S}B$ where A and B are already separated. A and B can be an arbitrary Boolean combination of simple pure formulas. If we convert the outer Boolean structure of A to conjunctive normal form and the one of B to disjunctive normal form, we can then use distribution to "split" $A\mathcal{S}B$ into a Boolean combination of formulas of the form $C\mathcal{S}D$, where the outer Boolean structures are much simpler, namely C is a disjunctive clause of simple pure formulas and D is a conjunctive clause of simple pure formulas. We then only have to worry about "pulling" the simple pure $\mathcal{U}$s from inside the $\mathcal{S}$ in this narrower case where the left side is a disjunctive clause and the right side is a conjunctive clause. We then eliminate one $\mathcal{U}$ from inside the $\mathcal{S}$, removing it from both sides at once. There are eight possibilities for each, it can occur only on the left, or only on the right, or in both, and it can appear negated or not. Having pulled out one of the $\mathcal{U}$s, we continue separating the result recursively.

When we write $(\neg?)$ in the algorithm, we mean to allow the possibility of a negation occurring there.

The normal form conversions are only over the outer Boolean structure of a formula, treating all simple formulas as atoms. And we assume that a normal form has no repeated literals and no complementary literals.

The *temporal depth* of a formula is defined later, but it is simply the maximal depth of temporal operator nesting, analogous to quantifier depth.

```
**procedure** Sep($X$)
    $^0$**if** $X$ is separated **then return** $X$
    $^1$**else if** $X = \neg A$ **then return** $\neg$Sep$(A)$
    $^2$**else if** $X = A \vee B$ **then return** Sep$(A) \vee$ Sep$(B)$
    $^3$**else if** $X = A \wedge B$ **then return** Sep$(A) \wedge$ Sep$(B)$
    $^4$**else if** $X = A\mathcal{S}B$ with $A$ a separated disjunctive clause, and $B$ a separated conjunctive clause
        **let** $\bigvee_{i=1}^{m} (\neg?)A_i := A$ ; **let** $\bigwedge_{j=1}^{n} (\neg?)B_j := B$
        **let** $\mathbf{C} := \{(\neg?)A_i \mid A_i$ is a pure future formula, $i \in [1,m]\}$ ; **let** $\mathbf{A} := \{(\neg?)A_i \mid i \in [1,m]\} - \mathbf{C}$
        **let** $\mathbf{D} := \{(\neg?)B_j \mid B_j$ is a pure future formula, $j \in [1,n]\}$ ; **let** $\mathbf{B} := \{(\neg?)B_j \mid j \in [1,n]\} - \mathbf{D}$
        **let** $(\neg?)(F\mathcal{U}G) \in \mathbf{C} \cup \mathbf{D}$ with maximal temporal depth in $\mathbf{C} \cup \mathbf{D}$
        **let** $A' := \bigvee((\mathbf{A} \cup \mathbf{C}) - \{F\mathcal{U}G, \neg(F\mathcal{U}G)\})$ ; **let** $B' := \bigwedge((\mathbf{B} \cup \mathbf{D}) - \{F\mathcal{U}G, \neg(F\mathcal{U}G)\})$
        $^{4.1}$**if** $F\mathcal{U}G \in \mathbf{C}$ and $F\mathcal{U}G, \neg(F\mathcal{U}G) \notin \mathbf{D}$ **then return** Sep($\mathcal{T}_1(A', B', F, G)$)
        $^{4.2}$**else if** $F\mathcal{U}G, \neg(F\mathcal{U}G) \notin \mathbf{C}$ and $F\mathcal{U}G \in \mathbf{D}$ **then return** Sep($\mathcal{T}_2(A', B', F, G, )$)
        $^{4.3}$**else if** $F\mathcal{U}G \in \mathbf{C}$ and $F\mathcal{U}G \in \mathbf{D}$ **then return** Sep($\mathcal{T}_3(A', B', F, G)$)
        $^{4.4}$**else if** $\neg(F\mathcal{U}G) \in \mathbf{C}$ and $F\mathcal{U}G, \neg(F\mathcal{U}G) \notin \mathbf{D}$ **then return** Sep($\mathcal{T}_4(A', B', F, G)$)
        $^{4.5}$**else if** $F\mathcal{U}G, \neg(F\mathcal{U}G) \notin \mathbf{C}$ and $\neg(F\mathcal{U}G) \in \mathbf{D}$ **then return** Sep($\mathcal{T}_5(A', B', F, G, )$)
        $^{4.6}$**else if** $F\mathcal{U}G \in \mathbf{C}$ and $\neg(F\mathcal{U}G) \in \mathbf{D}$ **then return** Sep($\mathcal{T}_6(A', B', F, G, )$)
        $^{4.7}$**else if** $\neg(F\mathcal{U}G) \in \mathbf{C}$ and $\neg(F\mathcal{U}G) \in \mathbf{D}$ **then return** Sep($\mathcal{T}_7(A', B', F, G, )$)
        $^{4.8}$**else if** $\neg(F\mathcal{U}G) \in \mathbf{C}$ and $F\mathcal{U}G \in \mathbf{D}$ **then return** Sep($\mathcal{T}_8(A', B', F, G, )$)
    $^5$**else if** $X = A\mathcal{S}B$ with $A$ and $B$ separated
        **let** $\bigwedge_{i=1}^{m} A_i$ be a CNF of $A$ ; **let** $\bigvee_{j=1}^{n} B_j$ be a DNF of $B$
        **if** $m = 0$ **then** set $m$ to $1$ and $A_1$ to $\top$ ; **if** $n = 0$ **then** set $n$ to $1$ and $B_1$ to $\bot$
        **return** $\bigwedge_{i=1}^{m} \bigvee_{j=1}^{n}$ Sep$(A_i\mathcal{S}B_j)$
    $^6$**else if** $X = A\mathcal{S}B$ **then return** Sep(Sep$(A)\mathcal{S}$Sep$(B)$)
    $^7$**else if** $X = A\mathcal{U}B$ **then return** $\mathrm{Dual}($Sep$(\mathrm{Dual}(X)))$
```

