

Nondeterministic stack register machines*

P. Clote[†]

Abstract

For integer $k \geq 0$, let $\text{SRM}(n^{O(1)}, k)$ denote the collection of relations computable by a stack register machine with stack registers bounded by a polynomial $p(n)$ in the input n , and work registers bounded by k . Let $\text{NSRM}(n^{O(1)}, k)$ denote the analogous class accepted by nondeterministic stack register machines. In this paper, nondeterminism is shown to provide no additional power. Specifically,

$$\begin{aligned}\text{NSRM}(n^{O(1)}, 0) &= \text{SRM}(n^{O(1)}, 0) \\ \text{NSRM}(n^{O(1)}, 1) &= \text{SRM}(n^{O(1)}, 1) \\ \text{NSRM}(n^{O(1)}, k) &= \text{SRM}(n^{O(1)}, k), \text{ for } k \geq 4 \\ \text{SRM}(n^{O(1)}, k) &= \text{ALINTIME}, \text{ for } k \geq 4.\end{aligned}$$

1 Introduction

In a recently rediscovered letter from K. Gödel to J. von Neumann dated March 1956,¹ Gödel raised a problem about length of proofs, which has since been seen to be equivalent to the $P = NP$ question.

Around the same time, a few other logicians began investigating other problems in what was later defined to be the field of computational complexity. In 1955, Asser [1] asked whether the complement of a spectrum is a spectrum (by work of Jones-Selman [24] equivalent to $NEXP = co - NEXP$) and in 1956, Grzegorzcyk [16] defined a hierarchy \mathcal{E}^n , $n \geq 0$, within the primitive recursive functions, and asked whether \mathcal{E}_*^0 is properly contained in \mathcal{E}_*^2 .

Grzegorzcyk defined \mathcal{E}^n , for $n \geq 0$, as the smallest class of functions containing the projections, the principal functions f_0, \dots, f_n , and closed under composition and bounded recursion, where f_0 is the successor function, f_1 is addition, f_2 is multiplication, f_3 is exponentiation, etc. In that paper, he showed that

*The results of this paper were presented at the spring meeting of the Association for Symbolic Logic, March 19–22, 1992 [9].

[†]Part of this research supported by NSF CCR-9102896, INT-8914569.

Address: Institut für Informatik, Ludwig-Maximilians-Universität München, Theresienstr. 39, D-80333 München, Germany. clote@informatik.uni-muenchen.de

¹See [10] for a copy of the letter (in German), along with a translation, historical notes, and recent contributions to Gödel's NP -complete problem about proof length.

$\cup \mathcal{E}^n$ is the class of primitive recursive functions, that \mathcal{E}^3 is the class of Kalmar-Csillig elementary functions, that $\mathcal{E}^n \subset \mathcal{E}^{n+1}$ for $n \geq 0$, and that $\mathcal{E}_*^n \subset \mathcal{E}_*^{n+1}$ for $n \geq 2$, where \mathcal{C}_* denotes the class of relations whose characteristic function belongs to \mathcal{C} .

Though Grzegorzczuk's $\mathcal{E}_*^0 \subset \mathcal{E}_*^2$ question is still open, several significant results have shed light on the problem. In 1961, R. Ritchie [31] showed that \mathcal{E}_*^2 equals deterministic linear space.² It follows from work of Bennett [5] and Wrathall [37] that the linear time hierarchy LTH is contained in \mathcal{E}_*^0 . Thus $\mathcal{E}_*^0 \subset \mathcal{E}_*^2$ implies that $\text{LTH} \subset \text{LINSPEACE}$.

In [4], A. Bel'tyukov introduced the stack register machine SRM, a new variant of the successor random access machine having stack registers t_0, \dots, t_k and work register r . Bel'tyukov showed that sets belonging to LINSPEACE [resp. LTH] are exactly those sets computed on a SRM where all registers contain numbers bounded by a polynomial in the input³ [resp. additionally where the work register is at all times empty]. Bel'tyukov used his machine model to prove that $\mathcal{E}_*^0(\beta_s(x)) = \mathcal{E}_*^0$, for $\beta_s(x) = x + x^{1-1/s}$, and other results shedding a bit of light on Grzegorzczuk's question.⁴

In [27], J. Paris and A. Wilkie studied stack register machines whose work registers contain no number larger than k , for $k = 1, 2, 3$. In [28], W. Handley, J. Paris and A. Wilkie related the concept of *counting modulo a finite group* with SRM. These results are summarized later.

In this paper, we extend the stack register machine model in two directions: nondeterminism is introduced, and different branching test instructions are added. Using group theoretic and complexity theoretic techniques, we prove essentially that nondeterminism does not add any computational power.

With appropriate test conditions and bounds, stack register machines can be used to characterize a number of complexity classes related to a bounded version of Kleene's arithmetic hierarchy (i.e. defined by alternations of certain kinds of bounded quantifiers). Examples are the polynomial time hierarchy PH, the linear time hierarchy LH, the logtime hierarchy LTH, and Presburger sets (definable by first order formulas in the signature $\{0, +, =\}$), etc.

In particular, this paper studies a hierarchy of complexity classes between PH and PSPACE that are characterized using stack register machines by varying a natural parameter: the *work register*. Although it is a trivial observation that the polynomial time hierarchy is closed under nondeterminism, it is not obvious that the classes in this new hierarchy between PH and PSPACE are closed under nondeterminism. One of the principal contributions of this paper is to show

²Work of Cobham [14] and Ritchie [31] spawned a host of machine independent characterizations of complexity classes, including PTIME [14], PSPACE [35], LINSPEACE [31], LOGSPACE [25], AC^0 , AC^k , NC^k , ALOGTIME [13], $AC^0(2)$, $AC^0(6)$, TC^0 [12].

³i.e. numbers whose length is linear in the length of the input.

⁴ $\mathcal{E}^0(f)$ is the smallest class of functions containing 0, f , the successor, the projections and closed under composition and bounded recursion (see [4]). Bel'tyukov's result should be contrasted with the easily shown fact that $\mathcal{E}_*^0(2x) = \mathcal{E}_*^1$.

that levels 0, 1, and $k \geq 4$ are closed under nondeterminism. (This paper leaves open the question of closure of levels 2 and 3. These levels have since been shown to be closed under nondeterminism by an elegant analysis of semigroups by W.G. Handley [17].)

Our paper does not address the issue of polynomial time bounded algorithms with limited nondeterminism, but should be seen as a contribution towards revitalizing Grzegorzczuk's $\mathcal{E}_*^0 \subset \mathcal{E}_*^2$ question. Our results additionally present a unifying scheme for viewing classes between LTH and LINSPEACE, and between PH and PSPACE.

In a certain sense, as seen later, adding an extra stack register to a stack register machine corresponds to adding a bounded quantifier, hence to nondeterminism. However, it is not obvious that nondeterminism in the sense of ambiguity in the branching instructions contributes no additional power to the model. Thus another principal contribution of this paper could be viewed as presenting a better understanding of how to program such machines.

In forthcoming work, using these techniques we characterize low-level uniform boolean circuit classes AC^0 , $AC^0(2)$, $AC^0(6)$, TC^0 and ALOGTIME via modified stack register machines.⁵ Despite advances by Razborov [30] and Smolensky [32], it is still unknown whether $AC^0(6) = TC^0$, or even if $AC^0(6) = \text{ALOGTIME}$. Perhaps our forthcoming characterizations of these classes in terms of stack register machines may facilitate an approach to these problems.

2 Preliminaries

In [4], Bel'tyukov introduced stack register machines and characterized the class RUD of rudimentary sets as well as all levels $(\mathcal{E}^n)_*$ of the Grzegorzczuk hierarchy via stack register machines. Of particular relevance for this paper are his characterizations of the linear time hierarchy LTH and linear space, as well as similar type characterizations of extensions of LTH under counting due to Paris-Handley-Wilkie [28]. In this paper, we generalize stack register machines to arbitrary test conditions in branching instructions.

Definition 1 Let \mathcal{F} be a set of functions. We define a stack register machine over \mathcal{F} , denoted $\text{SRM}[\mathcal{F}]$, to be a machine consisting of a finite number of input registers x_1, \dots, x_m , a finite number of stack registers t_0, \dots, t_k , and a single work register r . Each register may hold an arbitrary integer. A $\text{SRM}[\mathcal{F}]$ has four types of instructions:

- (i) if $f(z_1, \dots, z_n) = z_{n+1}$ then a else b
- (ii) $t_i := t_i + 1$

⁵ AC^0 is the collection of languages recognized by a logtime uniform, unbounded fan-in boolean circuit family of constant depth and polynomial size. $AC^0(m)$ additionally admits gates which count modulo m . TC^0 is the collection of languages recognized by a logtime uniform family of constant depth, polynomial size threshold gate circuits. See [34] for more on uniform circuit classes.

- (iii) $r := z$
- (iv) halt.

In the *branching* instructions (i), f is an n -ary function in \mathcal{F} , each z_i and z is contained in $\{x_1, \dots, x_m, t_0, \dots, t_k, r, 0\}$, and a, b are line numbers.⁶ The z_i 's and z need not be distinct. For instance, $f(r, r, t_0, t_1, t_0)$ is allowed.

When executed, if the test condition is true, then line a is next executed, else line b . In the *incremental* instructions (ii), $0 \leq i \leq k$ and the effect is to increment register t_i and simultaneously set to 0 all t_j for $j < i$. In the *save* instruction (iii), $z \in \{x_1, \dots, x_m, t_0, \dots, t_k, 0\}$. A *program* is a finite sequence of instructions I_1, \dots, I_p with line numbers $1, \dots, p$, where I_p is the halt instruction, and where, for each $i \leq k$, *there is at most one incremental instruction for t_i* . By adding another stack register, we may suppose WLOG that I_1 is the incremental instruction $t_0 := t_0 + 1$. In the execution of a program, the input registers contain the input and are never modified, the stack and work registers are initialized to 0, and except for branching instructions, sequential flow of control is followed. The input x_1, \dots, x_m is *accepted* by a $\text{SRM}[\mathcal{F}]$ if there is a halting computation where the *top* stack t_k (i.e. stack with largest index) contains 0. If M is a $\text{SRM}[\mathcal{F}]$, then $L(M)$ is the relation accepted by M , and satisfies $L(M) = \{(x_1, \dots, x_m) \in \mathbf{N}^m : M \text{ accepts } (x_1, \dots, x_m)\}$.

Definition 2 If f, g are functions, then $\text{SRM}[\mathcal{F}](f, g)$ is the collection of all relations $L \subseteq \cup_{m \in \mathbf{N}} \mathbf{N}^m$ accepted by a $\text{SRM}[\mathcal{F}]$ such that

- (i) on every input x_1, \dots, x_m the machine eventually halts,
- (ii) on every input x_1, \dots, x_m at all times during the computation, it is the case that $t_i \leq f(\max\{x_1, \dots, x_m\})$ for $i \leq k$, and $r \leq g(\max\{x_1, \dots, x_m\})$.

Note that condition (ii) concerns bounds on the integers themselves, rather than their lengths, in contrast to most complexity measures. If Φ, Ψ are classes of unary functions, then $\text{SRM}[\mathcal{F}](\Phi, \Psi) = \cup_{g \in \Phi, h \in \Psi} \text{SRM}[\mathcal{F}](g, h)$.

Example 3 The following is an example of a stack register machine program P to compute $x_1 \leq x_2$. Machine M has input registers x_1, x_2 , stack registers t_0, t_1 and work register r . An unconditional “goto i ” statement is an abbreviation for “if $t_0 = t_0$ then i else i ”.

1. if $t_0 = x_2$ then 2 else 3
2. if $t_0 = x_1$ then 7 else 6
3. if $t_0 = x_1$ then 7 else 4

⁶In [4, 28] and other previous work, \mathcal{F} was taken to be the set of multivariate polynomials with non-negative integer coefficients, i.e. built up by composition from constants $n \in \mathbf{N}$, the projection functions $i_k^n(x_1, \dots, x_n) = x_k$, addition and multiplication. For this reason, later in the paper we write $\text{SRM}(g, h)$ to abbreviate $\text{SRM}[+, \times](g, h)$.

4. $t_0 := t_0 + 1$
5. goto 1
6. $t_1 := t_1 + 1$
7. halt

It is easy to check that M halts with empty stack t_1 iff $x_1 \leq x_2$. Thus the inequality relation belongs to $\text{SRM}[id](n, 0)$, where $id(x) = x$ is the identity function.

Definition 4 A $\text{SRM}[\mathcal{F}]$ M can compute an m -ary partial function f whose value on x_1, \dots, x_m is the value of the top stack register (i.e., the register with the largest index) at the end of a computation (if the computation terminates). The class of total functions computed by a $\text{SRM}[\mathcal{F}]$ with stack register bound g and work register bound h is denoted by $\text{FSRM}[\mathcal{F}](g, h)$. Formally, $\text{FSRM}[\mathcal{F}](g, h)$ is the collection of all functions $f : \mathbf{N}^m \rightarrow \mathbf{N}$ such that

- (i) on every input x_1, \dots, x_m the machine eventually halts and the value of the top stack is $f(x_1, \dots, x_m)$,
- (ii) on every input x_1, \dots, x_m at all times during the computation, it is the case that $t_i \leq g(\max\{x_1, \dots, x_m\})$ for $i \leq k$, and $r \leq h(\max\{x_1, \dots, x_m\})$.

Remark 5 If M is a stack register machine which computes the value of a function $f(x_1, \dots, x_n)$ in *any* fixed stack register t_j within stack register bound g and work register bound h , then there exists a stack register machine M' which computes the value $f(x_1, \dots, x_n)$ in the top stack register t_k within the same bounds g, h . This is easy to see: with $t_k = 0$ compute $f(\vec{x})$ in t_j , and if value is t_k halt; else increment t_k , recompute $f(\vec{x})$ in t_j , and if this value is t_k halt; else etc.

Lemma 6 For any unary functions g, h and any class \mathcal{F} of functions, $\text{FSRM}[\mathcal{F}](g, h)$ is closed under composition.

Proof Suppose that $u, v_1, \dots, v_m \in \text{FSRM}[\mathcal{F}](g, h)$ and that

$$f(x_1, \dots, x_n) = u(v_1(x_1, \dots, x_n), \dots, v_m(x_1, \dots, x_n)).$$

Let $M_u, M_{v_1}, \dots, M_{v_m}$ be stack register machines having respectively μ, ν_1, \dots, ν_m many stack registers and computing respectively u, v_1, \dots, v_m within stack register bound g and work register bound h .

