AN INFINITE PEBBLE GAME AND APPLICATIONS

A.J. Kfoury*
Department of Computer Science
Boston University
Boston, MA 02215, U.S.A.

A.P. Stolboushkin†
Program in Computing
UCLA Department of Mathematics
Los Angeles, CA 90024, U.S.A.

September 30, 1996

Contents

*Partly supported by NSF grant CCR–9417382.
†Partly supported by NSF grant CCR–9403809.

1 Introduction

We generalize the pebble game to infinite *dag*'s (*d*irected *a*cyclic *g*raphs), and we use this generalization to give new and shorter proofs of well-known results in two areas of computer science (as diverse as "logic of programs" and "formal language theory").

Our main application is a short and perspicuous proof of a theorem that first appeared in Tiuryn [18] and (independently) in Erimbetov [2, 3], asserting that unbounded memory increases the power of logics of programs. The driving idea in these papers is an analysis of bounded-memory program logics in a free algebra of one binary function, involving back-and-forth constructions based on a variant of Ehrenfeucht-Fraïssé games. A careful implementation of this idea, however, becomes quite technical and lengthy. Musikaev and Taitslin [11] point to some, apparently correctable, problems in [18] and present another construction proving a stronger result from which the main theorem of [18, 2] follows, although comprehensibility remains an issue, to say the least.

By contrast, our proofs are not only shorter, but also elementary. All we need is essentially finite induction and some freshmen-level logic. We feel that our technique can be widely applied in computer science, and indeed we apply it to give a simple proof of a result due to Salomaa [15], asserting the existence of a context-free grammar with infinite index.

The peculiar structure $\mathcal{N}$ defined in Section 6 below was invented by the first author in [7], and the proof of Lemma 3 given in Section 4 was written by the second author as a result of reading [7] in 1986. However, without recent encouragement from D. Kozen, Y. Moschovakis, and J. Tiuryn, this paper might not have been written. We express our thanks to them all, as well as to M.A. Taitslin for his valuable comments. Special thanks are due to S. Gao and to the anonymous referees for their help in improving the present paper.

2 The Infinite Pebble Game

The standard pebble game on a finite dag $\mathcal{D}$ is defined as follows. At any point in the game, some nodes of $\mathcal{D}$ will have pebbles on them. A *configuration* is a subset of the nodes, comprising just those nodes that have pebbles on them. A *legal move* consists in either:

- removing a pebble from a node a, or
- placing a pebble on a node a' such that all the immediate predecessors of a' have pebbles on them, or
- leaving the configuration unchanged (this one is needed for technical reasons).

An input node b is one that has no immediate predecessors, i.e. its in-degree $= 0$, and therefore a pebble may be placed on b at any time.

A legal move goes from a configuration C to a new configuration C', and is therefore represented by the pair (C, C'). A *calculation* is a sequence of configurations, each successive pair of which being a legal move. A calculation is said to *reach* a node a if it includes a in one of its configurations.

The pebble game on a finite dag is usually examined to study questions of time-space trade-offs. "Time" corresponds to the number of moves in a calculation and "space" to the maximum number of nodes in any configuration in this calculation. The dag's usually have exactly one output node, i.e. a node with out-degree $= 0$. We can state the aim of the pebble game on a finite dag $\mathcal{D}$ as follows:

> *Determine a lower bound on the number of pebbles required by a calculation that begins with the empty configuration and reaches the output node of $\mathcal{D}$.*

We have just described the basic pebble game on finite dag's, the so-called "black pebble game". There are several generalizations of the game on finite dag's, introduced for various applications in computer science.[1]

We now introduce the *infinite pebble game*. The game is played on an infinite dag $\mathcal{D}$. The concepts of "configuration" and "legal move" are the same as in the case of the finite pebble game (a configuration C is a subset of the nodes of $\mathcal{D}$, while a legal move either removes a node a from C or adds a node a' to C provided all the immediate predecessors of a' are in C). The aim of the pebble game on an infinite dag $\mathcal{D}$ is:

> *Determine a lower bound on the number of pebbles required by a calculation (necessarily infinite) that begins with the empty configuration and reaches every node of $\mathcal{D}$.*

We say that a dag $\mathcal{D}$ is *n-accessible*, or *accessible* to the n-pebble game, if there is a calculation that begins with the empty configuration, reaches the output node of $\mathcal{D}$ (if $\mathcal{D}$ is finite) or reaches all the nodes of $\mathcal{D}$ (if $\mathcal{D}$ is infinite), and uses at most n pebbles. The ω-pebble game is the one which can use countably many pebbles. We say that a dag $\mathcal{D}$ is *ω-accessible* if it is accessible to the ω-pebble game.

It is easy to see that a dag $\mathcal{D}$ is ω-accessible iff every node in $\mathcal{D}$ has finitely many predecessors (not necessarily all immediate). No calculation starting from the empty configuration can reach all the nodes of a dag which is not ω-accessible. We therefore limit our attention to dag's $\mathcal{D}$ such that:

1. $\mathcal{D}$ is ω-accessible, and
2. $\mathcal{D}$ contains only finitely many input nodes

Requirement 1, ω-accessibility, is stronger than well-foundedness; in a well-founded dag there are no infinite descending chains, but this does not preclude the presence of nodes with infinitely many predecessors. Requirement 2, the restriction to finitely many input nodes, is added in order to make some of the definitions below compatible.

