Are there formal languages complete for $SymSPACE(\log n)$?

Klaus-Jörn Lange

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
Sand 13, D72076 Tübingen

Abstract. This article discusses the existence of $SymSPACE(\log n)$–complete formal languages. It is shown that a recent approach of Alvarez and Greenlaw to define symmetric versions of one-way devices doesn't lead to $SymSPACE(\log n)$–complete problems when applied to linear context-free or to one-counter languages.

1 Introduction

Investigating the fundamental relation of determinism and nondeterminism the intermediate concept of symmetry has been introduced by Lewis and Papadimitriou [8]. While nondeterminism and symmetry are easily seen to be equivalent for time bounded classes, the case of space bounded computations seems to be more difficult. It is not known whether the inclusion $SymSPACE(\log n) \subseteq NSPACE(\log n)$ is strict or not. There are results indicating that these two classes should be different. Several interesting problems are complete for symmetric space, the most important example being $UGAP$, the connectivity problem for undirected graphs [2,8].

There are many close connections between formal languages (defined by one-way devices) and complexity classes. But these in most cases pertain to nondeterministic and to deterministic classes. Unfortunately, it seems hard to relate the concept of symmetry, which is apparently based on two-way devices, with formal languages, which are built on models with a one-way input. In a recent report, Alvarez and Greenlaw introduced symmetry for one-way finite automata [2]. As a first result of this paper, the complexity of this model is determined. Second, their concept is extended to two families of formal languages related to the class $NSPACE(\log n)$. It turns out that in neither of the two cases we get a characterization of $SymSPACE(\log n)$, but stay at $NSPACE(\log n)$ or come down to NC^1.

2 Preliminaries

For notions of formal languages or complexity theory the reader is referred to standard text books. For the circuit class NC^1 and its relationship to the regular languages we refer to the work of Barrington [4].

2.1 Formal Languages and Complexities

There are characterizations for relevant complexity classes in terms of families of formal languages in the way that a family of formal languages $\mathcal{A}$ is contained in a complexity class $\mathcal{B}$ and that $\mathcal{A}$ contains a $\mathcal{B}$–complete language. For example, the family of regular languages REG corresponds to the class NC^1 [4], both the linear context-free languages LIN and the one-counter languages characterize the class $NSPACE(\log n)$ [6,13,14], the context-free languages CFL are related to the class $NAuxPDA\text{-}TISP(pol, \log n)$ [15], and the indexed languages are representative fot the class NP [12]. Via deterministic one-way automata or grammars restricted by LL(k)- or LR(k)–conditions these relations carry over to the corresponding deterministic classes. But nothing like that is known for symmetry.

3 Symmetry

The intuition behind the idea of symmetry is to allow to go computational steps backwards. Symmetry is a restricted type of nondeterminism and is able to simulate determinism because of the tree-like structure of deterministic configuration graphs. To define this concept formally needs some care. Lewis and Papadimitriou used Turing machines which have something like working heads of width two. We avoid to go into these details and refer the reader to the article of Lewis and Papadimitriou [8]. We will use terms like symmetric time or symmetric space and notations like $SymSPACE(\log n)$ or $SymTISP(f, g)$, the later denoting the class of all languages recognized by symmetric machines which are simultaneously time bounded by $O(f)$ and space bounded by $O(g)$. By Lewis and Papdimitriou we know:

Theorem 1. *Nondeterministic time coincides with symmetric time.*

It is unknown whether the inclusion of symmetric space in nondeterministic space is strict.

The reachability problem for undirected graphs is $SymSPACE(\log n)$–complete [8]. A collection of further complete problems is contained in the compendium of Alvarez and Greenlaw [2].

