Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Theoretical Computer Science 350 (2006) 325–344

Theoretical Computer Science

www.elsevier.com/locate/tcs

On the expressive power of monadic least fixed point logic ☆

Nicole Schweikardt*

Institut für Informatik, Humboldt-Universität Berlin, Unter den Linden 6, D-10099 Berlin, Germany

Abstract

Monadic least fixed point logic MLFP is a natural logic whose expressiveness lies between that of first-order logic FO and monadic second-order logic MSO. In this paper, we take a closer look at the expressive power of MLFP. Our results are:

(1) MLFP can describe graph properties beyond any fixed level of the monadic second-order quantifier alternation hierarchy.
(2) On strings with built-in addition, MLFP can describe at least all languages that belong to the linear time complexity class DLIN.
(3) Settling the question whether

$$\text{addition-invariant MLFP} \stackrel{?}{=} \text{addition-invariant MSO} \quad \text{on finite strings}$$

or, equivalently, settling the question whether

$$\text{MLFP} \stackrel{?}{=} \text{MSO} \quad \text{on finite strings with addition}$$

would solve open problems in complexity theory: "$=$" would imply that PH $=$ PTIME whereas "$\neq$" would imply that DLIN $\neq$ LINH.

Apart from this we give a self-contained proof of the previously known result that MLFP is strictly less expressive than MSO on the class of finite graphs.

Keywords: Finite model theory; Descriptive complexity theory; Fixed point logic; Monadic second-order logic; Linear time complexity classes

1. Introduction

A central topic in Finite Model Theory has always been the comparison of the expressive power of different logics on finite structures. One of the main motivations for such studies is an interest in the expressive power of query languages

☆ Journal version of ICALP'04 paper [27]. Parts of this work were done while the author was supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD) in order to visit the Laboratory for Foundations of Computer Science, University of Edinburgh, Scotland, U.K.

* Tel.: +49 30 2093 3086; fax: +49 2093 3081.
E-mail address: schweika@informatik.hu-berlin.de
URL: http://www.informatik.hu-berlin.de/~schweika.

0304-3975/$ - see front matter
doi:10.1016/j.tcs.2005.10.025

for *relational databases* or for *semi-structured data* such as XML-documents. Relational databases can be modeled as finite relational structures, whereas XML-documents can be modeled as finite labeled trees. Since *first-order logic* FO itself is too weak for expressing many interesting queries, various extensions of FO have been considered as query languages.

When restricting attention to strings and labeled trees, *monadic second-order logic* MSO seems to be "just right": it has been proposed as a yardstick for expressiveness of XML query languages [8] and, due to its connection to finite automata (cf., e.g., [30]), the model-checking problem for (Boolean and unary) MSO-queries on strings and labeled trees can be solved with polynomial time data complexity (cf., e.g., [7]). On finite relational structures in general, however, MSO can express complete problems for all levels of the polynomial time hierarchy [1], i.e., MSO can express queries that are believed to be far too difficult to allow efficient model-checking.

The main focus of the present paper lies on *monadic least fixed point logic* MLFP, which is an extension of first-order logic by a mechanism that allows to define unary relations *by induction*. Precisely, MLFP is obtained by restricting the least fixed point logic FO(LFP) (cf., e.g., [18,6]) to formulas in which only *unary* relation variables are allowed. The expressive power of MLFP lies between the expressive power of FO and the expressive power of MSO. On finite relational structures in general, MLFP has the nice properties that (1) the model-checking problem can be solved with polynomial time and linear space data complexity, and (2) MLFP is "on-spot" for the description of many important problems. For example, the transitive closure of a binary relation, or the set of winning positions in games on finite graphs (cf., e.g., [9] or [6, Exercise 8.1.10]) can be specified by MLFP-formulas. And on strings and labeled trees, MLFP even has exactly the same expressiveness as MSO (with respect to Boolean and unary queries, cf. [30,8]). But for all that, the logic MLFP has received surprisingly little attention in recent years. Considerably more attention has already been paid to monadic fixed point extensions of *propositional modal logic*, which are used as languages for hardware and process specification and verification. A particularly important example of such a logic is the *modal μ-calculus* (cf., e.g., [2]), which can be viewed as the modal analogue of MLFP. *Monadic datalog*, the monadic fixed point extension of *conjunctive queries* (a subclass of FO), has recently been proposed as a database and XML query language that has a good trade-off between the expressive strength, on the one hand, and the complexity of query evaluation, on the other hand [8]. On relational structures in general, however, neither monadic datalog nor the modal μ-calculus can express all of FO, whereas all three logics are included in MLFP.

As already mentioned, the expressive power of MLFP ranges between that of FO and that of MSO. Dawar [3] has shown that 3-*colorability* of finite graphs is not definable in infinitary logic $L^{\omega}_{\infty\omega}$. Since all of MLFP can be expressed in $L^{\omega}_{\infty\omega}$, this implies that the (NP-complete) 3-colorability problem is definable in MSO (even, in *existential* monadic second-order logic $\mathrm{Mon}\Sigma^1_1$), but not in MLFP. Grohe [15] exposed a *polynomial time solvable* graph problem that is not MLFP-definable, but that is definable in FO(LFP) and, as a closer inspection shows, also in MSO. Both results show that on finite graphs MLFP is strictly less expressive than MSO. The first main result of the present paper states that, nevertheless, MLFP has a certain expressive strength, as it can define graph problems beyond any fixed level of the monadic second-order quantifier alternation hierarchy:

Theorem 1.1. *For each $k \geqslant 1$, there is an* MLFP-*definable graph problem that does not belong to the kth level of the monadic second-order quantifier alternation hierarchy.*

When shifting attention from finite graphs to finite strings or labeled trees, the picture is entirely different: there, MLFP, MSO, and $\mathrm{Mon}\Sigma^1_1$ have the same expressive power, namely, of expressing exactly the *regular* languages (cf. [30]). To increase the expressive power of MLFP and MSO on the class of finite strings, one can allow formulas to also use the ternary relation $+$ which is interpreted as the graph of the addition function. For a logic $\mathcal{L}$ we write $\mathcal{L}(+)$ to explicitly indicate that the addition predicate $+$ may be used in $\mathcal{L}$-formulas. In [21,22] Lynch has shown that $\mathrm{NTIME}(n) \subseteq \mathrm{Mon}\Sigma^1_1(+)$ on the class of finite strings with built-in addition. I.e., every string-language decidable by a nondeterministic multi-tape Turing machine with linear time bound is definable by a sentence in $\mathrm{Mon}\Sigma^1_1(+)$. Building upon this, one can show (cf. [24]) that $\mathrm{MSO}(+) = \mathrm{LINH}$, i.e., MSO(+) can define exactly those string-languages that belong to the *linear time hierarchy* (which is the linear time analogue of Stockmeyer's polynomial time hierarchy). Lynch's result was strengthened by Grandjean and Olive [13]: they showed that $\mathrm{Mon}\Sigma^1_1(+)$ can even define all string-languages that belong to the complexity class NLIN. The class NLIN and its deterministic version DLIN are based on linear time random access machines and were introduced by Grandjean in a series of papers [10–12]. As argued in [12], DLIN and NLIN can be seen as "the" adequate mathematical formalizations of linear time complexity.

For example, NLIN contains NTIME(n) and all 21 NP-complete problems listed by Karp in [19]. The class DLIN contains all problems in DTIME(n), i.e. all problems decidable in linear time by a deterministic multi-tape Turing machine. But DLIN also contains problems such as CHECKSORT (given two lists $\ell_1 = s_1, \dots, s_n$ and $\ell_2 = t_1, \dots, t_n$ of strings, decide whether ℓ_2 is the lexicographically sorted version of ℓ_1) which are conjectured not to belong to DTIME(n) (see [28]). In the present paper we show the following analogue of the result of [13]:

Theorem 1.2. *All string-languages that belong to the linear time complexity class* DLIN *are definable in* MLFP(+).

One area of research in Finite Model Theory considers extensions of logics which allow *invariant* uses of some auxiliary relations. For example, *order-invariant* formulas may use a linear ordering of a given structure's universe, but they must not depend on the particular choice of linear ordering. This corresponds to the "real world" situation where the physical representation of a graph or a database, stored in a computer, induces a linear order on the vertices of the graph or the tuples in the database. But this particular order is hidden to the user, because one wants the user's queries to be independent of the particular physical representation of the data. Therefore, for formulating queries, the user may be allowed to use the fact that *some* order is there, but he cannot make his queries depend on any *particular* order, because he does not know which order the data comes with. Similarly, *successor*- or *addition*-invariant formulas may use a successor-relation or an addition-relation on a structure's universe, but must be independent of the particular choice of successor- or addition-relation. Such kinds of invariance have been investigated with respect to first-order logic, e.g., in [17,25,4]. In the present paper we consider *addition-invariant* formulas on finite strings and show that both, the *equivalence* of addition-invariant MLFP and MSO, as well as a *separation* of addition-invariant MLFP from MSO would solve open problems in complexity theory: Let PH denote Stockmeyer's *polynomial time hierarchy* [29], and let LINH be the *linear time hierarchy* (cf., e.g., [5]), i.e., the linear time analogue of PH.

Theorem 1.3. (a) *If addition-invariant* MLFP $\neq$ *addition-invariant* MSO *on the class of finite strings*, *then* DLIN $\neq$ LINH.
(b) *If addition-invariant* MLFP $=$ *addition-invariant* MSO *on the class of finite strings*, *then* PH $=$ PTIME.

In other words, it is most likely that addition-invariant MLFP is strictly less expressive than addition-invariant MSO on strings—but actually *proving* this can be expected to be rather difficult, since it would imply the separation of the complexity class DLIN from the linear time hierarchy LINH.

The paper is structured as follows: Section 2 fixes the basic notations and gives an example of the present paper's use of MLFP-formulas. Theorem 1.1 is proved in Section 3. In Section 4, we give a self-contained proof of the previously known (cf. [3]) result that MLFP is strictly less expressive than MSO on the class of finite graphs. Section 5 concentrates on the proof of Theorem 1.2. Section 6 deals with the proof of Theorem 1.3. Some open questions are pointed out in Section 7.

2. Preliminaries

For an alphabet $\mathbb{A}$ we write $\mathbb{A}^+$ to denote the set of all finite non-empty strings over $\mathbb{A}$. For a set $\mathcal{U}$ we write $2^{\mathcal{U}}$ to denote the power set of $\mathcal{U}$, i.e., $2^{\mathcal{U}} := \{X : X \subseteq \mathcal{U}\}$. We use $\mathbb{N}$ to denote the set $\{0, 1, 2, \dots\}$ of natural numbers. For every $n \in \mathbb{N}$ we write $[n]$ for the set $\{0, \dots, n-1\}$. The logarithm of n with respect to base 2 is denoted $\lg n$.

A (relational) *signature* τ is a finite set of relation symbols. Each relation symbol $R \in \tau$ has a fixed arity $ar(R)$. A τ-structure $\mathcal{A}$ consists of a set $\mathcal{U}^{\mathcal{A}}$ called the *universe* of $\mathcal{A}$, and an interpretation $R^{\mathcal{A}} \subseteq (\mathcal{U}^{\mathcal{A}})^{ar(R)}$ of each relation symbol $R \in \tau$. *All structures considered in this paper are assumed to have a finite universe.*

We assume that the reader is familiar with *first-order logic* FO, *monadic second-order logic* MSO, *least fixed point logic* FO(LFP), and *infinitary logic* $L^{\omega}_{\infty\omega}$ (cf., e.g., the textbooks [6,18]). The kth level, MonΣ^1_k, of the monadic second-order quantifier alternation hierarchy consists of all MSO-formulas that are in prenex normal form, having a prefix of k alternating blocks of set quantifiers, starting with an existential block, and followed by a first-order formula.

We write $\exists X$ FO to denote the class of MonΣ^1_1-formulas that have at most one existential set quantifier.

For a logic $\mathcal{L}$ we use $\mathcal{L}(\tau)$ to denote the class of all $\mathcal{L}$-formulas of signature τ. We write $\varphi(x_1, \dots, x_k, X_1, \dots, X_\ell)$ to indicate that the free first-order variables of the formula φ are $x_1, \dots, x_k$ and the free second-order variables are

$X_1, \ldots, X_\ell$. Sometimes we use $\overline{x}$ and $\overline{X}$ as abbreviations for sequences $x_1, \ldots, x_k$ and $X_1, \ldots, X_\ell$ of variables. A *sentence* φ of signature τ is a formula that has no free variable.

Let τ be a signature, let $\mathcal{C}$ be a class of τ-structures, and let $\mathcal{L}$ be a logic. We say that a set $L \subseteq \mathcal{C}$ *is $\mathcal{L}$-definable in $\mathcal{C}$* if there is a sentence $\varphi \in \mathcal{L}(\tau)$ such that $L = \{\mathcal{A} \in \mathcal{C} : \mathcal{A} \models \varphi\}$. Similarly, we say that a set L of τ-structures is $\mathcal{L}$-definable, iff it is $\mathcal{L}$-definable in the class of all finite τ-structures.

