Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Presburger Functional Synthesis: Complexity and Tractable Normal Forms

S. Akshay[1] , **A. R. Balasubramanian**[2] , **Supratik Chakraborty**[1] , **Georg Zetzsche**[2]
[1]Indian Institute of Technology Bombay, Mumbai, India
[2]Max Planck Institute for Software Systems (MPI-SWS), Germany
{akshayss,supratik}@cse.iitb.ac.in, {bayikudi,georg}@mpi-sws.org

Abstract

Given a relational specification between inputs and outputs as a logic formula, the problem of functional synthesis is to automatically synthesize a function from inputs to outputs satisfying the relation. Recently, a rich line of work has emerged tackling this problem for specifications in different theories, from Boolean to general first-order logic. In this paper, we launch an investigation of this problem for the theory of Presburger Arithmetic, that we call Presburger Functional Synthesis (PFnS). We show that PFnS can be solved in EXPTIME and provide a matching exponential lower bound. This is unlike the case for Boolean functional synthesis (BFnS), where only conditional exponential lower bounds are known. Further, we show that PFnS for one input and one output variable is as hard as BFnS in general. We then identify a special normal form, called PSyNF, for the specification formula that guarantees poly-time and poly-size solvability of PFnS. We prove several properties of PSyNF, including how to check and compile to this form, and conditions under which any other form that guarantees poly-time solvability of PFnS can be compiled in poly-time to PSyNF. Finally, we identify a syntactic normal form that is easier to check but is exponentially less succinct than PSyNF.

1 Introduction

Automated synthesis, often described as a holy grail of computer science, deals with the problem of automatically generating correct functional implementations from relational specifications. Specifications are typically presented as relations, encoded as first-order logic (FOL) formulas over a set of free variables that are partitioned into inputs and outputs. The goal of automated functional synthesis is to synthesize a function from inputs to outputs such that for every valuation of the inputs, if it is possible to satisfy the specification, then the valuation of outputs produced by the function also satisfies it. The existence of such functions, also called *Skolem functions*, is well-known from the study of first-order logic (Enderton 1972; Huth and Ryan 2004). However, it is not always possible to obtain succinct representations or efficiently executable descriptions of Skolem functions (Chakraborty and Akshay 2022). This has motivated researchers to study the complexity of functional synthesis in different first-order theories, and investigate specific normal forms for specifications that enable efficient functional synthesis.

In the simplest setting of Boolean (or propositional) specifications, Boolean functional synthesis (henceforth called BFnS) has received significant attention in the recent past (John et al. 2015; Rabe and Seshia 2016; Fried, Tabajara, and Vardi 2016; Chakraborty et al. 2018; Golia, Roy, and Meel 2020; Akshay, Chakraborty, and Jain 2023; Lin, Tabajara, and Vardi 2024) among others. Even for this restricted class, functional synthesis cannot be done efficiently unless long-standing complexity theoretic conjectures are falsified (Akshay et al. 2021). Nevertheless, several practical techniques have been developed, including counter-example guided approaches (John et al. 2015; Akshay et al. 2021; Golia, Roy, and Meel 2020; Golia et al. 2021), input-output separation based approaches (Chakraborty et al. 2018), machine learning driven approaches (Golia, Roy, and Meel 2020; Golia et al. 2021), BDD and ZDD based approaches (Fried, Tabajara, and Vardi 2016; Lin, Tabajara, and Vardi 2022; Lin, Tabajara, and Vardi 2024). Researchers have also studied *knowledge representations* or normal forms for specifications that guarantee efficient BFnS (Akshay et al. 2021; Akshay et al. 2019; Akshay, Chakraborty, and Jain 2023; Akshay, Chakraborty, and Shah 2024), with (Shah et al. 2021) defining a form that precisely characterizes when BFnS can be solved in polynomial time and space.

Compared to BFnS, work on functional synthesis in theories beyond Boolean specifications has received far less attention, even though such theories are widely applicable in real-life specifications. One such important extension is to theories of linear arithmetic over reals and integers. The work of (Kuncak et al. 2010; Kuncak et al. 2013) deals with complete functional synthesis for quantifier-free linear real arithmetic (QF_LRA) and linear integer arithmetic (QF_LIA). Similarly, (Jiang 2009) goes beyond Boolean specifications, and points out that Skolem functions may not always be expressible as terms in the underlying theory of the specification, necessitating an extended vocabulary. For specifications in QF_LIA, (Fedyukovich and Gupta 2019; Fedyukovich, Gurfinkel, and Gupta 2019) build tools for synthesizing (or extracting) Skolem functions as terms.

In this paper, our goal is to study functional synthesis from specifications in Presburger arithmetic (PrA for short), that extends QF_LIA with modular constraints. PrA has been extensively studied in the literature (see (Haase 2018) for a survey) and admits multiple interpretations, including

geometric and logic-based interpretations; see, e.g. (Chistikov 2024). Recent work has shown significant improvements in the complexity of quantifier elimination for PrA; see e.g. (Haase et al. 2024; Chistikov, Mansutti, and Starchak 2024). Since PrA admits effective quantifier elimination, it follows from (Chakraborty and Akshay 2022) that for every PrA specification, Skolem functions for all outputs can be synthesized as halting Turing machines. Unfortunately, this does not give good complexity bounds on the time required to compute Skolem functions. Our focus in this paper is to fill this gap by providing *optimal complexity results for* PFnS *as well as normal forms for tractable synthesis*.

Before we proceed further, let us see an example of a PrA specification, and an instance of PFnS. Consider a factory with two machines M_1 and M_2. Suppose M_1 must pre-process newly arrived items before they are further processed by M_2. Suppose further that M_1 can start pre-processing an item at any integral time instant k (in appropriate time units), and takes one time unit for pre-processing. M_2, on the other hand, can start processing an item only at every 2nd unit of time, and takes one time unit to process. Suppose items $I_1, \ldots, I_n$ arrive at times $t_1, \ldots, t_n$ respectively, and we are told that the job schedule must satisfy three constraints. First, M_1 must finish pre-processing each item exactly 1 time unit before M_2 picks it up for processing; otherwise, the item risks being damaged while waiting for M_2. In general, this requires delaying the start-time of pre-processing I_i by δ_i (≥ 0) time units so that the end-time of pre-processing aligns with one time instant before $2r$, for some $r \in \mathbb{N}$. Second, the (pre-)processing windows for different items must not overlap. Third, the total weighted padded delay must not exceed a user-provided cap Δ, where the weight for item i is i. Formalizing the above constraints in PrA, we obtain the specification $\varphi \equiv \varphi_1 \wedge \varphi_2 \wedge \varphi_3$, where $\varphi_1 \equiv \bigwedge_{i=1}^{n} \big(t_i + \delta_i + 1 \equiv 1 \pmod 2\big)$, $\varphi_2 \equiv \bigwedge_{1 \leq i < j \leq n} \big((t_i + \delta_i + 1 < t_j + \delta_j) \vee (t_j + \delta_j + 1 < t_i + \delta_i)\big)$, $\varphi_3 \equiv \bigwedge_{i=1}^{n} (\delta_i \geq 0) \wedge \big(\sum_{i=1}^{n} i.\delta_i \leq \Delta\big)$. Here, $t_1, \ldots, t_n$ and Δ are input variables, while $\delta_1, \ldots, \delta_n$ are output variables. The *functional synthesis problem* then asks us to synthesize the delays, i.e., functions $f_1, \ldots, f_n$ that take $t_1, \ldots, t_n, \Delta$ as inputs and produce values of $\delta_1, \ldots, \delta_n$ such that φ is satisfied, whenever possible.

Our contributions. As a first step, we need a representation for Skolem functions, for which we propose *Presburger circuits*, constructed by composing basic Presburger "gates". We identify a (minimal) collection of these gates such that every Presburger-definable function (closely related to those defined in (Ibarra and Leininger 1981)) can be represented as a circuit made of these gates. Using Presburger circuits as representations for Skolem functions, we examine the complexity of PFnS and develop knowledge representations that make PFnS tractable. Our main contributions are:

1. We provide a *tight complexity-theoretic characterization* for PFnS. Specifically:
   (a) We show that for every PrA specification $\varphi(\bar{x}, \bar{y})$, we can construct in $\mathcal{O}\big(2^{|\varphi|^{\mathcal{O}(1)}}\big)$ time a Presburger circuit of size $\mathcal{O}\big(2^{|\varphi|^{\mathcal{O}(1)}}\big)$ that represents a Skolem function for $\bar{y}$. This exponential upper bound significantly improves upon earlier constructions (Cherniavsky 1976; Ibarra and Leininger 1981) for which we argue that the resulting Presburger circuits would be of at least triply- or even quadruply-exponential size, respectively.
   (b) We show that the exponential blow-up above is unavoidable, by exhibiting a family $(\mu_n)_{n \geq 0}$ of PrA specifications of size polynomial in n, such that any Presburger circuit for any Skolem function for μ_n must have size at least $2^{\Omega(n)}$. This unconditional lower bound for PFnS stands in contrast to the Boolean case (BFnS), where lower bounds are conditional on long-standing conjectures from complexity theory.
   (c) We show that PFnS from one-input-one-output specifications is already as hard as BFnS in general. As a corollary, unless NP $\subseteq$ P/poly, the size of Skolem functions for one-input-one-output specifications must grow super-polynomially in the size of the specification in the worst-case.
2. The above results imply that efficient PFnS algorithms do not exist, and so, we turn to *knowledge representations*, i.e., studying normal forms of PrA specifications that admit efficient Skolem function synthesis.
   (a) For one-output PrA specifications, we define the notion of *modulo-tameness*, and prove that every y-modulo tame specification $\varphi(\bar{x}, y)$ admits polynomial-time synthesis of Presburger circuits for a Skolem function.
   (b) We lift this to PrA specifications with multiple output variables, and provide a *semantic normal form* called PSyNF that enjoys the following properties:
      i. PSyNF is universal: every PrA specification can be compiled to PSyNF in worst-case exponential time (unavoidable by our hardness results above).
      ii. PSyNF is good for existential quantification and synthesis: Given any specification in PSyNF, we can effectively construct Presburger circuits for Skolem functions in time polynomial in the size of the specification. Additionally, we can also existentially quantify output variables in polytime.
      iii. PSyNF is effectively (in the sense of recursion theory) checkable, and with reasonable complexity: Given any PrA specification, deciding if it is in PSyNF is coNP-complete.
      As a byproduct of independent interest, we obtain that the (truth problem for the) $\exists^*\forall$ fragment of $\langle \mathbb{Z}; +, <, 0, 1\rangle$ is NP-complete.
      iv. PSyNF is optimal for one output: For every universal normal form of single-output PrA specifications that admits polynomial-time existential quantification of the output, we can compile formulas in that form to PSyNF in polynomial time.
   (c) We provide a *syntactic normal form* for PrA specifications, called PSySyNF, that is universal and efficiently checkable (in time linear in the size of the formula), but is exponentially less succinct than PSyNF.