Let $M'_u, M'_{v_1}, \dots, M'_{v_m}$ and M' be machines having $\mu + \nu_1 + \dots + \nu_m$ many stack registers. By an obvious modification of the instructions of M_{v_i} , machine M'_{v_i} uses only stack registers with index k where

$$\mu + \nu_1 + \dots + \nu_{m-i} < k \leq \mu + \nu_1 + \dots + \nu_{m-i+1}$$

and computes the value of $v_i(x_1, \dots, x_n)$ in stack register with index $\mu + \nu_1 + \dots + \nu_{m-i+1}$. Machine M' applies the instructions of $M'_{v_1}, \dots, M'_{v_m}$ successively (where the halt instruction for the program of M'_{v_i} is replaced by the first instruction of $M'_{v_{i+1}}$). By an obvious modification of the program for M_u (replacing the input x_i by the value in stack register with index $\mu + \nu_1 + \dots + \nu_{m-i+1}$), M' then computes $u(v_1(\vec{x}), \dots, v_m(\vec{x}))$ in the stack register with index μ . Now M' can compute $u(v_1(\vec{x}), \dots, v_m(\vec{x}))$ in stack register t_μ and by the previous remark, a stack register machine M can be found which computes $u(v_1(\vec{x}), \dots, v_m(\vec{x}))$ in its top stack register and respects the same stack register bound g and work register bound h . ■

We use the multitape Turing machine (with random access if the time bound is sublinear) and assume familiarity with nondeterministic and alternating machines (see [8]) and with the linear time hierarchy LTH (see [37]) and the polynomial time hierarchy PH.

Definition 7 For language $A \subseteq \{0, 1\}^*$, let

$$\begin{aligned}
\sum_0^{L,A} &= DLINTIME^A \\
\sum_{i+1}^{L,A} &= \{B \subseteq \{0, 1\}^* : (\exists C \in \sum_i^{L,A})(B = M^C) \\
&\quad \text{for some nondeterministic linear time bounded} \\
&\quad \text{Turing machine } M \text{ with oracle } C\} \\
LTH^A &= \bigcup_{i \in \mathbb{N}} \sum_i^{L,A} \\
LTH &= LHT^\emptyset \\
LTH(\mathcal{C}) &= \bigcap \{\mathcal{D} : \mathcal{C} \subseteq \mathcal{D} \wedge (\forall A \in \mathcal{D})(LTH^A \subseteq \mathcal{D})\} \\
\sum_0^{P,A} &= PTIME^A \\
\sum_{i+1}^{P,A} &= \{B \subseteq \{0, 1\}^* : (\exists C \in \sum_i^{P,A})(B = M^C) \\
&\quad \text{for some nondeterministic polytime bounded} \\
&\quad \text{Turing machine } M \text{ with oracle } C\} \\
PH^A &= \bigcup_{i \in \mathbb{N}} \sum_i^{P,A} \\
PH &= PH^\emptyset \\
PH(\mathcal{C}) &= \bigcap \{\mathcal{D} : \mathcal{C} \subseteq \mathcal{D} \wedge (\forall A \in \mathcal{D})(PH^A \subseteq \mathcal{D})\}.
\end{aligned}$$

Definition 8 Let θ be a first order formula. An existential bounded quantifier is of the form $(\exists x \leq y)$. A universal bounded quantifier is of the form $(\forall x \leq y)$. The interpretation of

$$(\exists x \leq y)\theta$$

is

$$(\exists x)[x \leq y \wedge \theta].$$

The interpretation of

$$(\forall x \leq y)\theta$$

is

$$(\forall x)[x \leq y \rightarrow \theta].$$

Definition 9 Let \mathcal{F} be a collection of function symbols. Then $\Delta_0(\mathcal{F})$ is the smallest class of formulas \mathcal{C} such that

- (i) $t_1 \leq t_2$ and $t_1 = t_2$ belong to \mathcal{C} , where t_1, t_2 are first order terms built from variables and function symbols of \mathcal{F}
- (ii) if θ, φ belong to \mathcal{C} then so do $\neg\theta, \theta \wedge \varphi, \theta \vee \varphi$
- (iii) if θ belongs to \mathcal{C} then so do $(\exists x \leq t)\theta, (\forall x \leq t)\theta$ for any variable x and term t built from variables and function symbols of \mathcal{F} , provided that x is not free in t .

Moreover, $\Delta_0^N(\mathcal{F})$ is the collection of predicates having $\Delta_0(\mathcal{F})$ definitions; i.e. an n -ary predicate $P \subseteq \mathbf{N}^n$ belongs to $\Delta_0^N(\mathcal{F})$ iff there is a formula $\theta(x_1, \dots, x_n) \in \Delta_0(\mathcal{F})$ having free variables x_1, \dots, x_n such that

$$P = \{(m_1, \dots, m_n) \in \mathbf{N}^n : \mathbf{N} \models \theta(m_1, \dots, m_n)\}.$$

If $\mathcal{F} = \{0, 1, +, \times\}$ then we simply write Δ_0 in place of $\Delta_0(\mathcal{F})$.⁷ If $A \subseteq \mathbf{N}^k$, then a predicate P is in $\Delta_0^{N,A}(\mathcal{F})$ if it is definable in \mathbf{N} by a $\Delta_0(\mathcal{F}')$ formula, where $\mathcal{F}' = \mathcal{F} \cup \{c_A\}$ and function symbol c_A is interpreted by the characteristic function of A . The *graph* $Gr_f(\vec{x}, y)$ of a function f satisfies $Gr_f(\vec{x}, y) \equiv f(\vec{x}) = y$. A function f is Δ_0 if its graph is Δ_0 definable and it is *polynomially bounded*; i.e. for some multivariate polynomial p , for all \vec{x} , $f(x_1, \dots, x_n) \leq p(x_1, \dots, x_n)$. The *length* of the binary representation of integer x is denoted by $|x|$. A function f has *polynomial* [resp. *linear*] growth if for some multivariate polynomial p [resp. constant c], and all \vec{x} , $|f(x_1, \dots, x_n)| \leq p(|x_1|, \dots, |x_n|)$ [resp. $|f(x_1, \dots, x_n)| \leq c \cdot \sum_{i=1}^n |x_i|$]. Clearly, polynomially bounded means the same as linear growth.

Definition 10 Let θ be a first order formula. For integer k , the *counting mod k quantifier* is of the form $(C_k x \leq y)$. The interpretation of

$$(C_k x \leq y)\theta$$

is that

$$|\{x \leq y : \theta\}| \equiv 0 \pmod{k}.$$

Definition 11 Let \mathcal{F} be a collection of function symbols. For a set M of integers, $C_M \Delta_0(\mathcal{F})$ is the smallest class \mathcal{C} of formulas satisfying (i), (ii), (iii) of definition 9 together with

⁷We use \times and \cdot interchangeably, the choice being made for typographic reasons.

- (iv) if θ belongs to \mathcal{C} then so does $(C_k x \leq t)\theta$, for any $k \in M$, any variable, and any term t built from variables and function symbols of \mathcal{F} , provided that x is not free in t .

Moreover, $C_M \Delta_0^N(\mathcal{F})$ is the collection of predicates having $C_M \Delta_0(\mathcal{F})$ definitions. We write $C_k \Delta_0(\mathcal{F})$ in place of $C_{\{k\}} \Delta_0(\mathcal{F})$. If $\mathcal{F} = \{0, 1, +, \times\}$ then we write simply $C_k \Delta_0$. As in definition 9, this definition can be relativized to $A \subseteq \mathbf{N}^m$, producing $C_k \Delta_0^{N,A}(\mathcal{F})$. Turing machines accept languages $L \subseteq \{0, 1\}^*$ and more generally k -ary relations $R \subseteq (\{0, 1\}^*)^k$. Using binary encoding an integer $x = \sum_{i=0}^n x_i \cdot 2^i$ corresponds to word $x_n \cdots x_0 \in \{0, 1\}^*$ while using dyadic encoding, an integer $x = \sum_{i=1}^n (x_i + 1) \cdot s^i$ *uniquely* corresponds to a word $x_n \cdots x_0 \in \{0, 1\}^*$. Via either encoding Turing machines can be assumed to accept sets of integers or k -ary predicates on the integers.

To extend counting to arbitrary finite groups, we introduce the following definitions.

Definition 12 A monoid is a set with a binary associative operation, denoted by \circ , sometimes called *multiplication*. Let M_k denote the monoid of all functions from $\{0, \dots, k-1\}$ into $\{0, \dots, k-1\}$. Let G be a finite monoid, whose elements are identified with $1, \dots, |G|$. If $f : \mathbf{N} \rightarrow G$, then $\bar{f} : \mathbf{N} \rightarrow G$ is defined by $\bar{f}(x) = f(0) \circ \cdots \circ f(x)$. If \mathcal{C} is a complexity class of functions, then $G(\mathcal{C})$, the *closure of \mathcal{C} under counting modulo G* , is defined to be

$$\cap \{ \mathcal{D} : \mathcal{C} \subseteq \mathcal{D} \wedge (\forall f : \mathbf{N} \rightarrow G) (Gr_f \in \mathcal{D} \rightarrow Gr_{\bar{f}} \in \mathcal{D}) \}.$$

The class \mathcal{C} is *closed under counting mod G* if $G(\mathcal{C}) = \mathcal{C}$.

Definition 13 If G is a monoid, and \mathcal{C} is a class of predicates, then $GLTH(\mathcal{C})$ is the smallest class \mathcal{D} of predicates such that

- $\mathcal{C} \subseteq \mathcal{D}$
- $G(\mathcal{D}) = \mathcal{D}$
- $LTH(\mathcal{D}) = \mathcal{D}$.

$GLTH(\emptyset)$ is denoted by $GLTH$. Similarly, $GPH(\mathcal{C})$ is the smallest class \mathcal{D} of predicates such that

- $\mathcal{C} \subseteq \mathcal{D}$
- $G(\mathcal{D}) = \mathcal{D}$
- $PH(\mathcal{D}) = \mathcal{D}$.

$GPH(\emptyset)$ is denoted by GPH .

The proof that $LTH = \Delta_0^N$ readily yields that $LTH^A = \Delta_0^{N,A}$ and hence that $GLTH = G\Delta_0^N$, the latter studied in [28].

Let I be the collection of all projection functions $i_k^n(x_1, \dots, x_n) = x_k$, for all n and $1 \leq k \leq n$. Let 0 be the constant (nullary function) with value zero, let $S(n) = n + 1$, let \max be the binary maximum function, $f_1(x) = \max(1, 2x)$, $f_2(x) = \max(2, x^2)$, $f_3(x) = 2^x$, $f_{n+4}(x) = f_{n+3}^{(x)}(1)$, where $f^{(i)}(x)$ is $f(f(\dots f(x)\dots))$ with i many occurrences of f . For $s \geq 1$, let $\beta_s(x) = \max(1, x + \lceil x^{1-1/s} \rceil)$. Let COMP be the operation of function composition, and BR be the operation of bounded recursion, allowing the definition of f from g, h, k as follows:

$$\begin{aligned} f(0, \vec{y}) &= g(\vec{y}) \\ f(x+1, \vec{y}) &= h(x, \vec{y}, f(x, \vec{y})) \end{aligned}$$

provided that $f(x, \vec{y}) \leq k(x, \vec{y})$. By $[f_1, \dots, f_m; \mathcal{O}_1, \dots, \mathcal{O}_n]$ denote the smallest class of functions containing f_1, \dots, f_m and closed under the operations $\mathcal{O}_1, \dots, \mathcal{O}_n$. Define $\mathcal{E}f$ to be $[f, 0, I, S; \text{COMP}, \text{BR}]$, $\mathcal{E}^0 = \mathcal{E}0$, and $\mathcal{E}^{n+1} = \mathcal{E}f_{n+1}$. If \mathcal{F} is a class of functions, then \mathcal{F}_* denotes the class of relations whose characteristic function belongs to \mathcal{F} . With this notation, Bel'tyukov proves the following.⁸

Theorem 14 (Bel'tyukov [4])

- (i) $\mathcal{E}f = \text{SRM}(f^{(O(1))}, f^{(O(1))})$
- (ii) $\mathcal{E}_*^2 = \mathcal{E}_*^1$ implies $\mathcal{E}_*^2 = \mathcal{E}_*^0$
- (iii) For $s \geq 1$, $\mathcal{E}_*^0 = (\mathcal{E}\beta_s)_*$
- (iv) $\text{SRM}(n^{O(1)}, 0) = \Delta_0^N$.

To situate Bel'tyukov's work, note that it is well-known that if $LTH = \text{Linspace}$ then $\mathcal{E}_*^0 = \mathcal{E}_*^2$. Modulo well-known results of C. Wrathall [37] and J. Bennett [5] equating Δ_0^N and LTH , and of R.W. Ritchie [31] equating \mathcal{E}_*^2 with Linspace , we mention the following result.

Theorem 15 (Bel'tyukov [4])

$$\begin{aligned} \text{SRM}(n^{O(1)}, 0) &= LTH \\ \text{SRM}(n^{O(1)}, n^{O(1)}) &= \text{Linspace} . \end{aligned}$$

In [27], J. Paris and A. Wilkie extend Bel'tyukov's result to characterize $\text{SRM}(n^{O(1)}, 1)$, $\text{SRM}(n^{O(1)}, 2)$ and $\text{SRM}(n^{O(1)}, 3)$, the latter two of which are surprisingly equal.

⁸Part (i) of Theorem 14 states that $\mathcal{E}f$ is the class of functions computed by a stack register machine where the registers are bounded by an *iterate* (and not a power) of the function f . In [4] Bel'tyukov actually proves in (iv) of Theorem 14 that $\text{SRM}(n^{O(1)}, 0) = \text{RUD}$, where RUD is the class of rudimentary predicates. In [5], J. Bennett proves that $\text{RUD} = \Delta_0^N$.

Theorem 16 (Paris, Wilkie [27])

$$\begin{aligned} (i) \text{ SRM}(n^{O(1)}, 1) &= C_2 \Delta_0^N \\ (ii) \text{ SRM}(n^{O(1)}, 2) &= C_6 \Delta_0^N \\ (iii) \text{ SRM}(n^{O(1)}, 3) &= C_6 \Delta_0^N. \end{aligned}$$

Letting S_n denote the full symmetric group of all permutations on n letters, Paris, Handley and Wilkie proved the following extension of Bel'tyukov's work.⁹

Theorem 17 ([28])

$$\text{SRM}(n^{O(1)}, k) = S_{k+1} \text{LTH}.$$

Definition 18 Let Q be a finite set $\{a_1, \dots, a_n\}$ of integers. A Q -SRM is a stack register machine which allows one instruction of the form

$$\left\{ \begin{array}{l} \text{if } \phi_1 \text{ then } t_i = t_i + a_1 \\ \text{if } \phi_2 \text{ then } t_i = t_i + a_2 \\ \dots \\ \text{if } \phi_n \text{ then } t_i = t_i + a_n \end{array} \right.$$

for each i and each $q \in Q$, where the ϕ_i form a partition, and each ϕ_i is a quantifier free formula built up from $0, 1, +, \times$ in the variables \vec{x}, \vec{t}, r .

