On the power of built-in relations in certain classes of program schemes

S.R. Chauhan [a,1] and I.A. Stewart [b,2]

[a] *Department of Computer Science, University of Wales Swansea, Swansea SA2 8PP, U.K.*

[b] *Department of Mathematics and Computer Science, Leicester University, Leicester LE1 7RH, U.K.*

Key words: Computational complexity. Descriptive complexity. Finite model theory. Program Schemes.

1 Introduction

One of the fundamental results of finite model theory is Immerman's characterization [10,11] of the complexity class NL (non-deterministic logspace) as the class of problems definable in transitive closure logic augmented with a built-in successor relation, $(\pm\mathrm{TC})^*[\mathrm{FO}_s]$. Essentially, by "built-in successor relation" we mean that any sentence of the logic has access to a successor relation on the universe of the (finite) structure in which it is interpreted (and where this successor relation does not necessarily come as part of the structure). So that our sentences should define problems, i.e., sets of structures closed under isomorphism, we only ever consider the invariant sentences; that is, those sentences with the property that every (appropriate) structure either satisfies the sentence no matter which successor relation is chosen or it doesn't satisfy the sentence no matter which successor relation is chosen. Unfortunately, the property of being an invariant sentence is not decidable, even for first-order logic with a built-in successor relation [7], and so one should not really say that one has a truly "logical" characterization of NL. The general question of whether complexity classes such as NL and P (polynomial-time)

[1] Supported by EPSRC Ph.D. Studentship 93317145.

[2] Partially supported by British-German ARC Project 604 and EPSRC Grant GR/H 81108. Much of the work in this paper was done whilst the author was at the University of Wales Swansea.

actually possess "logical" characterizations is the subject of much research in finite model theory (see, for example, [5] and [13]).

Immerman proved yet more: he actually produced a normal form for the sentences of $(\pm\mathrm{TC})^*[\mathrm{FO}_s]$. As observed in [14], this normal form result can be re-interpreted as a result about the expressive power of a very simple class of program schemes in the presence of a built-in successor relation. A program scheme of NPS takes a finite structure as input and is essentially a non-deterministic while-program with a constant number of variables, these variables taking values from the universe of the input structure, and where the tests in any while-instruction are quantifier-free first-order formulae (these are "poor tests", in the parlance of [12]; as opposed to "rich tests" which are first-order formulae). If we allow the program schemes of NPS to additionally have access to a built-in successor relation then we denote the resulting class of program schemes by NPS(*succ*). Re-interpreting Immerman's result yields a characterization of NL as the class of problems accepted by the program schemes of NPS(*succ*) (a similar result, though in a different context, was derived earlier in [8]).

The question arises as to whether this very simple class of program schemes NPS can be augmented with other different built-in relations so as to yield a characterization of NL; and it is this question that provides the initial motivation for this paper (other studies arising from similar motivations can be found in, for example, [6] and [15]). Our choice of other built-in relations to consider is influenced by what has been studied already in other contexts in finite model theory. In particular, in [1] it is proven that first-order logic augmented with the built-in relations $\leq$, a linear order, and BIT, a binary relation such that $BIT(i, j)$ holds iff "the ith bit of the binary representation of the natural number j is 1", characterizes the complexity class AC^0. As remarked in, for example, [2], this logic is equivalent to the extension of first-order logic with the built-in binary relation $\leq$ and the built-in ternary relations $+$, an addition, and $\times$, a multiplication. Hence, the four built-in relations we shall consider are *succ*, $\leq$, $+$ and $\times$. In fact, we completely classify the relative expressibilities of the program schemes of NPS augmented with these built-in relations, as is depicted in Fig. 1 (a bold line joining two classes indicates that the upper class is more expressive than the lower class, and a dashed line that the classes are incomparable).

