

# 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  $\text{FO}^+$  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:  $\text{FO}^+$  is equivalent to  $\text{LTL}^+$ , and its two-variable fragment  $\text{FO}^{2+}$  with (resp. without) successor available is equivalent to  $\text{UTL}^+$  with (resp. without) the “next” operator  $X$  available. This shows that despite previous negative results, the class of  $\text{FO}^+$ -definable languages exhibits some form of robustness. We then exhibit an example of an  $\text{FO}$ -definable monotone language on one predicate, that is not  $\text{FO}^+$ -definable, refining the example from [12] with 3 predicates. Moreover, we show that such a counter-example cannot be  $\text{FO}^2$ -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 ReCiPro

## 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  $\mu X$  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  $\mu\text{MALL}$  [2], or in the  $\mu$ -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 ( $\text{FO}^+$  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. 23:1–23:15

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  $\text{FO}^+$  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  $\text{FO}^2$  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  $\text{FO}^2$ -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  $\text{FO}^+$ ,  $\text{FO}^{3+}$  and  $\text{LTL}^+$  are equivalent in Theorem 20. We prove that the fragment  $\text{FO}^{2+}$  with (resp. without) successor predicate is equivalent to  $\text{UTL}^+$  with (resp. without) X and Y operators available in Theorem 26 (resp. Corollary 27).

In Section 4, we give a characterisation of monotonicity using monoids (Lemma 28) 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  $\text{FO}^+$ ). Indeed, we show that the counter-example can be adapted to  $\text{FO}^2$  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 characterization of  $\text{FO}^2[<]$  from Thérien and Wilke [19] stating that  $\text{FO}^2[<]$  is equivalent to  $\Sigma_2 \cap \Pi_2$  where  $\Sigma_2$  and  $\Pi_2$  are fragments of FO with bounded quantifier alternation. We show that  $\text{FO}^2$ -monotone is characterized by  $\Sigma_2^+ \cap \Pi_2^+$ .

At last, we show that no counter-example for FO can be found in  $\text{FO}^2$  (without successor available) in Corollary 41. We conclude by leaving open the problem of expressive equivalence between  $\text{FO}^{2+}$  and  $\text{FO}^2$ -monotone, as well as decidability of membership in  $\text{FO}^{2+}$  for regular languages (see Conjecture 42).

## 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,  $=, \neq, \leq, <, \text{succ}$  and  $\text{nsucc}$ , 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  $\nu$  from the free (non-quantified) variables of a formula  $\varphi$  to the positions of  $u$  satisfies  $\varphi$  if  $u$  satisfies the closed formula obtained from  $\varphi$  by replacing each free variable with its image by  $\nu$ .

► **Definition 2.** Let  $\varphi$ , a formula with  $n$  free variables,  $x_1, \dots, x_n$ , and  $u$  a word. Let  $\nu$  be a function of  $\{x_1, \dots, x_n\}$  in  $\llbracket 0, |u| - 1 \rrbracket$ . We say that  $(u, \nu)$  satisfies  $\varphi$ , and we define  $u, \nu \models \varphi$  by induction on  $\varphi$  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),

- $u, \nu \models \varphi \wedge \psi$  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  $\varphi$  because two of its letters contain an  $a$ , but  $\{a, b, c\}\{b\}\emptyset$  does satisfy  $\varphi$ .

► **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  $\text{FO}^+$ .

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

► **Definition 5.** The grammar of  $\text{FO}^+$  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]).**  $\text{FO}^+$  formulas are monotone in unary predicates, i.e. if a model  $(u, \nu)$  satisfies a formula  $\varphi$  of  $\text{FO}^+$ , and  $u \leq_{\mathcal{P}(\Sigma)^*} v$ , then  $(v, \nu)$  satisfies  $\varphi$ .

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  $\text{FO}^3$  is the subset of FO formulas using only three different variables, which can be reused. We also define  $\text{FO}^{3+}$  for formulas with three variable and without negation. Similarly, we define  $\text{FO}^2$  and  $\text{FO}^{2+}$  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  $\text{FO}^3$  formula, and even an  $\text{FO}^{3+}$  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  $\text{FO}^2$ .

