Positive and monotone fragments of FO and LTL

Simon Iosti
LIP, ENS Lyon, FRance

Denis Kuperberg
CNRS, LIP, ENS Lyon, France

Quentin Moreau
ENS Lyon, France

Abstract

We study the positive logic FO^+ on finite words, and its fragments, pursuing and refining the work initiated in [12]. First, we transpose notorious logic equivalences into positive first-order logic: FO^+ is equivalent to LTL^+, and its two-variable fragment FO^{2+} with (resp. without) successor available is equivalent to UTL^+ with (resp. without) the "next" operator X available. This shows that despite previous negative results, the class of FO^+-definable languages exhibits some form of robustness. We then exhibit an example of an FO-definable monotone language on one predicate, that is not FO^+-definable, refining the example from [12] with 3 predicates. Moreover, we show that such a counter-example cannot be FO^2-definable. Finally, we provide a new example distinguishing the positive and monotone versions of FO^2 without quantifier alternation. This does not rely on a variant of the previously known counter-example, and witnesses a new phenomenon.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Regular languages; Theory of computation → Logic and verification

Keywords and phrases Positive logic, LTL, separation, first-order, monotone

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Funding *Denis Kuperberg*: ANR ReCiProg

1 Introduction

In various contexts, monotonicity properties play a pivotal role. For instance the field of monotone complexity investigates negation-free formalisms, and turned out to be an important tool for complexity in general [7]. From a logical point of view, a sentence is called monotone (with respect to a predicate P) if increasing the set of values where P is true in a structure cannot make the evaluation of the formula switch from true to false. This is crucial e.g. when defining logics with fixed points, where the fixed points binders μX can only be applied to formulas that are monotone in X. Logics with fixed points are used in various contexts, e.g. to characterise the class PTIME on ordered structures [9, 21], as extensions of linear logic such as μMALL [2], or in the μ-calculus formalism used in automata theory and model-checking [3]. Because of the monotonicity constraint, it is necessary to recognise monotone formulas, and understand whether a syntactic restriction to positive (i.e. negation-free) formulas is semantically complete. Logics on words have also been generalised to inherently negation-free frameworks, such as in the framework of cost functions [4].

This motivates the study of whether the semantic monotone constraint can be captured by a syntactic one, namely the removing of negations, yielding the class of positive formulas. For instance, the formula $\exists x, a(x)$ states that an element labelled a is present in the structure. It is both monotone and positive. However, its negation $\forall x, \neg a(x)$ is neither positive nor monotone, since it states the absence of a, and increasing the domain where predicate a is true in a given structure could make the formula become false.

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:25
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Lyndon's preservation theorem [14] states that on arbitrary structures, every monotone formula of First-Order Logic (FO) is equivalent to a positive one (FO^+ syntactic fragment). The case of finite structures was open for two decades until Ajtai and Gurevich [1] showed that Lyndon's theorem does not hold in the finite, later refined by Stolboushkin [19] with a simpler proof. Recently, this preservation property of FO was more specifically shown to fail already on finite graphs and on finite words by Kuperberg [12], implying the failure on finite structures with a more elementary proof than [1, 19]. However, the relationship between monotone and positive formulas is still far from being understood. On finite words in particular, the positive fragment FO^+ was shown [12] to have undecidable membership (with input an FO formula, or a regular language), which could be interpreted as a sign that this class is not well-behaved. This line of research can be placed in the larger framework of the study of preservation theorems in first-order logic, and their behaviour in the case of finite models, see [18] for a survey on preservation theorems.

In this work we will concentrate on finite words, and investigate this "semantic versus syntactic" relationship for fragments of FO and Linear Temporal Logic (LTL). We will in particular lift the classical equivalence between FO and LTL [10] to their positive fragments, showing that some of the robustness aspects of FO are preserved in the positive fragment, despite the negative results from [12]. This equivalence between FO and LTL is particularly useful when considering implementations and real-world applications, as LTL satisfiability is PSPACE-complete while FO satisfiability is non-elementary. It is natural to consider contexts where specifications in LTL can talk about e.g. the activation of a sensor, but not its non-activation, which would correspond to a positive fragment of LTL. We could also want to syntactically force such an event to be "good" in the sense that if a specification is satisfied when a signal is off at some time, it should still be satisfied when the signal is on instead. It is therefore natural to ask whether a syntactic constraint on the positivity of LTL formulas could capture the semantic monotonicity, in the full setting or in some fragments corresponding to particular kinds of specifications.

We will also pay a close look at the two-variable fragment FO^2 of FO and its LTL counterpart. It was shown in [12] that there exists a monotone FO-definable language that is not definable in positive FO. We give stronger variants of this counter-example language, and show that such a counter-example cannot be defined in $FO^2[<]$. This is obtained via a stronger result characterising FO^2-monotone in terms of positive fragments of bounded quantifier alternation. We also give precise complexity results for deciding whether a regular language is monotone, refining results from [12]. Finally, we provide a counterexample showing that the positive and alternation-free fragment of FO^2 does not capture all alternation-free monotone languages from FO^2.

The goal of this work is to understand at what point the phenomenon discovered in [12] comes into play: what are the necessary ingredients for such a counter-example (FO-monotone but not FO positive) to exist? And on the contrary, which fragments of FO are better behaved, and can capture the monotonicity property with a semantic constraint, and allow for a decidable membership problem in the positive fragment?

Outline and Contributions

We begin by introducing two logical formalisms in Section 2: First-Order Logic (2.1) and Temporal Logic (2.2).

Then, we lift some classical logical equivalences to positive logic in Section 3. First we show that FO^+, FO^{3+} and LTL^+ are equivalent in Theorem 20. We prove that the fragment FO^{2+} with (resp. without) successor predicate is equivalent to UTL^+ with (resp. without)

X and Y operators available in Theorem 26 (resp. Corollary 28).

In Section 4, we give a characterisation of monotonicity using monoids (Theorem 29) and we deduce from this an algorithm which decides the monotonicity of a regular language given by a monoid (Section 4.2), completing the automata-based algorithms given in [12]. This leads us to the Proposition 31 which states that deciding the monotonicity of a regular language is in LogSpace when the input is a monoid while it is NL-complete when the input is a DFA. This completes the previous result from [12] showing PSpace-completeness for NFA input.

Finally, we study the relationship between semantic and syntactic positivity in Section 5. We give some refinements of the counter-example from [12] (a regular and monotone language FO-definable but not definable in FO^+). Indeed, we show that the counter-example can be adapted to FO^2 with the binary predicate "between" in Proposition 33, and we show that we need only one predicate to find a counter-example in FO in Proposition 34.

We also consider a characterisation of $FO^2[<]$ from Thérien and Wilke [20] stating that $FO^2[<]$ is equivalent to $\Sigma_2 \cap \Pi_2$ where Σ_2 and Π_2 are fragments of FO with bounded quantifier alternation. We show that FO^2-monotone is characterised by $\Sigma_2^+ \cap \Pi_2^+$.

We then show that no counter-example for FO can be found in FO^2 (without successor available) in Corollary 36, and leave open the problem of expressive equivalence between FO^{2+} and FO^2-monotone, as well as decidability of membership in FO^{2+} for regular languages (see Conjecture 37).

To conclude, we provide in Section 5.3 a negative answer to this problem in the context of the alternation-free fragment of FO^2, providing an example of a monotone language definable in FO^2 without quantifier alternation, but not definable in the positive fragment of FO^2 without alternation. This exhibits a new discrepancy between a positive fragment and its monotone counterpart, not relying on previous constructions.

Due to space constraints, some proofs are omitted or only sketched, and can be found in the Appendix.

2 FO and LTL

We work with a set of atomic unary predicates $\Sigma = \{a_1, a_2, ...a_{|\Sigma|}\}$, and consider the set of words on alphabet $A = \mathcal{P}(\Sigma)$. To describe a language on this alphabet, we use logical formulas. Here we present the different logics and how they can be used to define languages.

2.1 First-order logics

Let us consider a set of binary predicates, $=$, $\neq$, $\leq$, $<$, succ and nsucc, which will be used to compare positions in words. We define the subsets of predicates $\mathfrak{B}_0 := \{\leq, <, \mathrm{succ}, \mathrm{nsucc}\}$, $\mathfrak{B}_< := \{\leq, <\}$ and $\mathfrak{B}_{\mathrm{succ}} := \{=, \neq, \mathrm{succ}, \mathrm{nsucc}\}$, and a generic binary predicate is denoted $\mathfrak{b}$. As we are going to see, equality can be expressed with other binary predicates in $\mathfrak{B}_0$ and $\mathfrak{B}_<$ when we have at least two variables. This is why we do not need to impose that $=$ belongs to $\mathfrak{B}_0$ or $\mathfrak{B}_<$. The same thing stands for $\neq$. Generally, we will always assume that predicates $=$ and $\neq$ are expressible.

Let us start by defining first-order logic FO:

▶ **Definition 1.** *Let $\mathfrak{B}$ be a set of binary predicates. The grammar of* $FO[\mathfrak{B}]$ *is as follows:*

$$\varphi, \psi ::= \bot \mid \top \mid \mathfrak{b}(x, y) \mid a(x) \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \exists x, \varphi \mid \forall x, \varphi \mid \neg\varphi$$

where $\mathfrak{b}$ belongs to $\mathfrak{B}$.

Closed FO formulas (those with no free variable) can be used to define languages. Generally speaking, a pair consisting of a word u and a function ν from the free (non-quantified) variables of a formula φ to the positions of u satisfies φ if u satisfies the closed formula obtained from φ by replacing each free variable with its image by ν.

▶ **Definition 2.** *Let φ, a formula with n free variables, x_1, ..., x_n, and u a word. Let ν be a function of $\{x_1, ..., x_n\}$ in $[\![0, |u| - 1]\!]$. We say that (u, ν) satisfies φ, and we define $u, \nu \models \varphi$ by induction on φ as follows:*

- *$u, \nu \models \top$ and we never have $u, \nu \models \bot$,*
- *$u, \nu \models x < y$ if $\nu(x) < \nu(y)$,*
- *$u, \nu \models x \leq y$ if $\nu(x) \leq \nu(y)$,*
- *$u, \nu \models \mathrm{succ}(x, y)$ if $\nu(y) = \nu(x) + 1$,*
- *$u, \nu \models \mathrm{nsucc}(x, y)$ if $\nu(y) \neq \nu(x) + 1$,*
- *$u, \nu \models a(x)$ if $a \in u[\nu(x)]$ **(note that we only ask inclusion here)**,*
- *$u, \nu \models \varphi \wedge \psi$ if $u, \nu \models \varphi$ and $u, \nu \models \psi$,*
- *$u, \nu \models \varphi \vee \psi$ if $u, \nu \models \varphi$ or $u, \nu \models \psi$,*
- *$u, \nu \models \exists x, \varphi(x, x_1, ..., x_n)$ if there is i of u such that we have $u, \nu \cup [x \mapsto i] \models \varphi$,*
- *$u, \nu \models \forall x, \varphi(x, x_1, ..., x_n)$ if for any index i of u, $u, \nu \cup [x \mapsto i] \models \varphi$,*
- *$u, \nu \models \neg\varphi$ if we do not have $u, \nu \models \varphi$.*

For a closed formula, we simply note $u \models \varphi$.

