

ON TWO-VARIABLE GUARDED FRAGMENT LOGIC WITH EXPRESSIVE LOCAL PRESBURGER CONSTRAINTS

CHIA-HSUAN LU ^a AND TONY TAN ^b

^a Department of Computer Science, University of Oxford, United Kingdom.
e-mail address: chia-hsuan.lu@cs.ox.ac.uk

^b Department of Computer Science, University of Liverpool, United Kingdom.
e-mail address: tonytan@liverpool.ac.uk

ABSTRACT. We consider the extension of the two-variable guarded fragment logic with local Presburger quantifiers. These are quantifiers that can express properties such as “the number of incoming blue edges plus twice the number of outgoing red edges is at most three times the number of incoming green edges” and captures various description logics with counting, but without constant symbols. We show that the satisfiability problem for this logic is EXP-complete. While the lower bound already holds for the standard two-variable guarded fragment logic, the upper bound is established by a novel, yet simple deterministic graph-based algorithm.

1. INTRODUCTION

In this paper we consider the extension of two-variable guarded fragment logic with the so-called *local Presburger quantifiers*, which we denote by GP^2 . These are quantifiers that can express local numerical properties such as “the number of outgoing red edges plus twice the number of incoming green edges is at most three times the number of outgoing blue edges.” It was first considered by Bednarczyk, et. al. [BOPT21] and trivially subsumes GC^2 (two-variable guarded fragments with counting quantifiers), which captures various description logics with counting such as \mathcal{ALCIHQ} and $\mathcal{ALCIHb}^{\text{self}}$ [BHLS17, BCM⁺03, Grä98]. Bednarczyk, et. al. [BOPT21] showed that both satisfiability and finite satisfiability problems are in 3-NEXP by reduction to the two-variable logic with counting quantifiers.

We show that the satisfiability problem for this logic is EXP-complete. The lower bound is already known for the standard two-variable guarded fragments [Corollary 4.6][Grä99]. Our contribution is the upper bound, which is established by a novel, yet simple deterministic exponential time algorithm which works similarly to the type elimination approach, first introduced by Pratt [Pra79]. Intuitively, it starts by representing the input sentence as a graph whose vertices and edges represent the allowed types. It then successively eliminates the vertex or edge that contradicts the input sentence until there is no more vertex or edge to eliminate.

Key words and phrases: Two-variable guarded fragment, local counting constraints, satisfiability, EXP-complete.

Our algorithm has a markedly different flavour from the standard tableaux method usually used to establish the upper bound of guarded fragments. Note that the tableaux method works by exploiting the so-called *tree-like model* property, where it tries to construct a tree-like model using a polynomial space alternating Turing machine and a configuration of the Turing machine corresponds to an element in the model. To apply this method, it is essential that there is only a polynomial bound on the branching degree of each node in the tree and it is not clear whether this bound still holds for GP^2 . In fact, already for GC^2 , the degree can be exponential (when the counting quantifiers are encoded in binary). To circumvent the exponential blow-up, the EXP upper bound for the satisfiability problem of GC^2 is obtained by first reducing it to three-variable guarded fragments, before applying the tableaux method [Kaz04]. It was not clear *a priori* how this technique can be extended to the satisfiability of GP^2 .

Acknowledgement. Recently Bednarczyk and Fiuk [BF22] independently obtained the same EXP upper bound for the satisfiability problem for GP^2 . Their proof uses the tableaux method. To avoid the exponential branching degree, they restrict each node in the tableau to correspond only to the *type* of an element in a model and it branches only to the different types of the children element. This method yields an alternating polynomial space algorithm, hence, a deterministic exponential time algorithm. A brief comparison between their algorithm and ours is presented in Section 4.5.

Other related work. The guarded fragment is one of the most prominent decidable fragments of first-order logic [ANvB98]. The satisfiability problem is 2-EXP-complete and it becomes EXP-complete when the number of variables or the arity of the signature is fixed [Theorem 4.4 and Corollary 4.6][Grä99]. Various description (DL) and modal (ML) logics are captured by the fragment when the arity is fixed to two [Grä98, BCM⁺03, BHLS17]. The key reason for the decidability of the guarded fragment is the *tree-like model property* which allows the application of the tableaux method [Var96].

Recently, a deterministic exponential time algorithm for a fragment of the two-variable logic was proposed by Lin, et. al. [LLT21]. The algorithm there is also a graph-based algorithm, and has a similar flavour to type-elimination algorithm. However, it is not clear, *a priori*, how Lin, et. al.'s algorithm can be extended to GP^2 .

Pratt-Hartmann [Pra07] proposed an elegant reduction of the satisfiability problem for GC^2 formulas to the solvability of (exponential size) homogeneous instances of Integer Linear Programming (ILP) which are of the form $A\bar{x} = 0 \wedge B\bar{x} \geq \bar{c}$, where A and B are matrices with integer entries, \bar{x} is a (column) vector of variables and \bar{c} is a vector of integers. To check whether $A\bar{x} = 0 \wedge B\bar{x} \geq \bar{c}$ admits a solution in \mathbb{N} , it is sufficient to check whether it admits a solution in \mathbb{Q} , which is known to be in PTIME. This implies the EXP upper bound for both the satisfiability and finite satisfiability problems for GC^2 .

In general, GP^2 captures various description logics with counting such as $\mathcal{ALCITHQ}$ and $\mathcal{ALCITHb}^{\text{self}}$, but without nominals [BCM⁺03]. Note that allowing nominals in GC^2 makes the complexity of the satisfiability problem becomes NEXP-complete [Tob01].

Some logics that allow similar quantifiers as the local Presburger quantifiers were proposed and studied by various researchers [Baa17, BBR20, DL10, KP10]. The decidability result is obtained by the tableaux method, but their logics do not allow the inverses of binary relations.

The extension of *one-variable logic* with quantifiers of the form $\exists^S x \phi(x)$, where S is a ultimately periodic set, is NP-complete [Bed20]. This logic is similar to the quantifier-free fragment of Boolean Algebra with Presburger Arithmetic (QFBAPA) introduced by Kuncak and Rinard [KR07]. Semantically $\exists^S x \phi(x)$ means the number of x where $\phi(x)$ holds is in the set S . The extension of two-variable logic with such quantifiers is later shown to be 2-NEXP by Benedikt, et. al. [BKT20] and the proof makes heavy use of the *biregular graph method* introduced by Kopczynski and Tan [KT15] to analyse the spectrum of two-variable logic with counting quantifiers. These proofs and results do not apply in our setting since the logics they considered already subsume the two-variable logic.

Organisation. This paper is organised as follows. In Section 2 we define *linear constraints* and review two structures of natural numbers for their semantics. One structure is the standard structure of natural numbers and the other is its extension with ∞ value. In Section 3 we present the formal definition of GF². The main result is presented in Section 4. We conclude with Section 5.

2. PRESBURGER ARITHMETIC

Let \mathbb{N} denote the set of natural numbers including 0.

Linear constraints. We assume a countable infinite set \mathcal{X} of variables. A *linear constraint* ξ is a constraint of the form:

$$\xi := \kappa_1 x_1 + \cdots + \kappa_k x_k \quad \circledast \quad \delta + \lambda_1 y_1 + \cdots + \lambda_\ell y_\ell,$$

where $x_1, \dots, x_k, y_1, \dots, y_\ell$ are variables from \mathcal{X} , all $\delta, \kappa_1, \dots, \kappa_k, \lambda_1, \dots, \lambda_\ell$ are from \mathbb{N} and \circledast is one of the symbols $=, \neq, \leq, \geq, <, >$. \equiv_d or $\not\equiv_d$, where $d \in \mathbb{N} - \{0\}$. The relation \equiv_d is intended to mean congruence modulo d , i. e., $n \equiv_d m$ if and only if $m, n \in \mathbb{N}$ and there is $k_1, k_2 \in \mathbb{N}$ such that $n + k_1 d = m + k_2 d$.

