



HAL
open science

Positive and monotone fragments of FO and LTL

Denis Kuperberg, Quentin Moreau

► **To cite this version:**

Denis Kuperberg, Quentin Moreau. Positive and monotone fragments of FO and LTL. 2024. hal-04774430

HAL Id: hal-04774430

<https://hal.science/hal-04774430v1>

Preprint submitted on 8 Nov 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Positive and monotone fragments of FO and LTL

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.

2012 ACM Subject Classification Theory of computation \rightarrow Modal and temporal logics; Theory of computation \rightarrow Regular languages; Theory of computation \rightarrow 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, 20], 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.

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 [18] with a simpler proof. Recently, this preservation property of FO was more specifically shown



© Denis Kuperberg and Quentin Moreau;
licensed under Creative Commons License CC-BY 4.0

42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:21



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to fail already on finite graphs and on finite words by Kuperberg [12], implying the failure on finite structure with a more elementary proof than [1, 18]. 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 [17] 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 $\text{FO}^2[<]$. This is obtained via a stronger result characterizing 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].

The goal of this work is to understand at what point the phenomenon discovered in [12] come 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 (Lemma 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 32 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 34 and we show that we need only one predicate to find a counter-example in FO in Proposition 35.

We also consider a characterization of $\text{FO}^2[<]$ from Thérien and Wilke [19] stating that $\text{FO}^2[<]$ 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 characterized by $\Sigma_2^+ \cap \Pi_2^+$.

At last, we show that no counter-example for FO can be found in FO^2 (without successor available) in Corollary 42. We conclude by leaving 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 43).

2 FO and LTL

We work with a set of atomic unary predicates $\Sigma = \{a_1, a_2, \dots, a_{|\Sigma|}\}$, and consider the set of words on alphabet $\mathcal{P}(\Sigma)$. 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 $\mathfrak{B}_0 := \{\leq, <, \text{succ}, \text{nsucc}\}$, $\mathfrak{B}_< := \{\leq, <\}$ and $\mathfrak{B}_{\text{succ}} := \{=, \neq, \text{succ}, \text{nsucc}\}$, 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 $\text{FO}[\mathfrak{B}]$ is as follows:

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

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, \dots, x_n , and u a word. Let ν be a function of $\{x_1, \dots, x_n\}$ in $\llbracket 0, |u| - 1 \rrbracket$. 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 $u, \nu \models \perp$,
- $u, \nu \models x < y$ if $\nu(x) < \nu(y)$,
- $u, \nu \models x \leq y$ if $\nu(x) \leq \nu(y)$,
- $u, \nu \models \text{succ}(x, y)$ if $\nu(y) = \nu(x) + 1$,
- $u, \nu \models \text{nsucc}(x, y)$ 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),

23:4 Positive and monotone fragments of FO and LTL

- $u, \nu \models \varphi \wedge \psi$ if $u, \nu \models \varphi$ and $u, \nu \models \psi$,
- $u, \nu \models \varphi \vee \psi$ if $u, \nu \models \varphi$ or $u, \nu \models \psi$,
- $u, \nu \models \exists x, \varphi(x, x_1, \dots, x_n)$ if there is i of u such that we have $u, \nu \cup [x \mapsto i] \models \varphi$,
- $u, \nu \models \forall x, \varphi(x, x_1, \dots, x_n)$ 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 $\text{FO}^+[\mathfrak{B}_<]$ 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 $\text{FO}[\mathfrak{B}_0]$ or $\text{FO}[\mathfrak{B}_<]$, 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 $u \leq_{\mathcal{P}(\Sigma)^*} v$.

► **Definition 7.** Let L be a language. We say that L is monotone when for any word u of L , any word greater than u belongs to L .

► **Proposition 8 ([12]).** FO^+ formulas are monotone in unary predicates, i.e. if a model (u, ν) satisfies a formula φ of FO^+ , and $u \leq_{\mathcal{P}(\Sigma)^*} v$, 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 on 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 less.