Here is an example:

▶ **Example 3.** The formula $\varphi = \exists x, \forall y, (x = y \vee \neg a(y))$ describes the set of non-empty words that admit at most one a. For example, $\{a\}\{a, b\}$ does not satisfy φ because two of its letters contain an a, but $\{a, b, c\}\{b\}\emptyset$ does satisfy φ.

▶ Remark 4. The predicates succ and nsucc can be expressed in $\mathrm{FO}^+[\mathfrak{B}_<]$ with three variables. If there are no restriction on variables, in particular if we can use three variables, all binary predicates in $\mathfrak{B}_0$ can be expressed from those in $\mathfrak{B}_<$. Thus, we will consider the whole set of binary predicates available when the number of variables is not constrained, and we will note FO for $\mathrm{FO}[\mathfrak{B}_0]$ or $\mathrm{FO}[\mathfrak{B}_<]$, which are equivalent, and similarly for FO^+.

Let us now turn our attention to FO^+, the set of first-order formulas without negation. We recall definitions from [12].

▶ **Definition 5.** *The grammar of FO^+ is that of FO without the last constructor, $\neg$.*

Let us also define monotonicity properties, starting with an order on words.

▶ **Definition 6.** *A word u is lesser than a word v if u and v are of the same length, and for any index i (common to u and v), the i-th letter of u is included in the i-th letter of v. When a word u is lesser than a word v, we note $u \leq_{A^*} v$.*

▶ **Definition 7.** *Let L be a language. We say that L is monotone when for any word u, v such that u is lesser than v and $u \in L$, we have $v \in L$.*

▶ **Definition 8.** *If L is a language, we define its monotone closure $L^\uparrow := \{v \in A^* \mid \exists u \in L, u \leq_{A^*} v\}$. This is the smallest monotone language containing L.*

▶ **Proposition 9** ([12]). *FO^+ formulas are monotone in unary predicates, i.e. if a model (u, ν) satisfies a formula φ of FO^+, and $u \leq_{A^*} v$, then (v, ν) satisfies φ.*

We will also be interested in other logical formalisms, obtained either by restricting FO, or several variants of temporal logics.

First of all, let us review classical results obtained when considering restrictions on the number of variables. While an FO formula on words is always logically equivalent to a three-variable formula [10], two-variable formulas describe a class of languages strictly included in that described by first-order logic. In addition, the logic FO is equivalent to Linear Temporal Logic (see below).

Please note: these equivalences are only true in the framework of word models. In other circumstances, for example when formulas describe graphs, there are formulas with more than three variables that do not admit equivalents with three variables or fewer.

▶ **Definition 10.** *The set* FO^3 *is the subset of* FO *formulas using only three different variables, which can be reused. We also define* FO^{3+} *for formulas with three variable and without negation. Similarly, we define* FO^2 *and* FO^{2+} *with two variables.*

▶ **Example 11.** The formula $\exists y, \mathrm{succ}(x,y) \wedge (\exists x, (y \leq x) \wedge b(x) \wedge (\forall z, (x \leq z) \vee (z < y) \vee a(z)))$ (a formula with one free variable x that indicates that the letter labeled by x will be followed by a factor of the form $aaaaa...aaab$) is an FO^3 formula, and even an FO^{3+} formula: there is no negation, and it uses only three variables, x, y and z, with a reuse of x. On the other hand, it does not belong to FO^2.

2.2 Temporal logics

Some logics involve an implicit temporal dimension, where positions are identified with time instants. For example, Linear Temporal Logic (LTL) uses operators describing the future, i.e. the indices after the current position in a word. This type of logic can sometimes be more intuitive to manipulate, and present better complexity properties, as explained in the introduction. As mentioned above, FO^2 is not equivalent to FO. On the other hand, it is equivalent to UTL, a restriction of LTL to its unary temporal operators.

To begin with, let us introduce LTL, which is equivalent to FO.

▶ **Definition 12.** *The grammar of* LTL *is as follows:*

$$\varphi, \psi ::= \bot \mid \top \mid a \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \mathrm{X}\varphi \mid \varphi \mathrm{U} \psi \mid \varphi \mathrm{R} \psi \mid \neg\varphi.$$

Removing the last constructor gives the grammar of LTL^+*.*

This logic does not use variables. To check that a word satisfies an LTL formula, we evaluate the formula at the initial instant, that is to say, the word's first position. The X constructor then describes constraints about the next instant, i.e. the following position in the word. So the word $a.u$, where a is a letter, satisfies $\mathrm{X}\varphi$ if and only if the suffix u satisfies φ. The construction $\varphi\mathrm{U}\psi$ (φ Until ψ) indicates that the formula ψ must be verified at a given point in time and that φ must be verified until then. We define $\varphi\mathrm{R}\psi$ (φ Releases ψ) as being equal to $\neg(\neg\varphi\mathrm{U}\neg\psi)$. This is similar to $\psi\mathrm{U}(\varphi \wedge \psi)$, but allowing for the option of φ never being satisfied and ψ being true until the end. The formal definition of the semantics of LTL will be given in a more general setting with additional operators in Definition 16.

▶ Remark 13. Let us call $\varphi\mathrm{XU}\psi$ the formula $\mathrm{X}(\varphi\mathrm{U}\psi)$, for any pair (φ, ψ) of LTL formulas. The advantage of XU is that X and U can be redefined from XU. The notation U for XU is regularly found in the literature.

LTL is included in Temporal Logic, TL. While the former speaks of the future, i.e. of the following indices in the word, thanks to X, U and R, the latter also speaks of the past.

Indeed, we introduce Y, S (Yesterday, Since) and Q the respective past analogues of X, U and R.

▶ **Definition 14.** *The grammar of* TL *is as follows:*

$$\varphi, \psi ::= \mathrm{LTL} \mid \mathrm{Y}\varphi \mid \varphi \mathrm{S} \psi \mid \varphi \mathrm{Q} \psi.$$

Similarly, the grammar of TL^+ *is that of* LTL^+ *extended with* Y, S *and* Q.

▶ Remark 15. As for XU, we will write $\varphi\mathrm{YS}\psi$ for $\mathrm{Y}(\varphi\mathrm{S}\psi)$, and similarly for XR and YQ. We also note $\mathrm{P}\varphi$, $\mathrm{F}\varphi$, $\mathrm{H}\varphi$ and $\mathrm{G}\varphi$ for $\top\mathrm{YS}\varphi$, $\top\mathrm{XU}\varphi$, $\bot\mathrm{YQ}\varphi$ and $\bot\mathrm{XR}\varphi$ respectively. The formulas $\mathrm{F}\varphi$ and $\mathrm{G}\varphi$ mean respectively that the formula φ will be satisfied at least once in the future (F as Future), and that φ will always be satisfied in the future (G as Global). Similarly, the operators P (as Past) and H are the respective past analogues of F and G.

When evaluating an LTL or TL formula on a word $u = u_0 \dots u_m$, we start by default on the first position u_0. However, we need to define more generally the evaluation of a TL formula on a word from any given position:

▶ **Definition 16.** *Let* φ *be a* TL *formula,* $u = u_0 ... u_{m-1}$ *a word, and* $i \in [\![0, m-1]\!]$*. We define* $u, i \models \varphi$ *by induction on* φ*:*

- $u, i \models \top$ *and we never have* $u \models \bot$,
- $u, i \models a$ *if* $a \in u_i$,
- $u, i \models \varphi \wedge \psi$ *if* $u, i \models \varphi$ *and* $u, i \models \psi$,
- $u, i \models \varphi \vee \psi$ *if* $u, i \models \varphi$ *or* $u, i \models \psi$,
- $u, i \models \mathrm{X}\varphi$ *if* $u, i+1 \models \varphi$,
- $u, i \models \varphi\mathrm{U}\psi$ *if there is* $j \in [\![i, m-1]\!]$ *such that* $u, j \models \psi$ *and for all* $k \in [\![i, j-1]\!]$, $u, k \models \varphi$,
- $u, i \models \psi\mathrm{R}\varphi$ *if* $u, i \models \neg(\neg\psi\mathrm{U}\neg\varphi)$,
- $u, i \models \neg\varphi$ *if we do not have* $u, i \models \varphi$,
- $u, i \models \mathrm{Y}\varphi$ *if* $u, i-1 \models \varphi$,
- $u, i \models \varphi\mathrm{S}\psi$ *if there is* $j \in [\![0, i]\!]$ *such that* $u, j \models \psi$ *and for all* $k \in [\![j+1, i]\!]$, $u, k \models \varphi$.
- $u, i \models \psi\mathrm{Q}\varphi$ *if* $u, i \models \neg(\neg\psi\mathrm{S}\neg\varphi)$.

We will write $u \models \varphi$ as a shorthand for $u, 0 \models \varphi$

Finally, let us introduce UTL and UTL^+, the Unary Temporal Logic and its positive version. The UTL logic does not use the U or R operator, but only X, F and G to talk about the future. Similarly, we cannot use S or Q to talk about the past.

▶ **Definition 17.** *The grammar of* UTL *is as follows:*

$$\varphi, \psi ::= \bot \mid \top \mid a \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \mathrm{X}\varphi \mid \mathrm{Y}\varphi \mid \mathrm{P}\varphi \mid \mathrm{F}\varphi \mid \mathrm{H}\varphi \mid \mathrm{G}\varphi \mid \neg\varphi.$$

We define $\mathrm{UTL}[\mathrm{P}, \mathrm{F}, \mathrm{H}, \mathrm{G}]$ *from this grammar by deleting the constructors* X *and* Y.

The grammar of UTL^+ *is obtained by deleting the last constructor, and similarly, we define* $\mathrm{UTL}^+[\mathrm{P}, \mathrm{F}, \mathrm{H}, \mathrm{G}]$ *by deleting the negation in* $\mathrm{UTL}[\mathrm{P}, \mathrm{F}, \mathrm{H}, \mathrm{G}]$.

▶ Remark 18. In the above definition, H and G can be redefined with P and F thanks to negation, but are necessary in the case of UTL^+.

When two formulas φ and ψ are logically equivalent, i.e. admit exactly the same models, we denote it by $\varphi \equiv \psi$. Note that a closed FO formula can be equivalent to an LTL formula, since their models are simply words. Similarly, we can have $\varphi \equiv \psi$ when φ is an FO formula with one free variable (having models of the form (u, i)) and ψ is a LTL or TL formula, this time not using the default first position for TL semantics. We will use both these ways of stating equivalence in the following.

3 Logical equivalences

We want to lift to positive fragments some classical theorems of equivalence between logics, such as these classical results:

▶ **Theorem 19** ([10, 5]).
- FO *and* LTL *define the same class of languages.*
- FO^2 *and* UTL *define the same class of languages.*

3.1 Equivalences to FO^+

We aim at proving the following theorem, lifting classical results from FO to FO^+:

▶ **Theorem 20.** *The logics* FO^+, LTL^+ *and* FO^{3+} *describe the same languages.*

▶ **Lemma 21.** *The set of languages described by* LTL^+ *is included in the set of languages recognised by* FO^{3+}.

The proof is direct, see Appendix A.1 for details. From LTL^+ to FO^+, we can interpret in FO^+ all constructors of LTL^+.