We need a finer view of the accessibility properties of a dag $\mathcal{D}$. Following standard terminology in algebra, we say $\mathcal{D}$ is *locally finite* if every node in $\mathcal{D}$ has finitely many successors (not necessarily

[1] Some of the generalizations of the game on finite dag's are the "black-and-white pebble game", the "pebble game with auxiliary pushdown stores", the "edging game", and the "parallel pebble game" (see [13], [14], [9], and [17], respectively). Other relevant research on finite pebble games can be found in [1], [6], and in their references.

all immediate), and that $\mathcal{D}$ is *uniformly locally finite* if there is a constant $k \geqslant 1$ such that every node in $\mathcal{D}$ has at most k successors (not necessarily all immediate). We say $\mathcal{D}$ is *uniformly locally finite w.r.t. bounded space* if for every $n \geqslant 1$, there is a $k_n \geqslant 1$ such that every calculation of the n-pebble game on $\mathcal{D}$ reaches at most k_n nodes.

If $\mathcal{D}$ is uniformly locally finite, then $\mathcal{D}$ is uniformly locally finite w.r.t. bounded space; in Section 4 we show the opposite implication is not always true.

It is also clear that if $\mathcal{D}$ is uniformly locally finite, then $\mathcal{D}$ is locally finite, and the opposite implication does not always hold. On the other hand, the class of locally finite dag's and the class of dag's that are uniformly locally finite w.r.t. bounded space are incomparable, i.e. neither contains the other. This last result, although not used in the applications of Sections 5 and 6, is easily derived from the analysis of Section 4.

3 Applying the Infinite Pebble Game

A finite dag $\mathcal{D}$ is always accessible to the $|\mathcal{D}|$-pebble game, where $|\mathcal{D}|$ is the number of nodes in $\mathcal{D}$. What makes the finite pebble game interesting is the possibility that fewer than $|\mathcal{D}|$ pebbles will also suffice. Using fewer than $|\mathcal{D}|$ pebbles will generally increase the time (the number of moves) required to reach the output node of $\mathcal{D}$, and the hard question is usually to determine how time increases as the number of pebbles is decreased.

Similarly, what makes the infinite pebble game interesting is the fact that an infinite ω-accessible dag $\mathcal{D}$ may *not* be n-accessible for every finite $n \geqslant 1$. In each of the applications below we want to prove that a certain resource cannot be finitely bounded. The resource in question in the Salomaa theorem is the number of "occurrences of variables in derivations in a context-free grammar"; in the Tiuryn-Erimbetov theorem, the resource is the number of "memory locations used by a program". In each case this resource is simulated by the number of pebbles used in a game on an appropriately defined infinite dag.

In our infinite pebble game there are no time considerations. From the moment we prove that infinitely many pebbles are required to reach all the nodes of a dag, there is clearly no question of trading time (moves) for space (pebbles).

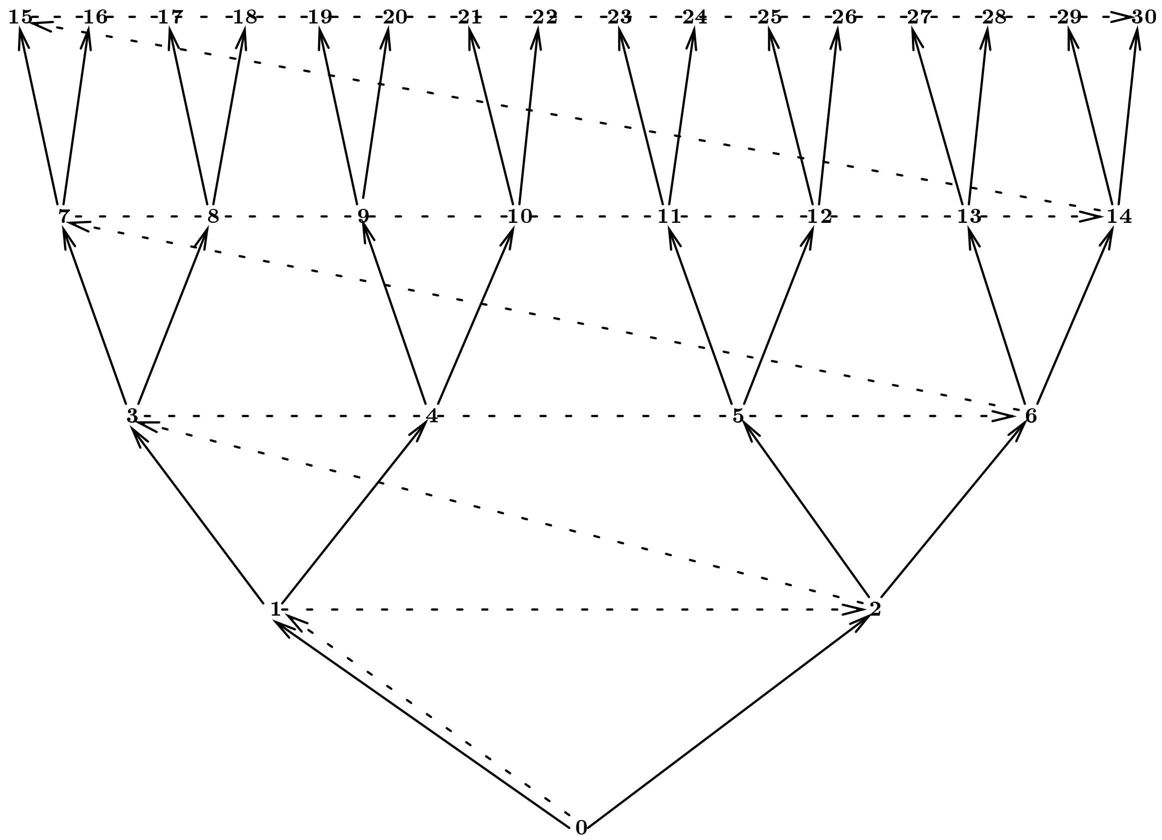
4 Two Specific Infinite Dag's

Let $\langle f, a\rangle$ be a first-order signature, where f is a binary function symbol and a is a zero-ary (constant) symbol. We define $\mathcal{A}$ to be the set of all terms over $\langle f, a\rangle$, i.e., $\mathcal{A}$ is the least set such that:

