Synthesis for prefix first-order logic on data words

Synthesis for Prefix First-Order Logic on Data Words

Julien Grange[1] and Mathieu Lehaut[2](✉)

[1] Univ Paris Est Creteil, LACL, 94010 Creteil, France
julien.grange@lacl.fr

[2] University of Gothenburg, Gothenburg, Sweden
lehaut@chalmers.se

Abstract. We study the reactive synthesis problem for distributed systems with an unbounded number of participants interacting with an uncontrollable environment. Executions of those systems are modeled by data words, and specifications are given as first-order logic formulas from a fragment we call prefix first-order logic that implements a limited kind of order. We show that this logic has nice properties that enable us to prove decidability of the synthesis problem.

1 Introduction

Distributed algorithms have been increasingly more common in recent years, and can be found in a wide range of domains such as distributed computing, swarm robotics, multi-agent systems, and communication protocols, among others. Those algorithms are often more complex than single-process algorithms due to the interplay between the different processes involved in the computations. Another complication arises in the fact that some algorithms must be designed for distributed systems where the number of participants is not known in advance, which is often the case in applications where agents can come and leave at a moment's notice as is the case, for instance, in ad-hoc networks. Those properties make it hard for programmers to design such algorithms without any mistake, thus justifying the development of formal methods for their verification.

In this paper, we focus on the *reactive synthesis* problem. This problem involves systems that interact with an uncontrollable environment, with the system outputting some values depending on the inputs that are given by the environment. Given a specification stating what are the allowed behaviors of the whole system, the goal is to automatically build a program that would satisfy the specification. This problem dates all the way back to Church [4], whose original statement focused on sequential systems, and was solved in this context by Büchi and Landweber [3]. They reformulated this problem as a two-player synthesis game between the System and an adversarial Environment alternatively choosing actions from a finite alphabet. The goal of System is for the resulting sequence

Supported by the ERC Consolidator grant D-SynMA (No. 772459).

V. Castiglioni and A. Francalanza (Eds.): FORTE 2024, LNCS 14678, pp. 82–98, 2024.
https://doi.org/10.1007/978-3-031-62645-6_5

of actions to satisfy the specification, while Environment wants to falsify it. A winning strategy for System, if it exists, can then be seen as a program ensuring that the specification is always met.

At the cost of having one copy of the alphabet for each participant, one can adapt this setting to distributed systems where the number of participants is fixed. However, a finite alphabet is too restrictive to handle systems with an unbounded number of participants such as those we described earlier. Indeed, with a finite amount of letters in the alphabet, one cannot distinguish every possible participant when there can be any number of those, potentially more than the number of letters. At most, one can deal by grouping participants together in a finite number of classes and consider all that belong to the same class to be equivalent. This however restrict the behaviors of the system and what one could specify over the system. It is therefore worth extending this problem to infinite action alphabets. To that end, we turn to *data words*, as introduced by Bojanczyk et al. [2]. A data word is a sequence of pairs consisting of an action from a finite alphabet and a datum from an infinite alphabet. In our context the datum represents the identity of the process doing the action, meaning that a data word is seen as an execution of the system describing sequentially which actions have been taken by each process. In the corresponding synthesis game, the two players alternatively choose both an action and a process, and the resulting play is a data word.

The last ingredient needed to properly define the synthesis problem is the choice of a formalism in which specifications are written. Unfortunately, there is no strong candidate for a "standard" way of representing sets of data words, in contrast to finite automata in the case of simple words. Many formalisms have been proposed so far, but all of them lack either good closure properties (union, complementation, etc.), have bad complexity for some basic decision procedures (membership, ...), are lacking in expressivity power or do not have a good equivalent automata $\Leftrightarrow$ logic characterization. Let us cite nonetheless register automata who were considered first by Kaminsky and Francez [11] but have seen different extensions over time, pebble automata by Neven et al. [13] and data automata [2]. On the logical side, several formalisms have been proposed, such as a variant of first-order logic [2], Freeze LTL [6] and the logic of repeating values [5]. We refer the reader to Ahmet Kara's dissertation for a more comprehensive survey [12]. Most of the previous works study the membership problem (in the case of automata) and the satisfiability problem (for logics), which are useful for model-checking applications but not enough in the synthesis context, as they lack the adversarial environment factor that is central to the problem. A few attempts have been made in this direction, notably for the logic of repeating values by Figueira and Praveen [8] and for register automata by Exibard et al. [7].

We follow previous work [1,9] and focus on synthesis for first-order logic extended to data words. In this extension, we add a predicate $\sim$ to the logic such that $x \sim y$ is true when two positions x and y of a data word share the same data value, i.e. when both actions have been made by the same process. Moreover, we partition the set of processes into System processes and Environment processes, and restrict each player to their own processes. The reason for this is two-fold:

first, when processes are shared, the synthesis problem has been shown to be undecidable even for the simplest logic possible $\mathrm{FO}^2[\sim]$, where only two variable names are allowed. Second, inputs and outputs are usually physically located in different components of the system, such as sensors being disjoint from motors in a drone; it thus makes sense to see them as different processes.

As always, there is a trade-off between expressivity of the logic and decidability of its synthesis problem. The well-known LTL undecidability result from Pnueli and Rosner [14] occurs because the logic allows specifying properties that the system is too weak to satisfy. We must therefore find a balance between making the logic expressive enough to be useful, while limiting its power so that it cannot specify properties the system is not expected to be able to satisfy in the first place. Our previous results ([9, Theorems 3 and 4]) have shown that adding any kind of order on the positions of the data word, such as either the immediate successor predicate or the happens-before predicate, makes the synthesis problem undecidable. While those two predicates are fine from a centralized point of view that sees the whole system as a sequential machine, they are not that well suited for real-life systems. Indeed, it is ambitious to expect each process to know everything that happened on other processes in the exact order those actions happened; this would require every process to be informed instantly after every action happening in the system. What is more reasonable is simply to expect a process to know the order of occurrence of its own actions only, without knowing whether those actions happened before or after actions made by other processes.

This leads us to introduce a new operator $\lesssim$, for which $x \lesssim y$ if $x \sim y$ and x occurs before y in the data word. We call $\mathrm{FO}[\lesssim]$ the extension of FO that includes only this new predicate. It is strictly more expressive than $\mathrm{FO}[\sim]$, as the $\sim$ predicate can easily be simulated by $\lesssim$. Similar to $\mathrm{FO}[\sim]$, we can study separately the class of each process. Whereas $\mathrm{FO}[\sim]$ could only count how many times each action happened (up to some threshold), we can now express anything that $\mathrm{FO}[<]$ can express on (simple) words. Unfortunately, the decidability of the synthesis problem for $\mathrm{FO}[\lesssim]$ remains open. In this paper, we show a positive result for a restriction of this logic that we call *prefix first-order logic*, denoted by $\mathrm{FO}^{\mathrm{PREF}}[\lesssim]$. In this restriction, the first variable quantified for each process is called a bounding variable, and every subsequently defined variable belonging to the same process must occur before the bounding variable. In other words, the first variable for each class pins down a finite prefix of the class and throws away the rest of the class; the rest of the variables can only talk about positions that fall inside this prefix. This restriction allows us to obtain good properties that we leverage to show decidability of the synthesis problem.