Let us introduce definitions that will be used in the proof of the next lemma.

▶ **Definition 22.** *Let* $\mathrm{qr}(\varphi)$ *be the quantification rank of a formula* φ *of* FO^+ *defined inductively by:*
- *if* φ *contains no quantifier then* $\mathrm{qr}(\varphi) = 0$,
- *if* φ *is of the form* $\exists x, \psi$ *or* $\forall x, \psi$ *then* $\mathrm{qr}(\varphi) = \mathrm{qr}(\psi) + 1$,
- *if* φ *is of the form* $\psi \vee \chi$ *or* $\psi \wedge \chi$ *then* $\mathrm{qr}(\varphi) = \max(\mathrm{qr}(\psi), \mathrm{qr}(\chi))$.

▶ **Definition 23.** *A* separated formula *is a positive Boolean combination of purely past formulas (not using future operators* $\mathrm{X}, \mathrm{U}, \mathrm{R}$*), purely present formulas (not using any temporal operators) and purely future formulas (not using past operators* $\mathrm{Y}, \mathrm{S}, \mathrm{Q}$*).*

We will adapt previous work to show the following auxiliary result:

▶ **Lemma 24.** *Let* φ *be a* TL^+ *formula with possible nesting of past and future operators. There is a separated formula of* TL^+ *that is equivalent to* φ.

Now we are ready to show the main result of this section:

▶ **Lemma 25.** *The set of languages described by* FO^+ *is included in the set of languages recognised by* LTL^+.

Proof. We follow [13, Prop. 1, Appendix B.3], which shows a translation from FO to TL by induction on the quantification rank. We have adapted this to suit our needs.

Let $\varphi(x)$ be an FO^+ formula with a single free variable. Let us show by induction on $\mathrm{qr}(\varphi)$ that φ is equivalent to a formula of TL^+. Notice that we want to show this regardless of the set of unary predicates used, so we might use the induction hypothesis on formulas with an enriched signature, containing additional unary predicates.

Initialisation:

If $\mathrm{qr}(\varphi)$ is zero, then $\varphi(x)$ translates directly into the TL^+ formula. Indeed, disjunctions and conjunctions translate immediately into TL^+. Furthermore, unary predicates of the form $a(x)$ translate into a and binary predicates trivialize into $\top$ and $\bot$ (e.g. $x < x$ translates into $\bot$ and $x = x$ into $\top$). For example, $(x \leq x \wedge a(x)) \vee (b(x) \wedge c(x)) \vee x < x$ translates into $(\top \wedge a) \vee (b \wedge c) \vee \bot$.

Heredity:

Suppose that any FO$^+$ single free variable formula of quantification rank strictly less than $\mathrm{qr}(\varphi)$ translates into a TL$^+$ formula, and $\mathrm{qr}(\varphi)$ is strictly positive.

If φ is a disjunction or conjunction, we need to transform its various clauses. So, without loss of generality, let us assume that $\varphi(x)$ is of the form $\exists y, \psi(x,y)$ or $\forall y, \psi(x,y)$.

Let us denote $a_1, \ldots, a_n$ where n is a natural number, the letters (which are considered as unary predicates) in $\psi(x,y)$ applied to the free variable x, i.e. ignoring those under the scope of a quantification reusing the variable name x.

For any subset S of $[\![1,n]\!]$, we note $\psi^S(x,y)$ the formula $\psi(x,y)$ in which each occurrence of $a_i(x)$ is replaced by $\top$ if i belongs to S and by $\bot$ otherwise, for any integer i of $[\![1,n]\!]$.

We then have the logical equivalence:

$$\psi(x,y) \equiv \bigvee_{S \subseteq [\![1,n]\!]} \left(\bigwedge_{i \in S} a_i(x) \wedge \bigwedge_{i \notin S} \neg a_i(x) \wedge \psi^S(x,y) \right).$$

We are going to show that the negations in the above formula are optional. Let us note:

$$\psi^+(x,y) \equiv \bigvee_{S \subseteq [\![1,n]\!]} \left(\bigwedge_{i \in S} a_i(x) \wedge \psi^S(x,y) \right).$$

Let us then show the equivalence of the formulas $\psi(x,y)$ and $\psi^+(x,y)$ using the monotonicity of ψ as an FO$^+$ formula. First, it is clear that any model satisfying $\psi(x,y)$ satisfies $\psi^+(x,y)$.

Conversely, suppose $\psi^+(x,y)$ is satisfied. We then have a subset S of $[\![1,n]\!]$ such that $(\wedge_{i \in S} a_i(x)) \wedge \psi^S(x,y)$ is satisfied. In particular, according to the values taken by the unary predicates in x, there exists a subset S' of $[\![1,n]\!]$ containing S such that $(\wedge_{i \in S'} a_i(x)) \wedge (\wedge_{i \notin S'} \neg a_i(x)) \wedge \psi^S(x,y)$ is satisfied. Now, ψ is monotone in the different predicates $a_1, \ldots, a_n$. So $(\wedge_{i \in S'} a_i(x)) \wedge (\wedge_{i \notin S'} \neg a_i(x)) \wedge \psi^{S'}(x,y)$ is also satisfied, and $\psi(x,y)$ is therefore satisfied.

The rest of the proof is similar to the proof from [13]: the quantifiers on y commute with the disjunction on S and the conjunction on i of the formula ψ^+. We can therefore fix a subset S of $[\![1,n]\!]$ and simply consider $\exists y, \psi^S(x,y)$ or $\forall y, \psi^S(x,y)$. We then replace $\psi^S(x,y)$ with a formula that depends only on y by replacing each binary predicate of the form $\mathfrak{b}(x,z)$ with a unary predicate $\mathrm{P}_{\mathfrak{b}}(z)$. For example, we can replace $x < z$, $z < x$ or $x = z$ by a unary predicate $\mathrm{P}_{>}(z)$, $\mathrm{P}_{<}(z)$ or $\mathrm{P}_{=}(z)$. We then obtain a formula $\psi'^S(y)$ on an enriched signature with these new unary predicates $\mathrm{P}_{>}$, $\mathrm{P}_{<}$ or $\mathrm{P}_{=}$. We can apply to $\psi'^S(y)$ the induction hypothesis, since there is only one free variable. This yields a formula χ from TL$^+$, equivalent to $\psi'^S(y)$. Notice that words on which χ is evaluated also have to specify the values of the unary predicates $\mathrm{P}_{>}$, $\mathrm{P}_{<}$ or $\mathrm{P}_{=}$. If Σ is the original set of unary predicates, let $\Sigma' = \Sigma \cup \{\mathrm{P}_{>}, \mathrm{P}_{<}, \mathrm{P}_{=}\}$. For any $u \in \mathcal{P}(\Sigma)^*$, and valuation ν giving the value of the free variable x in u, there is a unique $f_\nu(u) \in \mathcal{P}(\Sigma')^*$ such that u is the projection of $f_\nu(u)$ on Σ, and the values of the new predicates in $f_\nu(u)$ reflect their intended semantic with respect to x, e.g. $P_{=}(z)$ is true if and only if $x = z$. We then have for all such u, ν:

$$u, \nu \models \exists y, \psi^S(x,y) \iff f_\nu(u), \nu(x) \models \mathrm{P}\chi \vee \chi \vee \mathrm{F}\chi,$$
and
$$u, \nu \models \forall y, \psi^S(x,y) \iff f_\nu(u), \nu(x) \models \mathrm{H}\chi \wedge \chi \wedge \mathrm{G}\chi.$$

Let χ' be the formula obtained: either $\mathrm{P}\chi \vee \chi \vee \mathrm{F}\chi$ or $\mathrm{H}\chi \wedge \chi \wedge \mathrm{G}\chi$. This formula χ' involves unary predicates of the form $\mathrm{P}_{\mathfrak{b}}$. We then use Lemma 24 to transform χ' into a positive Boolean combination of purely past, present and future positive formulas, where predicates $\mathrm{P}_{\mathfrak{b}}$ trivialize into $\top$ or $\bot$, on words of the form $f_\nu(u)$. For example, $\mathrm{P}_<$ trivializes into $\top$ in purely past formulas, into $\bot$ in purely present or future formulas.

This completes the induction. From a formula in FO^+, we can construct an equivalent formula in TL^+. Ultimately, we can return to a future formula. We want to evaluate in $x = 0$, so the purely past formulas, isolated by the separation lemma (Lemma 24), trivialize into $\bot$ or $\top$.

Now, to translate a closed formula φ from FO^+ to LTL^+, we can add a free variable by setting $\varphi'(x) = \varphi \wedge (x = 0)$. Then, by the above, φ' translates into a formula χ from LTL^+, logically equivalent to φ.

◀

Putting together Lemma 21 and Lemma 25 achieves the proof of Theorem 20.

3.2 Equivalences in fragments of FO^+

▶ **Theorem 26.** *The languages described by* $\mathrm{FO}^{2+}[\mathfrak{B}_0]$ *formulas with one free variable are exactly those described by* UTL^+ *formulas.*

Proof. First, let us show the UTL^+ to FO^{2+} direction. In the proof of Lemma 21, as is classical, three variables are introduced only when translating U. By the same reasoning as for X, it is clear that translating Y introduces two variables. It remains to complete the induction of Lemma 21 with the cases of P, F, H and G, but again we can restrict ourselves to future operators by symmetry:

- $[\mathrm{F}\varphi](x) = \exists y, x < y \wedge [\varphi](y)$;
- $[\mathrm{G}\varphi](x) = \forall y, y \leq x \vee [\varphi](y)$.

For the converse direction from FO^{2+} to UTL^+, we draw inspiration from [5, Theorem 1]. This proof is similar to that of [13] used previously in the proof of Lemma 25: we perform a disjunction on the different valuations of unary predicates in one free variable to build a formula with one free variable. However, the proof of Lemma 25 cannot be adapted as is, since it uses the separation theorem which does not preserve the membership of a formula to UTL, see [6, Lem 9.2.2]. However, the article [5] uses negations, and we must therefore construct our own induction case for the universal quantifier that is treated in [5] via negations.

The beginning of the proof is identical to that of Lemma 25, starting with a formula $\exists y, \psi(x, y)$ or $\forall y, \psi(x, y)$. Using the same notations, we can reduce to the case of a formula $\exists y, \psi^S(x, y)$ or $\forall y, \psi^S(x, y)$ with no unary predicate applied to x in $\psi^S(x, y)$, since all those predicates have been replaced with $\top$ or $\bot$ according to S. Contrarily to the previous proof, we cannot directly replace binary predicates with unary predicates, because this relied on the separation theorem.

Let us consider, as in [5], the *position formulas*, $y < x \wedge \mathrm{nsucc}(y, x)$, $\mathrm{succ}(y, x)$, $y = x$, $\mathrm{succ}(x, y)$ and $x < y \wedge \mathrm{nsucc}(x, y)$, whose set is denoted T.

We then have the logical equivalence:

$$\psi^S(x, y) \equiv \bigvee_{\tau \in \mathrm{T}} \tau(x, y) \wedge \psi^S_\tau(y) \equiv \bigwedge_{\tau \in \mathrm{T}} \tau(x, y) \implies \psi^S_\tau(y)$$

where $\psi^S_\tau(y)$ is obtained from the formula $\psi^S(x, y)$ assuming the relative positions of x and y are described by τ. The above equivalence holds because T forms a partition of the possibilities for the relative positions of x and y: exactly one of the five formulas $\tau(x, y)$

from T must hold. Since x and y are the only two variables, any binary predicate involving x is a binary predicate involving x and y (or else it involves only x and is trivial). Binary predicates are therefore trivialized according to the position described by τ.