The transformations are defined in the following tables, when called with arguments (A, B, F, G).

$\mathcal{T}_1$	$P \wedge P'\mathcal{S}B$
	• $N := (\neg G \wedge \neg B)\mathcal{S}(\neg A \wedge \neg B)$
	• $P' := N \rightarrow G \vee F$
	• $P := N \rightarrow G \vee (F \wedge F\mathcal{U}G)$
$\mathcal{T}_2$	$H_< \vee H_\geq$
	• $H_< := A\mathcal{S}(G \wedge A \wedge (A \wedge F)\mathcal{S}B)$
	• $H_\geq := (A \wedge F)\mathcal{S}B \wedge (G \vee (F \wedge F\mathcal{U}G))$
$\mathcal{T}_3$	$H_< \vee H_\geq$
	• $H_< := P \wedge P'\mathcal{S}(G \wedge F\mathcal{S}B)$
	• $H_\geq := F\mathcal{S}B \wedge (G \vee (F \wedge F\mathcal{U}G))$
	• $N := \neg G\mathcal{S}\neg A$
	• $P' := N \rightarrow G \vee F$
	• $P := N \rightarrow G \vee (F \wedge F\mathcal{U}G)$

$\mathcal{T}_4$	$\neg\mathcal{T}_2(\neg B, \neg A \wedge \neg B, F, G) \vee \blacklozenge B$
$\mathcal{T}_5$	$A\mathcal{S}(\neg F \wedge \neg G \wedge A \wedge (A \wedge \neg G)\mathcal{S}B)$
	$\vee\ (A \wedge \neg G)\mathcal{S}B \wedge \neg G \wedge (\neg F \vee \neg(F\mathcal{U}G))$
$\mathcal{T}_6$	$\mathcal{T}_1(A, (A \wedge \neg G)\mathcal{S}B \wedge \neg G \wedge \neg F \wedge A, F, G)$
	$\vee\ \mathcal{T}_3(A, (A \wedge \neg G)\mathcal{S}B \wedge \neg G \wedge \neg F, F, G)$
	$\vee\ (A \wedge \neg G)\mathcal{S}B \wedge \neg G \wedge (\neg F \vee \neg(F\mathcal{U}G))$
$\mathcal{T}_7$	$\neg(\mathcal{T}_3(\neg B, \neg A, F, G)) \wedge \mathcal{T}_5(\top, B, F, G)$
$\mathcal{T}_8$	$\mathcal{T}_4(A, (A \wedge F)\mathcal{S}B \wedge G \wedge A, F, G)$
	$\vee\ \mathcal{T}_7(A, (A \wedge F)\mathcal{S}B \wedge G, F, G)$
	$\vee\ (A \wedge F)\mathcal{S}B \wedge (G \vee (F \wedge F\mathcal{U}G))$

