Circuits and Context-Free Languages

Pierre McKenzie[1], Klaus Reinhardt[2], and V. Vinay[3]

[1] Informatique et recherche opérationnelle, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal (Québec), H3C 3J7 Canada. Research performed while on leave at Universität Tübingen. mckenzie@iro.umontreal.ca
[2] Wilhelm-Schickard Institut für Informatik, Universität Tübingen, Sand 13, D-72076 Tübingen, Germany. reinhard@informatik.uni-tuebingen.de
[3] Department of Computer Science and Automation, Indian Institute of Science Bangalore-560 012 India. vinay@csa.iisc.ernet.in

Abstract. Simpler proofs that DAuxPDA-TIME(polynomial) equals LOG(DCFL) and that SAC^1 equals LOG(CFL) are given which avoid Sudborough's multi-head automata [Sud78]. The first characterization of LOGDCFL in terms of polynomial proof-tree-size is obtained, using circuits built from the multiplex select gates of [FLR96]. The classes L and NC^1 are also characterized by such polynomial size circuits: "self-similar" logarithmic depth captures L, and bounded width captures NC^1.

1 Introduction

The class LOGCFL (an NL machine equipped with an auxiliary pushdown) occupies a central place in the landscape of parallel complexity classes. LOGCFL sits between two interesting classes, NL and AC^1: NL is often viewed as the space analog of NP, and AC^1 characterizes the problems solvable on a PRAM in $O(\log n)$ time using a polynomial number of processors. As a class, LOGCFL has attracted a lot of attention due to its seemingly "richer structure" than that of NL. Sudburough [Sud78] showed that LOG(CFL) (logspace closure of CFL) was in fact LOGCFL by using multi-head pushdown automata. Then, Ruzzo [Ruz80] characterized LOGCFL alternating Turing machines. This was followed by Venkateswaran's surprising semi-unbounded circuits characterization [Ven91]: this clean circuit characterization of LOGCFL stands in contrast to its somewhat "messy" machine definitions. Then came a flurry of papers, for example showing closure of LOGCFL under complementation [BCD$^+$88], characterizing LOGCFL in terms of groupoids [BLM93], linking it to depth reduction [Vin96,AJMV98], and studying the descriptive complexity of LOGCFL [LMSV99].

At the same time, progress was made in understanding LOGDCFL. The initial results of Sudburough ([Sud78], discussed below) were followed by Cook [Coo79] who showed that LOGDCFL is contained in deterministic space $O(\log^2 n)$. Dymond and Ruzzo [DR86] drew an important equivalence between between LOGDCFL and owner-write PRAM models. A variety of papers have since extended these results [FLR96,MRS93].

As mentioned, Sudburough proved that DAuxPDA-TIME(pol) $\subseteq$ LOG(DCFL) and that NAuxPDA-TIME(pol) $\subseteq$ LOG(CFL) by reducing the numbers of heads

T. Asano et al. (Eds.): COCOON'99, LNCS 1627, pp. 194–203, 1999.

[Iba73,Gal74] in the multi-head automata characterizing the pushdown classes D(N)AuxPDA-TIME(pol) [Har72]. In the light of the many advancements since, it is natural to wonder whether simpler proofs cannot be given. Indeed, we exhibit simpler and more intuitive proofs here.

In Section 3 we show how a standard nondeterministic pushdown automaton (with no auxiliary work tape) can evaluate a suitably represented semi-unbounded circuit. This establishes $\mathrm{SAC}^1 \subseteq \mathsf{LOG(CFL)}$ in a direct way. As in [Rei90], where alternating pushdown automata were considered, this direct route from circuits to AuxPDAs exploits the fact that the work of a PDA with an auxiliary work tape can be separated into a logspace transduction followed by a pure pushdown part.

In Section 4 we extend our strategy to circuits built from *multiplex select* gates. These gates (defined precisely in Section 2) have as inputs a sequence of *data* wire bundles and a single *selection* wire bundle whose setting identifies the input data bundle to be passed on as gate output.

Multiplex select gates were introduced to better reflect the difficulty of speeding up parallel computations [Rei97], and they turned up again in an incisive characterization of the power of Dymond and Ruzzo's owner-write PRAMs [FLR96]. Here we observe that a natural notion of *proof tree size* exists for multiplex circuits. Using this notion and further observations, we simplify the harder direction (common to [DR86] and [FLR96]) in the proof that CROW-PRAMs efficiently simulate AuxDPDAs. Coupled with an application of our strategy from Section 3, this implies that LOGDCFL is characterized by polynomial size multiplex circuits having polynomial size proof trees. Proof tree size characterizations are known for LOGCFL [Ruz80,Ven91], but this is the first such characterization of LOGDCFL.