▶ **Example 27.** For $\psi^S(x, y) = \mathrm{nsucc}(x, y) \land a(y) \land (\forall x, x \leq y \lor b(y))$ and for the position formula $\tau = y < x \land \mathrm{nsucc}(y, x)$, we have $\psi^S_\tau(y) = \top \land a(y) \land (\forall x, x \leq y \lor b(y))$. We do not replace the bound variable x. We have obtained a formula with one free variable, so we can indeed use the induction hypothesis.

If we started with the existential form $\exists y, \psi^S(x, y)$, we will use the disjunctive form $\psi^S(x, y) \equiv \bigvee_{\tau \in T} \tau(x, y) \land \psi^S_\tau(x, y)$. This allows us to have the quantifier commute with the disjunction, and obtain $\bigvee_{\tau \in T} \exists y, \tau(x, y) \land \psi^S_\tau(x, y)$. Similarly, if starting with the universal form $\forall y, \psi^S(x, y)$, we will use the conjunctive form $\psi^S(x, y) \equiv \bigwedge_{\tau \in T} \tau(x, y) \implies \psi^S_\tau(x, y)$.

We then need to translate $\exists y, \tau(x, y) \land \psi^S_\tau(y)$ and $\forall y, \tau(x, y) \implies \psi^S_\tau(y)$, which we note respectively $[\tau]_\exists$ and $[\tau]_\forall$, in UTL^+, for any position formula τ. We assume ψ^S_τ fixed in this context so it is omitted from this notation. However, it is important to note that $[\tau]_\exists$ and $[\tau]_\forall$ will still depend on ψ^S_τ. In each case, we note χ for the UTL^+ formula obtained by induction from $\psi^S_\tau(y)$:

$$[y < x \land \mathrm{nsucc}(y, x)]_\exists \equiv \mathrm{YP}\chi$$

$$[y < x \land \mathrm{nsucc}(y, x)]_\forall \equiv \mathrm{YH}\chi$$

$$[\mathrm{succ}(y, x)]_\exists \equiv [\mathrm{succ}(y, x)]_\forall \equiv \mathrm{Y}\chi$$

$$[y = x]_\exists \equiv [y = x]_\forall \equiv \chi$$

$$[\mathrm{succ}(x, y)]_\exists \equiv [\mathrm{succ}(x, y)]_\forall \equiv \mathrm{X}\chi$$

$$[x < y \land \mathrm{nsucc}(x, y)]_\exists \equiv \mathrm{XF}\chi$$

$$[x < y \land \mathrm{nsucc}(x, y)]_\forall \equiv \mathrm{XG}\chi.$$

This achieves the proof of Theorem 26. ◀

▶ **Corollary 28.** *The logic* $\mathrm{FO}^{2+}[\mathfrak{B}_<]$ *is equivalent to* $\mathrm{UTL}^+[\mathrm{P}, \mathrm{F}, \mathrm{H}, \mathrm{G}]$.

Proof. For the right-to-left direction, it suffices to notice that the predicates used to translate the constructors of $\mathrm{UTL}^+[\mathrm{P}, \mathrm{F}, \mathrm{H}, \mathrm{G}]$ in the previous proof belong to $\mathfrak{B}_<$.

For the left-to-right direction, simply replace the set T in Theorem 26 proof by $\mathrm{T}' = \{y < x, y = x, x < y\}$. Once again, we obtain an exhaustive system of mutually exclusive position formulas that allow us to trivialize binary predicates. The proof of Theorem 26 can thus be lifted immediately to this case. ◀

We showed that several classical logical equivalence results can be transposed to their positive variants.

4 Characterisation of monotonicity

So far, we have focused on languages described by positive formulas, from which monotonicity follows. Here, we focus on the monotonicity property and propose a characterisation. We then derive a monoid-based algorithm that decides, given a regular language L, whether it is monotone, refining results from [12] focusing on automata-based algorithms.

4.1 Characterisation by monoids

We assume the reader familiar with monoids (see Appendix B.1 for detailed definitions).

We will note $(\mathbf{M}, \cdot)$ a monoid and $\mathbf{M}_L$ the syntactic monoid of a regular language L and $\leq_L$ the syntactic order.

▶ **Theorem 29.** *Let $L \subseteq A^*$ be a regular language, and $\leq_L$ be its syntactic order. The language L is monotone if and only if we have:*

$$\forall (s, s') \in A^2, s \subseteq s' \implies h(s) \leq_L h(s')$$

where $h : A^* \to \mathbf{M}_L$ denotes the syntactic morphism onto the syntactic monoid.

Proof. For the left-to-right direction let L be a monotone language and $s \subseteq s'$. Let m and n be two elements of $\mathbf{M}_L$ such that $mh(s)n \in h(L)$. Since $h : A^* \to \mathbf{M}_L$ is surjective, let $u \in h^{-1}(m)$ and $v \in h^{-1}(n)$. Then $usv \in L$ since h recognises L. So $us'v \in L$ by monotonicity of L. Thus $mh(s')n \in h(L)$. We can conlude that $h(s) \leq_L h(s')$.

For the converse direction, suppose that $\leq_L$ verifies the condition of Theorem 29. We can remark that $\leq_L$ is compatible with the product of the monoid. Let $u \in L$ and v be two words such that $u \leq_{A^*} v$. Then $h(u) \leq_L h(v)$. By definition of the syntactic order, and since $h(u)$ is in the accepting set of the monoid, we must have $h(v)$ accepted as well, hence $v \in L$. Thus L is monotone.

◀

4.2 An algorithm to decide monotonicity

We immediately deduce from Theorem 29 an algorithm for deciding the monotonicity of a regular language L from its syntactic monoid. Indeed, it is sufficient to check for any pair of letters (s, s') such that s is included in s' whether $m \cdot h(s) \cdot n \in h(L)$ implies $m \cdot h(s') \cdot n \in h(L)$ for any pair (m, n) of elements of the syntactic monoid, where h denotes the syntactic morphism onto the syntactic monoid.

This algorithm works for any monoid that recognises L through a surjective $h : A^* \to M$, not just its syntactic monoid. Indeed, for any monoid, we start by restricting it to $h(A^*)$ to guarantee that h is surjective. Then, checking the above implication is equivalent to checking whether $s \leq_L s'$ for all letters s and s' such that s is included in s'.

This is summarised in the following proposition:

▶ **Proposition 30.** *There is an algorithm which takes as input a monoid $(\mathbf{M}, \cdot)$ recognising a regular language L through a morphism h and decides whether L is monotone in $O(|A|^2|\mathbf{M}|^2)$.*

It was shown in [12, Thm 2.5] that deciding monotonicity is PSPACE-complete if the language is given by an NFA, and in P if it is given by a DFA.

Recall that in general, the syntactic monoid of a language L may be exponentially larger than the minimal DFA of L.

We give a more precise result for DFA, and give also a refined complexity result for monoid input.

▶ **Proposition 31.** *Deciding whether a regular language is monotone is in LOGSPACE when the input is a monoid while it is* NL − complete *when it is given by a DFA.*

5 Semantic and syntactic monotonicity

The paper [12, Definition 4.2] exhibits a monotone language definable in FO but not in FO^+. The question then arises as to how simple such a counter-example can be. For instance, can it be taken in specific fragments of FO, such as FO^2? This section presents a few lemmas that might shed some light on the subject, followed by some conjectures, and a new counterexample to the equivalence between syntactic and semantic monotonicity in the alternation-free fragment of FO^2.

5.1 Refinement of the counter-example in the general case

In [12, Sec. 4], the counter-example language that is monotone and FO-definable but not FO^+-definable uses three predicates a, b and c and is as follows:

$$K = ((abc)^*)^\uparrow \cup A^* \top A^*$$

where $^\uparrow$ is the monotone closure from Definition 8, and $\top = \{a, b, c\}$ is the letter of A containing all three predicates. Any word containing $\top$ is therefore in K, which intuitively corresponds to ignoring any phenomenon involving this letter, while ensuring monotonicity.

It uses the following words to find a strategy for Duplicator in EF_k^+:

$$u_0 = (abc)^n \text{ and } u_1 = \left(\binom{a}{b}\binom{b}{c}\binom{c}{a}\right)^n \binom{a}{b}\binom{b}{c}$$

where n is greater than 2^k, and $\binom{s}{t}$ is just a compact notation for the letter $\{s, t\}$ for any predicates s and t.

This in turns allows to show the failure on Lyndon's preservation theorem on finite structures [12, Sec. 5]. Our goal in this section is to refine this counter-example to more constrained settings. We hope that by trying to explore the limits of this behaviour, we achieve a better understanding of the discrepancy between monotone and positive.

In Section 5.1.1, we give a smaller fragment of FO where the counter-example can still be encoded. In Section 5.1.2, we show that the counter-example can still be expressed with a single unary predicate. This means that it could occur for instance in LTL^+ where the specification only talks about one sensor being activated or not.

5.1.1 Using the between predicate

Here we investigate the robustness of the counter-example by restricting the fragment of FO^+ in which it can be expressed.

First, let us define the "between" binary predicate introduced in [11].

▶ **Definition 32.** *[11] For any unary predicate a (not only predicates from Σ but also Boolean combination of them), a also designates a binary predicate, called between predicate, such that for any word u and any valuation ν, $(u, \nu) \models a(x, y)$ if and only if there exists an index i between $\nu(x)$ and $\nu(y)$ excluded such that $(u, [z \mapsto i]) \models a(z)$.*

We denote $\mathfrak{be}$ the set of between predicates and $\mathfrak{be}^+$ the set of positive between predicates, i.e. positive boolean combination of predicates of the form $a^\uparrow(x, y)$ with $a \in \Sigma$.

It is shown in [11] that $\mathrm{FO}^2[\mathfrak{B}_0 \cup \mathfrak{be}]$ is strictly less expressive than FO.

▶ **Proposition 33.** *There exists a monotone language definable in $\mathrm{FO}^2[\mathfrak{B}_0 \cup \mathfrak{be}]$ which is not definable in $\mathrm{FO}^{2+}[\mathfrak{B}_0 \cup \mathfrak{be}^+]$.*

Proof. We will use the following language:

$$K' := K \cup A^* \left(\binom{a}{b}^2 \cup \binom{b}{c}^2 \cup \binom{c}{a}^2 \cup \binom{a}{b}\binom{c}{a} \cup \binom{b}{c}\binom{a}{b} \cup \binom{c}{a}\binom{b}{c} \right) A^*.$$

Indeed, in [12, Sec. 4], it is explained that to recognise K in FO, we need to look for some "anchor positions". Such positions resolve the possible ambiguity introduced by double letters of the form $\binom{a}{b}$, that could play two different roles for witnessing membership in $((abc)^*)^{\uparrow}$. Indeed, if $\binom{a}{b}$ appears in a word, we cannot tell whether it stands for an a or a b. In contrast, anchor letters have only one possible interpretation. They may be singletons ($\{a\}$, $\{b\}$, $\{c\}$) or consecutive double letters such as $\binom{a}{b}\binom{c}{a}$ which can only be interpreted one way, here as bc. For our language K', we accept any word containing an anchor of the second kind. This means that in remaining words we will only be interested in singleton anchors. Thus, we need two variables only to locate consecutive anchors and between predicates to check if the letters between the anchors are double letters. See Appendix C.1 for a more detailed description of a formula.