This paper is organized as follows. We first define prefix first-order logic $\mathrm{FO}^{\mathrm{PREF}}[\lesssim]$ in Sect. 2, and the synthesis problem and its equivalent games in Sect. 3. We then show the synthesis problem for $\mathrm{FO}^{\mathrm{PREF}}[\lesssim]$ to be decidable of in Sect. 4, before concluding in Sect. 5. Omitted proofs can be found in the full version of this paper [10].

2 Prefix First-Order Logic

2.1 Data Words and Preliminaries

Fix two disjoint alphabets Σ_S and Σ_E, which are respectively the System and Environment *actions*. Let $\Sigma = \Sigma_S \uplus \Sigma_E$ denote their union. In the following, we write ε for the empty word, and $u \cdot v$ for the concatenation of the words u and v. Let $\mathbb{P}_\mathrm{S}$ and $\mathbb{P}_\mathrm{E}$ be two disjoint sets of System and Environment *processes*, respectively.

A *data word* is a (finite or infinite) sequence $\mathfrak{w} = (a_0, p_0)(a_1, p_1)\ldots$ of pairs $(a_i, p_i) \in (\Sigma_S \times \mathbb{P}_\mathrm{S}) \cup (\Sigma_E \times \mathbb{P}_\mathrm{E})$. A pair (a, p) indicates that action a has been taken by process p. The *class* of a process p is the word $w_p = a_{k_0} a_{k_1} \cdots \in \Sigma^\star$ where $(k_i)_i$ is exactly the sequence of positions in $\mathfrak{w}$ where actions are made by p. We see data words as logical structures (and will conveniently identify a data word with its associated structure) over the vocabulary consisting of two binary predicates $\sim$, $<$, and two unary predicates $\mathbb{P}_\mathrm{S}$, $\mathbb{P}_\mathrm{E}$ as well as one additional unary predicate for each letter of Σ. The universe of a data word has one element for each process (called *process elements* – these are needed in order to quantify over processes that have not played any action, and will drastically increase the expressive power of $\mathrm{FO}^{\mathrm{PREF}}[\lesssim]$), and one element for each position in the data word. Predicate $\mathbb{P}_\mathrm{S}$ (resp. $\mathbb{P}_\mathrm{E}$) is interpreted as the set of all System (resp. Environment) process elements. For $a \in \Sigma$, the predicate a holds on every position which correspond to action a. Predicate $<$ is interpreted as the linear order on the set of positions corresponding to their order in the data word (and is thus not defined on process elements), and $\sim$ is interpreted as an equivalence relation which has one equivalence class for every process, encompassing both its process element and all positions of its class. It will be convenient to use $\mathbb{P}(x)$ as an alias for "$\mathbb{P}_\mathrm{S}(x) \vee \mathbb{P}_\mathrm{E}(x)$" and $x \lesssim y$ as an alias for "$x \sim y \wedge x < y$".

Let $\mathrm{DW}_\Sigma^\mathbb{P}$ denote the set of all data words over actions Σ and processes $\mathbb{P}$. We write $\mathfrak{w}[i \ldots j]$ for the factor of $\mathfrak{w}$ occurring between positions i and j (both included), and $\mathfrak{w}[i \ldots]$ for the suffix starting at position i; we extend both notations to regular words as well.

2.2 Prefix First-Order Logic on Data Words

We define *prefix first-order logic on data words* $\mathrm{FO}^{\mathrm{PREF}}[\lesssim]$ by induction on its formulas. We write $\varphi(X^{\mathrm{proc}}; X^{\mathrm{bnd}}; X^{\mathrm{pref}})$ to mean that the free variables of φ belong to the pairwise disjoint union X of the three sets X^{proc} (the *process variables*), X^{bnd} (the *bounding variables*) and X^{pref} (the *prefix variables*).

$$\begin{array}{ll}
\varphi(X^{\text{proc}};\ X^{\text{bnd}};X^{\text{pref}}) ::= & \\
\quad x = y & (x, y \in X) \\
\mid \mathbb{P}_{\text{S}}(x) \quad \mid \mathbb{P}_{\text{E}}(x) & (x \in X^{\text{proc}}) \\
\mid a(x) & (x \in X^{\text{bnd}} \cup X^{\text{pref}}, a \in \Sigma) \\
\mid x \lesssim y & (x, y \in X^{\text{pref}}) \\
\mid x \sim y & ((x, y) \in X^{\text{proc}} \times X) \\
\mid \varphi(X^{\text{proc}};X^{\text{bnd}};X^{\text{pref}}) \wedge \varphi(X^{\text{proc}};X^{\text{bnd}};X^{\text{pref}}) & \\
\mid \neg\varphi(X^{\text{proc}};X^{\text{bnd}};X^{\text{pref}}) & \\
\mid \exists x, \mathbb{P}(x) \ \wedge\ \varphi(X^{\text{proc}} \cup \{x\};X^{\text{bnd}};X^{\text{pref}}) & (x \notin X) \\
\mid \exists x, \neg\mathbb{P}(x) \ \wedge\ \Big(\bigwedge_{y \in X^{\text{bnd}}} x \not\sim y\Big) \wedge \varphi(X^{\text{proc}};X^{\text{bnd}} \cup \{x\};X^{\text{pref}}) & (x \notin X) \\
\mid \exists x,\ x \lesssim y \wedge \varphi(X^{\text{proc}};X^{\text{bnd}};X^{\text{pref}} \cup \{x\}) & (x \notin X, y \in X^{\text{bnd}})
\end{array}$$

with $X = X^{\text{proc}} \uplus X^{\text{bnd}} \uplus X^{\text{pref}}$. The intuition is as follows.

We allow one to quantify over the process elements with process variables. Bounding and prefix variables are used to quantify over the elements of the process classes; when quantifying (existentially or universally) over an element of a process class (that is, an actual position of the data word), one must first use a bounding variable. From then on, only prefix variables can be used on this process class, which can only quantify earlier positions in the class (i.e. positions which are $\lesssim$ to the bounding position). Note that one can still quantify over other classes, using new bounding variables.

The semantics is defined as usual. As always, the *quantifier depth* of a formula is the maximal number of nested quantifiers (without regard to whether they quantify process, bounding or prefix variables).

By construction, $\text{FO}^{\text{PREF}}[\lesssim]$ is a fragment of $\text{FO}[\sim, <]$ first-order logic with $\sim$ and $<$). Example 1 below illustrates that $\text{FO}^{\text{PREF}}[\lesssim]$ encompasses $\text{FO}[\sim]$.

Example 1. Let us fix the alphabets $\Sigma_E := \{a_E\}$ and $\Sigma_S := \{a_S\}$. There exists an $\text{FO}^{\text{PREF}}[\lesssim]$ formula $\varphi_{1a_E \leftrightarrow 1a_S}$ of quantifier depth 3 stating that there exists a process with exactly one a_E if and only if there exists a process with exactly one a_S. Indeed, the existence of a process with exactly one a_E (and similarly for a_S) can be stated as

$$\begin{aligned}
\exists x,\ \mathbb{P}(x) \quad \wedge \quad & \big(\exists y,\ \neg\mathbb{P}(y)\ \wedge\ y \sim x\ \wedge\ a_E(y)\big) \\
\wedge \quad & \neg\big(\exists y,\ \neg\mathbb{P}(y)\ \wedge\ y \sim x\ \wedge\ a_E(y)\ \wedge\ \exists z,\ z \lesssim y \wedge a_E(z)\big)\,.
\end{aligned}$$

Here, x is a process variable, both occurrences of y are bounding variables and z is a prefix variable.