► **Definition 9.** 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 10.** The formula $\exists y, \text{succ}(x, y) \wedge (\exists x, b(x) \wedge (\forall z, z \geq x \vee z < y \vee a(z)))$ (a formula with one free variable x that indicates that the letter labeled by x will be followed by a factor of the form $aaaaa\dots aab$) 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, see 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 11.** *The grammar of LTL is as follows:*

$$\varphi, \psi ::= \perp \mid \top \mid a \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid X\varphi \mid \varphi U\psi \mid \varphi R\psi \mid \neg\varphi.$$

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 $X\varphi$ if and only if the suffix u satisfies φ . The construction $\varphi U\psi$ (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 $\varphi R\psi$ as being equal to $\neg(\neg\varphi U\neg\psi)$. Let us define this formally:

► **Definition 12.** *Let φ be an LTL formula, and $u = u_0\dots u_{m-1}$ be a word. We say that u satisfies φ and define $u \models \varphi$ by induction on φ as follows:*

- $u \models \top$ and we never have $u \models \perp$,
- $u \models a$ if $a \in u[0]$,
- $u \models \varphi \wedge \psi$ if $u \models \varphi$ and $u \models \psi$,
- $u \models \varphi \vee \psi$ if $u \models \varphi$ or $u \models \psi$,
- $u \models X\varphi$ if $u_1\dots u_{m-1} \models \varphi$,
- $u \models \varphi U\psi$ if there is $i \in \llbracket 0, m-1 \rrbracket$ such that $u_i\dots u_{m-1} \models \psi$ and for all $j \in \llbracket 0, i-1 \rrbracket$, $u_j\dots u_{m-1} \models \varphi$,
- $u \models \varphi R\psi$ if $u \models (\psi U(\psi \wedge \varphi))$ or for all $i \in \llbracket 0, m-1 \rrbracket$ we have $u_i\dots u_{m-1} \models \psi$,
- $u \models \neg\varphi$ if we do not have $u \models \varphi$.

► **Remark 13.** Let us call $\varphi XU\psi$ the formula $X(\varphi U\psi)$, 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 (since) and Q the respective past analogues of X, U and R.

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

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

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

► **Remark 15.** As for XU, we will write $\varphi YS\psi$ for $Y(\varphi S\psi)$. We also note $P\varphi$, $F\varphi$, $H\varphi$ and $G\varphi$ for $\top YS\varphi$, $\top XU\varphi$, $\varphi YS\perp$ and $\varphi XU\perp$ respectively. The formulas $F\varphi$ and $G\varphi$ 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.

23:6 Positive and monotone fragments of FO and LTL

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, $u = u_0 \dots u_{m-1}$ a word, and $i \in \llbracket 0, m-1 \rrbracket$. We define $u, i \models \varphi$ by induction on φ :

- $u, i \models \top$ and we never have $u \models \perp$,
- $u, i \models a$ 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 $u, i \models \psi$,
- $u, i \models X\varphi$ if $u, i+1 \models \varphi$,
- $u, i \models \varphi U \psi$ if there is $j \in \llbracket i, m-1 \rrbracket$ such that $u, j \models \psi$ and for all $k \in \llbracket i, j-1 \rrbracket$, $u, k \models \varphi$,
- $u, i \models \psi R \varphi$ if $u, i \models \neg(\neg\psi U \neg\varphi)$,
- $u, i \models \neg\varphi$ if we do not have $u, i \models \varphi$,
- $u, i \models Y\varphi$ if $u, i-1 \models \varphi$,
- $u, i \models \varphi S \psi$ if there is $j \in \llbracket 0, i \rrbracket$ such that $u, j \models \psi$ and for all $k \in \llbracket j+1, i \rrbracket$, $u, k \models \varphi$.

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:

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

We define $UTL[P, F, H, G]$ 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 $UTL^+[P, F, H, G]$ by deleting the negation in $UTL[P, F, H, G]$.

► **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 starting position for TL semantics.

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 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 $\text{qr}(\varphi)$ be the quantification rank of a formula φ of FO^+ defined inductively by:*

- *if φ contains no quantifier then $\text{qr}(\varphi) = 0$,*
- *if φ is of the form $\exists x, \psi$ or $\forall x, \psi$ then $\text{qr}(\varphi) = \text{qr}(\psi) + 1$,*
- *if φ is of the form $\psi \vee \chi$ or $\psi \wedge \chi$ then $\text{qr}(\varphi) = \max(\text{qr}(\psi), \text{qr}(\chi))$.*

► **Definition 23.** *A separated formula is a positive Boolean combination of purely past formulas (which do not depend on the present and future), purely present formulas (which do not depend on the past and future) and purely future formulas (which do not depend on the past and present).*

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 .*

Proof (Sketch). Our starting point is the proof given by Kuperberg and Vanden Boom in [13, lemma 5], 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.