We let $\text{var}(\xi)$ denote the set of variables that appear in the constraint ξ mentioned above, i.e., $\text{var}(\xi) = \{x_1, \dots, x_k, y_1, \dots, y_\ell\}$. The coefficients in ξ are the natural numbers $\delta, \kappa_1, \dots, \kappa_k, \lambda_1, \dots, \lambda_\ell$ which include d when \circledast is \equiv_d or $\not\equiv_d$.

A finite set \mathcal{C} of linear constraints is also called a *system of linear constraints*, or in short, *system*. When convenient, we will also view a system \mathcal{C} as a conjunction of its constituent linear constraints. We let $\text{var}(\mathcal{C})$ denote the set $\bigcup_{\xi \in \mathcal{C}} \text{var}(\xi)$. The coefficients in \mathcal{C} are the coefficients in the linear constraints in \mathcal{C} .

We will consider two structures \mathfrak{N} and \mathfrak{N}_∞ in which the satisfaction of a linear constraint is defined. Intuitively \mathfrak{N} is the standard structure of natural numbers and \mathfrak{N}_∞ is its extension with the value ∞ .

The structure \mathfrak{N} . The semantics of linear constraints can be naturally defined over the structure of natural numbers $\mathfrak{N} = (\mathbb{N}, +, \cdot, \leq, 0, 1)$ where $+, \cdot, \leq, 0, 1$ are interpreted in the standard way. Note that by using the relation \leq and the equality predicate $=$, we can define the relations $<$ and \equiv_d in \mathfrak{N} as follows.

- $m < n$ if and only if $m \leq n$ and $m \neq n$, for every $n, m \in \mathbb{N}$.
- $n \equiv_d m$ if and only if there is $k_1, k_2 \in \mathbb{N}$ such that $n + k_1 d = m + k_2 d$.

An *assignment* to the variables in ξ is a mapping $F : \text{var}(\xi) \rightarrow \mathbb{N}$. It satisfies the linear constraint ξ , if ξ holds in \mathfrak{N} when each variable x is assigned with the value $F(x)$. A *solution* to a system \mathcal{C} is an assignment $F : \text{var}(\mathcal{C}) \rightarrow \mathbb{N}$ that satisfies every linear constraint in \mathcal{C} . If such a solution exists, we say that \mathcal{C} *admits a solution* in \mathfrak{N} . The following theorem establishes useful bounds on the solution of a system of linear constraints.

Theorem 2.1. *There are constants $c_1, c_2 \in \mathbb{N}$ such that for every system \mathcal{C} of linear constraints, the following holds where $t = |\mathcal{C}|$ and M is the maximal coefficient in \mathcal{C} .¹*

- (1) [ES06, Theorem 1] *If \mathcal{C} admits a solution in \mathfrak{N} , then it admits a solution in \mathfrak{N} in which the number of variables assigned with non-zero values is at most $c_1 t \log_2(c_2 t M)$.*
- (2) [Pap81, Theorem]² *If \mathcal{C} admits a solution in \mathfrak{N} , then it admits a solution in \mathfrak{N} in which every variable is assigned with a value at most $c_1 t (tM)^{c_2 t}$.*

Combining (1) and (2) in Theorem 2.1, we obtain the following corollary, which will be useful to establish the upper bound of our algorithm.

Corollary 2.2. *There are constants $c_1, c_2 \in \mathbb{N}$ such that for every system \mathcal{C} of linear constraints, if \mathcal{C} admits a solution in \mathfrak{N} , then it admits a solution in \mathfrak{N} in which the number of variables assigned with non-zero values is at most $c_1 t \log_2(c_2 t M)$ and every variable is assigned with a value at most $c_1 t (tM)^{c_2 t}$, where $t = |\mathcal{C}|$ and M is the maximal coefficient in the system \mathcal{C} .*

Proof. Let c_1, c_2 be the constant in Theorem 2.1. Let \mathcal{C} be a system of linear constraints, where $t = |\mathcal{C}|$ and M is the maximal coefficient in \mathcal{C} . Suppose \mathcal{C} admits a solution in \mathfrak{N} . By (1) in Theorem 2.1, it admits a solution in \mathfrak{N} in which the number of variables assigned with non-zero values is at most $c_1 t \log_2(c_2 t M)$. Let \mathcal{X} be the set of variables in $\text{var}(\mathcal{C})$ that are assigned with non-zero values. Thus, $|\mathcal{X}| \leq c_1 t \log_2(c_2 t M)$.

Let \mathcal{C}' be the system obtained from \mathcal{C} by assigning all the variables not in \mathcal{X} with the zero value. Thus, $\text{var}(\mathcal{C}') = \mathcal{X}$. Let $t' = |\mathcal{C}'|$ and M' be the maximal coefficient in \mathcal{C}' . Note that $t' \leq t$ and $M' \leq M$ and that \mathcal{C}' admits a solution in \mathfrak{N} . By (2) in Theorem 2.1, \mathcal{C}' admits a solution in \mathfrak{N} in which every variable is assigned with a value at most $c_1 t' (t' M')^{c_2 t'} \leq c_1 t (tM)^{c_2 t}$. This solution of \mathcal{C}' can be extended to a solution of \mathcal{C} where every variable not in \mathcal{X} is assigned with zero value. \square

The structure \mathfrak{N}_∞ . We assume a constant symbol ∞ which symbolises the infinite value. The extension of \mathfrak{N} with ∞ is the structure $\mathfrak{N}_\infty = (\mathbb{N} \cup \{\infty\}, +, \cdot, \leq, 0, 1)$ where $+, \cdot, \leq$ involving elements in \mathbb{N} are defined as in \mathfrak{N} . When ∞ is involved, they are defined as follows. $n < \infty$ and $n + \infty = \infty + n = \infty + \infty = \infty$ for every $n \in \mathbb{N}$, $n \cdot \infty = \infty \cdot n = \infty$ for every $n \neq 0$, and $0 \cdot \infty = \infty \cdot 0 = 0$. An *assignment* is a mapping $F : \text{var}(\xi) \rightarrow \mathbb{N} \cup \{\infty\}$. The notion of a system admitting a solution in \mathfrak{N}_∞ is defined in the similar manner as in \mathfrak{N} .

Note that the constraint $x = x + 1$ does not admit a solution in \mathfrak{N} , but admits a unique solution in \mathfrak{N}_∞ where x is assigned with the value ∞ . Thus, we can view the satisfiability

¹The results by Eisenbrand and Shmonin [ES06, Theorem 1] and Papadimitriou [Pap81, Theorem] are stated in terms of Integer Linear Programming (ILP), which is a conjunction of a finite number of linear constraints not involving \equiv_d and its negation $\not\equiv_d$. Since $a \equiv_d b$ is equivalent to $a + k_1 d = b + k_2 d$ for some $k_1, k_2 \in \mathbb{N}$, it is obvious that the results also hold when \equiv_d is involved. Note that when \mathcal{C} contains a linear constraint involving \equiv_d , the integer d is considered a coefficient in \mathcal{C} .

²There is only one theorem in the paper by Papadimitriou [Pap81] and it is without a number.

of a system of linear constraints in \mathfrak{N} as a special case of the satisfiability in \mathfrak{N}_∞ , where we have the linear constraint $x \neq x + 1$ for every variable x .

It is a folklore result that Corollary 2.2 can be extended to \mathfrak{N}_∞ , see, e.g., [Pra15, Section 2.3]. We state it formally in Corollary 2.3.

