LCNS, Vol 583, pp. 387–400, Springer 1992

Unambiguous Simulations of Auxiliary Pushdown Automata and Circuits[*]

(Preliminary Version)

Rolf Niedermeier Peter Rossmanith

Technische Universität München, Institut für Informatik, Arcisstr. 21, D-8000 München 2

Abstract

In this paper time-bounded auxiliary push-down automata (AuxPDA), i.e. time and space bounded Turing machines with additional pushdown store, are considered. We investigate the power of *unambiguous* AuxPDA, i.e., machines that have at most one accepting computation, and ambiguity bounded AuxPDA.

Recently, it was shown by Buntrock, Hemachandra, and Siefkes that space bounded Turing machines, whose computation trees have moderate ambiguity, can be efficiently simulated by unambiguous AuxPDA with same space bound. This paper shows that such an efficient simulation is also possible for AuxPDA. The simulation incorporates no space and time penalty, unless the ambiguity is very high.

By similar methods it is shown that unambiguous AuxPDA can efficiently simulate a certain class of unambiguous, semi-unbounded fan-in circuits, which answers an open question posed by Lange.

Finally, obliviousness for AuxPDA is considered and it is proved that unambiguous AuxPDA work w.l.o.g. with a very limited amount of space on the pushdown store, a result that is already known for deterministic and nondeterministic AuxPDA.

1 Introduction

One of the oldest problem in complexity theory is the relationship between deterministic and nondeterministic complexity classes. One approach to solve this question in the case of polynomial time was to consider machines with a limited number of accepting computations. This led to the definition of the classes *UP* (unambiguous polynomial time) and *FewP* [Val76, All86], defined by NP machines with a number of accepting computations bounded by one and a polynomial, respectively. Unambiguous complexity classes play a crucial role in cryptography and for the existence of one-way functions [GS88].

A more general approach to find concepts between determinism and nondeterminism was presented in [BHS90], where the notion of *ambiguity* for space bounded Turing machines was introduced. A Turing machine is $a(n)$-ambiguity bounded if for all inputs up to length n and each configuration A there are at most $a(n)$ computation paths from the initial configuration to A. An $s(n)$ space bounded nondeterministic Turing machine can be deterministically simulated using only $s(n)^2$ space due to the theorem of Savitch. Buntrock, Hemachandra, and Siefkes showed that an unambiguous Turing machine can do this simulation better in certain cases. If an $NSPACE(\log n)$ machine has ambiguity $a(n)$, then it can be simulated unambiguously with $\log(n) \cdot \log(a(n))$ space.

[*]This research was partially supported by the Deutsche Forschungsgemeinschaft, SFB 342, Teilprojekt A4 "Klassifikation und Parallelisierung durch Reduktionsanalyse"

Theorem 1 [BHS90]

1. *For $s(n)$-space computable $a(n)$ with $\log(a(n)) = O(s(n))$ and $s(n) \geq \log n$, (NSPACE)-AMBIGUITY$(s(n), a(n))$ $\subseteq$ UnambAPDA-SPACE-STACKSIZE$(s(n), s(n) \cdot \log(a(n)))$.*

2. *For logspace computable $a(n)$, $a(n)$ polynomially bounded, (NSPACE)-AMBIGUITY$(\log n, a(n)) \subseteq$ UnambAPDA-TISP$(n^{O(1)}, \log n)$.*

This is better than Savitch's simulation if $a(n)$ grows slower than all polynomials. Additionally, the unambiguous simulation uses only polynomial time which is also better than Savitch's method and the additional space that is needed can be arranged as a pushdown store rather than an ordinary Turing tape. Meanwhile, it could also be shown that any language accepted by some polynomially ambiguity bounded log space machine is contained in *LOGDCFL* (see [BJLR90]).

In [BHS90] a new inductive counting method was used to show Theorem 1. In Section 2 techniques of [BHS90] and [LR90] will be combined to improve Theorem 1. We show that nondeterministic auxiliary pushdown automata can be simulated by unambiguous ones which use more space and time. However, in the most important cases the amount of additional space and time used depends only on the ambiguity of the simulated machine. If this ambiguity is bounded by some polynomial, we even need neither additional space nor time for the simulation. For this purpose, we will introduce ambiguity bounded auxiliary pushdown automata and circuits. While in [BHS90] the ambiguity bound applies only to reachable configurations, we will restrict the number of paths *between any two configurations* of auxiliary pushdown automata due to the parallel nature of this model.

