THEORY OF COMPUTING, Volume 10 (8), 2014, pp. 199–215
www.theoryofcomputing.org

Symmetry Coincides with Nondeterminism for Time-Bounded Auxiliary Pushdown Automata

Eric Allender* Klaus-Jörn Lange

Received December 20, 2011; Revised September 12, 2014; Published September 17, 2014

Abstract: We show that every language accepted by a nondeterministic auxiliary pushdown automaton in polynomial time (that is, every language in $\mathsf{SAC}^1 = \mathsf{Log}(\mathsf{CFL})$) can be accepted by a symmetric auxiliary pushdown automaton in polynomial time.

ACM Classification: F.1.3, F.1.2, F.1.1

AMS Classification: 68Q15, 68Q10, 68Q45, 68Q05

Key words and phrases: complexity theory, complexity classes, circuit complexity, nondeterminism, symmetry, reversibility

1 Introduction

Most of the fundamental questions in complexity theory hinge on the relationship between deterministic and nondeterministic computation. The intermediate notion of *symmetric* computation was introduced by Lewis and Papadimitriou [23]. (Informally, a nondeterministic machine is "symmetric" if the inverse of every legal move is also a legal move; more careful definitions appear in Section 2.) Lewis and Papadimitriou introduced symmetry primarily as a tool to characterize the complexity of the graph accessibility problem for undirected graphs; they showed that this problem is complete for Symmetric Logspace[1] (SL). The question of the relationship between SL and deterministic logspace (L) was finally answered by Reingold [26], who showed that $\mathsf{SL} = \mathsf{L}$.

A preliminary version of this work appeared in Proc. IEEE Conf. on Computational Complexity (CCC'10).

*Supported by National Science Foundation grants CCF-0832787 and CCF-1064785

[1] Observe that this holds true for the pure reachability problem only. The shortest path problem for undirected graphs is complete for nondeterministic logspace (NL)! (See, e. g., [31].)

 DOI: 10.4086/toc.2014.v010a008

In contrast to the situation with space-bounded computation, where symmetric computation coincides with determinism, in the case of time bounded computation symmetry is as powerful as unrestricted nondeterminism ([23]). Briefly, this is because if there is no space restriction a machine can keep track of the entire sequence of nondeterministic choices, which makes the computation graph tree-like. Hence, walking "backwards" along edges in the computation graph does not introduce new (erroneous) paths from the start configuration to an accepting state.

In this paper, we consider the role of symmetry in another setting that highlights the potential difference in power between deterministic and nondeterministic computation: $\mathsf{Log}(\mathsf{CFL})$ (a nondeterministic class) and $\mathsf{Log}(\mathsf{DCFL})$ (the corresponding deterministic class). Our main result is that symmetry and nondeterminism coincide in this setting. Let us briefly review some of the most important results that motivate interest in these classes.

$\mathsf{Log}(\mathsf{CFL})$ was defined by Sudborough [30] to be the class of problems logspace-reducible to context-free languages. Venkateswaran [33] gave a circuit-based characterization of $\mathsf{Log}(\mathsf{CFL})$; he showed that $\mathsf{Log}(\mathsf{CFL})$ coincides with SAC^1. Here, SAC^1 denotes the class of problems computable by polynomial-sized "semiunbounded" circuits of logarithmic depth, where a circuit is said to be "semiunbounded" if the AND gates have bounded fan-in and the OR gates have no restriction on the fan-in. Borodin et al. showed that $\mathsf{Log}(\mathsf{CFL})$ is closed under complement [6]. One of the contributions of Sudborough's original paper on $\mathsf{Log}(\mathsf{CFL})$ was to give an automata-theoretic characterization of $\mathsf{Log}(\mathsf{CFL})$, as the class of languages recognized by logspace-bounded nondeterministic *auxiliary pushdown automata* that run in polynomial time.

An *auxiliary pushdown automaton* is a (deterministic or nondeterministic) logspace-bounded Turing machine, that also has a pushdown store that is not subject to the space bound. (In this paper, we only consider auxiliary pushdown automata that are logspace-bounded; thus our notation will not mention the space bound explicitly.) We mention that some authors use the term "stack machines" to refer to auxiliary pushdown automata (e. g., [12]). Deterministic and nondeterministic auxiliary pushdown automata were introduced by Cook [9], who showed that these automata recognize precisely the languages in P, when no restriction is placed on the running time (equivalently, when the running time is bounded by $2^{n^{O(1)}}$). We use the following notation to express this equality:

$$\mathsf{P} = \mathsf{DAuxPDA\text{-}TIME}\left(2^{n^{O(1)}}\right) = \mathsf{NAuxPDA\text{-}TIME}\left(2^{n^{O(1)}}\right).$$

Summarizing, we have:

Proposition 1.1 (Sudborough, Venkateswaran)**.** $\mathsf{NAuxPDA\text{-}TIME}\left(n^{O(1)}\right) = \mathsf{Log}(\mathsf{CFL}) = \mathsf{SAC}^1$.

The class $\mathsf{Log}(\mathsf{DCFL})$ (the class of problems reducible to *deterministic* context-free languages) was also defined by Sudborough [30], who showed $\mathsf{DAuxPDA\text{-}TIME}\left(n^{O(1)}\right) = \mathsf{Log}(\mathsf{DCFL})$. Subsequently, $\mathsf{Log}(\mathsf{DCFL})$ was studied by Dymond and Ruzzo, who showed that $\mathsf{Log}(\mathsf{DCFL})$ consists precisely of the problems solvable in logarithmic time on a CROW-PRAM [14]. Cook showed that deterministic context-free languages can be recognized in polynomial time by machines using $O(\log^2 n)$ space, and thus lie in the class SC^2 [10]. Summarizing, we have:

Proposition 1.2 (Sudborough, Dymond–Ruzzo, Cook)**.**

$$\mathsf{DAuxPDA\text{-}TIME}\left(n^{O(1)}\right) = \mathsf{CROW\text{-}TIME}(\log n) = \mathsf{Log}(\mathsf{DCFL}) \subseteq \mathsf{SC}^2.$$

Symmetry is just one of several intermediate notions between deterministic and nondeterministic computation that have received attention. Two other such notions are *unambiguity* and *randomness*. Unambiguous AuxPDAs, by definition, never have more than one accepting computation path on any input. It is known that, if there is any problem in $\mathsf{Dspace}(n)$ that requires circuits of exponential size, then every problem in $\mathsf{Log}(\mathsf{CFL})$ is accepted by an unambiguous AuxPDA running in polynomial time [4] (and, unconditionally, every problem in $\mathsf{Log}(\mathsf{CFL})$ is reducible via nonuniform projections to a language accepted by an unambiguous AuxPDA running in polynomial time [27]). In contrast, if we require that an AuxPDA have *many* accepting paths if it has any at all, then we arrive at the notion of probabilistic AuxPDAs with one-sided error. Venkateswaran studied such machines (even in the more powerful two-sided error model) in [34] and is working on a proof of the claim that all languages accepted by such machines lie in SC^2 [32]. Thus, if this claim holds, it would be a significant advance if, say, such machines could be shown to recognize all problems in NL (since this would imply $\mathsf{NL} \subseteq \mathsf{SC}^2$).