If a (finite or infinite) word is seen as a data word with exactly one data class, then $\text{FO}^{\text{PREF}}[\lesssim]$ can in particular be seen a logic on words: it is equivalent to the restriction of first-order logic on words where

– the formula must start with a universal or existential quantification on the variable $\overline{x}$,

– after that, each new quantification must be of the form $\exists x < \overline{x}$ or $\forall x < \overline{x}$.

We will refer to this logic on words as $\mathrm{FO}^{\mathrm{PREF}}$.

Example 2. In order to get a better understanding of the expressive power of $\mathrm{FO}^{\mathrm{PREF}}$ and its limitation, let us consider a finite alphabet containing, among others, the two symbols `open` and `close`.
The property stating that every occurrence of `close` must be preceded by an occurrence of `open` can be formulated as follows in $\mathrm{FO}^{\mathrm{PREF}}$:

$$\forall \bar{x},\ \mathtt{close}(\bar{x}) \quad \rightarrow \quad \exists y,\ y < \bar{x}\ \wedge\ \mathtt{open}(y)\,.$$

In contrast, one cannot state in $\mathrm{FO}^{\mathrm{PREF}}$ that each occurrence of `open` is followed by an occurrence of `close`. Indeed, for a fixed quantifier depth, the two words $w = \mathtt{open} \cdot \mathtt{close} \cdot \mathtt{open} \cdot \ldots \cdot \mathtt{close}$ and $w' = w \cdot \mathtt{open}$ cannot be distinguished if w is long enough.

For any word w, we let $\langle w \rangle^k_{\mathrm{FO}^{\mathrm{PREF}}}$ denote its $\mathrm{FO}^{\mathrm{PREF}}$-type of depth k, i.e. the set of all sentences of $\mathrm{FO}^{\mathrm{PREF}}$ with quantifier depth at most k satisfied in w. Having fixed an alphabet, we denote by $\mathrm{Types}^k_{\mathrm{FO}^{\mathrm{PREF}}}$ the set of $\mathrm{FO}^{\mathrm{PREF}}$-types of depth k on words.

Lemma 3. *Let $k \in \mathbb{N}$ and let $\mathfrak{w}$ and $\widehat{\mathfrak{w}}$ be two finite or infinite data-words on the same alphabet Σ, such that for every $\tau \in \mathrm{Types}^k_{FO^{PREF}}$, the number of classes of type τ in $\mathfrak{w}$ and $\widehat{\mathfrak{w}}$ are either the same or both at least k. Then $\mathfrak{w}$ and $\widehat{\mathfrak{w}}$ agree on all $FO^{PREF}[\lesssim]$-sentences of quantifier depth at most k.*

As a direct corollary of Lemma 3, in order to decide whether a data word $\mathfrak{w}$ satisfies an $\mathrm{FO}^{\mathrm{PREF}}[\lesssim]$ formula of quantifier depth k, it is enough to know, for each k-type τ for $\mathrm{FO}^{\mathrm{PREF}}$, how many (up to k) classes of $\mathfrak{w}$ have type τ. We shall use this fact later in proofs, and refer to this abstraction of a data word as its *collection* of types.

2.3 Properties of $\mathrm{FO}^{\mathrm{PREF}}$ Types

Let us now try to understand the behavior of $\mathrm{FO}^{\mathrm{PREF}}$ on words. In the following, we fix an alphabet Σ and an integer k.

First, note that the equivalence relation "have the same $\mathrm{FO}^{\mathrm{PREF}}$-type" is not a congruence in the monoid of finite words. Indeed, one can convince themselves (or prove formally, using the Ehrenfeucht-Fraïssé games introduced in the full version of this paper) that for any $k \in \mathbb{N}$, the two words

$$u = ababa \cdots aba$$

and

$$v = ababa \cdots abab$$

have the same $\mathrm{FO}^{\mathrm{PREF}}$ k-type (as long as they are long enough with respect to k). However, $u \cdot a$ and $v \cdot a$ can be separated by the $\mathrm{FO}^{\mathrm{PREF}}$-sentence of quantifier depth 3 stating the existence of two consecutive a's.

Lemma 4. *Let u, v be words such that $\langle u\rangle^k_{FO^{PREF}} = \langle u \cdot v\rangle^k_{FO^{PREF}} = \tau$. Then for every prefix w of v, $\langle u \cdot w\rangle^k_{FO^{PREF}} = \tau$.*

This lemma has a straightforward consequence in the case of infinite words: for every infinite word u, there exists some $\tau \in \text{Types}^k_{\text{FO}^{\text{PREF}}}$ and an index $n \in \mathbb{N}$ such that for every $m \geq n$, $u[0 \dots m]$ has type τ. Indeed, $\text{Types}^k_{\text{FO}^{\text{PREF}}}$ is finite, thus there must be some type appearing infinitely often in the prefixes of u. Lemma 4 ensures that such a type is unique. We refer to this type as the *stationary type* of u.

Next, we prove that the stationary type of an infinite word is none other than its own type:

Lemma 5. *Let u be an infinite word with stationary type $\tau \in \text{Types}^k_{FO^{PREF}}$. Then $\langle u\rangle^k_{FO^{PREF}} = \tau$.*

Combining those results we get the following:

Corollary 6. *Let $k \in \mathbb{N}$, u be an infinite word, and $\tau = \langle u\rangle^k_{FO^{PREF}}$. Then there exists $n \in \mathbb{N}$ such that for all $m > n$, $u[0 \dots m]$ also has type τ.*

Let us now try to understand the structure of $\text{Types}^k_{\text{FO}^{\text{PREF}}}$. We consider the binary relation $\rightharpoonup_k$ defined on $\text{Types}^k_{\text{FO}^{\text{PREF}}}$ as follows: $\tau \rightharpoonup_k \tau'$ if and only if there exists two finite words u and v such that $\langle u\rangle^k_{\text{FO}^{\text{PREF}}} = \tau$ and $\langle uv\rangle^k_{\text{FO}^{\text{PREF}}} = \tau'$. We refer to $(\text{Types}^k_{\text{FO}^{\text{PREF}}}, \rightharpoonup_k)$ as the *graph of* FO^{PREF} *-types* of depth k.

The following lemmas break down its properties. First, the choice of u and v above does not really matter:

Lemma 7. *Let $k \geq 1$, let τ and τ' be such that $\tau \rightharpoonup_k \tau'$ and let w be a finite word such that $\langle w\rangle^k_{FO^{PREF}} = \tau$. There exists some finite word w' such that $\langle ww'\rangle^k_{FO^{PREF}} = \tau'$.*

Second, $\rightharpoonup_k$ is an order:

Lemma 8. *The binary relation $\rightharpoonup_k$ is a partial order on $\text{Types}^k_{FO^{PREF}}$, with minimum $\langle \varepsilon\rangle^k_{FO^{PREF}}$.*

From those two lemmas, we conclude that $(\text{Types}^k_{\text{FO}^{\text{PREF}}}, \rightharpoonup_k)$ can be seen as a finite directed tree rooted in $\langle \varepsilon\rangle^k_{\text{FO}^{\text{PREF}}}$. This is illustrated in Fig. 1.

3 Synthesis and Token Games

In this section we define the standard synthesis game, and then give equivalent games that are more suitable for our purpose.