$$\mathcal{A} \supseteq \; \{a\} \;\cup\; \{f(t,u) \mid t,u \in \mathcal{A}\}$$

To turn $\mathcal{A}$ into a dag, we place an arrow from $t \in \mathcal{A}$ to $u \in \mathcal{A}$ iff $u = f(t,t')$ or $u = f(t',t)$ for some $t' \in \mathcal{A}$. It is clear that $\mathcal{A}$ is ω-accessible (every node has finitely many predecessors).

We consider another dag $\mathcal{B}$. The set of nodes of $\mathcal{B}$ is ω, the set of all the natural numbers. For every integer $n \geqslant 0$ we place an arrow from n to $n+1$, as well as an arrow from n to $2n+1$ and an arrow from n to $2n+2$. The dag $\mathcal{B}$ is conveniently viewed as two superimposed dag's, $\mathcal{B}_1$ and $\mathcal{B}_2$. $\mathcal{B}_1$ is the ordinary linear order of the natural numbers. $\mathcal{B}_2$ is the complete binary tree, where 0 is the root node and the nodes of the k-th level are $2^k-1, \ldots, 2^{k+1}-2$. It is clear that $\mathcal{B}$ is also an infinite ω-accessible dag. An initial fragment of $\mathcal{B}$ is shown below: $\mathcal{B}_1$ is described by the dashed arrows, and $\mathcal{B}_2$ by the plain arrows.



Lemmas 1 and 2 are simple and intuitive facts about the pebble game on an arbitrary dag (not only $\mathcal{A}$ or $\mathcal{B}$), finite or infinite. We give formal proofs for the sake of completeness. If a and b are nodes in a dag $\mathcal{D}$, we write $a \longrightarrow b$ if there is an arrow from a to b. The transitive closure of $\longrightarrow$ is denoted $\xrightarrow{+}$, and its reflexive transitive closure $\xrightarrow{*}$.

Lemma 1 *Let C be a configuration of the n-pebble game on a dag $\mathcal{D}$, with $n \geqslant 1$. Let $a \in \mathcal{D}$ and $b \in C$ such $a \xrightarrow{+} b$. If τ is a calculation of the n-pebble game on $\mathcal{D}$ starting with C such that:*

- *in every configuration C' of τ, at least one element $c \in C'$ is such that $b \xrightarrow{*} c$, and*
- *τ reaches a,*

then there exists a calculation $\widehat{\tau}$ of the $(n-1)$-pebble game on $\mathcal{D}$ that starts with the configuration $C \setminus \{b\}$ and reaches a.

Proof: Intuitively, because b is not a predecessor of a, it follows that b (and all its successors) can be omitted from the calculation τ reaching a. More formally, let τ be the sequence of configurations $C_0, C_1, C_2, \ldots$, where $C_0 = C$. Let $k \in \omega$ be the least number such that $a \in C_k$. Define a new sequence of configurations $\widehat{C}_0, \widehat{C}_1, \ldots, \widehat{C}_k$, where $\widehat{C}_j = C_j \setminus \{c \mid b \stackrel{*}{\longrightarrow} c\}$ for every $j = 1, \ldots, k$. Clearly, $|\widehat{C}_j| \leqslant (n-1)$ for every $j = 1, \ldots, k$ and, moreover, if (C_j, C_{j+1}) is a legal move then so is $(\widehat{C}_j, \widehat{C}_{j+1})$. The sequence $\widehat{C}_0, \widehat{C}_1, \ldots, \widehat{C}_k$ is therefore a calculation of the $(n-1)$-pebble game which, moreover, reaches a because $a \in \widehat{C}_k$. The desired calculation τ' is $\widehat{C}_0, \widehat{C}_1, \ldots, \widehat{C}_k$. *Q.E.D.*

A subdag $\mathcal{E}$ of a dag $\mathcal{D}$ is obtained by removing: (1) some of the arrows, and (2) some of the nodes together with all arrows adjacent to these nodes. We can identify a dag by its set of arrows. The *inward closure* $[\mathcal{E}]$ of subdag $\mathcal{E}$ of $\mathcal{D}$ is the least set of arrows such that:

$$[\mathcal{E}] \;\supseteq\; \mathcal{E} \;\cup\; \{\; a \longrightarrow b \in \mathcal{D} \mid \text{there is a node } c \text{ such that } c \longrightarrow b \in [\mathcal{E}] \;\}$$

Nodes a and c in this definition are not necessarily distinct. $\mathcal{E}$ is *inward closed* if $\mathcal{E} = [\mathcal{E}]$.