We will mainly consider the *monadic least fixed point logic* MLFP, which is the restriction of least fixed point logic FO(LFP), where fixed point operators are required to be *unary*. For the precise definition of MLFP we refer the reader to the textbook [6] (MLFP is denoted FO(M-LFP) there). *Simultaneous monadic least fixed point logic* S-MLFP is the extension of MLFP by operators that allow to compute the simultaneous least fixed point of several unary operators. In other words: S-MLFP is obtained by restricting simultaneous least fixed point logic FO(S-LFP) to unary fixed point relations. For the formal definition of FO(S-LFP) we, again, refer to [6]. The following example illustrates the present paper's use of S-MLFP-formulas.

Example 2.1. Let $\tau_{<,+}$ be the signature that consists of a binary relation symbol $<$ and a ternary relation symbol $+$. For every $n \in \mathbb{N}$ let $\mathcal{A}_n$ be the $\tau_{<,+}$-structure with universe $[n] = \{0, \ldots, n-1\}$, where $<$ is interpreted by the natural linear ordering and $+$ is interpreted by the graph of the addition function, i.e., $+$ consists of all triples (a, b, c) over $[n]$ where $a+b = c$. Consider the formulas

$$\begin{aligned}\varphi_S(x, S, P) := {} & \text{“}x{=}0\text{”} \vee \text{“}x{=}1\text{”} \ \vee \\ & \exists x_1 \exists x_2 \Big(x_1{<}x_2 \wedge x_2{<}x \wedge S(x_1) \wedge S(x_2) \\ & \qquad \wedge \forall z \big((x_1{<}z \wedge z{<}x_2) \rightarrow P(z) \big) \wedge \ \text{“}x - x_2 = x_2 - x_1 + 2\text{”} \Big), \\ \varphi_P(y, S, P) := {} & \exists x_1 \exists x_2 \Big(x_1{<}x_2 \wedge x_2{<}y \wedge S(x_1) \wedge S(x_2) \\ & \qquad \wedge \forall z \big((x_1{<}z \wedge z{<}x_2) \rightarrow P(z) \big) \wedge \ \text{“}y - x_2 < x_2 - x_1 + 2\text{”} \Big).\end{aligned}$$

Of course, the subformulas written in quotation marks "$\cdots$" can easily be resolved by proper FO($\tau_{<,+}$)-formulas. In the structure $\mathcal{A}_n$, the simultaneous least fixed point $(S^{(\infty)}_{\mathcal{A}_n}, P^{(\infty)}_{\mathcal{A}_n})$ of (φ_S, φ_P) is evaluated as follows: we start with the 0th stage, where S and P are interpreted by the sets $S^{(0)}_{\mathcal{A}_n} = P^{(0)}_{\mathcal{A}_n} = \emptyset$. Inductively, for every $i \in \mathbb{N}$, the $(i+1)$st stage is obtained via

$$\begin{aligned} S^{(i+1)}_{\mathcal{A}_n} &:= \Big\{ a \in [n] : \mathcal{A}_n \models \varphi_S \Big(a, S^{(i)}_{\mathcal{A}_n}, P^{(i)}_{\mathcal{A}_n} \Big) \Big\}, \\ P^{(i+1)}_{\mathcal{A}_n} &:= \Big\{ b \in [n] : \mathcal{A}_n \models \varphi_P \Big(b, S^{(i)}_{\mathcal{A}_n}, P^{(i)}_{\mathcal{A}_n} \Big) \Big\}. \end{aligned}$$

In particular,

$$\begin{array}{llll} S^{(1)}_{\mathcal{A}_n} = \{0, 1\}, & S^{(2)}_{\mathcal{A}_n} = \{0, 1, 4\}, & S^{(3)}_{\mathcal{A}_n} = \{0, 1, 4, 9\}, & S^{(4)}_{\mathcal{A}_n} = \{0, 1, 4, 9, 16\}, \\ P^{(1)}_{\mathcal{A}_n} = \emptyset, & P^{(2)}_{\mathcal{A}_n} = \{2, 3\}, & P^{(3)}_{\mathcal{A}_n} = \{2, 3, 5, 6, 7, 8\} & \cdots \end{array}.$$

At some stage i (with $i \leqslant n$), this process arrives at a fixed point, i.e., at a situation where $S^{(i)}_{\mathcal{A}_n} = S^{(i+1)}_{\mathcal{A}_n} = S^{(j)}_{\mathcal{A}_n}$ and $P^{(i)}_{\mathcal{A}_n} = P^{(i+1)}_{\mathcal{A}_n} = P^{(j)}_{\mathcal{A}_n}$, for every $j > i$. This particular tuple $\Big(S^{(i)}_{\mathcal{A}_n}, P^{(i)}_{\mathcal{A}_n}\Big)$ is called the *simultaneous least fixed point* $\Big(S^{(\infty)}_{\mathcal{A}_n}, P^{(\infty)}_{\mathcal{A}_n}\Big)$ of (φ_S, φ_P) in $\mathcal{A}_n$. It is not difficult to see that for our example formulas φ_S and φ_P we obtain that $S^{(\infty)}_{\mathcal{A}_n}$ is the set of all square numbers in $[n]$, whereas $P^{(\infty)}_{\mathcal{A}_n}$ is the set of all non-square numbers in $[n]$.

Now, $[\text{S-LFP}_{x,S,y,P}\, \varphi_S, \varphi_P]_S(u)$ is an S-MLFP-formula that is satisfied by exactly those elements u in $\mathcal{A}_n$'s universe that belong to $S^{(\infty)}_{\mathcal{A}_n}$, i.e., that are square numbers. Similarly, $[\text{S-LFP}_{x,S,y,P}\, \varphi_S, \varphi_P]_P(u)$ is an S-MLFP-formula that is satisfied by those elements u in $\mathcal{A}_n$'s universe that belong to $P^{(\infty)}_{\mathcal{A}_n}$, i.e., that are non-square numbers.

In the above example we have seen that, given the addition relation $+$, the set of square numbers is definable in S-MLFP. It is known (cf., e.g., [16, Corollary 4.4]) that MLFP has the same expressive power as S-MLFP.

Since S-MLFP-definitions of certain properties or relations are sometimes easier to find and more convenient to read than equivalent MLFP-definitions, we will often present S-MLFP-definitions instead of MLFP-definitions.

3. MLFP and the MSO quantifier alternation hierarchy

In this section, we show that MLFP can define graph problems beyond any fixed level of the monadic second-order quantifier alternation hierarchy.

Let τ_{graph} be the signature that consists of a binary relation symbol E. We write $\mathcal{C}_{\text{graphs}}$ for the class of all finite directed graphs. A graph $G = \langle V^G, E^G \rangle$ is called *undirected* if the following is true: for every $v \in V^G$, $(v, v) \notin E^G$, and for every $(v, w) \in E^G$, also $(w, v) \in E^G$. We write $\mathcal{C}_{\text{ugraphs}}$ to denote the class of all finite undirected graphs. Let $\tau_{\text{grid}} := \{S_1, S_2\}$ be a signature consisting of two binary relation symbols. The *grid* of height m and width n is the τ_{grid}-structure

$$\underline{[m, n]} := \langle \{1, .., m\} \times \{1, .., n\}, S_1^{m,n}, S_2^{m,n} \rangle ,$$

where $S_1^{m,n}$ is the "vertical" successor relation consisting of all tuples $\big((i, j), (i+1, j)\big)$ in $\{1, .., m\} \times \{1, .., n\}$, and $S_2^{m,n}$ is the "horizontal" successor relation consisting of all tuples $\big((i, j), (i, j+1)\big)$. We define $\mathcal{C}_{\text{grids}} := \big\{\underline{[m, n]} : m, n \geqslant 1\big\}$ to be the class of all finite grids. It was shown in [23] that the monadic second-order quantifier alternation hierarchy is *strict* on the class of finite graphs and the class of finite grids. In the present paper we will use the following result:

Theorem 3.1 (*Matz et al. [23]*). *For every $k \geqslant 1$ there is a set L_k of finite grids such that L_k is definable in* $\text{Mon}\Sigma_k^1$ *but not in* $\text{Mon}\Sigma_{k-1}^1$ (*in the class $\mathcal{C}_{\text{grids}}$ of finite grids*).

Using the construction of [23] and the fact that MLFP is as expressive as S-MLFP, it is an easy (but tedious) exercise to show the following

Corollary 3.2. *For every $k \geqslant 1$ the set L_k is definable in* MLFP *and in* $\text{Mon}\Sigma_k^1$, *but not in* $\text{Mon}\Sigma_{k-1}^1$ (*in the class $\mathcal{C}_{\text{grids}}$ of finite grids*).

Proof (*sketch*). For every $k \geqslant 1$ we inductively define functions $f_k : \mathbb{N} \to \mathbb{N}$ via $f_1(m) := 2^m$ and $f_{k+1}(m) := f_k(m) \cdot 2^{f_k(m)}$, for all $m \in \mathbb{N}$.

For the set $L_k := \Big\{\underline{[m, f_k(m)]} : m \geqslant 1\Big\}$ it was shown in [23] that there is a $\text{Mon}\Sigma_k^1$-sentence but no $\text{Mon}\Sigma_{k-1}^1$-sentence that is satisfied by exactly those grids that belong to L_k. We will now point out that the sets L_k are definable in MLFP.

Let us start with the set L_1. Given a grid $G = \underline{[m, n]}$ one can check whether G's width is $f_1(m) = 2^m$ by writing binary representations of length m of the numbers $0, 1, 2, .., 2^m - 1$ into successive columns of the grid. Precisely, the *column-numbering* of a grid $G = \underline{[m, n]}$ is the uniquely defined subset C of G's universe that satisfies the following conditions:

(1) $(i, 1) \notin C$, for all $i \leqslant m$, and
(2) for all $j > 1$ we have
  - $(i, j-1) \in C$ and $(i, j) \notin C$, for all $i \leqslant m$, or
  - $\sum_{i=1}^m c_{i,j-1} \cdot 2^{m-j} + 1 = \sum_{i=1}^m c_{i,j} \cdot 2^{m-j}$, where, for all $(i', j') \in \{1, .., m\} \times \{1, .., n\}$, $c_{i',j'} := 1$ if $(i', j') \in C$ and $c_{i',j'} := 0$ otherwise.

Obviously, a grid G belongs to L_1 iff G's rightmost column is the unique column that is completely contained in G's column-numbering.

To continue the proof of Corollary 3.2, we need the following:

Lemma 3.3. *There is an* MLFP-*formula column-numbering*(x) *such that, for all grids G and all vertices x in G's universe, we have $G \models$ column-numbering(x) if, and only if, x belongs to the column numbering of G.*

Proof. Since MLFP has the same expressive power as S-MLFP (cf., Section 2), we may define simultaneously, by induction on the columns of the grid, the column-numbering C and its complement D by an S-MLFP-formula *column-numbering*(x) of the form $[\text{S-LFP}_{x,C,y,D}\varphi_C, \varphi_D]_C(x)$.

The formula $\varphi_C(x, C, D)$ states that

- there is a vertex x_0 in the same column as x such that the horizontal predecessor x_0' of x_0 belongs to D and all vertices below x_0' and in the same column as x_0' belong to C, and either $x{=}x_0$ or x is vertically above x_0 and the horizontal predecessor of x belongs to C.

The formula $\varphi_D(y, C, D)$ states that

- y is in the leftmost column of the grid, or
- $C(y')$ is true for all vertices y' in the column directly left to y's column, or
- there is a vertex y_0 in the same column as y such that the horizontal predecessor y_0' of y_0 belongs to D and all vertices below y_0' and in the same column as y_0' belong to C, and either y is vertically below y_0, or y is vertically above y_0 and the horizontal predecessor of y belongs to D.

It is straightforward to formalize this by MLFP-formulas φ_C and φ_D that are positive in the set variables C and D, and to check that the resulting formula

$$\textit{column-numbering}(x) := [\text{S-LFP}_{x,C,y,D}\ \varphi_C, \varphi_D]_C(x)$$

has the desired property. This completes the proof of Lemma 3.3. □

Let us now continue with the proof of Corollary 3.2.

Using Lemma 3.3, one can easily formulate an MLFP-sentence φ_{L_1} that is satisfied by exactly those grids that belong to L_1: the formula φ_{L_1} states that the formula *column-numbering*(x) is true for all vertices in the rightmost column of the grid, and that if the formula *column-numbering*(x) is true for all vertices of a column of the grid then this is the grid's rightmost column.

Let us now concentrate on the definition of the set L_2. (The definition of L_k for $k > 2$ will be a straightforward generalization of the construction for L_2.)