Multiplex select gates are thus intriguing: designing a circuit from such gates, rather than from the more usual boolean gates, forces one to pay closer attention to *routing* aspects which (amazingly) precisely capture the owner-write restriction of a PRAM or the deterministic restriction of an AuxDPDA. So how does this routing aspect mesh with further restrictions on circuit resources? We already know that log depth and poly size yields LOGDCFL. In Section 5, we examine a restriction, called *self-similarity*, which allows log depth poly size circuits to capture the class L. Finally, we point out that bounded-width poly size multiplex circuits characterize NC^1: this follows from the known characterization of NC^1 by bounded-width poly size *boolean* circuits [Bar89], despite the fact that a multiplex gate in a bounded-width circuit can a priori access all the circuit inputs.

2 Preliminaries

We assume basic familiarity with complexity theory such as can be found in recent texts on the subject. We recall that LOG(CFL) (resp. LOG(DCFL)) is the class of languages logspace-reducible to a context-free (resp. deterministic context-free) language. A PDA is a pushdown automaton and an AuxPDA is a

machine with a work tape, which alone is subjected to space bounds. along with an auxiliary pushdown. LOGCFL(resp. LOGDCFL) is the class of languages recognized by non-deterministic (resp deterministic) logspace auxiliary pushdown automata with polynomial running time.

Definition [FLR96]. A *multiplex* circuit is a circuit in the usual sense, whose only gates are *multiplex select* gates (see Figure 1). The inputs to a multiplex select gate are grouped into a bundle of $k \in O(\log n)$ steering bits and 2^k equal size bundles of $l \in O(\log n)$ data bits; the gate output is a single bundle of l data bits. One special gate in this circuit is the output gate, whose data bit bundles have size 1. Other special gates are the input gates $1, 2, \ldots, n$, also considered to have an output bundle of size 1. Bit bundles cannot be split or merged, that is, if the output of a gate A is input to a gate B, then the complete output bundle of A enters B as a indivisible and complete (steering or data) bundle; the only exception to this rule is that bundles can be extended by any (logarithmic) number of high order constant bits.

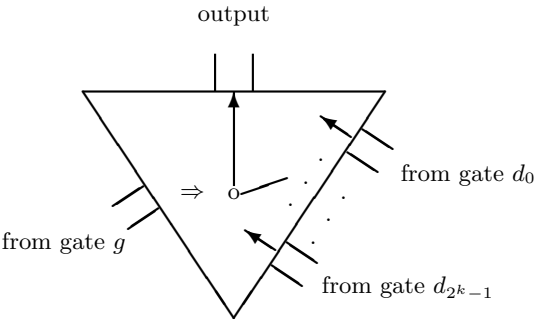

Fig. 1. A multiplex select gate

On input $x_1x_2\cdots x_n \in \{0,1\}^*$, the multiplex circuit input gates respectively take on the values $x_1, x_2, \ldots, x_n$. Then each multiplex select gate having as in Figure 1 a bundle g of k steering bits eventually determines its output value by interpreting its steering bits as the binary encoding of a number j, $0 \le j < 2^k$, and by passing on its jth input bundle as output. The circuit accepts $x_1x_2\cdots x_n$ iff the output gate outputs 1.

For the purposes of this abstract, we use logspace uniformity for multiplex circuit families, although tighter uniformities work as well. Hence we say that a multiplex circuit family is uniform iff its direct connection language is logspace-computable, where the direct connection language consists of the sequence of encodings of each gate in the circuit (a gate as in Figure 1 is described by its number, and by the numbers of the gates $g, d_0, \ldots d_{2^k-1}$ combined, as the case may be, with high order constant bits).

The *size* and the *depth* of a multiplex circuit (or family) are defined in the usual way. A multiplex circuit can be unraveled into a tree like any other circuit (by duplicating nodes having outdegree greater than one, all the way down to, and including, the input gates). Given a multiplex circuit C, a *proof tree on input*

x is a pruning of the unraveled circuit which eliminates all the gates except the root of T and, for each non-input gate g kept, both the gate computing the steering bundle of g and the gate computing the data bundle selected by g. The *Prooftreesize* of C is the maximum, over all inputs x of the appropriate size, of the size of the proof tree on input x.

