On the complexity of fragments of second-order logic

Joel Rybicki

December 18, 2011
Abstract

Existential second order logic (ESO) is an extension of first-order logic (FO) that allows the existential quantification of relation and function symbols. The seminal result due to Fagin states that the properties expressible in ESO coincide with the decision problems in the complexity class NP. On the other hand, the monadic fragment of ESO is considerably weaker as it corresponds to the class of regular languages.

In this work, we examine the computational complexity of fragments of ESO which restrict the arity of the second-order quantifications and the number of universal first-order quantifications in the formulas. In particular, we focus on the fragment allowing only existential quantification of unary relation and function symbols together with a single first-order universal quantification.

We prove a result due to Grandjean stating that the properties definable in this logic correspond to structures recognizable with non-deterministic random-access machines whose running time is linearly bounded as a function of the cardinality of the input structure. In some structures, such as graphs, this means that properties expressible in this logic are computable in sub-linear time in the size of the input.

As an application, we classify various graph properties expressible in this logic, such as connectivity, the existence of Hamiltonian circuits and small dominating sets. To complement this, some non-expressibility results are also given. For example, expressing whether the graph is a tree or has small vertex covers is not possible in this logic implying lower bounds for the time complexity of these problems.
Contents

1 Introduction 2

2 Preliminaries 4
   2.1 Relations and functions .................................. 4
   2.2 Vocabularies and structures ............................... 4
   2.3 Graphs .................................................. 5
   2.4 First-order and existential second-order logic ............ 6

3 Non-deterministic random access machines 9
   3.1 Input and registers ....................................... 10
   3.2 Instructions and computation .............................. 11
   3.3 Problems and complexity classes ......................... 12

4 The complexity of ESO 12
   4.1 Linear order with a single variable ...................... 13
   4.2 A characterization of linear time computation ............ 21

5 Classifying graph problems 32
   5.1 Problems recognizable in vertex-linear time .......... 32
   5.2 Problems outside the class VNLIN ....................... 36

6 Conclusions and open problems 40
1 Introduction

One of the fundamental open problems in computer science and mathematical logic is the P versus NP question [1]: is the set of decision problems recognizable by polynomial time Turing machines equal to the set of decision problems recognizable by non-deterministic polynomial time Turing machines? Informally, if a solution to a problem can be verified efficiently, can the solution itself be computed efficiently? It is well-known that this problem has turned out to be notoriously difficult to solve. Lance Fortnow has written an informal overview of the history of the problem and how different techniques have so far failed to produce a proof either way [6].

The P vs NP question, among other fundamental open questions in computational complexity, such as whether the class NP is closed under complement, have motivated researchers to develop numerous new techniques for solving these problems. One approach is known as descriptive complexity theory which combines mathematical logic and model theory with computational complexity [14]. Instead of studying how much time or space computing whether a property \( A \) holds for the input, it is possible to ask how powerful logics are needed to express whether the input structure has the property \( A \).

It has been shown that some logics coincide with fundamental complexity classes, such as P and NP, under certain assumptions. The seminal result due to Fagin states that existential-second order logic captures the class NP over finite models [5]. That is, properties definable in existential second-order logic are computable in polynomial time with non-deterministic Turing machines and vice versa. Similarly, there exist logics that capture properties computable in polynomial time over ordered finite models [13] and for various other complexity classes as well [14, 15].

Therefore, the P vs NP question can be posed as “can second-order existential logic express properties of finite models that the logics capturing the class P cannot?” This work will not consider the P vs NP question as such, but rather concentrates on some of the more restricted classes of computational complexity at the boundary of these two well-known classes.

While complexity theorists often identify the efficiency of computation with polynomial time computability, in practice this is not always the case. For example, a polynomial time algorithm with running time \( O(n^{4108}) \) may be considerably slower than an exponential time algorithm for most realistic input sizes.

It is however possible to achieve finer control on the time complexity by using different logics which, for example, impose syntactic restrictions on the formulas. This line of research was premiered by Lynch [16] together with Grandjean and Olive [11] who refined the classic result due to Fagin [5]. First, Lynch [16] showed that the computation of a non-deterministic Turing machine with running time \( O(n^d) \) can be simulated with a sentence whose
relational symbols are at most $d$-ary if a ternary addition predicate is given. Later, Grandjean and Olive [11] showed that for the case $d = 1$ such sentences can simulate so-called non-deterministic random-access machines whose running time is bounded by $O(n)$.

The random-access machine model resembles much more closely to real-world computers than the tape-based Turing machine model. Thus, variants of the RAM models are often used in algorithmics research. While the random-access machines are somewhat more powerful, they are not too powerful compared to Turing machines as we will argue later in Section 3.

If quantification of function symbols is allowed, Grandjean and Olive [12] have shown that for each polynomial of degree $d$ and input structures of cardinality $n$, the properties computable with non-deterministic random-access machines in time $O(n^d)$ corresponds to a fragment of existential second-order logic which imposes the following syntactic restrictions on the formulas: the arity of the second-order quantified relation and function symbols can be at most $d$ and the formula may contain at most $d$ universal first-order quantifications. We denote this logic as $ESO[\#d, \forall d]$. It is known that many other fragments of existential second-order logic capture fundamental complexity classes [8], and for example, by Büchi’s theorem, we know that the logic restricting the second-order quantifications to unary relation symbols captures the class of regular languages over strings [15].

In this work, we take a somewhat of an extreme approach to efficiency: what can be computed using linear-time random-access machines and how much non-determinism will help if the running time is restricted in such a drastic manner? In particular, we examine which properties are expressible in the logic $ESO[\#1, \forall 1]$. It turns out that the class of problems solvable in linear-time (in the cardinality of the input structure) with non-deterministic random-access machines is rather interesting; the class contains many classic NP-complete problems, whereas many easy polynomial-time solvable problems reside outside it. In addition, for many languages such as the language of relational graphs, this implies that the problems can be solved in sub-linear time in the size of the input. Furthermore, it has been shown that this class is not closed under complement — a question which has not yet been settled for the class NP.

This work is structured as follows. Section 2 contains formal definitions of models, graphs, first- and second-order logics. Section 3 defines the model of computation, the non-deterministic random-access machine, and its semantics. Section 4 contains the logical characterization of linear-time computation on NRAMs. Section 5 applies the results of Section 4 to obtain expressibility and non-expressibility results for the logic $ESO[\#1, \forall 1]$. These imply lower and upper bounds for the time complexity of the corresponding decision problems. Finally, Section 6 reviews some open problems.
2 Preliminaries

In this section we fix the notation used throughout this work and recall some fundamental notions of both finite model theory and graph theory.

2.1 Relations and functions

All numbers considered in this work are natural numbers unless otherwise specified. The set of all natural numbers \{0, 1, \ldots, \} is denoted by \( \mathbb{N} \). The set of positive natural numbers is \( \mathbb{N}_+ = \mathbb{N} \setminus \{0\} \).

Let \( A \) be a set. If \( A \) is finite, we denote the size of \( A \) by \( |A| \). For every \( n, k \in \mathbb{N}_+ \), we write \([n]\) for the set \{0, \ldots, n - 1\} and the \( k \)-ary Cartesian power of set \( A \) is the set \( A^k = \{(a_0, \ldots, a_{k-1}) : a_i \in A, i \in [k]\} \).

An \( n \)-ary relation \( R \) over the set \( A \) is a subset \( R \subseteq A^n \). We say that a total function \( f: A^n \to A \) is an \( n \)-ary function of \( A \). As a special case, 0-ary (nullary) functions are considered as constants. The restriction of function \( f \) to domain \( B \subseteq A \) is denoted as \( f|B \).

We now define some properties of binary relations. Let \( R \) be a binary relation \( R \subseteq A^2 \). The relation \( R \) is said to be reflexive if for all \( a \in A \) it holds that \((a, a) \in R\) and anti-reflexive if \((a, a) \notin R\) for all \( a \in A \). A symmetric relation has the property that \((a, b) \in R\) implies \((b, a) \in R\), whereas the relation \( R \) is anti-symmetric if \{\((a, b), (b, a)\}\} \subseteq R\) implies that \( a = b \). Furthermore, we say that \( R \) is transitive if for all \( a, b, c \in A \) it holds that \{\((a, b), (b, c)\}\} \subseteq R\) implies \((a, c) \in R\). Finally, a relation \( R \) is total if \((a, b) \in R\) or \((b, a) \in R\) for all \( a, b \in A \).

We say that a binary relation is a linear order if the relation is anti-reflexive, anti-symmetric, transitive and total. Linear orders are denoted by the symbol \(<\).

Finally, let \( f: A^k \to A \) be a function where \( k > 0 \). For each \( i \in \mathbb{N} \), we define the iterated function \( f^i \) as

\[
\begin{align*}
   f^0(x) &= x \\
   f^{i+1}(x) &= f(f^i(x)).
\end{align*}
\]

2.2 Vocabularies and structures

A vocabulary \( \tau \) is a set of relation and function symbols. The relation symbols of the vocabulary \( \tau \) consist of the set \( \text{Rel}(\tau) = \{R_0, \ldots, R_t\} \), whereas the set of function symbols of \( \tau \) is \( \text{Fun}(\tau) = \{f_0, \ldots, f_s\} \). Each symbol \( s \in \tau \) has an associated arity \( \#s \in \mathbb{N} \). If \( s \in \text{Rel}(\tau) \), then \( \#s \geq 1 \). If \( s \in \text{Fun}(\tau) \), then \( \#s \geq 0 \). We do not distinguish constant symbols from function symbols. Thus, function symbols with arity 0 are considered as constant symbols. The
set of constant symbols is \( \text{Con}(\tau) = \{ s \in \text{Fun}(\tau) : \#s = 0 \} \). The arity of the vocabulary is \( \#\tau = \max\{\#s : s \in \tau \} \).

**Definition 1.** Let \( \tau = \{ R_0, \ldots, R_t, f_0, \ldots, f_s \} \) be a vocabulary and \( M \neq \emptyset \) a set. A \( \tau \)-structure \( \mathcal{M} \) is a tuple

\[
(M, R^M_0, \ldots, R^M_t, f^M_0, \ldots, f^M_s),
\]

where \( \text{Dom}(\mathcal{M}) = M \) is the domain of the structure. Each relation and function symbol of the vocabulary has an interpretation in the structure. For every relation symbol \( R_i \in \tau \), there is a relation \( R^M_i \) of arity \( \#R_i \) over \( \text{Dom}(\mathcal{M}) \). Similarly, for function symbol \( f_i \in \tau \), we have a function \( f^M_i \) of arity \( \#f_i \).

A \( \tau \)-structure \( \mathcal{M} \) is said to be finite if both the vocabulary \( \tau \) and the domain \( \text{Dom}(\mathcal{M}) \) is finite. The set of all finite \( \tau \)-structures is denoted by \( \text{Str}(\tau) \). In this work, we only consider finite structures and vocabularies.

The cardinality of a structure \( \mathcal{M} \) is the size of its domain, that is, \( \text{Card}(\mathcal{M}) = |\text{Dom}(\mathcal{M})| \).

To lighten the notation, we often write the interpretation \( s^M \) simply as \( s \) if both the structure \( \mathcal{M} \) and the vocabulary we refer to are clear from the context. In addition, \( \mathcal{M} = \langle M, \tau \rangle \) denotes a \( \tau \)-structure with \( \text{Dom}(\mathcal{M}) = M \) and interpretations for each symbol in \( \tau \).

For any vocabulary \( \sigma \subseteq \tau \) and \( \tau \)-structure \( \mathcal{M}(M, \tau) \), we write \( \mathcal{M}|\sigma \) to denote the \( \sigma \)-structure \( \langle M, \sigma \rangle \).

### 2.3 Graphs

In this section, we give definitions for the graph theoretic concepts used throughout this work. We will mostly follow the presentation of Diestel [2].

For any set \( A \) and \( k \in \mathbb{N} \), let \( [A]^k \) denote the set \( \{ B \subseteq A, |B| = k \} \). A graph is a pair \( \mathcal{G} = (V, E) \) where \( V \) is the set of vertices and the set of edges \( E \subseteq [V]^2 \) consists of a subset of the 2-subsets of \( V \). For an edge \( e = \{ u, v \} \), we say that \( u \) and \( v \) are the end-points of the edge \( e \). We also denote the set of vertices and edges of graph \( \mathcal{G} \) as \( V(\mathcal{G}) \) and \( E(\mathcal{G}) \), respectively. We assume that the set of vertices is a finite subset of natural numbers, that is, \( V = [n] \) for some \( n \in \mathbb{N}_+ \).

Observe that we can easily represent graphs as relational structures where the vocabulary is \( \{ E \} \) and \#\( E = 2 \). The interpretation of the relation symbol \( E \) is always anti-reflexive and symmetric. The relation \( E \) is called the edge relation of the graph.

Let \( \mathcal{G} \) and \( \mathcal{G}' \) be graphs. The graphs are said to be isomorphic if there exists a bijection \( \pi : V(\mathcal{G}) \to V(\mathcal{G}') \) such that \( \{ u, v \} \in E(\mathcal{G}) \iff \{ \pi(u), \pi(v) \} \in E(\mathcal{G}') \) for all \( u, v \in V(\mathcal{G}) \). If \( V(\mathcal{G}) \subseteq V(\mathcal{G}') \) and \( E(\mathcal{G}) \subseteq E(\mathcal{G}') \), then \( \mathcal{G}' \) is said to be a subgraph of \( \mathcal{G}' \).
The complement graph \( \overline{G} \) of any graph \( G \) is the graph with \( V(\overline{G}) = V(G) \) and \( E(\overline{G}) = [V(G)]^2 \setminus E(G) \). For any edge \( e \in E(\overline{G}) \), the notation \( G + e \) denotes the graph \( G' \) with \( V(G') = V(G) \) and \( E(G') = E(G) \cup \{e\} \).

An \( n \)-path is a graph \( P_n \) such that the set of vertices is \( V(P_n) = [n+1] \) and the set of edges is \( E(P_n) = \{\{i, i+1\}: i \in [n]\} \). We say that the vertices 0 and \( n \) are the end-points of the path. The length of a path is the number of edges in it. We say that there is a path between two nodes \( u \) and \( v \) in \( G \), if there exists a subgraph \( G' \) of \( G \) isomorphic to \( P_k \) for some \( k \) such that \( u \) and \( v \) are the end-points of the path.