Given a grid $G = \underline{[m, n]}$ one can check whether G's width is

$$f_2(m) := f_1(m) \cdot 2^{f_1(m)}$$

by writing binary representations of length $f_1(m)$ of the numbers $0, 1, 2, \ldots, 2^{f_1(m)-1}$ into the first row of the grid. Precisely, an f_1*-numbering* of a grid $G = \underline{[m, n]}$ is a set Y_1 of top-row vertices of G, satisfying the following conditions:

(1) $(1, j) \notin Y_1$, for all $1 \leqslant j \leqslant f_1(m)$, and
(2) for all $1 \leqslant b \leqslant n/f_1(m) - 1$ we have
- $\big(1,\ f_1(m)\cdot(b{-}1) + j\big) \in Y_1$ and $\big(1,\ f_1(m)\cdot b + j\big) \notin Y_1$, for all $1 \leqslant j \leqslant f_1(m)$, or
- $\sum_{j=1}^{f_1(m)} y_{f_1(m)\cdot(b-1)+j} \cdot 2^{f_1(m)-j} + 1 = \sum_{j=1}^{f_1(m)} y_{f_1(m)\cdot b+j} \cdot 2^{f_1(m)-j}$, where, for all $v \in \{1, \ldots, n\}$, we define $y_v := 1$ if $(1, v) \in Y_1$ and $y_v := 0$ otherwise.

Using the formula *column-numbering*(x) of Lemma 3.3 it is not difficult to formulate, in analogy to the proof of Lemma 3.3, an S-MLFP-formula f_1*-numbering*(x) which expresses that x is a top-row vertex that belongs to the f_1-numbering of the underlying grid. Generalizing this construction and the definition of f_1-numbering from f_1 to f_k in the obvious way, it is straightforward to show the following:

Lemma 3.4. *For every $k \geqslant 1$ there is an* MLFP*-formula f_k-numbering(x) such that, for all grids G and all vertices x in the top row of G, we have*

$$G \models f_k\textit{-numbering}(x)$$

if, and only if, x belongs to the f_k-numbering of G.

Using this together with Theorem 3.1, one easily obtains Corollary 3.2. □

Note that the above corollary deals with structures over the signature τ_{grid} that consists of two binary relation symbols. In the remainder of this section we will transfer this to the classes $\mathcal{C}_{\text{graphs}}$ and $\mathcal{C}_{\text{ugraphs}}$. To this end, we need a further result of [23] which uses the notion of *strong first-order reductions*. The precise definition of this notion is of no particular importance for the present paper—for completeness, it is given in Definition 3.5 below. What *is* important is that a strong first-order reduction from a class $\mathcal{C}$ of τ-structures to a class $\mathcal{C}'$ of τ'-structures is an injective mapping $\Phi : \mathcal{C} \to \mathcal{C}'$ such that every structure $\mathcal{A} \in \mathcal{C}$ can be interpreted in the structure $\Phi(\mathcal{A})$ and, vice versa, $\Phi(\mathcal{A})$ can be interpreted in $\mathcal{A}$.

Definition 3.5 (*strong first-order reduction, Matz et al. [23]*). Let $n \geqslant 1$, and let $\mathcal{C}$ and $\mathcal{C}'$ be classes of structures over the relational signatures τ and τ', respectively. A *strong first-order reduction from $\mathcal{C}$ to $\mathcal{C}'$ with rank n* is an injective mapping $\Phi : \mathcal{C} \to \mathcal{C}'$ such that

(1) For every structure $\mathcal{A} \in \mathcal{C}$, the universe of $\Phi(\mathcal{A})$ is a disjoint union of n copies of the universe of $\mathcal{A}$. Precisely, $\mathcal{U}^{\Phi(\mathcal{A})} = \bigcup_{i=1}^{n} \big(\{i\} \times \mathcal{U}^{\mathcal{A}}\big)$.

(2) There is an FO(τ')-formula $\varphi_{\text{rep}}(x_1, \ldots, x_n)$ which describes the n-tuples of the form $\big((1, a), \ldots, (n, a)\big)$, which serve as representatives of elements a in $\mathcal{U}^{\mathcal{A}}$. Precisely, for all $\mathcal{A} \in \mathcal{C}$, all $a_1, \ldots, a_n \in \mathcal{U}^{\mathcal{A}}$, and all $i_1, \ldots, i_n \in \{1, \ldots, n\}$, we have

$$\Phi(\mathcal{A}) \models \varphi_{\text{rep}}\big((i_1, a_1), \ldots, (i_n, a_n)\big) \iff i_j = j \text{ and } a_j = a_1, \text{ for all } j \in \{1, \ldots, n\}.$$

(3) For every relation symbol $R \in \tau$ of arity $r := ar(R)$, there is an FO(τ')-formula $\varphi^R(x_1, \ldots, x_r)$ such that, for all $\mathcal{A} \in \mathcal{C}$ and all $a_1, \ldots, a_r \in \mathcal{U}^{\mathcal{A}}$,

$$\mathcal{A} \models R(a_1, \ldots, a_r) \iff \Phi(\mathcal{A}) \models \varphi^R\big((1, a_1), \ldots, (1, a_r)\big).$$

(4) For every relation symbol $R' \in \tau'$ of arity $r' := ar(R')$ and every tuple $\kappa = (\kappa_1, \ldots, \kappa_{r'}) \in \{1, \ldots, n\}^{r'}$, there is an FO($\tau$)-formula $\varphi_\kappa^{R'}$ such that, for all $\mathcal{A} \in \mathcal{C}$ and all $a_1, \ldots, a_{r'} \in \mathcal{U}^{\mathcal{A}}$,

$$\mathcal{A} \models \varphi_\kappa^{R'}(a_1, \ldots, a_{r'}) \iff \Phi(\mathcal{A}) \models R'\big((\kappa_1, a_1), \ldots, (\kappa_{r'}, a_{r'})\big).$$

Note that the formulas in items (2) and (3) allow to "simulate" a τ-structure $\mathcal{A}$ in the τ'-structure $\Phi(\mathcal{A}) \in \mathcal{C}'$, whereas the formulas from item 4. allow to "simulate" the τ'-structure $\Phi(\mathcal{A})$ in the τ-structure $\mathcal{A}$.

The fundamental use of strong first-order reductions comes from the following result:

Theorem 3.6 (*Matz et al. [23, Theorem 33]*). *Let $\mathcal{C}$ and $\mathcal{C}'$ be classes of structures over the relational signatures τ and τ', respectively. Let Φ be a strong first-order reduction from $\mathcal{C}$ to $\mathcal{C}'$. Let $\mathcal{L}$ be one of the logics* MonΣ_k^1, *for some $k \geqslant 0$, or the logic*[1] $L_{\infty\omega}^{\omega}$. *Let the image $\Phi(\mathcal{C}) := \{\Phi(\mathcal{A}) : \mathcal{A} \in \mathcal{C}\}$ of Φ be $\mathcal{L}$-definable in $\mathcal{C}'$. Then, the following is true for every $L \subseteq \mathcal{C}$*:

$$L \text{ is } \mathcal{L}\text{-definable in } \mathcal{C} \iff \Phi(L) \text{ is } \mathcal{L}\text{-definable in } \mathcal{C}'.$$

In the present paper, the following strong first-order reductions will be used:

Proposition 3.7 (*Matz et al. [23, Proposition 38]*). (a) *There exists a strong first-order reduction Φ_1 from $\mathcal{C}_{\text{grids}}$ to $\mathcal{C}_{\text{graphs}}$, and the image $\Phi_1(\mathcal{C}_{\text{grids}})$ of Φ_1 is* MonΣ_2^1-*definable and* MLFP-*definable in $\mathcal{C}_{\text{graphs}}$.*

(b) *There exists a strong first-order reduction Φ_2 from $\mathcal{C}_{\text{graphs}}$ to $\mathcal{C}_{\text{ugraphs}}$, and the image $\Phi_2(\mathcal{C}_{\text{graphs}})$ of Φ_2 is* FO-*definable in $\mathcal{C}_{\text{ugraphs}}$.*

This directly allows to transfer Theorem 3.1 from finite grids to finite graphs and finite undirected graphs, respectively. To also transfer Corollary 3.2 from $\mathcal{C}_{\text{grids}}$ to $\mathcal{C}_{\text{graphs}}$ and $\mathcal{C}_{\text{ugraphs}}$, we need the following easy lemma:

Lemma 3.8. *Let $\mathcal{C}$ and $\mathcal{C}'$ be classes of structures over the relational signatures τ and τ', respectively. Let Φ be a strong first-order reduction from $\mathcal{C}$ to $\mathcal{C}'$. Every* MLFP(τ)-*sentence ψ can be translated into an* MLFP(τ')-*sentence ψ' such that, for every $\mathcal{A} \in \mathcal{C}$, $\mathcal{A} \models \psi \iff \Phi(\mathcal{A}) \models \psi'$.*

[1] The logic $L_{\infty\omega}^{\omega}$ is not explicitly mentioned in [23], but it is straightforward to see that the proof given there also works for $L_{\infty\omega}^{\omega}$.

Proof. Let $\mathcal{C}$ and $\mathcal{C}'$ be classes of structures over the signatures τ and τ', respectively. Let Φ be a strong first-order reduction from $\mathcal{C}$ to $\mathcal{C}'$. By induction on the construction of the MLFP(τ)-formulas ψ we define MLFP(τ')-formulas ψ' as follows:

- If ψ is an atomic formula of the form Xx or $x = y$ then $\psi' := \psi$.
- If ψ is an atomic formula of the form $R(x_1, \ldots, x_r)$, then $\psi' := \varphi^R(x_1, \ldots, x_r)$.
- If ψ is of the form $\psi_1 \wedge \psi_2$, then $\psi' := \psi'_1 \wedge \psi'_2$. Similarly, if $\psi = \psi_1 \vee \psi_2$ (or $\psi = \neg\psi_1$), then $\psi' := \psi'_1 \vee \psi'_2$ (or $\psi' := \neg\psi'_1$), respectively).
- If ψ is of the form $\exists x \psi_1$, then $\psi' := \exists x\big(\psi'_1 \wedge \exists x_2 \cdots \exists x_n \varphi_{\text{rep}}(x, x_2, \ldots, x_n)\big)$.
  I.e., quantification is relativized to elements that belong to the *first* disjoint copy of the original structure's universe.
- If ψ is of the form $[\text{LFP}_{x,X}\ \psi_1](y)$, then
  $\psi' := [\text{LFP}_{x,X}\ \psi'_1 \wedge \exists x_2 \cdots \exists x_n \varphi_{\text{rep}}(x, x_2, \ldots, x_n)](y)$.

I.e., fixed points will only contain elements that belong to the first disjoint copy of the original structure's universe. Now let ψ be an arbitrary MLFP(τ)-formula with free set variables $X_1, \ldots, X_t$ and free first-order variables $x_1, \ldots, x_m$. It is straightforward to check that the following is true for all $\mathcal{A} \in \mathcal{C}$ and all sets $U_1, \ldots, U_t \subseteq \mathcal{U}^{\mathcal{A}}$ and all elements $a_1, \ldots, a_m \in \mathcal{U}^{\mathcal{A}}$:

$$\begin{aligned} &\mathcal{A} \models \psi(U_1, \ldots, U_t, a_1, \ldots, a_m) \quad \iff \\ &\Phi(\mathcal{A}) \models \psi'\big(\{1\} \times U_1, \ldots, \{1\} \times U_t, (1, a_1), \ldots, (1, a_m)\big). \end{aligned}$$

If ψ is a *sentence*, this in particular means that $\mathcal{A} \models \psi$ if, and only if, $\Phi(\mathcal{A}) \models \psi'$. This completes the proof of Lemma 3.8. □

Using this, it is not difficult to prove this section's main result:

Theorem 3.9. *For every $k \geqslant 2$ there is a set D_k of finite directed graphs* (*respectively*, *a set U_k of finite undirected graphs) such that D_k* (*respectively*, *U_k*) *is definable in* MLFP *and* MonΣ^1_k, *but not in* MonΣ^1_{k-1}.

Proof. Let $k \geqslant 2$, and let L_k be the set of grids from Corollary 3.2. I.e., L_k is MonΣ^1_k-definable and MLFP-definable, but not MonΣ^1_{k-1}-definable in $\mathcal{C}_{\text{grids}}$. We use the strong first-order reduction Φ_1 from $\mathcal{C}_{\text{grids}}$ to $\mathcal{C}_{\text{graphs}}$ obtained from Proposition 3.7, and we choose $D_k := \Phi_1(L_k)$. From Theorem 3.6 we conclude that $\mathcal{D}_k$ is MonΣ^1_k-definable, but not MonΣ^1_{k-1}-definable in $\mathcal{C}_{\text{graphs}}$. To show that D_k is MLFP-definable, let ψ_k be an MLFP(τ_{grid})-sentence that defines L_k in $\mathcal{C}_{\text{grids}}$. Let ψ'_k be the MLFP(τ_{graph})-sentence obtained from ψ_k with Lemma 3.8. Note that for every directed graph $D \in \mathcal{C}_{\text{graphs}}$ we have that

$$D \in D_k \quad \iff \quad D \in \Phi(\mathcal{C}_{\text{grids}}) \quad \text{and} \quad D \models \psi'_k \quad \iff \quad D \models \hat{\phi}_1 \wedge \psi'_k\,,$$

where $\hat{\phi}_1$ is an MLFP(τ_{graph})-sentence that defines the image $\Phi_1(\mathcal{C}_{\text{grids}})$ of Φ_1.