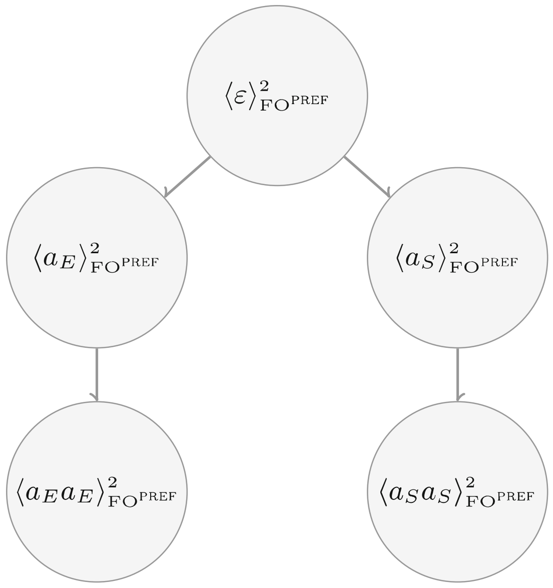
$\langle\varepsilon\rangle^2_{\mathrm{FO}^{\mathrm{PREF}}}$
$\langle a_E\rangle^2_{\mathrm{FO}^{\mathrm{PREF}}}$
$\langle a_S\rangle^2_{\mathrm{FO}^{\mathrm{PREF}}}$
$\langle a_E a_E\rangle^2_{\mathrm{FO}^{\mathrm{PREF}}}$
$\langle a_S a_S\rangle^2_{\mathrm{FO}^{\mathrm{PREF}}}$

Fig. 1. A partial representation of $\mathrm{Types}^2_{\mathrm{FO}^{\mathrm{PREF}}}$ for $\Sigma_E = \{a_E\}$ and $\Sigma_S = \{a_S\}$. Types of words containing both a_E and a_S have been omitted, as data classes cannot have such a type.

3.1 Standard Synthesis Game

Given a formula φ, the *standard synthesis game* is a game played between two players, System and Environment, who collaborate to create a data word. System's goal is to make the created data word satisfy φ, while Environment wants to falsify it. Formally, a *strategy* for System is a function $\mathcal{S} : \mathrm{DW}^{\mathbb{P}}_{\Sigma} \to (\Sigma_S \times \mathbb{P}_{\mathrm{S}}) \cup \{\varepsilon\}$ which given a data word created so far (the *history*) returns either an action and process to play on, or passes its turn on output ε. A data word $\mathfrak{w} = (a_0, p_0)(a_1, p_1)\ldots$ is *compatible* with $\mathcal{S}$ if for all i, $a_i \in \Sigma_S$ implies $\mathcal{S}(\mathfrak{w}[0\ldots i-1]) = (a_i, p_i)$. Furthermore, $\mathfrak{w}$ is *fair* with $\mathcal{S}$ if either $\mathfrak{w}$ is finite and $\mathcal{S}(\mathfrak{w}) = \varepsilon$, or $\mathcal{S}(\mathfrak{w}[0\ldots i]) \neq \varepsilon$ for infinitely many $i \in \mathbb{N}$ implies $a_i \in \Sigma_S$ for infinitely many $i \in \mathbb{N}$. Intuitively, a fair data word prevents the pathological case where System wants to do some action but Environment forever prevents it by continually playing its own actions instead. Strategy $\mathcal{S}$ is said to be *winning* if all compatible and fair data words satisfy φ. System wins a synthesis game if there exists a winning strategy for System.

The *existential synthesis problem* asks, for a given alphabet Σ and formula φ, whether there exists a set of processes $\mathbb{P} = \mathbb{P}_{\mathrm{S}} \uplus \mathbb{P}_{\mathrm{E}}$ such that System wins the corresponding synthesis game. In the case of $\mathrm{FO}^{\mathrm{PREF}}[\lesssim]$, since the logic can only compare process identities with respect to equality, it is easy to see that the actual sets $\mathbb{P}_{\mathrm{S}}$ and $\mathbb{P}_{\mathrm{E}}$ do not matter: only their cardinality does. With that in mind, we slightly reformulate the existential synthesis problem to ask whether there exists a pair $(n_S, n_E) \in \mathbb{N}^2$ such that System wins the synthesis game for all sets of processes $\mathbb{P}_{\mathrm{S}}$ and $\mathbb{P}_{\mathrm{E}}$ of respective size n_S and n_E. If (n_S, n_E) is such a pair, we say that System has a (n_S, n_E)-winning strategy for φ.

3.2 Symmetric Game

Notice that the standard synthesis game is asymmetric: while Environment can interrupt System at any point (provided that System gets the possibility to play infinitely often if they want to), System does not choose exactly the timing of their moves. In particular, System is never guaranteed to be able to play successive moves in the game. Let us define a variation of the game, which turns out to be equivalent with respect to the logic $\mathrm{FO}^{\mathrm{PREF}}[\lesssim]$, in which System can play arbitrarily many successive moves. This makes the game symmetric, and will make the following proofs simpler.

The *symmetric game* is defined in the same way as the standard game, with the following exceptions. Instead of a function from $\mathrm{DW}_{\Sigma}^{\mathbb{P}}$ to $(\Sigma_S \times \mathbb{P}_{\mathrm{S}}) \cup \{\varepsilon\}$, a strategy for System is now a function from $\mathrm{DW}_{\Sigma}^{\mathbb{P}}$ to $(\Sigma_S \times \mathbb{P}_{\mathrm{S}})^{\star} \cup \{\varepsilon\}$, i.e. System is allowed to play an arbitrary (but finite) amount of actions at once. Moreover, players strictly alternate, starting with Environment. The rest is defined as in the standard game. It is obvious that System has an easier time winning the symmetric game than the standard game. It turns out the symmetric game offers System no advantage when the relative positions of the classes are incomparable:

Lemma 9. *Let φ be a $FO^{PREF}[\lesssim]$-sentence, and $n_S, n_E \in \mathbb{N}$. System has an (n_S, n_E)-winning strategy for φ in the symmetric game if and only if System has an (n_S, n_E)-winning strategy for φ in the standard game.*

Note that when positions between classes can be compared, the standard and the symmetric game do not necessarily agree on who wins for a given formula: consider for instance the formula making System the winner if they manage to play twice in a row: the symmetric game is easily won by System, but is won by Environment in the standard setting.

3.3 Token Game on Words

Our goal is to give an alternative game played on a finite arena that would still be equivalent to the symmetric game. As an intermediate step, consider the (infinite) graph of all finite words over Σ: $\mathcal{A}_{\Sigma} = (\Sigma^{\star}, \varepsilon, \Delta)$ where $\Sigma^{\star}$ is the set of nodes, ε is the initial node and $\Delta \subseteq \Sigma^{\star} \times \Sigma \to \Sigma^{\star}$ is the transition function such that $\Delta(w, a) = w \cdot a$. We use this graph as an arena over which a number of tokens are located, each token representing one process and its location being the history of what has been played on this process. We have two sets of System and Environment tokens, numbering n_S for System tokens and n_E for Environment tokens, all of which are initially placed in the ε node. Alternatively and starting from Environment, each player picks one of their token, move it along one edge, and repeat those two operations a finite amount of times. A player can also opt not to move any of its tokens. Then the other player does the same, and this goes on forever. The winning condition is defined by the formula φ: given a play, we take the limit word reached by each token, and see if the collection of those words satisfy φ. It is easy to see that this game is simply a different view of the symmetric game: playing a word $w = (a_0, p_0) \dots (a_n, p_n)$ is equivalent to picking the token representing p_0, moving it along the a_0-labeled edge, and so on. Thus System wins in the symmetric game if and only if System wins the token game on words, for any choice of token sets of correct sizes.