First of all, [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, and 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 Boolean combination of purely past, present and future formulas themselves in TL^+ . ◀

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], 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 $\text{qr}(\varphi)$ that φ is equivalent to a formula of TL^+ .

Initialisation:

If $\text{qr}(\varphi)$ 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 \perp (e.g. $x < x$ translates into \perp and $x = x$ into \top). For example, $(x \leq x \wedge a(x)) \vee (b(x) \wedge c(x)) \vee x < x$ translates into $(\top \wedge a) \vee (b \wedge c) \vee \perp$.

Heredity:

Suppose that any FO^+ free single-variable formula of quantification rank strictly less than $\text{qr}(\varphi)$ translates into a TL^+ formula, and $\text{qr}(\varphi)$ 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, \dots, a_n where n is a natural number, the letters (which are considered as unary predicates) in $\psi(x, y)$ applied to x .

For any subset S of $\llbracket 1, n \rrbracket$, 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 \perp otherwise, for any integer i of $\llbracket 1, n \rrbracket$.

We then have the logical equivalence:

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

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

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

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 of all, 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 $\llbracket 1, n \rrbracket$ such that $(\bigwedge_{i \in S} a_i(x)) \wedge \psi^S(x, y)$ is satisfied. In particular, according to the values taken by the unary predicates in x , there exists a subset S' of $\llbracket 1, n \rrbracket$ containing S such that $(\bigwedge_{i \in S'} a_i(x)) \wedge (\bigwedge_{i \notin S'} \neg a_i(x)) \wedge \psi^S(x, y)$ is satisfied. Now, ψ is monotone in the different predicates a_1, \dots, a_n . So $(\bigwedge_{i \in S'} a_i(x)) \wedge (\bigwedge_{i \notin S'} \neg a_i(x)) \wedge \psi^{S'}(x, y)$ 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 $\llbracket 1, n \rrbracket$ 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 $P_{\mathfrak{b}}(z)$. For example, we can replace $x < z$, $z < x$ or $x = z$ by a unary predicate $P_{>}(z)$, $P_{<}(z)$ or $P_{=}(z)$. We then obtain a formula $\psi'^S(y)$ on which we can apply the induction hypothesis (since there is only one free variable). This yields a formula χ from TL^+ , equivalent to $\psi'^S(y)$ and we have:

$$\exists y, \psi^S(x, y) \equiv P\chi \vee \chi \vee F\chi, \quad \text{and} \quad \forall y, \psi^S(x, y) \equiv H\chi \wedge \chi \wedge G\chi.$$

Let χ' be the formula obtained ($P\chi \vee \chi \vee F\chi$ or $H\chi \wedge \chi \wedge G\chi$). The resulting formula χ' then involves unary predicates of the form $P_{\mathfrak{b}}$. We then use Lemma 24 to transform χ' into a positive Boolean combination of purely past, present and future positive formulas, where predicates $P_{\mathfrak{b}}$ trivialize into \top or \perp . For example, $P_{<}$ trivializes into \top in purely past formulas, into \perp 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. Indeed, we want to evaluate in $x = 0$, so the purely past formulas, isolated by the separation lemma (Lemma 24), trivialize into \perp 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 . ◀

We can now turn to the proof of Theorem 20.

Proof of Theorem 20. By Lemma 21, we have the inclusion of the languages described by LTL^+ in those described by FO^{3+} , which is trivially included in FO^+ . By Lemma 25, the converse inclusion of FO^+ into LTL^+ holds. So we can conclude that the three logical formalisms are equi-expressive. ◀

3.2 Equivalences in fragments of FO^+

► **Theorem 26.** *The languages described by $\text{FO}^{2+}[\mathfrak{B}_0]$ 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:

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

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 it 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. Using the same notations, let us consider a formula $\psi^S(x, y)$ with no unary predicate applied to x . 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*, $y < x \wedge \text{nsucc}(y, x)$, $\text{succ}(y, x)$, $y = x$, $\text{succ}(x, y)$ and $x < y \wedge \text{nsucc}(x, y)$, whose set is denoted T .

We then have the logical equivalence:

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

where $\psi_\tau^S(y)$ 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 $\psi^S(x, y) = \text{nsucc}(x, y) \wedge a(y) \wedge (\forall x, x \leq y \vee b(y))$ and for the position formula $\tau = y < x \wedge \text{nsucc}(y, x)$, we have $\psi_\tau^S(y) = \top \wedge a(y) \wedge (\forall x, x \leq y \vee b(y))$. 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.