Unambiguity seems to be a concept restricted to sequential computation. Recently, however, well known models of parallel computations were recognized to behave unambiguously in some sense. The first candidates for such unambiguous parallel models were CREW-PRAMs, i.e., parallel random access machines with concurrent read and exclusive write access. In [Ryt87] it was shown that CREW-PRAMs can recognize any unambiguous context-free language. The computational power of CREW-PRAMs lies between CRCW-PRAMs and CROW-PRAMs [Sni82, DR86]. For CRCW-PRAMs concurrently writing to the same memory cell is allowed. The class of languages accepted by this type of PRAM in $O(log^k n)$ time using polynomially many processors is exactly AC^k, the class of languages accepted by polynomial size and $O(log^k n)$ depth circuits of unbounded fan-in. AC^1 contains *LOGCFL*, the closure of context-free languages under log space many-one reductions [Ruz80]. On the other hand, CROW-PRAMs are a restriction of CREW-PRAMs, where each memory cell has one owner who is the only one allowed to write into this memory cell. CROW-PRAMs with polynomially many processors recognize in $O(\log n)$ time *LOGDCFL*.

An AuxPDA is called *strongly unambiguous*, if it fulfills the additional restriction that there is at most one computation path between any two configurations. This restriction, which must hold for all possible input words, is called *strong unambiguity.* Note that strong unambiguity also means that there is exactly one accepting configuration, in particular, strongly unambiguous AuxPDA are unambiguous. Speaking of strongly and weakly unambiguous models, unambiguous Turing machines are *weakly unambiguous* and CREW-PRAMs are *strongly unambiguous.* For sequential models of computation weak unambiguity seems to be the natural choice for an unambiguity notion, while it is strong unambiguity for parallel models. For auxiliary pushdown automata it even seems reasonable to consider both notions because an auxiliary pushdown automaton in principle is a sequential model that has also important relations to parallelity (see e.g. [Ruz80, Ven87]). Further evidence of relationship between CREW-PRAMs and unambiguous complexity classes followed from a circuit characterization of CREW-PRAMs. Analogously to CRCW^k = AC^k (see [SV85]), a similar equation CREW^k = $UnambAC^k$ was proved in [Lan90]. For this purpose, semi-unbounded, *unambiguous circuits* had to be defined.

An AC^k-circuit is called *unambiguous* if for all inputs there is no unbounded fan-in OR-gate in the circuit that receives more than one 1 from its predecessors. Similarly, no unbounded fan-in AND-gate is allowed to receive more than one 0. Note that these restrictions need only hold for all *unbounded fan-in gates*, there is no restriction for gates that have only two inputs. Unambiguous circuits of semi-unbounded fan-in were considered, too. This led to the two classes $UnambSAC^k$ and $UnambRAC^k$. The first one consists of all languages that are recognized by $UnambAC^k$-circuits which do not contain any unbounded fan-in AND-gates. The second one is even more restricted: $UnambRAC^k$ contains all languages that are recognized by semi-unbounded fan-in circuits, in which *all* OR-gates receive at most one 1. In contrast to $UnambSAC^k$, in $UnambRAC^k$ circuits also OR-gates of bounded fan-in have to be vulnerable.

In [Lan90] it was shown that $UnambRAC^k$ circuits can be simulated by $c^{\log^k n}$ time bounded unambiguous auxiliary pushdown automata. On the other hand, $UnambRAC^1$ contains $LOGSPACE$. In the subsequent paper [LR90] the fine structure of $UnambRAC^1$ was investigated. $UnambRAC^1$ is exactly the class of languages recognized by polynomially time bounded strongly unambiguous auxiliary pushdown automata. On the other hand, if we claim gates to be vulnerable within an accepting subcircuit, only, such a circuit is called *weakly unambiguous* and the class of languages recognized by these circuits is denoted by $WeakUnambRAC^1$. Another result of [LR90] is that $WeakUnambRAC^1$ is exactly the class of languages recognized by some unambiguous auxiliary pushdown automata.

We suppose all circuit families to be U_{BC}-uniform, that is, a description of the nth circuit can be computed by some $z(n)$ space bounded Turing machine, where $z(n)$ is the size of the circuit (cf. [Ruz81]). In the case of polynomial size circuits this means logspace uniformity. We also assume all circuits to be *leveled*, i.e., informally spoken, each circuit gate of depth d receives its inputs from gates at depth $d-1$. This property is also called synchronous in [BCD$^+$89]. It can be seen (although it is not trivial) that this normal form can be established for all circuits considered in this paper. By now, no nontrivial upper bounds were known for $UnambSAC^k$. In Section 3 we show the inclusion of $UnambSAC^k$ in $UnambAPDA^k$ using an inductive counting technique. This surprising relationship answers an open question of [Lan90], where this inclusion was conjectured to be wrong or at least very hard to obtain. Furthermore, we show $UnambSAC^k$ to be closed under complementation.