3.4 Token Game on Types

We already established as a consequence of Lemma 3 that we do not actually need to keep track of the full history of each token to know whether φ (which has depth k) is satisfied; counting how many tokens (up to threshold k) there are for each k-type of $\mathrm{FO}^{\mathrm{PREF}}$ is enough to decide. Therefore, the final step is to use the finite graph of $\mathrm{FO}^{\mathrm{PREF}}$-types as an arena instead of using the infinite graph of all words. As for the acceptance condition, the formula φ is abstracted by a set α_φ of k-counting functions of the form $\kappa : \mathrm{Types}^k_{\mathrm{FO}^{\mathrm{PREF}}} \to \{0, 1, \dots, k-1, k+\}$. Each such function gives a count of how many tokens (up to k) can be found in each type, and the acceptance condition α_φ is exactly the set of those functions that satisfy φ.

Let the $\mathrm{FO}^{\mathrm{PREF}}$-arena of depth k be $\mathcal{A}_k = (\mathrm{Types}^k_{\mathrm{FO}^{\mathrm{PREF}}}, \langle\varepsilon\rangle^k_{\mathrm{FO}^{\mathrm{PREF}}}, \rightharpoonup_k)$ where $\mathrm{Types}^k_{\mathrm{FO}^{\mathrm{PREF}}}$ is the set of nodes, $\langle\varepsilon\rangle^k_{\mathrm{FO}^{\mathrm{PREF}}}$ is the initial node, and $\rightharpoonup_k$ is the transition function as defined in Sect. 2.3. Given a pair $(n_S, n_E) \in \mathbb{N}^2$, let us fix two arbitrary disjoint sets of System and Environment *tokens* $\mathbb{T}_\mathrm{S}$ and $\mathbb{T}_\mathrm{E}$ of sizes n_S and n_E respectively, and let $\mathbb{T}$ denote their union. The *token game* over $\mathbb{T}$ for a $\mathrm{FO}^{\mathrm{PREF}}[\lesssim]$ formula φ of depth k is given by the tuple $\mathfrak{G}^{\mathbb{T}}_\varphi = (\mathbb{T}, \mathcal{A}_k, \alpha_\varphi)$.

A *configuration* of this game is a mapping $C : \mathbb{T} \to \mathrm{Types}^k_{\mathrm{FO}^{\mathrm{PREF}}}$ indicating where each token lies in the arena. The initial configuration C_0 maps every token to the initial type $\tau_0 = \langle\varepsilon\rangle^k_{\mathrm{FO}^{\mathrm{PREF}}}$. Starting from Environment, players alternatively pick a number of their respective tokens and move them in the arena following transitions from $\rightharpoonup_k$. A *move* for System (resp. Environment) is a mapping $m_\mathrm{S} : \mathbb{T}_\mathrm{S} \to \mathrm{Types}^k_{\mathrm{FO}^{\mathrm{PREF}}}$ (resp. $m_\mathrm{E} : \mathbb{T}_\mathrm{E} \to \mathrm{Types}^k_{\mathrm{FO}^{\mathrm{PREF}}}$) indicating where to move each token such that for all $t \in \mathbb{T}_\mathrm{S}$, $C(t) \rightharpoonup_k m_\mathrm{S}(t)$ (and similarly for Environment). In particular, in a given configuration C, an empty System move is simply a move equal to C restricted to $\mathbb{T}_\mathrm{S}$, indicating that all System tokens should stay where they are.

Then a *play* π is a sequence of configuration and moves starting from an Environment move and alternating players: $\pi = C_0 \xrightarrow{m^0_\mathrm{E}} C_1 \xrightarrow{m^1_\mathrm{S}} C_2 \xrightarrow{m^2_\mathrm{E}} \cdots$ such that each new configuration is the result of applying the previous move to the previous configuration. Let $\mathrm{Plays}^k_{\mathbb{T}}$ denote the set of plays. A play is *maximal* if it is infinite.

For a given token $t \in \mathbb{T}$, a maximal play π generates a sequence of types $\langle\varepsilon\rangle^k_{\mathrm{FO}^{\mathrm{PREF}}} = \tau_0 \rightharpoonup_k \tau_1 \rightharpoonup_k \dots$ such that $\tau_i = C_{2i}(t)$. Note that it is fine to skip every other configuration, as a token can only be moved during either a System or Environment move but not both. This infinite sequence eventually loops in some type τ_t forever due to the graph of types being a finite tree. The *limit configuration* for π, denoted by C^π_∞, is the configuration that returns τ_t for every token t. Note that there must exists some $i \geq 0$ such that for all $j > i$, $C_j = C^\pi_\infty$. We slightly abuse notations and denote by $\pi(t)$ the type of token t in either the last configuration of π if it is finite or its limit configuration if it is infinite. A play is *winning* if its limit configuration satisfies the acceptance condition α_φ, that is if there is a function $\kappa \in \alpha_\varphi$ such that for all $\tau \in \mathrm{Types}^k_{\mathrm{FO^{PREF}}}$,

$$\begin{cases} |\{t \in \mathbb{T} \mid \pi(t) = \tau\}| = \kappa(\tau) & \text{if } \kappa(\tau) < k\,, \text{ and} \\ |\{t \in \mathbb{T} \mid \pi(t) = \tau\}| \geq k & \text{if } \kappa(\tau) = k+\ . \end{cases}$$

A *strategy* for System is a function $\mathcal{S}$ that given a play returns a System move. A play is *compatible* with $\mathcal{S}$ if all System moves in that play are those given by $\mathcal{S}$. A strategy for System is winning if all maximal plays compatible with it are winning. Finally, we say that a pair $(n_S, n_E) \in \mathbb{N}^2$ is winning for System if System has a winning strategy in the token game (for any choice of token sets $\mathbb{T}_\mathrm{S}$ and $\mathbb{T}_\mathrm{E}$ of corresponding sizes).

Lemma 10. *A pair (n_S, n_E) is winning for System in the token game for φ if and only if System has a (n_S, n_E)-winning strategy for φ in the standard synthesis game.*

For a fixed pair (n_S, n_E), the token game is a finite, albeit very large, game. Remember that our goal is to find whether there exists such a pair that is winning. We show in the next section how to reduce the search to a (large but) finite space.

4 Double Cutoff for Solving the Synthesis Problem

Recall that in terms of expressive power, $\mathrm{FO^{PREF}}[\lesssim]$ is located somewhere between $\mathrm{FO}[\sim]$ whose existential synthesis problem is known to be decidable [9]) and $\mathrm{FO}[\sim, <]$ for which is it undecidable already when restricting to two variables, i.e. for $\mathrm{FO}^2[\sim, <]$ [9]).