## 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  $\varphi$ . The construction  $\varphi U\psi$  ( $\varphi$  until  $\psi$ ) indicates that the formula  $\psi$  must be verified at a given point in time and that  $\varphi$  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  $\varphi$  be an LTL formula, and  $u = u_0...u_{m-1}$  be a word. We say that  $u$  satisfies  $\varphi$  and define  $u \models \varphi$  by induction on  $\varphi$  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  $u_1...u_{m-1} \models \varphi$ ,
- $u \models \varphi U\psi$  if there is  $i \in \llbracket 0, m-1 \rrbracket$  such that  $u_i...u_{m-1} \models \psi$  and for all  $j \in \llbracket 0, i-1 \rrbracket$ ,  $u_j...u_{m-1} \models \varphi$ ,
- $u \models \varphi R\psi$  if  $u \models (\psi U(\psi \wedge \varphi))$  or for all  $i \in \llbracket 0, m-1 \rrbracket$  we have  $u_i...u_{m-1} \models \psi$ ,
- $u \models \neg\varphi$  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  $(\varphi, \psi)$  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:*

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

*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  $\varphi$  will be satisfied at least once in the future ( $F$  as Future), and that  $\varphi$  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  $\varphi$  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  $\varphi$ :

- $u, i \models \top$  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  $\varphi$  and  $\psi$  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  $\varphi$  is an FO formula with one free variable (having models of the form  $(u, i)$ ) and  $\psi$  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 $\text{FO}^+$

We aim at proving the following theorem, lifting classical results from  $\text{FO}$  to  $\text{FO}^+$ :

► **Theorem 20.** *The logics  $\text{FO}^+$ ,  $\text{LTL}^+$  and  $\text{FO}^{3+}$  describe the same languages.*

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

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

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  $\varphi$  of  $\text{FO}^+$  defined inductively by:*

- *if  $\varphi$  contains no quantifier then  $\text{qr}(\varphi) = 0$ ,*
- *if  $\varphi$  is of the form  $\exists x, \psi$  or  $\forall x, \psi$  then  $\text{qr}(\varphi) = \text{qr}(\psi) + 1$ ,*
- *if  $\varphi$  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  $\varphi$  be a  $\text{TL}^+$  formula with possible nesting of past and future operators. There is a separated formula of  $\text{TL}^+$  that is equivalent to  $\varphi$ .*

**Proof.** TOPROVE 0 ◀

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

► **Lemma 25.** *The set of languages described by  $\text{FO}^+$  is included in the set of languages recognised by  $\text{LTL}^+$ .*

**Proof.** TOPROVE 1 ◀

We can now turn to the proof of Theorem 20.

**Proof.** TOPROVE 2 ◀

### 3.2 Equivalences in fragments of $\text{FO}^+$

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

**Proof.** TOPROVE 3 ◀

► **Corollary 27.** *The logic  $\text{FO}^{2+}[\mathfrak{B}_{<}]$  is equivalent to  $\text{UTL}^+[\text{P}, \text{F}, \text{H}, \text{G}]$ .*

**Proof.** TOPROVE 4 ◀

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 28.** *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 29.** *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.** TOPROVE 5 ◀

### 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 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 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  $O(|\mathcal{P}(\Sigma)|^2 |\mathbf{M}|^2)$ .*

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 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.*

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  $\text{FO}^+$ . 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  $\text{FO}^2$ . 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  $\text{FO}^+$ -definable uses three predicates  $a$ ,  $b$  and  $c$  and is as follows:

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

It uses the following words to find a strategy for Duplicator in  $\text{EF}_k^+$ :

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

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]. 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  $\text{LTL}^+$  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 32.** [11] *For any unary predicate  $a$  (not only predicates from  $\Sigma$  but also Boolean combination of them),  $a$  also designates a binary predicate, called between predicate, such that for any word  $u$  and any valuation  $\nu$ ,  $(u, \nu) \models a(x, y)$  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  $\mathbf{be}$  the set of between predicates and  $\mathbf{be}^+$  the set of between predicates associated to the set of positive unary predicates.*