Definition 13. A *path* is a finite sequence over $\{\mathcal{S}, \mathcal{U}\}$. We use $\langle\rangle$ to denote the empty path.

To avoid confusion, we call a path (in the usual sense of graph theory) on the syntax tree of a formula, a *syntax path*. And we call the path corresponding to a syntax path the sequence obtained by considering the labels on the syntax path and then deleting the labels outside of $\{\mathcal{S}, \mathcal{U}\}$. This leads to the definition of the paths of formula.

Definition 14. Given a formula A, $\mathrm{Paths}(A)$ is the set of paths corresponding with the syntax paths from the root of the syntax tree of A to a leaf.

The following definition, used in the algorithm, is why we needed to define Paths beforehand.

Definition 15. The *temporal depth* of a formula A is the maximum length of a path of A.

Definition 16. Given a path π, the *degree* of π, $\mathcal{D}(\pi)$, is the number of adjacent pairs in π with both $\mathcal{S}$ and $\mathcal{U}$. Or, perhaps more intuitively, it is the number of transitions from $\mathcal{S}$ to $\mathcal{U}$ and vice-versa.

Definition 17. Let π be a path. We define $\mathcal{L}_1(\pi)$ to be the length of the last *homogeneous segment*, except in the case where the segment is the whole path, in which case we take $\mathcal{L}_1(\pi)$ to be zero. That is:

$$\mathcal{L}_1(\pi) := \begin{cases} n+1 & \text{if } \pi = \lambda\mathcal{S}\mathcal{U}^{n+1} \text{ or } \pi = \lambda\mathcal{U}\mathcal{S}^{n+1} \\ 0 & \text{otherwise} \end{cases}$$

For convenience, we combine these into one.

Definition 18. Let π be a path. Then $\mathrm{Score}_1(\pi) := \langle \mathcal{D}(\pi), \mathcal{L}_1(\pi) \rangle$.

So far these have been over paths, we now extend the definitions to formulas.

Whenever we have a Cartesian product of orders, the intended order on it is the usual lexicographic order. That is, if $\langle x, y\rangle, \langle x', y'\rangle \in X \times Y$ with $<_X$ an order over X and $<_Y$ an order over Y, $\langle x, y\rangle < \langle x', y'\rangle$ if and only if $x <_X x'$ or ($x = x'$ and $y <_Y y'$). If all such orders are well-founded then so is the lexicographic order, and if they are total then so is the lexicographic order.

Definition 19. Let $A \in \mathcal{L}_{\mathsf{TL}}$. Then:

$$\mathrm{Score}_1(A) := \begin{cases} \langle 0, 0\rangle & \text{if } \mathrm{Paths}(A) = \emptyset \\ \max \mathrm{Score}_1[\mathrm{Paths}(A)] & \text{otherwise} \end{cases}$$

To obtain something that is reduced by every transformation, we need an additional thing, which we call leader count. And whose definition requires the idea of contexted formula.

Definition 20. A *contexted formula* is a pair $\langle \pi, A \rangle$ where π is a path and A is a formula.

Definition 21. Let A be a formula. Then $\mathcal{C}(A)$ is a set of contexted formulas, consisting of the subformulas of A together with their context (the path corresponding to the syntax path that leads to their subtree).