In this section, we make a step towards closing the gap by proving our main result:

Theorem 11. *The existential synthesis problem for $FO^{PREF}[\lesssim]$ is decidable.*

To prove Theorem 11 we follow a double cutoff strategy. We first show in Sect. 4.1 that there is no point in considering too many tokens for Environment, where the bound depends on the quantifier depth k of the formula φ but, importantly, not on the number of System tokens. As a second step, we prove in Sect. 4.2 that given a fixed number of Environment tokens (which is a reasonable assumption in view of the previous point), one can restrict one's study to a space where the number of System tokens is bounded by a function of k and the number of Environment tokens. The reasoning is detailed in Sect. 4.3.

4.1 Having More Tokens Makes Things Easier for Environment

First, we prove that beyond some threshold, if Environment can win with some number of tokens, then they can *a fortiori* win with a larger number of tokens. Let us stress that this is not true when the number of tokens is small, as witnessed by the formula φ stating the existence of at least two Environment tokens: in that case, System can benefit from Environment having more tokens.

Lemma 12. *For every $k \in \mathbb{N}$, there exists some $f_E(k)$ such that for any $n_S \in \mathbb{N}$, any $n_E \geq f_E(k)$, and any FO$^{PREF}[\lesssim]$-sentence φ of depth k, if $(n_S, n_E + 1)$ is winning for System then (n_S, n_E) is winning for System.*

Proof. Let $k \in \mathbb{N}$ and let φ be a FO$^{\text{PREF}}[\lesssim]$-sentence of depth k. Let us fix three token sets $\mathbb{T}_{\text{S}}^{n_S}$, $\mathbb{T}_{\text{E}}^{n_E}$, and $\mathbb{T}_{\text{E}}^{n_E+1}$ of sizes n_S, n_E, n_E $+1$ respectively. We note $\mathbb{T} = \mathbb{T}_{\text{S}}^{n_S} \uplus \mathbb{T}_{\text{E}}^{n_E}$ and $\mathbb{T}_+ = \mathbb{T}_{\text{S}}^{n_S} \uplus \mathbb{T}_{\text{E}}^{n_E+1}$. Let $\mathfrak{G} = (\mathbb{T}, \mathcal{A}_k, \alpha_\varphi)$ be the token game over $\mathbb{T}$ for φ and $\mathfrak{G}_+ = (\mathbb{T}_+, \mathcal{A}_k, \alpha_\varphi)$ the same over $\mathbb{T}_+$. We show how to build a winning strategy for System in $\mathfrak{G}$ from a winning strategy in $\mathfrak{G}_+$. But first, let us define some useful properties.

For all types $\tau \in \text{Types}^k_{\text{FO}^{\text{PREF}}}$ we define the height of τ, denoted by $h(\tau)$, as its height in the tree of types, e.g. $h(\tau) = 0$ for any leaf in the tree. Let $h_{\max}$ denote the height of the root $\tau_0 = \langle \varepsilon \rangle^k_{\text{FO}^{\text{PREF}}}$.

In any given configuration obtained by following a play in $\mathfrak{G}$, we say that a type τ is *large* in that configuration if there are at least k Environment tokens in τ. Intuitively, this means that the acceptance condition α_φ cannot differentiate between a configuration with a large type τ and the same configuration with even more tokens in τ. Therefore, if we can ensure that System has a strategy in $\mathfrak{G}$ that simulates the $\mathfrak{G}_+$ winning strategy while always keeping the missing Environment token in a large type, then that strategy would also be winning as α_φ (which is the same in both $\mathfrak{G}$ and $\mathfrak{G}_+$) has no way of distinguishing them.

To that end, we define what it means to have a *huge* number of Environment tokens in one type τ. This is given by a lower bound $F(\tau) = k \cdot |\text{Types}^k_{\text{FO}^{\text{PREF}}}|^{h(\tau)}$ that depends only on k and the height of τ. It guarantees the following properties:

1. A type that is huge is *a fortiori* large.
2. If τ is a huge type with $h(\tau) = 0$, it contains at least $F(\tau) = k$ Environment tokens. And since it is a leaf, all those tokens will stay in this type forever. Thus τ will remain large from this point on.
3. If τ is a huge type with height greater than 0, after any Environment move either τ still remains large, or by the pigeonhole principle there exists another type τ' whose height is strictly lower than $h(\tau)$, that can be reached from τ (i.e. such that $\tau \rightharpoonup_k \tau'$), and such that the number of Environment tokens in τ' is greater than F(τ'), ensuring τ' is also huge.

We then define $f_E(k) = F(\tau_0) = k \cdot |\text{Types}^k_{\text{FO}^{\text{PREF}}}|^{h_{\max}}$. Note that it depends only on k.

By these definitions, and since we assume that $n_E \geq f_E(k)$, τ_0 is huge in the initial configuration because all Environment tokens start in type τ_0. By the third property, this means that after any move by Environment, either τ_0 is still large, or there is (at least) one huge type τ_1 reachable from τ_0. This can be repeated until either a type of height 0 is reached, which will be large forever according to the second property, or we stay in the same large type forever. Formally, we define inductively a function $lt : \mathrm{Plays}^k_{\mathbb{T}} \to \mathrm{Types}^k_{\mathrm{FO}^{\mathrm{PREF}}}$ (lt for "large type") such that $lt(C_0) = \tau_0$, $lt(\pi \xrightarrow{m_\mathrm{S}} C) = lt(\pi)$, and $lt(\pi \xrightarrow{m_\mathrm{E}} C) = \tau$ where τ is either a minimal (in terms of height) type such that $lt(\pi) \rightharpoonup_k \tau$ and τ is huge in C if such a type exists, or $\tau = lt(\pi)$ otherwise. This is well-defined due to the above-mentioned third property, and we easily obtain that $lt(\pi)$ is large in the last configuration of π for any play π.

Now assume $\mathcal{S}_+$ is a winning strategy for System in $\mathfrak{G}_+$. We define a strategy $\mathcal{S}$ for System in $\mathfrak{G}$ using lt and additionally maintaining a play π_+ of $\mathfrak{G}_+$ with the following invariant: for all plays π of $\mathfrak{G}$ that are $\mathcal{S}$-compatible, π_+ is a $\mathcal{S}_+$-compatible play such that the configuration reached after π_+ has the same number of tokens in each type as the configuration reached after π, plus one extra Environment token in $lt(\pi)$. The strategy $\mathcal{S}$ is simply defined as $\mathcal{S}(\pi) = \mathcal{S}_+(\pi_+)$, i.e. it mimics the actions of $\mathcal{S}_+$ on play π_+. Assuming that π_+ is properly defined and that the previously mentioned invariant holds, it is then easy to prove that $\mathcal{S}$ is winning. Indeed, for every configuration that can be reached from a $\mathcal{S}$-compatible play π, there is an almost similar configuration that can be reached by following π_+, which is $\mathcal{S}_+$-compatible, with the only difference being one extra Environment token in the type designated by lt. Since this type is large by definition of lt, α_φ cannot distinguish between the two configurations. Thus, for any maximal play π compatible with $\mathcal{S}$, its limit configuration is indistinguishable from the limit configuration of π_+, which satisfies α_φ by assumption of $\mathcal{S}_+$ being winning. This proves that $\mathcal{S}$ is indeed a winning strategy for System in $\mathfrak{G}$.