Structure. The paper is organized as follows. In Section 2, we start with preliminaries and define the problem statement and representations in Section 3. Our main complexity results for PFnS are in Section 4. We present our semantic normal forms in Section 5 and syntactic forms in Section 6 and conclude in Section 7. Many proofs and details, omitted here due to space restrictions, can be found in the full version at (Akshay et al. 2025).

Related Work. A circuit representation similar to ours, but using a slightly different set of gates, was (implicitly) studied in (Ibarra and Leininger 1981, Theorem 6) in the context of representing Presburger-definable functions. However, their formalism is closely tied to the setting of natural numbers, making it somewhat cumbersome in the setting of integers, for which our circuit representation appears more natural. In addition, specialized programming languages for describing Presburger-definable functions have been studied in the literature, examples being SL (Gurari and Ibarra 1981) and L_+ (Cherniavsky 1976), among others. However, because of the loopy nature of these programming languages, such programs do not guarantee as efficient evaluation of the functions as circuits do.

The problem of functional synthesis is intimately related to that of quantifier elimination, and our work leverages recent advances in quantifier elimination for PrA (Haase et al. 2024; Chistikov, Mansutti, and Starchak 2024). However, being able to effectively eliminate quantifiers does not automatically yield an algorithm for synthesizing Presburger circuits. Hence, although our work bootstraps on recent results in quantifier elimination for PrA, and draws inspiration from knowledge representation for Boolean functional synthesis, the core techniques for synthesizing Skolem functions are new. In fact, our knowledge compilation results yield a new alternative approach to quantifier elimination from PrA formulas, that can result in sub-exponential (even polynomial) blow-up in the size of the original formula, if the formula is in a special form. This is in contrast to state-of-the-art quantifier elimination techniques (Haase et al. 2024; Chistikov, Mansutti, and Starchak 2024) that always yield an exponential blow-up.

2 Preliminaries

Presburger Arithmetic: Presburger arithmetic (PrA) is the first-order theory of the structure $\langle \mathbb{Z}, +, <, 0, 1\rangle$. Presburger arithmetic is well-known to admit quantifier elimination, as originally shown by Mojżesz Presburger in 1929 (Presburger 1929) (see (Haase 2018) for a modern survey). That is, every formula in PA with quantifiers can be converted into an equivalent one without quantifiers, at the cost of introducing *modulo constraints*, which are constraints of the form $\sum_{i=1}^{n} a_i x_i \equiv r \pmod M$, where $x_1, \ldots, x_n$ are variables, and $a_1, \ldots, a_n, r, M$ are integer constants with $0 \leq r < M$. The constraint $\sum_{i=1}^{n} a_i x_i \equiv r \pmod M$ is semantically equivalent to $\exists k \in \mathbb{Z} : \sum_{i=1}^{n} a_i x_i = kM + r$. We say M is the modulus of the constraint, and r its residue. For notational convenience, we sometimes use $\sum_{i=1}^{m} a_i x_i \equiv_M r$ for $\sum_{i=1}^{m} a_i x_i \equiv r \pmod M$. Hence, technically, we are working over the structure $\langle \mathbb{Z}, +, <, (\equiv_M)_{M\in\mathbb{Z}}, 0, 1\rangle$. For

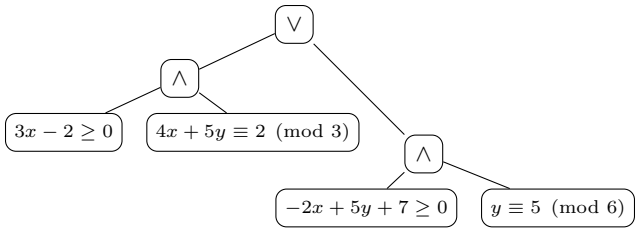

Figure 1: Tree representation of a Presburger formula

variables $\bar{x} = (x_1, \ldots, x_n)$ and vectors $\bar{r} = (r_1, \ldots, r_n)$ of constants $r_1, \ldots, r_n \in [0, M-1]$, we will use the shorthand $\bar{x} \equiv \bar{r} \pmod M$ to mean $\bigwedge_{i=1}^{n} x_i \equiv r_i \pmod M$.

A linear inequality is a formula of the form $a_1x_1 + \cdots + a_nx_n + b \geq 0$ (or equivalently, $a_1x_1 + \cdots + a_nx_n + b + 1 > 0$) for variables $x_1, \ldots, x_n$ and constants $a_1, \ldots, a_n, b \in \mathbb{Z}$. An atomic formula is either a linear inequality or a modulo constraint. Note that every quantifier-free formula over $\langle \mathbb{Z}, +, <, (\equiv_M)_{M\in\mathbb{Z}}, 0, 1\rangle$ is simply a Boolean combination of atomic formulas. Throughout this paper, we assume that all constants appearing in formulas are encoded in binary. We use variables with a bar at the top, viz. $\bar{x}$, to denote a tuple of variables, such as $(x_1, \ldots, x_n)$. With abuse of notation, we also use $\bar{x}$ to denote the underlying set of variables, when there is no confusion.

A quantifier-free PrA formula $\varphi(\bar{x})$ is said to be in *negation normal form (NNF)* if no sub-formulas other than atomic sub-formulas, are negated in φ. By applying DeMorgan's rules, a quantifier-free PrA formula can be converted to NNF in time linear in the size of the formula. Therefore, we assume all quantifier-free PrA formulas are in NNF. We represent such a formula as a *tree* in which each internal node is labeled by $\wedge$ or $\vee$, and each leaf is labeled by a linear inequality of the form $\sum_{k=1}^{n} a_k x_k + b \geq 0$, or by a modulo constraint of the form $\sum_{k=1}^{n} a_k x_k \bowtie r \pmod M$, where $\bowtie \in \{\equiv, \not\equiv\}$, $a_1, \ldots, a_n$, b, r and M are integers, and $0 \leq r < M$. We identify every node v in the tree with the sub-formula of φ represented by the sub-tree rooted at v. Specifically, the root of the tree is identified with the formula φ. The *size* of a quantifier-free PrA formula φ, denoted $|\varphi|$, is the sum of the number of nodes in the tree representation of φ, the number of variables, and the number of bits needed to encode each constant in the atomic formulas in the leaves. As an example, Fig. 1 shows a tree representing the formula $\big((3x - 2 \geq 0) \wedge (4x + 5y \equiv 2 \pmod 3)\big) \vee \big((-2x + 5y + 7 \geq 0) \wedge (y \equiv 5 \pmod 6)\big)$. This tree has 7 nodes, 2 variables and uses 37 bits to represent all constants in the atomic formulas in binary. binary.

3 Presburger Functional Synthesis

The central problem in this paper is *Presburger functional synthesis* (PFnS). Intuitively, we have a tuple of input variables $\bar{x}$ and a tuple of output variables $\bar{y}$, with each variable ranging over $\mathbb{Z}$. In addition, we are also given a quantifier-free PrA formula $\varphi(\bar{x}, \bar{y})$ that we interpret as a relational specification between the inputs and outputs. Our task is to find (and represent) a function f with inputs $\bar{x}$ and outputs

$\bar{y}$ such that the specification φ is satisfied by this function, whenever possible. Such a function is called a *Skolem function*. More formally:

Definition 3.1. *Let $\varphi(\bar{x}, \bar{y})$ be a quantifier-free* PrA *formula, where $\bar{x}$ denotes $(x_1, \ldots, x_n)$ and $\bar{y}$ denotes $(y_1, \ldots, y_m)$. A function $f\colon \mathbb{Z}^n \to \mathbb{Z}^m$ is called a* Skolem function for the existentially quantified variables ($\bar{y}$) in $\forall\bar{x}\exists\bar{y}\colon \varphi(\bar{x}, \bar{y})$ *if for every value $\bar{u} \in \mathbb{Z}^n$ of $\bar{x}$, $\exists\bar{y}\colon \varphi(\bar{u}, \bar{y})$ holds if and only if $\varphi(\bar{u}, f(\bar{u}))$ holds.*

A syntax for Skolem functions Since our goal is to synthesize Skolem functions, we need a syntax to represent them. We introduce such a syntax, called *Presburger circuits*, which are a variant of a syntax studied implicitly by Ibarra and Leininger (1981). The notion of Presburger circuits is designed to achieve two key properties:

1. Efficient evaluation: Given a Presburger circuit for a function $f\colon \mathbb{Z}^n \to \mathbb{Z}^m$ and a vector $\bar{u} \in \mathbb{Z}^n$, one can compute $f(\bar{u})$ in polynomial time.
2. Completeness: Every Presburger formula has a Skolem function defined by some Presburger circuit.

Let us describe Presburger circuits in detail. A Presburger circuit consists of a set of *gates*, each of which computes a function from a set of *atomic functions*. The atomic functions are

1. linear functions with integer coefficients, i.e. $\mathbb{Z}^n \to \mathbb{Z}$, $(u_1, \ldots, u_n) \mapsto a_0 + \sum_{i=1}^{n} a_i u_i$ for $a_0, \ldots, a_n \in \mathbb{Z}$.
2. the maximum function $\max\colon \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$.
3. the equality check function, i.e. $\mathsf{E}\colon \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ with $\mathsf{E}(x, y) = y$ if $x = 0$ and $\mathsf{E}(x, y) = 0$ if $x \neq 0$.
4. division functions $\mathsf{div}_m\colon x \mapsto \lfloor x/m \rfloor$ for $m \in \mathbb{N} \setminus \{0\}$.