Paris, Handley and Wilkie additionally proved the following.

Theorem 19 ([28])

$$\begin{aligned} (i) \{1, 2\} - \text{SRM}(n^{O(1)}, 0) &= \text{LTH} \\ (ii) \{1, n+1\} - \text{SRM}(n^{O(1)}, 0) &= \mathbf{Z}_{n!} \text{LTH} \\ (iii) \{1, 2, n+1\} - \text{SRM}(n^{O(1)}, 0) &= S_n \text{LTH} \end{aligned}$$

In [22], J. Hicks raised the question of whether nondeterminism increases the power of stack register machines. In this paper, we partially answer this question by showing that nondeterminism for stack register machines with constant bounds for the work register is no more powerful than determinism. It is important to mention that our work left open the case for work register bounds of 2 and 3. Using different techniques, W.G. Handley later solved this problem in [17].

⁹Warning. In the notation of [28], Theorem 17 is stated as $\text{Space}(k) = S_k \Delta_0^N$, where the authors there define $\text{Space}(k)$ to be the set of languages computed by a stack register machine with polynomial bounds on stack registers, and where numbers in the work register are *strictly* less than k (our notation allows $\leq k$, hence the discrepancy).

3 Presburger and Skolem Arithmetic

As a warm up to the study of nondeterminism, we illustrate Bel'tyukov's proof technique by studying Presburger and Skolem arithmetic. Sets $A \subseteq \mathbf{N}^n$ definable by first order formulas in the signature $\{0, 1, +, =\}$ [resp. $\{0, 1, \times, =\}$] are known as *Presburger* [resp. *Skolem*] sets. By quantifier elimination, such sets are equivalently definable by bounded quantifier formulas in an appropriately enriched signature. Since stack registers correspond loosely to bounded quantifiers (the proof of equivalence uses Bel'tyukov's *looping lemma* technique explained below), these new results serve as a simple introduction to the techniques we later refine to prove the main results of this paper.

Recall that $O(x) = \{f : (\exists c, d)(\forall x)(f(x) \leq c \cdot x + d)\}$.

Definition 20 A set $A \subseteq \mathbf{N}^k$ is a *Presburger set* if there is a first order formula ϕ in the signature $\{0, 1, +, =\}$ such that

$$A = \{\vec{x} \in \mathbf{N}^k : \mathbf{N} \models \phi(\vec{x})\}.$$

Lemma 21 *If A is Presburger then $A \in SRM[+](O(n), 0)$.*

Proof If $A \subseteq \mathbf{N}^n$ is Presburger, then by the well-known quantifier elimination theorem of Presburger (see p. 320 of [33] for example) A can be defined by a boolean combination of *quantifier free* formulas in the extended signature $\{0, 1, +, \leq, \text{MOD}\}$, where the ternary predicate $\text{MOD}(x, y, z)$ means $x \equiv y \pmod{z}$, and by convention we will assume that $\neg \text{MOD}(x, y, z)$ holds for $z = 0$ or $z = 1$. A $SRM[+](O(n), 0)$ program for addition $x_1 + x_2$ is given by the following.

1. if $x_1 + x_2 = t_0$ then 4 else 2
2. $t_0 := t_0 + 1$
3. goto 1
4. halt

By lemma 6 together with the program of example 3, it follows that any set defined by atomic formula $t_1(x_1, \dots, x_n) \leq t_2(x_1, \dots, x_n)$ can be computed in $SRM[+](O(n), 0)$. The predicate $\text{MOD}(x, y, z)$ can be computed by the following pseudocode, where we recall that a stack register machine *accepts* if it halts with top register 0.

```

begin
  if z=0 or z=1
    return 1
  else if x=y
    return 0
  else if x<y
    u:=x

```

```

        v:=y
    else
        u:=y
        v:=x
    end if
    % If no answer yet returned, then u<v is ensured
    w:=z
label:
    if u+w>v
        return 1
    else if u+w=v
        return 0
    else
        w:=w+z
    end if
    goto label
end

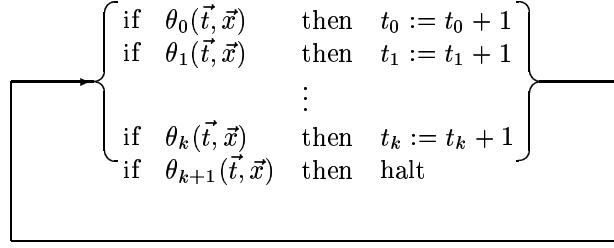
```

To implement the pseudocode on a stack register machine, input x, y, z initially appears in input registers. Letting u [resp. v] denote the top [resp. second to top] stack register, by the technique mentioned in remark 5, one can implement the assignment $u := x, v := y, w := z$, as well as the increment instruction $w := w + z$. The branching instructions can be supported in $SRM[+]$, as well as the goto instruction. The computation always terminates, and all numbers appearing at any time in the stack registers will be bounded by $x + y + z$. Thus $\text{MOD}(x, y, z) \in SRM[+](O(n), 0)$. Finally if $B, C \in SRM[+](O(n), 0)$ then it is straightforward to see that $\mathbf{N}^k - B, B \cup C, B \cap C$ are as well. By induction it follows that any set definable by a quantifier free formula in signature $\{0, 1, +, \leq, \text{MOD}\}$ belongs to $SRM[+](O(n), 0)$. ■

Lemma 22 *If $A \in SRM[+](\infty, 0)$ then A is Presburger.*

Proof Let M be a stack register machine with input registers x_1, \dots, x_m , stack registers t_1, \dots, t_k and no work register.

Claim 1 (Normalization) Machine M is equivalent to a machine M' defined by



where the θ_i are first order formulas in signature $\{0, 1, +, \leq\}$ forming a partition.

Proof of claim Let I_i be the instruction

$$t_i := t_i + 1$$

if it occurs in M 's programs. For $i \leq k, j \leq k+1$ if $t_0, \dots, t_k, x_1, \dots, x_m$ are the contents of M 's registers after executing I_i , then define formula $\phi_{i,j}(t_0, \dots, t_k, x_1, \dots, x_m)$ so that

- $j \leq k$ and the next increment instruction to be executed is I_j .
- $j = k+1$ and M will halt before performing another incremental instruction.

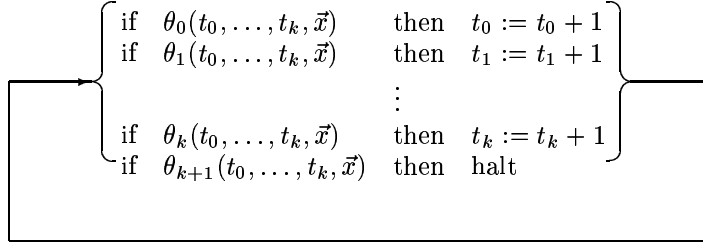
Now M 's program has a finite number ℓ of instructions. Since M 's program has no save instructions $r := t$ involving work register r , it follows that M can execute at most ℓ instructions between any two incremental instructions — otherwise M would loop forever. It is now clear that $\phi_{i,j}(\vec{t}, \vec{x})$ can be expressed as a quantifier free formula in signature $\{0, 1, +, \leq\}$.

For $j \leq k+1$ define θ_j to be

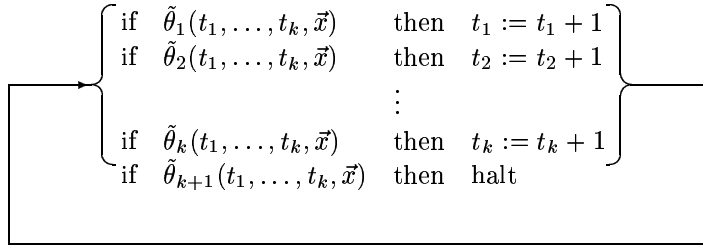
$$\bigvee_{i \leq k} (\phi_{i,j}(\vec{t}, \vec{x}) \wedge t_i \neq 0 \wedge \bigwedge_{p < i} t_p = 0) \vee (j = 0 \wedge \bigwedge_{p \leq k} t_p = 0).$$

Clearly the θ_j are quantifier free formulas in the signature $\{0, 1, +, \leq\}$ satisfying the claim. \square

Claim 2 (looping lemma) Suppose M is a machine of the form



where the θ_i are first order formulas in signature $\{0, 1, +, \leq\}$, and $k \geq 1$. Then there is an equivalent machine M' of the form



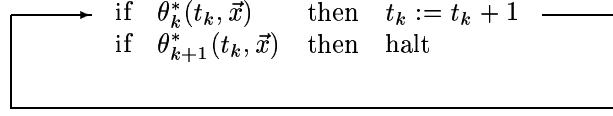
where the $\tilde{\theta}_i$ are first order formulas in signature $\{0, 1, +, \leq\}$. Note that the θ_i depend on t_0, t_1, \dots, t_k while the $\tilde{\theta}_i$ depend only on t_1, \dots, t_k . The effect of the looping lemma is to find an equivalent machine with one fewer stack register.

Proof of claim In the computation of M on input x_1, \dots, x_m suppose that M currently has $t_0 = 0$ whereas t_1, \dots, t_k have any arbitrary values. Now as long as θ_0 holds, M will increment t_0 . Since the first instant where θ_0 does not hold can be defined in a first order manner, one can dispense with stack register t_0 altogether. Formally, for $1 \leq i \leq k + 1$ define $\tilde{\theta}_i$ to be

$$\exists s \forall s' < s [\theta_0(s', t_1, \dots, t_k, \vec{x}) \wedge \neg \theta_0(s, t_1, \dots, t_k, \vec{x}) \wedge \theta_i(s, t_1, \dots, t_k, \vec{x})].$$

Clearly the $\tilde{\theta}_i$ satisfy the requirements of the claim. \square

Returning to the proof of the lemma, if $A \in SRM[+](\infty, 0)$, then let M be a stack register machine computing A . The only requirement on M is that it terminate on every input, regardless of stack register bound. By claim 1, M is equivalent to a normalized machine $M^{(1)}$. Applying claim 2 successively k times, each time eliminating the smallest indexed stack register, produces machine $M^{(k)}$ with pseudocode



Since a stack register machine accepts input \vec{x} iff the top stack t_k is empty, the set A can be defined by $\vec{x} \in A \Leftrightarrow \theta_{k+1}^*(0, \vec{x})$. ■

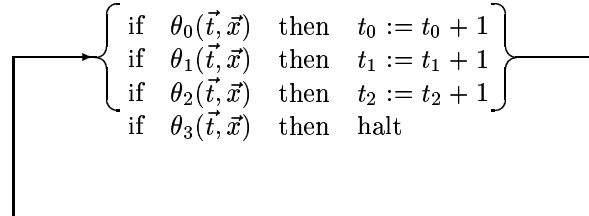
Example 23 By convention, a *SRM* program begins with the instruction $t_0 := t_0 + 1$. Modifying example 3 produces the following program M for inequality $x_1 \leq x_2$.

1. $t_0 := t_0 + 1$
2. if $t_1 = x_2$ then 3 else 4
3. if $t_1 = x_1$ then 8 else 7
4. if $t_1 = x_1$ then 8 else 5
5. $t_1 := t_1 + 1$
6. if $t_0 = t_0$ then 2 else 2
7. $t_2 := t_2 + 1$
8. halt

To present the normalization M' of M , the formulas $\phi_{i,j}(t_0, t_1, t_2, x_1, x_2)$ must be described, for $i \leq 2, j \leq 3$.

- $\phi_{0,0}(\vec{t}, \vec{x}), \phi_{1,0}(\vec{t}, \vec{x}), \phi_{2,0}(\vec{t}, \vec{x}), \phi_{2,1}(\vec{t}, \vec{x}), \phi_{2,2}(\vec{t}, \vec{x})$ are $0 \neq 0$
- $\phi_{0,1}(\vec{t}, \vec{x}), \phi_{1,1}(\vec{t}, \vec{x})$ are $t_1 \neq x_2 \wedge t_1 \neq x_1$
- $\phi_{0,2}(\vec{t}, \vec{x}), \phi_{1,2}(\vec{t}, \vec{x})$ are $t_1 = x_2 \wedge t_1 \neq x_1$
- $\phi_{0,3}(\vec{t}, \vec{x}), \phi_{1,3}(\vec{t}, \vec{x})$ are $(t_1 \neq x_2 \wedge t_1 = x_1) \vee (t_1 = x_2 \wedge t_1 = x_1)$, which simplifies to $t_1 = x_1$.
- $\phi_{2,3}(\vec{t}, \vec{x})$ is $0 = 0$.