For nondeterministic and deterministic, polynomially time bounded AuxPDA it is known that their pushdown height can be bounded by $O(\log^2 n)$ [Ruz80, DR86]. In Section 4 we obtain the same result for unambiguous and strongly unambiguous AuxPDA. Moreover, we consider oblivious AuxPDA, i.e., machines where the movements of all heads do not depend on the input except its length. We show that nondeterministic, unambiguous, and strongly unambiguous AuxPDA can be simulated by such machines which, furthermore, are oblivious. Considering a special kind of obliviousness, we finally obtain AuxPDA characterizations for $WeakUnambRAC^k$ and $UnambRAC^k$, extending a result of [LR90], where only a characterization for $k=1$ was obtained.

2 Simulating Ambiguous AuxPDA by Unambiguous AuxPDA

In some respects this part of the paper offers a fusion of results and ideas of [LR90] and [BHS90]. Roughly speaking, we proceed in the following manner. In Subsection 2.2 we make use of the simulation of unambiguous AuxPDA by unambiguous circuits to be found in [LR90]. Using the same construction (in revised form) we prove a result stating that ambiguity bounded AuxPDA can be simulated by ambiguity bounded circuits. Then, in Subsection 2.3, we give a simulation of ambiguity bounded circuits by unambiguous AuxPDA. By means of these two results we obtain our main theorem, the unambiguous simulation of ambiguity bounded AuxPDA. This simulation only demands a moderate penalty in space and time usage. For the most interesting cases this penalty depends only on the ambiguity bound. Up to polynomial ambiguity there is no penalty at all. A similar result was proved by Buntrock, Hemachandra, and Siefkes who showed a similar result for space bounded classes. The additional space again depends on the ambiguity bound and can be organized as a pushdown store rather than a general Turing tape. However, there is no additional space necessary only in the case of constant ambiguity.

For reasons of clarity, we firstly give the basic definitions needed for this and further sections and, immediatly afterwards, come to out main result, presented in Subsection 2.1.

We deal with auxiliary pushdown automata. AuxPDA are space bounded Turing machines with an additional, unbounded pushdown store ([Coo71], [Ruz80]). As usual, we make the technical assumption that accepting computations always end with an empty stack and an empty working tape, all heads at a fixed position, and there is exactly one final state, i.e., there is exactly one accepting configuration. This normal form can always be achieved for deterministic and nondeterministic auxiliary pushdown automata, but this does not seem to be the case for unambiguous and strongly unambiguous automata. However, acceptance by empty pushdown store seems to be the natural choice. Additionally, we require the machine either to push or pop one symbol in every step. $NAPDA^k$ is the class of languages recognized by nondeterministic AuxPDA using logarithmic working space in time $2^{O(\log^k n)}$. In the usual way, we call an AuxPDA M *unambiguous* if for every input there is at most one accepting computation. M is *strongly unambiguous* if for every pair of configurations of M there is at most one computation path between them. We denote the class of languages recognized by $O(\log n)$ space and $2^{O(\log^k n)}$ time bounded unambiguous (resp. strongly unambiguous) auxiliary pushdown automata by $UnambAPDA^k$ (resp. $StUnambAPDA^k$).

Definition 2 We say AuxPDA M has *ambiguity* $a(n)$ if and only if for all inputs of length n there exist at most $a(n)$ computation paths between two arbitrary configurations of M.

Note that we do not mean surface configurations here and that this restriction must even hold for configurations that are not reachable from the initial configuration.Clearly, an ambiguity bound of one coincides with the notion of strong unambiguity. Here the difference between the definition of ambiguity of [BHS90] and ours should be emphasized. Buntrock, Hemachandra, and Siefkes define ambiguity as the maximum over the number of paths leading to any configuration reachable by the start configuration. In contrast, the above definition restricts ambiguity also for configurations not reachable by the start configuration.

As detailed below, we firstly define the ambiguity of a circuit C by means of the definition of ambiguity of a gate of C. The motivation for the definition arises from the notion of accepting subtrees of a circuit $\tilde{C}$ with root g: The nodes of an accepting subtree are gates of $\tilde{C}$. The root node is g. If g is an OR-gate, then in the accepting subtree g exactly has one child gate h which must have value 1. If g is an AND-gate, then all input gates of g (which must have value 1) are children of g. The rest is done by a straightforward induction. In this context, the ambiguity of any gate g of C simply corresponds to the number of accepting subtrees of subcircuit $\tilde{C}$ of C with output gate g. The ambiguity of C then is the maximum over all ambiguities of gates of C.

Definition 3 Let C be a circuit with input $x \in \{0,1\}^n$.