More formally, a *Presburger circuit* is a collection of gates, each labeled either with an atomic function or with an input variable x_i, $i = 1, \ldots, n$. If a gate g is labeled by an atomic function $f\colon \mathbb{Z}^k \to \mathbb{Z}$, then there are n edges $e_1, \ldots, e_k$, each connecting some gate g_i to g. Intuitively, these edges provide the inputs to the gate g. Hence, $g_1, \ldots, g_k$ are called the *input gates* of g. Finally, there is a list of m distinguished *output gates* $g_{o,1}, \ldots, g_{o,m}$ that compute the output vector $\in \mathbb{Z}^m$ of the Presburger circuit.

A Presburger circuit must be *acyclic*, meaning the edges between gates form no cycle. This acyclicity allows us to evaluate a Presburger circuit for a given input $(u_1, \ldots, u_n)$: First, the gates labeled by input variables evaluate to the respective values. Then, a gate labeled with an atomic function $f\colon \mathbb{Z}^n \to \mathbb{Z}$ evaluates to $f(u_1, \ldots, u_n)$, where u_i is the result of evaluating the i-th input gate of g. Finally, the *output* of the Presburger circuit is the output (i.e. evaluation result) of the distinguished output gates. Overall, the Presburger circuit computes a function $\mathbb{Z}^n \to \mathbb{Z}^m$.

To simplify terminology, Presburger circuits that compute Skolem functions will also be called *Skolem circuits*.

Properties of Presburger circuits First, it is obvious that a Presburger circuit can be evaluated in polynomial time. Moreover, we will show that Presburger circuits are expressively complete for Skolem functions in Presburger arithmetic. Indeed, the following is a direct consequence of Theorem 4.1, which will be shown in Section 4:

Theorem 3.2. *For every quantifier-free formula $\varphi(\bar{x}, \bar{y})$, there exists a Skolem circuit for the existentially quantified variables in $\forall\bar{x}\exists\bar{y}\colon \varphi(\bar{x}, \bar{y})$.*

Equivalently, Presburger circuits describe exactly those functions that can be defined in Presburger arithmetic. Formally, a function $f\colon \mathbb{Z}^n \to \mathbb{Z}^m$ is *Presburger-definable* if there exists a Presburger formula $\varphi(\bar{x}, \bar{y})$, $\bar{x} = (x_1, \ldots, x_n)$, $\bar{y} = (y_1, \ldots, y_m)$, such that for all $\bar{u} \in \mathbb{Z}^n$ and $\bar{v} \in \mathbb{Z}^m$, we have $\varphi(\bar{u}, \bar{v})$ if and only if $f(\bar{u}) = \bar{v}$. The following is an alternative characterization:

Theorem 3.3. *A function $\mathbb{Z}^n \to \mathbb{Z}^m$ is computable by a Presburger circuit if and only if it is Presburger-definable.*

Note that Theorem 3.3 follows directly from Theorem 3.2: If a function $f\colon \mathbb{Z}^n \to \mathbb{Z}^m$ is Presburger-definable by some Presburger formula $\varphi(\bar{x}, \bar{y})$, then clearly f is the only possible Skolem function in $\forall\bar{x}\exists\bar{y}\colon \varphi(\bar{x}, \bar{y})$. Hence, the circuit provided by Theorem 3.2 must compute f. Conversely, given a Presburger circuit $\mathcal{C}$, it is easy to construct a Presburger formula that defines the function $\mathcal{C}$ computes.

Remark 3.4. In the full version (Akshay et al. 2025), we also show that if we restrict the division functions to those of the form div_p for primes p, then (i) one can still express the same functions and (ii) the set of atomic functions is *minimal*. This means, removing any of the functions $\max$, E, or div_p will result in some Presburger-definable function being not representable as a Presburger circuit.

Presburger functional synthesis, formally We are ready to state our main problem of interest. *Presburger functional synthesis* (PFnS) is the following task:

Given A quantifier-free Presburger formula $\varphi(\bar{x}, \bar{y})$ representing a relational specification between $\bar{x}$ and $\bar{y}$.

Output A Presburger circuit $\mathcal{C}$ that computes a Skolem function for the existentially quantified variables in $\forall\bar{x}\exists\bar{y}\colon \varphi(\bar{x}, \bar{y})$.

Intuitively, for every possible value $\bar{u} \in \mathbb{Z}^n$ of $\bar{x}$, a Skolem circuit $\mathcal{C}$ produces $\mathcal{C}(\bar{u}) \in \mathbb{Z}^m$ with the following guarantee: The relational specification $\varphi(\bar{u}, \mathcal{C}(\bar{u}))$ is true iff there is some $\bar{v} \in \mathbb{Z}^m$ for which $\varphi(\bar{u}, \bar{v})$ is true. Hence, the value of $\mathcal{C}(\bar{u})$ matters only when $\exists\bar{y} : \varphi(\bar{u}, \bar{y})$ holds. If, however, there is no $\bar{v} \in \mathbb{Z}^m$ with $\varphi(\bar{u}, \bar{v})$, then any value produced by $\mathcal{C}(\bar{u})$ is fine.

Remark 3.5. Every Presburger specification admits a Presburger-definable function as a Skolem function.

See the full version (Akshay et al. 2025) for a proof.

4 Presburger Functional Synthesis for General Formulas

In this section, we consider Presburger functional synthesis for arbitrary quantifier-free relational specifications. Our main results here are an exponential upper bound, as well as an exponential lower bound.

An exponential upper bound Our first result is an exponential upper bound for Presburger functional synthesis.

Theorem 4.1. *Given a quantifier-free formula $\varphi(\bar{x}, \bar{y})$, there exists a Skolem circuit for the existentially quantified variables in $\forall\bar{x}\exists\bar{y}\colon \varphi(\bar{x}, \bar{y})$. Moreover, this circuit can be constructed in time $2^{|\varphi|^{\mathcal{O}(1)}}$.*

This exponential upper bound result significantly improves on existing methods related to constructing Presburger Skolem functions. The two related lines of work that we are aware of, namely, Presburger functions defined by Ibarra and Leininger (Ibarra and Leininger 1981) and translation of Presburger-definable functions into L_+-programs by Cherniavsky (Cherniavsky 1976, Thm. 5) would yield, respectively, *quadruply-exponential* and *triply-exponential* upper bounds. See the full version (Akshay et al. 2025) for an analysis.

Theorem 4.1 can also be deduced from our normal form results, presented in later sections (i.e. by using Theorem 5.3 and either Theorem 5.4, or Theorem 6.2). However, we find it instructive to provide a direct proof without conversion into normal forms.

We now present a sketch of the construction of Theorem 4.1. The details can be found in the full version (Akshay et al. 2025). The crux of our approach is to use the geometric insight underlying a recent quantifier elimination technique in (Haase et al. 2024). This geometric insight refines solution bounds to systems $A\bar{x} \leq \bar{b}$ of linear inequalities. Standard bounds provide a solution that is small compared to $\|A\|$ and $\|\bar{b}\|$. The bound from (Haase et al. 2024) even applies when $\|\bar{b}\|$ itself cannot be considered small. Instead, the result provides a solution that is "not far from $\bar{b}$": The solution can be expressed as an affine transformation of $\bar{b}$ with small coefficients. To state the result, we need some notation. For a rational number $r \in \mathbb{Q}$, its *fractional norm* $\|r\|_{\mathsf{frac}}$ is defined as $|a| + |b|$, where $\frac{a}{b} = r$ is the unique representation with co-prime a, b. The fractional norm of vectors and matrices, written $\|A\|_{\mathsf{frac}}$ and $\|\bar{x}\|_{\mathsf{frac}}$, is then the maximum of the fractional norms of all entries. The geometric insight is the following, which appeared in (Haase et al. 2024, Prop. 4.1).

Proposition 4.2. *Let $A \in \mathbb{Z}^{\ell\times n}$ and $\bar{b} \in \mathbb{Z}^{\ell}$, and let Δ be an upper bound on the absolute values of the subdeterminants of A. If the system $A\bar{x} \leq \bar{b}$ has an integral solution, then it has an integral solution of the form $D\bar{b} + \bar{d}$, where $D \in \mathbb{Q}^{n\times\ell}$, $\bar{d} \in \mathbb{Q}^n$ with $\|D\|_{\mathsf{frac}}, \|\bar{d}\|_{\mathsf{frac}} \leq n\Delta^2$.*

Crucially, the bound $n\Delta^2$ only depends on A, not on $\bar{b}$. By the Hadamard bound for the determinant (Hadamard 1893), this means the number of bits in the description of D and $\bar{d}$ is polynomial in the number of bits in A.

Proof sketch of Theorem 4.1. (A detailed proof can be found in the full version (Akshay et al. 2025).) To apply Proposition 4.2, we first remove modulo constraints in φ, in favor of new output variables. For example, a constraint $x_1 \equiv a \bmod b$ is replaced with $x_1 = by' + a$, where y' is a fresh output variable. These new output variables can just be ignored in the end, to yield a circuit for the original formula. By bringing φ into DNF, we may assume that φ is a disjunction of r-many systems of inequalities $A_i\bar{y} \leq B_i\bar{x} + \bar{c}_i$. Here, r is at most exponential, and each A_i, B_i, and $\bar{c}_i$ has at most polynomially many bits.

Now for each $i \in [1, r]$, Proposition 4.2 yields s-many candidate pairs $(D_{i,j}, \bar{d}_{i,j})$ for solutions $\bar{y}$ to $A_i\bar{y} \leq B_i\bar{x} + \bar{c}_i$. Here, s is at most exponential, and we know that if the system has a solution for a given $\bar{x}$, then it has one of the form $D_{i,j}(B_i\bar{x} + \bar{c}_i) + \bar{d}_{i,j}$ for some $j = 1, \ldots, s$.