Along similar lines, there has been some speculation that perhaps Reingold's deterministic simulation of space-bounded symmetric computation could be extended to the model of auxiliary pushdown automata [17]. As a consequence of our results, any such extension would constitute a significant advance in our understanding of the complexity of not only NL, but of $\mathsf{Log}(\mathsf{CFL})$ as well.

We also need to make use of some of the properties of *reversible* computation. Reversibility is a restriction of deterministic computation that can also be viewed as a restriction of symmetric computation, in the sense that running "backward" from any configuration will lead only to configurations that are also reachable by running forward. More precisely: a Turing machine M is *reversible* if its configuration graph has indegree and outdegree at most one. The following theorem from [21] will be useful for us:

Theorem 1.3 (Lange, McKenzie, Tapp)**.** *Any injective function computable in space equal to the input size is computable in the same space bound by a reversible machine.*

(A version of Theorem 1.3 actually holds even for non-injective functions [21], but for our purposes it is sufficient to consider the simpler case where only injective functions are considered.)

Thus, when we build an AuxPDA that carries out a particular segment of its computation *deterministically* without moving its input head or using the pushdown store, in such a way that the configuration at the end of the segment uniquely determines what configuration the AuxPDA was in when the segment began (so that this segment corresponds to an injective function on inputs of length m computable in space m, where $m = \log n$ is the size of the worktape), it follows that the AuxPDA can be programmed so that this segment is actually reversible. Thus if we add "backward" moves to the AuxPDA to make it symmetric, the only additional computations that arise from the original deterministic segment are computations that correspond to running the segment backward.

Of course, there will be occasions when our AuxPDA will have to move its input head (or use its stack), and Theorem 1.3 does not directly allow us to conclude that certain simple deterministic computations can be carried out reversibly. Thus we appeal to the following simple proposition:

Proposition 1.4. *The following computations can be performed deterministically and reversibly:*

- *Start with the input head on the left endmarker, with the worktape blank, and end with the input head on the left endmarker, with the length of the input recorded in binary on the worktape.*

- *Start with the input head on the left endmarker and a number j on the worktape, and end with the input head on the left endmarker and the pair (j,a) on the worktape, where a is the jth input symbol.*
- *Start with a string y on a worktape (which we will call the "buffer"), and end with y pushed onto the stack with the buffer empty.*
- *Start with a blank section of worktape of length r and a string yz on the stack where $|y| = r$, and end with y in that section of the worktape and popped off the stack (so that the stack holds z).*

Proof. Note that, by Theorem 1.3, there is a deterministic and reversible computation that starts with a number j written on the worktape, and increments it. Similarly, there is a deterministic and reversible computation that starts with a number j and a bit b and decrements j (and flips the bit b if $j = 0$).

Thus, in order to prove the first item in Proposition 1.4, it suffices to observe that the following routine can be implemented reversibly:

- Write "0" on the blank worktape, and then repeat the following steps until the input head scans the right endmarker:
  1. Move the input head to the right, and
  2. Increment the counter on the worktape.
- When the loop is exited, move the input head back to the left end of the tape.

The second item in the proposition is proved with very similar techniques:

- Insert a "0" before j, so that the worktape holds "$0j$", and then repeat the following steps until the first bit of the worktape is 1:
  1. Move the input head to the right, and
  2. Decrement j (and flip the first bit if $j = 0$). [Thus, as long as the first bit is 0, the machine is moving its head toward position j of the input tape.]
- Remember the symbol a currently scanned on the input tape, and then repeat the following steps until the input head is at the left end of the tape:
  1. Move the input head to the left, and
  2. Increment j (and flip the first bit if j was equal to 0). [Thus, as long as the first bit is 1, we remember the symbol a at position j on the input tape, and the input head is moving toward the start of the tape.]
- At this point, the input head is back on the left endmarker and the worktape holds $0j$. Replace $0j$ with (j,a).

For the third item, consider a machine with states $q_{\text{push}}, q_{\text{move}}, q_{\text{return}}$, and q_a for each symbol a. A sequence of moves that starts the process of pushing the buffer onto the stack starts in state q_{push}. In state q_{push}, if the machine scans a worktape symbol a other than the end-of-buffer marker, it enters state q_a and replaces the a with a blank symbol. (If it scans the end-of-buffer marker, it enters state q_{return}.)

In state q_a, it replaces a blank on top of the stack with an a, and moves to q_{move}. (The only "backward" move from state q_a is to change a blank on the worktape to an a and move to state q_{push}.)

In state q_{move}, it moves the worktape head to the right, and moves to state q_{push}. (The only "backward" move from state q_{move} is to pop some symbol a off the stack, and move to state q_a. There are no other moves that enter q_{push}; thus the only "backward" move from q_{push} is to move the worktape head to the left, and move to state q_{move}.)

In state q_{return}, the machine moves the worktape head to the left end of the buffer. (The only "backward" moves take the machine back to the right end of the buffer, where it enters state q_{push}.)

It is easy to verify that each configuration of this machine has one incoming edge and one outgoing edge, and that it carries out the transformation of the third item above.

The fourth and final item in this proposition is essentially the inverse of the transformation from item three, and it is proved with very similar techniques. □

In Section 3, we establish our main result: that nondeterminism and symmetry coincide for polynomial-time bounded AuxPDAs. Since, by Theorem 1.3, reversibility coincides with determinism for space-bounded computation, and since reversibility plays an important role in the proof of Theorem 3.1, it is natural to wonder about the computational power of *reversible* AuxPDAs. We are not able to settle this question, but in Section 4 we summarize what we are able to establish about the power of reversible AuxPDAs, and present some open questions.