We use disjunction in the case of an existential quantifier (as in [5]) and conjunction in the case of a universal quantifier. We then need to translate $\exists y, \tau(x, y) \wedge \psi_\tau^S(y)$ and $\forall y, \tau(x, y) \implies \psi_\tau^S(y)$, which we note respectively $[\tau]_\exists$ and $[\tau]_\forall$, in UTL^+ , for any position formula τ . For readability we omit ψ_τ^S in this notation, but $[\tau]_\exists$ and $[\tau]_\forall$ will depend on ψ_τ^S . In each case, we note χ for the UTL^+ formula obtained by induction from $\psi_\tau^S(y)$:

$$\begin{aligned} [y < x \wedge \text{nsucc}(y, x)]_\exists &\equiv \text{YP}\chi, \\ [y < x \wedge \text{nsucc}(y, x)]_\forall &\equiv \text{YH}\chi, \\ [\text{succ}(y, x)]_\exists &\equiv \text{succ}(y, x)]_\forall \equiv \text{Y}\chi, \\ [y = x]_\exists &\equiv [y = x]_\forall \equiv \chi, \\ [\text{succ}(x, y)]_\exists &\equiv \text{succ}(x, y)]_\forall \equiv \text{X}\chi, \\ [x < y \wedge \text{nsucc}(x, y)]_\exists &\equiv \text{XF}\chi, \\ [x < y \wedge \text{nsucc}(x, y)]_\forall &\equiv \text{XG}\chi. \end{aligned}$$

◀

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

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

For the left-to-right direction, simply replace the set T in Theorem 26 proof by $\text{T}' = \{y < x, y = x, x < y\}$. 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.

► **Lemma 29.** *Let $L \subseteq \mathcal{P}(\Sigma)^*$ be a regular language. Then L is monotone if and only if there is an order $\leq_{\mathbf{M}_L}$ on \mathbf{M}_L compatible with the product \cdot and included in \leq_L which verifies:*

$$\forall (u, v) \in \mathcal{P}(\Sigma)^* \times \mathcal{P}(\Sigma)^*, u \leq_{\mathcal{P}(\Sigma)^*} v \implies h(u) \leq_{\mathbf{M}_L} h(v),$$

where h denotes the canonical projection.

The proof is left in Appendix B.2.

► **Theorem 30.** *Let $L \subseteq \alpha^*$ be a regular language, and \leq_L be its syntactic order. The language L is monotone if and only if we have:*

$$\forall (s, s') \in \mathcal{P}(\Sigma)^2, s \subseteq s' \implies h(s) \leq_L h(s'),$$

where $h : \mathcal{P}(\Sigma)^* \rightarrow \mathbf{M}_L$ denotes the canonical projection 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 $h : \mathcal{P}(\Sigma)^* \rightarrow \mathbf{M}_L$ 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 conclude that $h(s) \leq_L h(s')$.

For the converse direction, suppose that \leq_L verifies the condition of Theorem 30. We can remark that \leq_L is compatible with the product of the monoid. Therefore, the conditions of Lemma 29 are verified by \leq_L . ◀

4.2 An algorithm to decide monotonicity

We immediately deduce from Theorem 30 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 canonical projection onto the syntactic monoid.

This algorithm works for any monoid that recognises L through a surjective $h : \mathcal{P}(\Sigma)^* \rightarrow M$, not just its syntactic monoid. Indeed, for any monoid, we start by restricting it to $h(\mathcal{P}(\Sigma)^*)$ 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 31.** *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 $O(|\mathcal{P}(\Sigma)|^2 |\mathbf{M}|^2)$.*

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.

We give a more precise result for DFA, and give also the complexity for monoid input:

► **Proposition 32.** *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.*

See Appendix B.3 for the proof.

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.

From now on we will write A the alphabet $\mathcal{P}(\Sigma)$.

5.1 Refinement of the counter-example in the general case

In [12], 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:

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

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

$$u_0 = (abc)^n \text{ and } u_1 = \left(\begin{pmatrix} a \\ b \end{pmatrix} \begin{pmatrix} b \\ c \end{pmatrix} \begin{pmatrix} c \\ a \end{pmatrix} \right)^n \begin{pmatrix} a \\ b \end{pmatrix} \begin{pmatrix} b \\ c \end{pmatrix},$$

where n is greater than 2^k , and $\begin{pmatrix} s \\ t \end{pmatrix}$ 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]. 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

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

► **Definition 33.** [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 $(u_i, \nu) \models a$, where u_i is the i -th letter of u .*

We denote \mathfrak{bc} the set of between predicates and \mathfrak{bc}^+ the set of between predicates associated to the set of positive unary predicates.