Whilst, admittedly, the scenario described above is rather esoteric, it turns out that as well as the above classification, we can actually obtain analogous classifications for both bounded-variable infinitary logic, $\mathcal{L}^{\omega}_{\infty\omega}$, and its existential fragment, $\exists\mathcal{L}^{\omega}_{\infty\omega}$. The logic $\mathcal{L}^{\omega}_{\infty\omega}$ features widely in finite model theory: for instance, transitive closure logic, least fixed point logic and partial fixed point logic are all fragments of $\mathcal{L}^{\omega}_{\infty\omega}$ (see, for example, [5]). The classification for extensions of $\exists\mathcal{L}^{\omega}_{\infty\omega}$ is exactly that in Fig. 1 with "NPS" replaced by "$\exists\mathcal{L}^{\omega}_{\infty\omega}$",

and "NL =" omitted. The classification for $\mathcal{L}^{\omega}_{\infty\omega}$ is as follows.

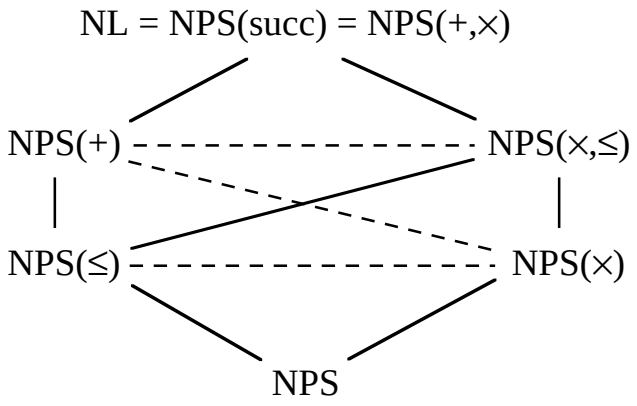

Figure 1. How the classes of program schemes relate.

Theorem 1 $\mathcal{L}^{\omega}_{\infty\omega} \subset \mathcal{L}^{\omega}_{\infty\omega}(\times) \subset \mathcal{L}^{\omega}_{\infty\omega}(succ) = \mathcal{L}^{\omega}_{\infty\omega}(+) = \mathcal{L}^{\omega}_{\infty\omega}(\leq)$.

Our classifications follow from Proposition 5 where our basic methodology is to exhibit problems in certain program-scheme classes which are not definable in certain infinitary-logic classes, and to use the simple observation that a program-scheme class is a fragment of the analogous infinitary-logic class. In order to prove our inexpressibility results for $\mathcal{L}^{\omega}_{\infty\omega}$, we use well established pebble games. Let us remark that if we allow first-order tests in the program schemes of NPS, i.e.," rich tests", then we obtain a result identical to Theorem 1 except with "NPS" replacing "$\mathcal{L}^{\omega}_{\infty\omega}$".

2 Program schemes and logics

The reader is referred to [5] for details of any finite model theoretic concepts and notions not covered here. Any program scheme is defined to be over some fixed signature, with a *signature* τ being a finite tuple $\langle R_1, \ldots, R_r, C_1, \ldots, C_c \rangle$, where each R_i is a relation symbol, of some fixed positive arity d_i, and where each C_j is a constant symbol (function symbols are not allowed in our signatures). A *program scheme* $\rho \in \mathrm{NPS}(\tau)$ is a finite sequence of instructions of one of the following types):

- *atoms* consist of: the variables $\{x_i : i = 1, 2, \ldots\}$; the constant symbols of τ; and the constant symbols 0 and *max* (which we ensure do not occur in any signature)
- *assignment instructions* are of the form:
  - $var := atom$, where *var* denotes some variable and *atom* some atom
  - $guess(var)$, where *var* denotes some variable

- *test instructions* are of the form WHILE t DO $i_1, i_2, \ldots, i_k$ OD, for some $k \geq 0$ and instructions $i_1, i_2, \ldots, i_k$, where t is a boolean combination of *boolean tests*, or their negations, of the form:
  - $y_1 = y_2$, where y_1 and y_2 are atoms
  - $R(y_1, y_2, \ldots, y_d)$, where R is a d-ary relation symbol of τ and $y_1, y_2, \ldots, y_d$ are atoms
- *input/output instructions* are of the form $input(x_1, x_2, \ldots, x_m)$ and $output(x_1, x_2, \ldots, x_m)$, where $x_1, x_2, \ldots, x_m$ are the input/output variables.

The first (resp. last) instruction of any program scheme is the input (resp. output) instruction, and any program scheme has exactly one input and one output instruction. Also, the input/output variables are exactly those variables involved in any program scheme. Without loss of generality, we may assume that there is available an IF ... THEN ... (ELSE ...) FI instruction (see [14]).