2 Preliminaries and overview

A *configuration* C of an AuxPDA encodes complete information about the state of the machine at a given point in a computation (including positions of all heads, contents of all tapes, buffers, and pushdowns), and as usual we let $C \vdash D$ denote the relation on configurations where the machine can start in configuration C and move in one step to configuration D. A subset of the states is labeled as "accepting", and we say that the machine *accepts* an input if there is a computation path starting from the initial configuration and reaching an accepting state.

Lewis and Papadimitriou ([23]) introduced the concept of symmetric computation. A nondeterministic Turing machine is symmetric if, for any configurations C and D, we have that $C \vdash D$ if and only if $D \vdash C$.

With symmetric machines, it is impossible to require that computations halt, and thus we use the convention that a symmetric AuxPDA runs in time $t(n)$ if, for every input x of length n, if there is any accepting computation path at all on input x, then there is an accepting computation path of length at most $t(n)$.

Definition 2.1. Let $\mathsf{SymAuxPDA\text{-}TIME}\big(n^{O(1)}\big)$ denote the class of languages accepted by symmetric logspace-bounded AuxPDAs that run for polynomial time.

We remark that symmetric AuxPDAs were also defined independently by Kintali, in connection with a study of various graph reachability problems [17].

In order to describe the symmetric algorithms that we present, we will use the following approach. First, we will present a nondeterministic (non-symmetric) AuxPDA that clearly accepts a given language. The AuxPDA will be designed using some conventions that allow us to reason clearly about its behavior. Then, we will "symmetrize" the AuxPDA, by introducing new moves, so that if $C \vdash D$ we ensure that also $D \vdash C$, and we will argue that this will not change the language that is accepted.

Here are some conventions that we will follow, in our AuxPDA algorithms. The logspace-bounded worktape will have two sections: a storage area, and a buffer. The buffer is used to push and pop items to and from the pushdown store; data will be pushed and popped in units of length $m = O(\log n)$, pops will only be initiated when the buffer is empty, and pushes will have the effect of emptying the buffer. Since pushes and pops are done deterministically (and reversibly, by Theorem 1.3 and Proposition 1.4), it is no loss of generality to treat these multi-step operations as *basic* operations (since, once begun, either the entire push (pop) is completed, or else the operation is run back to the start, as if it had never been begun). In order to simplify the definitions, we assume that the computation begins with $\log n$ space marked off on the worktape and the buffer (with endmarkers) and we assume that the read-only input tape also has endmarkers, and the bottom of the stack is marked.

3 Main result

In this section we show that every language in SAC^1 is accepted by some symmetric auxiliary pushdown automaton in polynomial time. Thus, by Proposition 1.1, this establishes our main theorem:

Theorem 3.1. $\mathsf{NAuxPDA\text{-}TIME}\left(n^{O(1)}\right) = \mathsf{SymAuxPDA\text{-}TIME}\left(n^{O(1)}\right)$.

Proof. Let L be a language in $\mathsf{Log}(\mathsf{CFL}) = \mathsf{SAC}^1$. Thus L is accepted by a logspace-uniform family of circuits $\{C_n\}$, where without loss of generality we may assume the following:

- The gates of the circuit C_n are partitioned into levels $\ell_0, \ell_1, \ldots, \ell_{d(n)}$, where the depth of the circuit is $d(n) = O(\log n)$.

- The input level ℓ_0 of the circuit C_n consists of input gates that are connected either to input symbols x_i or to negated input symbols $\overline{x_i}$, $1 \le i \le n$. The wires that lead out of the input gates feed into AND gates at level 1.

- If $i > 0$ is even, then all of the gates in level ℓ_i are OR gates. If i is odd, then all of the gates in level ℓ_i are AND gates.

- Each AND gate h has fan-in exactly two.

- Wires from any level ℓ_i are directed toward gates in level ℓ_{i+1}, and a logspace computation can tell, given g and h, if there is an edge from g to h.

- For each n, the output gate g_{out} of C_n is an OR gate at level $d(n)$, and the function that maps n to $(g_{\text{out}}, d(n))$ is computable in logspace.

We now describe a nondeterministic (non-symmetric) AuxPDA M accepting L. We will then create a symmetric AuxPDA M' from M, and argue that it also accepts L in polynomial time. We will use the "symbol" $[g,i]$ to denote the contents of the worktape when our AuxPDA M is attempting to determine if the gate g in level ℓ_i evaluates to 1. We use the "symbol" $\overline{[g,i]}$ to denote the contents of the worktape when our AuxPDA has successfully verified that g evaluates to 1. (The symbols $[g,i]$ and $\overline{[g,i]}$ will only be used when g is an OR gate, and i is even.) In addition, we will use a "protocol symbol" $\langle g,h\rangle$ to denote the fact that our AuxPDA is trying to verify that the OR gate g evaluates to 1, by verifying that the AND gate h that feeds into g evaluates to 1. Observe that each of these "symbols" require $O(\log n)$ bits to write down.

We first create a nondeterministic (non-symmetric) AuxPDA M operating as follows: On input x, with the stack empty, our AuxPDA M uses a deterministic and reversible computation to record the input length n on the worktape, and then (by appealing to logspace-uniformity) places the symbol $[g_{\text{out}},d(n)]$ on the worktape. This computation is deterministic and injective, and can be done via a reversible computation by Theorem 1.3 and Proposition 1.4.

For any configuration where the worktape holds $[g,i]$, our AuxPDA M checks first to see if $i=0$. If $i=0$, then g is an input gate. Hence by logspace-uniformity, M can use deterministic, reversible computation to compute an index j such that gate g is an input gate in level ℓ_0 that depends on bit j of the input. Then, by Proposition 1.4, M can record the jth bit of the input on the worktape (via a deterministic and reversible computation). M then checks (via a deterministic, reversible computation) if g evaluates to 1 and if so, it replaces $[g,i]$ with $\overline{[g,i]}$. (If the gate g evaluates to 0, then M halts and rejects.)