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

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

**Proof.** TOPROVE 6



### 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  $\Sigma$  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  $\text{FO}^+$ .*

**Proof.** TOPROVE 7 ◀

## 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  $\Sigma_2$ -definable and  $\Pi_2$ -definable where  $\Sigma_2$  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  $\varphi$  does not have any quantifier and  $\Pi_2$ -formulas are negations of  $\Sigma_2$ -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  $\Sigma_2$  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  $\text{FO}^{2+}$  captures the set of monotone  $\text{FO}^2$ -definable languages, we can state the following theorem:

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

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

► **Definition 36.** *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 37.** 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 38.** *The set  $\Sigma_2^+$  captures the set of monotone  $\Sigma_2$ -definable languages.*

**Proof.** TOPROVE 8 ◀

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

► **Lemma 39.** *The set  $\Sigma_2^-$  ( $\Sigma_2$ -formulas with negations on all predicates) captures the set of downwards closed  $\Sigma_2$ -definable languages.*

We can now deduce the following lemma:

► **Lemma 40.** *The set  $\Pi_2^+$  captures the set of monotone  $\Pi_2$ -definable languages.*

**Proof.** TOPROVE 9 ◀

Finally, we can prove Theorem 35:

**Proof.** TOPROVE 10 ◀

This last result shows how close to capture monotone  $\text{FO}^2$ -definable languages  $\text{FO}^{2+}$  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  $\text{FO}^+$  cannot be in  $\text{FO}^2[\mathfrak{B}_<]$  as stated in the following corollary:

► **Corollary 41.** *Any monotone language described by an  $\text{FO}^2[\mathfrak{B}_<]$  formula is also described by an  $\text{FO}^+$  formula.*

If the monotone closure  $L^\uparrow$  of a language  $L$  described by a formula of  $\text{FO}^2[\mathfrak{B}_<]$  is in  $\text{FO}^+$ , 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  $\text{FO}^2$  can stand either for  $\text{FO}^2[\mathfrak{B}_<]$  or for  $\text{FO}^2[\mathfrak{B}_0]$

► **Conjecture 42.**

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

Since we can decide whether a language is definable in  $\text{FO}^2$  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'Ávila 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. TOPROVE 11 ◀

## B

 Monoids

### B.1 Algebraic definitions

► **Definition 43.** A semigroup is a pair  $(S, \cdot)$  where  $\cdot$  is an associative internal composition law on the non-empty set  $S$ .

► **Remark 44.** We allow ourselves the abuse of language which consists in speaking of the semigroup  $S$  instead of the semigroup  $(S, \cdot)$ .

► **Definition 45.** A monoid is a pair  $(M, \cdot)$  which is a semigroup, and which has a neutral element noted  $1_M$  (or simply  $1$  when there is no ambiguity), i.e. which verifies:

$$\forall m \in M, 1 \cdot m = m \cdot 1 = m.$$

► **Definition 46.** 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 47.** 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 48.** 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 49.** 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 50.** Let  $L$  be a regular language. We define the syntactic monoid of  $L$  as  $\mathbf{M}_L = L / \sim_L$ .

► **Remark 51.** 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 28

Proof. TOPROVE 12 ◀

## B.3 Proof of Proposition 31

Proof. TOPROVE 13 ◀

## C

 An  $\text{FO}^2[\mathfrak{B}_0 \cup \mathfrak{be}]$ -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  $\text{FO}^2[\mathfrak{B}_{<} \cup \mathfrak{be}]$ , so results from [11] about the fragment  $\text{FO}^2[<, \mathfrak{be}]$  apply to  $\text{FO}^2[\mathfrak{B}_0 \cup \mathfrak{be}]$  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  $(abc)^\dagger$ .

## 23:14 Positive and monotone fragments of FO and LTL

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 for the language of Proposition 33.

## 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  $\text{FO}^+$ . Although we did not manage to use them in the present work, we include here a variant that could be suited for proving  $\text{FO}^{2+}$  inexpressibility results.

► **Definition 52.** 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  $\delta$  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  $u_\delta$ . 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  $\nu_0$  and  $\nu_1$  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  $\Sigma$ , 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 53.** *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.** TOPROVE 14