Corollary 2.3. *There are constants $c_1, c_2 \in \mathbb{N}$ such that for every system \mathcal{C} of linear constraints, if \mathcal{C} admits a solution in \mathfrak{N}_∞ , then it admits a solution in \mathfrak{N}_∞ in which the number of variables assigned with non-zero values is at most $c_1 t \log_2(c_2 t M)$ and every variable is assigned with either ∞ or a value at most $c_1 t (tM)^{c_2 t}$, where $t = |\mathcal{C}|$ and M is the maximal coefficient in \mathcal{C} .*

3. TWO-VARIABLE GUARDED FRAGMENT WITH LOCAL PRESBURGER QUANTIFIERS (GP²)

We fix a vocabulary Σ consisting of only unary and binary predicates. We do not allow for constant symbols and we will consider the logic with the equality predicate. As usual, for a vector \bar{x} of variables, $\varphi(\bar{x})$ denotes a formula φ whose free variables are exactly those in \bar{x} .

Let \mathcal{A} be a structure and let a be an element in \mathcal{A} . For a formula $\varphi(x, y)$, we denote by $|\varphi(x, y)|_{\mathcal{A}}^{x/a}$ the number of elements b such that $\mathcal{A}, x/a, y/b \models \varphi(x, y)$. When the value $|\varphi(x, y)|_{\mathcal{A}}^{x/a}$ is not finite, we write $|\varphi(x, y)|_{\mathcal{A}}^{x/a} = \infty$.

Local Presburger (LP) quantifiers. The *local Presburger* (LP) quantifiers are quantifiers of the form:

$$\mathcal{P}(x) := \sum_{i=1}^n \lambda_i \cdot \#_y^{r_i}[\varphi_i(x, y)] \quad \otimes \quad \delta + \sum_{i=1}^m \kappa_i \cdot \#_y^{s_i}[\psi_i(x, y)],$$

where $\delta, \lambda_i, \kappa_i \in \mathbb{N}$; each r_i, s_i is an atom $R(x, y)$ or $R(y, x)$ for some binary relation R ; each $\varphi_i(x, y), \psi_i(x, y)$ is a formula with free variables x and y ; and \otimes is one of the symbols $=, \neq, \leq, \geq, <, >, \equiv_d$ or $\not\equiv_d$, where $d \in \mathbb{N} - \{0\}$. Note that $\mathcal{P}(x)$ has precisely one free variable x . We say that $\mathcal{P}(x)$ holds in $\mathcal{A}, x/a$, denoted $\mathcal{A}, x/a \models \mathcal{P}(x)$, if the (in)equality \otimes holds in \mathfrak{N}_∞ when each $\#_y^{r_i}[\varphi_i(x, y)]$ and $\#_y^{s_i}[\psi_i(x, y)]$ are substituted with $|r_i(x, y) \wedge \varphi_i(x, y)|_{\mathcal{A}}^{x/a}$ and $|s_i(x, y) \wedge \psi_i(x, y)|_{\mathcal{A}}^{x/a}$, respectively.

Note that the negation of the LP quantifier $\mathcal{P}(x)$ stated above is obtained by changing the relation symbol \otimes to its negated counterpart, i.e., the symbol $=$ is changed to \neq, \leq to $>, \equiv_d$ to $\not\equiv_d$ and so on. For example, the negation of $3 \cdot \#_y^{R(x, y)}[\varphi(x, y)] \equiv_5 7$ is $3 \cdot \#_y^{R(x, y)}[\varphi(x, y)] \not\equiv_5 7$.

In the following, to avoid clutter, we will allow the constants in an LP quantifier to be negative integers and write it in the form:

$$\mathcal{P}(x) := \sum_{i=1}^n \lambda_i \cdot \#_y^{r_i}[\varphi_i(x, y)] \quad \otimes \quad \delta$$

When we do so, it is implicit that we mean the LP quantifier obtained after rearranging the terms so that none of the coefficients are negative integers. For example, when we write:

$$3 \cdot \#_y^{R(x, y)}[\varphi(x, y)] - 2 \cdot \#_y^{S(y, x)}[\psi(x, y)] = -7,$$

we mean:

$$2 \cdot \#_y^{S(y, x)}[\psi(x, y)] = 7 + 3 \cdot \#_y^{R(x, y)}[\varphi(x, y)].$$

We say that a quantifier $\mathcal{P}(x)$ is in *basic form*, if it is of the form:

$$\mathcal{P}(x) := \sum_{i=1}^n \lambda_i \cdot \#_y^{R_i(x,y)}[\varphi_i(x,y)] \otimes \delta,$$

where each $\varphi_i(x,y)$ is either the equality $x = y$ or the inequality $x \neq y$.

Remark 3.1. With LP quantifiers one can state whether an element has finite or infinitely many outgoing edges. Consider the following LP quantifier:

$$\mathcal{P}_\infty(x) := \#_y^{R(x,y)}[\top] = \#_y^{R(x,y)}[\top] + 1.$$

Since the equality only holds when $|R(x,y)|_{\mathcal{A}}^{x/a} = \infty$, it follows that $\mathcal{A}, x/a \models \mathcal{P}_\infty(x)$ if and only if there are infinitely many outgoing R -edges from a in the structure \mathcal{A} .

Note that the negation of $\mathcal{P}_\infty(x)$ is:

$$\mathcal{P}_{\text{fin}}(x) := \#_y^{R(x,y)}[\top] \neq \#_y^{R(x,y)}[\top] + 1.$$

Thus, $\mathcal{A}, x/a \models \mathcal{P}_{\text{fin}}(x)$ if and only if there are finitely many outgoing R -edges from a in the structure \mathcal{A} .

The guarded fragment class. The class GF of guarded fragment logic is the smallest set of first-order formulas such that:³

- GF contains all atomic formulas $R(\bar{x})$ and equalities between variables.
- GF is closed under boolean combinations.
- If $\varphi(\bar{x})$ is in GF, $R(\bar{z})$ is an atom and $\bar{x}, \bar{y} \subseteq \bar{z}$, then both $\exists \bar{y} R(\bar{z}) \wedge \varphi(\bar{x})$ and $\forall \bar{y} R(\bar{z}) \rightarrow \varphi(\bar{x})$ are also in GF.

We define the class GP to be the extension of GF with LP quantifiers, i.e., by adding the following rule.

- An LP quantifier $\mathcal{P}(x) := \sum_{i=1}^n \lambda_i \cdot \#_y^{r_i}[\varphi_i(x,y)] \otimes \delta$ is in GP if and only if each $\varphi_i(x,y)$ is in GP.

We denote by GP² the restriction of GP to formulas using only two variables: x and y . As before, when considering GP² formulas, we assume that the vocabulary is Σ , i.e., the arities of the predicates is at most 2 and there is no constant symbol.

Remark 3.2. The standard quantifiers \forall and \exists in GF² can be expressed using LP quantifiers. The universal quantifier $\forall y r(x,y) \rightarrow \varphi(x,y)$ is equivalent to:

$$\#_y^{r(x,y)}[\neg\varphi(x,y)] = 0,$$

and $\exists y r(x,y) \wedge \varphi(x,y)$ is equivalent to:

$$\#_y^{r(x,y)}[\varphi(x,y)] \geq 1.$$

It is also for this reason that we call LP quantifiers “local Presburger quantifiers” as it allows us the Presburger arithmetic reasoning on the neighbourhood of an element. The term “quantifier” in this case is similar in spirit with Härtig quantifiers (cardinality comparison quantifiers) [Här62, HaAPV91] and ultimately periodic counting quantifiers [BKT20].

³In this paper $\bar{x}, \bar{y}, \bar{z}$ denote vectors of variables and $\bar{x} \subseteq \bar{z}$ means that all the variables that appear in \bar{x} also appear in \bar{z} .

We denote by SAT(GP²) the problem that on input GP² formula φ , decides if it is satisfiable. The following lemma states that it suffices to consider only GP² formulas in a normal form.