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

► **Proposition 34.** *There exists a monotone language definable in $\text{FO}^2[\mathfrak{B}_0 \cup \mathfrak{bc}]$ which is not definable in $\text{FO}^{2+}[\mathfrak{B}_0 \cup \mathfrak{bc}^+]$.*

Proof. We can use the same words u_0 and u_1 defined above with the following language:

$$K \cup A^* \left(\begin{pmatrix} a \\ b \end{pmatrix}^2 \cup \begin{pmatrix} b \\ c \end{pmatrix}^2 \cup \begin{pmatrix} c \\ a \end{pmatrix}^2 \cup \begin{pmatrix} a \\ b \end{pmatrix} \begin{pmatrix} c \\ a \end{pmatrix} \cup \begin{pmatrix} b \\ c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} \cup \begin{pmatrix} c \\ a \end{pmatrix} \begin{pmatrix} b \\ c \end{pmatrix} \right) A^*.$$

Indeed, in [12], it is explained that we need to look for some “anchor position” to know whether a word belongs to K . 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 as bc . Here, 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 for a more detailed description of a formula. \blacktriangleleft

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

Proof. 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 encode each predicate from $\{a, b, c\}$ and a new letter # (the separator) into A^* as follows:

$$\left\{ \begin{array}{l} [a] = 001 \\ [b] = 010 \\ [c] = 100 \\ [\#] = 100001 \end{array} \right. .$$

Thus, the letter $\binom{a}{b}$ will be encoded by $[ab] = 011$, etc. We will encode the language K as follows:

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

First, we can notice that $[K]$ is monotone.

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 factor 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 34) 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 $([a][\#][b])^\uparrow$. 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 $[a]$ with $[c]$, $[c]$ would still be an anchor but would not be compatible with $[bc]$. This achieves the description of an FO-formula for $[K]$.

Furthermore, it is not FO^+ -definable. Indeed, let $k \in \mathbb{N}$ be an arbitrary number of rounds for an EF^+ -game. 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 $[ab] = 011$, $[bc] = 110$ 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. ◀

5.2 Stability through monotone closure

It has been shown by Thérien and Wilke [19] that languages $\text{FO}^2[\mathfrak{B}_<]$ -definable are exactly those who are both Σ_2 -definable and Π_2 -definable where Σ_2 is the set of FO-formulas of the form $\exists x_1, \dots, x_n \forall y_1, \dots, y_m \varphi(x_1, \dots, x_n, y_1, \dots, y_m)$ 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 [16] showed that Σ_2 describes the unions of languages of the form $A_0^*.s_0.A_1^*.s_1.\dots.s_t.A_{t+1}^*$, where t is a natural integer, s_i are letters from A and A_i are subalphabets of A .

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 36.** *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.*

In order to prove Theorem 36, we shall introduce a useful definition:

► **Definition 37.** *For any language L , we write $L^\wedge = ((L^c)^\downarrow)^c$ the dual closure of L , where L^c stands for the complement of L and L^\downarrow is the downwards monotone closure of L .*

► **Remark 38.** It is straightforward to show that L^\wedge is the greatest monotone language included in L for any language 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 39.** *The set Σ_2^+ captures the set of monotone Σ_2 -definable languages.*

Proof. First, it is clear that Σ_2^+ describes monotone Σ_2 -definable languages.

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 [16] 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:

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

where $B(x)$ means $\bigvee_{b \in B} b(x)$ for any subalphabet B , $x_{-1} < y < x_0$ means $y < x_0$ and $x_t < y < x_{t+1}$ means $x_t < y$. ◀

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

► **Lemma 40.** *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 41.** *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^\wedge is Π_2^+ -definable according to Remark 38.

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 40. Therefore, $\neg\varphi$ is a formula from Π_2^+ describing L^\wedge . ◀

Finally, we can prove Theorem 36:

Proof. Thanks to [19], 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 39, L belongs to Σ_2^+ . Similarly, L belongs to Π_2^+ by Lemma 41. ◀

This last result shows how close to capture monotone FO^2 -definable languages FO^{2+} is. However, it does not seem easy to lift the equivalence $\Sigma_2 \cap \Pi_2 = \text{FO}^2$ to their positive fragments as we did for the other classical equivalences in Section 3. Indeed, the proof from [19] relies itself on the proof of [16] which is mostly semantic while we are dealing with syntactic equivalences.