An \( n \)-cycle is a graph \( C_n \) such that \( V(C_n) = [n] \) and \( E(C_n) = E(P_{n-1}) \cup \{\{0, n-1\}\} \). Similarly, the length of a cycle is the number of edges in it. A graph \( G \) is connected if for all \( u, v \in V(G) \) there exists a path between \( u \) and \( v \). A tree is a connected graph that does not contain a cycle as a subgraph. A subgraph \( G' \) of \( G \) spans \( G \) if \( V(G') = V(G) \). In particular, a tree \( T \) is a spanning tree of \( G \), if \( T \) spans \( G \).

A Hamiltonian path of a graph is a path that spans the graph. Similarly, a graph \( G \) has a Hamiltonian circuit if it contains a subgraph isomorphic to a cycle graph that spans \( G \). That is, a Hamiltonian path visits all the nodes in the graph exactly once and a graph has a Hamiltonian circuit if it has a Hamiltonian path such that the end-points of the path are connected by an edge. A graph \( G \) has an Eulerian circuit if there exists a tuple \( (v_0, e_0, \ldots, e_{k-1}, v_{k-1}) \) such that \( v_0 = v_{k-1}, e_i \neq e_j \) for all \( i, j \in [k] \) and \( k = |E(G)| \). That is, the circuit visits all the edges in the graph exactly once.

### 2.4 First-order and existential second-order logic

This work assumes that the reader is familiar with the syntax, semantics and properties of first-order logic abbreviated as FO. These are defined in any basic textbook on mathematical logic or finite model theory \[4, 14, 15\]. However, we will now briefly overview the basic definitions in order to fix our notation.

We assume \( \{\land, \neg\} \) as the basic set of connectives for FO. In addition to these connectives, we allow existential \( \exists \) and universal \( \forall \) quantification of variables. Since \( \neg \exists x \varphi \equiv \forall x \neg \varphi \) and \( \neg \forall x \varphi \equiv \exists x \neg \varphi \), we consider negative existential quantification \( \neg \exists \) as universal quantification \( \forall \) and negative universal quantification \( \neg \forall \) as existential quantification \( \exists \).

The set of first-order formulas of vocabulary \( \tau \) is denoted by \( \text{FO}^\tau \). Recall that an FO-formula \( \psi \) is quantifier-free if neither of the quantifiers \( \exists \) and \( \forall \) appear in \( \psi \), and that an occurrence of variable \( x_i \) is free in \( \psi \) if it is not bound by some quantifier. A sentence is a formula with no free variables. We denote the set of all variables in formula \( \varphi \) with \( \text{Var}(\varphi) \). The notation \( \varphi(x/y) \) denotes the formula obtained by replacing all occurrences of symbol \( x \) in \( \varphi \) with symbol \( y \).

If a model \( \mathcal{M} \) satisfies some first-order sentence \( \varphi \), we denote this by
writing \( M \models \varphi \). If \( M \models \varphi \), we say that \( M \) is a model of \( \varphi \). An assignment \( s: \text{Var}(\varphi) \to \text{Dom}(M) \). The assignment gives each free variable symbol in \( \varphi \) an interpretation in the domain of the structure. We write \( M \models^s \varphi \) if the structure \( M \) satisfies the formula \( \varphi \) under assignment \( s \). A modified assignment is the assignment \( s(x_i/a)(x_j) = \begin{cases} a & \text{if } x_j = x_i, \\ s(x_j) & \text{otherwise} \end{cases} \) where \( x_i, x_j \in \text{Var}(\varphi) \) and \( a \in \text{Dom}(M) \).

The existential second-order logic (abbreviated as ESO) is an extension of first-order logic which allows the existential quantification of relations and functions. Let \( \tau \) and \( \sigma = \{s_0, \ldots, s_n\} \) be vocabularies such that \( \tau \cap \sigma = \emptyset \). Any formula of the form
\[
\varphi = \exists s_0 \ldots \exists s_n \psi
\]
where \( \psi \in \text{FO}^{\tau \cup \sigma} \) is said to be an existential second-order formula of vocabulary \( \tau \). We say that \( \psi \) is the first-order part of the formula \( \varphi \). The set of ESO-formulas of vocabulary \( \tau \) is denoted by \( \text{ESO}^{\tau} \).

The semantics of second-order logic are defined as follows. For any structure \( M = \langle M, \tau \rangle \), structure \( M \) satisfies \( \varphi \) if there exists a structure \( M^* = \langle M, \tau, s_0, \ldots, s_m \rangle \) such that \( M^* \models \psi \). As with FO-sentences, we write \( M \models \varphi \) if \( M \) satisfies the ESO-sentence \( \varphi \). For any sentence \( \varphi \), the set \( \text{Mod}(\varphi) = \{M: M \models \varphi\} \) denotes the set of models of \( \varphi \).

For clarity, let us define some syntactic short-hands for common formulas. First, we will often replace the block of second-order quantifiers with \( \exists \sigma \) to denote the second-order quantification of all symbols in \( \sigma \). Second, we use the short-hand \( (t_0, \ldots, t_n) = (t'_0, \ldots, t'_n) \) to denote the conjunction
\[
\bigwedge_{i=0}^n t_i = t'_i.
\]

Finally, we often use the following infix notation instead of the prefix notation when using binary relations in the formulas. If \( < \) is a binary relation symbol and \( t \) and \( t' \) are terms, then the short-hand \( t < t' \) denotes the formula \( <(t, t') \).

It is possible to normalize ESO-formulas by replacing first-order existential quantifications with second-order function quantifications. Such normalized formulas are said to be in Skolem normal form due to the classic result by Thoralf Skolem [19]. We give a proof of this result that follows the presentation of Väänänen [20].

**Theorem 1** (Skolem Normal Form). Every ESO-formula \( \varphi \equiv \exists \sigma \psi \) where \( \psi \in \text{FO}^{\tau \cup \sigma} \) is logically equivalent to an ESO-formula
\[
\varphi^* \equiv \exists f_0 \exists f_1 \cdots \exists f_n \forall x_1 \cdots \forall x_m \psi'
\]
where \( \psi' \) is a quantifier-free FO-formula and \( f_0, \ldots, f_n \) are function symbols.
Proof. The proof is an induction over the formula $\varphi$. For the base case, if $\varphi$ is atomic or a negation of an atomic formula, the formula is already in Skolem normal form. For conjunction, suppose $\varphi \equiv \psi \land \theta$ where $\psi$ and $\theta$ are already normalized formulas of the form

$$
\psi \equiv \exists f_1 \cdots \exists f_n \forall x_1 \cdots x_m \psi' \text{ and } \theta \equiv \exists g_1 \cdots \exists g_n \forall y_1 \cdots y_m \theta'.
$$

We can change bound symbols such that both the sets of quantified function symbols and variable symbols are disjoint for $\psi$ and $\theta$. Similarly, we can assume that $\{f_1, \ldots, f_n\}$ and $\{g_1, \ldots, g_n\}$ are disjoint. Now, the Skolem normal form for $\varphi$ is

$$
\exists f_1 \cdots \exists f_n \exists g_1 \cdots \exists g_n \forall x_1 \cdots x_m \forall y_1 \cdots y_m (\psi' \land \theta').
$$

Similarly, if $\varphi \equiv \psi \lor \theta$, then the Skolem normal form of $\varphi$ is

$$
\exists f_1 \cdots \exists f_n' \exists g_1 \cdots \exists g_n \forall x_1 \cdots x_m \forall y_1 \cdots y_m (\psi' \lor \theta').
$$

Suppose $\varphi \equiv \forall x_0 \theta$ where $\theta$ satisfies the induction hypothesis as before. Let $\{x_1, \ldots, x_m\}$ be the set of variables in $\theta$ in which $x_0$ does not appear. We introduce new function symbols $f_1', \ldots, f_n'$ such that $\# f_i' = \# f_i + 1$ for each $f_i$ where $i \in [n]$. Now, inductively replace each occurrence of $f_i(t_0, \ldots, t_{\# f_i - 1})$ in $\theta$ with $f_i'(x_0, t_0, \ldots, t_{\# f_i - 1})$ and let $\theta'$ denote the obtained formula. The Skolem normal form for $\varphi$ is then

$$
\exists f_1' \cdots \exists f_n' \forall x_0 \forall x_1 \cdots \forall x_m \theta'.
$$

For the case $\varphi \equiv \exists x_0 \theta$, replace the existential quantification of $x_0$ with a new nullary function symbol $f_0$ and replace all occurrences of $x_0$ in $\theta$ bound by the replaced quantifier with the symbol $f_0$. Let $\theta'$ denote the obtained formula. Now the Skolem normal form of $\varphi$ is $\exists f_0 \theta'$.

For second-order quantifications, consider the case $\varphi \equiv \exists R \theta$ where $R$ is a $k$-ary relational symbol. We introduce a new $k$-ary function symbol $f_{n+1}$ and a nullary function symbol $f_{n+2}$. Let $\theta'$ denote the formula obtained by replacing all occurrences of the atomic formulas $R(t_0, \ldots, t_{k-1})$ in $\theta$ with the formula $f_{n+1}(t_0, \ldots, t_{k-1}) = f_{n+2}$. The Skolem normal form for $\varphi$ is then $\exists f_{n+1} \exists f_{n+2} \theta'$. Finally, for the case $\varphi \equiv \exists f \theta$, the Skolem normal form is simply $\exists f \theta$. \qed

Throughout this work, we say that a formula $\varphi$ is Skolemized to refer to the transformation into the normal-form $\varphi^*$.

**Definition 2.** Let $\tau$ and $\sigma$ be vocabularies and $\varphi \equiv \exists \sigma \psi$ a formula in $\text{ESO}^\tau$ such that $\psi \in \text{FO}^{\tau \cup \sigma}$. We define the following fragments of ESO:

- $\varphi \in \text{ESO}^\tau[\text{Var} \; d]$ if $\varphi$ contains at most $d$ different FO variable symbols,
• \( \varphi \in \text{ESO}^\tau[\forall d] \) if \( \psi \equiv \forall x_0 \cdots \forall x_{d-1} \psi' \) where \( \psi' \) is quantifier-free,

• \( \varphi \in \text{ESO}^\tau[#k, \forall d] \) if \( \varphi \in \text{ESO}^\tau[\forall d] \) and \( #\sigma \leq k \).

To denote the set of all formulas of in any of these logics over finite vocabularies, we omit the superscript \( \tau \).

It is easy to see that the following inclusions hold:

\[
\text{ESO}^\tau[#k, \forall d] \subseteq \text{ESO}^\tau[\forall d] \subseteq \text{ESO}^\tau[\text{Var } d].
\]

In fact, for any vocabulary \( \tau \) and \( d \in \mathbb{N}_+ \), it is possible to prove the following equivalence [12]

\[
\text{ESO}^\tau[#d, \forall d] = \text{ESO}^\tau[\forall d] = \text{ESO}^\tau[\text{Var } d].
\]

However in this work, we shall omit this proof and we concentrate solely on
the logic \( \text{ESO}[#d, \forall d] \), and in particular the logic \( \text{ESO}[#1, \forall 1] \) with unary second-order quantifications and only one universal first-order quantification. It has been shown that if \( #\tau \leq 1 \) the logics \( \text{ESO}^\tau[#2, \forall 1] \) and \( \text{ESO}^\tau[#1, \forall 1] \) coincide and in special cases, any additional universal first-order variables do not increase the expressibility of the formulas [3].

The following lemma will be often utilized when constructing formulas in
the logic \( \text{ESO}[#1, \forall 1] \).

**Lemma 2.1.** Let \( \sigma \) be a vocabulary such that \( #\sigma \leq 1 \). Any formula

\[
\varphi \equiv \exists \sigma \forall x_0 \exists x_1 \cdots \exists x_m \psi
\]

where \( \psi \) is quantifier-free first-order formula can be expressed in \( \text{ESO}[#1, \forall 1] \).

**Proof.** By Skolemizing the first-order part of \( \varphi \) we get a second-order formula of the form \( \varphi' \equiv f_0 \cdots f_m \psi' \) where \( #f_i = 1 \) for all \( i \in [k] \). Now, it is easy to check the second-order formula \( \exists \sigma \varphi' \) satisfies the syntactic conditions of the logic \( \text{ESO}[#1, \forall 1] \). \( \square \)

### 3 Non-deterministic random access machines

In this section, we define the non-deterministic random-access machine which
is the model of computation used in this work. This model has notable
 differences compared to the more traditional Turing machines. Random-access machines, often abbreviated as RAMs, are somewhat easier to program
and resemble real-world computing machines much more closely than the well-established and tape-based Turing machines. Similarly, non-deterministic
RAMs (NRAMs) introduce non-determinism to the computation much like
non-determinism in Turing machines (NTM).
While both models are Turing complete, the type of random-access machines considered in this work is slightly more powerful than Turing machines if the running time is bounded. That is, a RAM machine may need fewer steps of computation than a Turing machine to solve the same problem. Furthermore, memory lookup is a constant-time operation and each memory location, or register, can store $O(\log n)$ bits. In contrast, Turing machines need to search the tape for the correct memory location and each location can store only a constant number of bits.

For example, Monien [17] studied how much more powerful RAMs are compared to Turing machines. First, a random-access machine with running time $T(n)$ can be simulated with a Turing machine that uses at most $O(T^2(n) \log T(n))$ steps. Second, a linear-time NRAM can be simulated in $O(n(\log n)^3)$ steps with an NTM, whereas a linear-time NRAM can simulate an NTM whose running time is bounded by $O(n \log n)$. The NRAM model of Monien is essentially the same as the one studied in this work.

With these differences in mind, it is easy to conclude that the RAM model is a feasible alternative for Turing machines as they are not too powerful compared to Turing machines, and in addition, are somewhat easier to work with.