Lemma 2 *If a dag $\mathcal{D}$ is accessible to the n-pebble game for some $n \geqslant 1$, then every inward closed subdag $\mathcal{E}$ of $\mathcal{D}$ is also accessible to the n-pebble game.*

Proof: Suppose $\mathcal{D}$ is infinite (the finite case is similar). Let $X = \mathcal{D} \setminus \mathcal{E}$. Let $C_0, C_1, \ldots$ be a calculation that reaches all the nodes of $\mathcal{D}$. Then $(C_0 \setminus X), (C_1 \setminus X), \ldots$ is a calculation that reaches all the nodes of $\mathcal{E}$. *Q.E.D.*

Lemma 3 is specific to the dag $\mathcal{B}$ defined earlier. On the dag $\mathcal{B}$, the relation $i \stackrel{*}{\longrightarrow} j$ coincides with $i \leqslant j$ (there is a directed path, not necessarily a single arrow, from i to j). Let C be a configuration in $\mathcal{B}$, i.e., C is here a finite set of natural numbers, and $k \geqslant 0$ is an integer. We define the *k-neighborhood* of C, denoted $\mathbf{N}(C, k)$, as:

$$\mathbf{N}(C, k) = \;\{ j \mid (\exists i \in C \cup \{0\}) \;[\; j \geqslant i \text{ and } j - i \leqslant k \;]\;\}$$

$\mathbf{N}(C, k)$ is clearly a finite set. We define the function α on the natural numbers inductively:

$$\alpha(0) = 0 \qquad \text{and} \qquad \alpha(n+1) = 4 \cdot (n+1) \cdot (\alpha(n) + 1) + 1 \quad .$$

Lemma 3 *Let C be a configuration of the n-pebble game on $\mathcal{B}$, with $n \geqslant 1$. Every calculation of the n-pebble game on $\mathcal{B}$ that begins with configuration C can only reach nodes in $\mathbf{N}(C, \alpha(n))$.*

Proof: By induction on n. The lemma is obviously true for $n = 1$. Proceeding inductively, assume it is true for any number of pebbles $< n$. Assume for a moment that the lemma fails for n. Then there exists a finite calculation τ in the n-game starting with C that reaches some $k \notin \mathbf{N}(C, \alpha(n))$. Fix this k. Take the largest $j \in C \bigcup \{0\}$ such that $j < k$. Clearly,

- $k - j > \alpha(n)$, and
- the set $\{j+1, j+2, \ldots, k\}$ contains no element in C .

Fix this j as well.

Let $C_0 = C$, and C_i be the i-th configuration in the calculation τ. Then let $\tau_0 = \tau$, and τ_i be τ without the first i configurations.

Further let $m = \lceil (k-j+1)/2 \rceil + j + 1$, and let $F = \{m, m+1, \ldots, k\}$. In other words, F consists of the second half of the sequence $\{j+1, j+2, \ldots, k\}$. It can be seen that F satisfies the following properties:

- no element of C belongs to F , and
- $\lfloor k/2 \rfloor < m$.

Let ℓ be the least index such that for every $i \geqslant \ell$, the set C_i contains an element in F. Clearly,

- the set $C_{\ell-1}$ contains no element of F, and
- for every $j \in F$ there is an $i \geqslant \ell$ such that C_i contains j

Fix this ℓ. According to the rules of the pebble game, in order to place pebbles on the elements $\{m, m+1, \ldots, k\}$, it is necessary to have pebbles on $\lceil m/2 \rceil - 1, \lceil (m+1)/2 \rceil - 1, \ldots, \lceil k/2 \rceil - 1$. Let $F' = \{\lceil m/2 \rceil - 1, \lceil (m+1)/2 \rceil - 1, \ldots, \lceil k/2 \rceil - 1\}$. It follows from the properties of ℓ that for every $j \in F'$ there exists an $i \geqslant \ell$ such that $j \in C_i$. Since every $j \in F'$ is less than m, we may now apply Lemma 1. Thus there exists a calculation $\widehat{\tau}$ in the $(n-1)$-game starting with the configuration $\widehat{C}_\ell = C_\ell \setminus \{m\}$ which, in contradiction with the induction hypothesis, reaches nodes outside of $\mathbf{N}(\widehat{C}_\ell, \alpha(n-1))$. Indeed, all the elements in the set F' are reachable, while the cardinality of the set F' is

$$\left|F'\right| \;=\; \left\lfloor \frac{|F|}{2} \right\rfloor \;\geqslant\; \frac{k-j}{4} \;>\; \frac{\alpha(n)}{4} \;=\; \frac{4n(\alpha(n-1)+1)+1}{4} \;>\; n(\alpha(n-1)+1) \;.$$

At the same time, $\left|\mathbf{N}(\widehat{C}_\ell, \alpha(n-1))\right| \leqslant n(\alpha(n-1)+1)$. *Q.E.D.*

Theorem 4 is an immediate consequence of Lemma 3 . It first appeared in [7], stated differently (Proposition 3, page 53), with a substantially longer proof (several journal pages).

Theorem 4 *$\mathcal{B}$ is uniformly locally finite w.r.t. bounded space, but not uniformly locally finite.*

Corollary 5 *For every $n \geqslant 1$, the dag $\mathcal{B}$ is not accessible to the n-pebble game.*

The following result follows from Lemma 2, Corollary 5 and the fact that the dag $\mathcal{B}$ is an inward closed subdag of the dag $\mathcal{A}$.