For the purposes of our algorithm, it suffices to look at the subset of these contexted subformulas where the subformula is simple and pure, which leads to the following definition.

Definition 22. Let $A \in \mathcal{L}_{\mathsf{TL}}$. Then $\mathcal{SPC}(A) := \{\langle \pi, B \rangle \in \mathcal{C}(A) \mid B \text{ is simple and pure}\}$.

We now extend the definition of Paths and Score_1 to contexted formulas in a natural way.

Definition 23. Let $\langle \pi, A \rangle$ be a contexted formula. Then:

$$\mathrm{Paths}(\langle \pi, A \rangle) := \{\pi\lambda \mid \lambda \in \mathrm{Paths}(A)\}$$

$$\mathrm{Score}_1(\langle \pi, A \rangle) := \begin{cases} \langle 0, 0 \rangle & \text{if } \mathrm{Paths}(\langle \pi, A \rangle) = \emptyset \\ \max \mathrm{Score}_1[\mathrm{Paths}(\langle \pi, A \rangle)] & \text{otherwise} \end{cases}$$

With this, we can define the concepts of leader and leader count.

Definition 24. Let $A \in \mathcal{L}_{\mathsf{TL}}$ and $\langle \pi, B \rangle \in \mathcal{SPC}(A)$.

Then we say $\langle \pi, B \rangle$ is a *leader* of A if it has maximal Score_1 in $\mathcal{SPC}(A)$ and call $\mathrm{Leaders}(A)$ the set of leaders of A. The leader count of A, $\mathcal{N}(A)$, is the number of leaders of A with distinct formulas, that is: $\mathcal{N}(A) := \#\{B \mid \langle \pi, B \rangle \in \mathrm{Leaders}(A)\}$.

So far we have defined Score_1 and $\mathcal{N}$ on formulas, and indeed these together are enough to obtain something that is reduced by every transformation when used in the conditions of the algorithm.

We now define another pair of functions, quite similar to $\mathcal{L}_1$ and Score_1, which are necessary for case 6 of the algorithm.

Definition 25. Let π be a path. Then $\mathcal{L}_0(\pi)$ is the length of the first homogeneous segment, with the exception of a fully homogeneous path. Score_0 is analogous to Score_1.

$$\mathcal{L}_0(\pi) := \begin{cases} n+1 & \text{if } \pi = \mathcal{S}^{n+1}\mathcal{U}\lambda \text{ or } \pi = \mathcal{U}^{n+1}\mathcal{S}\lambda \\ 0 & \text{otherwise} \end{cases} \qquad \mathrm{Score}_0(A) := \begin{cases} \langle 0, 0 \rangle & \text{if } \mathrm{Paths}(A) = \emptyset \\ \max \mathrm{Score}_0[\mathrm{Paths}(A)] & \text{otherwise} \end{cases}$$

$$\mathrm{Score}_0(\pi) := \langle \mathcal{D}(\pi), \mathcal{L}_0(\pi) \rangle$$

For convenience, we bundle Score_1, $\mathcal{N}$ and Score_0 into a single function, called *weight*.

Definition 26. Let A be a formula. Then $\mathcal{W}(A) := \langle \mathrm{Score}_1(A), \mathcal{N}(A), \mathrm{Score}_0(A) \rangle$.

Definition 27. We define the binary relations on formulas $<_{\mathcal{W}}$, $\prec$, $\lessdot$, $\lhd_l$, $\lhd_r$ and $\lhd_u$ as:

- $A <_{\mathcal{W}} B$ if and only if $\mathcal{W}(A) < \mathcal{W}(B)$
- $A \prec B$ if and only if A is an immediate proper subformula of B
- $A \lessdot B$ if and only ($A = C'\mathcal{S}D$, $B = C\mathcal{S}D$ and $C' \prec^+ C$) or ($A = C\mathcal{S}D'$, $B = C\mathcal{S}D$ and $D' \prec^+ D$)
- $A \lhd_l B$ if only if $A = C'\mathcal{S}D$, $B = C\mathcal{S}D$, C is not in CNF and C' is a CNF of C
- $A \lhd_r B$ if only if $A = C\mathcal{S}D'$, $B = C\mathcal{S}D$, D is not in DNF and D' is a DNF of D
- $A \lhd_u B$ if only if $A = C\mathcal{S}D$ and $B = \mathrm{Dual}(A)$

Definition 28. The binary relation $< \subseteq \mathcal{L}_{\mathsf{TL}} \times \mathcal{L}_{\mathsf{TL}}$ is defined as $< := (<_{\mathcal{W}} \cup \prec \cup \lessdot \cup \lhd_l \cup \lhd_r \cup \lhd_u)^+$.