Lemma 3.3. *There is a linear time algorithm that converts a GP² formula into an equisatisfiable first-order formula in the normal form (over an extended signature):*

$$\Psi := \forall x \gamma(x) \quad \wedge \quad \bigwedge_{i=1}^k \forall x \forall y \alpha_i(x, y) \quad \wedge \quad \bigwedge_{i=1}^{\ell} \forall x (q_i(x) \rightarrow \mathcal{P}_i(x)), \quad (3.1)$$

where

- $\gamma(x)$ is a quantifier-free formula,
- each $\alpha_i(x, y)$ is a quantifier-free formula of the form:

$$(R(x, y) \wedge x \neq y) \rightarrow \beta(x, y)$$

where $R(x, y)$ is an atomic formula and $\beta(x, y)$ is a quantifier-free formula,

- each $q_i(x)$ is an atomic formula,
- each $\mathcal{P}_i(x)$ is an LP quantifier in the basic form.

Proof. The proof uses a routine renaming technique (see, e.g., [Kaz04, Pra07]). For completeness, we present it here. By pushing the negation inward, we can assume that the input formula is in negation normal form, i.e., every negation is applied only on atomic formulas. Intuitively we rename each subformula of one free variable with a fresh unary predicate. The renaming is done bottom-up starting from the subformula with the lowest quantifier rank. We consider four cases.

- Case 1: A subformula in the form of an LP quantifier:

$$\mathcal{P}(x) := \sum_{i=1}^n \lambda_i \cdot \#_y^{\tau_i} [\varphi_i(x, y)] \quad \otimes \quad \delta \quad (3.2)$$

where each $\varphi_i(x, y)$ is quantifier-free.

- Case 2: A subformula with one free variable x of the form:

$$\psi(x) := \varphi_1(x) \wedge \varphi_2(x) \quad (3.3)$$

where $\varphi_1(x)$ and $\varphi_2(x)$ are quantifier-free.

- Case 3: A subformula with one free variable x of the form:

$$\psi(x) := \forall y r(x, y) \rightarrow \varphi(x, y) \quad (3.4)$$

where $\varphi(x, y)$ is quantifier-free.

- Case 4: A subformula with one free variable x of the form:

$$\psi(x) := \exists y r(x, y) \wedge \varphi(x, y) \quad (3.5)$$

where $\varphi(x, y)$ is quantifier-free.

All the other cases can be handled in a similar manner.

We start with Case 1. Let $\mathcal{P}(x)$ be an LP quantifier in form of (3.2). We will first convert it to the basic form. We introduce new binary predicates R_1, \dots, R_n and rewrite $\mathcal{P}(x)$ into $\mathcal{P}'(x)$ as:

$$\mathcal{P}'(x) := \sum_{i=1}^n \lambda_i \cdot \#_y^{R_i(x,y)} [x = y] + \lambda_i \cdot \#_y^{R_i(x,y)} [x \neq y] \quad \otimes \quad \delta.$$

It is useful to note that $\mathcal{P}'(x)$ is actually equivalent to $\sum_{i=1}^n \lambda_i \cdot \#_y^{R_i(x,y)}[\top] \otimes \delta$.

Obviously, $\mathcal{P}'(x)$ is in the basic form. Then we replace $\mathcal{P}'(x)$ with a fresh unary predicate symbol $q(x)$ and add the following conjunct to the original formula.

$$\forall x q(x) \rightarrow \mathcal{P}'(x).$$

We further add the conjunct that asserts that each $R_i(x, y)$ is equivalent to $r_i(x, y) \wedge \varphi_i(x, y)$:

$$\bigwedge_{i=1}^n \forall x \forall y R_i(x, y) \rightarrow (r_i(x, y) \wedge \varphi_i(x, y)) \wedge \bigwedge_{i=1}^n \forall x \forall y r_i(x, y) \rightarrow (\varphi_i(x, y) \rightarrow R_i(x, y)).$$

Finally note that each sentence of the form $\forall x \forall y S(x, y) \rightarrow \zeta(x, y)$, where $S(x, y)$ is atomic and $\zeta(x, y)$ is quantifier-free, can be rewritten into the form:

$$\forall x S(x, x) \rightarrow \zeta(x, x) \quad \wedge \quad \forall x \forall y S(x, y) \wedge x \neq y \rightarrow \zeta(x, y).$$

This finishes the renaming for Case 1.

For Case 2, i.e., a subformula $\psi(x)$ in the form of (3.3), we replace $\psi(x)$ with a fresh unary predicate symbol $q(x)$ and add the following conjunct to the original formula.

$$\forall x q(x) \rightarrow (\varphi_1(x) \wedge \varphi_2(x)).$$

Note that $\forall x q(x) \rightarrow (\varphi_1(x) \wedge \varphi_2(x))$ is in the form of the first part of (3.1). Since $\bigwedge_i \forall x \gamma_i(x)$ is equivalent to $\forall x \bigwedge_i \gamma_i(x)$, adding such conjunct does not violate the form of (3.1).

Finally, for Cases 3 and 4, i.e., a subformula $\psi(x)$ in the form of (3.4) or (3.5), we can rewrite it into an LP quantifier as in Remark 3.2 and proceed according to Case 1.

We perform such renaming procedure repeatedly until we obtain a GP^2 sentence in normal form (3.1). This completes the proof of Lemma 3.3. \square

Remark 3.4. We also note that if the sentence Ψ in normal form (3.1) is satisfiable, then it is satisfied by an infinite model. Indeed, let \mathcal{A} be a model of Ψ . We make infinitely many copies of \mathcal{A} , denoted by $\mathcal{A}_1, \mathcal{A}_2, \dots$, and disjoint union them all to obtain a new model \mathcal{B} . For every pair (a, b) , where a and b do not come from the same \mathcal{A}_i , we set (a, b) not to be in any binary relation $R^{\mathcal{B}}$. It is routine to verify that \mathcal{B} still satisfies Ψ .

We also note that there is a GP^2 sentence that is satisfied only by infinite models. Consider the following sentence with one unary predicate U and one binary predicate R .

$$\Phi := \forall x U(x) \wedge \forall x (U(x) \rightarrow \#_y^{R(x,y)}[\top] = 2) \wedge \forall x (U(x) \rightarrow \#_y^{R(y,x)}[\top] \leq 1)$$

Intuitively, Φ states that every element must belong to the unary predicate U and that every element has exactly two outgoing R -edges and at most one incoming R -edge. It is routine to verify that an infinite binary tree (whose vertices are all in U) satisfies Φ and that every model of Φ must be infinite.

4. THE SATISFIABILITY OF (GP^2) WITH ARITHMETIC OVER \mathfrak{N}_∞

We introduce some terminology in Section 4.1. Then, we show how to represent GP^2 formulas as graphs in Section 4.2. The algorithm is presented in Section 4.3 and the analysis is in Section 4.4. In Section 4.5 we make a brief comparison between our algorithm and the approach by Bednarczyk and Fiuk [BF22]. Throughout this section we fix a sentence Ψ in the normal form (3.1) over the signature Σ .

4.1. Terminology. A *unary type* (over Σ) is a maximally consistent set of atomic and negated atomic formulas using only variable x , including atoms such as $r(x, x)$ and their negations $\neg r(x, x)$. Similarly, a *binary type* is a maximally consistent set of binary atoms and negations of atoms containing $x \neq y$, where each atom or its negation uses two variables x and y . The binary type that contains only the negations of atomic predicates from Σ is called the *null type*, denoted by η_{null} . All the other binary types are called *non-null* types.