3.1 Input and registers

Let $\tau$ be a finite vocabulary. A non-deterministic random access machine of vocabulary $\tau$ takes a finite $\tau$-structure as input and outputs one bit to denote whether the machine accepts or rejects the input. We assume that the domain of the input is $[n]$ for some $n \in \mathbb{N}$. The input structure $\mathcal{M}$ is stored into read-only input registers. The cardinality of the input structure $\mathcal{M}$ is stored into the register $N$. For each function symbol $f \in \text{Fun}(\tau)$ and tuple $x \in [n]^{|f|}$, the register $f[x]$ stores the value $f^\mathcal{M}(x)$. If $|f| = 0$, then register $f$ simply stores a single value. For each relational symbol $R \in \text{Fun}(\tau)$ and $x \in [n]^{|R|}$, the register $R[x]$ stores the value 1 if $x \in R^\mathcal{M}$ and otherwise the value 0.

In addition to the read-only input registers, there are additional accumulator registers called $A, B_0, \ldots, B_r$ where $r = |\tau|$. These registers are used to access the input registers to store any temporary values. Finally, a $\tau$-NRAM has countably infinite number of registers $R_0, R_1, \ldots$ available as the main memory. Initially, each accumulator and main memory register contains value 0.

Figure 1 illustrates how graph structures are represented in the input registers of a $\{E\}$-NRAM. Essentially, the adjacency matrix of the graph is stored into the registers $E[x]$ where $x \in [n]^2$. 
3.2 Instructions and computation

A $\tau$-NRAM consists of a program which is a finite sequence $\langle I_0, \ldots, I_k \rangle$ of labeled instructions. The instructions can be of the following form:

1. $A \leftarrow N$
2. $A \leftarrow s[B_0, \ldots, B_q]$ where $q = \#s$
3. $A \leftarrow 0$
4. $A \leftarrow A + 1$
5. $A \leftarrow R_A$
6. $B_i \leftarrow A$ where $i \in [r + 1]$
7. $R_A \leftarrow B_i$ where $i \in [r + 1]$
8. $\text{guess}(A)$
9. if $A = B_i$ then jump to $I_a$ else jump to $I_b$, where $a, b \in [k + 1]$
10. accept
11. reject

where $s \in \tau$ and $r = \#\tau$. The notation $R_A$ denotes register $R_i$ where $i$ is the content of register $A$.

The computation begins from instruction $I_0$ and after executing instruction $I_m$ with $m \in [k + 1]$ the machines moves onto instruction $I_{m+1}$ with the exception of instructions (9), (10) and (11). In case the jump instruction (type 9) is encountered, the machine checks if the contents of accumulator $A$ equal to contents of accumulator $B_i$. If so, the machine executes instruction $I_a$ from which the computation proceeds sequentially. Otherwise, the machine executes instruction $I_b$.

If the machine encounters the accept instruction, the computation halts and the machine outputs accept. Similarly, if the machine encounters the reject instruction, the machine rejects the input structure.
The guess instruction introduces non-determinism to the model. The guess instruction stores any integer bounded by $O(n)$ into register $A$. If some computation accepts, then the machine will accept. Otherwise, the machine rejects the input. The running time of a machine on input structure $\mathcal{M}$ is the number of instructions executed by the machine during the computation.

3.3 Problems and complexity classes

Let $\tau$ be a vocabulary. A $\tau$-problem is a set $P \subseteq \text{Str}(\tau)$ which is closed under isomorphism. A $\tau$-machine $M$ that recognizes some problem $P$ is a machine that accepts the input structure $\mathcal{M}$ if $\mathcal{M} \in P$ and rejects $\mathcal{M}$ when $\mathcal{M} \notin P$. As an example, the Hamiltonian circuit problem consists of deciding whether a $\{E\}$-structure is in the set

$$\{ \mathcal{G} : \mathcal{G} \text{ is a graph with a Hamiltonian circuit} \} \subseteq \text{Str}(\{E\}).$$

The complement problem of any problem $P$ is the negation of the problem denoted by $\overline{P}$. For example, the complement of the Hamiltonian circuit problem is

$$\{ \mathcal{G} : \mathcal{G} \text{ is a graph with no Hamiltonian circuit} \} \subseteq \text{Str}(\{E\}).$$

Let $T : \mathbb{N}_+ \to \mathbb{N}_+$. We say that a $\tau$-problem $P$ belongs to the complexity class $\text{NTIME}^\tau[T(n)]$ if

(i) there exists a non-deterministic $\tau$-machine $M$ that recognizes problem $P$,

(ii) there exists a constant $c \in \mathbb{N}_+$ such that for all structures $\mathcal{M} \in P$, the running time of $M$ is bounded by $cT(n)$ where $n = \text{Card}(\mathcal{M})$.

In particular, if $P \in \text{NTIME}[n]$ we say that $P$ is solvable in linear-time on non-deterministic random-access machines. For any complexity class $C$, the complement class $\overline{C}$ consists of the complement problems of $C$.

In Section 4 we will see that the logic $\text{ESO}^{\#1, \forall 1}$ captures the complexity class $\text{NTIME}[n]$. We denote this correspondence by writing $\text{ESO}^{\#1, \forall 1} \equiv \text{NTIME}[n]$ which formally means that $P \in \text{NTIME}[n]$ if and only if there is a sentence $\varphi \in \text{ESO}^\tau[\#1, \forall 1]$ such that $\text{Mod}(\varphi) = P$.

4 The complexity of ESO

In this section, we will show that the logic $\text{ESO}^\tau[\#1, \forall 1]$ corresponds to linear time computation on NRAMs. The proof is similar to the proof of Fagin’s theorem. Unfortunately, unlike in the proof of Fagin’s theorem, the fragment $\text{ESO}^\tau[\#1, \forall 1]$ does not directly allow the quantification of a binary
relation or expressing that a relation is a linear order. However, a linear order is a crucial part of the proof.

Fortunately, Grandjean [9] presented a way to implicitly define a linear order over a finite model using a formula with unary second-order existential quantifications and a single first-order universal quantification. We begin by presenting this rather intricate construction before showing that ESO[♯₁, ∀₁] ≡ NTIME[n].

4.1 Linear order with a single variable

In this section, we will prove the following theorem due to Grandjean [9, 12]:

**Theorem 2.** The class of structures definable in the logic ESO[♯₁, ∀₁] is not enlarged by the addition of a second-order quantifier ∃₉ ORDER less than which states that the interpretation of < is a linear order.

To achieve this result, we construct a first-order formula with one universal quantifier that characterizes structures isomorphic to an arithmetical n-structure. All structures isomorphic to the arithmetical n-structure contain an implicitly defined linear order over the universe of the structure.

**Definition 3.** Let $v$ be a vocabulary that consists of the nullary function symbols \{0, 1, a, b\}, unary function symbols \{S, SUC, f₀, f₁, f₂, ℓ, r\}, and unary relation symbols \{A, REP, ORD\}. Let $n ≥ 4$ and $m = \lfloor √n \rfloor$. The $v$-structure $A_n = ⟨[n], v⟩$ is an arithmetical n-structure if the following conditions hold:

1. for function $S$: $[n] → [n]$, we have $S(i) = i + 1 \mod n$ for all $i ∈ [n]$;
2. $S(0) = 0$;
3. for each $e < n$: $e ∈ A$ if and only if $e < m$;
4. for each $e < m$: SUC($e$) = $S(e) \mod m$, $1 = 1$, and $a = m - 1$;
5. for each $e = im² + jm + k < n$ where $i, j, k ∈ [m]$ it holds that
   $$f₂(e) = i, \quad f₁(e) = j, \quad f₀(e) = k;$$
6. for each $e < n$: $e ∈ REP$ if and only if $e < m²$;
7. for each $e = jm + k < m²$ where $j, k ∈ [m]$ it holds that $ℓ(e) = j$ and $r(e) = k$ and $e ∈ ORD$ if and only if $j < k$. In addition, for each $e ≥ m²$ it holds that $e /∈ ORD$;
8. for each $e ≥ m$ it holds that SUC($e$) = 0 and for each $e ≥ m²$ we have $ℓ(e) = r(e) = 0.$
Before proceeding, we will first give an informal overview of the arithmetical structure. The first two conditions enforce that the interpretation of \( S \) is a successor function over the universe \([n]\) such that the successor of the “last element” \( b \) is the “first element” \( 0 \).

Conditions (3)–(7) ensure that the arithmetical structure contains an \( m \)-subset \( A \subset [n] \) on which the successor function \( \text{SUC} \) is defined. The elements \( 0 \) and \( a \) being the first and last elements of \( A \). Each ordered pair in the set \( A^2 \) has a representative in the set \( \text{REP} \). That is, each \((i,j)\in A^2\) is encoded into an element \( e \in \text{REP} \).

Moreover, the unary relation \( \text{ORD} \) defines a linear order over \( A \subset [n] \): an element \( e \in \text{ORD} \) if and only if \( \ell(e) <_A r(e) \) where \( _A \) is the linear order induced by the successor relation \( S \) on \( A \). Finally, condition (8) merely ensures that all values of \( \text{SUC} \), \( \ell \), and \( r \) are fixed.

Now, we will construct a formula \( \Phi_\upsilon \) that is satisfiable by a \( \upsilon \)-structure \( M \) if and only if it is isomorphic to the arithmetical structure \( A_n \). We will follow the presentation of Grandjean [9] in constructing the formula, and give the formula as a conjunction of twenty subformulas. We begin with the first six subformulas:

\[
\varphi_1: \forall x \left[ A(x) \rightarrow A(\text{SUC}(x)) \right]
\]

\[
\varphi_2: \forall x \left[ A(x) \rightarrow \exists y [A(y) \land x = \text{SUC}(y)] \right]
\]

\[
\varphi_3: A(0) \land A(a) \land SUC(a) = 0 \land a \neq 0
\]

\[
\varphi_4: \forall x \left[ (A(x) \land x \neq a) \rightarrow \exists y \left( (\ell(y), r(y)) = (x, \text{SUC}(x)) \land \text{ORD}(y) \right) \right]
\]

\[
\varphi_5: \forall x \left[ (\text{ORD}(x) \land r(x) \neq a) \rightarrow \exists y \left( (\ell(y), r(y)) = (\ell(x), \text{SUC}(r(x))) \land \text{ORD}(y) \right) \right]
\]

\[
\varphi_6: \forall x \left[ \ell(x) = r(x) \rightarrow \neg \text{ORD}(x) \right]
\]

**Lemma 4.1.** Let \( M \) be a finite \( \upsilon \)-structure such that \( M \models \bigwedge_{i \leq 6} \varphi_i \) and \(|A| = m\). Then the following holds:

1. The function \( \text{SUC}|A \) (restriction of \( \text{SUC} \) to domain \( A \)) is a permutation of \( A \) with one cycle.

2. If holds that \( m \geq 2 \) and each \( e \in A \) is the value of exactly one term of the form \( \text{SUC}^i(0) \) where \( i \in [m] \).

3. Let \( x <_A y \equiv \exists z [\text{ORD}(z) \land (\ell(z), r(z)) = (x, y)] \). For all \( i, j \in [m] \) it holds that \( i < j \) if and only if \( M \models \text{SUC}^i(0) <_A \text{SUC}^j(0) \).
Proof. (1) The formula $\varphi_1 \land \varphi_2$ ensures that each element $e \in A$ has a successor in $A$ and $e$ is a successor to some other element in $A$. Thus, $\text{SUC}|_A$ is surjective. Since $A \subseteq \text{Dom}(M)$ is finite, this also implies that $\text{SUC}|_A$ is injective and therefore bijective.

It follows from $\varphi_4$ that for all $e \in A \setminus \{a\}$ there is an element $t \in \text{ORD}$ encoding the order between $e$ and its successor $\text{SUC}(e)$. That is, $(\ell(t), r(t)) = (e, \text{SUC}(e))$. Moreover, $\varphi_5$ states that the implicit ordering $<_A$ is transitive: for each $t \in \text{ORD}$ such that $r(t) \neq a$, there is a pair $(\ell(t), e')$ where $e' = \text{SUC}(r(t))$.

To show that $\text{SUC}|_A$ has only one cycle, suppose the bijection has a cycle $C \subset A$ distinct from the the cycle of 0 and $a$. Let $i = |C|$ be the length of the cycle and $e \in C$. Now, by $\varphi_4 \land \varphi_5$ there exists $t \in \text{ORD}$ such that $(\ell(t), r(t)) = (e, \text{SUC}(e))$, which contradicts $\varphi_6$.

(2) It holds that $|A| \geq 2$, as formula $\varphi_3$ defines that 0 $\neq$ a and both belong to $A$. The second claim follows from (1) as $\text{SUC}|_A$ is a bijection with only one cycle.

(3) $\Rightarrow$: Let $i, j \in [m]$ such that $i < j$. Let $x = \text{SUC}^i(0)$. Now by $\varphi_4$ there exists a $t \in \text{ORD}$ which encodes the tuple $(x, \text{SUC}(x))$, and thus, $x <_A \text{SUC}(x)$. Now, suppose $x <_A \text{SUC}^k(x)$ for some $k < m - 1$: there is a $t' \in \text{ORD}$ such that encodes the pair $(x, \text{SUC}^k(x))$. Now by $\varphi_5$, there exists an element $t'' \in \text{ORD}$ that encodes $(x, \text{SUC}^{k+1}(x))$. Thus, for all $j > i$ there is a $t \in \text{ORD}$ that encodes $(\text{SUC}^i(0), \text{SUC}^j(0))$.

$\Leftarrow$: Suppose the opposite holds: for some $i, j \in [m]$ such that $j \leq i$ it holds that $M \models \text{SUC}^i(0) <_A \text{SUC}^j(0)$. Now, we have

$$
M \models \text{SUC}^i(0) <_A \text{SUC}^j(0) \iff M \models \exists z [\text{ORD}(z) \land (\ell(z), r(z)) = (\text{SUC}^i(0), \text{SUC}^j(0))].
$$

That is, for some $z \in \text{ORD}$, it holds that $(\ell(z), r(z)) = (\text{SUC}^i(0), \text{SUC}^j(0))$. Since $M \models \varphi_4 \land \varphi_5$ it also holds that there is an element $z' \in \text{ORD}$ such that $(\ell(z'), r(z')) = (\text{SUC}^i(0), \text{SUC}^{i-j}(\text{SUC}^j(0)))$ due to transitivity. Since $\text{SUC}^i(0) = \text{SUC}^{i-j}(\text{SUC}^j(0))$ it must hold that $M \not\models \varphi_6$ which is contradictory.