Altogether, we have seen that D_k is a set of directed graphs which is definable in MLFP and in MonΣ^1_k, but not in MonΣ^1_{k-1}.

To expose a similar set U_k of undirected graphs, we choose $U_k := \Phi_2(D_k)$, where Φ_2 is a strong first-order reduction from $\mathcal{C}_{\text{graphs}}$ to $\mathcal{C}_{\text{ugraphs}}$, obtained from Proposition 3.7. In a similar way as done above for D_k, we obtain that U_k is definable in MLFP and in MonΣ^1_k, but not in MonΣ^1_{k-1} (one just needs to replace D_k by U_k and L_k by D_k). This completes the proof of Theorem 3.9. □

4. MLFP is less expressive than MSO on finite graphs

In [3], Dawar exposed a clever and intricate proof that 3-colorability of finite graphs cannot be expressed in $L^{\omega}_{\infty\omega}$. Since MLFP is less expressive than $L^{\omega}_{\infty\omega}$ and 3-colorability is easily definable in MSO, this implies that MLFP is less expressive than MSO on finite graphs.

For the sake of completeness, this section gives an easy and relatively self-contained proof that MLFP is less expressive than MSO on finite graphs. To this end, a graph representation of the *satisfiability problem* is shown to be definable in MSO, but not in MLFP.

Let $\tau_{P,N} := \{P, N\}$ be a signature consisting of two binary relation symbols P and N. We identify every propositional formula θ in conjunctive normal form with at most three literals per clause (for short: θ is a 3-CNF formula) with a $\tau_{P,N}$-structure $\mathcal{A}_\theta$ as follows: The universe of $\mathcal{A}_\theta$ consists of an element a_κ, for every clause κ of θ, and an element a_x, for every propositional variable x of θ. The relations $P^{\mathcal{A}_\theta}$ and $N^{\mathcal{A}_\theta}$ indicate, which variables occur positively, respectively, negated in which clauses. I.e.,

$$P^{\mathcal{A}_\theta} := \{ (a_\kappa, a_x) : \text{variable } x \text{ occurs unnegated in clause } \kappa \},$$
$$N^{\mathcal{A}_\theta} := \{ (a_\kappa, a_x) : \text{variable } x \text{ occurs negated in clause } \kappa \}.$$

We define the *satisfiability problem* 3-SAT$_{P,N}$ as follows:

$$\text{3-SAT}_{P,N} := \left\{ \mathcal{A}_\theta \,:\, \theta \text{ is a } \textit{satisfiable} \text{ 3-CNF formula} \right\}.$$

The set of 3-SAT$_{P,N}$-instances is the set

$$\mathcal{C}_{\text{3-SAT}_{P,N}} := \{\mathcal{A}_\theta : \theta \text{ is a 3-CNF formula}\}.$$

Recall from Section 2 that we write $\exists X$ FO to denote the class of MonΣ^1_1-formulas that have at most one existential set quantifier.

Lemma 4.1. 3-SAT$_{P,N}$ *is* $\exists X$ FO-*definable in* $\mathcal{C}_{\text{3-SAT}_{P,N}}$.

Proof. Obviously, a 3-CNF formula θ is satisfiable if, and only if, there exists a subset X of θ's set of propositional variables, such that the following is true: in every clause κ of θ there is a variable x that occurs positively in κ and belongs to X, or there is a variable x that occurs negated in κ and does not belong to X.

Clearly, a node $b \in \mathcal{U}^{\mathcal{A}_\theta}$ represents a *clause* of θ iff there exists a node a such that $(b, a) \in P^{\mathcal{A}_\theta}$ or $(b, a) \in N^{\mathcal{A}_\theta}$. Therefore, 3-SAT$_{P,N}$ is defined in $\mathcal{C}_{\text{3-SAT}_{P,N}}$ by the $\exists X$ FO-sentence

$$\exists X \,\forall y \, \big(\exists x \; P(y,x) \vee N(y,x)\big) \rightarrow \exists x \, \big(P(y,x) \wedge X(x)\big) \vee \big(N(y,x) \wedge \neg X(x)\big). \qquad \square$$

We assume that the reader is familiar with the notion of *first-order reductions* (cf., e.g., [18]).

Lemma 4.2. 3-SAT$_{P,N}$ *is not* $L^{\omega}_{\infty\omega}$-*definable in* $\mathcal{C}_{\text{3-SAT}_{P,N}}$.

Proof. It is well-known that 3-SAT$_{P,N}$ is NP-complete with respect to first-order reductions (cf., e.g., [18, Proposition 7.17]). I.e., for every class $\mathcal{C}$ of finite structures and every problem $L \subseteq \mathcal{C}$ that belongs to the complexity class NP, there is a first-order reduction from $L \subseteq \mathcal{C}$ to 3-SAT$_{P,N} \subseteq \mathcal{C}_{\text{3-SAT}_{P,N}}$.

For the sake of contradiction let us now assume that 3-SAT$_{P,N}$ is $L^{\omega}_{\infty\omega}$-definable in $\mathcal{C}_{\text{3-SAT}_{P,N}}$. Since $L^{\omega}_{\infty\omega}$ is closed under first-order reductions, we then obtain that *every* problem in NP is $L^{\omega}_{\infty\omega}$-definable. However, it is well-known that e.g. the problem EVEN, consisting of all finite structures whose universe has even cardinality, is *not* $L^{\omega}_{\infty\omega}$-definable in the class of all finite structures (cf., e.g., [6, Example 3.3.13]). Since EVEN obviously belongs to NP, this yields a contradiction. $\square$

Representing 3-SAT$_{P,N}$-instances by finite graphs, we obtain this section's main result:

Theorem 4.3. *There is a set D of finite directed graphs and a set U of finite undirected graphs such that D and U are definable in* MSO (*even in* $\exists X$ FO) *but not in* MLFP (*even not in* $L^{\omega}_{\infty\omega}$).

Proof. We start with the construction of a set D of directed graphs that is $\exists X$ FO-definable but not $L^{\omega}_{\infty\omega}$-definable.

We choose D as a variant of 3-SAT$_{P,N}$ where every 3-CNF formula θ is represented by a directed graph G_θ rather than a $\{P, N\}$-structure $\mathcal{A}_\theta$. The graph G_θ is defined as follows: The universe of G_θ consists of a vertex a_κ, for every clause κ of θ, and two vertices b_x and c_x, for every propositional variable x of θ. A vertex of the form b_x is intended to encode the literal "x", whereas c_x shall encode the literal "$\neg x$". The graph G_θ has the following edges: for every clause κ and every variable x that occurs positively (respectively, negated) in κ, there is an edge from a_κ to b_x (respectively, to c_x). Furthermore, for every variable x there is an edge from b_x to b_x (a self-loop), an edge from b_x to c_x, and an edge from c_x to b_x.

Clearly, a vertex v of G_θ represents a literal "x" iff v has a self-loop; and v represents a literal "$\neg x$" iff "$\exists w\ E(v,w) \wedge E(w,w) \wedge E(w,v) \wedge \neg\, v{=}w$".

We choose

$$D := \text{3-SAT}_E := \{\, G_\theta : \theta \text{ is a satisfiable 3-CNF formula} \,\}.$$

The set of 3-SAT$_E$-instances is the set $\mathcal{C}_{\text{3-SAT}_E} := \{G_\theta : \theta \text{ is a 3-CNF formula}\}$. It is straightforward to see that 3-SAT$_E$ is $\exists X$ FO-definable in $\mathcal{C}_{\text{3-SAT}_E}$ (cf., Lemma 4.1) and that $\mathcal{C}_{\text{3-SAT}_E}$ is FO-definable in $\mathcal{C}_{\text{graphs}}$. Therefore, $D := \text{3-SAT}_E$ is $\exists X$ FO-definable in $\mathcal{C}_{\text{graphs}}$.

Similarly to the NP-completeness of 3-SAT$_{P,N}$ one obtains that 3-SAT$_E$ is NP-complete with respect to first-order reductions. Therefore, the fact that 3-SAT$_E$ is *not* $L^\omega_{\infty\omega}$-definable in $\mathcal{C}_{\text{graphs}}$ can be proved in the same way as Lemma 4.2.

Altogether, we obtain that the set $D := \text{3-SAT}_E$ of finite directed graphs is $\exists X$ FO-definable but not $L^\omega_{\infty\omega}$-definable in $\mathcal{C}_{\text{graphs}}$.

For transfering this result to *undirected graphs*, we use the strong first-order reduction Φ_2 from $\mathcal{C}_{\text{graphs}}$ to $\mathcal{C}_{\text{ugraphs}}$ obtained from Proposition 3.7. We define $U := \Phi_2(D)$. Since D is $\exists X$ FO-definable, we obtain from Theorem 3.6 (for $\mathcal{L} := \text{Mon}\Sigma^1_1$) that U is $\text{Mon}\Sigma^1_1$-definable (and a closer look at the construction shows that even $\exists X$ FO suffices). Furthermore, since D is not $L^\omega_{\infty\omega}$-definable, Theorem 3.6 (for $\mathcal{L} := L^\omega_{\infty\omega}$) implies that also U is not $L^\omega_{\infty\omega}$-definable. This completes the proof of Theorem 4.3. □

5. MLFP and linear time complexity

We identify a string $w = w_0 \cdots w_{n-1}$ of length $|w| = n \geqslant 1$ over an alphabet $\mathbb{A}$ with a structure $\underline{w}$ in the usual way: we choose $\tau_\mathbb{A}$ to consist of the binary relation symbol $<$ and a unary relation symbol P_a, for each letter $a \in \mathbb{A}$. We choose $\underline{w}$ to be the $\tau_\mathbb{A}$-structure $\langle \{0,\ldots,n{-}1\}, <, (P^w_a)_{a\in\mathbb{A}} \rangle$, where $<$ denotes the natural linear ordering of $[n] := \{0,\ldots,n{-}1\}$ and P^w_a consists of all positions of w that carry the letter a.

In this section, we equip the structure $\underline{w}$ with an additional ternary addition relation $+$. I.e., we identify the string w with the structure $\langle \underline{w}, + \rangle := \langle [n], <, +, (P^w_a)_{a\in\mathbb{A}} \rangle$, where $+$ consists of all triples $(a,b,c) \in [n]^3$ with $a + b = c$. We identify the set $\mathbb{A}^+$ of all non-empty strings over alphabet $\mathbb{A}$ with the set $\mathcal{C}_\mathbb{A} := \{\underline{w} : w \in \mathbb{A}^+\}$, respectively, with the set $\mathcal{C}_{\mathbb{A},+} := \{\langle \underline{w}, + \rangle : w \in \mathbb{A}^+\}$.

To give the precise definition of Grandjean's linear time complexity class DLIN, we need the following notion of *random access machines*, basically taken from [14].

A DLIN-RAM $\mathcal{R}$ is a random access machine that consists of two *accumulators* A and B, a *special register* M, *registers* R_i, for every $i \in \mathbb{N}$, and a *program* that is a finite sequence $\mathcal{I}(1),\ldots,\mathcal{I}(r)$ of *instructions*, each of which is of one of the following forms:

- $A := 0$,
- $A := 1$,
- $A := M$,
- $A := A + B$,
- $A := R_A$,
- $R_A := B$,
- $M := A$,
- $B := A$,
- IF $A{=}B$ THEN $\mathcal{I}(i_0)$ ELSE $\mathcal{I}(i_1)$,
- HALT.

The meaning of most of these instructions is straightforward. The "IF $A{=}B$ THEN $\mathcal{I}(i_0)$ ELSE $\mathcal{I}(i_1)$" instruction enforces to continue with program line i_0, if the contents of registers A and B are identical, and to continue with line i_1 otherwise. If the accumulator A contains a number i, then the execution of the instruction $A := R_A$ copies the content of register R_i into the accumulator A. Similarly, the execution of the instruction $R_A := B$ copies the content of accumulator B into register R_i. We stipulate that the last instruction, $\mathcal{I}(r)$, is the instruction HALT.

The input to $\mathcal{R}$ is assumed to be present in the first registers of $\mathcal{R}$ at the beginning of the computation. Precisely, an *input to* $\mathcal{R}$ is a function $f : [m] \to [m]$, for an arbitrary $m \in \mathbb{N}$. The initial content of the special register M is the number m, and for every $i \in \mathbb{N}$, the initial content of register R_i is $f(i)$ if $i \in [m]$, and 0 otherwise. The accumulators A and B are initialized to 0. The computation of $\mathcal{R}$ starts with instruction $\mathcal{I}(1)$ and finishes when it encounters a HALT statement. We say that $\mathcal{R}$ *accepts* an input f, if the content of register R_0 is non-zero when $\mathcal{R}$ reaches a HALT statement.