Theorem 6 *For every $n \geqslant 1$, the dag $\mathcal{A}$ is not accessible to the n-pebble game. (Note, however, that $\mathcal{A}$ is not uniformly locally finite w.r.t. bounded space.)*

5 A CFG of an Infinite Index

Let $G = (V, X, S, P)$ be a context-free grammar, where V is the set of variables, X the set of terminal letters, $S \in V$ the start symbol, and P the production rules. For a word $w \in (V \cup X)^*$, we write $\mathcal{V}(w)$ for the word obtained from w by erasing all letters of X. The *index* of a derivation:

$$D : S = w_0 \Rightarrow w_1 \Rightarrow \cdots \Rightarrow w_n = w$$

in G is defined by:

$$ind(D) = \ max\{ \ |\mathcal{V}(w_i)| \mid i = 0, 1, \ldots, n \ \}$$

where $|\mathcal{V}(w_i)|$ denotes the length of the word $\mathcal{V}(w_i)$. The index of a word $w \in L(G)$ in G is:

$$ind(w, G) = \ min\{ \ ind(D) \mid D : S \stackrel{*}{\Longrightarrow} w \ \}$$

The *index* of G, denoted $ind(G)$, is the smallest integer n such that, for all $w \in L(G)$, $ind(w, G) \leqslant n$. If no such n exists, G is said to be of *infinite index.* The *index* of a context-free language L is:

$$ind(L) = \ min\{ \ ind(G) \mid L = L(G) \ \}$$

Salomaa (see [15, 16]) proved that the following context-free grammar is of infinite index:

$$S \longrightarrow \ SS \mid (S) \mid \varepsilon$$

We prove that the same result for a slightly different grammar $\widetilde{G}$:

$$S \longrightarrow \ (SS) \mid ()$$

We associate with the language $\widetilde{L}$ generated by this grammar, the dag $\mathcal{A}$ of Section 4. If we consider the words of $\widetilde{L}$ as nodes, and we place an arrow from $v \in \widetilde{L}$ to $w \in \widetilde{L}$ iff $w = (vv')$ or $w = (v'v)$ for some $v' \in \widetilde{L}$, then the resulting dag is clearly isomorphic to $\mathcal{A}$ (with $()$ being mapped to a). This isomorphism between $\widetilde{L}$ and $\mathcal{A}$, together with Theorem 6, allows us to prove the following results in a straightforward manner.

Theorem 7 *The grammar $\widetilde{G}$ is of infinite index.*

Proof: Consider a derivation D. We scan this derivation in reverse order (starting with the last production), and play a pebble game as follows:

If at the current step the derivation applies the rule $S \longrightarrow ()$, we place a new pebble on a and say that this pebble corresponds to the occurrence of S.

If at the current step the derivation applies the rule $S \longrightarrow (SS)$, the two occurrences of S on the right-hand side correspond to some two pebbles in $\mathcal{A}$ that sit on some $t, u \in \mathcal{A}$, respectively. We then place a pebble on $f(t, u)$, and remove the two pebbles from t and u. The new pebble on $f(t, u)$ is said to correspond to the occurrence of S on the left-hand side of the rule.

It is clear that, as the result of this game determined by the derivation D, we place a pebble on the isomorphic image of the word that D yields. It is also clear, that the number of pebbles used in the game is the index of the derivation D, plus 1. A minor difference from the definition of the pebble game is that we do allow more than one pebble to be put on the same node. However, this can be straightened out very easily. The theorem follows. *Q.E.D.*

¿From his grammar result, Salomaa [15, 16] went further and showed using a simple argument that the *language* of well-formed sequences of parentheses is of infinite index as well. Using the same argument, it can be shown that our language $\widetilde{L}$ is of infinite index.

6 Bounded vs. Unbounded Memory

A fundamental result about program schemes is that unbounded memory adds to the computational power of programs.[2] An early version of this result, comparing flowchart schemes and recursive schemes, was first proved by Paterson and Hewitt [12]. The result was later refined and reproduced relative to other classes of program schemes.

We will be deliberately vague in defining “program schemes”. There are several closely related definitions in the literature (e.g. see [5, 10]), which all start from the same basic programming formalism, and gradually add to it various features such as: non-determinism, recursive calls (with or without parameters, with or without higher-order parameters), counters, pushdown stores (binary and algebraic), arrays, and others. This same basic programming formalism is varyingly called: “flowchart programs”, “regular programs”, “**while**-programs”, and “iterative programs”, which are all assumed deterministic, and restricted to finitely many ground variables and atomic tests on ground variables.

Let P be an arbitrary member from anyone of these classes of program schemes. We leave it to the interested reader to check, by going back to the relevant literature, that P can always be translated into a possibly infinite non-deterministic **if-fi** statement, denoted $eds(P)$, written as a recursively enumerable sequence of “guarded commands” (to use a well-established notion in programming). We call a recursively enumerable sequence of guarded commands an *effective definitional scheme*, abbreviated *eds*, where all the guarded commands are written over the same finite (first-order) signature and finite set of (first-order) variables. The uninterpreted function and relation symbols appearing in P form a finite signature, which is also the signature of $eds(P)$. Following common notation, e.g. see [4], we take a guarded command as an expression of the form $\varphi \to t$, where φ is a quantifier-free first-order formula and t is a first-order term, both containing no free variable other than the input variables of P. We can therefore write

$$eds(P) \quad = \quad (\ \varphi_i \to t_i \mid i \in I\ )$$

[2] Perhaps more accurately, we should say that it adds power to the “control structure” of programs, since we are dealing with program schemes (uninterpreted programs).