So far, the first six sentences only encode a linear order on the subset $A$ with respect to the successor function of $A$. We can now use $A$ to define the successor function $S$ over the whole domain of the model.

$\varphi_7$: $\forall x \exists y [S(y) = x]$

$\varphi_8$: $S(b) = 0$

$\varphi_9$: $\forall x [A(f_0(x)) \land A(f_1(x)) \land A(f_2(x))]$
Sentence \( \varphi_7 \) expresses that \( S \) is a permutation of the domain and \( \varphi_8 \) says that the successor of the “last element” \( b \) in the domain is 0. In the following, we use the short-hand \( f(t) \) to denote the triple \((f_2(t), f_1(t), f_0(t))\). Thus by sentence \( \varphi_9 \), the range of the implicit function \( f \) is a subset of \( A^3 \).

\[
\varphi_{10} \quad \forall x \left[ (x \neq b \land f_0(x) \neq a) \rightarrow f(S(x)) = (f_2(x), f_1(x), \text{SUC}(f_0(x))) \right]
\]

\[
\varphi_{11} \quad \forall x \left[ (x \neq b \land f_0(x) = a \land f_1(x) \neq a) \rightarrow f(S(x)) = (f_2(x), \text{SUC}(f_1(x)), 0) \right]
\]

\[
\varphi_{12} \quad \forall x \left[ (x \neq b \land f_0(x) = a \land f_1(x) = a) \rightarrow f(S(x)) = (\text{SUC}(f_2(x)), 0, 0) \right]
\]

\[
\varphi_{13} \quad \forall x \left[ f(x) = (0, 0, 0) \leftrightarrow x = 0 \right]
\]

The above statements use the successor function \( \text{SUC} \) of the subset \( A \) to define a successor function \( S \) for the whole domain of the structure. In particular, \( \text{SUC} \) allows us to refer to the lexicographic order of \( A^3 \) as follows. The conjunction \( \varphi_{10} \land \varphi_{11} \land \varphi_{12} \) implies that for each element \( x \neq b \) in the universe, \( f(S(x)) \) is the immediate successor of \( f(x) = (f_2(x), f_1(x), f_0(x)) \) in the lexicographic order of \( A^3 \).

We now prove some properties of the functions \( S \) and \( f \) defined by the above sentences.

**Lemma 4.2.** Let \( \mathcal{M} \) be a \( \nu \)-structure. If \( \mathcal{M} \models \bigwedge_{i \leq 13} \varphi_i \), \( n = \text{Card}(\mathcal{M}) \) and \( m = |A| \), then the following conditions hold:

1. Function \( S \) is a permutation of \( \text{Dom}(\mathcal{M}) \) with only one cycle and \( S(b) = 0 \).

2. Each element of \( \text{Dom}(\mathcal{M}) \) is the value of exactly one term \( S^h(0) \) where \( h \in [n] \). In particular, \( b = S^{n-1}0 \).

3. For each \( h \in [n] \) it holds that

\[
f(S^h(0)) = (\text{SUC}^i(0), \text{SUC}^j(0), \text{SUC}^k(0))
\]

where \( h = im^2 + jm + k \) and \( i, j, k \in [m] \).

**Proof.** (1) The formula \( \varphi_7 \) states that \( S \) is surjective. Thus, \( S \) is bijective since the domain of the structure is finite. Suppose \( S \) contains a cycle \( C \) distinct from the cycle of \( 0 \) and \( b \). Let \( e \in C \). From \( \varphi_9 \) it follows that \( f(e) \in A^3 \) and from Lemma 4.1.2 it follows that

\[
f(e) = (\text{SUC}^i(0), \text{SUC}^j(0), \text{SUC}^k(0))
\]
for some $i, j, k \in [m]$. Now, from $\varphi_{10} \land \varphi_{11} \land \varphi_{12}$ it follows that there exists $h$ such that after $h$ iterations of $S$ it holds that

$$f(S^h(e)) = (\text{SUC}^{m-1}(0), \text{SUC}^{m-1}(0), \text{SUC}^{m-1}(0)) = (a, a, a)$$

and from sentence $\varphi_{12}$ it follows that

$$f(S^{h+1}(e)) = (0, 0, 0) \iff S^{h+1}(e) = 0$$

which contradicts that the cycle $C$ which contradicts that the cycle $C$ was distinct from the cycle of $0$ and $b$.

(2) This follows from (1).

(3) We prove the claim by induction on $h$. The base case $h = 0$ holds trivially. Suppose the claim holds for $h < n$, that is,

$$f(S^h) = (\text{SUC}^i(0), \text{SUC}^j(0), \text{SUC}^k(0))$$

for some $i, j, k \in [m]$. We now consider three cases. First, if $k \neq m - 1$, then by $\varphi_{10}$ we have

$$f(S(S^h(0))) = f(\text{SUC}^i(0), \text{SUC}^j(0), \text{SUC}^{k+1}(0))$$

and the claim holds with $h + 1 = im^2 + jm + k + 1$. Second, if $k = m - 1$ and $j \neq m - 1$, by $\varphi_{11}$ we get

$$f(S(S^h(0))) = f(\text{SUC}^i(0), \text{SUC}^{j+1}(0), 0)$$

with $h + 1 = im^2 + (j + 1)m + 0$. Finally, the last case $i = j = m - 1$ is covered by the sentence $\varphi_{12}$ and we get $f(S(S^h(0))) = f(\text{SUC}^{i+1}(0), 0, 0)$. $\square$

Lemma 4.2 has some useful consequences. First, by Lemma 4.2.2 we know that a model of $\bigwedge_{i \leq 13} \varphi_i$ has a natural linear order: $S^j(0)$ precedes $S^i(0)$ if and only if $i < j$. By Lemma 4.2.3, for this linear order and for the lexicographic order of $A^3$, the implicit function $f : \text{Dom}(M) \to f(\text{Dom}(M)) \subseteq A^3$ is a bijection in $[(0, 0, 0), f(b)] \subseteq A^3$. Moreover, since for each $h < n$ it holds that $h = im^2 + jm + k < m^3$ we have that $n \leq m^3$.

$\varphi_{14}$: $\text{SUC}(0) = 1$

$\varphi_{15}$: $\forall z [\text{REP}(z) \iff f_2(z) = 0]$

$\varphi_{16}$: $\forall z [\text{REP}(z) \to (\ell(z), r(z)) = (f_1(z), f_0(z))]$

$\varphi_{17}$: $f(b) = (0, a, a) \lor [f_2(b) = 1 \land (f_1(b) = 0 \lor f_1(b) = 1)]$

From Lemma 4.2.3 it follows that sentence $\varphi_{15}$ defines the interpretation of $\text{REP}$ to be the set $\{S^h(0) : h \in [m^2]\}$. Sentence $\varphi_{16}$ states that each element $S^h(0) \in \text{REP}$ encodes a pair of the form $(\ell(z), r(z)) = (S^j(0), S^k(0))$ where
j, k ∈ [m] such that h = jm + k. Thus, each (x, y) ∈ A^2 has exactly one 
z ∈ REP such that (ℓ(z), r(z)) = (x, y).

Sentence ϕ_{17} implies that f(b) resides between the values (0, a, a) and 
(1, 1, a) in the lexicographic order of A^3. Thus, (0, a, a) corresponds to 
0m^2 + (m − 1)m + (m − 1) and (1, 1, a) corresponds to m^2 + m + (m − 1). 
Therefore, we have that

\[(0, a, a) \leq_{lex} f(b) \leq_{lex} (1, 1, a)\]
\[\iff m^2 - 1 \leq n - 1 \leq m^2 + 2m - 1\]
\[\iff m^2 \leq n < m^2 + 2m + 1 = (m + 1)^2\]
\[\iff m = \lfloor \sqrt{n} \rfloor\]

where \(\leq_{lex}\) denotes the lexicographic order of \(A^3\).

We can now give the final three sentences that complete the characteriza-
tion of arithmetical \(n\)-structures:

ϕ_{18}: \(∀x[\text{ORD}(x) \to \text{REP}(x)]\)
ϕ_{19}: \(∀x[(A(x) ∧ a \neq x) \to S(x) = SUC(x)]\)
ϕ_{20}: \(∀x[\neg A(x) \to SUC(x) = 0 ∧ \neg \text{REP}(x) \to (ℓ(x), r(x)) = (0, 0)]\)

Sentence ϕ_{18} simply states that ORD is a subset of REP. Sentence ϕ_{19} 
declares that the successor function \(S\) of the whole domain extends the 
successor function SUC over the subdomain \(A\). Sentence ϕ_{20} ensures that 
the interpretations SUC, ℓ, and r match the definition of the arithmetical 
\(n\)-structure.

Let \(Φ_v\) be a \(v\)-sentence such that

\[Φ_v ≡ \bigwedge_{i \leq 20} ϕ_i\]

Observe that no formula ϕ_i for \(0 \leq i \leq 20\) contains more than one universal 
quantification. Thus by Lemma 2.1, we can Skolemize the formula \(Φ_v\) which 
results in a logically equivalent formula in ESO[#1, ∀1].

Lemma 4.3. Let \(n \geq 4\) and \(A_n\) be an arithmetical \(n\)-structure. Then 
\(A_n \models Φ_v\).

Proof. Since \(Φ_v\) is a conjunct of twenty sentences, we will list the conditions 
given in the definition of arithmetical structures which satisfy each sentence.

Due to conditions (3) and (4) we have \(A_n \models ϕ_i\) for \(1 \leq i \leq 3\). Condition 
(7) implies \(A_n \models ϕ_i\) when \(4 \leq i \leq 6\). Formulas ϕ_7 and ϕ_8 hold due to 
conditions (1) and (2), whereas \(A_n \models ϕ_9\) follows from conditions (3) and (5). 
Moreover, condition (5) implies \(A_n \models ϕ_i\) for \(10 \leq i \leq 13\). 

Sentences ϕ_{14} and ϕ_{15} hold due to conditions (1) and (6), respectively. Sentence ϕ_{16} 
follows from conditions (5) and (6). Condition (5) implies that 
ϕ_{17} holds and conditions (6) and (7) yield ϕ_{18} holds in \(A_n\). By condition (4) 
sentence ϕ_{19} holds and finally, ϕ_{20} holds due to condition (8).
**Lemma 4.4.** Let $A_n$ be the standard arithmetical $n$-structure where $n \geq 4$ and $\mathcal{M} \in \text{Str}(\nu)$ with $\text{Card}(\mathcal{M}) = n$. Then the following holds:

$$\mathcal{M} \models \Phi_\nu \iff \mathcal{M} \cong A_n.$$  

**Proof.** Suppose $\mathcal{M}$ is a finite model of $\Phi_\nu$. We will now check that each condition given in Definition 3 holds.

**Conditions (1) and (2):** $S(i) = i + 1 \mod n$ and $S(b) = 0$. Let $\pi : A_n\{S, 0, b\} \cong \mathcal{M}\{S, 0, b\}$ be the bijection given by $\pi(i) = S^i(0)$. By Lemmas 4.2.1 and 4.2.2. $\pi$ is an isomorphism.

**Condition (3):** $e \in A \iff e < m$. Sentence $\varphi_{17}$ states that $|A| = m = \lceil \sqrt{n} \rceil$. Suppose $S^e(0) \in A$. By $\varphi_{19}$ we have $S^e(0) = \text{SUC}^e(0)$. Due to Lemma 4.1.2 it follows that $e < m$. Conversely, suppose $e < m$. Now $S^e(0) = \text{SUC}^e(0)$ and by Lemma 4.1.2 we have $\text{SUC}^e(0) \in A$.

**Condition (4):** $\text{SUC}(e) = e + 1 \mod m$, $1 = 1$, $a = m - 1$. The first claim was proven in Lemma 4.1.2. Sentences $\varphi_{14}$ and $\varphi_{19}$ give us $1 = \text{SUC}(0) = S(0)$. By $\varphi_{13}$ we get $\text{SUC}(a) = 0$ and by Lemma 4.1.1 it holds that $a = \text{SUC}^{m-1} = S^{m-1}$.

**Condition (5):** Lemma 4.2.1.

**Condition (6):** $e \in \text{REP} \iff e < m^2$. Recall that $a = S^{m-1}(0)$. Now, by sentence $\varphi_{15}$ we have

$$S^e(0) \in \text{REP} \iff f_2(S^e(0)) = 0$$

$$\iff e \leq (m - 1)m + (m - 1) = m^2 - 1.$$  

**Condition (7):** The first two conditions are satisfied by Lemma 4.2.3 and Lemma 4.1.3. For all $e \geq m^2$ we have $e \notin \text{ORD}$ since $e \in \text{ORD} \subset \text{REP}$ and $e \in \text{REP} \iff e < m^2$.

**Condition (8):** For all $e \geq m$, $\text{SUC}(e) = l(e) = r(e) = 0$. This follows from sentence $\varphi_{20}$. \hfill $\square$

We have now established the following proposition:

**Proposition 1.** If $\mathcal{M} \in \text{Str}(\nu)$ and $A$ is a arithmetical $n$-structure where $n \geq 4$, then the following holds:

$$\mathcal{M} \models \Phi_\nu \iff \mathcal{M} \cong A.$$  

To define a linear order over an arithmetical $n$-structure, let $\text{order}(x, y)$ denote the following first-order formula:

$$\text{order}(x, y) \equiv f_2(x) <_A f_2(y)$$

$$\lor \left[ f_2(x) = f_2(y) \land f_1(x) <_A f_1(y) \right]$$

$$\lor \left[ f_2(x) = f_2(y) \land f_1(x) = f_1(y) \land f_0(x) <_A f_0(y) \right].$$

19
Lemma 4.5. Let $A_n$ be an arithmetical $n$-structure where $n \geq 4$. The relation $R = \{(x, y) \in M^2 : \text{order}(x, y)\}$ is a linear order.

Proof. Recall that $<_A$ defines a linear order over the set $[m]$ in the arithmetical $n$-structure. Since $n \leq m^3$, the the formula $\text{order}(x, y)$ defines a lexicographic order over the domain $[n] \subseteq [m^3]$ of the arithmetical structure. \hfill \Box

Now all that remains is to prove the theorem stated in the beginning of this section.