A program scheme over the signature $\tau = \langle R_1, \ldots, R_r, C_1, \ldots, C_c \rangle$ takes as input a *finite structure S over τ*, or *τ-structure*, where $S = \langle \{0, 1, \ldots, n-1\}, R_1^S, \ldots, R_r^S, C_1^S, \ldots, C_c^S \rangle$, with $|S| = \{0, 1, \ldots, n-1\}$ the *universe* of S, and with each $R_i^S \subseteq |S|^{d_i}$ and each $C_j^S \in |S|$: we also write $|S|$ to denote the *size* n of S (this causes no confusion; and nor does our choice of leaving out superscripts denoting the structure in which a relation symbol, etc., is interpreted). We denote the class of all structures over some signature τ by STRUCT(τ). The interpretation of some program scheme ρ over τ in a τ-structure S of size n should be obvious except that we require: that the initial values (from $|S|$) of the input/output variables are given; that the constant symbols 0 and max are interpreted arbitrarily but differently in S (we assume all structures have size at least 2); and that any instruction $guess(x_i)$ nondeterministically assigns some value of $|S|$ to the variable x_i. We say that ρ *accepts* S, and write $S \models \rho$ if, and only if, with the input/output variables initially all set at 0, there exists a computation of ρ on input S such that ρ halts with the input/output variables all set at max.

As things stand, acceptance of a structure by a program scheme depends upon the interpretation of 0 and max, and as such is clearly unacceptable. We could remedy this situation by insisting that every structure S always comes with its own constants 0 and max; but this would force us to consider natural objects like graphs unnaturally, i.e., by supplying two additional distinguished vertices. We remedy this discrepancy by only ever considering certain program schemes. Henceforth, we only consider program schemes ρ for which the following is true: for every τ-structure S (where τ is the underlying signature of ρ) and for every $0, 0', max, max' \in |S|$ for which $0 \neq max$ and $0' \neq max'$, $(S, 0, max) \models \rho$ iff $(S, 0', max') \models \rho$. That is, we only ever consider program schemes with 2 *built-in constants*. Consequently, the set of structures accepted by a program scheme ρ constitute a *problem*; that is, a set of finite structures, over some fixed signature, which is closed under isomorphism. We write NPS

to denote the class of program schemes $\{\rho \in \mathrm{NPS}(\tau) : \tau \text{ some signature}\}$ (with the above proviso on each ρ), and also write NPS to denote those problems accepted by some program scheme of NPS (the same goes for other classes of program schemes defined later).

Just as we augmented program schemes with two built-in constants, so we can augment them with built-in relations. For example, to say that a program scheme has a *built-in successor relation* is to say that a program scheme has available: a binary relation *succ* that is always interpreted as a successor relation on the universe of an input structure, i.e., as a relation of the form $\{(u_0, u_1), (u_1, u_2), \ldots, (u_{n-2}, u_{n-1})\}$, where the input structure has size n; and also two constant symbols 0 and *max* that are always interpreted as the least and greatest elements of the successor relation, i.e., as u_0 and u_{n-1}, respectively. Just as before, we only ever consider those program schemes with a built-in successor relation which define problems, i..e, those program schemes ρ, over the signature τ, such that for every τ-structure S and for every pair of successor relations *succ* and *succ*$'$ (with associated "least" and "greatest constants"), $(S, succ, 0, max) \models \rho$ iff $(S, succ', 0', max') \models \rho$. We denote the class of problems accepted by program schemes with a built-in successor relation as NPS(*succ*).

A *built-in linear-order* $\leq$ is defined as was a built-in successor, and we write $\leq (u, v)$ as $u \leq v$; a *built-in addition* is defined by allowing a program scheme access to a ternary relation $+$ that is always interpreted as an "addition" on the universe of some input structure, and we write $+(u, v, w)$ as $u + v = w$; and a *built-in multiplication* is defined by allowing a program scheme access to a ternary relation $\times$ that is always interpreted as a "multiplication" on the universe of some input structure, and we write $\times(u, v, w)$ as $u \times v = w$. The built-in constants 0 and *max* are the least and greatest elements of the linear order $\leq$, addition $+$ and multiplication $\times$, respectively.