$\mathcal{R}$ *recognizes* a set $\mathcal{F} \subseteq \{f : [m] \to [m] : m \in \mathbb{N}\}$ *in time* $\mathcal{O}(m)$, if

(1) $\mathcal{R}$ accepts an input f if, and only if, $f \in \mathcal{F}$, and
(2) there is a number $d \in \mathbb{N}$ such that $\mathcal{R}$ is *d-bounded*, i.e., for every $m \in \mathbb{N}$ and every $f : [m] \to [m]$ the following is true: when started with input f, $\mathcal{R}$ performs less than $d \cdot m$ computation steps before reaching a HALT statement, and throughout the computation, each register and each accumulator contains numbers of size $< d \cdot m$.

To use DLIN-RAMs for recognizing *string*-languages, one represents strings w by functions f_w as follows (cf. [13]). W.l.o.g. we restrict attention to strings over the alphabet $\mathbb{A} := \{1, 2\}$. For every $n \geqslant 1$ we define $\ell(n) := \lceil \frac{1}{2}\lg(n+1) \rceil$ and $m(n) := \lceil n/\ell(n) \rceil$. A string w over $\mathbb{A} = \{1, 2\}$ of length n can (uniquely) be decomposed into substrings $w_0, w_1, \ldots, w_{m(n)-1}$ such that

- w is the concatenation of the strings $w_0, \ldots, w_{m(n)-1}$,
- w_i has length $\ell(n)$, for every $i < m(n)-1$, and
- $w_{m(n)-1}$ has length at most $\ell(n)$.

For each $i \in [m(n)]$ let w_i^{dy} be the integer whose dyadic representation is w_i. I.e., if $w_i = d_0 \cdots d_{\ell(n)-1}$ with $d_j \in \{1, 2\}$, then $w_i^{\mathrm{dy}} = \sum_{j<\ell(n)} d_j \cdot 2^j$. It is straightforward to see that $w_i^{\mathrm{dy}} < m(n)$. Now, w is represented by the function $f_w : [m(n)] \to [m(n)]$ with $f_w(i) := w_i^{\mathrm{dy}}$, for every $i \in [m(n)]$.

Definition 5.1 (DLIN, *Grandjean [12]*). A string-language L over alphabet $\mathbb{A} = \{1, 2\}$ belongs to the complexity class DLIN if, and only if, the set of its associated functions $\{f_w : w \in L\}$ is recognized by a DLIN-RAM in time $\mathcal{O}(m)$.

At first sight, the class DLIN may seem a bit artificial: a string w of length n is represented by a function f_w of domain $[m(n)]$ where $m(n)$ is of size $\Theta(n/\lg n)$. A DLIN-RAM with input f_w is allowed to perform only $\mathcal{O}(n/\lg n)$ computation steps, with register contents of size $\mathcal{O}(n/\lg n)$. However, as argued in [10–12,14], DLIN is a very reasonable formalization of the intuitive notion of "linear time complexity". In particular, DLIN contains all string-languages recognizable by a deterministic Turing machine in $\mathcal{O}(n)$ steps, and, in addition, also some problems (such as CHECKSORT, cf., Section 1) that are conjectured not to be solvable by Turing machines with time bound $\mathcal{O}(n)$.

Grandjean and Olive [13] showed that $\mathrm{Mon}\Sigma_1^1(+)$ can define (at least) all string-languages that belong to the nondeterministic version NLIN of DLIN. In the remainder of this section we show the following analogue of the result of [13]:

Theorem 5.2 (DLIN $\subseteq$ MLFP(+) *on finite strings with built-in addition*). *For every finite alphabet* $\mathbb{A}$ *and every string-language* $L \subseteq \mathbb{A}^+$ *in* DLIN *there is an* MLFP$(\tau_{\mathbb{A}} \cup \{+\})$*-sentence* φ_L *such that, for every* $w \in \mathbb{A}^+$ *we have* $w \in L$ *iff* $\langle \underline{w}, + \rangle \models \varphi_L$.

The proof of [13]'s result on NLIN and $\mathrm{Mon}\Sigma_1^1(+)$ uses, as an intermediate step, a characterization of the class NLIN by a logic that existentially quantifies unary functions. There also exists an algebraic characterization of the class DLIN via unary functions [14]. Unfortunately, this characterization is not suitable for being used as an intermediate step in the proof of Theorem 5.2. What *can* be used for the proof of Theorem 5.2, however, is the following representation, basically taken from [13], of a *run* of a d-bounded DLIN-RAM $\mathcal{R}$. A run of $\mathcal{R}$ with input $f : [m] \to [m]$ is fully described by 6 functions $I, A, B, M, R_A, R'_A : [d \cdot m] \to [d \cdot m]$:

$I(t)$ = the number of the instruction performed in computation step $t+1$,
$A(t)$ = content of the accumulator A directly *before* performing step $t+1$,
$B(t)$ = content of the accumulator B directly *before* performing step $t+1$,
$M(t)$ = content of the special register M directly *before* performing step $t+1$,
$R_A(t)$ = content of register $R_{A(t)}$ directly *before* performing step $t+1$,
$R'_A(t)$ = content of register $R_{A(t)}$ directly *after* performing step $t+1$.

It is not difficult to give inductive definitions of these functions:

$$I(0) := 1 \quad \text{and}$$

$$I(t{+}1) := \begin{cases} i_0 & \text{if } A(t) = B(t) \text{ and } \mathcal{I}\big(I(t)\big) = \text{“IF } A{=}B \text{ THEN } \mathcal{I}(i_0) \text{ ELSE } \mathcal{I}(i_1)\text{”}, \\ i_1 & \text{if } A(t) \neq B(t) \text{ and } \mathcal{I}\big(I(t)\big) = \text{“IF } A{=}B \text{ THEN } \mathcal{I}(i_0) \text{ ELSE } \mathcal{I}(i_1)\text{”}, \\ I(t) & \text{if } \mathcal{I}\big(I(t)\big) = \text{“HALT”}, \\ I(t){+}1 & \text{otherwise.} \end{cases}$$

$$A(0) := 0 \quad \text{and} \quad A(t{+}1) := \begin{cases} j & \text{if } \mathcal{I}\big(I(t)\big) = \text{“}A := j\text{”} \ \ (\text{with } j \in \{0,1\}) \\ M(t) & \text{if } \mathcal{I}\big(I(t)\big) = \text{“}A := M\text{”} \\ A(t) + B(t) & \text{if } \mathcal{I}\big(I(t)\big) = \text{“}A := A + B\text{”}, \\ R_A(t) & \text{if } \mathcal{I}\big(I(t)\big) = \text{“}A := R_A\text{”}, \\ A(t) & \text{otherwise.} \end{cases}$$

$$B(0) := 0 \quad \text{and} \quad B(t{+}1) := \begin{cases} A(t) & \text{if } \mathcal{I}\big(I(t)\big) = \text{“}B := A\text{”}, \\ B(t) & \text{otherwise.} \end{cases}$$

$$M(0) := m \quad \text{and} \quad M(t{+}1) := \begin{cases} A(t) & \text{if } \mathcal{I}\big(I(t)\big) = \text{“}M := A\text{”}, \\ M(t) & \text{otherwise.} \end{cases}$$

For defining the function R_A, note that for $i := A(t)$ the content of register R_i, directly before performing computation step $t{+}1$, can be derived as follows: if there does not exist an $s < t$ with $A(s) = i$, then R_i still contains its *initial* value, i.e., the value 0 in case that $i \geqslant m$, and the value $f(i)$ in case that $i \in [m]$. On the other hand, if $s{+}1$ is the largest computation step $\leqslant t$ before which the accumulator A had content i (i.e., $A(s) = i$), then, before performing step $t{+}1$, R_i still contains the value it had after finishing computation step $s{+}1$. I.e., R_i contains the value $R'_A(s)$. This leads to the following inductive definition of R_A:

$$R_A(t) := \begin{cases} 0 & \text{if (1.) there is no } s < t \text{ with } A(s) = A(t) \text{ and (2.) } A(t) \geqslant m, \\ f\big(A(t)\big) & \text{if (1.) there is no } s < t \text{ with } A(s) = A(t) \text{ and (2.) } A(t) < m, \\ R'_A(s) & \text{otherwise, where } s := \max\{s : s < t \text{ and } A(s) = A(t)\}. \end{cases}$$

$$R'_A(t) := \begin{cases} B(t) & \text{if } \mathcal{I}\big(I(t)\big) = \text{“}R_A := B\text{”}, \\ R_A(t) & \text{otherwise.} \end{cases}$$

The *flattening* $\widetilde{G}$ of a function $G : [d{\cdot}m] \to [d{\cdot}m]$ is the concatenation of the $\{0, 1\}$-strings $\widetilde{G}_0, \widetilde{G}_1, \ldots, \widetilde{G}_{dm-1}$, where $\widetilde{G}_i$ is the reverse binary representation of length $l := \lfloor \lg(d{\cdot}m) + 1 \rfloor$ of the number $G(i)$. I.e., $\widetilde{G}_i = b_0 b_1 \cdots b_{l-1}$ with $b_j \in \{0, 1\}$ and $G(i) = \sum_{j<l} b_j{\cdot}2^j$. It is straightforward to see that for every $d \in \mathbb{N}$ there is a $c \in \mathbb{N}$ such that the following is true for every $n \in \mathbb{N}$ and every function $G : [d{\cdot}m(n)] \to [d{\cdot}m(n)]$: The flattening $\widetilde{G}$ of G is a $\{0, 1\}$-string of length $\leqslant c \cdot n$. Consequently, $\widetilde{G}$ can be represented by c subsets $\widetilde{G}^{(0)}, \ldots, \widetilde{G}^{(c-1)}$ of $[n]$ as follows: For every $p \in [n]$ and $\gamma \in [c]$, the $(\gamma{\cdot}n + p)$-th position of $\widetilde{G}$ carries the letter 1 if, and only if, $p \in \widetilde{G}^{(\gamma)}$.

We write $\widetilde{G^\bullet}$ for the *complement* of $\widetilde{G}$, i.e., the $\{0, 1\}$-string obtained from $\widetilde{G}$ by replacing every 0 by 1 and every 1 by 0. Similarly, for $\gamma \in [c]$, $\widetilde{G^\bullet}^{(\gamma)}$ denotes the complement of the set $\widetilde{G}^{(\gamma)}$.

Clearly, given a string w of length n and its functional representation $f_w : [m(n)] \to [m(n)]$, the flattenings (and their complements) of the functions $I, A, B, M, R_A, R'_A : [d{\cdot}m(n)] \to [d{\cdot}m(n)]$ that describe the computation of $\mathcal{R}$ on input f_w, can be represented by a fixed number of subsets of $[n]$. Using the inductive definitions of the functions I, A, B, M, R_A, R'_A mentioned above, we can show the following:

Lemma 5.3. *Let $\mathbb{A} := \{1, 2\}$, let $L \subseteq \mathbb{A}^+$, let $d \in \mathbb{N}$, let $\mathcal{R}$ be a d-bounded* DLIN-RAM *that recognizes the set $\{f_w : w \in L\}$, and let $c \in \mathbb{N}$ be such that, for every $n \geqslant 1$, the flattening $\widetilde{G}$ of every function $G : [d{\cdot}m(n)] \to [d{\cdot}m(n)]$ is a $\{0, 1\}$-string of length $\leqslant c{\cdot}n$. For every symbol $S \in \mathscr{S} := \big\{I, A, B, M, R_A, R'_A, I^\bullet, A^\bullet, B^\bullet, M^\bullet, R_A{}^\bullet, R'_A{}^\bullet\big\}$ and every $\gamma \in [c]$ let $X_{S,\gamma}$ be a set variable. Let $\overline{X_{\mathscr{S},c}}$ be the list of the set variables $X_{S,\gamma}$ for all $S \in \mathscr{S}$ and all $\gamma \in [c]$.*

For every $S \in \mathscr{S}$ and every $\gamma \in [c]$ there is an MLFP$(\tau_{\mathbb{A}} \cup \{+\})$*-formula $\varphi_{S,\gamma}(x, \overline{X_{\mathscr{S},c}})$ such that the following is true for every string $w \in \mathbb{A}^+$:*

Let n be the length of w, let $f_w : [m(n)] \to [m(n)]$ be the functional representation of w, and let $I, A, B, M, R_A, R'_A : [d{\cdot}m(n)] \to [d{\cdot}m(n)]$ be the functions that describe the computation of $\mathcal{R}$ on input f_w. In the structure $\langle \underline{w}, + \rangle$, the simultaneous least fixed point of all the formulas $\varphi_{S,\gamma}$ (for all $S \in \mathscr{S}$ and $\gamma \in [c]$) consists exactly of the sets $\big(\widetilde{S}^{(\gamma)}\big)_{S\in\mathscr{S},\gamma\in[c]}$ that represent the flattenings, and their complements, of the functions I, A, B, M, R_A, R'_A.

Some details on the proof of Lemma 5.3 are given below. An important tool for the proof of Lemma 5.3, also used later in Section 6, is the following:

Lemma 5.4 (*full arithmetic and counting in* MLFP(+)).