where I is an r.e. subset of ω. Let $S = \mathit{eds}(P)$. Assuming that P takes one input value, stored in input variable x, and returns one output value,[3] the interpretation of P (or S) in a first-order structure $\mathcal{M}$ with universe M, denoted $P^{\mathcal{M}}$ (or $S^{\mathcal{M}}$), is the relation on M defined by the possibly infinite disjunction:

$$\bigvee_{i \in I} (\ \varphi_i(x)\ \wedge\ y = t_i(x)\ )$$

in the structure $\mathcal{M}$, i.e.

$$P^{\mathcal{M}}\ =\ S^{\mathcal{M}}\ =\ \{(a,b) \in M \times M \mid \text{there is an } i \in I \text{ such that } \mathcal{M} \models \varphi_i[a] \wedge\ b = t_i[a]\ \}\ \ .$$

For this definition to make sense, the signature of P and S is assumed to be contained in the signature of $\mathcal{M}$. Likewise, the interpretation of a term t in a structure $\mathcal{M}$ is denoted $t^{\mathcal{M}}$.

With every first-order term t we associate a finite dag, called $\mathit{dag}(t)$, with as many nodes as there are distinct subterms in t. The input nodes of $\mathit{dag}(t)$ are labelled with the variables and constant symbols appearing in t, and its single output node is labelled with the full expression for t. Put differently, $\mathit{dag}(t)$ is obtained from the parse tree of t by merging equal subterms. The pebble complexity of t is given by:

$$\mathit{pebble}(t)\ =\ \min\ \{\ n \mid \mathit{dag}(t) \text{ is accessible to the } n\text{-pebble game}\ \}\ .$$

The pebble complexity of a guarded command $\varphi \to t$ is:

$$\mathit{pebble}(\varphi \to t)\ =\ \max\ \{\ \mathit{pebble}(t)\ \} \cup \{\ \mathit{pebble}(u) \mid u \text{ is a term appearing in } \varphi\ \}\ .$$

The pebble complexity of an eds $S = (\ \varphi_i \to t_i \mid i \in I\ )$ is:

$$\mathit{pebble}(S)\ =\ \mathrm{lub}\ \{\ \mathit{pebble}(\varphi_i \to t_i) \mid i \in I\ \}\ .$$

We say that an eds S uses *bounded space* if $\mathit{pebble}(S) < \omega$. Otherwise, if $\mathit{pebble}(S) = \omega$, we say that S uses *unbounded space*. Let EDS denote the class of all effective definitional schemes. Let BOUNDED-EDS $\subset$ EDS be the subclass of eds's each restricted to use bounded space:

$$\text{BOUNDED-EDS}\ =\ \{\ S \in \text{EDS} \mid \mathit{pebble}(S) < \omega\ \}\ .$$

An eds $S = (\ \varphi_i \to t_i \mid i \in I\ )$ *unwinds* in a structure $\mathcal{M}$ if there is a finite approximation $\widetilde{S}$ of S, i.e. $\widetilde{S} = (\ \varphi_i \to t_i \mid i \in J\ )$ where J is a finite subset of I, such that $\widetilde{S}^{\mathcal{M}} = S^{\mathcal{M}}$. A structure $\mathcal{M}$ has the *unwind property* for a class $\mathcal{S}$ of eds's (or a class $\mathcal{P}$ of program schemes) if every $S \in \mathcal{S}$ (or every $S \in \mathit{eds}(\mathcal{P})$) unwinds in $\mathcal{M}$.

We define a particular first-order structure $\mathcal{N}$ with universe ω, based on the dag $\mathcal{B}$ of Section 4. We set:

$$\mathcal{N} =\ \langle \omega, =, g, 0 \rangle$$

[3]Our analysis is easily generalized to any program scheme with any finite number of input and output variables, not necessarily identical. Input and output variables are always ground variables; depending on the class of program schemes, higher-order variables may also be used but only during execution.

where $g : \omega \times \omega \to \omega$ is the function:

$$g(m,n) = \begin{cases} n+1, & \text{if } m = \lfloor n/2 \rfloor, \\ 0, & \text{otherwise.} \end{cases}$$

The correspondence between $\mathcal{N}$ and $\mathcal{B}$ is the following. For all natural numbers m, n, and p, if $p \neq 0$ then $g(m,n) = p$ iff there is an arrow from m to p and an arrow from n to p.

Using the correspondence between $\mathcal{N}$ and $\mathcal{B}$, Lemma 3, and the simulation of "memory locations" by "pebbles", the following is a straightforward result.

Lemma 8 *$\mathcal{N}$ has the unwind property for* BOUNDED-EDS.

Proof: Let $S \in$ BOUNDED-EDS and $S = (\ \varphi_i(x) \to t_i(x) \mid i \in I\ )$. Assume I is infinite, otherwise the lemma is trivial. Let T be the following set of first-order terms:

$$T \ = \ \{\ t_i(x) \mid i \in I\ \} \ \cup \ \{\ u(x) \mid u(x) \text{ appears in } \varphi_i(x),\ i \in I\ \}\ .$$

Because S uses bounded space, there is $n \geqslant 1$ such that $pebble(t) \leqslant n$ for every $t \in T$. Given an arbitrary $a \in \omega$, define the set $T^{\mathcal{N}}(a) \ = \ \{\ t^{\mathcal{N}}(a) \mid t \in T\ \} \ \subseteq \ \omega$. We view $dag(t)$ as prescribing the moves of a (finite) n-pebble game, where the initial configuration contains a single node (the value assigned to x); these are the moves (not necessarily unique) required to reach the output node of $dag(t)$ with at most n pebbles. By Lemma 3 and the correspondence between $\mathcal{N}$ and $\mathcal{B}$, the size of $T^{\mathcal{N}}(a)$ is finite and independent of a, namely

$$|T^{\mathcal{N}}(a)| \ \leqslant \ \mathbf{N}(\{a\}, \alpha(n)) \ \leqslant \ 2\alpha(n) + 2\ .$$