Theorem 2. The class of structures definable in $\text{ESO}^T[#1, \forall 1]$ is not enlarged by the addition of a second-order quantifier $\exists_{\text{ord}} <$ that states that the interpretation of $<$ is a linear order. That is, any formula

\[
\theta \equiv \exists_{\text{ord}} < \psi
\]

where $\psi \in \text{ESO}^{\tau \cup \{<\}}[#1, \forall 1]$ is equivalent to a formula $\theta' \in \text{ESO}^T[#1, \forall 1]$.

Proof. Suppose $\psi$ is a formula of the vocabulary $\tau \cup \{<\}$. Without loss of generality, we can assume that the binary relation symbol $<$ appears only positively in $\psi$: negative appearances of the subformula $\neg(x < y)$ can be replaced with $(y < x \lor x = y)$.

Let $\psi'$ be the formula obtained from replacing every subformula of the form $t_1 < t_2$ in $\psi$ with the formula $\text{order}(t_1, t_2)$. Now, we can define

\[
\theta' \equiv \exists_v [\Phi_v \land \psi']
\]

where $\exists_v$ is the second-order quantification of symbols in vocabulary $v$ and $\Phi_v$ is the formula characterizing arithmetical structures. However, this only works when the cardinality of the input structure is at least four.

For the cases where the cardinality is less than four, we will simply enumerate the linear order in the formula itself as follows. Let $x_0, \ldots, x_3$ be symbols that do not appear in $\psi$. Define $\text{order}_1(x, y) \equiv x \neq y$ to be a formula that is always false, $\text{order}_2(x, y) \equiv (x = x_0 \land y = x_1)$, and $\text{order}_3(x, y) \equiv \text{order}_1(x, y) \lor (x = x_1 \land y = x_2)$. For each $i \in \{1, 2, 3\}$, let $\psi'_i$ be the formula $\psi'$ with each occurrence of formula $\text{order}(x, y)$ replaced with $\text{order}_i(x, y)$. Now, we can construct the formula

\[
\theta'' \equiv \exists x_0 \exists x_1 \exists x_2 \forall x_3 \left( x_0 = x_3 \leftrightarrow \psi'_1 \land \left( (x_0 \neq x_1 \land (x_0 = x_3 \lor x_1 = x_3)) \leftrightarrow \psi'_2 \land \left( \land_{i,j \in [3], i \neq j} x_i \neq x_j \land \lor_{i \in [3]} x_i = x_3 \right) \leftrightarrow \psi'_3 \right) \right)
\]

which covers the models with cardinality less than four. Finally, the conjunction $\theta' \land \theta''$ can be written in the logic $\text{ESO}[^T, \forall 1]$. \hfill \Box
Observe that due to the construction of formula $\Psi$, it is always possible to define the corresponding unary successor function $S$ and the first (and last) element of the linear order in $\text{ESO}[\#1, \forall 1]$. For this reason, we adopt the notation $\exists_{\text{ord}}(<, S, 0)$ to denote the quantification of a linear order $<$, its successor function $S$ along with the first element $0$ in the ordering.

4.2 A characterization of linear time computation

Lemma 4.6. Let $\tau$ be a vocabulary and $d \in \mathbb{N}_+$. Then

$$\text{ESO}^\tau[\#d, \forall d] \subseteq \text{NTIME}[n^d].$$

Proof. Let $\phi \equiv \exists \sigma \forall x_0 \cdots \forall x_{d-1} \psi$ be a formula in $\text{ESO}^\tau[\#d, \forall d]$ such that $\psi$ is a quantifier-free first-order formula of vocabulary $\tau \cup \sigma$ and $\sigma = \{s_0, \ldots, s_{m-1}\}$ is a vocabulary such that $\#\sigma \leq d$. We will now describe an algorithm which recognizes the set $P = \text{Mod}(\phi)$ of $\tau$-structures. The algorithm can be implemented by an NRAM running in time $O(n^d)$.

Suppose $M = ([n], \tau) \in \text{Str}(\tau)$ is the input structure. First, the algorithm begins by non-deterministically constructing an interpretation for each symbol in $\sigma$. Each relation or function is stored into the main registers. Let $M' = ([n], \tau, \sigma)$ be the expanded structure with interpretations for symbols in $\sigma$. In the second step, the algorithm checks whether $M' \models \forall x_0 \cdots \forall x_{d-1} \psi$. That is, the algorithm iteratively checks that the expanded structure satisfies $\psi$ for all $n^d$ different assignments of the variables. If yes, the algorithm accepts $M$ and otherwise rejects $M$.

By construction, the algorithm accepts all structures in the set $P$. We now have to show that the algorithm is implementable with a $\tau$-NRAM whose running time bounded by $O(n^d)$. In the first step, the machine guesses the $m$ interpretations for the relation and function symbols in $\sigma$. For each symbol $s \in \sigma$ with $\#s \leq d$, the machine guesses a value for each vector in $a \in [n]^\#s$: if $s \in \text{Fun}(\sigma)$, the machine guesses the image of $s(a)$ and if $s \in \text{Rel}(\sigma)$ then the machine guesses the bit denoting whether $a \in s$. In total, the machine makes at most $mn^d \in O(n^d)$ guesses which can be implemented with $O(n^d)$ instructions.

Now, the second step of the algorithm is completely deterministic. Since $\phi$ and thus also $\psi$ are fixed, evaluating $\psi$ with a single assignment $y \in [n]^d$ of the $d$ variables can be done in constant-time: the length of $\psi$ is independent of $n$, and each atomic subformula or subterm of $\psi$ can be evaluated by checking either the read-only input registers when seeking the value of $s \in \tau$, or the main memory in case $s \in \sigma$. In total, the second step requires $n^d$ constant-time operations. Therefore, the running time of the whole algorithm is bounded by $O(n^d)$.

For the other direction of the proof, we only consider the case $d = 1$. As Grandjean [12] points out, the proof generalizes with some work. However,
we will first prove two technical lemmas needed in the proof of the main result.

**Lemma 4.7.** Let $\tau$ be a vocabulary. For any formula $\varphi \in \text{FO}^\tau$ there exists a logically equivalent formula $\varphi^* \in \text{FO}^\tau$ where all occurrences of any $f \in \text{Fun}(\tau)$ appear only in the atomic subformulas of the form $f(x_0, \ldots, x_k) = y$ and $\varphi^*$ has the same number of universal quantifications as $\varphi$.

*Proof.* Let $t$ be a term. We define the formula $\psi_t$ with a free variable $y$ as follows. If $t = f$ where $f$ is a first-order variable or a nullary function symbol in $\tau$, then $\psi_t \equiv y = f$. Otherwise, if $t = f(t_0, \ldots, t_k)$ where $f \in \text{Fun}(\tau)$ then define

$$\psi_t \equiv \exists x \exists y_0 \cdots \exists y_k [f(y_0, \ldots, y_k) = y \land \bigwedge_{i \in [k+1]} \psi_{t_i}(y/y_i)].$$

Let $\varphi$ be a formula in $\text{FO}^\tau$. We define the formula $\varphi^*$ inductively. We only need to consider the base case as the inductive step is trivial. If $\varphi \equiv t_0 = t_1$, then

$$\varphi^* \mapsto \exists y_0 \exists y_1 [\varphi_{t_0}(y/y_0) \land \varphi_{t_1}(y/y_1) \land y_0 = y_1].$$

If $\varphi \equiv R(t_0, \ldots, t_k)$ where $R \in \text{Rel}(\tau)$ then

$$\varphi^* \mapsto \exists y_0 \cdots \exists y_k \left( \bigwedge_{i \in [k+1]} \varphi_{t_i}(y/y_i) \land R(y_0, \ldots, y_k) \right).$$

It is easy to see that the inductive cases for connectives and quantifiers does not alter the formula.

The following lemma states that it is possible to simulate an ordered structure of domain $[cn]$ with an ordered structure of smaller domain $[n]$. Essentially, each element $x \in [cn]$ can be encoded as an element $(y, z) \in [c] \times [n]$.

**Lemma 4.8.** Fix $c \in \mathbb{N}_+$ and let $\tau$ and $\sigma$ be disjoint finite vocabularies where $\#\sigma = \#\tau = 1$ and $\sigma = \text{Fun}(\sigma)$. Let $\tau' = \tau \cup \{<, S, 0\}$ where $<$ is a binary relation symbol interpreted as a linear order, $S$ interpreted as a successor function and $0$ as the first element. Let $n > c$ and $A \in \text{Str}(\tau')$ be an ordered structure and $A' \in \text{Str}(\tau' \cup \sigma)$ be an extension of $A$ such that

1. $\text{Dom}(A) = [n]$ and $\text{Dom}(A') = [cn]$,
2. $<^A \subseteq <^{A'}$ and $R^A = R^{A'}$ for all $R \in \text{Rel}(\tau)$,
3. $f^{A'}|[n] = f^A$ for all $f \in \text{Fun}(\tau)$ and $f^{A'}(i) = 0$ for all $i \in [cn] \setminus [n]$.
For any formula $\varphi \in \text{FO}^{\tau \cup \sigma}$, there exists a vocabulary $\sigma^*$, a structure $A^* \in \text{Str}(\tau' \cup \sigma^*)$ where $\text{Dom}(A^*) = [n]$, and a formula $\varphi^* \in \text{FO}^{\tau' \cup \sigma^*}$ such that for all assignments $s$

$$A^' \models_s \varphi \iff A^* \models_{s^*} \varphi^*,$$

where $s^*$ is defined below and the formula $\varphi^*$ contains the same number of quantifications as $\varphi$. In addition, $A|\tau = A^*|\tau$.

Proof. Let $A$, $A'$ and $\varphi$ be as in the statement. We can assume that each quantified variable is unique and the variables appearing in $\varphi$ are from the set $\{x_0, \ldots, x_m\}$. Furthermore, by Lemma 4.7, we can assume that the functions appear only in the atomic subformulas of the form $f(x_i) = x_j$.

We now define a new vocabulary $\sigma^*$ by modifying the vocabulary $\sigma$ as follows. Define

$$\sigma^* = \{f_0, f_1 : f \in \sigma\} \cup \{h : h \in [c]\}.$$ 

The first set consists of nullary and unary function symbols and the second set consists of new constant symbols.

Since every $x \in [cn]$ can be written as $x = y(n - 1) + z$ where $y \in [c]$ and $z \in [n]$, we can construct the structure $A^*$ such that for every function symbol $f \in \sigma$ the equivalence

$$f^A'(y(n - 1) + z) = y'(n - 1) + z' \iff f_0^A^*(y) = y' \text{ and } f_1^A^*(z) = z'$$

holds. That is, every $x \in [cn]$ is represented as an element $(y, z) \in [c] \times [n]$.

For every $R \in \text{Rel}(\tau)$, the interpretations of relations remain as in $A'$, that is, $R^{A^*} = R^{A'} = R^A$. For function symbols, we remove the trivial points such that for each $f \in \text{Fun}(\tau)$ we have $f^{A^*} = f^{A'}|n = f^A$. That is, $A^*|\tau = A|\tau$.

Finally, define the interpretations for the constant symbols as $h^{A^*} = S^h(0)$ for each $h \in [c]$.

We now define the formula $\varphi^*$ inductively. For the base case, we consider the atomic subformulas of $\varphi$. For the basic equality, we have

$$(x_i = x_j)^* \mapsto y_i = y_j \land z_i = z_j.$$ 

For functions we will define two transformations depending on whether the symbol is in $\tau$ or $\sigma$. If $f \in \sigma$, we define the transformation as

$$(f(x_i) = x_j)^* \mapsto f_0(y_i) = y_j \land f_1(z_i) = z_j$$

and in case $f \in \tau$, we need to ensure that the trivial points of function $f^{A'}$ are handled correctly:

$$(f(x_i) = x_j)^* \mapsto \left( y_i = 0 \mapsto [f_0(y_i) = y_j \land f_1(z_i) = z_j] \land y_i \neq 0 \mapsto [y_j = 0 \land z_j = 0] \right).$$

23
Similarly, for relation symbols $R \in \tau$ we have

$$(R(x_i))^* \mapsto y_i = 0 \land R(z_i)$$

since $R^{A'} = R^A \subseteq [n]$. The linear order is replaced with a lexicographic order. Thus, for subformulas using the symbol $<$ we have

$$(x_i < x_j)^* \mapsto y_i < y_j \lor (y_i = y_j \land z_i < z_j).$$

The rest of the transformation is straightforward:

$$(\neg \psi)^* \mapsto \neg \psi^*$$

$$(\psi \land \theta)^* \mapsto \psi^* \land \theta^*$$

$$(\exists x_i \psi)^* \mapsto \bigvee_{h \in [c]} \exists z_i \psi^*(y_i/h)$$

$$(\forall x_i \psi)^* \mapsto \bigwedge_{h \in [c]} \forall z_i \psi^*(y_i/h).$$

It remains to show that $A' \models_s \varphi \iff A^* \models_s \varphi^*$. We prove this by induction over $\varphi$. We begin with the atomic subformulas. Since $s(x_i) = h(n - 1) + k$ define $s^*(z_i) = h$ and $s^*(y_i) = k$. For the first case, we see that

$$A' \models_s x_i = x_j \iff s(x_i) = s(x_j)$$

$$\iff s^*(y_i) = s^*(y_j) \text{ and } s^*(z_i) = s^*(z_j)$$

$$\iff A^* \models^* _s (x_i = x_j)^*.$$ 

For the case $f(x_i) = x_j$, first suppose $f \in \sigma$. Then

$$A' \models_s f(x_i) = x_j \iff f^A'(s(x_i)) = s(x_j)$$

$$\iff f_0^{A'}(s^*(y_i)) = s^*(y_j) \text{ and } f_1^{A'}(s^*(z_i)) = s^*(z_j)$$

$$\iff A^* \models^*_s (f(x_i) = x_j)^*.$$ 

If $f \in \tau$, we get

$$A' \models_s f(x_i) = x_j \iff \text{If } s(x_i) < n, \text{ then } f^A'(s(x_i)) = s(x_j).$$

Otherwise, $s(x_j) = 0$. 