This immediately implies that a counter-example separating FO -monotone from FO^+ cannot be in $\text{FO}^2[\mathfrak{B}_<]$ as stated in the following corollary:

► **Corollary 42.** *Any monotone language described by an $\text{FO}^2[\mathfrak{B}_<]$ formula is also described by an FO^+ formula.*

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

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

► **Conjecture 43.**

- *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.

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 *Proceedings 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](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](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 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>.
- 17 Benjamin Rossman. Homomorphism preservation theorems. *J. ACM*, 55, 07 2008.
- 18 Alexei P. Stolboushkin. Finitely monotone properties. In *LICS, San Diego, California, USA, June 26-29, 1995*, pages 324–330. IEEE Computer Society, 1995.
- 19 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.

- 20 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.

A 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.

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

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+} . ◀

B Monoids

B.1 Algebraic definitions

► **Definition 44.** A semigroup is a pair (\mathbf{S}, \cdot) where \cdot is an associative internal composition law on the non-empty set \mathbf{S} .

► **Remark 45.** We allow ourselves the abuse of language which consists in speaking of the semigroup \mathbf{S} instead of the semigroup (\mathbf{S}, \cdot) .

► **Definition 46.** A monoid is a pair (\mathbf{M}, \cdot) which is a semigroup, and which has a neutral element noted $1_{\mathbf{M}}$ (or simply 1 when there is no ambiguity), i.e. which verifies:

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

► **Definition 47.** 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

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

Similarly, if \mathbf{M} and \mathbf{M}' are just semigroups, h is a morphism if it preserves the semigroup structure.

► **Definition 48.** 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 49.** 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 $(\mathcal{P}(\Sigma)^*, \cdot)$ into (\mathbf{M}, \cdot) such that $L = h^{-1}(h(L))$.

► **Definition 50.** Let L be a regular language, and $u, v \in \mathcal{P}(\Sigma)^*$ be any two words. We define the equivalence relation of indistinguishability denoted \sim_L on $\mathcal{P}(\Sigma)^*$. We write $u \sim_L v$ if:

$$\forall (x, y) \in \mathcal{P}(\Sigma)^* \times \mathcal{P}(\Sigma)^*, xuy \in L \iff xvy \in L.$$

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

$$\forall (x, y) \in \mathcal{P}(\Sigma)^* \times \mathcal{P}(\Sigma)^*, xuy \in L \implies xvy \in L.$$

The \leq_L preorder is called the L syntactic preorder.

► **Definition 51.** Let L be a regular language. We define the syntactic monoid of L as $\mathbf{M}_L = L / \sim_L$.

► **Remark 52.** This is effectively a monoid, since \sim_L is compatible with left and right concatenation. Moreover, the syntactic monoid recognises L through canonical projection. 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 Lemma 29

Proof. The right-to-left direction follows from the definition of monotone languages. Indeed, suppose we have a language L and an order $\leq_{\mathbf{M}_L}$ on its syntactic monoid that verifies the assumptions. Let u be a word in L , and $v \geq_{\mathcal{P}(\Sigma)^*} u$. By hypothesis, we have $h(v) \geq_{\mathbf{M}_L} h(u)$. Again by hypothesis, since $h(u) \in h(L)$, we also have $h(v) \in h(L)$, so v belongs to L . We can conclude that L is monotone.

Conversely, let us consider a regular language L , and note h its canonical projection onto its syntactic monoid. Let \rightarrow be the binary relation induced by $\leq_{\mathcal{P}(\Sigma)^*}$ on \mathbf{M}_L , i.e. such that $m \rightarrow n$ if there are words u and v such that $m = h(u)$, $n = h(v)$ and $u \leq_{\mathcal{P}(\Sigma)^*} v$. The transitive closure of \rightarrow , denoted \rightarrow^* , is then an order relation.

First of all, it is clearly reflexive and transitive.

Then, to show antisymmetry, it is sufficient to show that \rightarrow^* is included in \leq_L .