In order to show that K' is not definable in $\mathrm{FO}^{2+}[\mathfrak{B}_0 \cup \mathfrak{bc}^+]$, it suffices to remark that the argument showing that K is not definable in FO^+ suffices, because the words u_0 and u_1 used in the EF^+-game are still separated by K'. This shows that K' is not even definable in FO^+ so a fortiori it cannot be definable in $\mathrm{FO}^{2+}[\mathfrak{B}_0 \cup \mathfrak{bc}^+]$.

◀

5.1.2 Only one unary predicate

Now, let us show another refinement. We can lift K to a counter-example where the set of predicates Σ is reduced to a singleton.

▶ **Proposition 34.** *As soon as there is at least one unary predicate, there exists a monotone language definable in* FO *but not in* FO^+.

Proof (Sketch). Suppose Σ reduced to a singleton. Then, A is reduced to two letters which we note 0 and 1 with 1 greater than 0. We will again rely on the proof from [12, Sec. 4]. We will encode each predicate from $\{a, b, c\}$ and a new letter # (the separator) into A^* as follows:

$$\begin{cases} [a] = 100 \\ [b] = 010 \\ [c] = 001 \\ [\#] = 100001 \end{cases}$$

We will encode the language K as follows:

$$[K] = (([a][\#][b][\#][c][\#])^*)^{\uparrow} \cup A^*1(A^4\backslash 0^4)1A^* \cup A^*1^5A^*.$$

It remains to verify that K is monotone and FO-definable, and that it is not FO^+-definable. The choice of encoding guarantees that this is indeed the case and that we can rely on properties of the original language K, without any ambiguities introduced by the encoding. See Appendix C.2 for details. ◀

5.2 Stability through monotone closure

It has been shown by Thérien and Wilke [20] that $\mathrm{FO}^2[\mathfrak{B}_<]$-definable languages are exactly those that are both Σ_2-definable and Π_2-definable where Σ_2 is the set of FO-formulas of the form $\exists x_1, ..., x_n \forall y_1, ..., y_m \varphi(x_1, ..., x_n, y_1, ...y_m)$ where φ does not have any quantifier and Π_2-formulas are negations of Σ_2-formulas. Hence, $\Sigma_2 \cup \Pi_2$ is the set of FO-formulas in prenex normal form with at most one quantifier alternation. Moreover, Pin and Weil [17] showed that Σ_2 describes the unions of languages of the form $A_0^*.s_0.A_1^*.s_1.....s_t.A_{t+1}^*$, where t is a natural integer, s_i are letters from A and A_i are subalphabets of A.

In the following, we will use $\mathfrak{B}_<$ as default set of binary predicates for FO^2.

Even though we do not know yet whether FO^{2+} captures the set of monotone FO^2-definable languages, we can state the following theorem:

▶ **Theorem 35.** *The set $\Sigma_2^+ \cap \Pi_2^+$ of languages definable by both positive Σ_2-formulas (written Σ_2^+) and positive Π_2-formulas (written Π_2^+) is equal to the set of monotone FO^2-definable languages.*

Proof (Sketch). We start by showing that Σ_2^+ captures the set of monotone Σ_2-definable languages. This can be done thanks to a syntactic characterisation of Σ_2-definable languages given by Pin and Weil [17]: a language is Σ_2-definable if and only if it is a union of languages of the form $A_0^*.s_0.A_1^*.s_1.....s_t.A_{t+1}^*$. Next, we introduce a dual closure operator $L^{\curlywedge} = ((L^c)^{\downarrow})^c$ that allows a finer manipulation of monotone languages. This allows us to lift the previous result to Π_2 languages. Taken together, these results show that the set of languages definable by both positive Σ_2-formulas and positive Π_2-formulas is exactly the set of monotone FO^2-definable languages. See Appendix D for details. ◀

This last result shows how close to capturing monotone FO^2-definable languages FO^{2+} is. However, it does not seem easy to lift the equivalence $\Sigma_2 \cap \Pi_2 = \mathrm{FO}^2$ to their positive fragments as we did for the other classical equivalences in Section 3. Indeed, the proof from [20] relies itself on the proof of [17] which is mostly semantic while we are dealing with syntactic equivalences.

Since languages in $\Sigma_2^+ \cap \Pi_2^+$ are in particular in FO^+, Theorem 35 implies that a counter-example separating FO-monotone from FO^+ cannot be in $\mathrm{FO}^2[\mathfrak{B}_<]$ as stated in the following corollary:

▶ **Corollary 36.** *Any monotone language described by an $\mathrm{FO}^2[\mathfrak{B}_<]$ formula is also described by an FO^+ formula.*

If the monotone closure $L^\uparrow$ of a language L described by a formula of $\mathrm{FO}^2[\mathfrak{B}_<]$ is in FO^+, nothing says on the other hand that $L^\uparrow$ is described by a formula of $\mathrm{FO}^2[\mathfrak{B}_<]$, or even of $\mathrm{FO}^2[\mathfrak{B}_0]$ as the counterexample $L = a^*bc^*ba^*$ shows. The monotone closure $L^\uparrow$ cannot be defined by an $\mathrm{FO}^2[\mathfrak{B}_0]$ formula. This can be checked using for instance Charles Paperman's software Semigroup Online [16]. Notice that the software uses the following standard denominations: **DA** corresponds to $\mathrm{FO}^2[\mathfrak{B}_<]$, and **LDA** to $\mathrm{FO}^2[\mathfrak{B}_0]$.

We give the following conjecture, where FO^2 can stand either for $\mathrm{FO}^2[\mathfrak{B}_<]$ or for $\mathrm{FO}^2[\mathfrak{B}_0]$

▶ **Conjecture 37.**

- *A monotone language is definable in FO^2 if and only if it is definable in FO^{2+}.*
- *It is decidable whether a given regular language is definable in FO^{2+}*

Since we can decide whether a language is definable in FO^2 and whether it is monotone, the first item implies the second one.

5.3 The alternation-free fragment of FO^{2+}

We consider in this section the alternation-free fragment of FO^2, and its FO^{2+} counterpart, as a first step towards understanding the full logic FO^{2+} (recall that we are using $\mathfrak{B}_<$ as the default set of binary predicates here). The alternation hierarchy of FO^2 has been studied extensively in recent years (for example in [22]), and might be an angle through which tackling Conjecture 37. The first level of this hierarchy (the alternation-free fragment) enjoys several characterisations. In particular, it corresponds to the variety of languages recognised by J-trivial monoids, which entails in particular the decidability of membership for this fragment. We exhibit a counter-example to an analog of Conjecture 37 for this fragment, namely a language that is monotone and definable in FO^2 without alternation, but not definable in FO^{2+} without alternation. This constitutes a new counter-example separating monotonicity from positivity, and the first one that is not relying on the language K from [12] but exploits instead a different phenomenon.

We will work over the set of unary predicates $\Sigma = \{1\}$, and $A = \mathcal{P}(\Sigma) = \{\emptyset, \{1\}\}$. To simplify notations, we will think of A as the set of letters $\{0, 1\}$ — 0 representing the empty set and 1 the singleton $\{1\}$. The language L is defined as the monotone closure of 1^+01^+, that is $L = 1^+01^+ + 1^+11^+$. The language L is monotone by definition. It is definable in FO^2 without alternation: FO^2 allows to describe the property of having a given subword appearing, or not appearing, in a word, and it is straightforward to give a description of L in terms of this kind of property. See Appendix E.1.

We now show that this language is indeed a counterexample to Conjecture 37 for the logic FO^2 without alternation:

▶ **Proposition 38.** *The language L is not definable in FO^{2+} without alternation.*

Proof (Sketch). We make use of the Ehrenfeucht-Fraïssé game for FO^{2+} without alternation (see Appendix E). Fix a natural number n, and consider the words $u = 1^{2n}01 \in L$ and $v = 1^{2n}0 \notin L$. We show that Duplicator wins $\mathrm{EF}^{2+}_{n,alt-free}(u, v)$, which entails that there is no FO^{2+} formula without alternation that can define L. The idea is that Spoiler has to play in u because v is a subword of u (so Duplicator can copy the moves of Spoiler); but when Spoiler plays in u, Duplicator can try to mimic Spoiler's moves in the block 1^{2n}, and Spoiler can only win by showing that the two words have different sizes, which cannot be done in less that $n + 1$ turns. The details of the proof can be found in Appendix E.2. ◀

Note that this counterexample contradicts the first item of an analog of Conjecture 37 for the alternation-free fragment of FO^2, but whether the alternation-free fragment of FO^{2+} has decidable membership remains open.

Conclusion

We have investigated how the tension between semantic and syntactic positivity behaves in FO-definable languages. We paid a close look to how this plays out in relation to monoids, complexity of procedures, LTL, and fragments of FO. We show that despite earlier negative results from [12] such as indecidabiilty of membership in FO^+, this logic retains some robustness, as equivalence with LTL^+ can be obtained, even at the level of specific fragments $\mathrm{FO}^{2+}[\mathfrak{B}_<]$ and $\mathrm{FO}^{2+}[\mathfrak{B}_0]$. We show that the counter-example language K from [12] can be lifted to show that some positive fragments of FO differ from their monotone counterpart, while for some other fragment, namely alternation-free FO^2, we provide a new counter-example relying on a different mechanism. We leave the question open for the FO^2 fragment, that we consider to be an interesting challenge.

References

1 Miklos Ajtai and Yuri Gurevich. Monotone versus positive. *J. ACM*, 34(4):1004–1015, October 1987.

2 David Baelde and Dale Miller. Least and greatest fixed points in linear logic. In Nachum Dershowitz and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 92–106, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

3 Julian Bradfield and Igor Walukiewicz. The mu-calculus and model checking. In *Handbook of Model Checking*, pages 871–919. Springer, 2018.

4 Thomas Colcombet. Regular Cost Functions, Part I: Logic and Algebra over Words. *Logical Methods in Computer Science*, Volume 9, Issue 3, August 2013. URL: https://lmcs.episciences.org/1221, doi:10.2168/LMCS-9(3:3)2013.

5 Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *BRICS Report Series*, 4(5), Jan. 1997. URL: https://tidsskrift.dk/brics/article/view/18784, doi:10.7146/brics.v4i5.18784.

6 Dov M. Gabbay, Ian Hodkinson, and Mark Reynolds. *Temporal logic (vol. 1): mathematical foundations and computational aspects.* Oxford University Press, Inc., USA, 1994.

7 Michelangelo Grigni and Michael Sipser. Monotone complexity. In *Poceedings of the London Mathematical Society Symposium on Boolean Function Complexity*, page 57–75, USA, 1992. Cambridge University Press.