Note that we require that each atom and the negation of an atom in a binary type explicitly mentions both x and y and $x \neq y$. This is a little different from the standard definitions, such as the ones defined by Gradel, et. al. [GKV97] and Pratt-Hartmann [Pra05], where a binary type may contain unary atoms or negations of unary atoms involving only x or y . The purpose of such deviation is to make the disjointness between the set of unary types and the set of binary types, which is only for technical convenience.

Note also that both unary and binary types can be identified with the quantifier-free formula formed as the conjunction of its constituent formulas. We will use the symbols π and η (possibly indexed) to denote unary and binary types, respectively. When viewed as formulas, we write $\pi(x)$ and $\eta(x, y)$, respectively. We write $\pi(y)$ to denote the formula $\pi(x)$ with x being substituted with y . We denote by $\bar{\eta}(x, y)$ the “reverse” of $\eta(x, y)$, i.e., the binary type obtained by swapping the variables x and y . Let Π denote the set of all unary types over Σ and let \mathcal{K} be the set of all *non-null* binary types over Σ .

For a structure \mathcal{A} (over Σ), the *type of an element* $a \in A$ is the unique unary type π that a satisfies in \mathcal{A} . Similarly, the *type of a pair* $(a, b) \in A \times A$, where $a \neq b$, is the unique binary type that (a, b) satisfies in \mathcal{A} . The *configuration* of a pair (a, b) is the tuple (π, η, π') where π and π' are the types of a and b , respectively, and η is the type of (a, b) . It is a non-null configuration when η is a non-null type. In this case we will also say that b is a (η, π') -neighbour of a in the structure \mathcal{A} . When (η, π') is clear from the context, we will simply say that b is a *neighbour* of a . The set of all neighbours of a is called the *neighbourhood* of a .

We say that a unary type π is *realised* in \mathcal{A} , if there is an element whose type is π . Likewise, a configuration (π, η, π') is *realised* in \mathcal{A} , if there is a pair (a, b) whose configuration is (π, η, π') .

The graph representation. In this paper the term graph means *finite edge-labelled directed* graph $G = (V, E)$, where $V \subseteq \Pi$ and $E \subseteq V \times \mathcal{K} \times V$. We can think of an edge (π, η, π') as a potential configuration for a pair in a structure. We allow multiple edges between two vertices provided that they have different labels. Similarly, we also allow multiple self-loops on a vertex provided that they have different labels.

Let $\mathcal{N}_G(\pi)$ denote the set $\{(\eta, \pi') \mid (\pi, \eta, \pi') \in E\}$, i.e., the set of all the edges going out from π . A graph $H = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E \cap (V' \times \mathcal{K} \times V')$. If $V' \neq \emptyset$, we call H a *non-empty subgraph* of G .

Definition 4.1 below provides the link between structures and graphs.

Definition 4.1. A structure \mathcal{A} *conforms* to a graph G , if all of the following conditions hold.

- If a type π is realised in \mathcal{A} , then π is a vertex in G .
- If a configuration (π, η, π') is realised in \mathcal{A} , where η is a non-null type, then (π, η, π') is an edge in G .

Next, we show how the sentence Ψ can be represented as a graph. Recall that Ψ is a GP^2 sentence in the normal form (3.1). We need the following two definitions.

Definition 4.2. A unary type π is *compatible with Ψ* , if $\pi(x) \models \gamma(x)$.

Definition 4.3. A configuration (π, η, π') is *compatible with Ψ* , if both π and π' are compatible with Ψ and for each $1 \leq i \leq k$:

$$\pi(x) \wedge \eta(x, y) \wedge \pi'(y) \models \alpha_i(x, y) \quad \text{and} \quad \pi(y) \wedge \bar{\eta}(x, y) \wedge \pi'(x) \models \alpha_i(x, y).$$

Recall that for each unary type π , for each predicate $U(x)$, exactly one of $U(x)$ or $\neg U(x)$ belongs to π . Thus, to determine whether $\pi(x) \models \gamma(x)$, it suffices to assign each atom $U(x)$ in $\gamma(x)$ according to the type π , i.e., we assign an atom $U(x)$ with true, if $U(x)$ belongs to π and false, otherwise, and evaluate the boolean value of $\gamma(x)$ according to the standard semantics of \wedge , \vee and \neg . Therefore, determining whether π is compatible with Ψ can be done in polynomial time in the size of π and the length of Ψ . Similarly, since a configuration (π, η, π') determines the truth value of each atom in each $\alpha_i(x, y)$, determining whether it is compatible with a formula Ψ can be done in polynomial time in the size of π, η, π' and the length of Ψ . Thus, listing all compatible unary types and configurations takes exponential time in the length of Ψ .

The sentence Ψ defines a directed graph G_Ψ , where the vertices are the unary types that are compatible with Ψ and the edges are (π, η, π') , for every (π, η, π') compatible with Ψ . Note that the graph G_Ψ is “symmetric” in the sense that (π, η, π') is an edge if and only if $(\pi', \bar{\eta}, \pi)$ is an edge.

Intuitively, the vertices in the graph G_Ψ are the unary types that do not violate $\forall x \gamma(x)$ and the edges are the binary types that do not violate the conjunct $\bigwedge_{i=1}^k \forall x \forall y \alpha_i(x, y)$. Note that if a structure \mathcal{A} satisfies Ψ , then it is necessary that \mathcal{A} conforms to the graph G_Ψ . In the next section, we will show how to analyse the graph G_Ψ to infer whether the conjunct $\bigwedge_{i=1}^\ell \forall x q_i(x) \rightarrow \mathcal{P}_i(x)$ can also be satisfied.

4.2. The characterisation of the satisfiability of Ψ . Recall that Ψ is a sentence in normal form (3.1). For each $1 \leq i \leq \ell$, let the LP quantifier $\mathcal{P}_i(x)$ be:

$$\mathcal{P}_i(x) := \sum_{j=1}^{t_i} \lambda_{i,j} \cdot \#_y^{R_{i,j}(x,y)} [x \neq y] + \sum_{j=1}^{t'_i} \lambda'_{i,j} \cdot \#_y^{R'_{i,j}(x,y)} [x = y] \quad \otimes_i \quad \delta_i.$$

For a graph G , a vertex π in G and $1 \leq i \leq \ell$, we define the linear constraint $\mathcal{Q}_i^{G,\pi}$:

$$\mathcal{Q}_i^{G,\pi} := \sum_{j=1}^{t_i} \lambda_{i,j} \cdot \left(\sum_{\substack{(\eta', \pi') \in \mathcal{N}_G(\pi) \\ \text{and } R_{i,j}(x,y) \in \eta'}} z_{\eta', \pi'} \right) + \sum_{j=1}^{t'_i} \lambda'_{i,j} \cdot \chi_{i,j} \quad \otimes_i \quad \delta_i,$$

where $\chi_{i,j}$ is 1, if $R'_{i,j}(x, x)$ is in π and 0, otherwise.

The variables in $\mathcal{Q}_i^{G,\pi}$ are $z_{\eta', \pi'}$, for every $(\eta', \pi') \in \mathcal{K} \times \Pi$. Intuitively, each $z_{\eta', \pi'}$ represents the number of (η', π') -neighbours of an element with type π . Since every element and every pair of elements has a unique unary and binary type, we can partition the neighbourhood of each element according to the unary and binary types. This is the reason

each $\#_y^{R_{i,j}(x,y)}[x \neq y]$ is replaced with the sum:

$$\sum_{\substack{(\eta', \pi') \in \mathcal{N}_G(\pi) \\ \text{and } R_{i,j}(x,y) \in \eta'}} z_{\eta', \pi'}.$$

The coefficient $\chi_{i,j}$ indicates whether an element with type π has a $R'_{i,j}$ -loop to itself. We formalise this intuition in Lemma 4.4.