Let m and n be two elements of \mathbf{M}_L such that $m \rightarrow^* n$. By definition, there are m_1, m_2, \dots, m_p p elements of \mathbf{M}_L such that $m \rightarrow m_1 \rightarrow m_2 \rightarrow \dots \rightarrow m_p \rightarrow n$, where p is a natural number. We then have $u_0, u_1, u'_1, u_2, u'_2, \dots, u_p, u'_p$, and u_{p+1} such that $m = h(u_0)$, $m_1 = h(u_1) = h(u'_1)$, $m_2 = h(u_2) = h(u'_2)$, \dots , $m_p = h(u_p) = h(u'_p)$ and $n = h(u_{p+1})$ and $u_0 \leq_{\mathcal{P}(\Sigma)^*} u_1$, $u'_1 \leq_{\mathcal{P}(\Sigma)^*} u_2$, $u'_2 \leq_{\mathcal{P}(\Sigma)^*} u_3$, \dots , $u'_p \leq_{\mathcal{P}(\Sigma)^*} u_{p+1}$.

Now let x and y be two words (they constitute a context). By monotonicity of L , if xu_0y belongs to L , then xu_1y belongs to L . Then, since $h(u_1) = h(u'_1)$, if xu_1y belongs to L , then so does xu'_1y . We immediately deduce that if xu_0y belongs to L , then so does $xu_{p+1}y$. This proves that \rightarrow^* is included in \leq_L .

So \rightarrow^* is an order, which we note $\leq_{\mathbf{M}_L}$.

Let us check its compatibility with the operation \cdot of the monoid. Let m, m', n and n' be elements of \mathbf{M}_L such that $m \leq_{\mathbf{M}_L} n$ and $m' \leq_{\mathbf{M}_L} n'$.

First, let us assume $m \rightarrow n$ and $m' \rightarrow n'$. We then have u, u', v and v' representing m, m', n and n' respectively, such that $u \leq_{\mathcal{P}(\Sigma)^*} v$ and $u' \leq_{\mathcal{P}(\Sigma)^*} v'$. So we have $uv \leq_{\mathcal{P}(\Sigma)^*} u'v'$ and thus, $mn \leq_{\mathbf{M}_L} m'n'$. Now, if we only have $m \rightarrow^* n$ and $m' \rightarrow^* n'$, then we have finite sequences $(m_i)_{i=1}^p$ and $(m'_i)_{i=1}^p$, which we can assume to be of the same length p by reflexivity of \rightarrow , such that $m \rightarrow m_1 \rightarrow \dots \rightarrow m_p \rightarrow n$ and $m' \rightarrow m'_1 \rightarrow \dots \rightarrow m'_p \rightarrow n'$. So we have $m \cdot m' \leq_{\mathbf{M}_L} m_1 \cdot m'_1$, but also $m_1 \cdot m'_1 \leq_{\mathbf{M}_L} m_2 \cdot m'_2, \dots, m_p \cdot m'_p \leq_{\mathbf{M}_L} n \cdot n'$. We then obtain the inequality $mn \leq_{\mathbf{M}_L} m'n'$ by transitivity.

Finally, it is clear that if $u \leq_{\mathcal{P}(\Sigma)^*} v$ then $h(u) \leq_{\mathbf{M}_L} h(v)$.

The relationship $\leq_{\mathbf{M}_L}$ therefore satisfies the constraints imposed. ◀

B.3 Proof of Proposition 32

Proof. First, in the algorithm from the Proposition 31, at any given time, we only need to code two letters from $\mathcal{P}(\Sigma)$ 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 F , all operations can be done in LOGSPACE. Thus, the algorithm from the Proposition 31 is in LOGSPACE.

To decide whether a DFA \mathcal{B} describes a monotone language, we can compute the NFA \mathcal{B}^\dagger 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}^\dagger describes the monotone closure of the language recognised by \mathcal{B} . Then, \mathcal{B} recognises a monotone language if and only if there is not path from an initial to a final state in the product automaton $\overline{\mathcal{B}} \times \mathcal{B}^\dagger$, 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. Thus, the monotonicity problem is NL – complete when the input is a DFA. ◀

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

Let us give a formula for the counter-example from Proposition 34.

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

23:20 Positive and monotone fragments of FO and LTL

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 $(abc)^\dagger$.

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

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

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 $\bigwedge_{d \in \Sigma} \neg d(x, y)$ 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:

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

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 ψ_a for the language of Proposition 34.

D Games

Erhenfeucht-Fraïssé games and their variants are traditionally used to prove negative expressivity results of FO fragments. This is why we were interested in Erhenfeucht-Fraïssé games matching fragments of FO^+ . Although we did not manage to use them in the present work, we include here a variant that could be suited for proving FO^{2+} inexpressibility results.