8 Ian M. Hodkinson and Mark Reynolds. Separation - past, present, and future. In Sergei N. Artëmov, Howard Barringer, Artur S. d'Avila Garcez, Luís C. Lamb, and John Woods, editors, *We Will Show Them! Essays in Honour of Dov Gabbay, Volume Two*, pages 117–142. College Publications, 2005.

9 Neil Immerman. Relational queries computable in polynomial time. *Information and Control*, 68(1):86–104, 1986. doi:https://doi.org/10.1016/S0019-9958(86)80029-8.

10 Hans Kamp. *Tense Logic and the Theory of Linear Order.* PhD thesis, University of California Los Angeles, 1968. Published as Johan Anthony Willem Kamp. URL: http://www.ims.uni-stuttgart.de/archiv/kamp/files/1968.kamp.thesis.pdf.

11 Andreas Krebs, Kamal Lodaya, Paritosh K. Pandya, and Howard Straubing. Two-variable logics with some betweenness relations: Expressiveness, satisfiability and membership. *Log. Methods Comput. Sci.*, 16(3), 2020. URL: https://lmcs.episciences.org/6765.

12 Denis Kuperberg. Positive first-order logic on words and graphs. *Log. Methods Comput. Sci.*, 19(3), 2023. URL: https://doi.org/10.46298/lmcs-19(3:7)2023.

13 Denis Kuperberg and Michael Vanden Boom. On the expressive power of cost logics over infinite words. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming*, pages 287–298, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

14 Roger C. Lyndon. Properties preserved under homomorphism. *Pacific J. Math.*, 9(1):143–154, 1959. URL: https://projecteuclid.org:443/euclid.pjm/1103039459.

15 Daniel Oliveira and João Rasga. Revisiting separation: Algorithms and complexity. *Log. J. IGPL*, 29(3):251–302, 2021. URL: https://doi.org/10.1093/jigpal/jzz081, doi:10.1093/JIGPAL/JZZ081.

16 Charles Paperman. Semigroup online. URL: https://paperman.name/semigroup/.

17 Jean-Éric Pin and Pascal Weil. Polynomial closure and unambiguous product. *Theory of Computing Systems*, 30:383–422, 1995. URL: https://api.semanticscholar.org/CorpusID:850708.

18 Benjamin Rossman. Homomorphism preservation theorems. *J. ACM*, 55, 07 2008.

19 Alexei P. Stolboushkin. Finitely monotone properties. In *LICS, San Diego, California, USA, June 26-29, 1995*, pages 324–330. IEEE Computer Society, 1995.

20 Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*,

STOC '98, page 234–240, New York, NY, USA, 1998. Association for Computing Machinery. doi:10.1145/276698.276749.

21 Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, page 137–146, New York, NY, USA, 1982. Association for Computing Machinery. doi:10.1145/800070.802186.

22 Philipp Weis and Neil Immerman. Structure Theorem and Strict Alternation Hierarchy for FO2 on Words. In Thomas Schwentick, Denis Thérien, and Heribert Vollmer, editors, *Circuits, Logic, and Games*, volume 6451 of *Dagstuhl Seminar Proceedings (DagSemProc)*, pages 1–22, Dagstuhl, Germany, 2007. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/DagSemProc.06451.6, doi:10.4230/DagSemProc.06451.6.

A Positive Linear Temporal Logic

A.1 Proof of Lemma 21

Proof. Let us show the lemma by induction on the LTL^+ formula. We inductively construct for any formula φ of LTL^+, a formula $\varphi^\star(x)$ of FO^{3+} with one free variable that describes the same language. This just amounts to remove the negation case in the classical proof, no additional difficulty here.

- $\bot^\star = \bot$,
- $\top^\star = \top$,
- $a^\star = a(x)$,
- $(\varphi \wedge \psi)^\star(x) = \varphi^\star(x) \wedge \psi^\star(x)$,
- $(\varphi \vee \psi)^\star(x) = \varphi^\star(x) \vee \psi^\star(x)$,
- $(\mathrm{X}\varphi)^\star(x) = \exists y, \mathrm{succ}(x, y) \wedge \varphi^\star(y)$,
- $(\varphi \mathrm{U} \psi)^\star(x) = \exists y, x \leq y \wedge \psi^\star(y) \wedge \forall z, (z < x \vee y \leq z \vee \varphi^\star(z))$,
- $(\psi \mathrm{R} \varphi)^\star(x) = (\varphi \mathrm{U}(\psi \wedge \varphi))^\star(x) \vee (\forall y, y < x \vee \varphi^\star(y))$.

The translation of a formula φ of LTL^+ into a closed formula of FO^{3+} is therefore $\exists x, x = 0 \wedge \varphi^\star(x)$, where $x = 0$ is short for $\forall y, y \geq x$.

This construction makes it possible to reuse the variables introduced. This is why we can translate the formulas of LTL^+ into FO^{3+}. ◀

A.2 Proof of Lemma 24

Proof (Sketch). This follows by simply combining observations from previous proofs of variants of the separation theorem. Our starting point is the proof given by Kuperberg and Vanden Boom in [13, Lemma 5, Appendix B.2], which proves the equivalence between generalisations of the logics FO and LTL, to the so-called cost FO and cost LTL. When specialised to FO and LTL, this corresponds to the case where negations appear only at the leaves of formulas. This brings us closer to our goal.

Let us emphasize that [13] proves a generalised version of the separation theorem from [8]. In [8], it is proven that any formula of TL is equivalent to a separated formula, while a particular attention to positivity is additionally given in [13]. Indeed, [13] also shows that such a Boolean combination can be constructed while preserving the formula's positivity. One can also check [15] to verify that positivity of a formula is kept when separating the formula. Thus, a formula in TL^+ can be written as a positive Boolean combination of purely past, present and future formulas themselves in TL^+. ◀

B Monoids

B.1 Algebraic definitions

▶ **Definition 39.** *A monoid is a pair* $(\mathbf{M}, \cdot)$ *where* $\cdot$ *is an associative internal composition law on the non-empty set* $\mathbf{M}$*, which has a neutral element noted* $1_{\mathbf{M}}$ *(or simply* 1 *when there is no ambiguity), i.e. which verifies:*

$$\forall m \in \mathbf{M}, 1 \cdot m = m \cdot 1 = m.$$

We allow ourselves the abuse of language which consists in speaking of the monoid $\mathbf{M}$ instead of the monoid $(\mathbf{M}, \cdot)$.

▶ **Definition 40.** *Let* $(\mathbf{M}, \cdot)$ *and* $(\mathbf{M}', \circ)$ *be two monoids. An application* h *defined from* $\mathbf{M}$ *into* $\mathbf{M}'$ *is a morphism of monoids if:*

$$\forall (m_1, m_2) \in \mathbf{M}^2, h(m_1 \cdot m_2) = h(m_1) \circ h(m_2)$$

and

$$h(1_{\mathbf{M}}) = 1_{\mathbf{M}'}.$$

▶ **Definition 41.** *Let* $(\mathbf{M}, \cdot)$ *be a monoid, and* $\leq$ *an order on* $\mathbf{M}$*. We say that* $\leq$ *is compatible with* $\cdot$ *if:*

$$\forall (m, m', n, n') \in \mathbf{M}^4, m \leq n \wedge m' \leq n' \implies m \cdot m' \leq n \cdot n'.$$

▶ **Definition 42.** *Let* L *be a language and* $(\mathbf{M}, \cdot)$ *a finite monoid. We say that* $\mathbf{M}$ *recognises* L *if there exists a monoid morphism* h *from* $(A^*, .)$ *into* $(\mathbf{M}, \cdot)$ *such that* $L = h^{-1}(h(L))$.

▶ **Definition 43.** *Let* L *be a regular language, and* $u, v \in A^*$ *be any two words. We define the equivalence relation of indistinguishability denoted* $\sim_L$ *on* A^**. We write* $u \sim_L v$ *if:*

$$\forall (x, y) \in A^* \times A^*, xuy \in L \iff xvy \in L.$$

Similarly, we write $u \leq_L v$ *if:*

$$\forall (x, y) \in A^* \times A^*, xuy \in L \implies xvy \in L.$$

The $\leq_L$ *preorder is called the* L *syntactic preorder.*

▶ **Definition 44.** *Let* L *be a regular language. We define the syntactic monoid of* L *as* $\mathbf{M}_L = L/\sim_L$.

▶ Remark 45. This is effectively a monoid, since $\sim_L$ is compatible with left and right concatenation. Moreover, the syntactic monoid recognises L through syntactic morphism. Moreover, we can see that the order $\leq_L$ naturally extends to an order compatible with the product on the syntactic monoid. We will use the same notation to designate both the pre-order $\leq_L$ and the order induced by $\leq_L$ on $\mathbf{M}_L$, which we will call syntactic order.

B.2 Proof of Proposition 31

Proof. First, in the algorithm from Proposition 30, at any given time, we only need to code two letters from $A = \mathcal{P}(\Sigma)$ and two elements from the monoid $\mathbf{M}$. So we can code s and s' with $|\Sigma|$ bits and increment them through the loop in order to go through the whole alphabet.

For example, if $\Sigma = \{a, b, c\}$ then a is coded by 001, $\{a, b\}$ by 010 and so on. In the same way, we only need $2\lceil \log_2(\mathbf{M}) \rceil$ bits to code (m, n). Using lookup tables for applying the function h, the product $\cdot$, and testing membership in $h(L)$, all operations can be done in LogSpace. Thus, the algorithm from Proposition 30 is in LogSpace.

To decide whether a DFA $\mathcal{B}$ describes a monotone language, we can compute the NFA $\mathcal{B}^\uparrow$ by adding to each transition (q_0, a, q_1) of $\mathcal{B}$ any transition (q_0, b, q_1) with b greater than a. Thus, $\mathcal{B}^\uparrow$ describes the monotone closure of the language recognised by $\mathcal{B}$. Then, $\mathcal{B}$ recognises a monotone language if and only if there is no path from an initial to a final state in the product automaton $\overline{\mathcal{B}} \times \mathcal{B}^\uparrow$, where $\overline{\mathcal{B}}$ is the complement of $\mathcal{B}$, obtained by simply switching accepting and non-accepting states. As NFA emptiness is in NL, DFA monotonicity is in NL as well.

Now, let us suppose we have an algorithm which takes a DFA as input and returns whether it recognises a monotone language. Notice that the DFA emptiness problem is still NL − complete when restricted to automata not accepting the empty word ε. We will use this variant to perform a reduction to DFA monotonicity. Suppose we are given a DFA $\mathcal{B}$ on an alphabet A which does not accept ε. We build an automaton $\mathcal{B}'$ on $A \cup \{\top\}$ by adding the letter $\top$ to A in $\mathcal{B}$, but without any $\top$-labelled transition. Now, let us equip $A \cup \{\top\}$ with an order $\leq$ such that $a \leq \top$ for any letter a of A. Then the new automaton $\mathcal{B}'$ recognises a monotone language if and only if $\mathcal{B}$ recognises the empty language. Indeed, suppose we have a word u of length n accepted by $\mathcal{B}$. Then, $\mathcal{B}'$ would accept u but not $\top^n$ which is bigger than u. Reciprocally, if $\mathcal{B}$ recognises the empty language then so does $\mathcal{B}'$ and the empty language is a monotone language. Notice that we omitted here the constraint of using a powerset alphabet in order to give a shorter description of the proof, but it is straightforward to additionally enforce the restriction of powerset alphabet. Thus, the monotonicity problem is NL − complete when the input is a DFA.