Thus M' is of the form



where

- $\theta_0(\vec{t}, \vec{x})$ is

$$(\phi_{0,0} \wedge t_0 \neq 0) \vee (\phi_{1,0} \wedge t_1 \neq 0) \vee (\phi_{2,0} \wedge t_2 \neq 0) \vee \wedge_{i \leq 2} t_i = 0.$$

Since $\phi_{0,0}$, $\phi_{1,0}$, and $\phi_{2,0}$, are $0 \neq 0$, the previous formula is equivalent to

$$(t_0 = 0 \wedge t_1 = 0 \wedge t_2 = 0).$$

- $\theta_1(\vec{t}, \vec{x})$ is

$$(\phi_{0,1} \wedge t_0 \neq 0) \vee (\phi_{1,1} \wedge t_1 \neq 0 \wedge t_0 = 0) \vee (\phi_{2,1} \wedge t_2 \neq 0 \wedge t_1 = 0 \wedge t_0 = 0).$$

Recall that $\phi_{2,1}$ is $0 \neq 0$, so the third disjunct can be dropped. This gives

$$(t_1 \neq x_1 \wedge t_1 \neq x_2 \wedge t_0 \neq 0) \vee (t_1 \neq x_1 \wedge t_1 \neq x_2 \wedge t_1 \neq 0 \wedge t_0 = 0)$$

or equivalently

$$(t_1 \neq x_1 \wedge t_1 \neq x_2) \wedge (t_0 \neq 0 \vee t_1 \neq 0).$$

- $\theta_2(\vec{t}, \vec{x})$ is

$$(\phi_{0,2} \wedge t_0 \neq 0) \vee (\phi_{1,2} \wedge t_1 \neq 0 \wedge t_0 = 0) \vee (\phi_{2,2} \wedge t_2 \neq 0 \wedge t_1 = 0 \wedge t_0 = 0).$$

Recall that $\phi_{2,2}$ is $0 \neq 0$, so the third disjunct can be dropped. This gives

$$(t_1 = x_2 \wedge t_1 \neq x_1 \wedge t_0 \neq 0) \vee (t_1 = x_2 \wedge t_1 \neq x_1 \wedge t_1 \neq 0 \wedge t_0 = 0)$$

which simplifies to

$$(t_1 = x_2 \wedge t_1 \neq x_1) \wedge (t_0 \neq 0 \vee t_1 \neq 0).$$

- $\theta_3(\vec{t}, \vec{x})$ is

$$(\phi_{0,3} \wedge t_0 \neq 0) \vee (\phi_{1,3} \wedge t_1 \neq 0 \wedge t_0 = 0) \vee (\phi_{2,3} \wedge t_2 \neq 0 \wedge t_1 = 0 \wedge t_0 = 0).$$

Recall that $\phi_{2,3}$ is $0 = 0$ and so can be dropped. We have

$$(t_1 = x_1 \wedge t_0 \neq 0) \vee (t_1 = x_1 \wedge t_1 \neq 0 \wedge t_0 = 0) \vee (t_2 \neq 0 \wedge t_1 = 0 \wedge t_0 = 0)$$

hence

$$(t_1 = x_1 \wedge (t_0 \neq 0 \vee t_1 \neq 0)) \vee (t_2 \neq 0 \wedge t_1 = 0 \wedge t_0 = 0).$$

In words, what this algorithm says is first to increment t_0 , then to increment t_1 until it equals x_1 or x_2 , then to increment t_2 if $x_2 < x_1$, while $t_2 = 0$ if $x_1 \leq x_2$, as desired.

From the previous two lemmas, the following theorem is immediate.

Theorem 24 $Presburger = SRM[+](\infty, 0) = SRM[+](O(n), 0)$.

Definition 25 A set $A \subseteq (\mathbf{N}^+)^k$ of k -tuples of positive integers is a *Skolem set* if there is a first order formula ϕ in the signature $\{0, 1, \times, =\}$ such that

$$A = \{\vec{x} \in (\mathbf{N}^+)^k : \mathbf{N} \models \phi(\vec{x})\}.$$

Skolem, and later Cegielski [7] proved a quantifier elimination theorem for the theory of integer multiplication, thus showing the theory to be decidable. Complexity analysis of Presburger and Skolem arithmetic have been carried out. See Smorynski's delightful text [33] for a discussion of various first order theories of arithmetic.

We have seen that

$$\begin{aligned} \text{PRES} &= SRM[+](O(n), 0) \\ &= SRM[+](n^{O(1)}, 0) \\ &= SRM[+](\infty, 0). \end{aligned}$$

In view of the formal similarities between Presburger and Skolem arithmetic, it is natural to conjecture that

$$\begin{aligned} \text{SKOLEM} &= SRM[\times](O(n), 0) \\ &= SRM[\times](n^{O(1)}, 0) \\ &= SRM[\times](\infty, 0). \end{aligned}$$

As will be shown, this is not true — stack register machines implicitly admit the successor function $s(x)$ and inequality \leq , which with multiplication allows them to compute sets more complicated than Skolem sets.

Lemma 26 *The divisibility predicate $x|y$ is computable in $SRM[\times](n, 0)$.*

Proof Assume first that $x, y \geq 1$. Machine M has input registers x, y and stack registers t_0, t_1 . At termination, M will contain 0 in the top register t_1 if $x|y$, else 1.

1. $t_0 := t_0 + 1$
2. if $x \cdot t_0 = y$ then 5 else 3
3. if $t_0 = y$ then 4 else 1
4. $t_1 := t_1 + 1$
5. halt

It is easy to add several instructions to handle the case where x or y is 0. ■

Definition 27 Integers x, y are *coprime*, denoted $x \perp y$, if the greatest common denominator $gcd(x, y)$ of x, y is 1.

Lemma 28 *The coprimality relation $x \perp y$ is computable in $\text{SRM}[\times](O(n), 0)$.*

Proof Assume first that $2 \leq x, y$ and that $\neg(x|y)$ and $\neg(y|x)$. Consider the following pseudocode.

```

 $t_0 := 2$ 
while  $t_0 < \min(x, y)$  do
  if  $t_0|x$  and  $t_0|y$  then
    halt
  else
     $t_0 := t_0 + 1$ 
  endif
end while
if  $t_0 = \min(x, y)$  then
   $t_1 := 1$ 
end if

```

A stack register machine to compute

$t_0 = \min(x, y)$

is given by the following stack register machine.

1. if $t_0 = x$ then 5 else 2
2. if $t_0 = y$ then 5 else 3
3. $t_0 := t_0 + 1$
4. goto 1
5. halt

By composing a modification of this program with a modified version of Example 3 for \leq and with a program arising from Lemma 26, it is an uninteresting and tedious exercise to describe a stack register machine program for the above pseudocode with no work register and where stack registers are bounded by $O(n)$, where $n = \max(x, y)$. Using the program in lemma 26, one can additionally handle the special cases where $x = 0, 1, y = 0, 1, x|y$, or $y|x$. ■

Theorem 29

$$\text{SRM}[\times](n^{O(1)}, 0) = \text{SRM}[+, \times](n^{O(1)}, 0) = \text{LTH}$$

Proof In [36], A. Woods proved that the predicates $x + y = z$ and $x \cdot y = z$ are definable by bounded quantifier formulas involving only \leq and \perp . By lemma 28, the predicate \perp is computable in $\text{SRM}[\times](n, 0)$, so by Woods' theorem (since bounded quantification is easy to simulate using stack register machines) the predicate $x + y = z$ is computable in $\text{SRM}[\times](n, 0)$. ■

Let REC denote the collection of recursive predicates.

Theorem 30

$$\text{SRM}[\times](\infty, 0) = \text{REC}.$$

Proof By Woods' theorem, the recursively enumerable relations are those which can be defined over the integers by a block of unbounded existential quantifiers followed by a bounded quantifier formula in the signature $\{\perp, \leq\}$. Recursive relations are those A such that A and its complement \bar{A} are so definable. Since coprimality is computable in $\text{SRM}[\times](n, 0)$, the result is immediate. ■

Note that our definition of $\text{SRM}(\infty, 0)$ requires that the stack register machine M terminate on all inputs, where stack registers are bounded by some arbitrary function g , and the work register is bounded by 0. It is for this reason that $\text{SRM}[\times](\infty, 0)$ defines the recursive predicates, rather than the arithmetic predicates.

The remainder of the paper investigates nondeterministic stack register machines and in part uses nondeterministic analogues of the normalization lemma and looping lemma illustrated in theorem 24. Hopefully this illustration will aid the reader in following our presentation of results on nondeterminism.

4 Nondeterminism

Definition 31 A nondeterministic stack register machine over \mathcal{F} , denoted $\text{NSRM}[\mathcal{F}]$, is defined by replacing the branching instruction (i) by the following instruction (i') in definition 1:

$$(i') \text{ if } f(z_1, \dots, z_n) = z_{n+1} \text{ then } a_1, \dots, a_r \text{ else } b_1, \dots, b_s.$$

Here, the a_i, b_j are line numbers.

Upon execution, if the condition $f(z_1, \dots, z_n) = z_{n+1}$ holds when tested, then the next instruction to be executed may be any one of a_1, \dots, a_r . If the condition does not hold, then any one of b_1, \dots, b_s may be executed. An input x_1, \dots, x_m is accepted by a $\text{NSRM}[\mathcal{F}]$ with program I_1, \dots, I_p if there is a sequence s_1, \dots, s_q of instructions such that $s_1 = 1$ (the line number of the first instruction, which by convention is $t_0 := t_0 + 1$), $s_q = p$ (by convention I_p is the halt instruction), and for every $i < q$ if s_i is the line number of a branching instruction and $f(z_1, \dots, z_n) = z_{n+1}$ holds then $s_{i+1} \in \{a_1, \dots, a_r\}$, else $s_{i+1} \in \{b_1, \dots, b_s\}$, where $a_1, \dots, a_r, b_1, \dots, b_s$ correspond to the line numbers mentioned in the nondeterministic branching instruction.

Definition 32 A *configuration* of an $\text{SRM}[\mathcal{F}]$ is an $m + k + 3$ -tuple $(i, x_1, \dots, x_m, t_0, \dots, t_k, r)$, where i is the current instruction to be executed, x_1, \dots, x_m is the input, t_0, \dots, t_x the contents of the stack registers, and r the contents of the work register. The *initial configuration* is

$(1, x_1, \dots, x_m, 0, \dots, 0, 0)$. A *halting configuration* is $(\ell, x_1, \dots, x_m, t_0, \dots, t_k, r)$, where ℓ is the number of instructions in program P . It is not difficult to show that all branching instructions may be assumed to be of the form

if $f(z_1, \dots, z_n) = z_{n+1}$ then a, b else c, d .

If I_i is the branching instruction

if $f(\vec{u}) = v$ then a, b else c, d

then

$(i, \vec{x}, \vec{t}, r) \vdash_M (a, \vec{x}, \vec{t}, r)$

and

$(i, \vec{x}, \vec{t}, r) \vdash_M (b, \vec{x}, \vec{t}, r)$

provided that $f(\vec{u}) = v$, where $f \in \mathcal{F}$ and \vec{u}, v are among $\vec{x}, \vec{t}, r, 0$, while

$(i, \vec{x}, \vec{t}, r) \vdash_M (c, \vec{x}, \vec{t}, r)$

and

$(i, \vec{x}, \vec{t}, r) \vdash_M (d, \vec{x}, \vec{t}, r)$

if $f(\vec{u}) \neq v$. If I_i is the incremental instruction

$t_j := t_j + 1$

then

$(i, \vec{x}, t_0, \dots, t_{j-1}, t_j, t_{j+1}, \dots, t_k, r) \vdash_M (i + 1, \vec{x}, 0, \dots, 0, t_j + 1, t_{j+1}, \dots, t_k, r)$.

If I_i is the save instruction

$r := z$

for z in $\vec{x}, \vec{t}, r, 0$, then

$(i, \vec{x}, \vec{t}, r) \vdash_M (i + 1, \vec{x}, \vec{t}, z)$.

The reflexive transitive closure of \vdash_M is noted by \vdash_M^* .

Definition 33 The *computation tree* $T_{\vec{x}}$ of a $SRM[\mathcal{F}]$ on input \vec{x} is a tree T such that

- the initial configuration $C_0 = (1, \vec{x}, 0, \dots, 0, 0) \in T$ and is the *root* of T
- if $C \in T$, $C \vdash_M D$, then $D \in T$, provided that for no configuration C_i , $i < n$, occurring on the path $C_0 \vdash_M C_1 \vdash_M \dots \vdash_M C_{n-1} \vdash_M C_n = C$ in T from the root C_0 to C is it the case that $C = C_i$.

If $C_0 \vdash_M \dots \vdash_M C_i \vdash \dots \vdash C_n = C$ and $C_i = C_n$ for some $i < n$, where C_0 is the root of T and $C_0, \dots, C_n \in T$ then C is called a *repeating configuration*. Note that leaves of T are either halting or repeating configurations.

Definition 34 A relation $L \subseteq \mathbb{N}^m$ is accepted by a $\text{NSRM}[\mathcal{F}]$ machine M with stack register bound f and work register bound g , denoted $L \in \text{NSRM}[\mathcal{F}](f, g)$ if L consists of those inputs x_1, \dots, x_m accepted by M and

- (i) for all x_1, \dots, x_m , the computation tree $T_{\vec{x}}$ is finite,
- (ii) for all x_1, \dots, x_m , and for any configuration $(i, \vec{x}, \vec{t}, r) \in T_{\vec{x}}$ it is the case that

$$t_i \leq f(\max\{x_1, \dots, x_m\}), \text{ for } i \leq k$$

and

$$r \leq g(\max\{x_1, \dots, x_m\}).$$

Let P be a program for a $\text{NSRM}[\mathcal{F}]$ with instructions numbered $1, \dots, \ell$. As before, we assume that the first instruction is $t_0 := t_0 + 1$ and the ℓ -th instruction is *halt*. Suppose that P has ℓ_0 branching instructions, the line numbers of which form the set

$$\text{BRANCH}_P = \{n_1, \dots, n_{\ell_0}\}$$

and ℓ_1 incremental instructions, the line numbers of which form the set

$$\text{INCR}_P = \{m_1, \dots, m_{\ell_1}\}.$$

Define the function

$$\text{test}_P(i) = \begin{cases} \Theta & \text{if } i \in \text{BRANCH}_P, \text{ and} \\ & i \text{ is the line number of} \\ & \text{“if } \Theta \text{ then } a, b \text{ else } c, d\text{”} \\ x_1 = x_1 & \text{else} \end{cases}$$

which maps the set $\{1, \dots, \ell\}$ of line numbers into the set of formulas with variables $x_1, \dots, x_m, t_0, \dots, t_k, r$. Define $\text{stack}_P : \text{INCR}_P \rightarrow \{1, \dots, \ell\}$ by $\text{stack}_P(i) = j$ if line i is of the form “ $t_j := t_j + 1$ ”. Suppose that P has ℓ_2 Save instructions, which form the set

$$\text{SAVE}_P = \{r_1, \dots, r_{\ell_2}\}.$$

A *setting* of P is a mapping

$$\sigma : \text{BRANCH}_P \rightarrow \{0, 1\}$$

where $\sigma(i) = 0$ (resp. 1) corresponds to the assertion that $\text{test}_P(i)$ holds (resp. does not hold).