$$\iff \text{If } s^*(y_i) = 0, \text{ then }$$

$$f_0^{A'}(s^*(y_i)) = s^*(y_j) \text{ and } f_1^{A'}(s^*(z_i)) = s^*(z_j).$$

Otherwise, $s^*(y_j) = s^*(z_j) = 0$. 

$$\iff A^* \models^*_s (f(x_i) = x_j)^*.$$ 

For the last case,

$$A' \models_s R(x_i) \iff s(x_i) \in R^{A'} = R^A = R^{A^*}$$

$$\iff s^*(y_i) = 0 \text{ and } s^*(z_i) = s^*(x_i) \in R^{A^*}$$

$$\iff A^* \models^*_s (R(x_i))^*.$$ 

24
Suppose the induction hypothesis holds for $\psi$: $A' \models_s \psi \iff A^* \models_{s^*} \psi^*$. For negation we have

$$A' \models_s \neg \psi \iff A' \not\models_s \psi \iff A^* \not\models_{s^*} \psi^* \iff A^* \models_{s^*} \neg \psi.$$  

Similarly for conjunction, suppose the induction hypothesis holds for both $\psi$ and $\psi'$. Then,

$$A' \models_s \psi \land \psi' \iff A' \models_s \psi \land A' \models_s \psi' \iff A^* \models_{s^*} \psi^* \land A^* \models_{s^*} \psi'^* \iff A^* \models_{s^*} (\psi \land \psi')^*.$$  

For existential quantification, recall that $h^{-A^*} = S^h(0)$ for each $h \in [c]$. Now,

$$A' \models_s \exists x_i \psi \iff A' \models_{s(x_i/a)} \psi \text{ for some } a \in [cn]$$

$$\iff A^* \models_{s^*} (y_i.h.x_i/k) \psi^* \text{ for some } h \in [c], k \in [n]$$

$$\iff A^* \models_{s^*} \bigvee_{h \in [c]} \exists z_i \psi^* (y_i/h).$$

Finally, the case for universal quantification is similar:

$$A' \models_s \forall x_i \psi \iff A' \models_{s(x_i/a)} \psi \text{ for all } a \in [cn]$$

$$\iff A^* \models_{s^*} (y_i.h.x_i/k) \psi^* \text{ for all } h \in [c], k \in [n]$$

$$\iff A^* \models_{s^*} \bigwedge_{h \in [c]} \forall z_i \psi^* (y_i/h).$$

\[\square\]

**Lemma 4.9.** Let $\tau$ be a vocabulary. Then

$$\text{NTIME}^\tau[n] \subseteq \text{ESO}^\tau[\#1, \forall 1]$$

where $n$ is the cardinality of the input structure.

**Proof.** Let $\tau$ be a vocabulary and $P$ be a $\tau$-problem recognized by a $\tau$-N Rams $M$ in at most $cn$ steps where $c \in \mathbb{N}_{+}$ and $n$ is the cardinality of the input structure. It can be assumed without loss of generality, that $M$ only uses integers less than $cn$: the integers generated by the guess instruction are bounded by $O(n)$, the NRAM $M$ can only increase the value in accumulator $A$ at most $cn$ times, and since the domain of the input structure is $[n]$, all functions of the input structure have a range of $[n]$.

Moreover, the computation has exactly $cn - 1$ steps and the final instruction at step $cn - 1$ is either the accept or reject instruction. In case the accept or reject instructions are executed earlier, the halted computation is simulated by repeating the instruction until time step $cn - 1$. Suppose the program associated to $M$ is $\langle I_0, \ldots, I_k \rangle$ where $k \in [cn]$. It suffices to assume that $I_k$ is the only accept instruction in the program; if not, simply replace all other accept instructions with suitable jump instructions.
We will define a ESO$^\tau$-sentence $\Phi_M$ which any $\tau$-structure satisfies if and only if $M$ accepts the structure. We begin by first constructing the first-order part of $\Phi_M$.

Let $A$ be an input $\tau$-structure. First, we characterize the computation over the domain $[cn]$ and later shrink the domain to $[n]$ using Lemma 4.8. Suppose that the interpretations for symbols in $\tau$ are extended to domain $[cn]$; relations remain unchanged while function symbols are trivially extended to map new values to 0. We call this extended structure as $A'$ and define our formulas to match the larger domain $[cn]$.

We will construct a new structure of vocabulary $\{<, S, 0\} \cup \sigma \cup \rho$ where $\sigma$ consists of unary function symbols

$$\sigma = \{I, A, R_A, R'_A, G, L_1, L_2, B_j: 0 \leq j \leq \#\tau\}$$

and the constant (nullary function) symbols $\rho = \{k, m, n\}$. These symbols are used to simulate the computation of the NRAM $M$ and will be quantified in the second-order part of formula $\Phi_M$.

We will describe the interpretations with respect to the canonical order of natural numbers. For the constant symbols $k$, $m$, and $n$, we will have $k = k$, $m = cn - 1$, and $n = n - 1$. For any integer $i \in \mathbb{N}$ we write $\hat{i}$ to denote the term $S^i(0)$ where $S$ is interpreted as the successor function associated to the linear order $<.

The interpretations of function symbols in $\sigma$ are used to describe the state of $M$ at the beginning of time step $t$, that is, right before the $t$th instruction is executed. We will construct formulas to enforce the following properties for the interpretations of symbols $\sigma$ for all $t \in [cn]$:

- $I(t)$ stores the instruction number to be executed at time step $t$. In particular, $I(0) = 0$ and $I(m) = k$. That is, the first instruction will be $I_0$ and the last $I_k$.
- $A(t)$ and $B_j(t)$ denote the values contained in accumulators $A$ and $B_j$ at time step $t$.
- $R_A(t)$ is the value contained in register $R_i$ where $i$ is the integer stored in accumulator $A$ at time step $t$.
- $R'_A(t)$ is the value contained in register $R_i$ after step $t$.
- $G(t)$ is a non-deterministic guess available to the NRAM at time step $t$.
- $L_1$ and $L_2$ are auxiliary functions used in encoding the semantics of $R_A$.

**Simulating the sequence of instructions.** We will begin by showing how to encode the function $I$ which describes the sequence of executed instructions, that is, the flow of computation. We need to consider three
types of instructions: (i) the instructions that do not alter the flow of the
computation, (ii) the jump instructions, and (iii) the terminating instructions
which are accept and reject.

In case an instruction of the first type is encountered, the next instruction
will simply be the instruction with the successive index. For jump instructions,
we must first check whether a condition of the form $A = B$ holds and then
decide the index of the next instruction. If a terminating instruction is
encountered, the computation remains at the terminating instruction until
the final time step.

We begin by encoding the behaviour of jump instructions. Let $I_j \subseteq \{1, \ldots, 11\}$
be the set of indices for each type of instruction $j \in \{1, \ldots, 11\}$ in the program
of $M$. For each $j$ let us construct the formula

$$\alpha_j(t) \equiv \bigvee_{a \in I_j} I(t) = a$$

which expresses whether instruction at index $I(t)$ is of type $j$. In addition,
define the set $J \subseteq |k| \times \#\tau + 1$ such that $(y, a, b, r) \in J$ if instruction $I_y$
is a jump instruction of the form “if $A = B$ then jump to $I_a$ else jump to $I_b$”.

Now, define the formula

$$\theta_{r,a,b}(t) \equiv (A(t) = B_r(t) \rightarrow I(S(t)) = a)$$

$$\wedge (A(t) \neq B_r(t) \rightarrow I(S(t)) = b)$$

and the formula

$$\Theta_1 \equiv \bigwedge_{(y,a,b,r) \in J} (I(t) = y \rightarrow \theta_{r,a,b}).$$

The formula $\Theta_1$ encodes the jump instruction behaviour. To encode the
instructions that do not alter the flow of computation and are non-terminating,
let

$$\Theta_2 \equiv \bigwedge_{1 \leq j \leq 7} \alpha_j(t) \rightarrow (I(S(t)) = S(I(t))).$$

That is, after executing the instruction at index $I(t) = i$ move to the
successive index $i + 1$.

To handle the case of a terminating instruction, i.e., either an accept or
reject instruction, we use the formula

$$\Theta_3 \equiv (\alpha_{10}(t) \lor \alpha_{11}(t)) \rightarrow I(S(t)) = I(t).$$

The correct sequence for the executed instructions is enforced by the formula

$$\Psi_1 \equiv \bigwedge_{1 \leq i \leq 3} \Theta_i \wedge I(0) = 0 \wedge I(m) = k.$$
Simulating the state. We now show to encode the semantics for the instructions that alter the contents of the registers. There are eight different instructions to consider.

1. \( A \leftarrow N \): This instruction simply sets the contents of accumulator \( A \) to be the cardinality of the input structure. Recall that the arithmetical structure contains the symbol \( b \) that indicates the final element in the ordering. Thus we have
   \[
   \eta_1 \equiv \alpha_1(t) \land A(S(t)) = n.
   \]

2. \( A \leftarrow s(B_0, \ldots, B_q) \) where \( s \in \tau \) and \( q = \#s \): We need to keep track which instructions refer to which symbols. Let \( K \subseteq [k] \times \tau \) where \( (i, s) \in K \) if instruction \( I_i \) is of the form \( A \leftarrow s(B_0, \ldots, B_{\#s}) \). The formula
   \[
   \eta_2 \equiv \alpha_2(t) \land \bigvee_{(i, s) \in K} \left( I_i(t) \land A(S(t)) = s(B_0(t), \ldots, B_{\#s}(t)) \right)
   \]
   then ensures that the simulated contents of register \( A \) correspond to the contents of the correct input register.

3. \( A \leftarrow 0 \): For the initialization instruction, we define the formula
   \[
   \eta_3 \equiv \alpha_3(t) \land A(S(t)) = 0
   \]
   which states that accumulator \( A \) contains the value zero after this instruction.

4. \( A \leftarrow A + 1 \): This instruction is handled similarly as the one before:
   \[
   \eta_4 \equiv \alpha_4(t) \land A(S(t)) = A(t).
   \]

5. \( A \leftarrow R_A \): Again, this formula is similar to the two previous formulas:
   \[
   \eta_5 \equiv \alpha_5(t) \land A(S(t)) = R_A(t).
   \]

6. \( B_i \leftarrow A \): Let \( K' \subseteq [k] \times [\#\tau + 1] \) be the set such that \( (j, r) \in K' \) iff the instruction \( I_j \) is of the form \( B_r \leftarrow A \). These instructions are simulated with the formula
   \[
   \eta_6 \equiv \alpha_6(t) \land \bigvee_{(j, r) \in K'} \left( I(t) = j \land B_r(S(t)) = A(t) \right)
   \]
   and the following formula ensures that the contents of these registers remain unchanged unless instructions of type 6 are executed
   \[
   \eta_6' \equiv \bigwedge_{r \in [\#\tau + 1]} \left[ \left( \bigwedge_{(j, r) \in K'} I(t) \neq j \right) \rightarrow B_r(S(t)) = B_r(t) \right].
   \]
7. $R_A \leftarrow B_i$: Let $K'' \subseteq [k] \times [\#\tau + 1]$ be the set such that $(j, r) \in K''$ iff the instruction $I_j$ is of the form $R_A \leftarrow B_r$. Now, let \( \eta_7 \equiv \psi_1 \land \psi_2 \) where

\[
\psi_1 \equiv \alpha_7(t) \rightarrow \bigvee_{(j, r) \in K''} \left( I(t) = j \land R_A(S(t)) = B_r(t) \land R'_A(t) = B_r(t) \right)
\]

simulates the altered state of $R_A$ and the last conjunct satisfies the condition that the value of $R'_A(t)$ is the value of register $R_A$ at the end of step $t$. Finally, we need to ensure that the simulated contents of register $R_A$ do not change arbitrarily, i.e., only instructions of type 7 may alter the contents of $R_A$:

\[
\psi_2 \equiv \neg \alpha_7(t) \rightarrow \left( R_A(S(t)) = R_A(t) \land R'_A(t) = R_A(t) \right).
\]

8. guess($A$): The formula simulating the non-deterministic guess instruction is defined using the function $G$ as

\[
\eta_8 \equiv \alpha_8(t) \land A(S(t)) = G(t).
\]

Like with registers $R_A$ and $B_r$, we also need to ensure that the contents of accumulator $A$ remain the same unless its contents are explicitly altered. To do this we use the formula

\[
\eta_9 \equiv \bigvee_{i \in X} \neg \alpha_i(t) \rightarrow \left( A(S(t)) = A(t) \right)
\]

where $X = \{1, \ldots, 5\} \cup \{8\}$ are the instructions that may alter the contents of $A$.

It remains to finalize the definition of function $R_A$. So far we have only defined how the assignments from and to $R_A$ behave, but not what are the actual contents of $R_A$. While $R_A(t)$ maps to the value of the register pointed by accumulator $A$ at the start of step $t$, the value $R'_A(t)$ matches the contents of register pointed by $A$ at the end of step $t$. Thus, we define $R_A$ using $R'_A$.

We need to consider two cases. First, observe that if the register addressed by $A$ has not been accessed before time step $t$, then the register contains its initial value, i.e., $R_A(t) = 0$. Second, suppose $u < t$ is the last time step before $t$ such that the accumulator $A$ holds the value $A(t)$. Then at the start of step $t$, the register pointed by $A$ contains the same value as at the end of step $u$. Thus, in this case, $R_A(t) = R'_A(u)$.

Equivalently, we can consider the lexicographic order of the pairs of form $(A(t), t)$. Suppose $(A(t), t)$ is the immediate successor of pair $(A(u), u)$. If $A(t) = A(u)$, then this matches the second case, and we have $R_A(t) = R'_A(u)$. Otherwise, $A(u) < A(t)$ and $u < t$. In particular, $A(t) \neq A(u)$ so the accumulator $A$ has not been given the value $A(t)$ at any step before $t$. This
means that time step $t$ is the first step when $A$ contains the value $A(t)$ and that the register addressed by $A(t)$ has not been accessed before. Thus, $R_A(t) = 0$.