To define width for a multiplex circuit, we follow [Bar89] and assume that the circuit is leveled, i.e. that each gate g, except $1, 2, \ldots, n$, can be assigned a unique level given by the length of any one of its paths to the circuit output gate[1]. The *width* of a leveled multiplex circuit is the maximal number of wires connecting one level to the next. (Wires coming from an input of the circuit are not counted, and this is a standard practice to allow for sub-linear widths) The *gate-width* of a leveled multiplex circuit is the number of non-input gates in its largest level[2]. Observe that the number of circuit input gates accessed by a (gate-width-bounded or width-bounded gate) is not subject to the width bound; in particular, any single gate could even access all the circuit inputs.

Definition: MDepthSize(d, s) is the class of languages recognized by a uniform family of multiplex circuits having depth d and size s [FLR96]. Analogously, MProoftreesize-Size(p, s), MGatewidth-Size(g, s), MWidth-Size(w, s) are defined using proof tree size p, gate-width g and width w. Finally, SMDepthSize(d, s), where the S stands for self-similar, is defined from leveled circuits having the property that all gates across a level have the same sequence of data inputs, meaning that d_j for every $0 \leq j < 2^k$ is identical for every gate across the level.

Accordingly, Prooftreesize-Size(pol,pol) is defined using SAC-circuits, where a prooftree for an input x in the language is an accepting subcircuit (certificate).

3 Simpler proof for LOG(CFL)

Lemma 1. Prooftreesize-Size(pol,pol) $\subseteq$ LOG(CFL).

Proof: We must reduce, in logspace, an arbitrary $Y \in$ Prooftreesize-Size(pol,pol) to a language recognized by a standard NPDA. The main idea is to code each gate and its connections in the SAC circuit for Y in a clever way. Concatenating the descriptions of all the gates yields the encoding of the circuit. To get around the restriction that the NPDA has a one-way input head, we concatenate a polynomial number of copies of the circuit encoding, the polynomial being at least the size of an accepting proof tree in the circuit. This will be the output of the logspace transducer.

The code for a gate g is a tuple with (1) the label of g, (2) the label of g in reverse (this is motivated later), (3) the type of g, and (4) the labels in reverse of the gates that are inputs to g.

[1] As in the case of boolean circuits, this assumption can be enforced at the expense of inconsequential increases in size, since any wire bundle can be made to contribute to the depth by replacing this bundle with a gate having this same bundle duplicated as steering, multiple data, and output.

[2] Width is usually defined in the context of bounded fan-in circuits, and no distinction is necessary between width as defined here and gate-width in that case.

Initially, the PDA starts by pushing the label corresponding to the output gate. At any intermediate point, the top of the stack contains a gate (label); we wish to assert that this gate evaluates to a 1. On the input tape, we move the head forward until we are at a gate that matches the gate on top of stack. The semantics of the gate type now takes over. If it is an input gate, the PDA checks if its value is 1 and continues with the remaining gates (if any) on the stack; if its value is 0 the PDA rejects and halts. If the gate is an AND gate, the PDA places the two inputs to the AND gate on the stack, but since their labels are reversed on the input tape, they are pushed in the correct order onto the stack for future comparisons. If the gate is an OR gate, one of its inputs gates is non-deterministically pushed onto the stack.

To finish, we need to explain the necessity of component (2), in the code for a gate. In the previous paragraph, we did not explain how the PDA matches the gate on the stack with one on the input tape. Of course, we can do this with non-determinism: we can simply move the input head non-deterministically until it is magically under the right gate label. But we would like to do this deterministically as we use the idea for a later simulation. (And we all know that non-deterministic bits are costly!) So we check if the gate under the input head matches the gate on top of stack by popping identical bits. If the gates are different, we will notice it at some point. Let the input head see $pbq, q^R bp^R, \ldots$ and the PDA have $pc \ldots$ on its stack (where c is the complement of b). The machine will now match p and detect a difference on bit b. Skip this bit on the input tape and push the remaining bits of the label onto the stack; the input tape now looks like $q^R bp^R, \ldots$ and the stack looks like $q^R c \ldots$. Now match the input head and the stack until there is a difference, at which point we simply push the remaining bits of the label onto the stack to recreate the original gate label on the stack.

The language $L_{polproof}$, which is accepted by the PDA is contextfree, where we need not care about words which do not encode a circuit. Thus every language in Prooftreesize-Size(pol,pol) can be reduced to $L_{polproof}$ by a logspace transducer from input x using the uniformity for $C_{|x|}$. □

Corollary 1. $\mathsf{SAC}^1 \subseteq \mathsf{LOG(CFL)}$.