◀

C Refinements of the counter-example language K

C.1 An $\mathrm{FO}^2[\mathfrak{B}_0 \cup \mathfrak{be}]$-formula for the counter-example

Let us give a formula for the counter-example from Proposition 33.

Let us notice that the successor predicate is definable in $\mathrm{FO}^2[\mathfrak{B}_< \cup \mathfrak{be}]$, so results from [11] about the fragment $\mathrm{FO}^2[<, \mathfrak{be}]$ apply to $\mathrm{FO}^2[\mathfrak{B}_0 \cup \mathfrak{be}]$ as well.

So it is easy to describe $A^*(\top \cup \binom{a}{b}^2 \cup \binom{b}{c}^2 \cup \binom{c}{a}^2 \cup \binom{a}{b}\binom{c}{a} \cup \binom{b}{c}\binom{a}{b} \cup \binom{c}{a}\binom{b}{c})A^*$ and to state that factors of length 3 are in $(abc)^\uparrow \cup (bca)^\uparrow \cup (cab)^\uparrow$.

If $d \in \Sigma$, let us note $\{d\}$ the predicate stating that d is the only true atomice predicate, i.e. $\{d\}(x) := d(x) \wedge \bigwedge_{e \in \Sigma \setminus \{d\}} \neg e(x)$.

Now, for any atomic predicates s and t (i.e. $s, t \in \{a, b, c\}$), let us pose:

$$\varphi_{s,t} = \forall x, \forall y, \left(s(x) \wedge t(y) \wedge x < y \wedge \neg \bigvee_{d \in \Sigma} \{d\}(x, y) \right) \implies \psi_{s,t}(x, y)$$

where $\psi_{s,t}(x, y)$ is a formula stating that the two anchors are compatible, i.e. either they both use the "upper component" of all the double letters between them, or they both use the "bottom component". Recall that $\neg \bigvee_{d \in \Sigma} \{d\}(x, y)$ means that there is no singleton letter between x and y.

For example, $\psi_{a,b}(x, y)$ is the disjunction of the following formulas:

$$\binom{b}{c}(x+1) \wedge \binom{a}{b}(y-1)$$
$$\binom{a}{b}(x+1) \wedge \binom{c}{a}(y-1)$$
$$x+1 = y$$

Indeed, the first case correspond to using the upper component of $\binom{b}{c}$ and $\binom{a}{b}$: anchor a in position x is followed by the upper b in position $x+1$, which should be consistent with the upper a in position $y-1$ followed by anchor b in position y, the factor from $x+1$ to $y-1$ being of the form $(\binom{b}{c}\binom{c}{a}\binom{a}{b})^+$. Similarly, the second case corresponds to the bottom component. The last case corresponds to anchors directly following each other, without an intermediary factor of double letters. This case appears only for $(s, t) \in \{(a, b), (b, c), (c, a)\}$

Now using the conjunction of all formulas $\varphi_{s,t}$ where s and t are atomic predicates a, b, c, we build a formula for the language of Proposition 33.

C.2 Proof of Proposition 34

Suppose Σ reduced to a singleton. Then, A is reduced to two letters which we note 0 and 1 with 1 greater than 0. We will again rely on the proof from [12, Sec. 4]. We will encode each predicate from $\{a, b, c\}$ and a new letter # (the separator) into A^* as follows:

$$\begin{cases} [a] = 100 \\ [b] = 010 \\ [c] = 001 \\ [\#] = 100001 \end{cases}$$

Thus, the letter $\binom{a}{b}$ will be encoded by $[ab] = 110$, etc. We will encode the language K as follows:

$$[K] = (([a][\#][b][\#][c][\#])^*)^\uparrow \cup A^*1(A^4\backslash 0^4)1A^* \cup A^*1^5A^*.$$

First, let us verify that $[K]$ is monotone. Indeed, $[K]$ is defined as the union of $K_1 = (([a][\#][b][\#][c][\#])^*)^\uparrow$ and $K_2 = A^*1(A^4\backslash 0^4)1A^* \cup A^*1^5A^*$. It is clear that K_1 is monotone since it is defined as a monotone closure, and K_2 is monotone as well, since it is defined by local monotone patterns. Since monotone languages are closed under union, $[K]$ is monotone. The role of adding K_2 here is to ensure that words breaking the encoding pattern because of monotonicity constraints are all accepted, and can therefore be ignored.

We now show that $[K]$ is FO-definable. Let us show how the separator $[\#]$ is used. Let w be a word over A^*. If w contains a factor of the form $1u1$ where u is a word of 4 letters containing the letter 1, then w immediately belongs to $[K]$. This is easy to check with an FO-formula so we can suppose that w does not contain such a factor. Similarly, we can suppose that 1^5 (corresponding to $\top$ in the original K) is not a factor of w. Then, it is easy to locate a separator since 100001 will always be a separator factor. Therefore, we can locate factors coding letters in w. Then we can do the same thing as [12] to find an FO-formula: we have to fix some anchors (factors coding letters whose roles are not ambiguous as explained in the proof of Proposition 33) and check whether they are compatible. For example, suppose w contains a factor of the form $[a][\#]([ab][\#][bc][\#][ca][\#])^n[bc]$. Then $[a]$ is an anchor. The last factor $[ca][\#][bc]$ is also an anchor since it can only be interpreted as $([a][\#][b])^\uparrow$ as a witness of membership in $[K]$. Since there are no anchors in between $[a]$ and $[bc]$ we just have to verify their compatibility. Here it is the case: in between the anchors, each $[ab]$ can be interpreted as $[b]^\uparrow$, $[bc]$ as $[c]^\uparrow$ and $[ca]$ as $[a]^\uparrow$. If we were to replace the first $[a]$ with $[c]$, this $[c]$ would still be an anchor but would not be compatible with the next anchor $[bc]$. This achieves the informal description of an FO-formula for $[K]$, adapted from the one from [12] to accomodate the encoding.

Furthermore, it is not FO^+-definable, for the same reason as for K. Indeed, let $k \in \mathbb{N}$ be an arbitrary number of rounds for an EF^+-game, as described in [12, Sec. 3.3]. We can choose $n > 2^k$ such that Duplicator has a winning strategy for u_0 and u_1 defined as follows:

$$[u_0] = ([a][\#][b][\#][c][\#])^n \text{ and } [u_1] = ([ab][\#][bc][\#][ca][\#])^n[ab][\#]$$

where $[ab] = 110$, $[bc] = 011$ and $[ca] = 101$.

We can adapt the strategy for u_0 and u_1 (from [12, Lemma 4.4]) to $[u_0]$ and $[u_1]$. For example, if Spoiler plays the i-th letter of a factor $[bc]$, then it is similar to playing the letter $\binom{b}{c}$ in u_1. Thus, if Duplicator answers by playing the j-th b or c in u_0, then he should answer by playing the i-th letter of the j-th $[b]$ or $[c]$ respectively, for any natural integers i and j. In the same way, if Spoiler plays in a separator character, then Duplicator should answer by playing the same letter of the corresponding separator character in the other word according to the strategy.

D Stability through monotone closure: Proof of Theorem 35

In order to prove Theorem 35, we shall introduce a useful definition:

▶ **Definition 46.** *For any language L, we write $L^{\curlywedge} = ((L^c)^\downarrow)^c$ for the dual closure of L, where L^c stands for the complement of L and $L^\downarrow := \{u \in A^* \mid \exists v \in L, u \leq_{A^*} v\}$ is the downwards monotone closure of L.*

▶ Remark 47. $L^{\curlywedge}$ is the greatest monotone language included in L for any language L. Indeed, for $u \in A^*$, we have $u \in L^{\curlywedge}$ if and only if $u \notin (L^c)^\downarrow$ if and only if $u^\uparrow \subseteq L$. In particular, a monotone language is both equal to its monotone closure and its dual monotone closure.

Now, let us show the following lemma:

▶ **Lemma 48.** *The set Σ_2^+ captures the set of monotone Σ_2-definable languages.*

Proof. First, any formula from Σ_2^+ is in particular a formula from Σ_2, and also a formula from FO^+, so the language it describes is monotone and Σ_2-definable.

Next, it is enough to show that the monotone closure of a Σ_2-definable language is Σ_2^+-definable.

So let us consider a Σ_2-definable language L. Since a disjunction of Σ_2^+ formulas is equivalent to a Σ_2^+ formula, we can suppose thanks to [17] that L is of the form $A_0^*.s_0.A_1^*.s_1 \dots s_t.A_{t+1}^*$ as explained above.

Therefore, $L^\uparrow$ is described by the following Σ_2^+-formula:

$$\exists x_0, ..., x_t, x_0 < ... < x_t \wedge \bigwedge_{i=0}^{t} s_i(x_i) \wedge \forall y, \bigwedge_{i=0}^{t+1} ((y \leq x_{i-1}) \vee (x_i \leq y) \vee A_i(y))$$

where $B(x)$ for a subalphabet B means $\bigvee_{b \in B} b(x)$, and $y \leq x_{-1}$ is $\top$. ◀

This immediately gives the following lemma which uses the same proof:

▶ **Lemma 49.** *The set Σ_2^- (Σ_2-formulas with negations on all predicates) captures the set of downwards-closed Σ_2-definable languages.*

We can now deduce the following lemma:

▶ **Lemma 50.** *The set Π_2^+ captures the set of monotone Π_2-definable languages.*

Proof. Then again, we only need to show the difficult direction.

Let L be a Π_2-definable language. It is enough to show that $L^{\curlywedge}$ is Π_2^+-definable according to Remark 47.

By definition of Π_2, the complement L^c of L is Σ_2-definable. Hence, $(L^c)^{\downarrow}$ is definable by a Σ_2^--formula φ given by Lemma 49. Therefore, $\neg\varphi$ is a formula from Π_2^+ describing $L^{\curlywedge}$. ◀

Finally, we can prove Theorem 35:

Proof. Thanks to [20], it is straightforward that any language from $\Sigma_2^+ \cap \Pi_2^+$ is monotone and FO^2-definable.

Let L be a monotone FO^2-definable language.

In particular, L belongs to Σ_2 and is monotone. Thus, by Lemma 48, L belongs to Σ_2^+. Similarly, L belongs to Π_2^+ by Lemma 50. ◀

E Games

Erhenfeucht-Fraïssé games and their variants are traditionally used to prove negative expressivity results of FO fragments. We provide here a variant that characterises the fragment FO^{2+} without quantifier alternations, which is used in the proof of proposition 38.

▶ **Definition 51.** *Let u and v be two words over an ordered alphabet A, and $k \in \mathbb{N}$. We denote by $EF^{2+}_{k,alt-free}(u,v)$ the following game (the* Erhenfeucht-Fraïssé game for FO^{2+} without alternations with k turns over u and v*), between two players Spoiler and Duplicator.*

Both players have two token (t_S^1, t_S^2 and t_D^1, t_D^2), initially not in play, that they will place on positions in the words u and v. Before the first turn, Spoiler chooses u or v, and will place his tokens only in that word; Duplicator will play only in the other word.