1. The *ambiguity* $GAmb_C(g,x)$ *of gate* g *in* C *on input* x is defined as follows:
   (a) If g is an input gate of C, then $GAmb_C(g,x)$ is 1 if $g = 1$ and 0, otherwise.
   (b) If $g = AND(g_1, g_2)$, then $GAmb_C(g,x) := GAmb_C(g_1,x) \cdot GAmb_C(g_2,x)$.
   (c) If $g = OR(g_1, \ldots, g_l), l \geq 2$, then $GAmb_C(g,x) := \sum_{i=1}^{l} GAmb_C(g_i,x)$.
2. The *ambiguity* $CAmb(C,x)$ *of circuit* C *on input* x is defined as maximum over the gate ambiguities of all gates in C:
   $CAmb(C,x) := \max_{g \in C} GAmb_C(g,x)$.

Again it is clear that semi-unbounded fan-in circuits of depth $O(\log^k n)$ and ambiguity one recognize exactly all languages in $UnambRAC^k$. The equality of $StUnambAPDA^1$ and $UnambRAC^1$, which was shown in [LR90], means in the terminology of ambiguity bounded classes that polynomially time bounded auxiliary pushdown automata of ambiguity one recognize the same class of languages as $O(\log n)$ depth semi-unbounded circuits with ambiguity one.

The subsequent proposition is evident.

Proposition 4 *A circuit C accepts on input x iff $CAmb(C, x) \geq 1$.*

Now we are able to define the ambiguity bounded AuxPDA and circuit classes dealt with in this section.

Definition 5 1. A language L is in $NAPDA\text{-}TISP\text{-}AMBIGUITY(t(n), s(n), a(n))$ if L is accepted by a nondeterministic $t(n)$ time-bounded and $s(n)$ space-bounded auxiliary pushdown-automaton which has ambiguity $a(n)$.

2. A language L is in $Semiunbounded\text{-}SZDP\text{-}AMBIGUITY(z(n), d(n), a(n))$ if L is accepted by a $O(d(n))$ depth-, $O(z(n))$ size-, and $a(n)$ ambiguity-bounded circuit which consists of AND-gates of bounded fan-in and OR-gates of arbitrary fan-in.

2.1 The Main Theorem and Further Results

In the beginning we state our main result for the unambiguous simulation of ambiguity bounded AuxPDA. Corollary 7, a special case of this result, is particularly interesting.

Theorem 6 *Let $m(n) = \max(a(n), 2^{s(n)}, t(n))$. Then*
$NAPDA\text{-}TISP\text{-}AMBIGUITY(t(n), s(n), a(n))$
$\subseteq UnambAPDA\text{-}TISP(m(n)^{O(1)}, \log(m(n)))$.

The proof of Theorem 6 is a combination of Theorem 9 and 14 of the following two subsections. Corollary 7 reveals the strength of our result. A polynomially ambiguity bounded AuxPDA with logarithmic working-tape and polynomial running-time can be simulated by an unambiguous AuxPDA without time and space penalty.

Corollary 7 $NAPDA\text{-}TISP\text{-}AMBIGUITY(n^{O(1)}, \log n, n^{O(1)})$
$\subseteq UnambAPDA\text{-}TISP(n^{O(1)}, \log n)$.

Buntrock, Hemachandra, and Siefkes showed that logspace and polynomially ambiguity bounded Turing machines can be simulated by unambiguous AuxPDA in time $n^{O(1)}$ and space $\log n$. Our results yields that these AuxPDA can even simulate logspace and polynomially ambiguity bounded AuxPDA, i.e., in contrast to [BHS90], we do not have to pay any penalty. Roughly speaking, our result says that we can simulate a more powerful computational model than [BHS90] by the same machines as they use. Note that in the unambiguous simulation of [BHS90] the space penalty for ambiguity $a(n)$ appears as a multiplicative factor $\log(a(n))$. In contrast to them, in our simulation we only have to pay an additive space penality of $\log(a(n))$. Note that this is the reason why in Corollary 7 both AuxPDA have same space complexity.

Eventually, the simulation by unambiguous AuxPDA also is possible if the simulated, ambiguity bounded AuxPDA have got the MOD_q acceptance mechanism.

Theorem 8 *Let $a(n) = 2^{O(s(n))}$ and $m(n) = \max(2^{s(n)}, t(n))$.*
Then $MOD_qAPDA\text{-}TISP\text{-}AMBIGUITY(t(n), s(n), a(n))$
$\subseteq UnambAPDA\text{-}TISP(m(n)^{O(1)}, \log(m(n)))$. *

*$MOD_qAPDA\text{-}TISP\text{-}AMBIGUITY$ is the $NAPDA\text{-}TISP\text{-}AMBIGUITY$ complexity class using the acceptance mechanism of the polynomial time class MOD_qP [BGH90].

Again the main task in the proof of Theorem 8 is to combine Theorem 9 and Theorem 14. But now, we must additionally change the mode of acceptance. Remember that for MOD_q-classes it only matters to count the number of accepting computation paths. A careful inspection especially of the proof of Theorem 6 will reveal that it is also possible to ascertain correctly this number.