There are results indicating that in case of space bounded classes symmetric computations are more restricted than nondeterministic ones. None of the following inclusions of $SymSPACE(\log n)$ is known to hold for $NSPACE(\log n)$:

Theorem 2. **a)** $SymSPACE(\log n) \subseteq \bigoplus SPACE(\log n)$ *[7]*
b) $SymSPACE(\log n) \subseteq SC^2$ *[10]*
c) $SymSPACE(\log n) \subseteq DSPACE(\log^{4/3} n)$ *[3]*
d) $SymSPACE(\log n) \subseteq CREW\text{-}TIME(\log n \log\log n)$ *[5]*

Analyzing Savitch's algorithm, Lewis and Papadimitriou were able to show the following relationship:

Theorem 3. $NTISP(f, g) \subseteq SymTISP(f \cdot g, g \cdot \log f)$

In particular, $NSPACE(\log n)$ is contained in $SymTISP(pol, \log^2 n)$, the symmetric analogue of SC^2. It is interesting to compare this inclusion with Nisan's result $SymSPACE(\log n) \subseteq SC^2 = DTISP(pol, \log^2 n)$.

Applying the ideas used in Theorem 3 to the algorithm of Lewis, Stearns, and Hartmanis instead to that of Savitch, the equivalence of nondeterminism and symmetry for polynomially time bounded auxiliary pushdown automata will be shown in [1]:

Theorem 4. $NAuxPDA\text{-}TISP(pol, \log n) = SymAuxPDA\text{-}TISP(pol, \log n)$

This result is not restricted to logarithmic space and polynomial time bounds, but holds in general if space bounds and time bounds are in an exponential relation.

3.1 Formal Languages and Symmetry

We now want to consider the possibilities of characterizing symmetric complexity classes in terms of formal languages. For time bounded classes there is no need to do so because of Theorem 1. Neither, this is necessary for polynomially time bounded auxiliary pushdown automata, i.e.: the class $LOG(CFL)$ [15], because of Theorem 4.

The class NC^1 is characterized by the regular languages [4]. Since deterministic and nondeterministic finite automata are equivalent, there seems to be no need to look for characterizations of symmetry. On the other hand, the relation "determinism $\leq$ symmetry $\leq$ nondeterminism" is valid only in the two-way case. and there was up to now no notion of a symmetric finite automaton with one-way input. A definition of this notion was given by Alvarez and Greenlaw [2]. They introduced symmetric finite automata by adding dual transitions, but without reverting the direction of the reading head. That is, an NFA $A = (Q, \Sigma, \delta, q_0, Q_f)$ with $\delta \subseteq Q \times \Sigma \times Q$ is *symmetric* iff with each $(p, a, q) \in \delta$ also (q, a, p) is an element of δ. Alvarez and Greenlaw were able to show that the emptiness for symmetric finite automata is indeed $SymSPACE(\log n)$–complete.

Observe that the languages accepted by symmetric finite automata are a proper subfamily of REG. In particular, no finite regular language containing a nonempty word can be symmetric. This indicates that this version is a strong restriction not able to simulate determinism. Nevertheless, the membership-problem of symmetric finite automata keeps the complexity of the whole class REG:

Theorem 5. *There exists a symmetric finite automaton which accepts a language with an NC^1–complete membership problem.*

Proof: Consider the DFA $A = (Q, \Sigma, \delta, id, \{id\})$ where $Q := \Sigma := S_5$, the group of all permutations over five elements, id is the identity, and δ is defined by $\delta(\pi, \sigma) := \sigma \circ \pi$, i.e.: from permutation π reading permutation σ we enter the composition of π followed by σ. Barrington showed that A accepts an NC^1–complete language [4]. A first idea now could be to augment A by the reversals

of all transitions. But the resulting symmetric automaton no longer represents S_5 since it would be necessary to mirror a transition $\delta(\pi,\sigma)=\rho$ by $\delta(\rho,\sigma^{-1})=\pi$ instead of $\delta(\rho,\sigma)=\pi$ in order to preserve the structure of the S_5.

Instead we will use the following easy trick, which will be used several times throughout the paper. The simple idea is to intertwine all the words of a language with a new symbol a. We set $B := (Q' := Q_a \cup Q, \Sigma' := \Sigma \cup \{a\}, \delta', id, \{id\})$, where a is a new input symbol, Q_a a copy of Q, and δ' is defined by $\delta'(\pi,a) := \pi_a$ and $\delta'(\pi_a,\sigma) := \delta(\pi,\sigma)$ for $\pi \in Q$ and $\sigma \in \Sigma$. $\delta'(\pi_a,a)$ and $\delta'(\pi,\sigma)$ are undefined. If we define the monomorphism $h : \Sigma^* \longrightarrow (\Sigma \cup \{a\})^*$ by $h(\sigma) := a\sigma$, we have $L(B) = h(L(A))$. Since h is one-to-one, $L(B)$ is NC^1–complete.