This relation is indeed a well-founded partial order.

Theorem 29. *For every $X \in \mathcal{L}_{TL}$ we have the following:*

(i) Any Sep calls that Sep(X) reduces to are with a smaller argument (i.e. Sep(X) terminates).

(ii) $X \equiv$ Sep(X)

(iii) Sep(X) is separated.

Proof. The proof is by induction on the order $<$ defined in 28. □

4 Translation

In this section we define an algorithm that given a FOMLO formula with at most one free variable, produces an equivalent temporal formula. This algorithm makes crucial use of the separation algorithm defined previously.

Intuitively, the main difficulty in performing this conversion is that in FOMLO we can always refer to any variable, no matter how many more levels of quantification there have been since the quantifier that introduced a variable, while temporal logic has a sequential nature, only allowing us to relate the "current point" with the next. This is overcome by converting the binary predicates in FOMLO to unary ones, "Before", "Now" and "After", that carry the same information, essentially converting FOMLO into truly monadic first-order logic without equality.

Definition 30. We say that a FOMLO formula φ is *pulled out* if for every subformula of φ of the form $P(x)$, if $P(x)$ occurs inside the scope of a quantifier then the deepest quantifier inside whose scope $P(x)$ occurs binds x.

Definition 31 (Pullout)**.** Given a FOMLO formula φ, Pullout(φ) is an equivalent pulled-out formula.

The following definition makes clear how we are getting rid of the binary predicates.

Definition 32 (Extend)**.** Given a variable t and a formula φ, Extend(t, φ) is a formula over a signature extended with three new unary predicate symbols Before, Now and After, obtained from φ by substituting $(t = t)$ with $\top$, $(t < t)$ with $\bot$, $(x < t)$ with $\mathrm{Before}(x)$, $(x = t)$ and $(t = x)$ with $\mathrm{Now}(x)$, and $(t < x)$ with $\mathrm{After}(x)$, whenever they occur in a context where t is free.

These predicates are ordinary predicates, all we have to do is add three new symbols to Pred.

And then to remove these extra monadic predicates:

Definition 33 (Unextend)**.** Given a separated temporal formula A over an extended signature, Unextend(A) is the formula over the unextended signature obtained from A by performing the following replacements:

- In every pure past subformula, Before, Now and After are replaced with $\top$, $\bot$ and $\bot$, respectively.
- In every pure present subformula, Before, Now and After are replaced with $\bot$, $\top$ and $\bot$, respectively.
- In every pure future subformula, Before, Now and After are replaced with $\bot$, $\bot$ and $\top$, respectively.

We can finally define the translation algorithm.

```
procedure Translate($\varphi$)
    return Translate'(Pullout($\varphi$))
procedure Translate'($\varphi$)
```

```
let $t$ be the free variable of $\varphi$ (or an arbitrary variable if $\varphi$ is a sentence)
$^0$if $\varphi = \bot$ then return $\bot$
$^1$else if $\varphi = \top$ then return $\top$
$^2$else if $\varphi = P(t)$ then return $p$
$^3$else if $\varphi = (t = t)$ then return $\top$
$^4$else if $\varphi = (t < t)$ then return $\bot$
$^5$else if $\varphi = \neg\alpha$ then return $\neg$Translate'$(\alpha)$
$^6$else if $\varphi = \alpha \vee \beta$ then return Translate'$(\alpha)$ $\vee$ Translate'$(\beta)$
$^7$else if $\varphi = \alpha \wedge \beta$ then return Translate'$(\alpha)$ $\wedge$ Translate'$(\beta)$
$^8$else if $\varphi = \exists_s\alpha$
    let $A =$ Translate'(Extend$(t, \alpha)$)
    return Unextend(Sep($\blacklozenge A \vee A \vee \lozenge A$))
$^9$else if $\varphi = \forall_s\alpha$
    let $A =$ Translate'(Extend$(t, \alpha)$)
    return Unextend(Sep($\blacksquare A \wedge A \wedge \Box A$))
```