In the following two subsections we will give the proof of the above results. Herein, Subsection 2.2 will not introduce any new proof techniques, but will show a new result for an already known circuit construction. Subsection 2.3 will present our new technique, namely inductive counting on leveled circuits and, finally the proofs of Theorem 6 and 8.

2.2 Simulating Ambiguous AuxPDA by Ambiguous Circuits

Here we are going to give the simulation of ambiguity bounded AuxPDA by ambiguity bounded, semi-unbounded circuits. Again, for reasons of clarity, we firstly state the main result of this subsection.

Theorem 9 *Let* $m(n) = \max(t(n), 2^{s(n)})$*. Then*
NAPDA-TISP-AMBIGUITY$(t(n), s(n), a(n))$
$\subseteq$ *Semiunbounded-SZDP-AMBIGUITY*$(m(n)^{O(1)}, \log(t(n)), a^2(n))$.

This subsection is organized as follows. First, we give the basic circuit construction of [LR90] (in revised form) to simulate AuxPDA by circuits. For this purpose, we have to take a closer look on computation paths of AuxPDA. Second, we show that the constructed circuit exactly fulfills the properties required in Theorem 9.

The following definitions and propositions arise from [LR90]. In the beginning, we define some basic concepts with respect to auxiliary pushdown automata. By capital letters we denote *surface configurations*. Surface configurations, we consider, contain the topmost symbol of the pushdown store, the actual state, and the contents of the auxiliary tape, as well as the positions of the input and working heads. Our purpose now is to identify certain computation paths of an auxiliary pushdown automaton M with some sort of binary trees. (These trees are the essential tool for constructing a circuit simulating M.) For this reason, we introduce the notion of *nodes*. A triple (A, B, i) is called a node if A and B are surface configurations and i is some number bounded by the running-time of M. How will finally turn out, we are mainly interested in *realizable* nodes, i.e., nodes for which there exists a computation from A to B in $2(i-1)$ steps and where the level of the pushdown is the same for A and B and does not go below this level during the computation. Note that there is an accepting computation of M from the start configuration S to the end configuration F (which both are uniquely determined) iff there exists an i_0 such that (S, F, i_0) is realizable.

The rough idea of binary trees is to recursively divide a computation path between configurations fulfilling the above mentioned properties in two uniquely determined paths. For this purpose, we introduce a relation '$\vdash$' between nodes and, in this way, between computation paths. Let $x = (A, B, i)$, $y = (C, D, j)$, and $z = (E, B, k)$. Then we write $y, z \vdash x$ and $z, y \vdash x$ iff

1. The level of the pushdown is equal for A, E, and B.
2. There exists a computation from A to C in one step, pushing a onto the pushdown store during this step.
3. There exists a computation from D to E in one step, popping a from the pushdown store.
4. $j + k = i$, where $j, k \geq 1$.

Now we are able to define the binary trees. For each inner node $x = (A, B, i)$, $i > 1$, both children y and z are determined by $y, z \vdash x$. Leaves of a binary tree are of shape $(A, B, 1)$. In this context, the meaning of the parameter i in the definition of nodes becomes clear: i denotes the number of leaves of a binary tree with root (A, B, i). Some simple considerations show that (A, B, i) is realizable iff there exists a binary tree with root (A, B, i), in which all the leaves are of shape $(C, C, 1)$. We aim at finding out whether there exists an i_0 such that (S, F, i_0) is realizable. For this target, the binary trees with root (S, F, i) serve in a straightforward manner to recursively compute the realizability of (S, F, i). (A node is realizable iff both its children are realizable or it is a leaf of shape $(A, A, 1)$.)

To gain a circuit with small depth, it is necessary to make a division of the binary trees into certain subtrees. To this, we have to consider *pairs of nodes* $(x, y) = ((A, B, i), (C, D, j))$, where $i \geq j$ holds. A pair of nodes (x, y) is called *realizable* iff there is a binary tree with root x, one leaf is y, and all other leaves are of shape $(E, E, 1)$. That is, the original binary tree is transformed into a binary tree where the subtree with root y is simply replaced by the node y (which now is considered as a leaf). We say the tree has gap y. Note that this tree has $i - j + 1$ leaves. Then, roughly speaking, we split a binary tree with root x in the binary tree with root x and gap y and in the tree with root y (more precisely, in the two trees of the children of y), where both arising subtrees have approximately the same number of leaves.