Now let C be the symmetric closure of A, i.e.: $C = (Q', \Sigma', \tau, id, \{id\})$, where τ is the defined by $\tau := \{(p,b,\delta(p,b))|p \in Q', b \in \Sigma'\} \cup \{(\delta(p,b),b,p)|p \in Q', b \in \Sigma'\}$. Clearly, every word accepted by C using a reversed transition of the type $(\delta(p,b),b,p)$ must contain a subword in $\Sigma^2 \cup \{aa\}$ and thus $L(C) \cap (a\Sigma)^* = L(B)$. Hence, $L(C)$ is NC^1–complete, as well. □

After discussing the situation for the classes NP, $LOG(CFL)$, and NC^1, the case of nondeterministic space remains to be treated. The class $NSPACE(\log n)$ is characterized by two well-known subclasses of CFL: the counter languages and the linear context-free languages. The latter coincide with the languages accepted by pushdown automata which make only one turn. But we will consider the representation by grammars since this will make our treatment easier. In the following we extend the approach of Alvarez and Greenlaw to these two families of formal languages.

We will have two possibilities to define symmetry with respect to the access of the pushdown store. In the *crossed version* we let the reversal of a transition which pushes a symbol or increments a counter be one which pops this symbol from the stack or decrements the counter. Since in the Alvarez and Greenlaw approach this principle is not applied to the input head, we also consider the *straight version* in which the reversal of a push (or increment) stays a push (or increment) and the reversal of a pop (or decrement) is a pop (or decrement).

One-Counter Languages We now consider nondeterministic counter automata with emptiness test. These are also called iterated counter automata. In order to investigate symmetric versions of these, we use them in the following normal form. The set of transitions of a counter automaton $A = (Q, \Sigma, \tau, q_0, Q_f)$ consists in four sets of transitions $\tau = (\tau_r, \tau_i, \tau_d, \tau_e)$. $\tau_r \subseteq Q \times \Sigma \times Q$ contains the *reading transitions* which neither test nor change the counter. All other transitions do not move the input head. $\tau_i \subseteq Q \times Q$ contains the *incrementing transitions* which cause the counter to be incremented by one. $\tau_d \subseteq Q \times Q$ contains the *decrementing transitions* which decrement the counter by one. The decrementing transitions can only be executed if the counter is nonempty. $\tau_e \subseteq Q \times Q$ contains the *emptiness tests*. A transition $(p,q) \in \tau_e$ can only be used if the counter is empty. We don't need explicit tests for nonemptiness since these could be simulated by a decrement followed by an increment.

As usual, a configuration of a counter automaton is a triple (p, i, w) where p is the actual state, i the current value of the counter, and w the remaining input to be read.

The following definition of symmetry for counter automata is given in the two versions described above: one changing the direction of the pushdown head when reversing a transition and one keeping the direction of the pushdown head.

Definition 6. **a)** A counter automaton $A = (Q, \Sigma, (\tau_r, \tau_i, \tau_d, \tau_e), q_0, Q_f)$ is called *symmetric in the crossed version* or *crossed symmetric* if we have

$$(p, a, q) \in \tau_r \text{ implies } (q, a, p) \in \tau_r, (p, q) \in \tau_i \text{ implies } (q, p) \in \tau_d,$$

$$(p, q) \in \tau_d \text{ implies } (q, p) \in \tau_i, \text{ and } (p, q) \in \tau_e \text{ implies } (q, p) \in \tau_e.$$

b) A counter automaton $A = (Q, \Sigma, (\tau_r, \tau_i, \tau_d, \tau_e), q_0, Q_f)$ is called *symmetric in the straight version* or *straight symmetric* if we have

$$(p, a, q) \in \tau_r \text{ implies } (q, a, p) \in \tau_r, (p, q) \in \tau_i \text{ implies } (q, p) \in \tau_i,$$

$$(p, q) \in \tau_d \text{ implies } (q, p) \in \tau_d, \text{ and } (p, q) \in \tau_e \text{ implies } (q, p) \in \tau_e.$$

We will now show that these two versions of symmetric counter automata behave differently w.r.t. complexity. While crossed symmetry stays $NSPACE(\log n)$–complete, straight symmetry allows only for the acceptance of regular languages. To show the hardness in the crossed case we first exhibit an example of a restricted counter automaton accepting an $NSPACE(\log n)$–complete language.