Theorem 34. *For every $\varphi \in \mathcal{L}_{FOMLO}$ with at most one free variable, interpretation structure I and point t_0 of I:*

$$I, [t \mapsto t_0] \Vdash_{FOMLO} \varphi \text{ if and only if } I, t_0 \Vdash_{TL} \textit{Translate}(\varphi)$$

Proof. The proof is by induction on the union of the subformula order and the order induced by quantifier depth. □

4.1 Extensions

4.1.1 LTL

This algorithm can also be adapted to translate FOMLO into LTL. But then we require that the order have a minimum element, and the equivalence only holds at that minimum element. This is quite natural as LTL cannot talk about the past since it has no past operators.

This can be done by performing the translation as before and then simply replacing any past formulas with $\bot$.

Definition 35 (ToLTL)**.** Let $A \in \mathcal{L}_{\mathrm{TL}}$. Then ToLTL$(A)$ is obtained from A by replacing every subformula of the form $B\mathcal{S}C$ with $\bot$.

Proposition 36. *For every $\varphi(t) \in \mathcal{L}_{FOMLO}$ with at most one free variable and all interpretations I with a minimum point t_0:*

$$I, [t \mapsto t_0] \Vdash_{FOMLO} \varphi \text{ if and only if } I, t_0 \Vdash_{LTL} \textit{ToLTL}(\textit{Translate}(\varphi))$$

Proof. The Translate algorithm always produces a separated formula, and so any subformula of the form $B\mathcal{S}C$ is pure past and is always false at the initial point. □

4.1.2 Expressively Complete Temporal Logics

The notion of separation we have described this text is known as *syntactic separation*. But a weaker notion of separation, called *semantic separation*, where we allow past operators inside future operators and vice-versa as long as e.g. each pure past formula does not "look" into the future, is sufficient for the translation algorithm to work.

Additionally, the algorithm makes no direct use of $\mathcal{S}$ or $\mathcal{U}$, the only temporal operators used are $\blacklozenge$ and $\lozenge$ ($\blacksquare$ and $\Box$ can be expressed using these). And it does make use of the fact that we have been working over complete and discrete linear orders. This means that the very same algorithm can be used to translate from FOMLO to other temporal logics, over any class of linear orders, provided they meet these conditions.

We now consider general temporal logics, and define one as being the usual propositional logic extended with any number of operators. And we say that such a logic can express ⧫ ($\Diamond$) if for every formula A in that logic there is a (computable) formula B such that ⧫A ($\Diamond A$) $\equiv B$.

Proposition 37. *Let L be a temporal logic able to express ⧫ and $\Diamond$ and Sep an algorithm that semantically separates L over a class of linear orders $\mathcal{C}$. Let I be an interpretation structure whose interpretation of $<$ is in $\mathcal{C}$ (not necessarily complete or discrete). Then, for every $\varphi \in \mathcal{L}_{\mathit{FOMLO}}$ and point t_0 of I:*

$$I, [t \mapsto t_0] \Vdash_{\mathit{FOMLO}} \varphi \text{ if and only if } I, t_0 \Vdash_L \mathit{Translate}(\varphi)$$

Proof. The proof is the same as the proof of theorem 34, noticing that the conditions on the logic imposed here are all that is used, and no other properties of the specific TL we have been using are required. □

5 Conclusions

We have written fairly simple algorithms to perform separation of TL and translation from FOMLO to TL over complete and discrete time. Since the actual translation algorithm is quite agnostic about the temporal logic used, it can also be used to translate to any expressively complete logic with computable separation. It is also possible to eliminate the discreteness requirement by extracting an algorithm from the proof of separation for complete time found in [8], perhaps even write a simple algorithm to perform this separation.