Our circuit works as follows. The idea is to try for each (i, j), in lexicographical order, whether $\sigma_{i,j}(\bar{x}) := D_{i,j}(B_i\bar{x} + \bar{c}_i) + \bar{d}_{i,j}$ is an integral solution to $A_i\bar{y} \leq B_i\bar{x} + \bar{c}_i$. In this case, let us say that (i, j) is a *solution*. If (i, j) is a solution, then our circuit outputs $\sigma_{i,j}(\bar{x})$. In order to check if (i, j) is a solution, we need to check two things: whether (a) $\sigma_{i,j}(\bar{x})$ is an integer vector and (b) whether it satisfies $A_i\sigma_{i,j}(\bar{x}) \leq B_i\bar{x} + \bar{c}_i$. Note that (a) is necessary because $D_{i,j}$ and $\bar{d}_{i,j}$ are over the rationals. However, we can check integrality of $\sigma_{i,j}(\bar{x})$ by way of div gates. To check (b), our circuit computes all entries of the vector $B_i\bar{x} + \bar{c}_i - A_i\sigma_{i,j}(\bar{x})$. Using summation, max, and E gates, it then computes the number of entries that are ≥ 0. If this number is exactly the dimension of the vector (which can be checked with an E gate), (i, j) is a solution.

To implement the lexicographic traversal of all (i, j), we have for each $(i, j) \in [r, s]$ a circuit that computes the function $F_{i,j}(\bar{x})$, which returns 1 if and only if (i) (i, j) is a solution, and (ii) for all (r, s) that are lexicographically smaller than (i, j), the pair (r, s) is not a solution. Based on this, we can compute the function $S_{i,j}(\bar{x})$, which returns $\sigma_{i,j}(\bar{x})$ if (i, j) is a solution, and zero otherwise. Note that $S_{i,j}(\bar{x})$ is non-zero for at most one pair (i, j). Finally, we define $f(\bar{x})$ to sum up $S_{i,j}(\bar{x})$ over all $(i, j) \in [1, r] \times [1, s]$. Then, f is clearly a Skolem function for φ. □

Remark 4.3. Our construction even yields a circuit of polynomial depth, and where all occurring coefficients (in linear combination gates) have at most polynomially many bits.

An exponential lower bound The second main result of this section is a matching exponential lower bound.

Theorem 4.4. *There are quantifier-free formulas $(\mu_n)_{n\geq 0}$ such that any Skolem circuit for μ_n has size at least $2^{\Omega(n)}$*

Let us point out that usually it is extremely difficult to prove lower bounds for the size of circuits. Indeed, proving an (unconditional) exponential lower bound for the size of circuits for Boolean functional synthesis is equivalent to one of the major open problems in complexity theory—whether the class NP is included in P/poly (which, in turn, is closely related to whether P equals NP):

Observation 4.5. *The following are equivalent: (i) Every Boolean relational specification φ has a Skolem function computed by a Boolean circuit of size polynomial in $|\varphi|$. (ii)* NP ⊆ P/poly.

Here, P/poly is the class of all problems solvable in polynomial time with a polynomial amount of advice (see e.g., (Arora and Barak 2009)). The implication "(i)⇒(ii)"

had been shown in (Akshay et al. 2021, Theorem 1). We prove "(ii)⇒(i)" in the full version (Akshay et al. 2025).

Nevertheless, we will prove an exponential lower bound for circuits for Presburger functional synthesis.

Proof sketch exponential lower bound For space reasons, we can only provide a rough sketch—with proof details in the full version (Akshay et al. 2025). Following a construction from (Haase et al. 2024, Section 6), we choose $\mu_n \equiv \forall x\colon \exists \bar{y}\colon \psi_n(x,\bar{y})$ so that the formula $\exists \bar{y}\colon \psi_n(x,\bar{y})$ defines a subset $S \subseteq \mathbb{Z}$ whose minimal period is doubly exponential. Here, the *minimal period* of S, is the smallest p such that for all but finitely many u, we have $u + p \in S$ if and only if $u \in S$. Moreover, we show that if μ_n had a Skolem function with a Presburger circuit $\mathcal{C}_n$ with e_n-many div-gates, and M is the least common multiple of all divisors occurring in that gate, then p must divide M^{e_n}. This proves that e_n is at least exponential, hence $\mathcal{C}_n$ must contain an exponential number of div-gates.

Hardness for one input, one output We now show that synthesizing Skolem functions for Presburger specifications with even just one input and one output variable is as hard as the general Boolean functional synthesis problem:

Observation 4.6. *Suppose every one-input one-output quantifier-free Presburger formula has a polynomial size Skolem circuit. Then every Boolean formula has a polynomial size Skolem circuit—impossible unless* $\mathsf{NP} \subseteq \mathsf{P/poly}$.

The proof uses the "Chinese Remaindering" technique, by which one can encode an assignment of n Boolean variables in a single integer: in the residues modulo the first n primes. See the full version (Akshay et al. 2025) for details.

5 Semantic Normal Form for PFnS

We now present a normal form for PrA specifications, called PSyNF, that guarantees efficient Skolem function synthesis. The normal form definition has two key ingredients, (i) modulo-tameness, and (ii) local quantification. Since the definition involves semantic conditions, we call it a *semantic normal form*.

Ingredient I: Modulo-tameness Recall that we represent quantifier-free PrA formulas as trees. A $\wedge$-labeled node in the tree representing φ is said to be a *maximal conjunction* if there are no $\wedge$-labeled ancestors of the node in the tree. A subformula is *maximal conjunctive* if it is the sub-formula rooted at some maximal conjunction.

Definition 5.1. *A quantifier-free* PrA *formula $\varphi(\bar{x}, y)$ is called* y-modulo-tame*, if it is in NNF, and for every maximal conjunctive sub-formula ψ of φ, there is an integer M^ψ such that all modulo constraints involving y in ψ are of the form $y \equiv r \pmod{M^\psi}$ for some $r \in [0, M^\psi - 1]$.*

Hence, the definition admits $y \equiv r_1 \pmod{M}$ and $y \equiv r_2 \pmod{M}$ in the same maximal conjunctive sub-formula, even if $r_1 \neq r_2$. It does not admit $y \equiv r_1 \pmod{M_1}$ and $y \equiv r_2 \pmod{M_2}$ in a maximal conjunctive sub-formula, if $M_1 \neq M_2$. The value of M can vary from one maximal conjunctive sub-formula to another; so the definition admits $y \equiv r_1 \pmod{M_1}$ and $y \equiv r_2 \pmod{M_2}$ in subtrees rooted at two different maximal conjunctions.

As an example, the formula represented in Fig. 1 is not y-modulo tame. This is because the maximal conjunctive sub-formula to the left of the root has the atomic formula $4x + 5y \equiv 2 \pmod 3$, which is not of the form $y \equiv r \pmod M$. If we replace $4x + 5y \equiv 2 \pmod 3$ by the semantically equivalent formula $\big((y \equiv 0 \pmod 3) \wedge 4x \equiv 2 \pmod 3) \vee (y \equiv 1 \pmod 3) \wedge 4x \equiv 0 \pmod 3) \vee (y \equiv 2 \pmod 3) \wedge 4x \equiv 1 \pmod 3)\big)$, then the new formula satisfies the condition of Definition 5.1. Hence, the resulting semantically equivalent formula is y-modulo tame.

Checking if a given formula $\varphi(\bar{x}, y)$ is y-modulo-tame is easy: look at each maximal conjunction in the tree representation of φ and check if all modulo constraints involving y are of the form $y \equiv r \pmod M$ for the same modulus M. Furthermore, this form is universal: any formula can be made y-modulo tame for any y.

Proposition 5.2. *Given a quantifier-free formula $\varphi(\bar{x}, y)$, let $\mathfrak{M}$ be the set of all moduli appearing in modular constraints involving y. We can construct an equivalent y-modulo-tame formula in $\mathcal{O}\big(|\varphi|.\big(\prod_{M\in\mathfrak{M}} M\big)\big)$ time.*

Since the moduli M's in φ are represented in binary, Proposition 5.2 implies an exponential blow-up in the formula size, when making it y-modulo tame. This blow-up is however unavoidable, by virtue of the hardness result in Observation 4.6 and a key result of this section (Theorem 5.7).

Ingredient II: Local quantification For PSyNF, we also need the concept of local quantification, which we introduce now. For a quantifier-free $\varphi(\bar{x}, y)$ in NNF and y-modulo-tame, we define $\exists^{\mathsf{local}} y\colon \varphi(\bar{x}, y)$ as the formula obtained by replacing each atomic subformula in φ that mentions y with $\top$. Clearly, $\exists y\colon \varphi(\bar{x}, y)$ implies $\exists^{\mathsf{local}} y\colon \varphi(\bar{x}, y)$.

Definition of PSyNF Suppose $\varphi(\bar{x}, \bar{y})$ is a quantifier-free Presburger formula in NNF with free variables $\bar{x} = (x_1, \ldots, x_n)$ and $\bar{y} = (y_1, \ldots, y_m)$. We define *$\varphi$ to be in* PSyNF *w.r.t. the ordering $y_1 \preceq \cdots \preceq y_m$*, if (i) φ is y_i-modulo-tame for each $i \in [1, m]$ and (ii) for every $i \in [1, m-1]$, the formula

$$\forall \bar{x} \forall y_1, \ldots, y_i\colon (\exists^{\mathsf{local}} y_{i+1}, \ldots, y_m\colon \varphi(\bar{x}, \bar{y}) \rightarrow \exists y_{i+1}, \ldots, y_m\colon \varphi(\bar{x}, \bar{y})) \quad (1)$$

denoted $\varphi^{(i)}$, holds. Note that the implication in the reverse direction holds trivially; hence $\varphi^{(i)}$ can be equivalently written using $\leftrightarrow$ in place of $\rightarrow$. In the following, we assume that every specification formula is annotated with an ordering on the output variables, and that PSyNF is w.r.t. that ordering.