(a) *There are* MLFP(+)*-formulas*

$$\varphi_<(x,y), \quad \varphi_\times(x,y,z), \quad \varphi_{Exp}(x,y,z), \quad \varphi_{Bit}(x,y), \quad \varphi_{Dy_1}(x,y), \quad \varphi_{Dy_2}(x,y),$$

such that for all $n \in \mathbb{N}$ and all $a,b,c \in [n]$, $\langle [n], + \rangle \models \varphi_<(a,b)$ (respectively, $\varphi_\times(a,b,c)$, $\varphi_{Exp}(a,b,c)$, $\varphi_{Bit}(a,b)$, $\varphi_{Dy_1}(a,b)$, $\varphi_{Dy_2}(a,b)$) if, and only if, $a<b$ (resp., $a\times b = c$, $a^b = c$, the b-th bit in the binary representation of a is 1, *the b-th bit in the dyadic representation of a is* 1, *resp.*, 2).

(b) *Let Y be a unary relation symbol. There is an* MLFP(+)*-formula $\varphi_\#(x,Y)$ such that for all $n \in \mathbb{N}$, all $a \in [n]$, and all $B \subseteq [n]$ we have that*

$$\langle [n], + \rangle \models \varphi_\#(a,B) \iff a = |B|.$$

Proof. (a) Clearly, one can choose $\varphi_<(x,y) := \neg x{=}y \wedge \exists z\ x{+}z = y$.

From Example 2.1, we know that there is an MLFP($<,+$)-formula $\varphi_{Squares}(x)$ that defines exactly the set of square numbers. It is known (cf., e.g., [26]) that, given the addition relation and the set of square numbers, the multiplication relation $\times$ is definable in first-order logic. Furthermore, it is well-known that having $+$ and $\times$ available, first-order logic can define the exponentiation function *Exp* and the *Bit* predicate *Bit* (cf., e.g., [18]).

The formulas $\varphi_{Dy_1}(x,y)$ and $\varphi_{Dy_2}(x,y)$ can easily be obtained by using the *Bit* predicate and the connection between binary and dyadic representation of natural numbers (cf., e.g., [13, Section 5]).

(b) For simplicity we assume that n is a power of 2 and a multiple of $\lg n$. The general case (for arbitrary natural numbers n) can be treated in a similar way.

We identify every set $B \subseteq [n]$ with a $\{0,1\}$-string $\mathbf{B} = b_0 \cdots b_{n-1}$ of length n in the usual way via $b_i := 1$ iff $i \in B$. $\mathbf{B}$ is the concatenation of $n/\lg n$ substrings $B_1, \ldots, B_{n/\lg n}$, each of length $\lg n$. For each such substring B_i there is a (unique) number $a_i \in [n]$ such that B_i is the (reverse) binary representation of a_i. From the *Bit Sum Lemma* (cf. [18, Lemma 1.18]) one obtains an FO($<$, *Bit*)-formula $\varphi_{\mathrm{BitSum}}(x,y)$ which expresses that y is the number of ones in the (reverse) binary representation of x. For every $i \leqslant n/\lg n$, let c_i be the number of ones in the (reverse) binary representation of a_i, and let C_i be the (reverse) binary representation of length $\lg n$ of c_i. Let $\mathbf{C} := C_1 \cdots C_{n/\lg n}$, and let $C \subseteq [n]$ be the subset of $[n]$ that corresponds to the $\{0,1\}$-string $\mathbf{C}$.

Note that $|B|$ is exactly the number of ones in the $\{0,1\}$-string $\mathbf{B}$ which, in turn, is the sum of the numbers c_i (for $i = 1, \ldots, n/\lg n$). We compute the sum of the c_i by maintaining "running sums" as follows: let $s_1 := c_1$, and for every $i < n/\lg n$ let $s_{i+1} := s_i + c_{i+1}$. Clearly, $s_{n/\lg n} = |B|$. Since each s_i is $\leqslant n$, it has a (reverse) binary representation S_i of length $\lg n$. Let $\mathbf{S} := S_1 \cdots S_{n/\lg n}$, and let $S \subseteq [n]$ be the subset of $[n]$ that corresponds to the $\{0,1\}$-string $\mathbf{S}$.

Using the formula $\varphi_{\mathrm{BitSum}}(x,y)$, it is straightforward to construct an MLFP($<$, *Bit*)-formula φ_C which, given a set B, specifies the corresponding set C. Using this set C, it is not difficult to find an S-MLFP($<$, *Bit*, $+$, $\times$)-formula φ_S which inductively defines the set S as well as S's complement. Finally, by construction of the set S, the number $|B|$ of elements in B is the number whose (reverse) binary representation is identical to the rightmost substring of length $\lg n$ of $\mathbf{S}$.

From (a) we know that the predicates $<$, *Bit*, $\times$ can be defined in MLFP(+). Altogether, this leads to an MLFP(+)-formula $\varphi_\#(x,Y)$ with the desired properties.

This completes the proof of Lemma 5.4. $\square$

Proof for Lemma 5.3 (*sketch*). For every $S \in \mathscr{S} = \{I, A, B, M, R_A, R'_A, I^\bullet, A^\bullet, B^\bullet, M^\bullet, R_A{}^\bullet, R'_A{}^\bullet\}$ let X_S be a set variable. Let $\overline{X_{\mathscr{S}}}$ be the list of the set variables X_S, for all $S \in \mathscr{S}$.

In what follows, we indicate how to construct, for every $S \in \mathscr{S}$, a formula $\psi_S(x, \overline{X_{\mathscr{S}}})$ that is positive in all set variables in $\overline{X_{\mathscr{S}}}$, such that the following is true for every string $w \in \mathbb{A}^+$: the simultaneous least fixed point of all the formulas ψ_S (for all $S \in \mathscr{S}$) in the structure $\langle [c \cdot n], <, +, (P_a^w)_{a \in \mathbb{A}} \rangle$ consists exactly of the flattenings $\widetilde{I}$, $\widetilde{A}$, $\widetilde{B}$, $\widetilde{M}$, $\widetilde{R_A}$, $\widetilde{R'_A}$ and their complements $\widetilde{I^\bullet}$, $\widetilde{A^\bullet}$, $\widetilde{B^\bullet}$, $\widetilde{M^\bullet}$, $\widetilde{R_A{}^\bullet}$, $\widetilde{R'_A{}^\bullet}$.

Note that once having constructed the formulas ψ_S, it is straightforward to obtain the formulas $\varphi_{S,\gamma}$ for $\gamma \in [c]$ whose existence is stated in Lemma 5.3.

Using the formulas from Lemma 5.4(a), it is not difficult to find formulas $\chi_m(m)$ and $\chi_l(l)$ which ensure that the variables m and l are interpreted by the values $m(n)$ and $\lfloor \lg(d \cdot m(n)) + 1 \rfloor$, respectively.

For the construction of the formulas $\psi_S(x, \overline{X_{\mathscr{S}}})$ we use the inductive definitions of the functions I, A, B, M, R_A, R'_A given above and the fact that we have available the arithmetic operations $+$, $\times$, *Bit*, etc. (cf., Lemma 5.4).

The formula $\psi_I(x, \overline{X_{\mathscr{S}}})$ is of the form

$$\exists t' \quad \big(t' \cdot l \leqslant x < (t'+1) \cdot l \big) \quad \wedge \quad \big(t' = 0 \rightarrow x = 0 \big)$$
$$\wedge \left(t' > 0 \rightarrow \quad \exists t \quad t+1 = t' \; \wedge \bigwedge_{1 \leqslant s \leqslant r} \text{“}I(t) \neq s\text{”} \vee \psi_{I,s}(t, t', x) \right).$$

The subformula $(t' = 0 \rightarrow x = 0)$ is for the induction start $I(0) := 1$ which means for $\widetilde{I}$ that the positions $0, \ldots, l-1$ of $\widetilde{I}$ are the reverse binary representation of the number 1, i.e., the string “$100 \cdots 00$”.

The subformula “$I(t) \neq s$” checks that at the positions p with $t \cdot l \leqslant p < t \cdot l$, $\widetilde{I}$ does *not* consist of the reverse binary representation of the number s. For this we assume, by induction, that on these positions X_I coincides with $\widetilde{I}$ and $X_{I^\bullet}$ coincides with $\widetilde{I^\bullet}$. We need both, X_I and $X_{I^\bullet}$, because we want a formula that is *positive* in the set variables $\overline{X_{\mathscr{S}}}$. Precisely, “$I(t) \neq s$” can be chosen to be a formula of the form

$$\bigvee_{j : Bit(s,j)} X_{I^\bullet}(t \cdot l + j) \quad \vee \quad \bigvee_{j : \neg Bit(s,j)} X_I(t \cdot l + j)\,.$$

The subformula $\psi_{I,s}(t, t', x)$ depends on the line $\mathcal{I}(s)$ of $\mathcal{R}$'s program:
If $\mathcal{I}(s) =$ “IF A=B THEN $\mathcal{I}(i_0)$ ELSE $\mathcal{I}(i_1)$”, then $\psi_{I,s}(t, t', x)$ is of the form

$$\left(\text{“}A(t) \neq B(t)\text{”} \vee \bigvee_{j : Bit(i_0,j)} x = t' \cdot l + j \right) \quad \wedge \quad \left(\text{“}A(t) = B(t)\text{”} \vee \bigvee_{j : Bit(i_1,j)} x = t' \cdot l + j \right).$$

Here, “$A(t) \neq B(t)$” can be checked via

$$\exists y \;\; \big(t \cdot l \leqslant y < (t+1) \cdot l \big) \wedge \Big(\big(X_A(y) \wedge X_{B^\bullet}(y) \big) \vee \big(X_{A^\bullet}(y) \wedge X_B(y) \big) \Big).$$

Similarly, “$A(t) = B(t)$” can be checked via

$$\forall y \;\; \big(t \cdot l \leqslant y < (t+1) \cdot l \big) \rightarrow \Big(\big(X_A(y) \wedge X_B(y) \big) \vee \big(X_{A^\bullet}(y) \wedge X_{B^\bullet}(y) \big) \Big).$$

If $\mathcal{I}(s) =$ “HALT”, then $\psi_{I,s}(t, t', x)$ is of the form “$X_I(x - l)$”.
If $\mathcal{I}(s)$ is neither a HALT-statement nor an IF-statement, then $\psi_{I,s}(t, t', x)$ is of the form

$$\exists z \; \exists z' \quad z + 1 = z' \wedge \text{“}I(t) = z\text{”} \wedge Bit(z', x - t \cdot l)\,.$$

Here, “$I(t) = z$” can be checked by the formula

$$\forall i \;\; \Big(Bit(z, i) \rightarrow \big(i < l \wedge X_I(t \cdot l + i) \big) \Big) \wedge \Big(\big(i < l \wedge \neg Bit(z, i) \big) \rightarrow X_{I^\bullet}(t \cdot l + i) \Big).$$

This completes the definition of the formula $\psi_I(x, \overline{X_{\mathscr{S}}})$.

The formulas $\psi_{I^\bullet}, \psi_A, \psi_{A^\bullet}, \psi_B, \psi_{B^\bullet}, \psi_M, \psi_{M^\bullet}, \psi_{R'_A}, \psi_{R'^\bullet_A}$ can be obtained in a similar way.

The formula $\psi_{R_A}(x, \overline{X_{\mathcal{S}}})$ is of the form

$$\exists\, t\ \big(\, t \cdot l \leqslant x < (t{+}1) \cdot l \,\big) \ \wedge\ \Big(\Big(\big(\forall\, s\ s < t \rightarrow \text{“}A(s) \neq A(t)\text{”}\big) \wedge \psi_{\text{init}}(t, x)\Big) \\ \vee \big(\exists\, s\ \ s < t \wedge \text{“}A(s) = A(t)\text{”} \wedge \big(\forall\, s'(s < s' < t) \rightarrow \text{“}A(s') \neq A(t)\text{”}\big) \wedge \psi_{\text{lookup}}(s, t, x)\big)\Big).$$

Here, $\psi_{\text{init}}(t, x)$ is a formula that checks that $i := A(t) < m$ and the $(x - t{\cdot}l)$-th bit in the (reverse) binary representation of $f_w(i)$ is 1. The number $f_w(i)$ can be obtained from the input string w by using the formulas φ_{Dy_1} and φ_{Dy_2} from Lemma 5.4.

The formula $\psi_{\text{lookup}}(s, t, x)$ is of the form “$X_{R'_A}\big(s{\cdot}l + (x - t{\cdot}l)\big)$”.

This completes the definition of the formula ψ_{R_A}. The formula $\psi_{R_A^\bullet}$ can be obtained in a similar way.

It is straightforward (but tedious) to check that the formulas $\psi_S(x, \overline{X_{\mathcal{S}}})$, for $S \in \mathcal{S}$, have the desired properties. Precisely, one can show that the sets obtained in the $(2{\cdot}t)$-th stage of the simultaneous least fixed point process coincide with the respective flattenings (at least) on all positions $< t \cdot l$.