A turn consists in Spoiler placing or moving one of his tokens t_S^r in his chosen word, and Duplicator answering by placing t_D^r in the other word, respecting the following rule: if t_S^r is on position i_S^r and t_D^r on position i_D^r, and $i_S^1 < i_S^2$, then $i_D^1 < i_D^2$; and similarly if $i_S^1 > i_S^2$ or $i_S^1 = i_S^2$. Note that this rule says nothing for the first turn, where no token is placed yet.

After each turn, Duplicator loses the game if either:

- *she could not place her token without breaking the rule;*
- *or the letters a_u^1, a_u^2, a_v^1 and a_v^2 in u and v pointed by the tokens (if they are placed) do not satisfy $a_u^r \leq a_v^r$.*

If Duplicator did not lose after a turn, and it was not the k-th turn, then a new turn begins. If Duplicator did not lose after turn k, then Duplicator wins the game. In the particular case $k = 0$, Duplicator wins automatically.

The fundamental property of this game, allowing to prove expressibility results about FO^{2+} without alternation, is the following:

▶ **Theorem 52.** *Spoiler has a winning strategy for $EF^{2+}_{k,alt-free}(u,v)$ if and only there is a closed formula φ in FO^{2+} without alternation, with quantifier depth at most k, such that $u \models \varphi$, but $v \not\models \varphi$.*

Proof. We adapt the standard proofs for Ehrenfeucht-Fraïssé games to our setting, in the context of positive logic (see [12, Theorem 5.7]). We start as usual by generalising the statement to formulas with free variables.

A *configuration* for a word u of length n is a partial map $c : \{x, y\} \to [\![1, n]\!]$ (representing the positions of the tokens t^1 and t^2 on u when defined, or the fact that the token is not yet placed when undefined). We show that Spoiler has a winning strategy for $EF^{2+}_{k,alt-free}(u, v)$ where some tokens are possibly initially placed on the words (respecting the fact that Spoiler and Duplicator must play in different words exclusively; in particular, Spoiler cannot choose his word if at least one token is already placed, and if $k = 0$, Duplicator wins if and only if the tokens are placed so as to respect the constraints of the game) if and only if there is a formula φ (non-necessarily closed) in FO^{2+} without alternation, with quantifier depth at most k, such that $u, c \models \varphi$, but $v, c' \not\models \varphi$, where c and c' are the configurations on u and v associated with the current placement of the tokens; moreover, if Spoiler has one or both tokens in u in the initial configuration, then φ does not contain the quantifier $\forall$; and if Spoiler has one or both tokens in v in the initial configuration, φ does not contain the quantifier $\exists$. This statement clearly implies the theorem.

We prove the statement by induction on k.

Case $k = 0$: Assume first that Spoiler wins the game. Since there is no turn to be taken, and Duplicator wins automatically if no token is placed, at least one token must be placed. Since Spoiler wins, this means that the positions of the tokens do not respect one of the constraints. For example, if t^1_S points to a position i^1_u in u, and t^1_D points to i^1_v in v, with the associated letters a_u and a_v such that $a_u \not\leq a_v$, then the formula $a_u(x)$ is satisfied by u but not by v. If the order of the tokens is not respected, we use a formula of the form $x < y$. All cases are similar to these.

Now, assume that there is a formula φ without quantifiers satisfied by (u, c) but not by (v, c'). Rewriting φ in disjunctive form $\bigvee \varphi_i$, there is i such that u satisfies φ but v does not. Since φ_i is a conjunction of atomic formulas (since there are no quantifiers), there is some atomic formula ψ satisfied by u, but not by v. The formula ψ cannot be $\top$ or $\bot$, so it has to be of the form $a(x)$, $x < y$ or $x \leq y$. In all cases, the formula ψ witnesses a failure of the tokens to respect the constraints of the game, so Spoiler wins immediately.

Case $k > 0$: First, assume that there is a formula φ with quantifier depth k, no quantifier alternation, satisfied by (u, c) but not (v, c'), and respecting the constraints on quantifiers appearing in it. Then as in the base case writing φ is disjunctive form, we obtain a formula ψ which is either atomic or starts with a quantifier, and that is satisfied by u but not by v. If ψ is atomic, as in the base case Spoiler wins immediately. Otherwise, ψ starts with a quantifier. We distinguish the cases where tokens are already placed, or not.

If there is no token already placed in the configuration, then assume that ψ is $\exists x, \theta$ (the other cases with variable y or quantifier $\forall$ are similar, just changing the token and the word in which Spoiler plays). Then Spoiler places token t^1_S in u on a position such that θ is satisfied by (u, c) where c is the new configuration in u. Since v does not satisfy ψ, Duplicator will place t^1_D in some position such that (v, c') is not satisfied by θ where c' is the new configuration on v. The quantifier depth of θ is at most $k - 1$, and θ cannot contain the quantifier $\forall$ since φ was alternation-free. By induction, Spoiler wins the game $EF^{2+}_{k-1,alt-free}(u, v)$ starting with these configurations, so also $EF^{2+}_{k,alt-free}(u, v)$ in the initial configuration.

If there is a token already placed for Spoiler in u (again, the other case is similar, replacing $\exists$ by $\forall$), then ψ has to start with a $\exists$, and we can apply the same strategy as previously since Spoiler keeps playing in the same word and we cannot encounter the quantifier $\forall$ since φ does not contain it.

This concludes the proof of the first direction.

Conversely, assume that Spoiler wins the game $EF^{2+}_{k,alt-free}(u,v)$ starting with configurations c and c' on u and v. Assume that Spoiler plays his first winning move — say by placing t^1_S — in u (either because the initial configurations force him to, or because he chooses to in his first move) — as usual, the other case is treated similarly by exchanging $\exists$ and $\forall$ — and denote by d, d' the new configurations after Duplicator answered this first move.

Spoiler must win the game $EF^{2+}_{k-1,alt-free}(u,v)$ on these new configurations, so by induction there is some formula $\psi_{d'}$ of quantifier depth at most $k-1$ containing no $\forall$ quantifier that is satisfied by (u,d) and not by (v,d') (note that this formula depends on d', i.e. on Duplicator's move). Consequently, the formula $\bigwedge_{d'} \psi_{d'}$ is satisfied by (u,d), but by none of the (v,d') where d' spans all possible configurations after Duplicator's first move. And finally, the formula $\varphi = \exists x, \bigwedge_{d'} \psi_{d'}$ is satisfied by (u,c), as is witnessed by the position played in Spoiler's first move, but is not satisfied by (v,c'), as is witnessed by all of Duplicator's possible answers.

This formula φ has quantifier depth at most k, and contains no quantifier $\forall$, so is indeed the formula we where looking for.

This concludes the proof. ◀

Note that this game and the associated theorem could be adapted to talk about sublogics of FO obtained by restricting the number of available variables (the corresponding game is the same, but using a different amount of tokens for each player); and by restricting the number of alternations to some number m (the corresponding game being the same, but allowing Spoiler to change the word in which he plays at most m times).

Finally, this theorem has the following immediate consequence, which allows to prove inexpressibility results in FO^{2+} without alternations:

▶ **Corollary 53.** *A language L over an ordered alphabet A is definable in FO^{2+} without alternations if and only if there is some k such that for all words $u \in L$ and $v \notin L$, Spoiler wins $EF^{2+}_{k,alt-free}(u,v)$.*

Proof. If L is definable, consider a formula φ defining it, and k its quantifier depth. Then by theorem 52, Spoiler wins $EF^{2+}_{k,alt-free}(u,v)$.

Conversely, recall first that there is finitely many inequivalent formulas in FO^{2+} without alternation of depth at most k (this is a consequence of the similar fact for FO^2, and the fact that all formulas in FO^{2+} are equivalent to a formula of FO^2 with the same depth).

If Spoiler wins $EF^{2+}_{k,alt-free}(u,v)$ for all $u \in L$ and $v \notin L$, then for all such (u,v) there is a formula $\varphi_{u,v}$ without alternation and of quantifier depth at most k satisfied by u and not by v. These formulas are finitely many (up to equivalence), and the infinite formula $\bigvee_{u\in L} \bigwedge_{v\notin L} \varphi_{u,v}$, which defines L, is in fact equivalent to a finite positive boolean combination of finitely many formulas $\varphi_{u,v}$. Hence, L is defined by this formula, which is without alternation and of depth at most k, as needed. ◀

E.1 An alternation-free formula for the counterexample of Section 5.3

The language 1^+11^+ can be defined by the formula stating that all letters are 1, and there are at least three letters:

$$\varphi_{111} = (\forall x, 1(x)) \wedge (\exists x, \exists y, (y > x) \wedge \exists x, (x > y))$$

and the language 1^+01^+ is defined by the conjunction of the formulas:

$$\psi_0 = \forall x(1(x) \vee \forall y, (y = x \vee 1(y)))$$

stating that there is at most one 0, and:

$$\psi_{101} = \exists x(\neg 1(x) \wedge \exists y, (y < x \wedge 1(y)) \wedge \exists y(y > x \wedge 1(y)))$$

stating that 101 appears as a subword (note that the only instance of a negation is in this last formula).

The language L is the union of these two languages, so in definable in FO^2 without alternation.

E.2 Proof of Proposition 38

We will make use of the Ehrenfeucht-Fraïssé game for FO^{2+} without alternation (see Appendix E for details on this game). Fix a natural number n, and consider the words $u = 1^{2n}01 \in L$ and $v = 1^{2n}0 \notin L$. We will show that Duplicator wins $EF^{2+}_{n,alt-free}(u, v)$, which will entail that there is no FO^{2+} formula without alternation that can define L.

First, assume that Spoiler plays in v. Then Duplicator can copy in u all positions played by Spoiler in v; the letters played by Spoiler and Duplicator are the same, so Duplicator wins the game (with any number of turns).

Now assume that Spoiler plays in u.If Spoiler plays a token in one of the first n positions, say at position $i \in [\![0, n-1]\!]$, then Duplicator copies the same position in v, except if this position is already taken by the other token and Spoiler did not play both tokens at the same place, in which case Duplicator plays i or $i-2$ depending on which of these two positions respects the order of the tokens in u. If Spoiler plays a token in the last $n+2$ positions, say at position $i \in [\![n, 2n+1]\!]$, then Duplicator answers by playing $i-1$ except if the position is already taken by the other token, in which case Duplicator applies the same strategy as before. Recall that it is allowed for Duplicator to answer a position labelled 1 in v to a move of Spoiler labelled 0 in u, as only an inequality between labels has to hold.

We see that one of the optimal plays for Spoiler against this strategy is to play position n as his first move, to which Duplicator answers with $n-1$, then "pushing" the tokens of Duplicator by playing successively $n-1$ with his other token (forcing Duplicator to play $n-2$), and so on, until Spoiler can play the first position 0, to which Duplicator cannot answer. This takes $n+1$ moves, so this is not enough for Spoiler to win in the n-round game. Another choice is to start by playing position $n-1$ in u, that Duplicator replicates in v, and then increasing positions one by one until Duplicator is forced to answer 1 with 0 at position $2n$. This takes $n+2$ moves, so Duplicator still wins the n-round game. Any deviation of Spoiler from these strategies will be useless, as Duplicator can end up in the same position as in these plays, but after Spoiler has consumed more rounds. This shows that Duplicator wins the n-round game with this strategy, and concludes the proof.