We will encode the above behaviour by first defining a lexicographic order over the $cn$ pairs of the form $(A(t), t)$. This is done in a similar fashion as in Section 4.1 where we defined a lexicographic order in the arithmetical structure. We will define a function $L: [cn] \to [cn]^2$ using the two unary functions $L_1$ and $L_2$ such that $L(x) = (L_2(x), L_1(x))$. Recall that the variable $t$ will be universally quantified. Define the two formulas

$$\theta_1 \equiv \exists x[(L_2(x), L_1(x)) = (A(t), t)]$$

and

$$\theta_2 \equiv (t \neq m) \rightarrow \left( L_2(t) < L_2(S(t)) \lor L_1(t) = L_1(S(t)) \right)$$

where $m$ corresponds to the value $cn$. When $t$ is universally quantified, then $\theta_1$ ensures that the set $\{(A(t), t) : t \in [cn]\}$ is the image of $L$. The second formula defines the lexicographic order using the linear order of the domain $[cn]$.

We can now finish encoding the semantics for manipulating register $R_A$ with the formula

$$\theta_3 \equiv (t \neq 0) \rightarrow \exists u \exists x \left( \begin{array}{l} L(x) = (A(u), u) \land \\ L(S(x)) = (A(t), t) \land \\ A(t) = A(u) \rightarrow R_A(t) = R'_A(u) \land \\ A(u) < A(t) \rightarrow R_A(t) = 0 \end{array} \right)$$

where $L(x)$ denotes $(L_2(x), L_1(x))$. We can now define

$$\eta_{10} \equiv \bigwedge_{1 \leq i \leq 3} \theta_i \land R_A(0) = 0$$

which captures the semantics of manipulating register $R_A$. Finally, we can take the conjunction of all the $\eta_i$ formulas and define the formula

$$\Psi_2 \equiv \bigwedge_{1 \leq i \leq 10} \eta_i$$

which essentially simulates the effects of the instructions and the different states of the registers in the given NRAM $M$.

**Reducing the size of the domain.** In the above construction, we built a formula assuming that the cardinality of the structure matched the running time $T(n) = cn$ instead of the input size $n$. We will now utilize Lemma 4.8 and construct a formula that simulates the machine $M$ in the original domain $[n]$. 

30
Let $\Phi \equiv \exists m \exists k \forall t (\Psi_1 \land \Psi_2 \land \Psi_3)$ where $\Psi_3 \equiv k = S^k(0) \land t \neq m \rightarrow t < m$ enforces the correct values for the constants $k$ and $m$. Next, suppose $A$ is an input $\tau$-structure with $\text{Dom}(A) = [n]$ and $A'$ the extended structure with domain $[cn]$.

The non-deterministic machine $M$ accepts the structure $A$ if and only if there exists an accepting computation, that is, there exist interpretations for the symbols in $\sigma$ together with a linear order $<$ with successor function $S$ and the zero element $0$ such that $\langle A', <, S, 0, \sigma \rangle \models \Phi$ where $n$ is interpreted as $S^n(0)$.

By applying Lemma 4.8 we get an ordered structure of domain $[n]$ and formula $\Phi^*$ such that $\langle A', <, S, 0, n, \sigma \rangle \models \Phi^*$ holds if and only if there are also interpretations for the symbols in the set $\sigma^* \cup \{<, S, 0, n, 1\}$ such that $\langle A^*, <, S, 0, n_0, n_1, \sigma^* \rangle \models \Phi^*$.

Observe that after transforming the nullary function symbol $n$ as in Lemma 4.8, we get two new nullary symbols $n_0$ and $n_1$. Since the symbol $n$ should be interpreted as the size of the domain, we define the new symbols so that they encode the pair $(0, n-1)$ with the formula $\Psi_n \equiv n_0 = 0 \land \forall x(x \neq n_1 \rightarrow x < n_1)$.

Thus, the machine $M$ accepts the structure $A$ if and only if there exist interpretations for symbols in $\sigma^* \cup \{<, S, 0, n_0, n_1\}$ such that

$$\langle A^*, <, S, 0, n_0, n_1, \sigma^* \rangle \models \Phi^* \land \Psi_n \iff \langle A^*, <, S, 0 \rangle \models \exists \sigma^* \exists n_0 \exists n_1 (\Phi^* \land \Psi_n) \iff A^* \models \exists_{\text{ord}} (\langle S, 0 \rangle \exists \sigma^* \exists n_0 \exists n_1 (\Phi^* \land \Psi_n)).$$

Let us denote the last sentence as $\Phi_M$. Since $A^* = A$, we have $A \models \Phi_M$ if and only if $M$ accepts $A$. It remains to show that the sentence $\Phi_M$ can be expressed in the logic ESO$^\tau[\#1, \forall 1]$.

First, observe that only the symbols in $\tau$ may have arity higher than one in the formula $\Phi$. Applying the transformation given in Lemma 4.8 does not introduce new universal quantifications or new symbols of arity higher than one. Therefore, also the symbols in $\{n_0, n_1\} \cup \sigma^*$ have arity at most one.

Second, since both formulas in the conjunction $\Phi^* \land \Psi_n$ contain only a single universal quantification, the formula $\Phi^* \land \Psi_n$ can be rewritten using only a single universal quantification. Finally by Theorem 2, quantifying a linear order and a successor function can be done using only one universal quantification and unary second-order existential quantifications. Thus, the sentence $\Phi_M$ is definable in ESO$^\tau[\#1, \forall 1]$. □
In conclusion, Lemma 4.6 and Lemma 4.9 together provide the following theorem [12]:

**Theorem 3.** \(\text{NTIME}^*[n] \equiv \text{ESO}^*[\#1, \forall 1]\).

## 5 Classifying graph problems

This section studies the complexity of various decision problems on graphs. First, we show that many graph properties are expressible in the logic \(\text{ESO}[\#1, \forall 1]\). As a consequence of the logical characterization given in Section 4, these problems are recognizable by non-deterministic random-access machines that run in linear time in the number of vertices. We will call this complexity class \(\text{NVLIN}\). In some sense, this is a complexity class that utilizes “only small amounts” of non-determinism.

Second, we will show that while many NP-complete problems are in \(\text{NVLIN}\), some problems of low complexity reside outside this class. The lower bound arguments essentially state that for some problems, any NRAM recognizing them must read \(\Omega(n^2)\) bits of the input. Conversely, this gives us non-expressibility results for the logic \(\text{ESO}[\#1, \forall 1]\) due to Theorem 3. In contrast, observe that many classical graph properties, such as independent sets, cliques, vertex covers, \(k\)-colourability, and so on, are easily definable in the logic \(\text{ESO}[\#1, \forall 2]\) which allows two universal first-order quantifications.

### 5.1 Problems recognizable in vertex-linear time

In this section, we will exhibit graph properties expressible in the logic \(\text{ESO}[\#1, \forall 1]\). The first three results are due to Grandjean [9].

Recall, that a Hamiltonian path is a path that contains each vertex of the graph exactly once. Similarly, a Hamiltonian circuit is a cycle that contains each vertex exactly once. Checking whether a graph has either a Hamiltonian path or a cycle is a classic NP-complete problem [7].

**Proposition 2.** Deciding the existence of a Hamiltonian circuit is in \(\text{NVLIN}\).

*Proof.* Let \(\mathcal{G} = \langle [n], E \rangle\) be a graph and consider the formula

\[
\theta \equiv \exists_{\text{ord}} (s, 0) \forall x E(x, S(x))
\]

which is clearly in \(\text{ESO}[\#1, \forall 1]\). To see that it characterizes the existence of a Hamiltonian circuit, suppose \(\mathcal{G} \models \theta\). Then the function \(S\) is a permutation of the set of vertices of the graph \(\mathcal{G}\) such that successive vertices are connected by an edge. In particular, by the proof of Theorem 2 the successor function \(S\) is cyclic and we have \(E(S^{n-1}(0), 0)\). \(\square\)

**Proposition 3.** Deciding the existence of a Hamiltonian path is in \(\text{NVLIN}\).
Proof. The formula defining the existence of a Hamiltonian path is almost identical to the one presented above. However, a Hamiltonian path does not require that the first and final vertex are connected. It is easy to see that the formula
\[
\theta \equiv \exists \text{ord}(\langle <, S, 0 \rangle \forall x \left( x \neq b \rightarrow E(x, S(x)) \right))
\]
is in ESO[#1, ∀1] and is true iff the graph has a Hamiltonian path. □

We will now prove the following lemma which states that spanning trees are definable in the logic ESO[#1, ∀1]. The basic idea is that we can quantify a so-called tree-order over the vertices [2, Ch. 1.5].

Lemma 5.1. Let \( G \) be a graph, \( P \) be a unary function symbol and \( r \) a constant (nullary function) symbol. There exists a ESO[#1, ∀1]-formula \( \text{spans}(P, r) \) such that if
\[
\langle G, E, P, r \rangle \models \text{spans}(P, r),
\]
then \( P \) defines a predecessor relation over the domain such that \( r \) has no predecessor. In other words, \( P \) defines a rooted tree with \( r \) as the root.

Proof. We want to ensure that \( r \) denotes the root of a tree and \( P \) defines the predecessor function in the tree. That is, \( P(x) \) is the parent of \( x \). Let us define the formula
\[
\theta(P, r) \equiv \exists \text{ord} < \forall x \left[ x \neq r \rightarrow \left( E(P(x), x) \land P(x) < x \right) \right].
\]
Suppose \( \langle G, E, P, r \rangle \models \theta(P, r) \). The first conjunct in the implication states that all nodes except for the root node \( r \) have an edge to the parent vertex given by \( P(x) \). The second conjunct forces the linear order over the vertices to be such that the for all \( x \), the children \( x \) succeed \( x \) in the linear order. That is, for all \( x \) and \( y \in \{ v \in G \colon x = P(v) \} \) it holds that \( x < y \). Thus, if the formula holds, \( P \) defines a rooted directed tree in the graph. See Figure 2 for illustration.

The first-order part of the above formula uses only one universal quantification, and by Theorem 2 we know that a linear order is definable in ESO[#1, ∀1]. Therefore, there exists the desired formula \( \text{spans}(P, r) \). □

We will now consider the problem of checking whether a graph is connected. A graph \( G = \langle G, E \rangle \) is connected if for all \( u, v \in G \) there exists a path from \( u \) to \( v \). It is well-known that deciding whether a graph is connected is not expressible in first-order logic [15, Ch. 3.6]. On the other hand, Reingold has shown that connectivity is recognizable in deterministic logarithmic space [18]. The following proposition shows that the connectivity problem is recognizable by linear-time NRAMs.

Proposition 4. Deciding whether a graph is connected is in NVLIN.
Figure 2: Defining a spanning tree over a graph. (a) The input graph $G = (G,E)$. (b) A guess for $r$ and $P$. The black node denotes the root node $r$. For each node $u$, the highlighted arrow leading to node $u$ begins at node $P(u)$. (c) The resulting spanning tree.

Proof. Since a graph $G$ is connected if and only if it has a spanning tree, by Lemma 5.1 the ESO$^{(E)}[\#1, \forall 1]$-sentence

$$\exists P\exists r[\text{spans}(P,r)]$$

expresses whether the graph has a spanning tree. \qed

We now give logical characterizations of three other classic NP-complete problems: the dominating set, isomorphic spanning tree, and set cover problems. Most of the classical NP-complete problems have already been shown to be recognizable with linear-time (in the size of the input) NRAMs [10]. However, we present logical characterizations for these three problems that so far do not seem to appear in the literature.

Before proceeding to the actual problems, let us define two simple but helpful subformulas. Since the formulas are quite obvious, we omit their proofs.

Lemma 5.2. Suppose $M$ is finite.

1. Let $f$ be a unary function symbol and

$$\text{bijective}(f) \equiv \forall x \exists y [x = f(y)].$$

Now, $(M,f)\models \text{bijective}(f)$ if the interpretation of $f$ is a bijection.

2. Let $X$ and $K$ be unary predicates and

$$\text{size}(X,K) \equiv \exists h \forall x [\text{bijective}(h) \land X(x) \rightarrow K(h(x))].$$

Now, $(M,X,K)\models \text{size}(X,K)$ iff $|X| \leq |K|$.

34
We will first consider the dominating set problem. A dominating set of a graph $G = (G, E)$ is a subset $D \subseteq G$ such that for each vertex $v \in G$ either $v \in D$ or there exists $(v, u) \in E$ such that $u \in D$. To encode our input, we assume that we are given an additional unary relation $K$ such that $|K| = k$ where $k \in \mathbb{N}_+$. The problem is NP-complete [7].

**Proposition 5.** Given a graph $G$ and $k \in \mathbb{N}_+$ as input, deciding whether a graph has a dominating set of size at most $k$ is in NL.

**Proof.** We will use a single second-order quantification of a unary relation symbol $D$ and apply Lemma 5.2. The formula

$$\exists D \left[ \text{size}(D, K) \land \forall x \left( D(x) \lor \exists y \left( D(y) \land E(x, y) \right) \right) \right]$$

expresses the existence of dominating sets of size $k$. The first conjunct simply ensures that for the guessed set $D$ has at most $k$ elements. The second conjunct is the definition of a dominating set. Since the formula is a conjunction of formulas each of which uses only one universal quantification, we can write the formula using a single universal quantification. By Lemma 2.1 the formula is in ESO[$\#1, \forall 1$] after Skolemization.

We now turn to the isomorphic spanning tree problem which is defined as follows. Given graph $G$ and a tree $T$ as input, decide if $G$ has a spanning tree isomorphic to $T$. The problem is NP-hard as the Hamiltonian path problem reduces to it by letting $T$ be a path of length $n - 1$.

It suffices to consider only the cases where $G$ and $T$ have the same number of vertices. Moreover, without loss of generality, we can assume that $\text{Dom}(G) = \text{Dom}(T)$. Therefore, we can encode our input as a model $\langle G, E, T \rangle$ where $E$ is the edge relation of $G$ and $T$ is the edge relation of $T$.

**Proposition 6.** Given a graph $G$ and a tree $T$ as input, deciding whether $G$ has a spanning tree isomorphic to $T$ is in NL.