4 New characterization of LOGDCFL

Consider a proof tree in a multiplex circuit. For a gate in a proof tree only two child gates have to be in the proof tree: the gate where the steering bundle comes from, and the gate where the selected bundle comes from. Thus we get the following:

Lemma 2. MDepth-Size(log,pol)⊆MProoftreesize-Size(pol,pol).

In the spirit of simplifying proofs, we can bypass the difficult direction in [DR86,FLR96] showing DAuxPDA-TIME(pol) ⊆ CROWTI-PR(log,pol) by the following naive algorithm simulating a DAuxPDA with only polynomial proof tree

size. The function works on extended surface configurations containing the state, the top of the pushdown, the contents of the work tape, the position of the head on the input tape, the time and the current height of the pushdown. We may assume w.l.o.g. that every configuration either pushes or pops.

function GO(x) :=
  if Configuration x pushes Push(x) $\neq \lambda$
    if going to Configuration y
    then z :=GO(y); **return** GO(POP(z,Push(x)))
  else return x

POP(z,a) is the configuration following z if z is a popping configuration and a is the symbol, which is popped from the pushdown.

The function outputs the last reachable surface configuration reached from x, without popping deeper symbols from the pushdown. W.l.o.g. GO(x_0) is the end configuration. The translation of this function to a multiplex-select circuit yields

Lemma 3. DAuxPDA-TIME(pol) $\subseteq$MProoftreesize-Size(pol,pol)

Proof: A circuit calculating GO(x) using subcircuits for GO(y) and GO(POP(z, a)) is partially depicted on Figure 2. Each extended surface configuration and therefore each gate appears only once in a proof tree, which thus has polynomial size. □

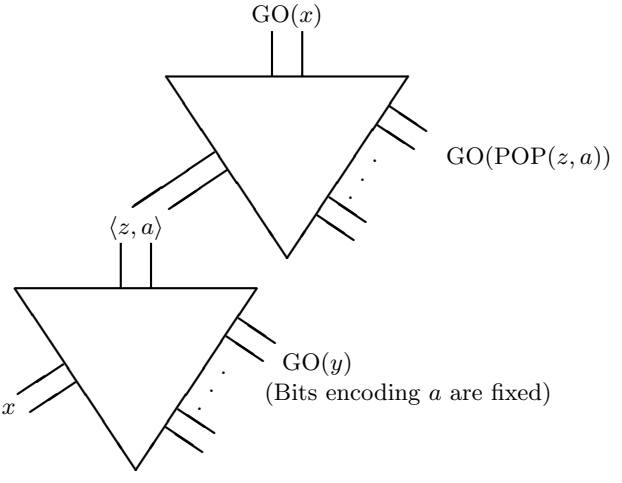

Fig. 2. Circuit used in proving Lemma 3

Lemma 4. MProoftreesize-Size(pol, pol) $\subseteq$LOG(DCFL).

Proof: As in the proof of Lemma 1, we encode for every j-th data input of a gate g

$$w_j^g := (bin_j \# g \$ g^R \# bin_j^R \# f_j^R \# d_j^R \#)$$

where bin_j is the binary encoding of j, f_j is the fixed part of the j-th data input and d_j is the binary encoding of the gate to which the j-th data input bundle is connected (d_j is optional). For every gate g having a steering input from gate s_g, we encode

$$w_g := (g\$g^R\#s_g^R)w_1^g w_2^g ... w_k^g$$

and let w_V be the concatenation of all w_g with $g \in V$. Let furthermore s be the proof tree size of the circuit. The logspace transducer generates $w := w_V^{2s}$ (which has polynomial length).

To evaluate a gate g, a pushdown automaton uses the same method as in Lemma 1 to find the right $(g\$g^R\#s_g^R)$ and pushes $g^R\#s_g^R$, which allows to evaluate the gate s, where the steering input came from and return with the value, which now becomes bin_j over g, which is still on the pushdown. Again the same method allows to find the right w_j^g, where $f_j^R\$d_j^R$ is pushed, which allows to evaluate the gate d_j and regard (because of the distinction between \$ and #) f_j concatenated with the returned value as the new value, which is to return. □

5 Multiplex circuits for L and for NC¹

Theorem 1. $\mathsf{L} = \mathsf{MGatewidth\text{-}Size}(1, pol) = \mathsf{MWidth\text{-}Size}(log, pol)$.