It only remains to explain how π_+ is defined and show it satisfies the invariant. Without loss of generality, assume that $\mathbb{T}_\mathrm{E}^{n_E+1} = \mathbb{T}_\mathrm{E}^{n_E} \uplus \{\gamma\}$. We strengthen the second part of the invariant so that the configuration reached after π_+ is such that every token in $\mathbb{T}_\mathrm{E}^{n_E}$ is in the same type as in the configuration reached after π, and with γ being the extra token in type $lt(\pi)$. Initially this play π_+ is the empty play C_0 of $\mathfrak{G}_+$, which trivially satisfies both conditions. Suppose now that π is a $\mathcal{S}$-compatible play and π_+ is a $\mathcal{S}_+$-compatible play that also satisfies the (strengthened) second part of the invariant.

- On any System move m_S in $\mathfrak{G}$ leading to play $\pi \xrightarrow{m_\mathrm{S}} C$, we simply update π_+ to $\pi_+ \xrightarrow{m_\mathrm{S}} C_+$ where C_+ is the result of applying m_S to the last configuration of π_+. Since $\mathcal{S}$ mimics $\mathcal{S}_+$, if $\pi \xrightarrow{m_\mathrm{S}} C$ is $\mathcal{S}$-compatible, then $\pi_+ \xrightarrow{m_\mathrm{S}} C_+$ is $\mathcal{S}_+$-compatible. Moreover, by induction hypothesis, the last configuration of π_+ is the last configuration of π with the extra token γ in $lt(\pi)$. By applying the same System move m_S to both, we easily obtain that C_+ is the same as C plus γ in $lt(\pi \xrightarrow{m_\mathrm{S}} C) = lt(\pi)$.
- On an Environment move m_E in $\mathfrak{G}$ resulting in play $\pi \xrightarrow{m_\mathrm{E}} C$, let $\tau = lt(\pi)$ and $\tau' = lt(\pi \xrightarrow{m_\mathrm{E}} C)$. Let m_E^γ be the Environment move that only affects γ by moving it from τ to τ' (the move can be empty if $\tau = \tau'$). We know such

a move is possible because by definition of lt we have that $\tau \rightharpoonup_k \tau'$. Then with m_{E}^{+} being the Environment move combining m_{E} and m_{E}^{γ}, we update π_+ to $\pi_+ \xrightarrow{m_{\mathrm{E}}^{+}} C_+$. It is trivially still $\mathcal{S}_+$-compatible since no System move has been made. The extra token γ was moved to the new large type given by lt, and all other Environment tokens made the same moves as in $\pi \xrightarrow{m_{\mathrm{E}}} C$, so the invariant still holds.

Thus π_+ is properly defined and always satisfy the required invariant, which concludes the proof. □

4.2 Too Many Tokens Are Useless to System

Dually, one can prove the following lemma, stating that only so many tokens can help the System win; beyond that point, no amount of additional tokens can turn the table and change a losing game into a winning one.

Lemma 13. *For every $k, n_E \in \mathbb{N}$, there exists some $f_S(k, n_E)$ such that for any $n_S \geq f_S(k, n_E)$, if System has an $(n_S + 1, n_E)$-winning strategy in a token game of depth k, then System has an (n_S, n_E)-winning strategy in that game.*

Note that contrary to Lemma 12, where the bound depends only on k, here $f_S(k, n_E)$ depends both on k and the number of tokens of Environment. This cannot be avoided, as showcased by the following example.

Example 14. Remember formula $\varphi_{1a_E \leftrightarrow 1a_S}$ from Example 1. We can show that the (n_S, n_E)-game for $\varphi_{1a_E \leftrightarrow 1a_S}$ cannot be won by System when $n_S < n_E$, as exemplified in Fig. 2. Note that we use for simplicity's sake the representation of $\mathrm{Types}^2_{\mathrm{FO}^{\mathrm{PREF}}}$ from Fig. 1 when we should consider $\mathrm{Types}^3_{\mathrm{FO}^{\mathrm{PREF}}}$, as $\varphi_{1a_E \leftrightarrow 1a_S}$ has depth 3 – this does not matter as $\varphi_{1a_E \leftrightarrow 1a_S}$ cannot distinguish $\langle a_E a_E \rangle^2_{\mathrm{FO}^{\mathrm{PREF}}}$ from $\langle a_E a_E a_E \rangle^2_{\mathrm{FO}^{\mathrm{PREF}}}$.

Starting from the initial configuration depicted in Fig. 2a, Environment can move one of their tokens from $\langle \varepsilon \rangle^2_{\mathrm{FO}^{\mathrm{PREF}}}$ to $\langle a_E \rangle^2_{\mathrm{FO}^{\mathrm{PREF}}}$ (Fig. 2b), forcing System to answer by moving on of their tokens from $\langle \varepsilon \rangle^2_{\mathrm{FO}^{\mathrm{PREF}}}$ to $\langle a_S \rangle^2_{\mathrm{FO}^{\mathrm{PREF}}}$ (Fig. 2c) in order to satisfy the win condition of $\varphi_{1a_E \leftrightarrow 1a_S}$. System could also directly move down other tokens from $\langle \varepsilon \rangle^2_{\mathrm{FO}^{\mathrm{PREF}}}$ to $\langle a_S a_S \rangle^2_{\mathrm{FO}^{\mathrm{PREF}}}$, but this would only make things worse.

Then by moving the same token to $\langle a_E a_E \rangle^2_{\mathrm{FO}^{\mathrm{PREF}}}$ as in Fig. 2d, Environment would force System to move as well their first token to $\langle a_S a_S \rangle^2_{\mathrm{FO}^{\mathrm{PREF}}}$ (cf. Figure 2e). Repeating this sequence a total of n_S times on different tokens would end up "using" all of System's tokens, which would all end up in $\langle a_S a_S \rangle^2_{\mathrm{FO}^{\mathrm{PREF}}}$, as illustrated in Fig. 2g. Environment then only needs to move one last token to $\langle a_E \rangle^2_{\mathrm{FO}^{\mathrm{PREF}}}$ to falsify the win conditions for $\varphi_{1a_E \leftrightarrow 1a_S}$.

Although the insight gained in the proof of Lemma 12 is useful when considering the proof of Lemma 13, the latter is much more involved. Let us nevertheless give the key ideas of the proof. This time, the goal is to convert an

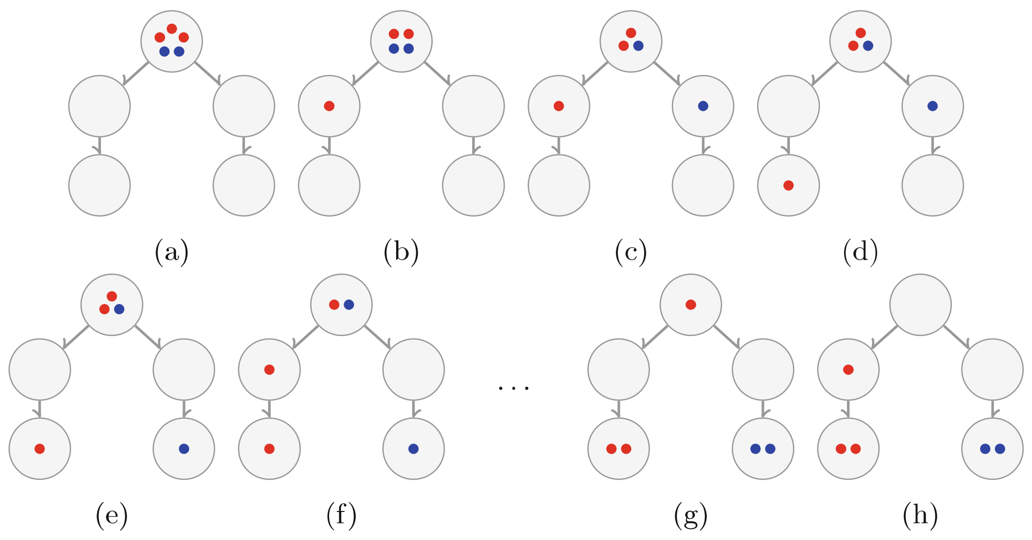

Fig. 2. System may need at least as many tokens as Environment to win.