This completes the proof sketch for Lemma 5.3. □

Using Lemma 5.3, Lemma 5.4, and the fact that MLFP has the same expressive power as S-MLFP (cf., Section 2), it is rather straightforward to find an MLFP$(\tau_{\mathbb{A}} \cup \{+\})$-sentence φ_L which, for every string $w \in \mathbb{A}^+$, is satisfied by $\langle \underline{w}, + \rangle$ if, and only if, $\mathcal{R}$ *accepts* input f_w. This, finally, will complete the proof of Theorem 5.2:

Proof for Theorem 5.2 (*sketch*). Let $\mathbb{A} := \{1, 2\}$, and let $\mathcal{R}$ be a d-bounded DLIN-RAM that recognizes the string-language $L \subseteq \mathbb{A}^+$. The aim is to find an MLFP$(\tau_{\mathbb{A}} \cup \{+\})$-sentence φ_L such that, for every $w \in \mathbb{A}^+$, we have $\langle \underline{w}, + \rangle \models \varphi_L \iff w \in L \iff \mathcal{R}$ *accepts* input f_w.

From Lemma 5.3 and the fact that MLFP is as expressive as S-MLFP we directly obtain MLFP$(\tau_{\mathbb{A}} \cup \{+\})$-formulas $\chi_{\widetilde{A}^{(\gamma)}}(x)$ and $\chi_{\widetilde{R'_A}^{(\gamma)}}(x)$, for $\gamma \in [c]$, which represent the flattenings of the functions A and R'_A as follows: If n is the length of an input string $w \in \mathbb{A}^+$, and $I, A, B, M, R_A, R'_A : [d{\cdot}m(n)] \rightarrow [d{\cdot}m(n)]$ are the functions that describe the computation of $\mathcal{R}$ on input f_w, then we have for each $\gamma \in [c]$ and every position $a \in [n]$ that

$$\langle \underline{w}, + \rangle \models \chi_{\widetilde{A}^{(\gamma)}}(a) \iff a \in \widetilde{A}^{(\gamma)}$$

and

$$\langle \underline{w}, + \rangle \models \chi_{\widetilde{R'_A}^{(\gamma)}}(a) \iff a \in \widetilde{R'_A}^{(\gamma)}.$$

Recall that $\mathcal{R}$ *accepts* input f_w if, and only if, the content of register R_0 is non-zero when $\mathcal{R}$ reaches a HALT statement. The content of register R_0 at the end of the computation can be obtained as follows: let $t \geqslant 0$ be such that $t{+}1$ is the *largest* computation step before which the accumulator A did contain the value 0.[2] Clearly, $R'_A(t)$ is the content of register R_0 directly after performing step $t{+}1$, and this is still is the content of register R_0 at the end of the computation (because after step $t{+}1$, accumulator A never has the content 0 again, and hence there is no chance of changing the value of register R_0 ever again).

Therefore, the formula φ_L which checks whether $\mathcal{R}$ accepts f_w, can be obtained as follows:

1. *Use the formulas* $\chi_{\widetilde{A}^{(\gamma)}}(x)$, *for* $\gamma \in [c]$, *to find the largest* $t < d{\cdot}m(n)$, *for which* $A(t) = 0$.
   The value $m := m(n)$, as well as the value $l := \lfloor \lg(d{\cdot}m) + 1 \rfloor$ can be obtained by using the formulas from Lemma 5.4 (a). Furthermore, $A(t) = 0$ if, and only if, $\widetilde{A}$ has the letter 0 at all positions p with $t{\cdot}l \leqslant p < (t{+}1){\cdot}l$. This, in turn, can be checked by inspecting the set $\widetilde{A}^{(\gamma)}$, for a suitable $\gamma \in [c]$.
2. *Use the formulas* $\chi_{\widetilde{R'_A}^{(\gamma)}}(x)$, *for* $\gamma \in [c]$, *to check whether* $R'_A(t) = 0$.
   This, of course, can be done in a similar way as checking whether $A(t) = 0$.

This completes the proof sketch for Theorem 5.2. □

[2] Such a t exists, because at the beginning of the computation, i.e., before computation step 1, the accumulator A has the initial content 0.

6. Addition-invariant MLFP

In this section, we concentrate on *addition invariant* formulas, i.e., on formulas that may use an addition relation on the underlying universe but that are independent of the particular choice of the addition relation.

The notion of “addition relation” is defined as follows: let $\mathcal{U}$ be a finite set, let $n := |\mathcal{U}|$, and let $\oplus$ be a ternary relation on $\mathcal{U}$. $\oplus$ is called *an addition relation on* $\mathcal{U}$ if there is a linear ordering $\oslash$ of $\mathcal{U}$ such that $\mathcal{U} = \{u_0, .., u_{n-1}\}$ with $u_0 \oslash \cdots \oslash u_{n-1}$ and $\oplus = \{ (u_i, u_j, u_k) : i + j = k \text{ and } i, j, k \in \{0, .., n-1\} \}$.

We say that $\oplus$ is the particular addition relation that fits to the linear ordering $\oslash$.

Definition 6.1 (*addition-invariance*). Let $\mathcal{L}$ be a logic, let τ be a signature, and let $\oplus$ be a ternary relation symbol that does not occur in τ. An $\mathcal{L}(\tau \cup \{\oplus\})$-formula $\varphi(x_1, .., x_k)$ is called *addition-invariant* if the following is true for all finite τ-structures $\mathcal{A}$: for any two addition relations $\oplus_1$ and $\oplus_2$ on $\mathcal{U}^{\mathcal{A}}$ and all $a_1, .., a_k \in \mathcal{U}^{\mathcal{A}}$ we have $\langle \mathcal{A}, \oplus_1 \rangle \models \varphi(a_1, .., a_k) \iff \langle \mathcal{A}, \oplus_2 \rangle \models \varphi(a_1, .., a_k)$.

Using Lemma 5.4, we can show

Lemma 6.2. *On linearly ordered structures, addition-invariant* MLFP *can define the particular addition relation that fits to the given linear ordering of the underlying structure.*

Proof. Let $\mathcal{U}$ be a finite universe, let $n := |\mathcal{U}|$, and let $<$ be the given linear ordering of $\mathcal{U}$. Let $v_0, .., v_{n-1}$ be such that $\mathcal{U} = \{v_0, .., v_{n-1}\}$ and $v_0 < \cdots < v_{n-1}$. The aim is to find an addition-invariant MLFP($<, \oplus$)-formula that defines the addition relation

$$\boxplus := \{ (v_i, v_j, v_k) : i + j = k \text{ and } i, j, k \in \{0, .., n-1\} \}.$$

All we know is that we are given an addition relation $\oplus$ that fits to some linear ordering $\oslash$ of $\mathcal{U}$. I.e., there are elements $u_0 \oslash \cdots \oslash u_{n-1}$ such that $\mathcal{U} = \{u_0, .., u_{n-1}\}$ and

$$\oplus = \{ (u_i, u_j, u_k) : i + j = k \text{ and } i, j, k \in \{0, .., n-1\} \}.$$

From Lemma 5.4 we obtain an MLFP($\oplus$)-formula $\varphi_{\#}(x, Y)$ such that for all $a \in \mathcal{U}$ and all $B \subseteq \mathcal{U}$ we have that

$$\langle \mathcal{U}, \oplus \rangle \models \varphi_{\#}(a, B) \iff a = u_{|B|}.$$

Let y be a first-order variable that does not occur in $\varphi_{\#}$.
Let $\psi(x, y)$ be the MLFP($<, \oplus$)-formula obtained from $\varphi_{\#}$ by replacing every atom of the form Yz with the atom $(z<y)$. It is not difficult to see that for all $i, j \in [n]$ we have

$$\langle \mathcal{U}, <, \oplus \rangle \models \psi(u_i, v_j) \iff i = j.$$

I.e., the formula ψ allows to translate predicates from the ordering $\oslash$ to the ordering $<$. In particular, the addition $\boxplus$ can be defined by the MLFP($<, \oplus$)-formula

$$\varphi_{\boxplus}(x, y, z) := \exists x' \, \exists y' \, \exists z' \;\; \psi(x', x) \wedge \psi(y', y) \wedge \psi(z', z) \wedge x' \oplus y' = z'.$$

This completes the proof of Lemma 6.2. □

From Theorem 5.2 and Lemma 6.2 one directly obtains

Corollary 6.3 (DLIN $\subseteq$ *addition-invariant* MLFP *on the class of finite strings*). *For every finite alphabet* $\mathbb{A}$ *and every string-language* $L \subseteq \mathbb{A}^+$ *in* DLIN *there is an addition-invariant* MLFP*-sentence* φ *of signature* $\tau_{\mathbb{A}} \cup \{\oplus\}$ *such that, for every string* $w \in \mathbb{A}^+$ *and every addition relation* $\oplus$ *on* $\underline{w}$*'s universe*, $w \in L$ iff $\langle \underline{w}, \oplus \rangle \models \varphi$.

Using this and the well-known result that the satisfiability problem for *quantified Boolean formulas with k alternations of quantifiers* is complete for the *k*th level of the polynomial time hierarchy, we can show that both, the equivalence of addition-invariant MLFP and MSO, as well as a separation of addition-invariant MLFP from MSO would solve open problems in complexity theory:

Theorem 6.4. (a) *If addition-invariant* MLFP $\neq$ *addition-invariant* MSO *on the class of finite strings*, *then* DLIN $\neq$ LINH.

(b) *If addition-invariant* MLFP $=$ *addition-invariant* MSO *on the class of finite strings*, *then* PH $=$ PTIME.

Proof. (a) It is known (cf. [24]) that

$$\text{MSO}(+) = \text{LINH} \quad \text{on the class } \mathcal{C}_{\mathbb{A},+} \text{ of finite strings with addition,}$$

i.e., MSO(+) can define exactly those string languages that belong to the *linear time hierarchy*. This, together with Lemma 6.2 and the fact that MSO can express all of MLFP, immediately implies that also

$$\text{addition-invariant MSO} = \text{LINH} \quad \text{on the class of finite strings}.$$

From Corollary 6.3 we know that

$$\text{DLIN} \subseteq \text{addition-invariant MLFP} \quad \text{on the class of finite strings}.$$

Therefore, if addition-invariant MLFP $\neq$ addition-invariant MSO on the class of finite strings, then DLIN$\subsetneq$LINH.

(b) It is straightforward to see that every fixed addition-invariant MLFP-sentence can be evaluated in a finite string in time polynomial in the size of the string.

In what follows we will show that for every level k there is a string-language L_k that is

(i) hard (with respect to PTIME-reductions) for the kth level Σ_k^P of the polynomial time hierarchy, and
(ii) definable by an addition-invariant MSO-formula.

Now, if addition-invariant MLFP = addition-invariant MSO on the class of finite strings, then L_k is definable by an addition-invariant MLFP-formula. But then, L_k is decidable in polynomial time. Since L_k is hard for Σ_k^P, this then implies that Σ_k^P is contained in PTIME, for every $k \in \mathbb{N}$, i.e., PH $=$ PTIME.

For every $k \in \mathbb{N}$, we will choose L_k to be a suitable encoding of the satisfiability problem for *quantified Boolean formulas* with k alternations of quantifiers. For the precise definition we need some notation.

For every $i \geqslant 1$ let $V_i\{x_v^{(i)} v \geqslant 1\}$ be a set of Boolean variables. We write CNF(k) for the set of all Boolean formulas over the variables $V_1 \cup \cdots \cup V_k$ in *conjunctive normal form*. An *assignment* A_i to V_i is a mapping $A_i : V_i \to \{0, 1\}$. For a formula $\Phi \in \text{CNF}(k)$ we write $\exists A_1 \forall A_2 \cdots Q_k A_k\ (\Phi = 1)$ as an abbreviation for "there exists an assignment A_1 to V_1 such that for all assignments A_2 to V_2 … such that under the assignments $A_1, \ldots, A_k$ the Boolean formula Φ is satisfied". From results of Stockmeyer [29] it follows that the problem

$$\text{QBF-CNF}(k) := \{\Phi \in \text{CNF}(k) : \exists A_1 \forall A_2 \cdots Q_k A_k\ (\Phi = 1)\}$$

is complete for the kth level Σ_k^P of the polynomial time hierarchy.

For every $\Phi \in \text{CNF}(k)$ we define a string w_Φ over the alphabet $\mathbb{A} := \{\texttt{C}, \texttt{V}, \texttt{p}, \texttt{n}, \texttt{-}\}$ in such a way that the string-language

$$L_k := \{w_\Phi : \Phi \in \text{QBF-CNF}(k)\}$$

is definable by an addition-invariant MSO formula and QBF-CNF(k) is polynomial time reducible to L_k (i.e., L_k is Σ_k^P-hard).

For the precise definition of w_Φ let $\Phi = \bigwedge_{j=1}^{n} C_j$ be a formula in CNF(k), where the C_j are *clauses*, i.e., disjunctions of unnegated or negated variables. W.l.o.g., no variable occurs both negated *and* unnegated in the same clause C_j.