► **Definition 53.** We note $\text{EF}_k^{n+}[\mathfrak{B}](u_0, u_1)$, the Ehrenfeucht-Fraïssé game associated with $\text{FO}^{n+}[\mathfrak{B}]$ at k turns on the pair of words (u_0, u_1) . When there is no ambiguity, we simply note $\text{EF}_k^{n+}(u_0, u_1)$. In $\text{EF}_k^{n+}(u_0, u_1)$, two players, Spoiler and Duplicator, play against each other on the word pair (u_0, u_1) in a finite number k of rounds. Spoiler and Duplicator will use tokens numbered $1, 2, \dots, n$ to play on the positions of the words u_0 and u_1 .

On each turn, Spoiler begins. He chooses δ from $\{0, 1\}$ and i from $\llbracket 1, n \rrbracket$ and moves (or places, if it has not already been placed) the i numbered token onto a position of the word u_δ . Duplicator must then do the same on the word $u_{1-\delta}$ with the constraint of respecting binary predicates induced by the placement of the tokens, and only in one direction for unary predicates. More precisely, if ν_0 and ν_1 are the valuations that to each token (considered here as variables) associates the position where it is placed in u_0 and u_1 respectively, then

- for any binary predicate $\mathfrak{b}(x, y)$, $(u_0, \nu_0) \models \mathfrak{b}(x, y)$ if and only if $(u_1, \nu_1) \models \mathfrak{b}(x, y)$,
- for any unary predicate $a(x)$ in Σ , if $(u_0, \nu_0) \models a(x)$ then $(u_1, \nu_1) \models a(x)$.

If Duplicator cannot meet the constraint, he loses and Spoiler wins.

In particular, for any $i \in \llbracket 1, n \rrbracket$, if the letter s_0 indicated by the token i on the word u_0 is not included in the letter s_1 indicated by the token i on the word u_1 , then Spoiler wins.

If after k rounds, Spoiler has not won, then Duplicator is declared the winner.

► **Theorem 54.** *Let L be a language and n a natural number. The language L is definable by a formula of $\text{FO}^{n+}[\mathfrak{B}]$ if and only if there exists a natural number k such that, for any pair of words (u_0, u_1) where u_0 belongs to L but u_1 does not, Spoiler has a winning strategy in $\text{EF}_k^{n+}[\mathfrak{B}](u_0, u_1)$.*

Proof. We generalise the proof from [12, Theorem 5.7], which treats the case of FO^+ , using a classical construction for FO with a bounded number of variables.

Let n be a natural number. Let us introduce the concept of initial configuration. For two words u_0 and u_1 of lengths l_0 and l_1 respectively, and two functions of 0, 1, 2, ..., or n variables among x_1, \dots, x_n, ν_0 and ν_1 with values in $\llbracket 0, l_0 - 1 \rrbracket$ and $\llbracket 0, l_1 - 1 \rrbracket$ respectively, the game $\text{EF}_k^{n+}[\mathfrak{B}](u_0, u_1)$ has initial configuration (ν_0, ν_1) if token i is placed in position $\nu_0(x_i)$ on word u_0 , when $\nu_0(x_i)$ is defined, for any integer i from $\llbracket 1, n \rrbracket$, and similarly with u_1 for the valuation ν_1 .

We then claim that for any natural number k and any formula φ of $\text{FO}^{n+}[\mathfrak{B}]$ (possibly with free variables) of quantification rank at most k , and for all models (u_0, ν_0) and (u_1, ν_1) , Duplicator wins the game $\text{EF}_k^{n+}[\mathfrak{B}](u_0, u_1)$ with initial configuration (ν_0, ν_1) , if and only if:

$$u_0, \nu_0 \models \varphi \implies u_1, \nu_1 \models \varphi.$$

Indeed, starting from the induction from the article [12], we have to adapt the base case to the set of binary predicates \mathfrak{B} considered. The proof is then similar: each element of \mathfrak{B} can impose a constraint in $\text{FO}^{n+}[\mathfrak{B}]$ which is reflected in the constraint on the positions of the tokens. Then, in the induction, we need to modify the valuation update. Indeed, as the number of variables (and therefore of tokens) is limited to n , when a variable x already in use is encountered, we do not need to add a variable to the valuation ν constructed, but modify the value taken by ν in x , to construct a new valuation ν' .

◀