We have not talked about the complexity of these algorithms so far, but it is in fact known that the translation algorithm must have non-elementary complexity. As explained in [9] and [10], it is known that there is a non-elementary succintness gap between FOMLO and TL, that is, there are FOMLO formulas whose smallest equivalent TL formula is non-elementarily larger. This means that a translation algorithm must also have non-elementary complexity. Another way to see this is that the decidability of FOMLO even over the naturals is known to be non-elementary (see [11]), while the decidability of TL is in PSPACE (see [12]). This implies that a translation from FOMLO to TL must be non-elementary as well, as otherwise we would be able to decide FOMLO with an elementary algorithm by translating to TL and back.

Although the complexity of translation is non-elementary, in empirical testing using large amounts (tens of thousands) of randomly generated formulas of the kind of size, number of quantifiers and quantifier depth at the upper limit of that would be written by a human in practice, the speed of execution seems good enough for use in practice. Using a conventional personal computer the vast majority of formulas are translated in around a millisecond, some rare ones take a few minutes, and in very rare cases one finds a formula whose translation takes up high amounts of space and time.

As far as we know, the worst-case complexity of an algorithm that performs either syntactic or semantic separation for any linear temporal logic is still not known, although it is suspected to be non-elementary. A possible way to investigate this might be to try to construct a translation algorithm that would be elementary if separation was as well, thus proving that separation must be non-elementary. Perhaps by performing the elimination of binary predicates over the whole formula as a first pass and then using separation only once. The author tried the naive approach to this and was unsuccessful.

References

[1] A. N. Prior. *Time and modality*. John Locke Lecture, 2003.

[2] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society. doi: 10.1109/SFCS.1977.32. URL `http://dx.doi.org/10.1109/SFCS.1977.32`.

[3] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '80, pages 163–173, New York, NY, USA, 1980. ACM. ISBN 0-89791-011-7. doi: 10.1145/567446.567462. URL `http://doi.acm.org/10.1145/567446.567462`.

[4] H. Kamp. *Tense logic and the theory of linear orders*. PhD thesis, University of California, Los Angeles, 1968.

[5] A. Rabinovich. A proof of kamp's theorem. *Logical Methods in Computer Science*, 10(1), 2014. doi: 10.2168/LMCS-10(1:14)2014. URL `https://doi.org/10.2168/LMCS-10(1:14)2014`.

[6] N. Markey. Temporal Logic with Past is Exponentially More Succinct. *EATCS Bulletin*, 79:122–128, 2003. URL `https://hal.archives-ouvertes.fr/hal-01194627`.

[7] D. Gabbay. *The declarative past and imperative future*, pages 409–448. Springer Berlin Heidelberg, Berlin, Heidelberg, 1989. ISBN 978-3-540-46811-0. doi: 10.1007/3-540-51803-7_36. URL `http://dx.doi.org/10.1007/3-540-51803-7_36`.

[8] D. M. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic (Vol. 1): Mathematical Foundations and Computational Aspects*. Oxford University Press, Inc., New York, NY, USA, 1994. ISBN 0-19-853769-7.

[9] I. M. Hodkinson and M. Reynolds. Separation-past, present, and future. In *We Will Show Them!(2)*, pages 117–142, 2005.

[10] K. Etessami and T. Wilke. An until hierarchy and other applications of an ehrenfeucht-fraïssé game for temporal logic. *Inf. Comput.*, 160(1):88–108, July 2000. ISSN 0890-5401. doi: 10.1006/inco.1999.2846. URL `http://dx.doi.org/10.1006/inco.1999.2846`.

[11] L. J. Stockmeyer. *The complexity of decision problems in automata theory and logic.* PhD thesis, Massachusetts Institute of Technology, 1974.

[12] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, July 1985. ISSN 0004-5411. doi: 10.1145/3828.3837. URL `http://doi.acm.org/10.1145/3828.3837`.

[13] V. Goranko and A. Galton. Temporal logic. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2015 edition, 2015.