Now the detailed construction follows: Gates labeled $\langle x\rangle$ will compute whether x is realizable. If $x = (A, B, 1)$, then $\langle x \rangle \equiv 1$ iff $A = B$ and $\langle x \rangle \equiv 0$, otherwise. If $x = (A, B, i)$, $i > 1$, then $\langle x\rangle$ is defined as

$$\langle x\rangle \quad \equiv \quad \exists_{y,y_1,y_2}\langle x, y\rangle \wedge \langle y_1\rangle \wedge \langle y_2\rangle \wedge \langle y_1, y_2 \vdash y\rangle,$$

where y_1, y_2, and y are nodes such that $y_1 \ll y_2$ holds, where $\ll$ is a logspace-computable total order. Herein, the last gate computes whether the relation $y_1, y_2 \vdash y$ holds. Observe that this only depends on at most input bits and the transition relation of the simulated AuxPDA. Obviously, this can easily be managed by the uniformity machine of the circuit. Assume that $y = (C, D, j)$, $y_1 = (E, G, j_1)$, and $y_2 = (H, D, j_2)$. Then y, y_1, and y_2 are additionally restricted by $j_1, j_2 \leq \lceil \frac{i}{2} \rceil < j$.

The gates $\langle x, y\rangle$ check whether (x, y) is realizable and are defined in an analogous way to the $\langle x\rangle$-gates. If $(x, y) = (x, x)$, then $\langle x, y\rangle \equiv 1$. Otherwise, they are defined as

$$\langle x, y\rangle \quad \equiv \quad \exists_{y_1,y_2,y_3}\langle x, y_1\rangle \wedge \langle y_2, y\rangle \wedge \langle y_3\rangle \wedge \langle y_2, y_3 \vdash y_1\rangle.$$

Herein the gates $\langle y_2, y_3 \vdash y_1\rangle$ are analogously defined to the above case. Let $x = (A, B, i)$, $y = (C, D, j)$, $y_1 = (E, G, j_1)$, $y_2 = (H, I, j_2)$, and $y_3 = (K, G, j_3)$. Then y_1, y_2, and y_3 are further restricted by $j_2 - j + 1 \leq \lceil \frac{i-j+1}{2} \rceil < j_1 - j + 1$.

Lemma 10 *For each binary tree belonging to the realizable node x resp. pair of nodes (x, y) the realizable (pairs of) nodes (x, y), y_1, y_2 resp. (x, y_1), (y_2, y), and y_3 are uniquely determined by the above construction.*

Proof. Lemma 10 is an immediate consequence of Lemma 3 and Lemma 4 of [LR90]. □

This completes the construction of the simulating circuit.

In order to prove Theorem 9 we have to do some technical groundwork, presented in the following three lemmata. Herein, the first two lemmata relate the number of computation paths to the number of binary trees. Eventually, the third lemma provides for the changeover from number of different binary trees for some node x resp. pairs of nodes (x, y) to the ambiguity of the corresponding gate in the constructed circuit.

Subsequently, it is assumed that both the children of a node in a binary tree are always ordered according to a logspace-computable ordering of their labelings. Note that for the proof of Theorem 9 we don not need Lemma 11 and Lemma 13 in its full sharpness, but they will be needed in such a form for proving Theorem 8.

Lemma 11 *If there exist exactly $k > 0$ different computation paths of length $l = 2m$ between two configurations A, B with same stack height of AuxPDA M, then there exist exactly k different binary trees with root $x = (A, B, m+1)$.*

Proof. (idea) The proof is done by induction on the length l of computation paths. Observe that l always is an even number because of the assumptions of the introductionary section and the same pushdown store height of configurations A and B.

There are two things to be shown: First, each computation path has a unique corresponding tree and, second, for two different computation paths the corresponding binary trees also differ. □

Lemma 12 *If there exist at most $k > 0$ computation paths between two arbitrary configurations of AuxPDA M, then there exist at most k^2 different binary trees with root x and gap y belonging to some (realizable) pair of nodes (x, y).*

Proof. (idea) The central idea of proof is as follows. A binary tree belonging to the pair of nodes (x, y) represents a computation with gap. Let $(x, y) = ((A, B, i), (C, D, j))$. Then the above means that the binary tree stands for a computation path from A to C, then there is a gap between C and D, and finally for a computation path from D to B. But this means that trees belonging to (x, y) represent at most k^2 different computation paths (with gap). Eventually, we can conclude in an analogous way to Lemma 11 that there are at most k^2 binary trees belonging to (x, y). □

Lemma 13 *1. Let x be a realizable node with m different binary trees corresponding to x Then gate $\langle x \rangle$ exactly has ambiguity m.*

2. Let (x, y) be a realizable pair of nodes with n different binary trees corresponding to (x, y). Then gate $\langle x, y \rangle$ exactly has ambiguity n.

Proof. Let $x = (A, B, i)$ and $y = (C, D, j)$ where $i \leq j$. The proof is done by induction on i for nodes x and by induction on $i - j + 1$ for pairs of nodes (x, y), i.e., the number of leaves of the binary tree.