Lemma 7. *There is counter automaton $A = (Q, \Sigma, (\tau_r, \tau_i, \tau_d, \tau_e), q_0, Q_f)$ accepting an $NSPACE(\log n)$–complete language fulfilling the following properties: a) No valid computation of A contains two or more successive nonreading steps. In particular, this means that a state reached by a nonreading transition can only be left by reading transitions. b) For each nonreading transition $(p, q) \in \tau_i \cup \tau_d \cup \tau_e$ there is no other transition in τ leading to q.*

Proof: We follow the usual construction [14]. The language accepted by A represents the reachability problem for topologically sorted, acyclic graphs where the inner nodes have outdegree two. A graph will be accepted if there is a path from node v_0 to v_n where v_n is the last node in the order. A will also accept inputs not following this paradigm, but not well-presented graphs without that path. An inner node v_i with its two successors v_j and v_k will be represented by a word $b^i c b^j c b^k d$. The graph is given by the sorted sequence of these words followed by the suffix $b^n e$. For lack of space we give no proof but instead give the exact construction of A, only. $Q := \{q_0, q_1, \cdots, q_{12}\}$, $\Sigma := \{b, c, d, e\}$, $Q_f := \{q_{12}\}$, $\tau_r := \{(q_0, c, q_1), (q_0, c, q_4), (q_0, e, q_{12}), (q_1, c, q_3), (q_2, b, q_1), (q_3, b, q_3), (q_3, d, q_7), (q_4, b, q_4), (q_4, c, q_5), (q_5, d, q_7), (q_6, b, q_5), (q_7, b, q_8), (q_8, b, q_8), (q_8, c, q_8) (q_8, d, q_7), (q_9, b, q_{10}), (q_{11}, b, q_{10})\}$, $\tau_i := \{(q_1, q_2), (q_5, q_6)\}$, $\tau_d := \{(q_7, q_9), (q_{10}, q_{11})\}$, $\tau_e := \{(q_{10}, q_0)\}$. □

Theorem 8. *There is a counter automaton which is symmetric in the crossed version recognizing an $NSPACE(\log n)$–complete language.*

Proof: Let $A = (Q, \Sigma, (\tau_r, \tau_i, \tau_d, \tau_e), q_0, Q_f)$ be the counter automaton of the previous lemma. We will now apply the trick used in Theorem 5 on A, intertwining $L(A)$ with a new terminal symbol a. Set $B := (Q' := Q_a \cup Q, \Sigma' := \Sigma \cup \{a\}, \tau' := (\tau'_r, \tau_i, \tau_d, \tau_e), q_0, Q_f)$ where Q_a is a copy of Q and $\tau'_r := \{(p, a, p_a) | p \in Q\} \cup \{(p_a, b, q) \mid (p, b, q) \in \tau_r\}$. Using again the monomorphism $h : \Sigma^* \longrightarrow \Sigma'^*$ defined by $h(b) := ab$, we have $L(B) = h(L(A))$. Since h is one-to-one, $L(B)$ is $NSPACE(\log n)$-complete. In addition, B still fulfills the properties of Lemma 7.