Clearly, there are at most 2^{ℓ} settings for a program P of ℓ lines. For each setting σ of P , we associate a digraph G_σ whose vertices $\{1, \dots, \ell\}$ form the set of line numbers of P , and whose (directed) edges are of the form (i, j) where $i \in \text{BRANCH}_P$ is the line number of a branching instruction of the form

$$\text{if } \Theta \text{ then } a, b \text{ else } c, d$$

and

$$\begin{aligned}\sigma(i) = 0 &\Rightarrow j \in \{a, b\} \\ \sigma(i) = 1 &\Rightarrow j \in \{c, d\}\end{aligned}$$

Note that there may be cycles in the digraph G_σ , and that G_σ is not strongly connected since program P contains at least one incremental instruction $t_0 := t_0 + 1$. A vertex i of G_σ is *terminal* if there is no j in $\{1, \dots, \ell\}$ for which (i, j) is a directed edge of G_σ . All terminal vertices of G_σ are line numbers of either incremental, save or halt instructions. The predicate $\text{ACC}_{P,\sigma}(i, j)$, meaning that instruction j is accessible from instruction i within setting σ is defined by

$$\text{ACC}_{P,\sigma}(i, j) \equiv \text{“there exists a (directed) path in } G_\sigma \text{ from } i \text{ to } j\text{.”}$$

Since P has ℓ lines, G_σ has ℓ vertices, so there is a path from i to j iff there is a path from i to j of length at most $\ell - 1$. It follows that $\text{ACC}_{P,\sigma}$ is expressible by a boolean combination of atomic formulas in the variables $x_1, \dots, x_m, t_0, \dots, t_k, r$. Note that all intermediate instructions in such a path from i to j are branching instructions which do not change the values in the stacks or work registers.

Definition 35 A *multivalued function* is an $n+1$ -ary predicate H satisfying

$$(\forall x_1, \dots, x_n)(\exists y)H(x_1, \dots, x_n, y)$$

though not necessarily

$$(\forall x_1, \dots, x_n)(\exists! y)H(x_1, \dots, x_n, y).$$

By abuse of notation, $H(\vec{x}) = y$ and $y \in H(\vec{x})$ may both be written in place of $H(\vec{x}, y)$.

If H is a multivalued function and r is the work register of a stack register machine, then the assignment statement

$$r := H(\vec{x})$$

is an abbreviation for the (nondeterministic) assignment

$$(\exists y)(H(\vec{x}, y) \wedge r := y).$$

This convention slightly simplifies notation below.

Lemma 36 (Normalization Lemma) *Let M be a NSRM[\mathcal{F}] machine with input registers x_1, \dots, x_m , stack registers t_0, \dots, t_k , work register r , and program*

P consisting of instructions numbered $1, \dots, \ell$. Then there is an equivalent program P' with one instruction and which is of the form

$$1 : \left\{ \begin{array}{l} \text{if } \phi_0 \text{ then } t_0 := t_0 + 1; r := H_0(\vec{x}, \vec{t}, r); \\ \quad \text{goto } 1; \\ \quad \vdots \\ \text{if } \phi_k \text{ then } t_k := t_k + 1; r := H_k(\vec{x}, \vec{t}, r); \\ \quad \text{goto } 1; \\ \text{if } \phi_{k+1} \text{ then halt.} \end{array} \right.$$

where the ϕ_i are exhaustive but not necessarily exclusive, and the H_i are multi-functions. Moreover, the ϕ_i and H_i are boolean combinations of atomic formulas in the variables \vec{x}, \vec{t}, r .

Proof For $\lambda < \ell$, define $\text{SEQSETTING}_{P,\lambda}$ to be

$$\{F : F \text{ maps } \{0, \dots, \lambda\} \text{ into } \{\sigma : \sigma \text{ a setting of } P\}\}$$

and $\text{SEQLINENOS}_{P,\lambda}$ to be

$$\{G : G \text{ maps } \{0, \dots, \lambda\} \text{ into } \{1, \dots, \ell\}\}$$

and define

$$\text{modify}_P(\Theta, i) = \begin{cases} \Theta & \text{if } i \notin \text{SAVE}_P \\ \Theta[r/z] & \text{if } i \in \text{SAVE}_P, \\ & \text{line } i \text{ of the form " } r := z \text{".} \end{cases}$$

Here $\Theta[r/z]$ is the result of substituting r by z in Θ . Let $\Theta^0 \equiv \Theta$ and $\Theta^1 \equiv \neg\Theta$. For $i \leq k$, let I_i be the line number of the (unique) incremental instruction $t_i := t_i + 1$. Additionally, let I_{k+1} be the line number of the (unique) halt instruction.

Suppose that program P has ℓ lines. For $F \in \text{SEQSETTING}_{P,\lambda}$ and $G \in \text{SEQLINENOS}_{P,\lambda}$, let $\text{PATH}(i, j, x_1, \dots, x_m, t_0, \dots, t_k, r, F, G, \lambda)$ be defined by

$$\begin{aligned} & (\bigwedge_{0 < \alpha < \lambda} G(\alpha) \in \text{SAVE}_P) \wedge G(0) = I_i \wedge G(\lambda) = I_j \wedge \\ & (\bigwedge_{0 \leq \alpha < \lambda-1} \text{ACC}_{P,F(\alpha)}(G(\alpha) + 1, G(\alpha + 1))) \wedge \\ & (\bigwedge_{0 \leq \alpha \leq \lambda} \bigwedge_{\beta \in \text{BRANCH}_P} \text{modify}_P(\text{test}_P(\beta)^{(F(\alpha))(\beta)}, G(\alpha))). \end{aligned}$$

The free variables of PATH include \vec{x}, \vec{t}, r because they can appear in test and modify . The intent is that $\text{PATH}_P(i, j, \vec{x}, \vec{t}, r, F, G, \lambda)$ holds iff F is a sequence of $\lambda + 1 \leq \ell$ settings, G is a sequence of $\lambda + 1 \leq \ell$ line numbers, all but the first and last of which are save instructions, between any two of which there is a sequence of only branching instructions (expressed using $\text{ACC}_{P,\sigma}$). Furthermore, in $\text{PATH}_P(i, j, x_1, \dots, x_m, t_0, \dots, t_k, r, F, G, \lambda)$ the values of t_0, \dots, t_k, r form the contents of the stack and work registers immediately after execution of the

incremental instruction $t_i := t_i + 1$. Branching instructions do not modify the contents of the stack and work registers. The formula

$$\text{modify}_P(\text{test}_P(\beta)^{(F(\alpha))(\beta)}, G(\alpha))$$

results from taking the test condition Θ (of the form $f(z_1, \dots, z_n) = z_{n+1}$) in instruction i of the form

if Θ then a, b else c, d

and replacing the contents of the registers according to instruction α and then taking either the resulting formula or its negation, according to the setting $F(\alpha)$. Note that $G(0), G(\lambda)$ are incremental instructions, and that for $0 < \alpha < \lambda$, $G(\alpha)$ is a save instruction. Hence, for $\alpha \leq \lambda$, $G(\alpha)$ is a terminal vertex of the digraph G_σ and that the next instruction executed by machine M after $G(\alpha)$ is $G(\alpha) + 1$, by sequential flow of control. The condition

$$\bigwedge_{0 \leq \alpha \leq \lambda} \bigwedge_{\beta \in \text{BRANCH}_P} \text{modify}_P(\text{test}_P(\beta)^{(F(\alpha))(\beta)}, G(\alpha))$$

ensures that the sequence of settings and line numbers correspond to a correct computation path.

For $0 \leq j \leq k + 1$, let $\phi_j(x_1, \dots, x_m, t_0, \dots, t_k, r)$ be defined by

$$\begin{aligned} & \bigvee_{\lambda < \ell} \bigvee_{F \in \text{SEQSETTING}_{P,\lambda}} \bigvee_{G \in \text{SEQLINENOS}_{P,\lambda}} \\ & \bigvee_{0 \leq i \leq k} [\text{PATH}_P(i, j, x_1, \dots, x_m, t_0, \dots, t_k, r, F, G, \lambda) \wedge \\ & (t_i \neq 0 \wedge \bigwedge_{i' < i} t_{i'} = 0) \vee (j = 0 \wedge \bigwedge_{i \leq k} t_i = 0)] \end{aligned}$$

For $j \leq k + 1$, let $H_j(x_1, \dots, x_m, t_0, \dots, t_k, r, y)$ be

$$\begin{aligned} & \bigvee_{\lambda < \ell} \bigvee_{F \in \text{SEQSETTING}_{P,\lambda}} \bigvee_{G \in \text{SEQLINENOS}_{P,\lambda}} \\ & \bigvee_{0 \leq i \leq k} [\text{PATH}_P(i, j, x_1, \dots, x_m, t_0, \dots, t_k, r, F, G, \lambda) \\ & \wedge G(\lambda - 1) \text{ is the line number of an instruction of the form } "r := t" \\ & \wedge y = t]. \end{aligned}$$

With ϕ_j, H_j thus defined, it is left to the reader to verify the conclusion of the lemma.

Since the number of lines ℓ of the program P is fixed and independent of the input, it is not difficult to see that $\mathcal{F} \in \text{SEQSETTING}_{P,\lambda}$ and $G \in \text{SEQLINENOS}_{P,\lambda}$ can be rewritten in a fashion using only boolean combinations of atomic formulas. ■

Definition 37 Let G be a finite monoid. If $R \subseteq \mathbf{N} \times G$ is a multivalued function, then $\bar{R} \subseteq \mathbf{N} \times G$ is the relation satisfying $\bar{R}(x, y)$ iff

$$(\exists f : \{0, \dots, x\} \rightarrow G)(\forall i \leq x)[R(i, f(i)) \wedge f(0) \circ \dots \circ f(x) = y].$$

The closure $NG(\mathcal{C})$ of a class \mathcal{C} under *nondeterministic counting mod G* is the smallest class \mathcal{D} containing \mathcal{C} satisfying

$$(\forall R \subseteq \mathbf{N} \times G)[R \text{ multivalued function} \wedge R \in \mathcal{D} \rightarrow \overline{R} \in \mathcal{D}].$$

Definition 38 *NGLTH* is the smallest class \mathcal{C} satisfying

- (1) $\phi \in \mathcal{C}$
- (2) $NG(\mathcal{C}) = \mathcal{C}$
- (3) $D \in \mathcal{C} \Rightarrow LTH(D) \in \mathcal{C}$

In the following, by *ALINTIME*, we mean alternating linear time on a Turing machine (see [8]).

Theorem 39 *Let G be any finite monoid. Then NGLTH \subseteq ALINTIME.*

Proof Clearly, *ALINTIME* satisfies conditions 1,3 of definition 38. It suffices to show that *ALINTIME* is closed under nondeterministic counting mod G , in order to prove that *NGLTH* \subseteq *ALINTIME*. To this end, let $R \subseteq \mathbf{N} \times G$ be a multi-function belonging to *ALINTIME*. Assume *WLOG* that $x \geq 1$. Recall that $\overline{R}(x, y)$ holds iff there exists a function $f : \{0, \dots, x\} \rightarrow G$ for which $(\forall i \leq x)R(i, f(i))$ and $\prod_{i < x} f(i) = y$.

We will show that \overline{R} belongs to *ALINTIME* by defining a 2-person game which on input $(x, y) \in \mathbf{N} \times G$ consists of $\lceil \log_2 x \rceil$ rounds (a round consists of a move by player 1 followed by a move of player 2). Player 1 will win the game if $\overline{R}(x-1, y)$ holds, otherwise player 2 will win. The entire game can be performed in linear time and hence corresponds to a computation in alternating linear time.

Let $y_0 = y$ and define the interval $I_0 = \{0, 1, \dots, 2^{\lceil \log_2 x \rceil} - 1\}$, and let MAX denote the maximum element of I_0 . Player 1 begins by claiming that $\overline{R}(x-1, y_0)$ holds and by furnishing two elements $y'_1, y''_1 \in G$, whose product is y_0 . The idea is that if $\overline{R}(x-1, y)$ holds exactly if there exists a function $f : I_0 \rightarrow G$ for which

$$(\forall i < 2^{\lceil \log_2 x \rceil})[(i < x \rightarrow R(i, f(i))) \wedge (x \leq i \rightarrow f(i) = id_G)]$$

and $\prod_{i \in I_0} f(i) = y$ where id_G is the identity element of G . Player 1 claims that $\prod_{i \in I'_1} = y'_1$ and $\prod_{i \in I''_1} = y''_1$, where I'_1, I''_1 are the equal sized left and right halves of I_0 . Player 2 challenges either the first or second claim of player 1.

In the i -th play, $0 \leq i < \text{MAX}$, player 1 does the following:

- (i) gives two elements $y'_{i+1}, y''_{i+1} \in G$ such that $y_i = y'_{i+1} \circ y''_{i+1}$,
- (ii) claims that $(\exists f'_{i+1} : I'_{i+1} \rightarrow G)$ such that

$$(\forall i \in I'_{i+1})[i < x \rightarrow R(i, f'(i)) \wedge x \leq i \rightarrow f'(i) = id_G]$$

and that

$$\prod_{i \in I'_{i+1}} f'(i) = y'_{i+1}$$

where I'_{i+1} is the left half of interval I_i ,

(iii) claims that $(\exists f''_{i+1} : I''_{i+1} \rightarrow G)$ such that

$$(\forall i \in I''_{i+1})[i < x \rightarrow R(i, f''(i)) \wedge x \leq i \rightarrow f''(i) = id_G]$$

and that

$$\prod_{i \in I''_{i+1}} f''(i) = y''_{i+1}$$

where I''_{i+1} is the right half of interval I_i .

As response in the i -th round for $i < \text{MAX}$, player 2 either challenges claim (i), or challenges claim (ii) and sets $I_{i+1} = I'_{i+1}, y_{i+1} = y'_{i+1}$ or challenges claim (iii) and sets $I_{i+1} = I''_{i+1}, y_{i+1} = y''_{i+1}$. Player 1 loses during the i -th round of the game, for $i < \text{MAX}$, if one of the following holds:

(i) $y'_{i+1} \notin G$ or $y''_{i+1} \notin G$,

(ii) $y_i \neq y'_{i+1} \circ y''_{i+1}$.

Player 1 loses at the end of the game if

(iii)

$$[I_{\text{MAX}} = \{u\} \wedge ((x \leq u \wedge y_{\text{MAX}} \neq id_G) \vee (u < x \wedge \neg R(u, y_{\text{MAX}}))].$$

Player 1 wins if he/she does not lose. Clearly $\overline{R}(x-1, y)$ holds iff player 1 has a winning strategy in the above game. Conditions (i),(ii) can be verified in constant time. At each round, player 2 writes a single bit, by the end of the game writing out value u satisfying $I_{\text{MAX}} = \{u\}$ at the end of the game (0 for left interval, 1 for right interval). By hypothesis on R , given $u < x$ and y_{MAX} in G , it can be verified in ALINTIME whether $\neg R(u, y_{\text{MAX}})$ holds. Finally, and perhaps most importantly, Bennett [5] has shown that the graph of exponentiation (i.e. the ternary relation $a^b = c$) belongs to LTH . Now one can existentially guess values $b, c < 2 \cdot x$ and in LTH verify that $2^b = c$ and that c is the least power of 2 greater than or equal to x . It follows that $\overline{R} \in \text{ALINTIME}$.