1. $i = 1$: This means $x = (A, A, 1)$ (because x is realizable) and $m = 1$. Then $\langle x \rangle \equiv 1$ and, therefore, the claim follows by the definition of circuit ambiguity.
   $i = 2$: It is obvious from the definition of the binary trees that there are as many binary trees (namely m) as different E exist, in which there is a computation in two steps from A via E to B. They all have the following structure: The root is labeled $x = (A, B, 2)$ and both leaves and, at the same time, children of x are labeled $y = (E, E, 1,)$ resp. $z = (B, B, 1)$, that is $y, z \vdash x$. Thus, the gate labeled $\langle x \rangle$ is an *OR*-gate over all inputs of shape $\langle x, x \rangle \wedge \langle y \rangle \wedge \langle z \rangle$. Note that all the three gates in the preceding conjunction evaluate to 1. Furthermore, x and z are uniquely determined and the number of 'different' $\langle y \rangle$-gates depends on the number of different E. Seeing that, it is evident that the circuit with output gate x exactly has ambiguity m.

2. $i - j + 1 = 1$: This means $(x, y) = (x, x)$ and it is analogous to $i = 1$ in the upper case, that is, $n = 1$ must hold.
   $i - j + 1 = 2$: As can be easily seen from the definitions of binary trees resp. the '$\vdash$'-relation on nodes, there are only two possibilities for gap y, namely $y = (C, D, i-1)$ $(D \neq B)$ and $y = (C, B, i-1)$. (These y stem from $(C, D, i-1), (B, B, 1) \vdash x$ and $(E, E, 1), (C, B, i-1) \vdash x$. Because of the definition of '$\vdash$' $y = (A, D, i-1)$ is impossible.) First, assume $y = (C, D, i-1)$. Then $z = (B, B, 1)$ and, because everything in this is uniquely fixed, $n = 1$ must hold. Second, assume $y = (C, B, i-1)$. Then $z = (E, E, 1)$ holds and the number n of different

binary trees belonging to (x, y) exactly corresponds to the number of different z (i.e., the number of different E). Because of the input gates of the gate labeled $\langle x, y\rangle$ (observe $y_1 = x$, $y_2 = y$, $y_3 = z$, and $\langle x, x\rangle \equiv \langle y, y\rangle \equiv 1$) we can conclude in an analogous way like above that gate $\langle x, y\rangle$ has ambiguity n in both cases for y mentioned above.

Induction step:

1. $i > 2$: At first, we subdivide the m different binary trees in equivalence classes. By Lemma 10, y, y_1, and y_2 from the construction of the circuit with output gate $\langle x\rangle$ are uniquely determined for each binary tree. Two binary trees are said to be equivalent iff they have exactly the same y, y_1, and y_2. Let r denote the number of equivalence classes. Then $m = \sum_{i=1}^{r} |i\text{th equivalence class}| =: \sum_{i=1}^{r} e_i$.

   From the construction of the circuit we have that there is always one AND-gate of value 1 for each equivalence class; that is, r inputs of the OR-gate for $\langle x\rangle$ have value 1.

   In the following, we make use of the division of a binary tree into three subtrees, namely one with root x and gap y (i.e., the tree belonging to (x, y)), one with root y_1, and one with root y_2. Let $Bintrees(x)$ resp. $Bintrees(x, y)$ denote the number of different binary trees with root x resp. root x and gap y. For each equivalence class the number of different binary trees only depends on the number of trees belonging to (x, y), y_1, and y_2. Now for this subtrees the induction hypothesis applies. For the r different equivalence classes we denote the r different values for (x, y), y_1, and y_2 by $(x, y)^k$, y_1^k, and y_2^k, respectively. So we have

$$\begin{aligned} m &= \textstyle\sum_{k=1}^{r} e_k \\ &= \textstyle\sum_{k=1}^{r} (Bintrees((x, y)^k) \cdot Bintrees(y_1^k) \cdot Bintrees(y_2^k)) \\ &= \textstyle\sum_{k=1}^{r} (GAmb(\langle x, y\rangle^k) \cdot GAmb(\langle y_1\rangle^k) \cdot GAmb(\langle y_2\rangle^k)) \\ &= \textstyle\sum_{k=1}^{r} GAmb(AND(\langle x, y\rangle^k, \langle y_1\rangle^k, \langle y_2\rangle^k)) \\ &= GAmb(\langle x\rangle), \end{aligned}$$

   i.e., the ambiguity of the circuit with output gate $\langle x\rangle$.

2. $i - j + 1 > 2$: The n binary trees are subdivided into equivalence classes in an analogous way like above, but now according to the values of y_1, y_2, and y_3. The rest of the proof is done in an equal manner as in 1.

□

Finally, we unify the preceding three lemmata into the proof of Theorem 9.

Proof. (Theorem 9) As a result of Lemma 11 and Lemma 12, for the AuxPDA with ambiguity $a(n)$ there exist at most $a^2(n)$ different binary trees corresponding to node x resp. pair of nodes (x, y) each.