If $i>0$, then via nondeterministic and symmetric moves, M guesses a string h, so that the worktape holds $([g,i],h)$. (Using the "backward" moves of these nondeterministic steps corresponds to merely erasing some of the guess "h" and thus involves revisiting an earlier configuration. This cannot happen in any accepting computation path of minimal length.) After the worktape holds $([g,i],h)$, M uses deterministic, reversible computation to verify that h is an AND gate in level ℓ_{i-1} that feeds in to g, and then computes the names g_1 and g_2 ($g_1<g_2$) of the two OR gates that feed in to h, and then writes $[g_1,i-2]$ on the worktape, and writes the string $[g_1,i-2]\triangleleft[g_2,i-2]\langle g,h\rangle$ onto the buffer, before pushing this string onto the pushdown. (Note that the initial part of this deterministic computation is *independent* of the input, and thus can be done reversibly via direct appeal to Theorem 1.3. Pushing information onto the pushdown can be done reversibly by Proposition 1.4)

For any configuration where the worktape holds $\overline{[g,i]}$, M first checks if $g=g_{\text{out}}$ and $i=d(n)$, in which case it will halt and accept.

Otherwise, M pops a string (of length equal to the buffer) off the stack and stores it in the buffer (via a deterministic, reversible computation). There are two valid cases that allow the computation to proceed:

- If the buffer is equal to $[g,i]\triangleleft[g',i]\langle g'',h\rangle$, then M will put $[g',i]$ on the worktape, and push the string $\overline{[g,i]}[g',i]\triangleleft\langle g'',h\rangle$, onto the stack (via a deterministic, reversible computation).

- If the buffer is equal to $\overline{[g',i]}\,\overline{[g,i]}\triangleleft\langle g'',h\rangle$, where $g'<g$ are the two OR gates that feed into h, then M will write the tuple $(\overline{[g'',i+2]},h)$ on the worktape and erase the buffer, via a deterministic and reversible computation (note that no information is lost here, since h determines the pair (g,g')), and then it will *erase* h, leaving only $\overline{[g'',i+2]}$ on the worktape. Clearly, this last segment is *not* reversible, since it destroys all information about h (and with it, all information about g' and g).

1	worktape	$[g,0]$	$\rightarrow$	$\overline{[g,0]}$		
	stack					
2	worktape	$[g,i]$	$\Leftrightarrow$	$[g,i],h$	$\rightarrow$	$[g_1,i-2]$
	stack					$[g_1,i-2] \triangleleft [g_2,i-2]\langle g,h\rangle$
3	worktape	$\overline{[g,i]}$	$\rightarrow$	$[g',i]$		
	stack	$[g,i] \triangleleft [g',i]\langle g'',h\rangle$		$\overline{[g,i]}[g',i] \triangleleft \langle g'',h\rangle$		
4	worktape	$\overline{[g,i]}$	$\rightarrow$	$\overline{[g'',i+2]},h$	$\mapsto$	$\overline{[g'',i+2]}$
	stack	$\overline{[g',i]}[g,i] \triangleleft \langle g'',h\rangle$				

Table 1: Forward moves of M. There are four types of moves (not counting the moves that do the initial set-up, and the moves that determine if conditions have been satisfied to move to an accepting state). Only moves of type one consult the input tape.
Transitions marked $\Leftrightarrow$ are symmetric.
Transitions marked $\rightarrow$ are deterministic and reversible.
Transitions marked $\mapsto$ are deterministic and non-reversible.

Note that, if we add backward transitions to make M symmetric, the new transitions that are added for this segment correspond to guessing arbitrary values of h. Thus these moves are dual, in some sense, to the nondeterministic and symmetric moves of M that guess h.)

The moves of M are summarized in Table 1.

When we create a symmetric AuxPDA M' from M by adding the required "backward" moves (as described in Table 2), we need to argue that the new machine M' does not accept any strings that were not already accepted by M. We express this as the following claim:

Claim 3.2. *For every even number i, gate g is a gate in level ℓ_i that evaluates to 1 if and only if there is a computation path of M' that starts with $[g,i]$ on the worktape, with an empty stack and buffer, and reaches $\overline{[g,i]}$, with an empty stack and buffer.*

Proof. The forward direction is obvious, and it is also obvious that in this case there is a computation path of polynomial length. To prove the backward direction, if there is a computation path of M' from $[g,i]$ to $\overline{[g,i]}$, we work with a *shortest* such path. The proof proceeds by induction on i.

If $i=0$, let us assume that there is a computation path of M' that starts with $[g,0]$ on the worktape, with an empty stack and buffer, and reaches $\overline{[g,0]}$. If this path consists of only forward moves of M, then this clearly implies that g evaluates to 1 (by construction). We need to show that (without loss of generality) no backward moves of M appear on this path. The only *forward* moves of M that lead *into* any configuration of M with $[g,0]$ on the worktape, are moves that perform a push. Such moves can not be executed in a backward direction by M' when the stack is empty. The only other backward moves that can occur along the computation from $[g,0]$ to $\overline{[g,0]}$ correspond to undoing (and re-doing) part of the deterministic, reversible computation between these two configurations, and hence will not occur along any accepting path of minimal length. (Throughout the rest of the proof, we assume that any

1	worktape	$[g,0]$	$\leftrightarrow$	$\overline{[g,0]}$		
	stack					
2	worktape	$[g,i]$	$\Leftrightarrow$	$[g,i],h$	$\leftrightarrow$	$[g_1,i-2]$
	stack					$[g_1,i-2] \triangleleft [g_2,i-2]\langle g,h\rangle$
3	worktape	$\overline{[g,i]}$	$\leftrightarrow$	$[g',i]$		
	stack	$[g,i] \triangleleft [g',i]\langle g'',h\rangle$		$\overline{[g,i]}[g',i] \triangleleft \langle g'',h\rangle$		
4	worktape	$\overline{[g,i]}$	$\leftrightarrow$	$\overline{[g'',i+2]},h$	$\Leftrightarrow$	$\overline{[g'',i+2]}$
	stack	$\overline{[g',i]}[g,i] \triangleleft \langle g'',h\rangle$				

Table 2: Moves of M'. Transitions marked $\leftrightarrow$ originated from deterministic and reversible steps of M, and hence constitute a subgraph of the configuration graph having degree two.
Transitions marked $\Leftrightarrow$ are symmetric, and configurations in these segments typically have degree larger than two.

deterministic, reversible computation segment that is begun is run to completion, since the only other way the computation can exit the segment is by revisiting the configuration where it began the segment.) This completes the proof of the basis step.

If $i > 0$, then as in the basis step, the computation of M' starting from $[g,i]$ cannot begin using backward moves of M, because it would involve undoing a push, and the stack is currently empty. Thus the only way to start is using symmetric moves of M to guess some value h, and then to use deterministic (reversible) moves of M that cause us to push the string $[g_1,i-2] \triangleleft [g_2,i-2]\langle g,h\rangle$ onto the stack, leaving $[g_1,i-2]$ on the worktape. Let us say that this configuration of M' is reached at time t_1.