To see an example of a PSyNF specification, consider a variant of the job scheduling problem discussed in Section 1. In this variant, we have only two items, and we require item 2 to be (pre-)processed before item 1. The variant specification is $\psi \equiv \psi_1 \wedge \psi_2 \wedge \psi_3$, where $\psi_1 \equiv \bigwedge_{i=1}^{2}(t_i + \delta_i + 1 \equiv 1 \pmod 2)$, $\psi_2 \equiv t_2 + \delta_2 + 1 < t_1 + \delta_1$ and $\psi_3 \equiv \bigwedge_{i=1}^{2}(\delta_i \geq 0) \wedge (\delta_1 + 2\delta_2 \leq \Delta)$. It is easy to see that ψ is not δ_i-modulo tame for any δ_i; hence it is not in

PSyNF. If we replace ψ_1 with the equivalent formula $\psi_1' \equiv \bigwedge_{i=1}^{2} \bigvee_{r=0}^{1} \big((\delta_i \equiv r \pmod 2) \wedge (t_i \equiv r \pmod 2)\big)$, the resulting specification $\psi' \equiv \psi_1' \wedge \psi_2 \wedge \psi_3$ is δ_i-modulo tame for $i \in \{1, 2\}$. However ψ' is not in PSyNF w.r.t. any ordering of δ_1, δ_2, since local quantifier elimination replaces the constraint $\delta_1 + 2\delta_2 \leq \Delta$ with $\top$, removing the cap on the cumulative weighted delay. To remedy this situation, consider $\psi'' \equiv \psi_1' \wedge \psi_2 \wedge \psi_3 \wedge (\psi_4 \vee \psi_5)$, where $\psi_4 \equiv (t_2 + \delta_2 + 1 < t_1) \wedge \bigvee_{r=0}^{1} \big((t_1 \equiv r \pmod 2) \wedge (2\delta_2 + r \leq \Delta)\big)$, and $\psi_5 \equiv (t_2 + \delta_2 + 1 \geq t_1) \wedge (t_2 - t_1 + 3\delta_2 + 2 \leq \Delta)$. It can be verified that ψ'' is semantically equivalent to ψ, and satisfies all conditions for PSyNF w.r.t. the order $\delta_1 \prec \delta_2$ (but not w.r.t. $\delta_2 \prec \delta_1$). Note that $(\psi_4 \vee \psi_5)$ constrains $t_1, t_2, \delta_2, \Delta$ in such a way that $(\psi_4 \vee \psi_5) \wedge \exists^{\mathsf{local}} \delta_1 \psi \leftrightarrow \exists \delta_1 \psi$ holds.

Main results about PSyNF. The first main result is that for formulas in PSyNF, we can easily solve PFnS.

Theorem 5.3. *Given a Presburger formula $\varphi(\bar{x}, \bar{y})$ in PSyNF, we can compute in time polynomial in the size of φ, a Skolem circuit for each existentially quantified variable in $\forall \bar{x} \exists \bar{y}\, \varphi(\bar{x}, \bar{y})$.*

Second, every formula has an equivalent in PSyNF, albeit with an exponential blow-up (unavoidable by Thm. 5.3, 4.4):

Theorem 5.4. *For every quantifier-free formula $\varphi(\bar{x}, \bar{y})$ and for every ordering of output variables y_i, there is an equivalent formula ψ in PSyNF w.r.t. the order, such that ψ is at most exponential in the size of φ.*

As a third important result, we show that checking whether a formula is in PSyNF has reasonable complexity:

Theorem 5.5. *Given a quantifier-free formula $\varphi(\bar{x}, \bar{y})$ in NNF, and an ordering of output variables, it is coNP-complete to decide whether φ is in PSyNF w.r.t. the order.*

Finally, we have a corollary of independent interest:

Corollary 5.6. *The truth problem for the $\exists^* \forall$ fragment of formulas over the structure $\langle \mathbb{Z}; +, <, 0, 1 \rangle$ is NP-complete.*

Here, the *truth problem* is to decide whether a given formula holds. In Corollary 5.6, it is crucial that the input formula is over the structure $\langle \mathbb{Z}; +, <, 0, 1 \rangle$, meaning it cannot contain modulo constraints. Indeed, a reduction similar to Observation 4.6 shows that with modulo constraints, even the $\exists\forall$ fragment is Σ_2^p-hard. Corollary 5.6 is somewhat surprising, since the $\forall\exists^*$ fragment of $\langle \mathbb{Z}; +, <, 0, 1 \rangle$ is coNEXP-complete (Haase 2014, Thm. 1) (the lower bound was already shown by Grädel (1989, Thm. 4.2)). Hence, in this setting, allowing an unbounded number of inner quantifiers is more expensive than allowing an unbounded number of outer quantifiers. Furthermore, Corollary 5.6 complements a result of Schöning (1997, Corollary), which states that the $\exists\forall$ fragment for the structure $\langle \mathbb{Z}; +, <, 0, 1 \rangle$ is NP-complete: Together, Corollary 5.6 and Schöning's result imply that for every $i \geq 1$, the fragment $\exists^i \forall$ is NP-complete.

The remainder of this section is devoted to proving Theorems 5.3 to 5.5 and Corollary 5.6. Of these proofs, Theorem 5.3 is the most involved. It is shown in two steps: First, we prove Theorem 5.3 in the case of one output variable (i.e. $m = 1$). Then, we show that this procedure can be used repeatedly to solve PFnS in general in polynomial time.

The case of one output We first prove Theorem 5.3 when $m = 1$. In this setting, PSyNF is equivalent to modulo-tameness w.r.t. the only output variable.

Theorem 5.7. *Let $\varphi(\bar{x}, y)$ be a y-modulo-tame quantifier-free PrA formula. A Skolem circuit for the existentially quantified variable in $\forall \bar{x} \exists y : \varphi(\bar{x}, y)$ can be computed in time polynomial in $|\varphi|$.*

Below, we give an outline of the proof of Theorem 5.7, leaving the details to the full version (Akshay et al. 2025).

Step I: Simplify modulo constraints We restrict ourselves to the case of φ being conjunctive (i.e. its top-most connective is a conjunction): else, one can first compute a Skolem circuit for each maximal conjunctive subformula, and then easily combine these circuits into a Skolem circuit for φ. Since φ is modulo-tame, there is an $M \in \mathbb{N}$ such that all modulo constraints on y in φ are of the form $y \equiv r \pmod M$ for some $r \in \mathbb{N}$. Let R denote the set of all such r; clearly, $|R| \leq |\varphi|$. Now, it suffices to construct a Skolem circuit $\mathcal{C}_r$ for each formula $\varphi_r := \big(\varphi \wedge y \equiv r \pmod M\big)$ for $r \in R$. This is because from these $|R|$ circuits, we can easily construct one for φ: Just compute $\mathcal{C}_r(u)$ for each $r \in R$, and check whether $\varphi(u, \mathcal{C}_r(u))$ holds; if it does, then output $\mathcal{C}_r(u)$ (if no $\mathcal{C}_r(u)$ works, then the output can be arbitrary).

However, $\varphi \wedge y \equiv r \pmod M$ is equivalent to a formula $\varphi' \wedge y \equiv r \pmod M$, where φ' contains no modulo constraints on y. Indeed, a modulo constraint on y in φ is either consistent with $y \equiv r \pmod M$ and can be replaced with $\top$, or it is inconsistent with $y \equiv r \pmod M$ and can be replaced with $\bot$. Thus, we may assume that our input formula is of the form $\varphi \wedge y \equiv r \pmod M$, where φ is *y-modulo-free*, meaning φ contains no modulo constraints on y.

Step II: Computing interval ends First, note that for any $\bar{u} \in \mathbb{Z}^n$, the set $V_{\bar{u}}$ of all $v \in \mathbb{Z}$ for which $\varphi(\bar{u}, v)$ holds can be represented as a finite union of intervals. This is because $\varphi(\bar{x}, y)$ has no modulo constraints on y, and thus every atomic formula is an inequality that either yields (i) an upper bound or (ii) a lower bound on y, given a value of $\bar{x}$.

Next, we *construct Presburger circuits that compute the ends of these finitely many intervals*. Once we do this, it is easy to construct a Skolem circuit for $\varphi \wedge y \equiv r \pmod M$: For each interval in some order, check (using div_M) whether it contains a number $\equiv r \pmod M$, and if so, output one.

Let us describe more precisely how a circuit computes the interval union $V_{\bar{u}}$. An *interval-computing circuit* is a Presburger circuit $\mathcal{C}$ that computes a function $\mathbb{Z}^n \to (\mathbb{Z} \times \mathbb{Z})^{k+2}$ for some $k \in \mathbb{N}$. It induces a function $F_{\mathcal{C}} \colon \mathbb{Z}^n \to 2^{\mathbb{Z}}$ as follows. If $\mathcal{C}(\bar{u}) = \langle r_0, s_0, r_1, s_1, \ldots, r_{k+1}, s_{k+1} \rangle$ for some $\bar{u} \in \mathbb{Z}^n$, then we set $F_{\mathcal{C}}(\bar{u}) := I \cup J_1 \cup \cdots \cup J_k \cup K$, where J_i is the closed interval $[r_i, s_i] = \{v \in \mathbb{Z} \mid r_i \leq v \leq s_i\}$; and I is the left-open interval $(-\infty, s_0]$ if $r_0 = 1$ and $I = \emptyset$ if $r_0 \neq 1$; and K is the right-open interval $K = [r_{k+1}, \infty)$ if $s_{k+1} = 1$ and $K = \emptyset$ if $s_{k+1} \neq 1$. Thus, while $r_1, s_1, \ldots, r_k, s_k$ represent end-points of k (possibly overlapping) intervals, r_0 (resp. s_{k+1}) serves as a flag indicating whether the left-open interval $(-\infty, s_0]$ (resp. right-open interval $[r_{k+1}, \infty)$) is to be included in $V_{\bar{u}}$.