We now consider the crossed symmetric closure of B: set $C := (Q', \Sigma', \hat{\tau} := (\hat{\tau}_r, \hat{\tau}_i, \hat{\tau}_d, \hat{\tau}_e), q_0, Q_f)$ where $\hat{\tau}_r := \tau'_r \cup \{(q, b, p) \mid (p, b, q) \in \tau'_r\}$, $\hat{\tau}_i := \tau_i \cup \tau_d^{-1}$, $\hat{\tau}_d := \tau_d \cup \tau_i^{-1}$, and $\hat{\tau}_e := \tau_e \cup \tau_e^{-1}$. Clearly we have $L(B) \subseteq L(C)$. Now consider a computation R of C accepting some word v in $(a\Sigma)^*$. Obviously, as in Theorem 5, R cannot use any reversal of a reading transition since only the reading transitions switch between the states of Q and of Q_a. Otherwise, the input has to contain a subword in $\Sigma^2 \cup \{aa\}$. But if R contains a reversal (q, p) of a nonreading transition (p, q) we know by the construction that q has been reached by (p, q) or by the inverse of a reading transition. Hence, the only way R could have reached q was using (p, q). But by construction of crossed symmetry, (p, q) and (q, p) cancel each other and can be removed from R resulting in a valid computation accepting v. The last possibility to consider is that C leaves the initial state by the inverse of a nonreading transition. But a simple case analysis shows that this either will end in a cancellation as above or will lead to the acceptance of an element not in $(a\Sigma)^*$. In total, we get $L(C) \cap (a\Sigma)^* = L(B)$ which gives us the $NSPACE(\log n)$–completeness of $L(C)$. □

While crossed symmetry of counter automata leads to the same complexity of the word problem as unrestricted nondeterminism, straight symmetry is a strong restriction and diminishes the complexity from $NSPACE(\log n)$ down to NC^1 since now only regular languages can be accepted.

Theorem 9. *Languages accepted by counter automata which are symmetric in the straight version are regular and hence in NC^1.*

Proof: Let $A = (Q, \Sigma, (\tau_r, \tau_i, \tau_d, \tau_e), q_o, Q_f)$ be symmetric in the straight version. We now construct a nondeterministic (not symmetric) finite automaton $A' = (Q', \Sigma, \tau', q'_0, Q'_f)$ which accepts $L(A)$. The idea is to use the fact that in a straight symmetric automaton every increment or decrement (p, q) can be repeated using the reversed transition to $(p, q)(q, p)(p, q)$ and so on. Instead of increasing or decreasing by one we can now add or subtract an arbitrary odd number. The state set of A' is $Q' := Q \times \{0, 1\} \times \{i, d, e\}$. The second component of a state $\langle p, x, y\rangle$ will represent the value of the counter modulo 2. The third component will mark the type of the last preceding nonreading transition. $q'_0 := \langle q_0, 0, e\rangle$, $Q'_f := Q_f \times \{0, 1\} \times \{i, d, e\}$, and τ' is set to be the following union:

$$\{(\langle p, x, y\rangle, a, \langle q, x, y\rangle) \mid (p, a, q) \in \tau_r, x \in \{0, 1\}, y \in \{i, d, e\}\} \cup$$

$$\{(\langle p, x, y\rangle, \langle q, 1 - x, i\rangle) \mid (p, q) \in \tau_i, x \in \{0, 1\}, y \in \{i, d, e\}\} \cup$$

$$\{(\langle p, x, y\rangle, \langle q, 1 - x, d\rangle) \mid (p, q) \in \tau_d, x \in \{0, 1\}, y \in \{i, d\}\} \cup$$

$$\{(\langle p,0,y\rangle,\langle q,0,e\rangle) \mid (p,q) \in \tau_e, y \in \{d,e\}\}.$$

The construction assures that a decrement cannot be applied when the last preceding nonreading transition was a successful test for emptiness. Further on, a test cannot be successfully passed if the last preceding nonreading transition was an increment or if the counter contains an odd value.

Clearly we have $L(A) \subseteq L(A')$ since every accepting computation of A can be mimicked in A' by forgetting the counter. To get the other inclusion it is sufficient to show the following: if there is a path of A' leading from a state $\langle p,0,e\rangle$ to a state $\langle q,x,y\rangle$ while reading the input word $v \in \Sigma^*$ then there is a valid computation $(p_0,i_0,w_0) \vdash (p_1,i_1,w_1) \cdots \vdash (p_n,i_n,w_n)$ of A such that $(p_0,i_0,w_0) = (p,0,v)$, $p_n = q$, w_n is the empty word, and $x = i_n$ modulo 2. The proof is done by induction over k, the length of the path of A'.