Similarly, the segment of the computation of M' that ends with $\overline{[g,i]}$ on the worktape, cannot end using backward moves of M, since this would involve undoing a push while the pushdown is empty, and thus this segment must consist of forward deterministic (reversible) moves of M, starting at some time t_m with some symbol $\overline{[g',i-2]}$ on the worktape, and a string of the form $\overline{[g'',i-2]}[g',i-2] \triangleleft \langle g,h'\rangle$ on top of the stack, for some h', g'', g', and proceeding to a configuration where the stack is empty and the worktape contains the tuple $(\overline{[g,i]},h')$, and ending with some *nonreversible* moves that erase h'. (There may actually be some alternation between forward and backward moves in this segment where some of h' is erased and re-guessed, but in a shortest accepting computation there will be only forward moves in this segment.)

Let us now analyze the portion of the computation of M' that takes place between times t_1 and t_m. We assume without loss of generality that this computation is of minimal length, and thus does not revisit any configuration of M' that occurs at any other time during its computation.

Since some of the $O(\log n)$ symbols on top of the stack at times t_1 and t_m differ, these symbols must have been popped off at some intermediate stage. Since, by construction, M' always completely fills the buffer when it performs a pop, we conclude that there is a first time after t_1 (call this time t_3), when M' pops the string $[g_1,i-2] \triangleleft [g_2,i-2]\langle g,h\rangle$ from the stack. Since all pushes and pops involve moving data between the stack and the buffer via deterministic (reversible) steps, we can see that the pop that takes place at time t_3 must correspond to forward moves of M (since backward moves of M would correspond to forward moves that push $[g_1,i-2]\lhd[g_2,i-2]\langle g,h\rangle$ onto the stack, which only happens if the worktape holds $[g_1,i-2]$ – which in turn means that the computation is retracing its steps back to the start of this segment, contrary to our assumption). Since the pop of $[g_1,i-2]\lhd[g_2,i-2]\langle g,h\rangle$ corresponds to a forward move of M, we see that this takes place in a deterministic (reversible) segment that can only take place if the worktape of M' holds $\overline{[g_1,i-2]}$. Let us denote by t_2 the time when this pop begins. Since time t_2 is the *first* time that these symbols have been popped off the stack, we can conclude that the computation of M' from time t_1 to t_2 begins with $[g_1,i-2]$ on the worktape, ends with $\overline{[g_1,i-2]}$ on the worktape, and can be accomplished with an empty stack. Thus, by induction, we conclude that gate g_1 evaluates to 1.

Recall that, between times t_2 and t_3, M' is executing forward moves of M corresponding to a pop of $[g_1,i-2]\lhd[g_2,i-2]\langle g,h\rangle$ from the stack, with $\overline{[g_1,i-2]}$ on the worktape. This only happens in the middle of a deterministic (reversible) segment (corresponding to moves of type 3 in Table 1). There can be no switch to backward moves of M during the middle of this segment without revisiting earlier configurations, contrary to assumption. Thus this segment executes to completion in a forward direction, resulting in a configuration at some time t_4 with $\overline{[g_1,i-2]}[g_2,i-2]\lhd\langle g,h\rangle$ on the stack, and $[g_2,i-2]$ on the worktape.

There are now two cases:

If there is no intermediate stage between t_4 and t_m where the stack is popped, then we have that $h=h'$, and hence $g_1=g''$ and $g_2=g'$ and there is a computation of M' that begins with $[g_2,i-2]$ on the worktape, ends with $\overline{[g_2,i-2]}$ on the worktape, and can be accomplished with empty stack. In this case, by induction, we have that gate g_2 evaluates to 1. Since h is the AND of g_1 and g_2, we have that h evaluates to 1. Also, since the protocol symbol $\pi=\langle g,h\rangle$ is only written onto the stack if the AND gate h feeds into the OR gate g, we conclude that g evaluates to 1, as desired.

Otherwise, there is some first time t_5, $t_4<t_5<t_m$, where the string $\overline{[g_1,i-2]}[g_2,i-2]\lhd\langle g,h\rangle$ is popped from the stack. If these symbols are popped via forward moves of M, it must be the case that the worktape contains $\overline{[g_2,i-2]}$, and once again we can conclude that g evaluates to 1, as desired. But in fact, this is the only possibility, since if this pop is accomplished via backward moves, it would correspond to moves of M that, if executed in a *forward* direction, would push $\overline{[g_1,i-2]}[g_2,i-2]\lhd\langle g,h\rangle$ on the stack, and this only happens from the configuration that M' is in at time t_4. That is, if this pop is accomplished via backward moves, it means that M' is revisiting the configuration it was in at time t_4, contrary to our assumption.

This completes the proof of the inductive step of the claim, and also completes the proof of the theorem. □□

We remark that the stack height on the symmetric AuxPDA M' is $O(\log^2 n)$ on any accepting computation (since the value i is bounded by $O(\log n)$). Thus we conclude:

Corollary 3.3. *Any language that is accepted by an AuxPDA in polynomial time is accepted by a symmetric AuxPDA running in polynomial time whose pushdown never contains more than* $\log^2 n$ *symbols.*

We remark that this implies that languages in NL are accepted by symmetric machines using space $\log^2 n$ and running in polynomial time.

This generalizes to other space and time bounds as well. The following definitions and Corollary 3.5 make this precise.

Definition 3.4. Let S and T be functions defined on the natural numbers.

- $\mathsf{NSpaceTime}(S(n), T(n))$ is the class of languages accepted by nondeterministic machines operating simultaneously in space $S(n)$ and time $T(n)$.
- $\mathsf{SymSpaceTime}(S(n), T(n))$ is the class of languages accepted by symmetric machines operating simultaneously in space $S(n)$ and time $T(n)$.
- $\mathsf{NSC} = \bigcup_k \mathsf{NSpaceTime}(\log^k n, n^k)$.
- $\mathsf{SymSC} = \bigcup_k \mathsf{SymSpaceTime}(\log^k n, n^k)$.

The classes NSC and SymSC are the nondeterministic and symmetric analogs of Steve's Class SC, respectively.