(Review the definition of $\mathbf{N}(C,k)$ in Section 4 to understand the factor 2 in "$2\alpha(n)$".) Put differently, because $\mathcal{B}$ is uniformly locally finite w.r.t. bounded space, at most $2\alpha(n)$ nodes are reached in the n-pebble game played according to $dag(t)$, for all $t \in T$ and all initial configuration $C = \{a\}$. As there are finitely many non-isomorphic partial substructures of $\mathcal{N}$ of size $\leqslant\ 2\alpha(n)$, the desired conclusion follows. (We say "partial" substructures because we do not require that their universe be closed under the function g.) *Q.E.D.*

Theorem 9 *There is an eds $\widehat{S} \in$ EDS not equivalent to any $S \in$ BOUNDED-EDS over $\mathcal{N}$.*

Proof: By Lemma 8, it suffices to define an eds $\widehat{S}$ that does not unwind in $\mathcal{N}$. Let $t_0, t_1, t_2, \ldots$ be an infinite sequence of closed first-order terms such that $t_0^{\mathcal{N}} = 0$, $t_1^{\mathcal{N}} = 1$, $t_2^{\mathcal{N}} = 2$, etc. The desired $\widehat{S}$ is $(\ x = t_i \to t_{i+1} \mid i \in \omega\ )$. $\widehat{S}^{\mathcal{N}}$ defines the successor function on ω. *Q.E.D.*

Theorem 9 generalizes the Paterson-Hewitt result, which says that the class FC of flowchart schemes is strictly weaker than the class REC of recursive schemes. If $P \in$ FC then $eds(P) \in$ BOUNDED-EDS and, therefore, P unwinds in $\mathcal{N}$. On the other hand, it is a straightforward programming exercise to write a $R \in$ REC such that $R^{\mathcal{N}}$ computes the successor function on ω and, therefore, R does not unwind in $\mathcal{N}$.[4]

The analogue of the Paterson-Hewitt result for logics of programs appeared in [18] and [2, 3], which asserts that the "first-order logic of FC" is less expressive than the "first-order logic of REC".

If $\mathcal{P}$ is a class of program schemes, the "first-order logic of $\mathcal{P}$" is denoted by $L(\mathcal{P})$. Briefly, $L(\mathcal{P})$ is conventional first-order logic to which we add a construct $\langle P\rangle\theta$ with the meaning that "there is an execution of program scheme $P \in \mathcal{P}$ after which the assertion θ holds". Hence, the symbols of $L(\mathcal{P})$ are the usual symbols of first-order logic (including function symbols, relation symbols, equality, quantifiers, etc.) in addition to the modal operator $\langle\ \rangle$ and the symbols of the programming formalism $\mathcal{P}$. (Another modal operator $[\ ]$ is usually considered also, but we can take $[\ ]$ as an abbreviation for $\neg\langle\ \rangle\neg$, just as $\forall$ is an abbreviation for $\neg\exists\neg$.) Further details on "logics of programs" can be found in [5] and [8].

Let $\mathcal{P}$ and $\mathcal{P}'$ be classes of program schemes. We say that the logic $L(\mathcal{P})$ is *reducible* to the logic $L(\mathcal{P}')$, in symbols $L(\mathcal{P}) \leqslant L(\mathcal{P}')$, iff for every formula θ in $L(\mathcal{P})$ there is a formula θ' in $L(\mathcal{P}')$ such that θ and θ' are equivalent in all interpretations. We write $L(\mathcal{P}) < L(\mathcal{P}')$ if $L(\mathcal{P}) \leqslant L(\mathcal{P}')$ and $L(\mathcal{P}') \nleqslant L(\mathcal{P})$.

The preceding definitions extend in the obvious way to classes of eds's. The logics L(EDS) and L(BOUNDED-EDS) are therefore well-defined, and it is meaningful to compare them.

Theorem 10 L(BOUNDED-EDS) $<$ L(EDS), *i.e.* L(BOUNDED-EDS) *is strictly less expressive than* L(EDS).

Proof: Since EDS (as well as BOUNDED-EDS) is closed under substitution, it is a trivial exercise to write an eds that uses the eds $\widehat{S}$ of Theorem 9, namely $(\ x = t_i \rightarrow t_{i+1} \mid i \in \omega\ )$, and defines the addition relation $x + y = z$ in $\mathcal{N}$. Similarly, multiplication in $\mathcal{N}$ is definable by an eds. As a matter of fact, all these eds's are very simple. Clearly then, the relations definable by the logic L(EDS) in $\mathcal{N}$ are exactly the arithmetical relations (we have shown the "at least" part, which is sufficient for our argument, but the other direction is simple too using the fact that, by definition, every eds is a r.e. sequence of guarded commands).

On the other hand, because every $S \in$ BOUNDED-EDS unwinds in $\mathcal{N}$, every formula of the logic L(BOUNDED-EDS) is equivalent to a first-order formula in $\mathcal{N}$. Moreover, 0 and 1 are first-order definable (trivially) in the structure $\langle\omega, =, +, \mathrm{succ}, 0\rangle$ of Presburger arithmetic, and so is our function $g : \omega \times \omega \rightarrow \omega$, because

$$g(x, y) = z \quad \Longleftrightarrow \quad (\ (x + x = y\ \vee\ x + x = y + 1)\ \wedge\ y + 1 = z\ )$$

Hence, the relations that are first-order definable in our structure $\mathcal{N}$ are also definable in Presburger arithmetic, and these do not include all arithmetical relations. *Q.E.D.*

[4] Such a recursive scheme R is given in Section 3 of [7], where it is called SUCC.