The statement obviously holds for $k = 1$. Now let there be a path in A' of length $k+1$ leading from $\langle p,0,e\rangle$ to $\langle q,x,y\rangle$ reading the word v. Further on, let $\langle p',x',y'\rangle$ be the state in this path reached after k steps. There is a decomposition $v = v'a$ where v' is read during the first k steps, and a is the empty word or an element of Σ depending of the type of the transition of the last step. By induction we know that there is a valid computation of A leading from configuration $(p,0,v)$ to configuration (p',i,a) for some i reading the word v'. If the last transition of A' mimicks an element of τ_r or of τ_i then obviously the computation of A reaching (p',i,a) can be extended as claimed in the statement. This is also true if we simulate an element of τ_d and $i > 0$. If $i = 0$, i.e.: the counter is empty, then by construction we know that the last state in this computation with a third component unequal d must have a third component i, but not e, since there is no way in A' from a third component e directly to a d. There is always an i in between. But by replacing in the existing computation of A this increment step S by the sequence S followed by the reversal of S followed by S we can increase the value of the counter by two compared to the original computation. Repeating this process we can change the value of the counter by an arbitrary even number so that we can finally extend the modified computation by the decrement. Further on, there is no test for emptiness after S so we obtain a valid computation of A.

The last case to consider is that A' mimicks an element T of τ_e. In this case we know $x = x' = 0$ and hence i is even. If $i = 0$ we consistently can extend the computation of A by T. Let's assume $i > 0$. By construction we then know that y' is either d or e. But if it were e the last nonreading transition would have been a successfully passed test for emptiness which contradicts $i > 0$. Hence $y' = d$ and the last nonreading transition was a decrement. Using the method described above we can decrease the value of the counter by an arbitrary even number which finally leads us to an empty counter where we successfully can apply T. Since there is no decrement after T, the modified computation is valid.

□

Linear Languages We now consider linear grammars. We will always assume them to be in normal form, that is a production is of the form $A \to a, A \to aB$, or $A \to Ba$. The two versions of symmetry now look as follows:

Definition 10. a) A linear grammar $G = (V, \Sigma, P, S)$ is called *symmetric in the crossed version* or *crossed symmetric* if we have

$$(A \to aB) \in P \text{ implies } (B \to Aa) \in P \text{ and}$$

$$(A \to Ba) \in P \text{ implies } (B \to aA) \in P.$$

b) A linear grammar $G = (V, \Sigma, P, S)$ is called *symmetric in the straight version* or *straight symmetric* if we have

$$(A \to aB) \in P \text{ implies } (B \to aA) \in P \text{ and}$$

$$(A \to Ba) \in P \text{ implies } (B \to Aa) \in P.$$

The word problem for linear context free languages is $NSPACE(\log n)$–complete ([6,13]). This complexity is not hidden in the knowledge of the place of the terminating rule:

Lemma 11. *There is a linear grammar $G = (V, \Sigma, P, S)$ generating a language complete for $NSPACE(\log n)$ such that there is a terminal symbol $\$ \in \Sigma$ which is the only used in terminating rules and which does not occur in nonterminating rules, i.e.: each rule in P is of the form $A \to \$, A \to aB$, or $A \to Ba$ for $a \neq \$$.*

Theorem 12. *There is a crossed symmetric linear grammar generating a language which is $NSPACE(\log n)$–complete.*

Proof: We start with a grammar $G = (V, \Sigma, P, S)$ from Lemma 11. Now set $\Sigma' := \Sigma_l \cup \Sigma_r \cup \{\$\}$ where $\Sigma_l := \{b_l \mid b \in \Sigma, b \neq \$\}$ and $\Sigma_r := \{b_r \mid b \in \Sigma, b \neq \$\}$. Construct $G' := (V, \Sigma', P', S)$ by $P' := \{A \to b_l B \mid (A \to bB) \in P\} \cup \{A \to Bb_r \mid (A \to Bb) \in P\} \cup \{A \to \$ \mid (A \to \$) \in P\}$. Then $L(G') \subseteq \Sigma_l^* \$ \Sigma_r^*$ is obtained by marking all terminal symbols left of \$ with the subscript l and those to the right of \$ with the subscript r. Obviously, $L(G')$ is still $NSPACE(\log n)$-complete.