$(n_S + 1, n_E)$-winning strategy $\mathcal{S}[+]$ for System in a token game of depth k to an (n_S, n_E)-winning strategy $\mathcal{S}[]$ for the same game.

In order to adapt $\mathcal{S}[+]$ to the situation where System has one less token, we will track a *ghost* token. The central idea of the proof is to guarantee at all time that the ghost shares its type with many other tokens, so that its presence or absence is of no import to the winning conditions of the game. Up to that point, the proof is very similar to that of Lemma 12. The main difference in this case is that the ghost is not a fixed token: its identity among the n_S +1 System tokens may vary depending on the way the play unfolds. But once again, if one starts with enough System tokens and is careful in the tracking of this ghost token, it is possible to "hide it" among many others throughout the entire play.

4.3 Solving the Synthesis Problem

Remember that our goal is to decide whether there exists a pair $(n_S, n_E) \in \mathbb{N}^2$ that is winning for a given formula φ of depth k. Using Lemmas 12 and 13, we have shown that we can restrict the search space to the set $N = \{(n_S, n_E) \in \mathbb{N}^2 \mid n_E \leq f_E(k) \wedge n_S \leq f_S(k, n_E)\}$ (which is finite and computable) in the sense that if there is no winning pair in N, then there will be no winning pair in $\mathbb{N}^2$.

Recall that for a fixed pair (n_S, n_E), the corresponding token game is a finite game with finite configurations. As such, it can be solved by seeing it as a game on the graph of configurations, with a Büchi winning condition where accepting configurations are exactly those that satisfy α_φ. From there, a winning strategy can be mapped back to a winning strategy in the (n_S, n_E) token game by looking at the history of a play and constructing the configurations from it. Finally, as the proofs of equivalence in Sect. 3 are constructive, one can then go from a winning strategy in the token game back to a winning strategy in the original synthesis game. Therefore, the synthesis problem can be solved by iterating on all pairs in N and solving the token game for that pair, and then accepting the first winning pair found if there is one, and rejecting if no such pair is found (Fig. 3).

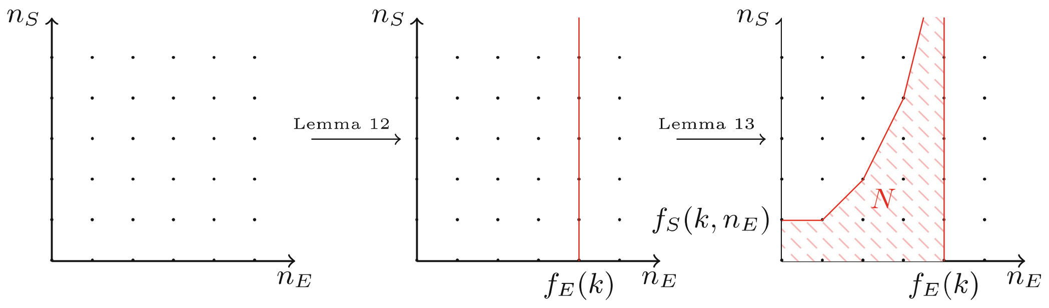

Fig. 3. Bounding the search space for winning pairs.

5 Conclusion

We have shown that the synthesis problem is decidable for $\mathrm{FO}^{\mathrm{PREF}}[\lesssim]$, the prefix first-order logic on data words. Our proof is based on successively bounding the number of Environment and System processes that are relevant for solving the problem, thus restricting the search space to a finite set.

The correctness of those bounds heavily relies on the nice properties exhibited by $\mathrm{FO}^{\mathrm{PREF}}[\lesssim]$. The most important of them is that $\mathrm{FO}^{\mathrm{PREF}}$-types on regular words form a tree, which would not be the case with the unrestricted $\mathrm{FO}[\lesssim]$. We show in the full version of this paper that $\mathrm{FO}^{\mathrm{PREF}}[\lesssim]$ is actually the finest restriction enjoying this property, in the sense that $\mathrm{FO}^{\mathrm{PREF}}$-types correspond exactly to connected components in the graph of types of $\mathrm{FO}[\lesssim]$. To the best of our knowledge, this natural restriction and the properties it enjoys have not been studied anywhere else.

On the synthesis side, our result narrows the gap between decidability and undecidability of first-order logic fragments on data words. Previous results had shown that adding unrestricted order to the allowed predicates immediately lead to undecidability, greatly reducing the kind of specifications that could be written. It turns out that the limited kind of order added by the predicate $\lesssim$ does not share the same outcome, so we obtain a fragment with more expressivity than before and for which synthesis is still decidable. The decidability for the full $\mathrm{FO}[\lesssim]$ remains open; our technique for tracking a missing ghost token cannot be easily adapted in that setting as the type structure for that logic is not as nice as in the $\mathrm{FO}^{\mathrm{PREF}}[\lesssim]$ setting. We conjecture that it remains decidable, and leave this as potential future works.

References

1. Bérard, B., Bollig, B., Lehaut, M., Sznajder, N.: Parameterized synthesis for fragments of first-order logic over data words. In: FoSSaCS 2020. LNCS, vol. 12077, pp. 97–118. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45231-5_6

2. Bojanczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: 21th IEEE Symposium on Logic in Computer Science LICS (2006)
3. Büchi, J.R., Landweber, L.H.: Solving sequential conditions by finite-state strategies. Transactions of the American Mathematical Society (1969)
4. Church, A.: Applications of recursive arithmetic to the problem of circuit synthesis. In: Summaries of the Summer Institute of Symbolic Logic, vol. 1 (1957)
5. Demri, S., d'Souza, D., Gascon, R.: Temporal logics of repeating values. J. Log. Comput. **22**(5), 1059–1096 (2012)
6. Demri, S., Lazić, R.: Ltl with the freeze quantifier and register automata. ACM Trans. Comput. Logic (TOCL) **10**(3), 1–30 (2009)
7. Exibard, L., Filiot, E., Khalimov, A.: A generic solution to register-bounded synthesis with an application to discrete orders. arXiv preprint arXiv:2205.01952 (2022)
8. Figueira, D., Praveen, M.: Playing with repetitions in data words using energy games. In: Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS (2018)
9. Grange, J., Lehaut, M.: First order synthesis for data words revisited. arXiv preprint arXiv:2307.04499 (2023)
10. Grange, J., Lehaut, M.: Synthesis for prefix first-order logic on data words. arXiv preprint arXiv:2404.14517 (2024)
11. Kaminski, M., Francez, N.: Finite-memory automata. Theoret. Comput. Sci. **134**(2), 329–363 (1994)
12. Kara, A.: Logics on data words (2016)
13. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Trans. Comput. Logic (TOCL) **5**(3), 403–435 (2004)
14. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science, pp. 746–757. IEEE (1990)