Corollary 3.5. *Let $S(n) \geq \log n$ and $T(n) \geq n$ be constructible functions in the sense that they are computable simultaneously in space $O(S(n) \log T(n))$ and time polynomial in $T(n)$, given input 1^n. Then*

$$\mathsf{NSpaceTime}(S(n), T(n)) \subseteq \mathsf{SymSpaceTime}(S(n) \log T(n), T(n)^{O(1)}).$$

Proof. A straightforward implementation of the construction from the proof of Savitch's theorem [29] shows that any language accepted by a nondeterministic Turing machine in time $T(n)$ and space $S(n)$ is also accepted by uniform semi-unbounded circuits of size $2^{O(S(n))}$ and depth $O(\log T(n))$. The argument from the proof of Theorem 3.1 shows that such circuits can be simulated by symmetric AuxPDAs that have a worktape bound of $S(n)$ and never have more than $S(n) \log T(n)$ symbols on the pushdown. The time required is $2^{O(\log T(n))}$ (to explore a witness of acceptance, corresponding to a subtree of the circuit, of depth $O(\log n)$). $\square$

The case when T is polynomial and $S(n) = \log^k n$ is of interest; in this case, nondeterministic polynomial time and $\log^k n$ space can be simulated by a symmetric polynomial-time machine in space $\log^{k+1} n$. It is natural to wonder whether $\log^k n$ space would be sufficient for this simulation, instead of $\log^{k+1} n$ space. At least for the case $k = 0$ (i. e., for finite automata), it is known that this is not possible. For a discussion of this, see [19]. For $k = 1$ (i. e., for logarithmic space), such a simulation would imply $\mathsf{NL} = \mathsf{L}$ by Reingold's result ([26]).

Note that Corollary 3.5 implies that $\mathsf{NSC} = \mathsf{SymSC}$. (For more results on NSC, see [1].)

Remark 3.6. Theorem 10 of [23] states the inclusion

$$\mathsf{NSpaceTime}(S, T) \subseteq \mathsf{SymSpaceTime}(S \log(1 + T/S), T),$$

where $S = S(n)$ and $T = T(n)$ are as above. This is clearly stronger than our Corollary 3.5. However, it appears that there may be a problem with the proof of Theorem 10 in [23]; see the discussion of this point in [3].

The space bound $S\log(1+T/S)$ can be achieved even with a deterministic simulation (although with a time bound exponential in the space bound), as was first shown by Denenberg [13]. By comparison, Corollary 3.5 incurs a polynomial slowdown, although the space bound is the same, unless S is rather close to T. Quoting from Denenberg [13]: "Unfortunately, this improvement is significant only when the time bound is at most slightly greater than the space bound. ... There is real improvement when the time bound is small; for example, if $T = S(\log S)^d$ for some d then the [naïve] simulation runs in space $O(S\log S)$ and the improved one uses space $O(S\log\log S)$." Although it is plausible that the stronger inclusion claimed in [23, Theorem 10] does hold, we have not been able to verify this.

4 Remarks on reversibility

The concept of reversibility is related to determinism in a way that is analogous to the relationship of symmetry to nondeterminism. The theoretical study of reversibility in computation was initiated in different contexts by Lecerf ([22]) and Bennett ([5]).

Let us now consider this relationship for auxiliary pushdown automata and thus for languages reducible to context-free languages.

Definition 4.1. We call a deterministic auxiliary pushdown automaton *reversible* if it is also backdeterministic. That is: for each configuration C, there is at most one possible configuration D such that $D \vdash C$.

By $\mathsf{RevAuxPDA\text{-}TIME}\left(n^{O(1)}\right)$ we denote the class of languages acceptable by reversible auxiliary pushdown automata in polynomial time.

Since deterministic and reversible computation have equivalent computational power both in the setting of space complexity [21] and time complexity [5]), one might be tempted to expect that this would hold for time-bounded auxiliary push-down automata as well. Note however, that there is an oracle relative to which deterministic computation is strictly more powerful than reversible computation, for machines with simultaneous time and space bounds [15].

In the following paragraphs, we investigate how $\mathsf{RevAuxPDA\text{-}TIME}\left(n^{O(1)}\right)$ relates to nearby complexity classes.

The complexity class $\mathsf{RUL} = \mathsf{RUspace}(\log n)$ was defined by Buntrock et al. [7] and has been studied subsequently by the authors [2, 20]. A language L is in RUL if there is an NL machine accepting L with the property that, on every input, the graph of reachable configurations is a tree, and there is a unique accepting configuration. It was shown by Buntrock et al. that RUL and a related class known as $\mathsf{ReachFewL}$ are contained in $\mathsf{Log}(\mathsf{DCFL})$. Analysis of their algorithm (which simply searches through the configuration graph of a $\mathsf{ReachFewL}$ machine) shows that it is a reversible algorithm. Hence we obtain the following inclusion:

Proposition 4.2. $\mathsf{ReachFewL} \subseteq \mathsf{RevAuxPDA\text{-}TIME}\left(n^{O(1)}\right)$.

(It was shown recently that $\mathsf{ReachFewL}$ *coincides* with RUL [16].)

We close this section with a list of open questions regarding reversible AuxPDAs. In particular, there are a number of "nearby" complexity classes which one might hope to relate in some way to $\mathsf{RevAuxPDA\text{-}TIME}\left(n^{O(1)}\right)$:

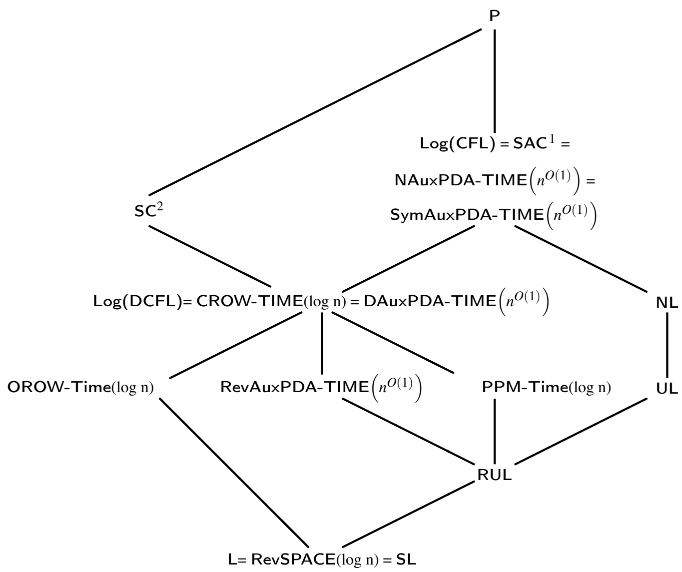

Figure 1: Inclusion relations among various subclasses of Log(CFL).