We use pebble games to prove our separation results. Let τ be some signature. The *k-pebble infinitary game* on the τ-structures S and T is played by two players, Spoiler and Duplicator, as follows. The board consists of the two structures S and T, and there are k pairs of pebbles $\{p_1, q_1\}, \{p_2, q_2\}, \ldots, \{p_k, q_k\}$. A move of the game consists of Spoiler picking up some pebble p_i (resp. q_i), which may or may not have previously been played, and placing it on some element of $|S|$ (resp. $|T|$): then Duplicator picks up pebble q_i (resp. p_i) and places it on some element of $|T|$ (resp. $|S|$). Spoiler wins after a move has been made if the substructure of S induced by the elements upon which pebbles are currently placed (and any constants) is not isomorphic to the substructure of T induced by the elements upon which pebbles are currently placed, where the isomorphism is that given by mapping the element of $|S|$ upon which pebble p_i is currently placed (if indeed it has been so placed) to that of $|T|$ upon which pebble q_i is currently placed and mapping a constant of S to the correspond-

ing constant of T (also, if this mapping is not well-defined then Spoiler wins). The game may be infinite. We say that Duplicator *has a winning strategy* in the k-pebble infinitary game on S and T if no matter how Spoiler moves, Duplicator can prolong the game indefinitely (we prefer to leave the notion of a "winning strategy" intuitive rather than define it rigorously).

The k-pebble infinitary game classifies definability in the logic $\mathcal{L}^k_{\infty\omega}$, which is defined as follows: $\mathcal{L}^k_{\infty\omega}$ is the fragment of the logic $\mathcal{L}_{\infty\omega}$ with at most k variables, where the logic $\mathcal{L}_{\infty\omega}$ is first-order logic where infinite conjunctions and disjunctions may be formed. *Bounded-variable infinitary logic* is defined as $\mathcal{L}^\omega_{\infty\omega} = \cup_k \mathcal{L}^k_{\infty\omega}$. The existential fragment, $\exists\mathcal{L}^\omega_{\infty\omega}$, of $\mathcal{L}^\omega_{\infty\omega}$ consists of all those formulae not involving the universal quantifier $\forall$. It is easy to see that NPS $\subseteq \exists\mathcal{L}^\omega_{\infty\omega}$ (and also in the presence of built-in relations).

Theorem 2 [3,9] *Let τ be some signature and let S and T be τ-structures. The Duplicator has a winning strategy in the k-pebble infinitary game on S and T iff S and T agree on all sentences of $\mathcal{L}^k_{\infty\omega}$. Moreover, the Duplicator has a winning strategy in the restricted k-pebble infinitary game on S and T where Spoiler only ever plays in S iff S and T agree on all sentences of $\exists\mathcal{L}^k_{\infty\omega}$.*

We close this section with some historical remarks. Program schemes were extensively studied in the seventies, without much regard being paid to resources, before a closer complexity analysis was undertaken in, mainly, the eighties (see [12] and the references therein). In the late eighties, program schemes were developed to work on finite structures [14], mindful of advances in descriptive complexity theory. We feel that the links between program schemes, logics for programs, dynamic logic, etc., and the logics studies in finite model theory, particularly those involving generalized quantifiers, have not been considered in the past as fully as they might have been, and that this is an interesting prospective area of research.

3 The basic results

Let the signature $\tau = \langle U, E, C, D\rangle$, where U is a unary relation symbol, E is a binary relation symbol and C and D are constant symbols. Define the problem BP as consisting of all those τ-structures such that there is a path in the digraph described by E from vertex C to vertex D of length at most ν, where $\nu = |\{u : U(u) \text{ holds}\}|$.