Proof: Since the output-bundle of a gate consists of a logarithmic number of wires, it follows from the definition that $\mathsf{MGatewidth\text{-}Size}(1, pol) \subseteq \mathsf{MWidth\text{-}Size}(log, pol)$.

We get $\mathsf{MWidth\text{-}Size}(log, pol) \subseteq \mathsf{L}$ by evaluating a circuit level by level in polynomial time. Since only the outputs of the previous level have to be stored, logarithmic space is sufficient.

To show $\mathsf{L} \subseteq \mathsf{MGatewidth\text{-}Size}(1, pol)$ we have to construct for a logspace machine a multiplex-select circuit, which consists of a single path of multiplex select gates, where the output bundle of a gate is connected to the steering input of the following gate. The value v of such a bundle encodes a surface configuration containing the state, the contents of the work tape, the position of the head on the input tape and on the lowest order bit the symbol (0 or 1) on the input tape at the current head position. The first gate gets on the steering input the start configuration. The v-th data input bundle of every (except the last) gate contains an encoding of the following configurations on v where the lowest order bit is connected to the corresponding bit on the input and all the other bits are given as constants[3]. The last gate has 1's on the data inputs whose numbers encode accepting configurations and 0's on the others. □

Theorem 2. $\mathsf{L} = \mathsf{SMDepth\text{-}Size(log,pol)}$.

[3] This means that the uniformity condition for the circuit has to be able to verify a single step of a logspace machine; the number of the input bit used is contained as head position in the encoding of the configuration.

Proof: $\subseteq$: Let l be the rounded up logarithm of the running time and k be the number of surface configurations. The first level consists of k gates of the same kind as in the proof of theorem 1 having the number $(c, 1)$ and getting the configuration c as (a constant) steering input. Thus their output will be the following configuration of c. The gates having the number (c, h) on level h with $1 < h \leq l$ get the output of gate $(c, h-1)$ as steering input and the output of gate $(c', h-1)$ as the c'-th data input, which means that the data input bundles on the h-th level are always connected the same way. By induction on h it follows that the output of gate number (c, h) will be the 2^h-th configuration following c. Thus the output of gate number (c_0, l) contains the end configuration [4] and is connected to the last gate, which, as in the proof of theorem 1 has 1's on the data inputs whose numbers encode accepting configurations and 0's on the others.

$\supseteq$: A logspace machine can evaluate a self-similar circuit in the following way: To evaluate a gate, it first recursively evaluates the gate where the steering input comes from; here it does not store the number of the gate but only the fact (one bit) that it is evaluating the steering input for a gate of that level. When it returns the value v for this steering input, it recursively evaluates the gate where the v-th data input comes from; because of self-similarity this is the same for all gates on this level thus the number of the exact gate is not necessary. Again only the fact (one bit) that it is evaluating a data input is stored for that level, telling the machine to return the value to the next level, if the value from the previous level is returned. The machine accepts if the evaluation of the last gate is 1 and rejects if it is 0. Since there is a logarithmic number of levels and the value always has a logarithmic number of bits, logarithmic space is sufficient. □

One difference between bounded-width (bounded-fan-in) boolean circuits and bounded-width multiplex circuits is that any gate in the latter can access all the circuit inputs. However, this extra ability of multiplex gates is useless within a bounded-width circuit:

Theorem 3. MWidth-Size(constant,pol) = NC^1.

Proof: $\subseteq$: Consider a multiplex circuit C of bounded width w. The width bound implies that the steering input bundle to any gate in C can only take a constant number of different values. Even if there were n circuit inputs attached to the data portion of this gate, only a constant number of them can be selected. These selectable inputs can be identified at "compile time", and those not selectable can be set to constants (while preserving the circuit uniformity). The result is an equivalent multiplex circuit using at most a constant number of circuit inputs per level. Each gate in this circuit has constant fan-in, omitting constant bits. Now any constant fan-in gate (thus also a multiplex select gate) can be replaced by a constant size NC subcircuit. Replacing all multiplex gates in this way yields a bounded-width polynomial size boolean circuit in the sense of [Bar89, Section 5], who shows that such circuits characterize NC^1.

[4] W.l.o.g. the machine stays in its end configuration.