Now let's look at the crossed symmetric closure of G'. Set $G'' := (V, \Sigma', P'', S)$ where $P'' := P' \cup \{B \to bA \mid (A \to Bb) \in P\} \cup \{B \to Ab \mid (A \to bB) \in P\}$. It is easy to see that the application of a reversed production leads to a word outside of $\Sigma_l^* \$ \Sigma_r^*$. Hence $L(G'') \cap \Sigma_l^* \$ \Sigma_r^* = L(G')$ which gives us the $NSPACE(\log n)$–completeness of $L(G'')$. □

Finally, we show that straight symmetric linear grammars can have membership problems, which are $NSPACE(\log n)$–complete. The main idea is again the simple trick used in Theorem 5.

Theorem 13. *There is a straight symmetric linear grammar generating a language which is $NSPACE(\log n)$–complete.*

Proof: Let $L \subseteq \Sigma^*$ be a linear $NSPACE(\log n)$-complete language as stated in Lemma 11. Let L be generated by the linear grammar $G = (V, \Sigma, P, S)$. Let a be a new terminal symbol not in Σ and set $\Sigma' := \Sigma \cup \{a\}$. Consider the monomorphism $h : \Sigma^* \longrightarrow \Sigma'^*$ defined by $h(b) := ab$. Then $h(L) \subseteq (a\Sigma)^*$ is both linear context-free and $NSPACE(\log n)$–complete. $h(L(G))$ is generated by the grammar $G' = (V', \Sigma', P', S)$ where $V' := V \cup V_a \cup (\Sigma \times V) \cup \{X\}$. Here V_a is a copy of V, $\Sigma \times V$ is the set of new nonterminals $\{\langle b, B\rangle \mid b \in \Sigma, B \in V\}$, and X is a single new nonterminal. Further on, P' is the following set of productions: $\{A \to Xb \mid (A \to b) \in P\} \cup \{X \to a\} \cup \{A \to B_a b \mid (A \to Bb) \in P\} \cup \{B_a \to Ba \mid B \in V\} \cup \{A \to a\langle b, B\rangle \mid (A \to bB) \in P\} \cup \{\langle b, B\rangle \to bB \mid b \in \Sigma, B \in V\}$.

CLAIM: Let $A \stackrel{*}{\Longrightarrow}_{G'} \alpha B \beta$ be a derivation in G' for some $A \in V$. Then the following implications are induced by the structure of G' and can inductively be shown without difficulty:

- If $B \in V$ then $\alpha, \beta \in (a\Sigma)^*$,
- If $B = X$ or $B \in V_a$ then $\alpha \in (a\Sigma)^*$ and $\beta \in \Sigma(a\Sigma)^*$, and
- If $B \in (\Sigma \times V)$ then $\alpha \in (a\Sigma)^* a$ and $\beta \in (a\Sigma)^*$.

Now consider the straight symmetric closure of G', i.e.: $G'' := (V', \Sigma', P'', S)$ where $P'' := P' \cup \{\langle b, B\rangle \to aA \mid (A \to bB) \in P\} \cup \{B \to b\langle b, B\rangle \mid B \in V, b \in \Sigma\} \cup \{B_a \to Ab \mid (A \to Bb) \in P\} \cup \{B \to B_a a \mid B \in V\} \cup \{X \to Ab \mid (A \to B) \in P\}$. By construction, we have $L(G') \subseteq L(G'')$.

Finally, we now show $L(G'') \cap (a\Sigma)^* = L(G')$ which implies that the straight symmetric linear grammar G'' generates an $NSPACE(\log n)$-complete language. Let $S = \alpha_0 A_0 \beta_0 \stackrel{1}{\Longrightarrow} \alpha_1 A_1 \beta_1 \stackrel{1}{\Longrightarrow} \cdots \alpha_n A_n \beta_n \stackrel{1}{\Longrightarrow} w$ be a derivation in G'' of a word $w \in (a\Sigma)^*$. Here $A_i \in V'$ and $\alpha_i, \beta_i \in \Sigma'^*$. Let j be the minimal number such that in the derivation step from $\alpha_j A_j \beta_j \stackrel{1}{\Longrightarrow} \alpha_{j+1} A_{j+1} \beta_{j+1}$ a reversed production, i.e.: a production from $P'' \setminus P'$, has been used. We distinguish three cases.