Lemma 4.4. *Let G be a graph and \mathcal{A} be a structure that conforms to G . Let $1 \leq i \leq \ell$. Suppose there is an element a in \mathcal{A} whose type is π and $\mathcal{A}, x/a \models \mathcal{P}_i(x)$. Then, $\mathcal{Q}_i^{G,\pi}$ admits a solution in \mathfrak{N}_∞ .*

Proof. Let G , \mathcal{A} , a and π be as in the hypothesis of the lemma. For every $(\eta', \pi') \in \mathcal{N}_G(\pi)$, let $\mathcal{D}_{\eta', \pi'}$ be the set of (η', π') -neighbours of a in \mathcal{A} .

Note that for every neighbour b of a , there is exactly one $(\eta', \pi') \in \mathcal{N}_G(\pi)$ where $b \in \mathcal{D}_{\eta', \pi'}$. In other words, the sets $\mathcal{D}_{\eta', \pi'}$'s partition the neighbourhood of a . Since $\mathcal{A}, x/a \models \mathcal{P}_i(x)$, by the semantics of the LP quantifier, the following holds.

$$\sum_{j=1}^{t_i} \lambda_{i,j} \cdot |R_{i,j}(x,y) \wedge x \neq y|_{\mathcal{A}}^{x/a} + \sum_{j=1}^{t'_i} \lambda'_{i,j} \cdot |R'_{i,j}(x,y) \wedge x = y|_{\mathcal{A}}^{x/a} \quad \otimes_i \quad \delta_i.$$

Observe also that:

$$|R_{i,j}(x,y) \wedge x \neq y|_{\mathcal{A}}^{x/a} = \sum_{\substack{(\eta', \pi') \in \mathcal{N}_G(\pi) \\ \text{and } R_{i,j}(x,y) \in \eta'}} |\mathcal{D}_{\eta', \pi'}|.$$

Furthermore, each $|R'_{i,j}(x,y) \wedge x = y|_{\mathcal{A}}^{x/a}$ is 1, if $R'_{i,j}(x,x)$ is in π ; and 0, otherwise. Hence, $|R'_{i,j}(x,y) \wedge x = y|_{\mathcal{A}}^{x/a}$ is precisely the definition of $\chi_{i,j}$. Thus, the assignment $z_{\eta', \pi'} \mapsto |\mathcal{D}_{\eta', \pi'}|$ for each $(\eta', \pi') \in \mathcal{N}_G(\pi)$ is a solution to the linear constraint $\mathcal{Q}_i^{G,\pi}$. \square

Next, we define a system of linear constraints that captures whether a certain configuration can be realised.

Definition 4.5. For an edge (π_1, η, π_2) in a graph G , let $\mathcal{Z}_{\pi_1, \eta, \pi_2}^G$ be the following system of linear constraints:

$$z_{\eta, \pi_2} \geq 1 \wedge \bigwedge_{i \text{ s.t. } q_i(x) \in \pi_1} \mathcal{Q}_i^{G, \pi_1}$$

Note that $\mathcal{Z}_{\pi_1, \eta, \pi_2}^G$ contains only the linear constraint \mathcal{Q}_i^{G, π_1} when the unary predicate $q_i(x)$ belongs to π_1 . The intuitive meaning of $\mathcal{Z}_{\pi_1, \eta, \pi_2}^G$ is as follows. If it does not admit a solution in \mathfrak{N}_∞ , then the configuration (π_1, η, π_2) is not realised in any model of Ψ . This is because either z_{η, π_2} must be zero, or \mathcal{Q}_i^{G, π_1} is violated for some i where $q_i(x) \in \pi_1$. Its formalisation is stated as Lemma 4.6.

Lemma 4.6. *Let G be a graph and \mathcal{A} be a structure that conforms to G . Suppose there is a pair (a, b) in \mathcal{A} whose configuration is (π_1, η, π_2) and that $\mathcal{A}, x/a \models \bigwedge_{i=1}^{\ell} (q_i(x) \rightarrow \mathcal{P}_i(x))$. Then, the system $\mathcal{Z}_{\pi_1, \eta, \pi_2}^G$ admits a solution in \mathfrak{N}_∞ .*

Proof. Let G, \mathcal{A}, a, b and (π_1, η, π_2) be as in the hypothesis. Using the same notation in Lemma 4.4, let $\mathcal{D}_{\eta', \pi'}$ denote the set of (η', π') -neighbours of a , for every $(\eta', \pi') \in \mathcal{K} \times \Pi$. Since the configuration of (a, b) is (π_1, η, π_2) , we have $\mathcal{D}_{\eta, \pi_2} \neq \emptyset$ and hence, $|\mathcal{D}_{\eta, \pi_2}| \geq 1$. Since $\mathcal{A}, x/a \models \bigwedge_{i=1}^{\ell} q_i(x) \rightarrow \mathcal{P}_i(x)$, it is immediate that the assignment $z_{\eta', \pi'} \mapsto |\mathcal{D}_{\eta', \pi'}|$ is a solution to the system $\mathcal{Z}_{\pi_1, \eta, \pi_2}^G$. \square

To infer the satisfiability of Ψ from the graph G_Ψ , we will need some more terminology.

Definition 4.7. An edge (π_1, η, π_2) is a *bad edge* in a graph G (w.r.t. the sentence Ψ), if the system $\mathcal{Z}_{\pi_1, \eta, \pi_2}^G$ does not admit a solution in \mathfrak{N}_∞ .

Note that by Lemma 4.6, if (π_1, η, π_2) is a bad edge in G , then there is no model \mathcal{A} that conforms to G such that the configuration (π_1, η, π_2) is realized in \mathcal{A} and that $\mathcal{A} \models \forall x (q_i(x) \rightarrow \mathcal{P}_i(x))$ for every $1 \leq i \leq \ell$.

Next, we define its analogue for the vertices in G .

Definition 4.8. A vertex π is a *bad vertex* in G (w.r.t. the sentence Ψ), if all of the following conditions hold.

- It does not have any outgoing edge in G .
- There is $1 \leq i \leq \ell$ such that π contains $q_i(x)$, but the system $\mathcal{Q}_i^{G, \pi}$ does not admit the zero solution, i.e., the solution where all the variables are assigned with zero.

The intended meaning of a bad vertex π is that there cannot be a model \mathcal{A} of Ψ in which π is realised. We are now ready for the final terminology which will be crucial for deciding the satisfiability of Ψ .

Definition 4.9. Let G be a graph and H be a non-empty subgraph of G . We say that H is a *good subgraph* of G , if all of the following conditions hold.

- There is no bad vertex and no bad edge in H (w.r.t. the sentence Ψ).
- It is symmetric in the sense that (π, η, π') is an edge in H if and only if $(\pi', \bar{\eta}, \pi)$ is an edge in H .

Theorem 4.10 states that the satisfiability of Ψ is equivalent to the existence of a good subgraph in G_Ψ .

Theorem 4.10. *Let Ψ be a GP^2 sentence in normal form (3.1). Then, Ψ is satisfiable if and only if there is a good subgraph in G_Ψ .*

Proof. (only if) Let $\mathcal{A} \models \Psi$. Let H be the graph where the vertices are the unary types realised in \mathcal{A} and the edges are the configurations realised in \mathcal{A} . Obviously, H is symmetric and a non-empty subgraph of G_Ψ such that \mathcal{A} conforms to H . It remains to show that there is no bad edge and no bad vertex in H .

Let (π_1, η, π_2) be an edge in H . By definition, there is a pair (a, b) in \mathcal{A} whose configuration is (π_1, η, π_2) . Since $\mathcal{A} \models \Psi$, we have $\mathcal{A}, x/a \models \bigwedge_{i=1}^{\ell} q_i(x) \rightarrow \mathcal{P}_i(x)$. By Lemma 4.6, the system $\mathcal{Z}_{\pi_1, \eta, \pi_2}^H$ admits a solution in \mathfrak{N}_∞ , and hence, by definition, (π_1, η, π_2) is not a bad edge in H .