For a formula $\varphi(\bar{x}, y)$ with one output y and no modulo-constraints on y, we say that an interval-computing circuit $\mathcal{C}$ is *equivalent to* φ if for every $\bar{u} \in \mathbb{Z}^n$ and every $v \in \mathbb{Z}$, $\varphi(\bar{u}, v)$ holds *if and only if* $v \in F_{\mathcal{C}}(\bar{u})$. The most technical ingredient in our construction is to show:

Claim 5.8. *Given a quantifier-free y-modulo-free Presburger formula, we can compute in polynomial time an equivalent interval-computing circuit.*

Proof sketch. We build the circuit by structural induction, beginning with atomic formulas. Each atomic formula imposes either a lower bound or an upper bound on y, which can be computed using linear functions and div. For example, if the formula is $-x_1 + 3x_2 + 5y \geq 0$, then this is equivalent to $y \geq \frac{1}{5}(x_1 - 3x_2)$, and thus we compute $\mathsf{div}_5(x_1 - 3x_2)$ as the only lower bound.

Building the circuit for a disjunction $\varphi_1 \vee \varphi_2$ is easy: Starting from circuits $\mathcal{C}_1$ and $\mathcal{C}_2$, we simply output all the closed intervals output by each circuit. The open intervals output by the circuits are combined slightly differently depending on the values of the flags. For example, if $\mathcal{C}_1(u)$ and $\mathcal{C}_2(u)$ include intervals $[t_1, \infty)$ and $[t_2, \infty)$, then the new circuit will produce the interval $[\min(t_1, t_2), \infty)$.

The difficult step is to treat conjunctions $\varphi_1 \wedge \varphi_2$. Here, we follow a strategy inspired from sorting networks (Cormen et al. 2009; Ajtai, Komlós, and Szemerédi 1983) to *coalesce-and-sort* the intervals output by each of $\mathcal{C}_1$ and $\mathcal{C}_2$. A basic *coalesce-and-sort* gadget takes as input a pair of (possibly overlapping) intervals $[r, s]$ and $[r', s']$, and coalesces them into one interval if they overlap; otherwise it leaves them unchanged. The gadget outputs two disjoint intervals $[t, u]$ and $[t', u']$, with $[t, u]$ "ordered below" $[t', u']$, such that $[t, u] \cup [t', u'] = [r, s] \cup [r', s']$, and either $[t, u] = \emptyset$ or $u < t'$. Thus, empty intervals are ordered below non-empty ones, and non-empty intervals are ordered by their end-points. A coalesce-and-sort network is a sorting network built using these gadgets. If $\mathcal{C}_i$ outputs q_i (possibly overlapping) intervals, feeding these to a coalesce-and-sort network yields at most q_i disjoint sorted intervals. The interval-computing circuit for $\varphi_1 \wedge \varphi_2$ now computes the $q_1 q_2$ pairwise intersections of these disjoint intervals, coalesce-and-sorts the resulting intervals, and returns the $\max(q_1, q_2)$ intervals at the top of the sorted order. This is sound because intersecting the union of q_1 disjoint intervals with the union of some other q_2 disjoint intervals yields at most $\max(q_1, q_2)$ non-empty disjoint intervals.

To keep the interval-computing circuit size under check, our construction maintains carefully chosen size invariants. Specifically, we ensure that the number of interval endpoints at the output of, and indeed the total size of the interval-computing circuit for $\varphi_1 \vee \varphi_2$ or $\varphi_1 \wedge \varphi_2$ is always bounded by a polynomial in $|\varphi_1| + |\varphi_2|$. Intuitively, since each interval endpoint at the output of the interval-computing circuit must originate from an atomic formula at a leaf in the tree representation of the specification, there are atmost a polynomial number of interval endpoints to track.

The reader is referred to the full version (Akshay et al. 2025) for details of the proof, and a pictorial depiction. □

The case of multiple output variables It remains to prove Theorem 5.3 in the general case (i.e. $m \geq 1$).

Proof of Theorem 5.3. Let $\widehat{\varphi}^{(i)}$ denote $\exists^{\mathsf{local}} y_{i+1}, \ldots, y_m \colon \varphi(\bar{x}, \bar{y})$, for $i \in [1, m-1]$, and let $\widehat{\varphi}^{(}m) = \varphi$. For each i in m down to 1, we obtain a Presburger circuit for a Skolem function f_i for the existentially quantified variable (y_i) in $\forall \bar{x} \forall y_1, \ldots, y_{i-1} \exists y_i \colon \widehat{\varphi}^{(i)}(\bar{x}, y_1, \ldots, y_i)$ using Theorem 5.7 for single output specifications. Note that due to the equivalence of local and (ordinary) existential quantification, f_i is then a Skolem function for $\forall \bar{x} \forall y_1, \ldots, y_{i-1} \exists y_i \colon \tilde{\varphi}^{(i)}$, where $\tilde{\varphi}^{(i)}$ is a quantifier-free equivalent of $\exists y_{i+1}, \ldots, y_m \colon \varphi(\bar{x}, \bar{y})$. Each such f_i expresses y_i in terms of $\bar{x}$ and $y_1, \ldots, y_{i-1}$. It is easy to see that by composing the resulting Presburger circuits, we can obtain Presburger circuits for Skolem functions for all existentially quantified variables in $\forall \bar{x} \exists \bar{y} \colon \varphi(\bar{x}, \bar{y})$. Each resulting Skolem function is expressed only in terms of $\bar{x}$. □

Achieving PSyNF We now prove Theorem 5.4. using (either of) the recent QE procedures.

Proof of Theorem 5.4. For each $i \in [1, m-1]$, let ψ_i be a quantifier-free equivalent of $\exists y_{i+1}, \ldots, y_m \colon \varphi(\bar{x}, \bar{y})$. By recent results on quantifier-elimination (Chistikov, Mansutti, and Starchak 2024; Haase et al. 2024), we can obtain such a ψ_i of at most exponential size in $|\varphi|$. The formula $\eta = \varphi \wedge \bigwedge_{i \in [1, m-1]} \psi_i$ is equivalent to φ, and satisfies the equivalence condition regarding local and global quantification. It remains to achieve modulo-tameness. For this, we notice that both recent QE procedures, (Chistikov, Mansutti, and Starchak 2024, Thm. 3) and (Haase et al. 2024, Thm. 3.1) produce an exponential disjunction of polynomial-sized formulas. We may thus assume that $\psi_i = \bigvee_{j=1}^{s} \psi_{i,j}$ for some exponential s for each $i \in [1, m-1]$. We can now write η equivalently as $\bigvee_{f \in F} \left(\varphi \wedge \bigwedge_{i \in [1, m-1]} \psi_{i, f(i)} \right)$, where F is the set of functions $f \colon [1, m-1] \to [1, s]$. Observe that each formula $\tau_f := \varphi \wedge \bigwedge_{i \in [1, m-1]} \psi_{i, f(i)}$ is of polynomial size, and thus the product M of all (polynomially many) moduli (which are at most exponential) ocurring in τ_f is at most exponential. We thus rewrite all modulo constraints in τ_f for variables y_k w.r.t. M, yielding an exponential-sized equivalent of τ_f which is y_k-modulo-tame for all k. The resulting formula has at most exponential size and is in PSyNF. □

Checking PSyNF: Step I Finally, we prove Theorem 5.5. We begin with an auxiliary result:

Theorem 5.9. *Given a y-modulo-tame quantifier-free formula $\varphi(\bar{x}, y)$, it is coNP-complete to decide whether $\forall \bar{x} \exists y \colon \varphi(\bar{x}, y)$ holds.*

Note that Theorem 5.9 implies Corollary 5.6, since a formula over the signature $\langle \mathbb{Z}; +, <, 0, 1 \rangle$ is automatically y-modulo-tame for each variable y.

Proof of Theorem 5.9. Since φ is y-modulo-tame, Theorem 5.7 allows us to compute a polynomial-sized Presburger

circuit $\mathcal{C}$ that computes a Skolem function f for the existentially quantified variable in $\forall\bar{x}\exists y\colon \varphi$. Now, we can build a polynomial-sized circuit $\mathcal{C}'$ for the function g with $g(\bar{x}) = 1$ if $\varphi(\bar{x}, f(\bar{x}))$, and $g(\bar{x}) = 0$ otherwise (see the full version (Akshay et al. 2025) for details). Then, we have $\forall\bar{x}\exists y\colon \varphi(\bar{x}, y)$ if and only if the circuit $\mathcal{C}'$ returns 1 true for every vector $\bar{x}$. Equivalently, $\forall\bar{x}\exists y\colon \varphi(\bar{x}, y)$ does not hold if and only if there is $\bar{x}$ such that $\mathcal{C}'$ evaluates to 0. The existence of such an $\bar{x}$ can be decided in NP by a reduction to existential Presburger arithmetic: Given $\mathcal{C}'$, we introduce a variable for the output of each gate, and require that (i) each gate is evaluated correctly and (ii) the circuit outputs 0. □

Checking PSyNF: Step II We can now show Thm. 5.5:

Proof of Theorem 5.5. We can clearly check whether φ is in NNF and whether φ is y_i-modulo-tame for every $i \in [1, m]$. It remains to check whether $\varphi^{(i)}$ in eq. (1) holds for every $i \in [1, m-1]$. This is the case iff each formula

$$\varphi^{\dagger(i)} := \forall\bar{x}\forall y_1, \ldots, y_i\colon (\exists^{\mathsf{local}} y_{i+1}, \ldots, y_m\colon \varphi(\bar{x}, \bar{y}) \\ \to \exists y_{i+1}\colon \exists^{\mathsf{local}} y_{i+2}, \ldots, y_m\colon \varphi(\bar{x}, \bar{y}))$$

holds for i in $m-1$ down to 1. Indeed, since we know from $\varphi^{\dagger(m-1)} = \varphi^{(m-1)}$ that y_m can be eliminated locally, we can plug that equivalence into $\varphi^{(m-2)}$ to obtain $\varphi^{\dagger(m-2)}$. By repeating this argument, we can see that the conjunction of all $\varphi^{\dagger(i)}$ implies the conjunction of all $\varphi^{(i)}$.