- The class DAuxPDA-TIME$\left(n^{O(1)}\right)$ was shown by Dymond and Ruzzo to coincide with the class CROW-TIME$(\log n)$ consisting of those languages accepted by PRAMs obeying the owner-write restriction working in logarithmic time with a polynomial number of processors ([14]). Rossmanith showed that the subclass OROW-TIME$(\log n)$ that results by replacing "concurrent read" by "owner read" contains L ([28]). Is there a relationship between RevAuxPDA-TIME $\left(n^{O(1)}\right)$ and OROW-TIME$(\log n)$?
- Another way to limit CROW-PRAMs is to restrict the set of arithmetic operations that the PRAMs have in their instruction set. By imposing this restriction, Cook and Dymond [11] introduced *parallel pointer machines*. (See also [18].) The class PPM-Time$(\log n)$ consisting of languages accepted by parallel pointer machines in logarithmic time contains not only L, but also RUL [2]. Is there a relationship between RevAuxPDA-TIME $\left(n^{O(1)}\right)$ and PPM-Time$(\log n)$?
- We were able to show that symmetric AuxPDAs running for polynomial time are able to compute with a pushdown of polylogarithmic height. Is this possible for reversible machines as well? It might be that the answer to this question is connected to our questions about the relation between RevAuxPDA-TIME $\left(n^{O(1)}\right)$ and the PRAM classes mentioned above.
- There are natural variants of the Monotone Planar Circuit Value Problem that are hard for L and lie in Log(DCFL) [24, 8]. Related techniques were used, in order to show that the "One-Dimensional Sandpile" problem also lies in Log(DCFL) [25]. It is not at all clear to us that the Log(DCFL) algorithms for these problems can be implemented on reversible AuxPDAs. Nor is it clear to us that either of these problems is reducible to the other, or that either one lies in any of the other classes between L and Log(DCFL) discussed in this section.

The known relations among these classes are summarized in Figure 1.

Acknowledgment

Some of this work was performed while the first author was a visiting scholar at the University of Cape Town. We gratefully acknowledge helpful discussions with Nutan Limaye, Luke Friedman, Shiva Kintali, and Fengming Wang.

References

[1] MANINDRA AGRAWAL, ERIC ALLENDER, SAMIR DATTA, HERIBERT VOLLMER, AND KLAUS W. WAGNER: Characterizing small depth and small space classes by operators of higher type. *Chicago J. Theor. Comput. Sci.*, 2000(2), 2000. See also at ECCC. [doi:10.4086/cjtcs.2000.002] 209

[2] ERIC ALLENDER AND KLAUS-JÖRN LANGE: $\mathsf{RUSPACE}(\log n) \subseteq \mathsf{DSPACE}(\log^2 n/\log\log n)$. *Theory of Comput. Syst.*, 31(5):539–550, 1998. Preliminary version in ISAAC'96. See also at ECCC. [doi:10.1007/s002240000102] 210, 211

[3] ERIC ALLENDER AND KLAUS-JÖRN LANGE: Symmetry coincides with nondeterminism for time-bounded auxiliary pushdown automata. In *Proc. 25th IEEE Conf. on Computational Complexity (CCC'10)*, pp. 172–180. IEEE Comp. Soc. Press, 2010. [doi:10.1109/CCC.2010.24] 209

[4] ERIC ALLENDER, KLAUS REINHARDT, AND SHIYU ZHOU: Isolation, matching, and counting: Uniform and nonuniform upper bounds. *J. Comput. System Sci.*, 59(2):164–181, 1999. Preliminary versions in CCC'98 (also available at ECCC) and an MFCS'98 workshop (available at ECCC). [doi:10.1006/jcss.1999.1646] 201

[5] CHARLES H. BENNETT: Logical reversibility of computation. *IBM J. Res. Develop.*, 17(6):525–532, 1973. [doi:10.1147/rd.176.0525] 210

[6] ALLAN BORODIN, STEPHEN A. COOK, PATRICK W. DYMOND, WALTER L. RUZZO, AND MARTIN TOMPA: Two applications of inductive counting for complementation problems. *SIAM J. Comput.*, 18(3):559–578, 1989. Preliminary version in SCT'88. See erratum. [doi:10.1137/0218038] 200

[7] GERHARD BUNTROCK, BIRGIT JENNER, KLAUS-JÖRN LANGE, AND PETER ROSSMANITH: Unambiguity and fewness for logarithmic space. In *Proc. 8th Internat. Conf. Fundamentals of Computation Theory (FCT'91)*, volume 529, pp. 168–179. Springer, 1991. [doi:10.1007/3-540-54458-5_61] 210

[8] TANMOY CHAKRABORTY AND SAMIR DATTA: One-input-face MPCVP is hard for L, but in $\mathsf{LogDCFL}$. In *Proc. Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *LNCS*, pp. 57–68. Springer, 2006. See also at ECCC. [doi:10.1007/11944836_8] 211

[9] STEPHEN A. COOK: Characterizations of pushdown machines in terms of time-bounded computers. *J. ACM*, 18(1):4–18, 1971. Preliminary version in STOC'69. [doi:10.1145/321623.321625] 200

[10] STEPHEN A. COOK: Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *Proc. 11th STOC*, pp. 338–345. ACM Press, 1979. [doi:10.1145/800135.804426] 200

[11] STEPHEN A. COOK AND PATRICK W. DYMOND: Parallel pointer machines. *Comput. Complexity*, 3(1):19–30, 1993. [doi:10.1007/BF01200405] 211

[12] MATEI DAVID AND PERIKLIS A. PAPAKONSTANTINOU: Tradeoff lower lounds for stack machines. *Comput. Complexity*, 23(1):99–146, 2014. Preliminary version in CCC'10. [doi:10.1007/s00037-012-0057-1] 200

[13] LARRY DENENBERG: A note on Savitch's theorem. Technical Report 25-81, Harvard University, 1981. 210

[14] PATRICK W. DYMOND AND WALTER L. RUZZO: Parallel RAMs with owned global memory and deterministic context-free language recognition. *J. ACM*, 47(1):16–45, 2000. Preliminary version in ICALP'86. [doi:10.1145/331605.331607] 200, 211

[15] MICHAEL P. FRANK AND M. JOSEPHINE AMMER: Relativized separation of reversible and irreversible space-time complexity classes. Available at CiteSeerX. 210