Theorem 3 *BP $\in$ NPS$(\leq) \setminus \mathcal{L}^\omega_{\infty\omega}(\times)$.*

PROOF. The following program scheme $\rho \in$ NPS$(\leq)$ accepts BP:

```
input(x,y,u,v)
IF C ≠ D THEN
   guess(u)
   WHILE ¬U(u) DO guess(u) OD
   guess(x)
   WHILE ¬E(C,x) DO guess(x) OD
   WHILE x ≠ D DO
      guess(v)
      WHILE ¬U(v) ∨ v ≤ u DO guess(v) OD
      guess(y)
      WHILE ¬E(x,y) DO guess(y) OD
      u := v
      x := y
      OD
   FI
x, y, u, v := max
output(x,y,u,v)
```

Suppose that BP is definable by a sentence $\psi \in \mathcal{L}^k_{\infty\omega}(\times)$, for some k.

Lemma 4 *For every k, there exists an n such that there are at least k distinct primes greater than $n/2$ and less than n.*

PROOF. From Tchebychev's Theorem [4, p.55], if $\Pi(x)$ denotes the number of primes not exceeding x then:

$$0.92\frac{x}{\ln x} \leq \Pi(x) \leq 1.22\frac{x}{\ln x},$$

for all sufficiently large x, and so $\Pi(2x) - \Pi(x) \to \infty$ as $x \to \infty$. $\Box$

Let n be such that $\{p_1, p_2, \ldots, p_{2k+2}\}$ is a set of distinct primes, each of which is greater than $n/2$ and less than n: given k, such an n exists by Lemma 4. Define the τ-structure $S = \langle \{0, 1, \ldots, n-1\}, U^S, E^S, C^S, D^S \rangle$ as follows:

$$U^S = \{p_1, p_2, \ldots, p_{k+1}\}, E^S = \{(i, i+1) : i = 0, 1, \ldots, k+1\}$$
$$C^S = 0, D^S = k+1.$$

Let T be defined exactly as was S except that $p_{k+1} \notin U^T$. Note that $S \in$ BP but $T \notin$ BP. Let the relation symbol $\times$ and the constant symbols 0 and max be interpreted as the natural multiplication on $|S| = |T|$, $0 \in |S| = |T|$ and $n-1 \in |S| = |T|$, respectively. By hypothesis, $(S, \times, 0, max) \models \psi$ and $(T, \times, 0, max) \not\models \psi$.

Consider the following winning strategy for Duplicator in the k-pebble Ehrenfeucht-Fraïssé game on $(S, \times, 0, max)$ and $(T, \times, 0, max)$. Write $V^S = \{p_{k+2}, p_{k+3}, \ldots, p_{2k+2}\}$, $V^T = \{p_{k+1}, p_{k+2}, \ldots, p_{2k+2}\}$, $W^S = |S| \setminus (U^S \cup V^S)$ and $W^T = |T| \setminus (U^T \cup V^T)$.

- If Spoiler plays on an unpebbled element in U^S (resp. U^T, V^S, V^T) then Duplicator plays on an unpebbled element in U^T (resp. U^S, V^T, V^S).
- If Spoiler plays on a pebbled element in U^S (resp. U^T, V^S, V^T) then Duplicator plays on the corresponding pebbled element in U^T (resp. U^S, V^T, V^S).
- If Spoiler plays in W^S (resp. W^T) then Duplicator plays on the element of the same name in W^T (resp. W^S).

Theorem 2 yields a contradiction. □

Proposition 5 *(i)* $NPS(succ) = NPS(+, \times) (= NL)$ *and* $\exists\mathcal{L}^{\omega}_{\infty\omega}(succ) = \exists\mathcal{L}^{\omega}_{\infty\omega}(+, \times)$.
(ii) $NPS(\leq) \subseteq NPS(+)$ *and* $\exists\mathcal{L}^{\omega}_{\infty\omega}(\leq) \subseteq \exists\mathcal{L}^{\omega}_{\infty\omega}(+)$.
(iii) There are problems in $NPS(succ)$ *which are not in* $\exists\mathcal{L}^{\omega}_{\infty\omega}(\leq)$.
(iv) There are problems in $NPS(+)$ *which are not in* $\exists\mathcal{L}^{\omega}_{\infty\omega}(\leq)$.
(v) There are problems in $NPS(\times)$ *which are not in* $\exists\mathcal{L}^{\omega}_{\infty\omega}(+)$.
(vi) There are problems in $NPS(+)$ *which are not in* $\exists\mathcal{L}^{\omega}_{\infty\omega}(\leq, \times)$.