7 Conclusion

Theorem 10 is our version of the Tiuryn-Erimbetov theorem. It implies several well-known results in first-order dynamic logic, such as Rec-DL > CF-DL, Array-DL > DL, Random-DL > DL, and the deterministic version of each of these inequalities. These results follow from the fact that the schemes used in DL or CF-DL have only finitely many memory locations each, and can therefore be simulated by pebble games with finitely many pebbles. While leading to the same results, our analysis is significantly shorter and more straightforward.

Our definition of a guarded command allows quantifier-free formulas only. Allowing *arbitrary* first-order formulas to appear in guarded commands leads to the definition of "elementary EDS", or EEDS. The bounded-space version of this class, BOUNDED-EEDS, imposes a bound on the quantifier-depth of formulas, in addition to bounding the pebble complexity of terms. These classes are somewhat exotic, as it is questionable whether they continue to reflect intuition of effective computations.

Nonetheless, all our results, including Theorem 10, can be proved for elementary schemes and logics. For instance, to prove Lemma 8, observe that in a first-order formula of pebble complexity n, each quantifier can be bounded by the distance $2\alpha(n)$ from the preceding variables. Therefore, any bounded-space elementary eds, with a bound k on quantifier-depth, is equivalent to a bounded-space eds (already quantifier-free) with $2k\alpha(n)$ new free variables *under a substitution that replaces the new free variables with expressions of the form* $x+i$ *or* $x-i$*, for some old free variable* x *and a constant* $i \leqslant 2k\alpha(n)$. This new eds unwinds in $\mathcal{N}$, and this implies that our original elementary eds does.

References

[1] Ben-Or, M., and Cleve, R., "Computing algebraic formulas using a constant number of registers", *Proc. of 20-th ACM Symp. on Theory of Computing*, pp 254–257, May 1988.

[2] Erimbetov, M.M., "On the expressive power of programming logics", in *Research in Theoretical Programming* (M.A. Taitslin, Ed.), pp 49–68, Alma-Ata, 1981 (in Russian).

[3] Erimbetov, M.M., "A fragment of a logic with infinite formulas", in *Research in Theoretical Programming* (M.A. Taitslin, Ed.), pp 30–48, Alma-Ata, 1981 (in Russian).

[4] Dijkstra, E.W., *A Discipline of Programming*, Prentice Hall, 1976.

[5] Harel, D., *First-Order Dynamic Logic*, LNCS, **68**, Springer-Verlag, 1979.

[6] Kalyasundaram, B., and Schnitger, G., "On the power of white pebbles", *Proc. of 20th ACM Symp. on Theory of Computing*, pp 258–266, May 1988.

[7] Kfoury, A.J., "Definability by programs in first-order structures", *Theoretical Computer Science*, **25**, pp 1–66, 1983.

[8] Kfoury, A.J., Stolboushkin, A.P., and Urzyczyn, P., "Some open problems in the theory of program schemes and dynamic logics", *Russian Mathematical Surveys*, **44**:1, pp 43–68, 1989. (English transl. of Russian original)

[9] Kozen, D., "Pebblings, edgings, and equational logic", *Proc. 16th ACM Symp. on Theory of Computing*, pp 428–435, 1984.

[10] Kozen, D., and Tiuryn, J., "Logics of Programs", in *Handbook of Theoretical Computer Science, Vol. B, Formal Methods and Semantics*, ed. J. van Leeuven, Elsevier Science Publ. and The MIT Press, pp 789–840, 1990.

[11] I.Kh. Musikaev and M.A. Taitslin, "Limitations of the program memory and the expressive power of dynamic logics", *Information and computation*, **103**:2, pp 195–203, 1993.

[12] Paterson, M.S., and Hewitt, C., "Comparative schematology", *MIT A.I. Lab Technical Memo No. 201* (also in *Proc. of Project MAC Conference on Concurrent Systems and Parallel Computation*), 1970.

[13] Pippenger, N., "Pebbling with an auxiliary pushdown", *J. Computer and System Sciences*, **23**, pp 151–165, 1981.

[14] Pippenger, N., "Advances in Pebbling", *Proc. of 9th ICALP*, LNCS no. 140, Springer-Verlag, 1982.

[15] Salomaa, A., "On the index of a context-free grammar and language", *Information and Control*, **14**, pp 474–477, 1969.

[16] Salomaa, A., *Formal Languages*, Academic Press, 1973.

[17] Savage, J., and Vitter, J., "Parallelism in space-time tradeoffs", *Proc. of International Workshop on VLSI: Algorithms and Architecture*, Amalfi, Italy, 1984.

[18] Tiuryn, J., "Unbounded program memory adds to the expressive power of first-order programming logic", *Information and Control*, **60**, nos. 1-3, pp 12–35, 1984. (An earlier version appeared in *Proc. of 22nd Symp. on Foundations of Comp. Science*, 1981.)

[19] Tiuryn, J., "Implicit definability of finite binary trees by sets of equations", in *Logic and Machines: Decision Problems and Complexity*, eds. Börger, Hasenjaeger, and Rödding, pp 320–332, LNCS no. 171, Spinger-Verlag, 1981.