[16] BRADY GARVIN, DERRICK STOLEE, RAGHUNATH TEWARI, AND N. VARIYAM VINODCHANDRAN: ReachFewL = ReachUL. *Comput. Complexity*, 23(1):85–98, 2014. Preliminary version in COCOON'11. See also at ECCC. [doi:10.1007/s00037-012-0050-8] 210

[17] SHIVA PRASAD KINTALI: Realizable paths and the NL vs L problem. *Electron. Colloq. on Comput. Complexity (ECCC)*, 17:158, 2010. ECCC. See also preprint at arXiv and doctoral thesis at OATD. 201, 203

[18] TAK WAH LAM AND WALTER L. RUZZO: The power of parallel pointer manipulation. In *Proc. 1st Ann. ACM Symp. on Parallel Algorithms and Architectures (SPAA'89)*, pp. 92–102. ACM Press, 1989. [doi:10.1145/72935.72946] 211

[19] KLAUS-JÖRN LANGE: Are there formal languages complete for $\mathsf{SymSPACE}(\log n)$? In *Foundations of Computer Science: Potential - Theory - Cognition, to Wilfried Brauer on the occasion of his sixtieth birthday.*, pp. 125–134. Springer, 1997. [doi:10.1007/BFb0052081] 209

[20] KLAUS-JÖRN LANGE: An unambiguous class possessing a complete set. In *Proc. 14th Symp. Theoretical Aspects of Comp. Sci. (STACS'97)*, volume 1200 of *LNCS*, pp. 339–350. Springer, 1997. [doi:10.1007/BFb0023471] 210

[21] KLAUS-JÖRN LANGE, PIERRE MCKENZIE, AND ALAIN TAPP: Reversible space equals deterministic space. *J. Comput. System Sci.*, 60(2):354–367, 2000. Preliminary version in CCC'97. [doi:10.1006/jcss.1999.1672] 201, 210

[22] YVES LECERF: Machines de Turing réversibles. Récursive insolubilité en $n \in N$ de l'équation $u = \theta^n u$, où θ est un 'isomorphism de codes'. *Comptes Rendus*, 257:2597–2600, 1963. Available at Gallica. 210

[23] HARRY R. LEWIS AND CHRISTOS H. PAPADIMITRIOU: Symmetric space-bounded computation. *Theoret. Comput. Sci.*, 19(2):161–187, 1982. Preliminary version in ICALP'80. [doi:10.1016/0304-3975(82)90058-5] 199, 200, 203, 209, 210

[24] NUTAN LIMAYE, MEENA MAHAJAN, AND JAYALAL M. N. SARMA: Upper bounds for monotone planar circuit value and variants. *Comput. Complexity*, 18(3):377–412, 2009. Preliminary version in STACS'06. See also at ECCC. [doi:10.1007/s00037-009-0265-5] 211

[25] PETER BRO MILTERSEN: The computational complexity of one-dimensional sandpiles. *Theory Comput. Syst.*, 41(1):119–125, 2007. Preliminary version in CiE'05. [doi:10.1007/s00224-006-1341-8] 211

[26] OMER REINGOLD: Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, 2008. Preliminary version in STOC'05 See also at ECCC. [doi:10.1145/1391289.1391291] 199, 209

[27] KLAUS REINHARDT AND ERIC ALLENDER: Making nondeterminism unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, 2000. Preliminary version in FOCS'97. See also at ECCC. [doi:10.1137/S0097539798339041] 201

[28] PETER ROSSMANITH: The owner concept for PRAMs. In *Proc. 8th Symp. Theoretical Aspects of Comp. Sci. (STACS'91)*, volume 480 of *LNCS*, pp. 172–183. Springer, 1991. [doi:10.1007/BFb0020797] 211

[29] WALTER J. SAVITCH: Relationships between nondeterministic and deterministic tape complexities. *J. Comput. System Sci.*, 4(2):177–192, 1970. Preliminary version in STOC'69. [doi:10.1016/S0022-0000(70)80006-X] 209

[30] IVAN HAL SUDBOROUGH: On the tape complexity of deterministic context-free languages. *J. ACM*, 25(3):405–414, 1978. Preliminary version in STOC'76. [doi:10.1145/322077.322083] 200

[31] TILL TANTAU: Logspace optimization problems and their approximability properties. *Theory Comput. Syst.*, 41(2):327–350, 2007. Preliminary version in FCT'05. See also at ECCC. [doi:10.1007/s00224-007-2011-1] 199

[32] H. VENKATESWARAN: Personal Communication. The author previously announced that an earlier version ([34]) had an incomplete proof. 201

[33] H. VENKATESWARAN: Properties that characterize LOGCFL. *J. Comput. System Sci.*, 43(2):380–404, 1991. Preliminary version in STOC'87. [doi:10.1016/0022-0000(91)90020-6] 200

[34] H. VENKATESWARAN: Derandomization of probabilistic auxiliary pushdown automata classes. In *Proc. 21st IEEE Conf. on Computational Complexity (CCC'06)*, pp. 355–370. IEEE Comp. Soc. Press, 2006. [doi:10.1109/CCC.2006.16] 201, 214

AUTHORS

Eric Allender
professor
Rutgers University, New Brunswick, NJ
allender@cs.rutgers.edu
http://www.cs.rutgers.edu/~allender

Klaus-Jörn Lange
professor
Universität Tübingen
lange@informatik.uni-tuebingen.de
http://fuseki.informatik.uni-tuebingen.de/en/mitarbeiter/lange

ABOUT THE AUTHORS

ERIC ALLENDER is a distinguished professor at Rutgers University. He has been at Rutgers since receiving his Ph. D. in 1985 at Georgia Tech, under the supervision of Kim N. King. While at Georgia Tech, he was the Backbone of the Seed and Feed Marching Abominable and he still plays trombone from time to time. He did his undergraduate work at the University of Iowa. He is a Fellow of the ACM, and currently serves as Editor-in-Chief of ACM Transactions on Computation Theory. Circuit complexity, Kolmogorov complexity, and complexity classes are his main research interests. He and his wife find happiness on the dance floor and toiling in their garden.

KLAUS-JÖRN LANGE has been a professor at the University of Tübingen since 1995. Before that, he was a professor of theoretical computer science at the Technical University of Munich for eight years. He got his degrees at the University of Hamburg under the supervision of Wilfried Brauer. His main field of research is the study of the relations between formal languages and complexity theory. As a university student he invested a lot of time (maybe too much) in the game of Go reaching the rank of sandan, and served some years as the head of the Go club in Hamburg.