PROOF. (*Sketch*)
(i) and (ii) Simple exercises.
(iii) and (iv) The problem ODD consisting of all those structures of odd size over the empty signature τ_ϵ is in NPS($succ$) and NPS(+). A simple Ehrenfeucht-Fraïssé game on two sufficiently large τ_ϵ-structures S_n and S_{n+1}, the first of odd size n and the second of even size $n+1$, yields that ODD $\notin \exists\mathcal{L}^{\omega}_{\infty\omega}(\leq)$ (take the natural linear orders in S_n and S_{n+1}, and let Duplicator's winning strategy be given by the function $f : |S_n| \to |S_{n+1}|$ defined as $f : i \mapsto i$, for all $i \in \{0, 1, \ldots, n-2\}$, and $f(n-1) = n$).
(v) The problem PERF1 consisting of all those structures over the signature τ_ϵ whose size is 1 plus a perfect square is in NPS($\times$). Let p be a sufficiently large even number. A simple Ehrenfeucht-Fraïssé game on two τ_ϵ-structures S_{p^2+1} and S_{2p^2+1} of sizes p^2+1 and $2p^2+1$, respectively, yields that PERF1 $\notin \exists\mathcal{L}^{\omega}_{\infty\omega}(+)$ (take the natural additions in S_{p^2+1} and S_{2p^2+1}, and let Duplicator's winning strategy be given by the function $f : |S_{p^2+1}| \to |S_{2p^2+1}|$ defined as $f : i \mapsto 2i$).
(vi) The problem ODD is in NPS(+). A simple Ehrenfeucht-Fraïssé game on two τ_ϵ-structures S_3 and S_4 of sizes 3 and 4, respectively, yields that any problem in $\exists\mathcal{L}^{\omega}_{\infty\omega}(\times)$ which contains S_3 must contain S_4 (take the natural multiplications in S_3 and S_4 and the natural linear orders in S_3 and S_4, and let Duplicator's winning strategy be given by the function $f : |S_3| \to |S_4|$

defined as $f(0) = 0$, $f(1) = 1$ and $f(2) = 3$.) $\square$

References

[1] D.A.M. Barrington, N. Immerman and H. Straubing, On uniformity within NC^1, *J. Comput. System Sci.* **41** (1990) 274–306.

[2] D.M. Barrington and N. Immerman, Time, hardware and uniformity, *Proc. 9th IEEE Ann. Conf. on Structure in Complexity Theory*, IEEE Press (1994) 176–185.

[3] J. Barwise, On Moschovakis closure ordinals, *J. Symbolic Logic* **42** (1977) 292–296.

[4] H. Davenport, *Multiplicative Number Theory*, Springer (1980).

[5] H.D. Ebbinghaus and J. Flum, *Finite Model Theory*, Springer-Verlag (1995).

[6] K. Etessami and N. Immerman, Reachability and the power of local ordering, *Theoret. Comp. Sci.* **148** (1995) 227–260.

[7] Y. Gurevich, Logic and the challenge of computer science, in: *Current Trends in Theoretical Computer Science* (ed. E. Börger), Computer Science Press (1988) 1–57.

[8] D. Harel and D. Peleg, On static logics, dynamic logics, and complexity classes, *Inf. Control* **60** (1984) 86-102.

[9] N. Immerman, Upper and lower bounds for first-order expressibility, *J. Comput. System Sci.* **25** (1982) 76–98.

[10] N. Immerman, Languages which capture complexity classes, *SIAM J. Comput.* **16** (1987) 760–778.

[11] N. Immerman, Nondeterministic space is closed under complementation, *SIAM J. Comput.* **17** (1988) 935–938.

[12] D. Kozen and J. Tiuryn, Logics of programs, in: *Handbook of Theoretical Computer Science Vol. B* (ed. J. van Leeuwen), Elsevier (1990) 789–840.

[13] M. Otto, Bounded variable logics and counting, *Lecture Notes in Logic Vol.* 9, Springer-Verlag (1997).

[14] I.A. Stewart, Logical and schematic characterization of complexity classes, *Acta Informat.* **30** (1993) 61–87.

[15] H. Straubing, *Finite Automata, Formal Logic and Circuit Complexity*, Birkhäuser (1994).