Next, we show that H does not contain any bad vertex. Assume to the contrary that there is a bad vertex π in H . By definition, π does not have any outgoing edge in H . By the construction of H , there is an element a in \mathcal{A} whose type is π and for every relation $R_{i,j}(x, y)$, we have $|R_{i,j}(x, y) \wedge x \neq y|_{\mathcal{A}}^{x/a} = 0$. Since $\mathcal{A} \models \Psi$, we have $\mathcal{A}, x/a \models q_i(x) \rightarrow \mathcal{P}_i(x)$

for every $1 \leq i \leq \ell$. In particular, for every $1 \leq i \leq \ell$ such that π contains $q_i(x)$, the system $\mathcal{Q}_i^{H,\pi}$ admits the zero solution. This contradicts the assumption that π is a bad vertex.

(if) Let $H = (V, E)$ be a good subgraph of G_Ψ . We will show how to construct a model $\mathcal{A} \models \Psi$ that conforms to H . Let Ψ be a GP² sentence as in Eq. (3.1). We consider two cases.

Case 1: There is a vertex $\pi \in V$ that does not have any outgoing edge in H .

Let \mathcal{A} be the structure that contains only one element whose unary type is π . Since H is a subgraph of G_Ψ , the unary type π is compatible with Ψ , i.e., $\pi(x) \models \gamma(x)$. Hence, \mathcal{A} satisfies the part $\forall x \gamma(x)$. Since \mathcal{A} contains only one element, it trivially satisfies the part $\bigwedge_{i=1}^k \forall x \forall y \alpha_i(x, y)$.

We now show that \mathcal{A} satisfies the third part $\bigwedge_{i=1}^\ell \forall x q_i(x) \rightarrow \mathcal{P}_i(x)$. Note that π is not a bad vertex since H is a good subgraph of G_Ψ . Since π does not have any outgoing edge, by definition, for every $1 \leq i \leq \ell$, where π contains $q_i(x)$, the system $\mathcal{Q}_i^{G,\pi}$ admits the zero solution. Since there is only one element in the structure \mathcal{A} , it has no neighbours and hence, the structure \mathcal{A} satisfies the part $\bigwedge_{1 \leq i \leq \ell} \forall x q_i(x) \rightarrow \mathcal{P}_i(x)$.

Case 2: Every vertex in V has at least one outgoing edge in H .

We will build a tree-like model structure \mathcal{A} that satisfies Ψ . In the construction of the tree, the term (η', π') -children of a node a means the children of a with unary type π' and binary type of (a, b) is η' .

The construction of \mathcal{A} is as follows. We pick a vertex $\pi \in V$ and start with a single element a and sets its unary type as π . We then pick an arbitrary outgoing edge (π, η, π_0) in H . Since H is a good subgraph of G_Ψ , the edge (π, η, π_0) is a good edge. So, by definition, the system $\mathcal{Z}_{\pi, \eta, \pi_0}^H$ admits a solution in \mathfrak{N}_∞ . Let $z_{\eta', \pi'} \mapsto M_{\eta', \pi'}$ for every η', π' be the solution. We add “fresh” elements b_1, b_2, \dots as the children of a such that for every η', π' , the number of (η', π') -neighbours of a is precisely $M_{\eta', \pi'}$.

We continue with the same process for the elements b_1, b_2, \dots . For each $j = 1, 2, \dots$, let π_j be the type of b_j and η_j be the type of (b_j, a) . Consider the system $\mathcal{Z}_{\pi_j, \eta_j, \pi}^H$ which admits a solution in \mathfrak{N}_∞ . Let $z_{\eta'', \pi''} \mapsto M_{\eta'', \pi''}$ for every η'', π'' be the solution. We add “fresh” elements $c_{j,1}, c_{j,2}, \dots$ as the children of b_j such that for every η'', π'' :

- If $(\eta'', \pi'') = (\eta_j, \pi_j)$, the (η'', π'') -children of b_j is precisely $M_{\eta'', \pi''} - 1$.
Here we need the term -1 since the parent node a is (η'', π'') -neighbour of b_j .
- If $(\eta'', \pi'') \neq (\eta_j, \pi_j)$, the (η'', π'') -children of b_j is precisely $M_{\eta'', \pi''}$.

We repeat the same process for the elements $c_{j,1}, c_{j,2}, \dots$ ad infinitum and obtain an infinite tree-like model \mathcal{A} . We now show \mathcal{A} satisfies Ψ . Note that H is a subgraph of G_Ψ . By the construction of G_Ψ , every unary type and configurations are compatible with Ψ . Since \mathcal{A} conforms to H , it is immediate that \mathcal{A} satisfies the part $\forall x \gamma(x)$, as well as the part $\bigwedge_{i=1}^k \forall x \forall y \alpha_i(x, y)$. That \mathcal{A} satisfies the part $\bigwedge_{i=1}^\ell \forall x q_i(x) \rightarrow \mathcal{P}_i(x)$ follows from the construction of the children of each element in \mathcal{A} . \square

4.3. The algorithm. Theorem 4.10 tells us that to decide if Ψ is satisfiable, it suffices to find if G_Ψ contains a good subgraph. It can be done as follows. First, construct the graph G_Ψ . Then, repeatedly delete the bad edges and bad vertices from G_Ψ . It stops when there is no more bad edge or vertex to delete. If the graph ends up not containing any vertices, then Ψ is not satisfiable. If the graph still contains some vertices, then it is a good subgraph of G_Ψ and by Theorem 4.10, Ψ is satisfiable. Its formal presentation can be found

in Algorithm 4.11. It is worth noting that deleting a bad edge may yield a new bad edge, hence the while-loop.

Algorithm 4.11.

Input: A sentence Ψ in normal form (3.1).

Output: Accept if and only if Ψ is satisfiable.

- 1: $G := G_\Psi$.
- 2: **while** G has a bad edge **do**
- 3: Delete the bad edge and its inverse from G .
- 4: Delete all bad vertices (if there is any) from G .
- 5: **ACCEPT** if and only if G is not an empty graph.

4.4. **The complexity analysis of Algorithm 4.11.** We start with the following lemma.

Lemma 4.12. *Algorithm 4.11 can be implemented using a quantifier-free Presburger arithmetic solver as a black box and the number of calls to such solver is bounded by 2^{4n+8m} , where n and m is the number of unary and binary predicates in the input sentence Ψ .*

Proof. Note that there are 2^{n+m} unary types and 2^{2m} binary types. Here it is useful to recall that atoms such as $R(x, x)$ are considered unary predicates. Thus, the graph G_Ψ has at most 2^{2n+4m} edges. In each iteration we check whether each edge is a bad edge, hence, there is at most 2^{2n+4m} number of calls to the Presburger arithmetic solver. After each iteration there is at most one less edge, hence, the 2^{4n+8m} upper bound. \square

Next, we show that checking whether the system $\mathcal{Z}_{\pi_1, \eta, \pi_2}^G$ admits a solution in \mathfrak{N}_∞ can be done in non-deterministic polynomial time, and hence, in deterministic exponential time, in the length of the input Ψ .

Lemma 4.13. *For every edge (π_1, η, π_2) in G , checking whether the system $\mathcal{Z}_{\pi_1, \eta, \pi_2}^G$ admits a solution in \mathfrak{N}_∞ takes non-deterministic polynomial time in the length of the input sentence.*

Proof. Let Ψ be the input sentence in the normal form (3.1), where n and m are the number of unary and binary relation symbols in Ψ . Thus, there are 2^{n+m} unary types and 2^{2m} binary types. Moreover, the system $\mathcal{Z}_{\pi_1, \eta, \pi_2}^G$ contains at most $\ell + 1$ linear constraints and 2^{n+3m} variables.