Note that $\varphi^{\dagger(i)}$ belongs to the $\forall^*\exists$ fragment, and the formula is modulo-tame w.r.t. the existentially quantified variable. By Theorem 5.9, we can decide the truth of $\varphi^{\dagger(i)}$ in coNP. For coNP-hardness, note that an NNF formula φ with free variables in $\bar{x}$ is in PSyNF if and only if $\forall\bar{x}\colon \varphi(\bar{x})$. Moreover, universality for NNF formulas is coNP-hard. □

Our final result in this section is that the PSyNF normal form is "optimal" for existential quantification and synthesis for single-output modulo-tame specifications. Specifically,

Theorem 5.10. *Let $\mathfrak{S}$ be a class of quantifier-free* PrA*-formulas in NNF on free variables $\bar{x}$ and y such that:*

1. *$\mathfrak{S}$ is universal, i.e. for every quantifier-free* PrA *formula $\psi(\bar{x}, y)$, there is a semantically equivalent formula in $\mathfrak{S}$*
2. *Every formula $\varphi(\bar{x}, y)$ in $\mathfrak{S}$ is y-modulo tame*
3. *There is a polynomial time algorithm that given $\varphi(\bar{x}, y) \in \mathfrak{S}$, computes a quantifier-free equivalent of $\exists y : \varphi(\bar{x}, y)$.*

Then there exists a polynomial time algorithm that compiles $\varphi(\bar{x}, y) \in \mathfrak{S}$ to φ' that is in PSyNF *wrt y.*

The proof is in the full version (Akshay et al. 2025). Assumption 3) is weaker than requiring PFnS to be efficiently solvable. This is due to the difference in vocabulary between Presburger formulas and circuits (unlike the Boolean case).

6 Syntactic Normal Form for PFnS

We now present a *syntactic normal form* for PFnS. This means, it has three properties: (i) It is *syntactic*, meaning one can check in polynomial time whether a given formula is in this normal form, (ii) it facilitates PFnS, meaning for formulas in normal form, PFnS is in polynomial time, and (iii) every formula can be brought into this normal-form (in exponential time). We call this normal form PSySyNF.

Definition of PSySyNF Recall that an *affine transformation* (from $\mathbb{Q}^k$ to $\mathbb{Q}^\ell$) is a map $\mathbb{Q}^k \to \mathbb{Q}^\ell$ of the form $\bar{x} \mapsto B\bar{x} + \bar{b}$, where $B \in \mathbb{Q}^{k\times\ell}$ is a $k \times \ell$ matrix over $\mathbb{Q}$ and $\bar{b} \in \mathbb{Q}^\ell$ is a vector in $\mathbb{Q}^\ell$. In particular, the affine transformation is described by the entries of B and $\bar{b}$. Consider a quantifier-free PrA formula $\varphi(\bar{x}, \bar{y})$, $\bar{x} = (x_1, \ldots, x_n)$, $\bar{y} = (y_1, \ldots, y_m)$. To simplify notation, we define for any vector $(\bar{u}, \bar{v}) \in \mathbb{Z}^{n+m}$ with $\bar{u} \in \mathbb{Z}^n$, $\bar{v} \in \mathbb{Z}^m$:

$$\bar{u}^i := (u_1, \ldots, u_n, v_1, \ldots, v_i), \quad \bar{v}^i := (v_{i+1}, \ldots, v_m).$$

The idea of PSySyNF is to encode Skolem functions in the formula: Each maximal conjunctive subformula (see Section 5) is annotated with affine transformations $A_1, \ldots, A_m$, where A_i could serve as Skolem functions for this subformula, when $\bar{y}^i$ are considered as output variables. This can be viewed as an analogue of weak DNNF (or wDNNF) in the Boolean setting (Akshay et al. 2021), where each maximal conjunctive subformula provides for each output variable a truth value for a Skolem function. Instead of concrete truth values, PSySyNF has affine transformations in $\bar{x}^i$.

We say that φ is in PSySyNF ("syntactic synthesis normal form") if for every maximal conjunctive subformula φ', there exists an $M \in \mathbb{Z}$ and for every $i \in [1, m]$, there exists an affine transformation $A_i\colon \mathbb{Q}^{n+i} \to \mathbb{Q}^{m-i}$ such that (a) φ is y_i-modulo-tame for every $i \in [1, m]$ and (b) every denominator in the coefficients in A_i divides M and (c) φ' is a positive Boolean combination of formulas of the form

$$\left(\psi(\bar{x}, \bar{y}) \vee \bigvee_{i=0}^{m} \bar{y}^i = A_i(\bar{x}^i)\right) \wedge \\ \bigwedge_{i=0}^{m} \psi(\bar{x}^i, A_i(\bar{x}^i)) \wedge (\bar{x}, \bar{y}) \equiv (\bar{r}, \bar{s}) \pmod{M}, \quad (2)$$

where $\psi(\bar{x}, \bar{y})$ is an atomic formula (with vector congruences allowed), $(\bar{r}, \bar{s}) \in [0, M-1]^{m+n}$, and where $A_i(\bar{r}^i) \in \mathbb{Z}^{m-i}$. Note that assuming (b) and $(\bar{x}, \bar{y}) \equiv (\bar{r}, \bar{s}) \pmod{M}$, the condition $A_i(\bar{r}^i) \in \mathbb{Z}^{m-i}$ is equivalent to $A_i(\bar{x}^i) \in \mathbb{Z}^{m-i}$ (see the full version (Akshay et al. 2025)).

Properties of PSySyNF Let us now show that PSySyNF indeed has the properties (i)–(iii) above. First, one can easily check (in polynomial time) whether a formula is in PSySyNF: In each parenthesis, the disjunction over $\bar{y}^i = A_i(\bar{x}^i)$ means the formula explicitly contains all coefficients of the affine transformation A_i, for every $i \in [1, m]$. Once these are looked up, one can verify that the subformulas $\psi(\bar{x}^i, A_i(\bar{x}^i))$ are obtained by plugging $A_i(\bar{x}^i)$ into $\psi(\bar{x}^i, \bar{y}^i)$ in place of $\bar{y}^i$. Property (ii) follows from:

Theorem 6.1. *Every formula in* PSySyNF *is also in* PSyNF.

Essentially, this is because the annotated affine transformations yield valuations for satisfying globally quantified subformulas. Thus, Theorem 5.3 yields a polynomial-time algorithm for PFnS for PSySyNF formulas.

Finally, we have property (iii): PSySyNF can be achieved with at most an exponential blow-up:

Theorem 6.2. *Every quantifier-free* PrA *formula has an equivalent in* PSySyNF *of at most exponential size.*

Note that Theorems 6.1 and 6.2 yield an alternative proof for Theorem 5.4. While Theorem 5.4 could be shown using either of the two recent quantifier elimination techniques (Chistikov, Mansutti, and Starchak 2024; Haase et al. 2024), Theorem 6.2 depends on the specific geometric insight from (Haase et al. 2024), namely Proposition 4.2.

Roughly speaking, the idea for proving Theorem 6.2 is to bring the formula into DNF where each co-clause only contains linear inequalities. For each co-clause we then apply Proposition 4.2 to yield exponentially many affine transformations A_i that yield candidate assignments for $\bar{y}$. From these A_i, we then construct the subformulas of the form (2).

Succinctness We have seen that compared to PSyNF, the form PSySyNF has the advantage that it is syntactic (i.e. easy to check). However, as we show now, PSyNF has the advantage that it can be *exponentially more succinct*. More specifically, there are formulas in PSyNF whose smallest equivalent in PSySyNF are exponentially larger:

Theorem 6.3. *There is a family* $(\Psi_n)_{n\geq 0}$ *of* PSyNF *formulas such that any equivalent* PSySyNF *has size* $2^{\Omega(|\Phi_n|)}$.

One can take $\Psi_n(x, y) := x < y \leq x + 2^n \wedge y \equiv 0 \pmod{2^n}$. For each x, there is exactly one $y \in [x, x + 2^n]$ with $\Psi_n(x, y)$, and there are exponentially many (2^n) possible differences between x and y. One can argue that for each such difference, a separate affine map has to appear in any PSySyNF (see the full version (Akshay et al. 2025)).

7 Discussion and Conclusion

Our work maps out the landscape of functional synthesis for Presburger specifications, setting up a new research agenda of normal forms for such specifications, and compilation to them. In doing so, our work exposes similarities and differences between functional synthesis from Boolean and Presburger specifications. Specifically the complexity upper bounds for PFnS match the best known algorithms for BFnS (EXPTIME), though for one-output specifications, BFnS is known to be poly-time solvable, while PFnS is at least NP-hard. This makes it necessary to design new normal forms for PrA specifications using new concepts of modulo-tameness and local quantification. Interestingly, local quantification may be viewed as a generalization of a core idea underlying SynNNF in BFnS (Akshay et al. 2019).

It is natural to ask at this point whether our complexity results for PFnS would change if we used a syntax for Skolem functions different from the one chosen in 3. First, our *upper bound* results (i.e. algorithms that construct Presburger circuits; from general PrA formulas or ones in normal forms) would still apply, as long as the syntax (i) has polynomial-size descriptions of the atomic functions described in Section 3, and (ii) is compositional, in the sense that descriptions can be composed as in circuits. Indeed, under these assumptions, any Presburger circuit can be translated into our syntax. In our view, these are reasonable assumptions, since the atomic functions are natural examples of Presburger-definable functions and arise as (unique) Skolem functions of simple formulas. Second, our exponential *unconditional lower bound* (Theorem 4.4) might not hold anymore (e.g. Skolem functions for $(\mu_n)_{n\geq 0}$ might just be part of the syntax). However, assuming *efficient evaluation*, i.e. functions in the chosen syntax can be evaluated in time polynomial in the size of the functions, we cannot avoid an exponential *conditional lower bound*. Indeed, assuming non-uniform ETH, an exponential size lower bound is known for Boolean functional synthesis for any representation of Boolean functions that admits evaluation in polynomial time (Akshay et al. 2021, p. 59). This implies the same for PFnS.