$\supseteq$: A binary multiplex select gate[5] (see Figure 1) simulates a NOT-gate if d_0 has the constant value 1 and d_1 has the constant value 0, it simulates an AND-gate if d_0 has the constant value 0, and it simulates an OR-gate if d_1 has the constant value 1. Thus any NC circuit is simulated within the same resource bounds by a multiplex circuit. In particular, the bounded-width polynomial size circuits of [Bar89] are simulated by bounded-width polynomial size multiplex circuits. □

6 Conclusion

We have simplified proofs involving LOGCFL and LOGDCFL, and obtained new characterizations of LOGDCFL, L, and NC^1. We would like to characterize NL in a similar way, by restricting SAC^1 circuits. Now we observe that the SAC^1 circuits obtained in a canonical way by encoding the NL-complete reachability problem have the property that they are robust against a simplified NL evaluation algorithm, which would of course make mistakes when evaluating general SAC^1-circuits. An open problem is whether this robustness property has a nice[6] characterization.

Acknowledgment. The first author acknowledges useful discussions with Markus Holzer.

References

AJMV98. E. Allender, J. Jiao, M. Mahajan and V. Vinay, Non-commutative arithmetic circuits: depth reduction and size lower bounds, *Theoret. Comput. Sci.* 209, pp. 47–86, 1998.

Bar89. D.A. Barrington, Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1, *Journal of Computer and System Sciences*, 38(1):150–164, 1989.

BCD+88. A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo, and M. Tompa. Two applications of complementation via inductive counting. In *Proc. of the 3rd IEEE Symp. on Structure in Complexity*, 1988.

BLM93. F. Bédard, F. Lemieux, and P. McKenzie. Extensions to barrington's M-program model. *Theoret. Comput. Sci. A*, 107(1):33–61, 1993.

Coo79. S. Cook, Deterministic CFL's are accepted simultaneously in polynomial time and log squared space, Proc. of the 11th ACM Symp. on Theory of Computing, 338–345, 1979.

DR86. P. W. Dymond and W. L. Ruzzo. Parallel RAMs with owned global memory and deterministic language recognition. In *Proc. of the 13th Int. Conf. on Automata, Languages, and Programming*, LNCS226, pages 95–104. Springer-Verlag, 1986.

FLR96. H. Fernau, K.-J. Lange, and K. Reinhardt. Advocating ownership. In V. Chandru, V. Vinay, editor, *Proc. of the 16th Conf. on Foundations of Software Technology and Theoret. Comp. Sci.*, vol. 1180 of *LNCS*, pages 286–297, Springer, 1996.

[5] having only 1 steering bit and 2 data bits

[6] by a condition which is in some way 'locally checkable'.

Gal74. Z. Galil. Two–way deterministic pushdown automata and some open problems in the theory of computation. In *Proc. 15th IEEE Symp. on Switching and Automata Theory*, pages 170–177, 1974.

Har72. J. Hartmanis. On non-determinancy in simple computing devices. *Acta Informatica*, 1:336–344, 1972.

Iba73. O. H. Ibarra. On two way multi-head automata. *Journal of Computer and System Sciences*, 7(1):28–36, 1973.

LMSV99. C. Lautemann, P. McKenzie, H. Vollmer and T. Schwentick, The descriptive complexity approach to LOGCFL, *Proc. of the 16th Symp. on the Theoret. Aspects of Comp. Sci.*, 1999.

MRS93. B. Monien, W. Rytter, and H. Schäpers. Fast recognition of deterministic cfl's with a smaller number of processors. *Theoret. Comput. Sci.*, 116:421–429, 1993. Corrigendum, 123:427, 1993.

Rei90. K. Reinhardt. Hierarchies over the context-free languages. In J. Dassow and J. Kelemen, editors, *Proceedings of the 6th International Meeting of Young Computer Scientists*, number 464 in LNCS, pages 214–224. Springer-Verlag, 1990.

Rei97. K. Reinhardt. Strict sequential P–completeness. In R. Reischuk, editor, Proceedings of the 14th STACS, number 1200 in LNCS, pages 329-338, Lübeck, February 1997.

Ruz80. W. Ruzzo, Tree-size bounded alternation, *Journal of Computer and System Sciences*, 21:218–235, 1980.

Sud78. I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25:405–414, 1978.

Ven91. H. Venkateswaran. Properties that characterize LOGCFL. *Journal of Computer and System Sciences*, 43:380–404, 1991.

Vin96. V. Vinay. Hierarchies of circuit classes that are closed under complement. In *Proc. of the 11th IEEE Conf. on Computational Complexity (CCC-96)*, pp. 108–117, 1996.