Here we invoke Corollary 2.3 which states that there are constants c_1, c_2 such that if $\mathcal{Z}_{\pi_1, \eta, \pi_2}^G$ admits a solution in \mathfrak{N}_∞ , then the number of variables assigned with non-zero values is $c_1 \ell \log_2(c_2 \ell M)$ and each value is either ∞ or at most $c_1 \ell (\ell M)^{c_2 \ell}$, where M is the maximal non- ∞ constant in the system $\mathcal{Z}_{\pi_1, \eta, \pi_2}^G$. Note that the value $c_1 \ell (\ell M)^{c_2 \ell}$ takes only polynomially many bits in the length of Ψ .

We can thus design a non-deterministic polynomial time algorithm for checking the satisfiability of $\mathcal{Z}_{\pi_1, \eta, \pi_2}^G$: Guess the set of variables that are supposed to take non-zero values and their values and **ACCEPT** if and only if it is indeed a solution for $\mathcal{Z}_{\pi_1, \eta, \pi_2}^G$. \square

Since constructing the graph G_Ψ takes exponential time, combining it with Lemmas 4.12 and 4.13 implies the exponential time upper bound of Algorithm 4.11.

Theorem 4.14. *Algorithm 4.11 runs in exponential time in the length of the input sentence.*

We remark that the exponential time upper bound of Algorithm 4.11 also holds when the semantics of LP quantifiers is defined on the structure \mathfrak{N} . The proof is exactly the same. The only difference is we invoke Corollary 2.2 instead of Corollary 2.3.

4.5. Comparison with the approach by Bednarczyk and Fiuk [BF22]. An alternating polynomial space algorithm was proposed by Bednarczyk and Fiuk [BF22] for deciding the satisfiability of GP² sentences.⁴ Intuitively, it tries to build a tree-like model that satisfies the input sentence (if satisfiable). It starts by guessing the unary type of the root node and all the unary types of its children as well as the binary types of the edges connecting them. Using Theorem 2.1, the number of different *unary types* of the children is polynomially bounded (in the length of the input sentence).⁵ To verify that the LP quantifiers are satisfied, it guesses a solution that respects the guessed binary types of the edges connecting the root nodes and the children.

In comparison, our algorithm is a straightforward deterministic algorithm that can be implemented easily using available Presburger solvers as a black box. Nevertheless our algorithm is worst-case optimal, as even in the best case scenario it still requires exponential time to build the graph representation, in contrast to the tableaux based approaches whose performance can be efficient depending on the instances.

5. CONCLUDING REMARKS

In this paper we consider the extension of GF² with the local Presburger quantifiers which can express rich Presburger constraints while maintaining a deterministic exponential time upper bound. It captures various natural DLs with counting up to $\mathcal{ALCITH}^{\text{self}}$ and $\mathcal{ALCITHQ}$ without constant symbols. The proof is via a novel, yet simple algorithm, which is reminiscent of the type-elimination approach.

We note that the proof of Theorem 4.10 relies on the infinity of the model. We leave a similar characterization for the finite models for future work. It is worth noting that the finite satisfiability of GP² has been shown to be decidable in 3-NEXP [BOPT21], though the precise complexity is still not known.

ACKNOWLEDGMENT

We thank Bartosz Bednarczyk for his comments on the preliminary drafts of this work. We would like to thank the anonymous reviewers for their detailed and constructive feedback which greatly improve the presentation of this paper. This work was done when both authors were in National Taiwan University. We acknowledge generous financial support of Taiwan Ministry of Science and Technology under grant no. 109-2221-E-002-143-MY3.

⁴It is well known that the class of languages decidable by alternating polynomial space Turing machines is equivalent to the class of languages decidable by deterministic exponential time Turing machines [CKS81].

⁵Note that the bound only holds for the number of unary types of the children, not the number of children itself which can be exponential.

REFERENCES

- [ANvB98] H. Andr eka, I. N emeti, and J. van Benthem. Modal languages and bounded fragments of predicate logic. *J. Philos. Log.*, 27(3):217–274, 1998.
- [Baa17] F. Baader. A new description logic with set constraints and cardinality constraints on role successors. In *FroCoS*, 2017.
- [BBR20] F. Baader, B. Bednarczyk, and S. Rudolph. Satisfiability and query answering in description logics with global and local cardinality constraints. In *ECAI*, 2020.
- [BCM⁺03] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
- [Bed20] B. Bednarczyk. One-variable logic meets presburger arithmetic. *Theor. Comput. Sci.*, 802:141–146, 2020.
- [BF22] B. Bednarczyk and O. Fiuk. Presburger b uchi tree automata with applications to logics with expressive counting. In *WoLLIC*, 2022.
- [BHLS17] F. Baader, I. Horrocks, C. Lutz, and U. Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.
- [BKT20] M. Benedikt, E. Kostylev, and T. Tan. Two variable logic with ultimately periodic counting. In *ICALP*, 2020.
- [BOPT21] B. Bednarczyk, M. Orlowska, A. Pacanowska, and T. Tan. On classical decidable logics extended with percentage quantifiers and arithmetics. In *FSTTCS*, 2021.
- [CKS81] A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.
- [DL10] S. Demri and D. Lugiez. Complexity of modal logics with presburger constraints. *J. Appl. Log.*, 2010.
- [ES06] F. Eisenbrand and G. Shmonin. Carath odory bounds for integer cones. *Oper. Res. Lett.*, 34(5):564–568, 2006.
- [GKV97] E. Gr adel, P. Kolaitis, and M. Vardi. On the decision problem for two-variable first-order logic. *Bull. Symbolic Logic*, 3(1):53–69, 03 1997.
- [Gr 98] E. Gr adel. Description logics and guarded fragments of first order logic. In *DL*, 1998.
- [Gr 99] E. Gr adel. On the restraining power of guards. *J. Symb. Log.*, 64(4):1719–1742, 1999.
- [HaAPV91] H. Herre, M. Krynicki and A. Pinus, and J. V an anen. The h artig quantifier: a survey. *Journal of Symbolic Logic*, 56:1153–1183, 1991.
- [H r62] K. H artig. Uber einen quantifikator mit zwei wirkungsbereichen, 1962.
- [Kaz04] Y. Kazakov. A polynomial translation from the two-variable guarded fragment with number restrictions to the guarded fragment. In *JELIA*, 2004.
- [KP10] C. Kupke and D. Pattinson. On modal logics of linear inequalities. In *AiML*, 2010.
- [KR07] V. Kuncak and M. Rinard. Towards efficient satisfiability checking for boolean algebra with presburger arithmetic. In *CADE*, 2007.
- [KT15] E. Kopczynski and T. Tan. Regular graphs and the spectra of two-variable logic with counting. *SIAM J. Comput.*, 44(3):786–818, 2015.
- [LLT21] T. Lin, C. Lu, and T. Tan. Towards a more efficient approach for the satisfiability of two-variable logic. In *LICS*, 2021.
- [Pap81] C. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981.
- [Pra79] V. R. Pratt. Models of program logics. In *FOCS*, 1979.
- [Pra05] I. Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information*, 14(3):369–395, 2005.
- [Pra07] I. Pratt-Hartmann. Complexity of the guarded two-variable fragment with counting quantifiers. *J. Log. Comput.*, 17(1):133–155, 2007.
- [Pra15] I. Pratt-Hartmann. The two-variable fragment with counting and equivalence. *Math. Log. Q.*, 61(6):474–515, 2015.
- [Tob01] S. Tobies. *Complexity results and practical algorithms for logics in knowledge representation*. PhD thesis, RWTH Aachen University, Germany, 2001.
- [Var96] M. Vardi. Why is modal logic so robustly decidable? In *DIMACS Workshop*, 1996.