There are several interesting questions that arise from a comparison of our work with that on knowledge compilation for Boolean functional synthesis. For instance, we saw that the space of all Skolem functions for Presburger specifications can be characterized using a set of intervals that can be represented using Presburger circuits. The corresponding characterization for Boolean Skolem functions using Skolem basis (see (Akshay, Chakraborty, and Jain 2023)) has the flavour of an on-set and a don't-care set. The relation between these two representations is unclear, and warrants further invesigation, especially because BFnS can be encoded as PFnS. Similarly, (Shah et al. 2021) give a precise characterization for polynomial-time and size solvable BFnS instances. We do not have such a characterization for PFnS, though Theorem 5.10 provides such a result for single-output specifications. We leave the precise characterization of polynomial-time and size solvable multi-output PFnS instances as an open problem.

Conjunctive Normal Form (CNF) is well-accepted as a standard form for Boolean formulas, and state-of-the-art Boolean reasoning engines (SAT-solvers, functional synthesis tools) often exploit the CNF representation for efficient processing. However, for Presburger Arithmetic, there is no such dominant representation form that we are aware of. For example, QF_LIA (quantifier-free linear integer arithmetic) benchmarks used by the SMT community are Presburger formulas sans modulo constraints, that are not always represented in CNF in benchmark suites. Hence, we did not assume the Presburger specification to be in CNF (or in a similar alternative form) in this work. The question of whether starting with Presburger specifications in CNF (or similar forms) can lead to practically efficient compilation techniques to PSyNF remains open. Interestingly, not all knowledge compilation based approaches for BFnS require CNF representation to start with (see e.g. (Akshay, Chakraborty, and Jain 2023; Akshay, Chakraborty, and Shah 2024))

As part of future work, we would like to improve our constructions to make them more efficient in practice. For instance, requiring modulo-tameness can lead to blowups that can potentially be avoided by finding alternate characterizations and normal forms. We expect our results to be a stepping stone towards practical implementability of Skolem function synthesis algorithms for Presburger arithmetic via knowledge compilation, and their wider use within the KR and SMT communities.

Acknowledgments

We are grateful to Christoph Haase for answering questions about the literature around Corollary 5.6 and to Dmitry Chistikov for pointing out the work of Schöning (1997). The authors are grateful to the Schloss Dagstuhl – Leibniz Center for Informatics; this collaboration started at the Dagstuhl Seminar 24171: "Automated Synthesis: Functional, Reactive and Beyond".

Funded by the European Union (ERC, FINABIS, 101077902), the Deutsche Forschungsgemeinschaft (389792660 TRR 248—CPEC), a SERB MATRICS (MTR/2023/01167) grant of Government of India, and a research award grant (RI/0224-10001626-001) of Indian Institute of Technology (IIT) Bombay. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union, the European Research Council Executive Agency, the Deutsche Forschungsgemeinschaft, Government of India or IIT Bombay. Neither the European Union, the Deutsche Forschungsgemeinschaft, Government of India, IIT Bombay nor the granting authorities can be held responsible for them.

References

Ajtai, M.; Komlós, J.; and Szemerédi, E. 1983. An 0(n log n) sorting network. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, 1–9. New York, NY, USA: Association for Computing Machinery.

Akshay, S.; Arora, J.; Chakraborty, S.; Krishna, S. N.; Raghunathan, D.; and Shah, S. 2019. Knowledge compilation for Boolean functional synthesis. In Barrett, C. W., and Yang, J., eds., *2019 Formal Methods in Computer Aided Design, FMCAD 2019, San Jose, CA, USA, October 22-25, 2019*, 161–169. IEEE.

Akshay, S.; Chakraborty, S.; Goel, S.; Kulal, S.; and Shah, S. 2021. Boolean functional synthesis: hardness and practical algorithms. *Formal Methods Syst. Des.* 57(1):53–86.

Akshay, S.; Balasubramanian, A. R.; Chakraborty, S.; and Zetzsche, G. 2025. Presburger functional synthesis: Complexity and tractable normal forms. *arXiv preprint arXiv:2508.07207*.

Akshay, S.; Chakraborty, S.; and Jain, S. 2023. Counterexample guided knowledge compilation for Boolean functional synthesis. In Enea, C., and Lal, A., eds., *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part I*, volume 13964 of *Lecture Notes in Computer Science*, 367–389. Springer.

Akshay, S.; Chakraborty, S.; and Shah, S. 2024. Tractable representations for Boolean functional synthesis. *Ann. Math. Artif. Intell.* 92(5):1051–1096.

Arora, S., and Barak, B. 2009. *Computational Complexity - A Modern Approach*. Cambridge University Press.

Chakraborty, S., and Akshay, S. 2022. On synthesizing computable Skolem functions for first order logic. In Szeider, S.; Ganian, R.; and Silva, A., eds., *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022, August 22-26, 2022, Vienna, Austria*, volume 241 of *LIPIcs*, 30:1–30:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Chakraborty, S.; Fried, D.; Tabajara, L. M.; and Vardi, M. Y. 2018. Functional synthesis via input-output separation. In *Proc. of FMCAD*.

Cherniavsky, J. C. 1976. Simple programs realize exactly Presburger formulas. *SIAM J. Comput.* 5(4):666–677.

Chistikov, D.; Mansutti, A.; and Starchak, M. R. 2024. Integer linear-exponential programming in NP by quantifier elimination. In Bringmann, K.; Grohe, M.; Puppis, G.; and Svensson, O., eds., *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, 132:1–132:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Chistikov, D. 2024. An introduction to the theory of linear integer arithmetic (invited paper). In Barman, S., and Lasota, S., eds., *44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2024, December 16-18, 2024, Gandhinagar, Gujarat, India*, volume 323 of *LIPIcs*, 1:1–1:36. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2009. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition.

Enderton, H. B. 1972. *A Mathematical Introduction to Logic*. New York,: Academic Press.

Fedyukovich, G., and Gupta, A. 2019. Functional synthesis with examples. In Schiex, T., and de Givry, S., eds., *Principles and Practice of Constraint Programming - 25th International Conference, CP 2019, Stamford, CT, USA, September 30 - October 4, 2019, Proceedings*, volume 11802 of *Lecture Notes in Computer Science*, 547–564. Springer.

Fedyukovich, G.; Gurfinkel, A.; and Gupta, A. 2019. Lazy but effective functional synthesis. In Enea, C., and Piskac, R., eds., *Verification, Model Checking, and Abstract Interpretation*, 92–113. Cham: Springer International Publishing.

Fried, D.; Tabajara, L. M.; and Vardi, M. Y. 2016. BDD-based Boolean functional synthesis. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, 402–421.

Golia, P.; Slivovsky, F.; Roy, S.; and Meel, K. S. 2021. Engineering an efficient Boolean functional synthesis engine. In *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2021, Munich, Germany, November 1-4, 2021*, 1–9. IEEE.

Golia, P.; Roy, S.; and Meel, K. S. 2020. Manthan: A data-driven approach for Boolean function synthesis. *Computer Aided Verification* 12225:611 – 633.

Grädel, E. 1989. Dominoes and the complexity of subclasses of logical theories. *Ann. Pure Appl. Log.* 43(1):1–30.

Gurari, E. M., and Ibarra, O. H. 1981. The complexity

of the equivalence problem for simple programs. *J. ACM* 28(3):535–560.

Haase, C.; Krishna, S. N.; Madnani, K.; Mishra, O. S.; and Zetzsche, G. 2024. An efficient quantifier elimination procedure for Presburger arithmetic. In Bringmann, K.; Grohe, M.; Puppis, G.; and Svensson, O., eds., *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, 142:1–142:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Haase, C. 2014. Subclasses of Presburger arithmetic and the weak EXP hierarchy. In *Proc. CSL-LICS 2014*, 47:1–47:10. ACM.

Haase, C. 2018. A survival guide to Presburger arithmetic. *ACM SIGLOG News* 5(3):67–82.

Hadamard, J. 1893. Rèsolution d'une question relative aux dèterminants. *B. Sci. Math.* 2(17):240–246.

Huth, M., and Ryan, M. 2004. *Logic in Computer Science: Modelling and Reasoning about Systems*. USA: Cambridge University Press.

Ibarra, O. H., and Leininger, B. S. 1981. Characterizations of Presburger functions. *SIAM J. Comput.* 10(1):22–39.

Jiang, J. R. 2009. Quantifier elimination via functional composition. In Bouajjani, A., and Maler, O., eds., *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, volume 5643 of *Lecture Notes in Computer Science*, 383–397. Springer.

John, A. K.; Shah, S.; Chakraborty, S.; Trivedi, A.; and Akshay, S. 2015. Skolem functions for factored formulas. In *2015 Formal Methods in Computer-Aided Design (FMCAD)*, 73–80. IEEE.

Kuncak, V.; Mayer, M.; Piskac, R.; and Suter, P. 2010. Complete functional synthesis. *SIGPLAN Not.* 45(6):316–329.

Kuncak, V.; Mayer, M.; Piskac, R.; and Suter, P. 2013. Functional synthesis for linear arithmetic and sets. *Int. J. Softw. Tools Technol. Transf.* 15(5-6):455–474.

Lin, Y.; Tabajara, L. M.; and Vardi, M. Y. 2022. ZDD Boolean synthesis. In Fisman, D., and Rosu, G., eds., *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, 64–83. Springer.

Lin, Y.; Tabajara, L. M.; and Vardi, M. Y. 2024. Dynamic programming for symbolic Boolean realizability and synthesis. In Gurfinkel, A., and Ganesh, V., eds., *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part III*, volume 14683 of *Lecture Notes in Computer Science*, 112–134. Springer.

Presburger, M. 1929. Über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt. *Comptes Rendus du I congres de Mathematiciens des Pays Slaves* 92–101.

Rabe, M. N., and Seshia, S. A. 2016. Incremental determinization. In *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, 375–392.

Schöning, U. 1997. Complexity of Presburger arithmetic with fixed quantifier dimension. *Theory Comput. Syst.* 30(4):423–428.

Shah, P.; Bansal, A.; Akshay, S.; and Chakraborty, S. 2021. A normal form characterization for efficient Boolean Skolem function synthesis. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, 1–13. IEEE.