■

Definition 40 If G is a group, then the *commutator* of elements $a, b \in G$ is the element $aba^{-1}b^{-1}$. The *commutator subgroup* of G , denoted by G' , is the subgroup of G generated by all the commutators of G .

Definition 41 Let G be a group with identity element e . A subgroup H of G is a *normal subgroup* of G if for every $h \in H, g \in G$ it is the case that $ghg^{-1} \in H$. The group G is *solvable* if there is a finite chain $G = G_0 \supset G_1 \supset \dots \supset G_n = \{e\}$ such that each G_{i+1} is a normal subgroup of G_i , and each factor group G_i/G_{i+1} is abelian.

Definition 42 Let G be a finite group, $G^{(1)} = G'$, the commutator subgroup of G , $G^{(2)} = (G^{(1)})'$, the commutator subgroup of $G^{(1)}$, etc. Let $[G, G]$ denote $G^{(n)}$ such that $G = G_0 \supset G_1 \supset \dots \supset G_n$ and $(G^{(n)})' = G^{(n)}$.

Fact 43 If G is a finite group, then G is solvable iff $[G, G] = \{e\}$.

Proof Lemma 5.10 on p. 211 of [21]. \square

By the previous fact, if G is a finite unsolvable group (such as S_k , for $k \geq 5$) then $[G, G]$ is its own commutator subgroup; i.e. for $a_i, b_i \in [G, G]$, the finite product $\prod_{i=1}^n a_i b_i a_i^{-1} b_i^{-1} \in [G, G]$. This key observation was used by D. Barrington [3] in his proof that (uniform) bounded width polynomial size branching programs compute exactly the functions in ALOGTIME .

Theorem 44 *Let G be any finite, unsolvable group. Then $\text{ALINTIME} = \text{GLTH} = \text{NGLTH}$.*

Proof Clearly $\text{GLTH} \subseteq \text{NGLTH}$ which by the previous result is contained in ALINTIME . For the converse direction, suppose that M is an alternating linear time bounded Turing machine which accepts a language $A \subseteq \Sigma^*$. Thus $M = \langle Q, \Sigma, q_{start}, \delta_0, \delta_1, k \rangle$, where

- (i) Q is a finite set of states, partitioned by $Q = Q_\vee \cup Q_\wedge \cup \{acc, rej\}$,
- (ii) Σ is a finite input alphabet, Γ is a finite alphabet for the k work tapes,
- (iii) $q_{start} \in Q_\vee$,
- (iv) δ_0, δ_1 are two transition functions satisfying

$$\begin{aligned} \delta_0, \delta_1 : Q_\vee \times \Gamma^k &\rightarrow Q_\wedge \times \Gamma^k \times \{L, R\}^{k+1} \\ \delta_0, \delta_1 : Q_\wedge \times \Gamma^k &\rightarrow (Q_\vee \cup \{acc, rej\}) \times \Gamma \times \{L, R\}^{k+1}. \end{aligned}$$

Without loss of generality, we assume that M terminates on every input x after exactly $c \cdot |x|$ steps, where c is even. As well, M can be assumed to existentially guess the input bit in Σ and then universally verify its correctness at the end of the computation. There is a linear time bounded (deterministic) Turing machine M' which, given input x, s where s encodes a ‘‘choice sequence’’ $(s(0), \dots, s(cn-1))$, with $s(i) \in \{0, 1\}$ for $i < cn$, and simulates M by applying the transition function $\delta_{s(i)}$ at the i -th step. Clearly $M(x)$ accepts iff

$$(\exists y_0 \leq 1)(\forall y_1 \leq 1) \dots (\forall y_{cn-1} \leq 1) M'(x, \langle y_0, \dots, y_{cn-1} \rangle) \text{ accepts.}$$

The computation tree $T(x)$ for the computation of the alternating machine M on input x thus gives rise to a fully balanced formula (circuit which is a tree), where leaves are labeled by acc, rej and consists of alternating \wedge, \vee levels with \vee at the root. By construction, the depth of $T(x)$ is thus $c|x|$, where c is even. Except for the case $x = 0$, $T(x)$ has $2^{c|x|} \leq 2^{\log_2(x^c)} \cdot 2^c = 2^c \cdot x^c$ many leaves. Let $\ell = \lceil \log_4(|G|) \rceil$ and $m = 4^\ell$. Thus m is the smallest power of 4 greater than or equal to $|G|$. Since G is non-solvable, $[G, G] \neq \{e\}$. Without loss of generality, assume $G = [G, G]$, hence every element $g \in G$ is a product $\prod_{i < m} a_i b_i a_i^{-1} b_i^{-1}$ of commutators (some of the a_i, b_i may be the identity e). We fix a look-up table which for each $g \in G$, associates a sequence $\langle a_i b_i a_i^{-1} b_i^{-1} : i < m \rangle$ of commutators whose product is g . Using Barrington’s argument [3], given an

element $g \in G$ and a tree $T(x)$ arising from a computation of M on x , we describe a word $w_{T(x)}$ in the elements of G such that

$$w_{T(x)} = \begin{cases} e & \text{if } M \text{ accepts } x \\ g & \text{else.} \end{cases}$$

By induction on the depth of node A in $T(x)$, we describe a word $w_A(g)$ such that

$$w_A(g) \equiv \begin{cases} e & \text{if node } A \text{ evaluates to } 1 \\ g & \text{else.} \end{cases}$$

Case 1. The depth is 0 and A is a leaf of $T(x)$.

In this case,

$$w_A(g) = \begin{cases} e & \text{if } A \text{ is an accepting node} \\ g & \text{else.} \end{cases}$$

Case 2. The depth $d + 1$ is even, $A = (B \vee C)$

In this case,

$$\begin{aligned} w_A(g) &= w_{B \vee C}(g) \\ &= \prod_{i < m} [w_A(a_i)w_B(b_i)w_A(a_i^{-1})w_B(b_i^{-1})] \cdot e^{12m(16m)^d} \end{aligned}$$

Case 3. The depth $d + 1$ is odd, $A = (B \wedge C)$

In this case,

$$w_A(g) = w_{B \wedge C}(g)$$

which is

$$\begin{aligned} &\prod_{i < m} [w_A(b_{m-i}^{-1})b_{m-i} \cdot e^{(16m)^{d-1}} \cdot w_B(a_{m-i}^{-1})a_{m-i} \cdot e^{(16m)^{d-1}} \\ &\quad w_A(b_{m-i})b_{m-i}^{-1} \cdot e^{(16m)^{d-1}} \cdot w_B(a_{m-i})a_{m-i}^{-1} \cdot e^{(16m)^{d-1}}] \cdot \\ &\quad e^{8m(16m)^{d-4m}} \cdot \prod_{i < m} a_i b_i a_i^{-1} b_i^{-1} \end{aligned}$$

It is easy to verify by induction on depth that

$$w_A(g) = \begin{cases} e & \text{if } A \text{ evaluates to } 1 \\ g & \text{else.} \end{cases}$$

Also, by induction on depth, one verifies that if A is a node of depth d in $T(x)$, then $|w_A(g)| = (16m)^d = (4^2 \cdot 4^\ell)^d = 4^{d(\ell+2)}$. When $d = c|x|$, a calculation shows that $|w_{T(x)}(g)| \leq 4x^{2c(\ell+2)}$.

Claim. There exists a Δ_0 function H such that for all $y < |w_{T(x)}(g)|$, $H(x, y)$ is the y -th element of the word $w_{T(x)}(g)$.

Proof Before proving the claim, note that from the claim,

$$x \in L(M) \iff w_{T(x)}(g) = e \iff \overline{H}(x, 4x^{2c(\ell+2)}) = e$$

which proves the theorem. From the linear time machine M' previously described, there is a linear time computable F such that for all x and all $i < 2^c \cdot x^c$, we have

$$F(x, i) = \begin{cases} 0 & \text{if } i\text{-th leaf of } T(x) \text{ is } rej \\ 1 & \text{if } i\text{-th leaf of } T(x) \text{ is } acc \\ 2 & \text{if } i \geq 4x^{2c(\ell+2)} \end{cases}$$

Consider the following algorithm, which on input x, y begins at the root of $T(x)$, keeps track of the current depth and sequence of bits (0 for left, 1 for right), and progresses towards the leaves in determining the element of G corresponding to the y -th symbol in the word $w_{T(x)}(g)$. This algorithm uses $O(|x|)$ space, and repeatedly looks up the new value of σ in the fixed look-up table of products of commutators, and then decides which of cases 1,2,3 applies.

Input. $g \in G, x \geq 1, y \geq 1$.

Output. w_y if $w_{T(x)} = w_1, \dots, w_{2^c|x|}$, for $1 \leq y \leq 2^c|x|$.

In the following, comments begin with % to the end of the line. We will assume that $1 \leq y \leq 2^c|x|$, otherwise output "out of range".

```

sigma := g;
d := c|x|;
z := y;
len := (16m)^d;
continue := (d>0);
bit := 0;
while continue do
  len := len div 16m;
  d := d-1;
  look up sigma = \prod_{i<m} a_i b_i a_i^{-1} b_i^{-1} from table
  if d even then % case 2
    M := 4*len;
    for i := 0 to m-1 do
      case
        z \in [iM+1, iM+len] :
          begin sigma := a_i; bit := 2*bit; end;
        z \in [iM+len+1, iM+2*len] :
          begin sigma := b_i; bit := 2*bit+1; end;
        z \in [iM+2*len+1, iM+3*len] :
          begin sigma := a_i^{-1}; bit := 2*bit; end;
        z \in [iM+3*len+1, iM+4*len] :
          begin sigma := b_i^{-1}; bit := 2*bit+1; end;
      case
    endfor;
  if z \in [4m*len+1, 16m*len] then

```

```

         $w_y := e$ ; continue := false;
    endif;
endif; %end of case 2
if d odd then %case 3
    M := 8*len;
    for i := 0 to m-1 do
        case
            z ∈ [iM+1, iM+len]:
                begin  $\sigma := b_{m-i}^{-1}$ ; bit := 2*bit; end;
            z = iM+len+1:
                begin  $w_y := b_{m-i}$ ; continue := false; end;
            z ∈ [iM+len+2, iM+2*len]:
                begin  $w_y := e$ ; continue := false; end;
            z ∈ [iM+2*len+1, iM+3*len]:
                begin  $\sigma := a_{m-i}^{-1}$ ; bit := 2*bit+1; end;
            z = iM+3*len+1:
                begin  $w_y := a_{m-i}$ ; continue := false; end;
            z ∈ [iM+3*len+2, iM+4*len]:
                begin  $w_y := e$ ; continue := false; end;
            z ∈ [iM+4*len+1, iM+5*len]:
                begin  $\sigma := b_{m-i}$ ; bit := 2*bit; end;
            z = iM+5*len+1:
                begin  $w_y := b_{m-i}^{-1}$ ; continue := false; end;
            z ∈ [iM+5*len+2, iM+6*len]:
                begin  $w_y := e$ ; continue := false; end;
            z ∈ [iM+6*len+1, iM+7*len]:
                begin  $\sigma := a_{m-i}$ ; bit := 2*bit+1; end;
            z = iM+7*len+1:
                begin  $w_y := a_{m-i}^{-1}$ ; continue := false; end;
            z ∈ [iM+7*len+2, iM+8*len]:
                begin  $w_y := e$ ; continue := false; end;
        endcase;
    endfor;
if z ∈ [8m*len+1, 16m*len-4m] then
     $w_y := e$ ; continue := false;
endif;
if z ∈ [16m*len-4m+1, 16m*len] then
    for i := 0 to m-1 do
        case
            z = 16m*len-4m+4i+1 :  $w_y := a_i$ ;
            z = 16m*len-4m+4i+2 :  $w_y := b_i$ ;
        endcase;
    endfor;
endif;

```

```

        z = 16m*len-4m+4i+3 :   wy := ai-1;
        z = 16m*len-4m+4i+4 :   wy := bi-1;
        endcase
        continue := false;
    endfor;
    endif;
endif; %end of case 3
continue := continue and (d>0);
endwhile;
if d=0 then %case1
    if M'(x, bit) accepts then wy := e else wy := σ endif
endif

```

We leave the verification of the algorithm to the reader. The operations of $a \operatorname{div} 4^b$, $a \operatorname{mod} 4^b$ are in linear time (even computable in the logtime hierarchy) and the fixed look-up table for products of commutators requires constant space. Addition, subtraction and testing inequality \leq of linear size integers can be done in linear time. The value $H(x, y) = \tau \in G$ exactly when there exists a sequence $\sigma_0, \sigma_1, \dots, \sigma_{c|x|}$ from G , a sequence $cont_0, cont_1, \dots, cont_{c|x|}$ from $\{\text{TRUE}, \text{FALSE}\}$, a sequence $bit_0, bit_1, \dots, bit_{c|x|}$ from $\{0, 1\}$ such that $\sigma_0 = g$, $cont_0 = \text{TRUE}$, $bit_0 = 0$, and for all $i < c|x|$ it is the case that the algorithm produces $\sigma_{i+1}, cont_{i+1}, bit_{i+1}$ from $\sigma_i, cont_i, bit_i$, and in the last step yields value $w_y = \tau$. Visibly, the graph of H belongs to Σ_2^L . It follows that $\text{ALINTIME} \subseteq \text{GLTH}$. ■

Part (a) of the following theorem is due to J. Paris [29] and part (b) was conjectured by him.

Theorem 45

- (a) $\text{SRM}(n^{O(1)}, 0) = \text{NSRM}(n^{O(1)}, 0) = \text{LTH}$,
- (b) $\text{SRM}(n^{O(1)}, 1) = \text{NSRM}(n^{O(1)}, 1)$.