For every $i \leqslant k$ let $W_i := W_i^{(\Phi)}$ be the set of all V_i-variables that occur in Φ. W.l.o.g., $W_i = \{x_1^{(i)}, \ldots, x_{s_i}^{(i)}\}$, for some $s_i = s_i^{(\Phi)} \geqslant 0$. For every $j \leqslant n$ and $i \leqslant k$ let $u_{(\Phi,j,i)}$ be the $\{\texttt{p}, \texttt{n}, \texttt{-}\}$-string $b_1 \cdots b_{s_i}$ such that, for every $v \in \{1, \ldots, s_i\}$,

$$b_v := \begin{cases} \texttt{p} & \text{if variable } x_v^{(i)} \text{ occurs unnegated in clause } C_j, \\ \texttt{n} & \text{if variable } x_v^{(i)} \text{ occurs negated in clause } C_j, \\ \texttt{-} & \text{if variable } x_v^{(i)} \text{ does not occur in clause } C_j \text{ at all.} \end{cases}$$

We define

$$w_{(\Phi,j)} := \texttt{V}\ u_{(\Phi,j,1)}\ \texttt{V}\ u_{(\Phi,j,2)}\ \cdots\ \texttt{V}\ u_{(\Phi,j,k)}\ \texttt{V}$$

and

$$w_\Phi := \mathtt{C}\ w_{(\Phi,1)}\ \mathtt{C}\ w_{(\Phi,2)}\ \cdots\ \mathtt{C}\ w_{(\Phi,n)}\ \mathtt{C}.$$

It should be clear that the mapping $f : \mathrm{CNF}(k) \to \mathbb{A}^+$ with $f(\Phi) := w_\Phi$ (for every $\Phi \in \mathrm{CNF}(k)$) is a polynomial time reduction from QBF-CNF(k) to the string-language L_k.

All that remains to show is that L_k is definable by an addition-invariant MSO-formula. First, let us construct an MSO-formula ψ_k that is satisfied by a string $w \in \mathbb{A}^+$ if, and only if, there is a $\Phi \in \mathrm{CNF}(k)$ such that $w = w_\Phi$. The formula ψ_k has to check that

(1) The string starts and ends with the letter C, and between any two occurrences of the letter C (between which no C occurs), there are exactly $k+1$ letters V, the first of which is directly right to the first C and the last of which is directly left to the second C.
(2) For all positions x and y that carry the letter C, and for all $i \leqslant k$, the following is true: if x' is the position that carries the ith letter V right to x, and y' is the position that carries the ith letter V right to y, then the number of positions between x' and the next occurrence of the letter V to the right of x' is exactly the same as the number of positions between y' and the next occurrence of the letter V to the right of y'.

Using the formula $\varphi_\#$ from Lemma 5.4, this can easily be formalized by an addition-invariant MSO($\tau_{\mathbb{A}} \cup \{\oplus\}$)-sentence ψ_k.

Next, we construct an MSO-sentence χ_k which, for every $\Phi \in \mathrm{CNF}(k)$, is satisfied by the string w_Φ if, and only if, $\Phi \in$ QBF-CNF(k), i.e, $\exists A_1 \forall A_2 \cdots Q_k A_k\ (\Phi = 1)$. To this end, we represent an assignment $A_i : W_i^{(\Phi)} \to \{0, 1\}$ by a set B_i of positions of w_Φ as follows: For all v with $x_v^{(i)} \in W_i^{(\Phi)}$, the set B_i contains

- *all* {p, n, -}-positions of w_Φ that are associated with the variable $x_v^{(i)}$, if $A_i\big(x_v^{(i)}\big) = 1$,
- *none* of the {p, n, -}-positions of w_Φ that are associated with the variable $x_v^{(i)}$, if $A_i\big(x_v^{(i)}\big) = 0$.

Using Lemma 6.2 and Lemma 5.4, it is not difficult to find an MSO($\tau_{\mathbb{A}} \cup \{\oplus\}$)-formula $\alpha_i(B)$ which ensures for an underlying string $w = w_\Phi$ and a set B of positions in w, that B represents an assignment $A_i : W_i^{(\Phi)} \to \{0, 1\}$. The formula $\alpha_i(B)$ just has to check that whenever x and y are positions that carry the ith occurrences of the letter V to the left of a C, then the following is true: if $x + z$ and $y + z$ are the next positions to the right of x and y, respectively, that carry the letter V, then we have for every u with $0 < u < z$ that $x + u \in B \iff y + u \in B$.

It is straightforward to construct a formula $\beta(B_1, \ldots, B_k)$ which, provided that $B_1, \ldots, B_k$ represent assignments in the way indicated above, expresses that $B_1, \ldots, B_k$ is a *satisfying* assignment for the Boolean formula Φ. The MSO-formula β just needs to express that between any two occurrences x and x' of the letter C there is an occurrence y of the letter V such that to the right of y (but to the left of the next occurrence of the letter V) there is a position z such that the following is true: position z either carries the letter p and $z \in B_i$ (where $i \in \{1, \ldots, k\}$ is such that y is the ith occurrence of the letter V to the right of x), or position z carries the letter n and $z \notin B_i$.

Now, we choose χ_k to be the MSO($\tau_{\mathbb{A}} \cup \{\oplus\}$)-sentence

$$\chi_k := \exists B_1\, \forall B_2\, \cdots\, Q_k\, B_k\ \bigwedge_{i=1}^{k} \alpha_i(B_i) \wedge \beta(B_1, \ldots, B_k)\,.$$

Here, $Q_k = \exists$ if k is odd, and $Q_k = \forall$ if k is even. It should be obvious that for every $\Phi \in \mathrm{CNF}(k)$ we have that

$$w_\Phi \models \chi_k \quad \iff \quad \exists A_1 \forall A_2 \cdots Q_k A_k\ (\Phi = 1)\,.$$

Altogether, we obtain that the MSO($\tau_{\mathbb{A}} \cup \{\oplus\}$)-sentence

$$\varphi_k := \psi_k \wedge \chi_k$$

is an addition-invariant MSO-sentence that defines the Σ_k^P-hard string-language $L_k := \{w_\Phi : \Phi \in \text{QBF-CNF}(k)\}$. This finally completes the proof of Theorem 6.4. $\square$

Note that the above proof also shows the following:

Corollary 6.5. (a) *If* MLFP $\neq$ MSO *on the class* $\mathcal{C}_{\mathbb{A},+} := \{\langle \underline{w}, +\rangle : w \in \mathbb{A}^+\}$ *of finite strings with addition*, *then* DLIN $\neq$ LINH.

(b) *If* MLFP $=$ MSO *on the class* $\mathcal{C}_{\mathbb{A},+}$, *then* PH $=$ PTIME.

7. Conclusion

The main results of the present paper are: (1) that MLFP can express graph properties beyond any fixed level of the monadic second-order quantifier alternation hierarchy, (2) that *addition-invariant* MLFP can express at least all string-problems that belong to the linear time complexity class DLIN, and (3) that settling the question whether addition-invariant MLFP has the same expressive power as addition-invariant MSO on finite strings would solve open problems in complexity theory.

Many interesting aspects of MLFP remain to be further investigated, for example:

- Is there a natural complexity class that is *exactly* captured by MLFP(+) on strings with built-in addition (analogous to the known result that MSO(+) exactly captures the linear time hierarchy LINH)? A promising candidate might be the time-space complexity class PTIME&LINSPACE of problems solvable by deterministic polynomial time, linear space bounded Turing machines.
- Is there a hierarchy within MLFP with respect to the alternation of *least* and *greatest* fixed point quantifiers? I.e., does MLFP have a hierarchy analogous to Bradfield's modal μ-calculus alternation hierarchy [2]? Note that every level of this MLFP alternation hierarchy is closed under first-order quantification. Therefore, the alternation hierarchy of MLFP might be viewed as a "deterministic" analogue of the *closed monadic hierarchy* of [1] rather than as an analogue of the monadic second-order quantifier alternation hierarchy of [23].
- Investigate the parameterized complexity of the model checking problem for MLFP on various classes of finite structures. E.g., is the model checking problem for MLFP fixed parameter tractable on the class of planar graphs? Partial answers to this question have been obtained by Lindell [20].
- Do Theorem 3.9 and Corollary 3.2 still hold when replacing MLFP with the modal μ-calculus?

Acknowledgements

I want to thank Martin Grohe for valuable discussions on the subject of this paper. Furthermore, I thank the anonymous referees for their helpful comments. In particular, one referee's remarks considerably simplified the presentation of Section 4.

References

[1] M. Ajtai, R. Fagin, L. Stockmeyer, The closure of Monadic NP, J. Comput. System Sci. 60 (3) (2000) 660–716 (Journal version of STOC'98 paper).
[2] J. Bradfield, The modal μ-calculus alternation hierarchy is strict, Theoret. Comput. Sci. 195 (2) (1998) 133–153 (Journal version of CONCUR'96 paper).
[3] A. Dawar, A restricted second order logic for finite structures, Inform. Comput. 143 (1998) 154–174 (Journal version of LCC'94 paper).
[4] A. Durand, C. Lautemann, M. More, Counting results in weak formalisms, Technical Report 1998-14, Université de Caen, France, 1998.
[5] A. Durand, M. More, Nonerasing, counting, and majority over the linear time hierarchy, Inform. and Comput. 174 (2002) 132–142.
[6] H.-D. Ebbinghaus, J. Flum, Finite Model Theory, second ed., Springer, Berlin, 1999.
[7] J. Flum, M. Frick, M. Grohe, Query evaluation via tree-decompositions, J. ACM 49 (6) (2002) 716–752 (Journal version of ICDT'01 paper).
[8] G. Gottlob, C. Koch, Monadic datalog and the expressive power of web information extraction languages, J. ACM 51 (1) (2004) 74–113 (Journal version of PODS'02 paper).
[9] E. Grädel, Model checking games, in: Proc. of WOLLIC 02, Electronic Notes in Theoretical Computer Science, Vol. 67. Elsevier, Amsterdam, 2002.
[10] E. Grandjean, Invariance properties of RAMs and linear time, Comput. Complexity 4 (1994) 62–106.
[11] E. Grandjean, Linear time algorithms and NP-complete problems, SIAM J. Comput. 23 (3) (1994) 573–597.
[12] E. Grandjean, Sorting, linear time, and the satisfiability problem, Ann. Math. Artif. Intell. 16 (1996) 183–236.

[13] E. Grandjean, F. Olive, Monadic logical definability of nondeterministic linear time, Comput. Complexity 7 (1) (1998) 54–97 (Journal version of CSL'94 paper).
[14] E. Grandjean, T. Schwentick, Machine-independent characterizations and complete problems for deterministic linear time, SIAM J. Comput. 32 (1) (2002) 196–230.
[15] M. Grohe, The structure of fixed-point logics, Ph.D. Thesis, Albert-Ludwigs Universität Freiburg, Germany, 1994.
[16] M. Grohe, N. Schweikardt, Comparing the succinctness of monadic query languages over finite trees, RAIRO Inf. Theor. Appl. 38 (2004) 343–373.
[17] M. Grohe, T. Schwentick, Locality of order-invariant first-order formulas, ACM Trans. Comput. Logic 1 (2000) 112–130 (Journal version of MFCS'98 paper).
[18] N. Immerman, Descriptive Complexity, Springer, Berlin, 1999.
[19] R.M. Karp, Reducibility among combinatorial problems, in: IBM Symposium 1972, Complexity of Computers Computations, Plenum Press, New York, 1972.
[20] S. Lindell, Linear-time algorithms for monadic logic, Short presentation at the 18th IEEE Symposium on Logic in Computer Science (LICS'03), 2003.
[21] J.F. Lynch, Complexity classes and theories of finite models, Math. Systems Theory 15 (1982) 127–144.
[22] J.F. Lynch, The quantifier structure of sentences that characterize nondeterministic time complexity, Comput. Complexity 2 (1992) 40–66.
[23] O. Matz, N. Schweikardt, W. Thomas, The monadic quantifier alternation hierarchy over grids and graphs, Inform. and Comput. 179 (2) (2002) 356–383.
[24] M. More, F. Olive, Rudimentary languages and second-order logic, Technical Report 1996-1, Université de Caen, France, 1996.
[25] B. Rossman, Successor-invariance in the finite, in: Proc. 18th IEEE Symp. on Logic in Computer Science (LICS'03), 2003.
[26] N. Schweikardt, Arithmetic, first-order logic, and counting quantifiers, ACM Trans. Comput. Logic 6(3) (2005) 634–671.
[27] N. Schweikardt, On the expressive power of monadic least fixed point logic, in: Proc. 31st Internat. Colloq. on Automata, Languages and Programming (ICALP'04), Lecture Notes in Computer Science, vol. 3142, Springer, Berlin, 2004, pp.1123–1135.
[28] T. Schwentick, Descriptive complexity, lower bounds and linear time, in: Proc. 12th Internat. Workshop on Computer Science Logic (CSL'98), Lecture Notes in Computer Science, Vol. 1584, Springer, Berlin, 1998, pp. 9–28 (Invited paper).
[29] L.J. Stockmeyer, The polynomial-time hierarchy, Theoret. Comput. Sci. 3 (1) (1976) 1–22.
[30] W. Thomas, Languages, automata, and logic, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Vol. 3, Springer, Berlin, 1996.