Case 1: $A_j \in V$. By the claim we then have $\alpha_j, \beta_j \in (a\Sigma)^*$. But the right hand sides of the reversed rules for elements of V either end with an a or begin with an element of Σ which leads to a contradiction to w being an element of $(a\Sigma)^*$.

Case 2: $A_j \in V_a$ or $A_j = X$. By the claim we then have $\beta_j \in \Sigma(a\Sigma)^*$. But the right hand sides of the reversed productions for elements of $V_a \cup \{X\}$ always end with a symbol from Σ. Again, this contradicts the fact that $w \in (a\Sigma)^*$.

Case 3: $A_j \in (\Sigma \times V)$. By the claim we have $\alpha_j \in (a\Sigma)^* a$. But the right hand sides of the reversed productions for elements of $\Sigma \times V$ always begin with the symbol a, in contradiction to $w \in (a\Sigma)^*$.

Thus in the derivation of w in G'' no reversed production can be used which implies $w \in L(G')$ and thus $L(G'') \cap (a\Sigma)^* = L(G')$. □

4 Discussion

Despite the many recent results concerning symmetric space and the many resulting relations to other complexity classes, it seems to be very difficult to

get characterizations of classes like $SymSPACE(\log n)$ in terms of families of formal languages. The approach of Alvarez and Greenlaw, applied here to two $NSPACE(\log n)$–complete families, either leads to no restriction in complexity or leads to a collapse down to the regular languages and the class NC^1. One reason for the latter result could be the strong normal form we used. Without the total separation of reading from nonreading steps the proof of Theorem 9 wouldn't work. But considering the other results obtained here, this should again lead to $NSPACE(\log n)$–complete problems and not to $SymSPACE(\log n)$-complete ones.

Acknowledgement

I wish to thank an anonymous referee for his careful reading of the paper.

References

1. E. Allender, K.–J. Lange. Symmetry coincides with nondeterminism for time bounded auxiliary pushdown automata. in preparation, 1997.
2. C. Alvarez and R. Greenlaw. A compendium of problems complete for symmetric logarithmic space. Report TR96-039, ECCC, 6 1996.
3. R. Armoni, A. Ta-shma, A. Wigderson, and S. Zhou. $SL \subseteq L^{4/3}$. submitted, 1996.
4. D.A. Barrington. Bounded-width polynomial-size branching programs can recognize exactly those languages in NC^1. *J. Comp. System Sci.*, 38:150–164, 1989.
5. Ka Wong Chong and Tak Wah Lam. Finding connected components in $O(\log n \log\log n)$ time on the EREW PRAM. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 11–20, 1993.
6. P. Flajolet and J. Steyaert. Complexity of classes of languages and operators. Rap. de Recherche 92, IRIA Laboria, Nov. 1974.
7. M. Karchmer and A. Wigderson. On span programs. In *Proc. of the 8th IEEE Structure in Complexity Theory Conference*, pages 102–111, 1993.
8. P. Lewis and C.H. Papadimitriou. Symmetric space-bounded computation. *Theoret. Comput. Sci.*, 19:161–187, 1982.
9. P. Lewis, R. Stearns, and J. Hartmanis. Memory bounds for recognition of context-free and context-sensitive languages. In *Proc. 6th Annual IEEE Symp. on Switching Circuit Theory and Logical Design*, pages 191–209, 1965.
10. N. Nisan. $RL \subseteq SC$. In *Proc. of the 24th Annual ACM Symposium on Theory of Computing*, pages 619–623, 1992.
11. W. Ruzzo. Tree-size bounded alternation. *J. Comp. System Sci.*, 21:218–235, 1980.
12. E. Shamir and C. Beeri. Checking stacks and context-free programmed grammars accept p-complete languages. In *Proc. of 2nd ICALP*, number 14 in LNCS, pages 277–283. Springer, 1974.
13. I. Sudborough. A note on tape-bounded complexity classes and linear context-free languages. *J. Assoc. Comp. Mach.*, 22:499–500, 1975.
14. I. Sudborough. On tape-bounded complexity classes and multi-head finite automata. *J. Comp. System Sci.*, 10:62–76, 1975.
15. I. Sudborough. On the tape complexity of deterministic context-free languages. *J. Assoc. Comp. Mach.*, 25:405–414, 1978.