Proof We prove only (b). For the direction from right to left, let M be $\text{NSRM}(n^{O(1)}, 1)$. Again, by the normalization lemma, every program P is equiv-

alent to a program P' of the form

$$1 : \left\{ \begin{array}{l} \text{if } \phi_0(x_1, \dots, x_m, t_0, \dots, t_k, r) \text{ then} \\ \quad t_0 := t_0 + 1; r := H_0(x_1, \dots, x_m, t_0, \dots, t_k, r); \\ \quad \text{goto } 1; \\ \text{if } \phi_1(x_1, \dots, x_m, t_0, \dots, t_k) \text{ then} \\ \quad t_1 := t_1 + 1; r := H_1(x_1, \dots, x_m, t_0, \dots, t_k, r); \\ \quad \text{goto } 1; \\ \quad \vdots \\ \text{if } \phi_k(x_1, \dots, x_m, t_0, \dots, t_k) \text{ then} \\ \quad t_k := t_k + 1; r := H_k(x_1, \dots, x_m, t_0, \dots, t_k, r); \\ \quad \text{goto } 1; \\ \text{if } \phi_{k+1}(x_1, \dots, x_m, t_0, \dots, t_k) \text{ then halt;} \end{array} \right.$$

where the ϕ_i, H_i are boolean combinations of atomic formulas, the H_i are multivalued functions and the ϕ_i are exhaustive; i.e. $(\forall \bar{x}, \bar{t}, r \bigvee_{i \leq k+1} \phi_i(\bar{x}, \bar{t}, r))$. Define as follows an equivalent program P'' which uses stack registers t_1, \dots, t_k and works in the same stack and work register bounds. Suppose that $\phi_0(x_1, \dots, x_m, 0, t_1, \dots, t_k, r)$ holds, so the current value of the 0-th stack register is 0 and the current value of the work register is r . Let $r_0 = r$. Define a multivalued function $R(\bar{x}, s, t_1, \dots, t_k, r)$ which gives the value of the work register after a sequence of s applications of the instruction

$$(*) \quad t_0 := t_0 + 1; r := H_0(\bar{x}, t_0, t_1, \dots, t_k, r)$$

provided that $(\forall i < s) \phi_0(\bar{x}, i, t_1, \dots, t_k, r_i)$ holds. Here $r_{i+1} = H_0(\bar{x}, t_0, t_1, \dots, t_k, r_i)$, so it will be the case that $r_{i+1} = R(\bar{x}, i+1, t_1, \dots, t_k, r_0)$. In the following, $R(i)$ will abbreviate $R(x_1, \dots, x_m, i, t_1, \dots, t_k, r_0)$. As well, let $h_i : \{0, 1\} \rightarrow \{0, 1\}$ be defined by $h_i(u) = H_0(\bar{x}, i, t_1, \dots, t_k, u)$. Note that since H_0 is a multivalued function, h_i is as well. Temporarily, define a *unique valued* function $F : \mathbb{N} \rightarrow M_2$ with Δ_0 graph by letting $F(i)$ be id_2 if $i \geq s$ or $Gr_{id_2} \subseteq h_i$ or $Gr_{(0,1)} \not\subseteq h_i$ and $F(i)$ be $(0, 1)$ else. Here, recall that M_2 is the monoid of all (unique valued) functions from $\{0, 1\}$ into $\{0, 1\}$, and that $(0, 1)$ denotes the cyclic permutation permuting 0 and 1.

In the following, notice that for $i < s$,

$$\overline{F}(s) \circ \overline{F}(i)^{-1} = F(s) \circ \dots \circ F(i+2) \circ F(i+1).$$

Since the domain $\{0, 1\}$ consists of only two elements,

$$F(s) \circ \dots \circ F(i+1) = id_2$$

iff

$$(\overline{F}(s) \circ \overline{F}(i)^{-1})(0) = 0.$$

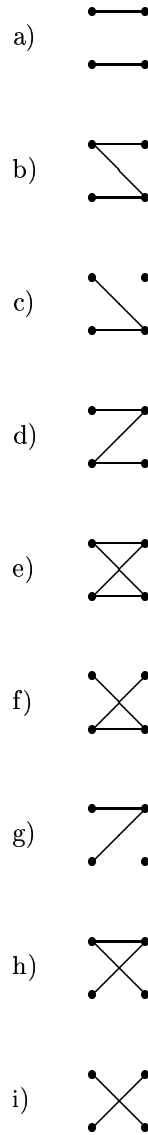
Similarly, letting τ denote the transposition $(0, 1)$

$$F(s) \circ \cdots \circ F(i + 1) = \tau$$

iff

$$(\overline{F}(s) \circ \overline{F}(i)^{-1})(0) = 1.$$

There are nine possible multivalued functions with domain and codomain $\{0, 1\}$. These functions are listed graphically as $(a), \dots, (i)$ below, where we think of domain element 0 [resp. 1] as lying in the lower left [resp. upper left] corner, and range element 0 [resp. 1] as lying in the lower right [resp. upper right] corner. Thus function (a) corresponds to the identity element id_2 on two elements, satisfying $id_2(x) = x$ for $x \in \{0, 1\}$. Composition of such multivalued functions then corresponds to composing their graphical representations from left to right.



Temporarily, for a finite multivalued function $h \subseteq \{0, 1\} \times \{0, 1\}$, let $SV(h)$ mean that h is a *single valued* function.

Case 1

$$(\exists s_0 < s)(\forall i < s)[SV(h_{s_0}) \wedge h_{s_0}(0) = h_{s_0}(1) \wedge (SV(h_i) \wedge s_0 < i \rightarrow h_i(0) \neq h_i(1))]$$

In other words, s_0 is the largest index for which h_{s_0} is of the form (c) or (g). In

this case, let $y_0 = h_{s_0}(0)$.

Case 2 Not Case 1, so for $i < s$, h_i is neither (c) nor (g).

In this case, let $y_0 = r$, and put $s_0 = -1$. By convention, set $\overline{F}(-1)^{-1}$ and $\overline{F}(-1)$ to be id_2 .

In both case 1 and 2, for all $s_0 < i \leq s$, $F(i) \subseteq h_i$ is a permutation – either id_2 or τ , where τ denotes the transposition $(0, 1)$. Define $R(\vec{x}, s, t_1, \dots, t_k, r) = y$ if

- $y = y_0$ and

$$(\overline{F}(s) \circ \overline{F}(s_0)^{-1})(0) = 0$$

OR

- $y = 1 - y_0$ and

$$(\overline{F}(s) \circ \overline{F}(s_0)(0)^{-1}) = 1.$$

Now R is a multi-function, and may have additional values defined as follows, in subcases (a),(b). Both case 1 and 2 have two subcases, which are identically treated (recall though that each case has different values of y_0, s_0).

Subcase (a)

$$(\exists i \leq s)[s_0 < i \wedge 0 \in h_i(0) \cap h_i(1) \wedge 1 \in h_i(0) \cap h_i(1)].$$

For such an index i , h_i is of the form (e). Define $R(\vec{x}, s, t_1, \dots, t_k, r) = 0$ and $R(\vec{x}, s, t_1, \dots, t_k, r) = 1$.

Subcase (b) Subcase (a) does not hold and

$$(\exists i \leq s)[s_0 < i \wedge h_i(0) \cap h_i(1) \neq \emptyset].$$

For any such index i , h_i is of the form (b), (d), (f), (h). Define $R(\vec{x}, s, t_1, \dots, t_k, r) = y$ iff

- $y = y_0$ and

$$(\exists i \leq s)[s_0 < i \wedge y_0 \in h_i(0) \cap h_i(1) \wedge (\overline{F}(s) \circ \overline{F}(i)^{-1})(0) = 0]$$

OR

- $y = 1 - y_0$ and

$$(\exists i \leq s)[s_0 < i \wedge y_0 \in h_i(0) \cap h_i(1) \wedge (\overline{F}(s) \circ \overline{F}(i)^{-1})(0) = 1]$$

OR

- $y = y_0$ and

$$(\exists i \leq s)[s_0 < i \wedge 1 - y_0 \in h_i(0) \cap h_i(1) \wedge (\overline{F}(s) \circ \overline{F}(i)^{-1})(0) = 1]$$

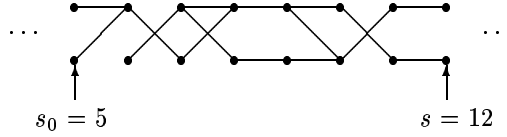
OR

- $y = 1 - y_0$ and

$$(\exists i \leq s)[s_0 < i \wedge 1 - y_0 \in h_i(0) \cap h_i(1) \wedge (\overline{F}(s) \circ \overline{F}(i)^{-1})(0) = 0].$$

Note that if neither subcase (a) nor (b) holds, then for all $s_0 < i \leq s$, h_i is a permutation, and hence $h_i = F(i)$. Thus this situation has already been treated.

A small example might be helpful. Suppose that H is of the form



Now case 1 holds, and s_0 is the largest index for which

$$SV(h_{s_0}) \wedge (\exists y_0)[y_0 \in h_{s_0}(0) \cap h_{s_0}(1)].$$

In the case at hand, $y_0 = 1$. Note that

$$\overline{F}(12) \circ \overline{F}(5)^{-1} = F(12) \circ \cdots \circ F(6) = \tau$$

where τ denotes the transposition $(0, 1)$. Thus $R(\vec{x}, 12, t_1, \dots, t_k, r) = h_{12}(0)$. Next, there are two values $s_0 < i \leq s$ for which $h_i(0) \cap h_i(1) \neq \emptyset$; namely $i = 7, 9$. That is, $h_7(0) = 1, h_7(1) = 1$ and $h_9(0) = 0, h_9(1) = 0$. Apply subcase (b) to h_9 . Note that

$$h_{11} \circ h_{10} = \overline{F}(11) \circ \overline{F}(9)^{-1}$$

is the transposition $\tau = (0, 1)$, hence we have that

$$R(\vec{x}, s, t_1, \dots, t_k, r) = h_{12} \circ \tau(0) = h_{12}(1).$$

Apply subcase (b) to h_7 . Note that $id_2 \subseteq h_9$, so $F(9) = id_2$ and $\overline{F}(11) \circ \overline{F}(7)^{-1}$ is the transposition $\tau = (0, 1)$. Hence we have as well that

$$R(\vec{x}, s, t_1, \dots, t_k, r) = h_{12} \circ \tau(1) = h_{12}(0).$$

Thus the multivalued function $R(\vec{x}, s, t_1, \dots, t_k, r)$ takes the values $h_{12}(0), h_{12}(1)$. Hopefully this illustrates how R is defined.

Now define the program P'' which refers only to stack registers t_1, \dots, t_k and is equivalent to P' by the following.

$$1 : \left\{ \begin{array}{l} \text{if } \psi_1(\bar{x}, t_1, \dots, t_k, r) \text{ then} \\ \quad t_1 := t_1 + 1; r := G_1(\bar{x}, t_1, \dots, t_k, r); \\ \quad \text{goto } 1; \\ \text{if } \psi_2(\bar{x}, t_1, \dots, t_k, r) \text{ then} \\ \quad t_2 := t_2 + 1; r := G_2(\bar{x}, t_1, \dots, t_k, r); \\ \quad \text{goto } 1; \\ \quad \vdots \\ \text{if } \psi_k(\bar{x}, t_1, \dots, t_k, r) \text{ then} \\ \quad t_k := t_k + 1; r := G_k(\bar{x}, t_1, \dots, t_k, r); \\ \quad \text{goto } 1; \\ \text{if } \phi_{k+1}(\bar{x}, t_1, \dots, t_k, r) \text{ then halt;} \end{array} \right.$$

Here $\psi_i(\bar{x}, t_1, \dots, t_k, r)$ is defined by

$$\begin{aligned} & (\exists s < (\max\{x_1, \dots, x_m\})^c) (\phi_i(x_1, \dots, x_m, s, t_1, \dots, t_k, R(s)) \\ & \wedge (\forall i < s) \phi_0(x_1, \dots, x_m, i, t_1, \dots, t_k, R(i))), \end{aligned}$$

and the multivalued function G_i is defined by $G_i(x_1, \dots, x_m, t_1, \dots, t_k, r, y)$ iff

$$\begin{aligned} & (\exists s < (\max\{x_1, \dots, x_m\})^c) (\phi_i(x_1, \dots, x_m, s, t_1, \dots, t_k, R(s)) \\ & \wedge \end{aligned}$$

$$(\forall i < s) \phi_0(x_1, \dots, x_m, i, t_1, \dots, t_k, R(i)) \wedge y = H_i(x_1, \dots, x_m, s, t_1, \dots, t_k, R(s))).$$

Iteration of this technique removes a stack and terminates in an equivalent program of the form

$$1 : \left\{ \begin{array}{l} \text{if } \Theta_k^*(x_1, \dots, x_m, t_k, r) \text{ then } t_k := t_k + 1; \\ \quad r := G_k^*(x_1, \dots, x_m, t_k, r); \text{ goto } 1; \\ \text{if } \Theta_{k+1}^*(x_1, \dots, x_m, t_k, r) \text{ then halt;} \end{array} \right.$$

Thus $M(x_1, \dots, x_m)$ accepts iff $\Theta_{k+1}^*(x_1, \dots, x_m, 0, 0)$ holds. The formula Θ_{k+1}^* is defined using bounded quantifiers and (deterministic) counting modulo 2. It follows that $\text{NSRM}(n^{O(1)}, 1) \subseteq \mathbf{Z}_2\text{LTH}$. It is straightforward to see that $\mathbf{Z}_2\text{LTH}$ is contained in $\text{SRM}(n^{O(1)}, 1)$ (see [28] for a sketch of a proof). This concludes the proof of part (b). ■

Remark 46 It is not obvious how to lift this technique to prove that

$$\text{SRM}(n^{O(1)}, k) = \text{NSRM}(n^{O(1)}, k)$$

for $k = 2, 3$. In particular, the previous proof used the fact that when h_i was of the form (b), (d), (f), (h), we either considered the case that $h_i(0) = h_i(1)$ or the *permutation* $F(i) \subset h_i$. Using different techniques, W.B. Handley [17] has proven that $\text{SRM}(n^{O(1)}, k) = \text{NSRM}(n^{O(1)}, k)$ for $k = 2, 3$ as well as given a different proof that $\text{SRM}(n^{O(1)}, k) = \text{NSRM}(n^{O(1)}, k)$ for $k \geq 4$.

Proposition 47 $\text{NSRM}(n^{O(1)}, k) \subseteq \text{NM}_{k+1}\text{LTH}$.