**Proof.** Suppose $\langle G, E, T \rangle$ is the input structure. The formula

$$\theta \equiv \exists P \exists r \exists h \left( \text{spans}(P, r) \land \text{bijective}(h) \land \forall x \left[ E(P(x), x) \leftrightarrow T(h(P(x)), h(x)) \right] \right)$$

expresses whether there exists a bijection that maps $T$ into the edge set of some spanning tree of $G$. By Lemmas 5.1 and 5.2, the first two conjuncts simply ensure that $P$ defines a spanning tree and that $h$ is a bijection on the vertices of the graph. The last conjunct states that $h$ is an isomorphism between the tree defined by $P$ and $T$. Again, as the formula consists of conjunction of formulas using only a single universal quantification, the formula can be written using only one universal quantification and therefore is in ESO[$\#1, \forall 1$].

35
Finally, we look at a slightly different problem known as the *set cover problem*: Let $U$ be a finite set called the universe. Given a collection $C = \{S_0, \ldots, S_m\} \subseteq \mathcal{P}(U)$ and a parameter $k \in \mathbb{N}_+$, decide whether there exists a subset $C' \subseteq C$ such that $|C'| \leq k$ and $\bigcup_{S \in C'} S = U$.

The set cover problem is NP-complete and it is a generalization of the vertex cover problem. In Section 5.2, we will see that the vertex cover problem is not solvable in vertex-linear time. However, the following formulation of the set cover problem as a graph problem can be recognized in vertex-linear time. This does not however imply that set cover is solvable in linear time in the cardinality of the universe.

We encode the set cover instance as a model $S = \langle U \cup C, K, V, E \rangle$ such that $|K| = k$, $V = U$ and $E$ defines the inclusion relation

\[ \{(S, e): S \in C, e \in S\} \]

Observe that the set cover problem can be regarded as a bipartite graph problem where the task is to dominate all vertices in the set $U$ with vertices in the set $C$.

**Proposition 7.** *Set cover is decidable in* $\text{ESO}[\#1, \forall 1]$.

**Proof.** The following formula states that $X \subseteq C$ and that the set $X$ covers all elements of the set $U$:

\[ \theta \equiv \forall x \left[ (X(x) \rightarrow \neg V(x)) \land (V(x) \rightarrow \exists y (X(y) \land E(x, y))) \right] . \]

Now, we can simply quantify the relation $X$ and ensure that the size of $X$ is at most $k$. Thus, the existence of a cover of size $k$ is captured by the sentence

\[ \exists X [\text{size}(X, K) \land \theta] . \]

It is easy to see that after Skolemization the sentence is in $\text{ESO}[\#1, \forall 1]$.

5.2 Problems outside the class VNLIN

As we saw in the previous section, the complexity class NVLIN contains problems considered to be computationally hard. However, there are also “easy” graph problems that do not lie in NVLIN, and therefore, cannot be captured by any sentence in $\text{ESO}^{(E)}[\#1, \forall 1]$. Intuitively, this follows from the fact that any NRAM running in vertex-linear time cannot read all of the input bits. If the graph has $\omega(n)$ edges, then the NRAM must ignore some bits related to the edges in the graph.

Alternatively, suppose that we have a NRAM $M$ that decides some property $P$ of the graph in vertex-linear time. Now, when given an input graph with $\omega(n)$ edges, the machine $M$ runs in sub-linear time in the size of the *whole input*. Therefore, expressing some graph properties checkable in deterministic linear time in the size of the whole input, such as bipartiteness, is not possible in the logic $\text{ESO}[\#1, \forall 1]$. We will now show that many natural graph problems are not solvable in vertex-linear time.
Lower bounds. For each problem $P \subseteq \text{Str}(\{E\})$, the lower bound construction consists of an infinite family of pairs of graphs and edges of the form $(G_n, F_n)$. The idea is that for each pair $(G_n, F_n)$ the graph $G_n$ has to be accepted, but if any single edge from $F_n$ is added to graph $G_n$, the new graph has to be rejected. In addition, $F_n$ is constructed in such a manner that it contains super-linear number (in $n$) of edges. Thus if there is an NRAM $M$ that decides the problem $P$, then the machine $M$ must read at least $\Omega(n^2)$ bits from the input registers to check whether any edge in $F_n$ is in the input graph. Otherwise, there exists some input graph on which $M$ erroneously accepts or rejects. The idea is formalized in the following lemma.

Lemma 5.3. Let $P$ be a graph problem. If there exists an infinite set

$$A = \{(G_n, F_n): G_n = \langle V, E \rangle, |V(G_n)| = n, F_n \subseteq E(G)\},$$

such that for any $(G_n, F_n) \in A$ the following properties hold:

(i) $|F_n| \in \Omega(n^2)$,

(ii) $G_n = \langle V, E \rangle \in P$,

(iii) $G + e \notin P$ for any $e \in F_n$,

then $P \notin \text{NVLIN}$.

Proof. Suppose $P \in \text{NVLIN}$ and such a set $A$ exists. This means that there exists an NRAM $M$ that recognizes $P$ in at most $cn$ steps for some fixed constant $c \in \mathbb{N}_+$. Since $A$ is infinite and $|F_n| \in \Omega(n^2)$ there exists some $n_0 \in \mathbb{N}_+$ and $n > n_0$ such that $(G_n, F_n) \in A$ and $|F_n| > cn$. Let $(G_n, F_n) \in A$ be such a pair.

Let $G'_n = G_n + e$ where $e \in F_n$. By assumption, the machine $M$ must accept $G_n$ and reject $G'_n$. Since both graphs have $n$ vertices, the machine $M$ can read at most $cn$ bits from the input registers. However, to distinguish the graph $G'_n$ from $G_n$, the machine has to read $|F_n| > cn$ bits in the worst-case. Therefore, $M$ cannot recognize $P$. \qed

We begin with a lower bound for the vertex cover problem. A vertex cover of graph $G = \langle G, E \rangle$ is a set $C \subseteq G$ such that all edges $e \in E$ are incident to some vertex $v \in C$. In the corresponding decision problem, the input consists of a graph $G$ and $k \in \mathbb{N}$ and the task is to decide whether $G$ has a vertex cover of size at most $k$.

We will use a so-called star graph in the construction. A star graph $\mathcal{X}$ is a graph with $V(\mathcal{X}) = [m]$ and $E(\mathcal{X}) = \{\{i, 0\}: i > 0, i \in [m]\}$. That is, a star is a tree of depth 1.

Proposition 8. The vertex cover problem is not in NVLIN.

37
Proof. Fix $k \in \mathbb{N}_+$. For each $n = km$ where $m \in \mathbb{N}_+$, we will define the pair $(G_n, F_n)$ as follows. Let $\mathcal{X}_0, \ldots, \mathcal{X}_{k-1}$ be $k$ copies of the star graph $\mathcal{X}$ such that for each $i \in [k]$ we have that $V(\mathcal{X}_i) = \{(i,j) : j \in V(\mathcal{X})\}$ and $E(\mathcal{X}_i) = \{(i,u), (i,v) : \{u,v\} \in E(\mathcal{X})\}$. The graph $G_n$ is now defined such that the set of vertices is

$$V(G_n) = \bigcup_{i \in [k]} V(\mathcal{X}_i)$$

and set of edges is

$$\bigcup_{i \in [k]} E(\mathcal{X}_i) \cup \{(i,0), (i+1,0) : i \in [k-1]\}.$$  

The graph $G_n$ is illustrated by Figure 3. Finally, the accompanying edge set $F_n$ is given by

$$F_n = \bigcup_{i \in [k]} E(\mathcal{X}_i).$$

It is easy to see that $|F_n| \in \Theta(n^2)$. Observe that the graph $G_n$ is optimally covered by the roots of the stars, that is, by the set $C = [k] \times \{0\}$. However, for all $\emptyset \neq F \subseteq F_n$ the graph $G'_n = (V(G_n), E(G_n) \cup F)$ does not have a vertex cover of size $k$. Therefore by Lemma 5.3, the vertex cover problem is not in NVLIN.

We will now overview some of lower bound results due to Grandjean and Olive [12]. These propositions give us some corollaries related to the composition of the complexity class NVLIN and how it compares to some other complexity classes.

Proposition 9. Deciding whether a graph is a tree is not in NVLIN.

Proof. This follows from the same construction as in Proposition 8 for the case $k = 1$. The graph $G_n$ is a tree by definition, but adding any edge $F_n$ introduces a cycle into the graph. The claim follows from Lemma 5.3. \qed

![Figure 3: The vertex cover lower bound construction. For the $k$-size vertex cover problem, the graph $G_n$ consists of $k$ disjoint star graphs $\mathcal{X}_i$ where $i \in [k]$. The star graphs are connected by their roots and each star has $m$ leaves. The nodes in the optimal $k$-size vertex cover are highlighted.](image-url)
Proposition 10. Deciding whether a graph is not a tree is not in NVLIN.

Proof. The idea is to take two disjoint paths as $G_n$ and all the edges connecting these two paths as $F_n$. Formally, for all $n = 2m$ where $m \in \mathbb{N}_+$, let $G_n$ be such that $V(G_n) = \{(0, i), (1, i) : i \in [m]\}$ and $E(G_n) = \{(i, j), (i, j+1) : i \in [2], j \in [m-1]\}$. Now let $F_n = \{((0, i), (1, j)) : i, j \in [m]\}$ be the set of edges between these two paths.

Observe $G_n$ is not connected, and hence not a tree. However, $G_n$ with any new edge $e \in F_n$ is connected and cycle free. Thus by Lemma 5.3, deciding whether a graph is not a tree is not in NVLIN.

We now consider the the problem of deciding the existence of an Eulerian circuit in a graph. The classic theorem due to Euler states that a graph has an Eulerian circuit if and only if it is connected and all the vertices have an even degree [2, Ch. 1.8]. Thus, the following proposition shows that even some “easy” problems reside outside the class NVLIN.

Proposition 11. Deciding whether a given graph has an Eulerian circuit is not in NVLIN.

Proof. For each $n \geq 3$, the lower bound is given by $\langle C_n, F_n \rangle$ where $C_n$ is the cycle graph and $F_n = E(C_n)$. As $|F_n| = n(n-1)/2$ and $m = n/2$, we have $|F_n| \in \Omega(n^2)$.

Proposition 12. Deciding whether a graph does not contain a Hamiltonian circuit is not in NVLIN.

Proof. Let $n = 2(m+1)$ where $m \in \mathbb{N}_+$. The graph $G_n$ will consist of two disjoint $m$-cycles $C_1$ and $C_2$ along with two additional nodes $a$ and $b$ where $a$ is connected to all nodes in the first cycle, $b$ to all nodes in the second cycle, and there is an edge between $a$ and $b$. Formally, the vertex set of graph $G_n$ is

$$V(G_n) = V(C_1) \cup V(C_2) \cup \{a, b\}$$

and the edge set is defined as

$$E(G_n) = E(C_1) \cup E(C_2) \cup \{v, a : v \in V(C_1)\} \cup \{v, b : v \in V(C_2)\} \cup \{a, b\}.$$ 

Let $F_n = \{u, v\} : u \in V(C_1), v \in V(C_2)$ consist of the set of edges between the two cycles in the complement of $G_n$. As $|F_n| = m(m-1)/2$ and $m = n/2 - 1$, we have $|F_n| \in \Omega(n^2)$.

The graph $G_n$ does not contain a Hamiltonian circuit: any path between the subgraph $C_1$ and subgraph $C_2$ must contain the edge $\{a, b\}$ and any Hamiltonian circuit would need to pass the edge $\{a, b\}$ twice. However, for any edge $e \in F_n$ the graph $G_n + e$ has a Hamiltonian circuit.

While the equality of the complexity classes NP and co-NP remains open, the case is settled for the class NVLIN as the following corollary shows.
Corollary. The class NVLIN is not closed under complement. That is,
\[ \text{NVLIN} \neq \overline{\text{NVLIN}}. \]

Proof. Proposition 2 and 12 show that the Hamiltonian circuit problem is in NVLIN, whereas its complement problem is not.

\[ \square \]

6 Conclusions and open problems

We have proven several results about the descriptive complexity of graph problems and the expressibility of the logic ESO[\#1, \forall 1]. We gave a detailed proof for the correspondence NTIME[n] \equiv ESO[\#1, \forall 1] which can be generalized to the case NTIME[n^d] \equiv ESO[\#d, \forall d].

Using this logical characterization of linear-time computation, it is possible to characterize computational and descriptive complexity of decision problems with relative ease. In particular, for several graph problems, we obtained linear in the number of vertices upper bounds for the non-deterministic time complexity. This implies sublinear time algorithms when measuring complexity in the size of the graph.

Furthermore, Section 5.2 gave tools for exhibiting lower bounds for graph problems. Finally as a corollary, it was shown that the complexity class NVLIN is not closed under complement. Such a result is not yet known for the classes NP and co-NP. Moreover, there are graph problems solvable in deterministic linear time in the size of the input that do not lie either in the class NVLIN or its complement class \overline{NVLIN}. Thus, in some sense, we can argue that the additional computational power of non-determinism is rather limited.

Several questions regarding the complexity and properties of various fragments second-order logic remain open. While many NP-complete problems have been classified with respect to the class NVLIN [12], one task is to classify the remaining classic NP-complete graph problems with respect to this class. In addition, as Grandjean and Olive point out, the graph properties studied so far are either checkable in time O(n) or require time \Omega(n^2) using NRAMs. Of course, it is easy to come up with some problems that fall between these two: a trivial but rather uninteresting problem is to decide whether a graph has over \( n^{3/2} \) edges. However, are there any natural graph problems recognizable in time \( o(n^2) \) which are outside the class NVLIN?

Moreover, are there a strict hierarchies between different fragments of logics ESO[\#k, \forall d]? Durand et al. [3] conjecture that for all vocabularies \( \tau \) where \#\( \tau \leq 1 \) and for each \( d \in \mathbb{N} \), it holds that ESO*[\#1, \forall d] \equiv ESO*[\forall 1]. The authors argue that this would imply that the logic ESO[\#k, \forall d] is strictly weaker than the logic ESO[\#k + 1, d'] for all \( d, d' \in \mathbb{N} \).
Finally, Immerman [14] lists the converse of Lynch’s theorem [16] as an open problem. That is, does the logic \( \text{ESO}[\#d] \) with only existential relation quantifications capture the class of properties checkable with non-deterministic Turing machines running in time \( O(n^d) \)?

**Acknowledgements**

I am grateful to Juha Kontinen for valuable discussions and feedback on this work.

**References**