Proof sketch Before giving the proof, note that k is the bound for the work register and $k + 1$ is the size of the domain and codomain for the monoid M_{k+1} . Fix parameters $x_1, \dots, x_m, t_0, \dots, t_k, r$, and define a multivalued function $F : \mathbb{N} \rightarrow M_{k+1}$ with Δ_0 graph by $F(i) = H_0(x_1, \dots, x_m, i, t_0, \dots, t_k, r)$. Define the multivalued function

$$R(x_1, \dots, x_m, s, t_1, \dots, t_k, r) = \overline{F}(s - 1)$$

which belongs to $\text{NM}_{k+1}\text{LTH}$, and as in the previous proof, give an equivalent program P'' involving only stack registers t_1, \dots, t_k and work register r . Iterate this construction and note as in the previous proof that $M(x_1, \dots, x_m)$ accepts iff $\Theta_{k+1}^*(x_1, \dots, x_m, 0, 0)$ holds. Here Θ_{k+1}^* is not definable using counting modulo 2, but does belong to $\text{NM}_{k+1}\text{LTH}$. ■

Proposition 48 (Paris, Handley, Wilkie [28]) For all integers $k \geq 0$,

$$\text{SRM}(n^{O(1)}, k) = M_{k+1}\text{LTH} .$$

Proof Theorem 7 of [28] (p. 358) states that $\text{SRM}(n^{O(1)}, k) = S_{k+1}\text{LTH}$, where S_{k+1} is the full symmetric group on $k + 1$ letters. At the top of p. 359 of [28], it is shown by induction that $(\forall n)(\forall m \geq n)[M_n\text{LTH} \subseteq S_m\text{LTH}]$. Thus $\text{SRM}(n^{O(1)}, k) = M_{k+1}\text{LTH}$. Alternatively, the proposition can be directly proved using the approach in the proof of Proposition 47. ■

Theorem 49 For $k \geq 4$, $\text{NSRM}(n^{O(1)}, k) = \text{SRM}(n^{O(1)}, k) = \text{ALINTIME}$.

Proof We have

$$\begin{aligned} \text{NSRM}(n^{O(1)}, k) &\subseteq \text{NM}_{k+1}\text{LTH} \\ &\subseteq \text{ALINTIME} \\ &\subseteq S_{k+1}\text{LTH} \\ &\subseteq M_{k+1}\text{LTH} \\ &\subseteq \text{SRM}(n^{O(1)}, k) \square \end{aligned}$$

Corollary 50

$$\text{NSRM}(n^{O(1)}, O(1)) = \text{SRM}(n^{O(1)}, O(1)),$$

$$\text{SRM}(n^{O(1)}, O(1)) = \text{ALINTIME} .$$

Using these techniques, the following theorem is straight-forward.

Theorem 51 (J. Paris [29])

$$\text{NSRM}(n^{O(1)}, n^{O(1)}) = \text{NLINSPACE} .$$

Given these results, how much nondeterminism is required to help? Using the idea that one stack register which stores a number whose length is $2L$ can be simulated by two stack registers which store a number of length at most L , it is easy to show the following (see [22] for details for the deterministic case).

Proposition 52

$$(N)SRM(n^{O(1)}, r(n)) = (N)SRM(n, r(n)).$$

Thus in particular, we have that

$$(N)SRM(n^{O(1)}, k) = (N)SRM(n, k).$$

Currently we have no information about the following.

Question 53 Does there exist a function $f = o(\log n)$ for which $SRM(n, \log n) \subseteq NSRM(n, f(n))$. In particular, is it the case that $SRM(n, \log n) \subseteq NSRM(n, \log \log n)$, or even $SRM(n, \log(n)) \subseteq ALINTIME$?

5 Miscellaneous

For completeness, we state analogues of the previous results for the polynomial time hierarchy PH. The *smash* function $\#$ is defined by $x\#y = 2^{|x| \cdot |y|}$, where $|x| = \lceil \log_2(x + 1) \rceil$ denotes the length of the binary representation of x . Since $x\#(x\#x)$ is approximately $2^{|x|^3}$, by repeatedly applying the smash function, any unary function of polynomial growth rate is eventually majorized by some function in $[I, 0, s, +, \cdot, \#; COMP]$. Obvious modifications of the proofs of the previous results relating the linear time hierarchy and its extensions to stack register machines yield the following theorem.

Theorem 54

$$\begin{aligned} NSRM[+, \times, \#](2^{|n|^{O(1)}}, 0) &= SRM[+, \times, \#](2^{|n|^{O(1)}}, 0) \\ NSRM[+, \times, \#](2^{|n|^{O(1)}}, 1) &= SRM[+, \times, \#](2^{|n|^{O(1)}}, 1) \\ NSRM[+, \times, \#](2^{|n|^{O(1)}}, k) &= SRM[+, \times, \#](2^{|n|^{O(1)}}, k), \text{ for } k \geq 4 \\ SRM[+, \times, \#](2^{|n|^{O(1)}}, k) &= PSPACE, \text{ for } k \geq 4 \\ NSRM[+, \times, \#](2^{|n|^{O(1)}}, 2^{|n|^{O(1)}}) &= PSPACE. \end{aligned}$$

Proof By obvious modifications of the proofs of related results for the linear time hierarchy, $NSRM[+, \times, \#](2^{|n|^{O(1)}}, k)$ is equal to $SRM[+, \times, \#](2^{|n|^{O(1)}}, k)$ for $k = 0, 1$ and $k \geq 4$. For $k \geq 4$, it similarly follows that $NSRM[+, \times, \#](2^{|n|^{O(1)}}, k) = APTIME$, where the latter denotes the class of alternating polynomial time computable functions. By the well-known theorem of Savitch, alternating polynomial time equals polynomial space which equals nondeterministic polynomial space. ■

In [6] Cai and Furst introduce the k -state *safe-storage* Turing machine M which has a read-only input tape, a read/write work tape, a polynomially long binary *counter* tape or *clock*, and a *safe-storage* device capable of remembering k states. Regularly after a polynomial number of computation steps, the worktape is erased and all tape heads are reset to their initial positions, thus squeezing the computation into a “bottleneck” where only the state of the safe-store is retained. The intuition is that the machine repeatedly crashes after a polynomial number of steps, where only a small amount of information is saved from the crash. The class SF_k , for $k \geq 1$, is defined to be the collection of languages accepted by a k -state safe-storage Turing machine where intermediate computations between bottlenecks are performed in polynomial time. Note that there may be an exponential number of bottlenecks since the counter tape holds a binary number of polynomial length, and that

$$\text{PTIME} = SF_1 \subseteq SF_2 \subseteq \dots \subseteq \text{PSPACE} .$$

It is easy to see that the boolean hierarchy is contained in SF_2 . A language is said to be *logspace serializable* if the computations between bottlenecks are performed in logspace. Using the technique of Barrington [3], Cai and Furst [6] prove that PSPACE is logspace serializable by a 5 state safe-storage machine, and hence that $SF_5 = \text{PSPACE}$.

The characterization of PSPACE from [6] is related to the fourth line of Theorem 54. By techniques of Bennett [5], it follows that the computation of a polynomial time bounded Turing machine can be arithmetized by a bounded formula in the signature $\{0, 1, +, \times, \#, \leq\}$ and hence by a stack register machine with a fixed number of stack registers, no work register, with $\mathcal{F} = \{+, \times, \#\}$ and bound $2^{|n|^{O(1)}}$. Thus from $SF_5 = \text{PSPACE}$ one can show that $\text{SRM}[+, \times, \#](2^{|n|^{O(1)}}, 4)$ equals PSPACE . Concerning the converse direction, since $\text{SRM}[+, \times, \#](2^{|n|^{O(1)}}, 0) = \text{PH}$, our results are insufficient to show that PSPACE is logspace serializable. Finally, since nondeterminism in the stack register machine model concerns ambiguity in the branching instructions, the other statements of Theorem 54 are unrelated to the results of [6]. In summary, the results of [6] concern a hierarchy between PTIME and PSPACE , whereas Theorem 54 concerns a hierarchy between PH and PSPACE .

Comparison of our results with those of [6] suggests the following type of question. Define NSF_k as in SF_k , but using a nondeterministic polynomial time bounded machine between bottlenecks. For $k \geq 5$, $SF_k = NSF_k = \text{PSPACE}$. What is the situation for $1 \leq k < 5$?

Question 55 For $1 \leq k < 5$, is it the case that $SF_k \subset NSF_k$?

6 Acknowledgements

I would like to thank J. Paris for correspondence and permission to state his theorems 45 (a) and 51. Thanks to E. Allender and S. Buss for mentioning the relevance of the work [6] of Cai and Furst, and suggesting the possibility of a similar characterization of alternating linear time. Thanks to E. Kranakis for suggesting a relation with Presburger arithmetic, and to two anonymous referees for many helpful remarks, corrections and suggestions.

References

- [1] G. Asser. Das Repräsentantenproblem im Prädikatenkalkül der ersten Stufe mit Identität. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 1:252 – 263, 1955.
- [2] D. Mix Barrington, N. Immerman, and H. Straubing. On uniformity in NC^1 . *J. Comp. Syst. Sci.*, 41(3):274–306, 1990. Preliminary version appeared in *Structure in Complexity Theory: 3rd Annual Conference*, IEEE Computer Society Press (1988), 47-59.
- [3] D.A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1 . *Journal of Computer and System Sciences*, 38:150–164, 1989. Preliminary version in *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*(1986) 1-5.
- [4] A. Bel'tyukov. A computer description and a hierarchy of initial Grzegorzcyk classes. *Journal of Soviet Mathematics*, 20:2280 – 2289, 1982. Translation from *Zap. Nauk. Sem. Lening. Otd. Mat. Inst.*, V. A. Steklova AN SSSR, Vol. 88, pp. 30 - 46, 1979.
- [5] J.H. Bennett. *On Spectra*. PhD thesis, Princeton University, 1962. Department of Mathematics.
- [6] J.-Y. Cai and M.L. Furst, $PSPACE$ survives three-bit bottlenecks. Proceedings of the Third Annual IEEE Conference on *Structure in Complexity Theory*, 94–102 (1988). Journal version in *International Journal of Foundations of Computer Science*, vol 2 (1991) 67–76.
- [7] P. Cegielski, Thèse de Doctorat de 3-ième Cycle. Université Paris VII (1980).
- [8] A. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the Association of Computing Machinery*, 28:114 – 133, 1981.
- [9] P. Clote. Stack register machines: nondeterminism, the linear time hierarchy, and boolean circuits. Contributed talk, Spring meeting of the Association for Symbolic Logic, March 19–22, 1992, Durham NC.
- [10] Eds. P. Clote and J. Krajíček. *Arithmetic, Proof Theory and Computational Complexity*. Oxford University Press, 1993. 428 pages.
- [11] P. Clote, B. Kapron, and A. Ignjatovic. Parallel computable higher type functionals. In *Proceedings of IEEE 34th Annual Symposium on Foundations of Computer Science*, Nov 3–5, 1993. Palo Alto CA. pp. 72–83.
- [12] P. Clote and G. Takeuti. First order bounded arithmetic and small boolean circuit complexity classes. *Feasible Mathematics II*. Eds. P. Clote and J. Remmel, Birkhäuser Inc. (1995).
- [13] P. Clote. Sequential, machine-independent characterizations of the parallel complexity classes $ALOGTIME$, AC^k , NC^k and NC . In P.J. Scott S.R. Buss, editor, *Feasible Mathematics*, pages 49–70. Birkhäuser, 1990.
- [14] A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science II*, pages 24–30. North-Holland, 1965.

- [15] S.A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.
- [16] A. Grzegorzczuk. Some classes of recursive functions. *Rozprawy Matematyczne*, 4, 1953.
- [17] W.G. Handley. Deterministic summation modulo \mathcal{B}_n , the semigroup of binary relations on $\{0, 1, \dots, n-1\}$. Accepted by *Theoretical Computer Science*, subject to revisions, May 1994.
- [18] W.G. Handley. Nondeterministic summation modulo \mathcal{T}_3 , the semigroup of functions on $\{0, 1, 2\}$. Submitted, June 1994.
- [19] W.G. Handley. Nondeterministic summation *mod* the semigroup of pregraphs over $\{0, 1, 2\}$ yields ALINTIME. Typescript, December 1994.
- [20] W.G. Handley. Nondeterministic summation modulo \mathcal{T}_4 , the semigroup of functions on $\{0, 1, 2, 3\}$. Typescript, September 1994.
- [21] I.N. Herstein, *Topics in Algebra*, 342 pages. Ginn and Co., 1964.
- [22] J. Hicks. *A machine characterization of quantification and primitive recursion with applications to low level complexity classes*. PhD thesis, Department of Mathematics, Oxford University, 1983.
- [23] N. Immerman. Expressibility and parallel complexity. *SIAM J. Comput.*, 18(3):625–638, 1989.
- [24] N.D. Jones and A.L. Selman. Turing machines and the spectra of first-order formulas. *Journal of Symbolic Logic*, 39:139–150, 1974.
- [25] J.C. Lind. Computing in logarithmic space. Technical Report Project MAC Technical Memorandum 52, Massachusetts Institute of Technology, September 1974.
- [26] V.A. Nepomnjascii. Rudimentary interpretation of two-tape turing computation. *Kibernetika*, 2:29–35, 1970. Translated in *Cybernetica* (1972) 43–50.
- [27] J. B. Paris and A. J. Wilkie. Counting problems in bounded arithmetic. In C. A. di Prisco, editor, *Methods in Mathematical Logic*, pages 317 – 340. Springer Verlag Lecture Notes in Mathematics, 1983. Proceedings of Logic Conference held in Caracas, 1983.
- [28] J. B. Paris, W.G. Handley, and A. J. Wilkie. Characterizing some low arithmetic classes. In *Theory of Algorithms*, pages 353 – 364. Akademie Kyado, Budapest, 1984. Colloquia Societatis Janos Bolyai.
- [29] J.B. Paris. Private correspondence.
- [30] A. A. Razborov. Lower bounds for the size of circuits of bounded depth in basis $\{\wedge, \oplus\}$. Preprint in English (translated from the Russian), 1986.
- [31] R.W. Ritchie. Classes of predictably computable functions. *Trans. Am. Math. Soc.*, 106:139–173, 1963.
- [32] R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, 1987. pp. 77 - 82.
- [33] C. Smorynski. *Logical Number Theory I*. Springer Verlag, 405 pages. 1991.

- [34] H. Straubing. *Finite Automata, Formal Languages, and Circuit Complexity*. Birkhäuser. Series *Progress in Theoretical Computer Science* 1994.
- [35] D.B. Thompson. Subrecursiveness: machine independent notions of computability in restricted time and storage. *Math. Systems Theory*, 6:3–15, 1972.
- [36] A.R. Woods. *Some problems in logic and number theory and their connections*. Ph.D. Dissertation, Department of Mathematics, University of Manchester, 132 pages, 1981.
- [37] C. Wrathall. Complete sets and the polynomial time hierarchy. *Theoretical Computer Science*, 3:23 – 33, 1976.