According to Lemma 13 the simulating circuit of the given construction has at most ambiguity $a^2(n)$, because due to the above the ambiguity of each gate $\langle x\rangle$ resp. $\langle x, y\rangle$ is bounded by $a^2(n)$.

The depth and size properties follow from the proof in [LR90] because we use essentially the same circuit construction. □

2.3 Simulating Ambiguous Circuits by Unambiguous AuxPDA

At this point, we come to a main proof technique of this paper — inductive counting on leveled circuits. With help of this technique we are able to give a simulation of ambiguity bounded circuits by unambiguous AuxPDA. In this way, the last precondition for simulating ambiguity bounded AuxPDA by unambiguous ones is created.

Theorem 14 *Let $m(n) = \max(a(n), 2^{d(n)}, z(n))$. Then*
Semiunbounded-SZDP-AMBIGUITY$(z(n), d(n), a(n))$
$\subseteq$ UnambAPDA-TISP$(m(n)^{O(1)}, \log(z(n)))$.

By means of the technique of Theorem 14 we naturally get the corresponding result for the complement of the above ambiguity bounded complexity classes.

Corollary 15 *Let $a(n) = O(z(n))$ and $m(n) = \max(2^{d(n)}, z(n))$. Then*
Co-Semiunbounded-SZDP-AMBIGUITY$(z(n), d(n), a(n))$
$\subseteq$ UnambAPDA-TISP$(m(n)^{O(1)}, \log(z(n)))$.

Proof. (sketch) Let C be the given, semi-unbounded circuit. First of all, it should be mentioned that we assume C to be leveled. This is admissible, because (inter alia) for semi-unbounded circuits it is possible to construct an equivalent, leveled one, which has the same depth and the size is only increased by some polynomial. Observe that the construction of leveled circuits mentioned in the introductionary section preserves the ambiguity of the circuit (and, in particular, the ambiguity of the output gate) obtained from Theorem 9.

The main tool now is inductive counting on leveled circuits. For each layer i of C we will count the sum LA_i over the ambiguities of all gates in layer i. In this way we will find out the ambiguity of the output gate g of C and, therefore, whether C accepts or rejects input x. Clearly, C accepts iff g has ambiguity greater than 0.

Subsequently, we assume to have a subroutine which for any gate g of C verifies that g has an ambiguity not less than a given value. Moreover, if the given value and the actual ambiguity of g coincide, then the verification is done unambiguously. The subsequent Lemma 16 will show the existence of such an subroutine with suitable space and time bounds.

For the layer of the input gates of C we already know the sum LA_1. It equals n according to the ambiguity definition and the fact that all inputs of C are given in negated and nonnegated form. Now suppose that we already know LA_{i-1}. To determine the ambiguity sum LA_i of layer i, for each gate h of layer i the simulating AuxPDA M carries out the following: For each gate f of layer $i-1$, M guesses its ambiguity value *amb* and verifies by means of the algorithm of Lemma 16 that f actually has at least ambiguity *amb*. Furthermore, a counter U for the ambiguity sum of layer $i-1$ is increased by *amb*. If f is an input gate to h, then *amb* will serve to compute the ambiguity of h in a straightforward way. Finally, if we went through all the gates of layer $i-1$, M checks whether U is equal to LA_{i-1}. If not, M rejects because in this case we must have guessed a too low ambiguity value for at least one gate in layer $i-1$. Such we may ascertain the ambiguity value of each gate h of layer i and, therefore, LA_i. Subsequently, we will refer to the above algorithm as *SUM_UP*.

The following observation implies the unambiguity of *SUM_UP*. The AuxPDA of Lemma 16 checks in *SUM_UP* whether the ambiguity of some gate was not guessed too big. But it does not detect whether the guessed value for the ambiguity of the gate is too small. Now the auxiliary variable U comes into play. If $U \neq LA_{i-1}$, then *SUM_UP* aborts. But if $U = LA_{i-1}$ and at least one ambiguity value for some gate in layer $i-1$ was guessed too small, then there must have been some gate of layer $i-1$ where the ambiguity value was guessed too big and, therefore, the AuxPDA of Lemma 16 would have rejected.

From this and the unambiguity of the AuxPDA of Lemma 16 for correctly guessed ambiguity values, we conclude the unambiguity of *SUM_UP*. This works even if the ambiguity bound $a(n)$ is not space constructible, since, in this case, we can make use of the following fact. For layer $i \geq 1$ we know the ambiguity sum LA_{i-1} of layer $i-1$ and LA_{i-1}^2 is an upper bound for the ambiguity of each gate in layer i[†].

[†]Buntrock, Hemachandra, and Siefkes required the ambiguity of space bounded Turing machines to be space constructible in [BHS90]. This restriction can be dropped by essentially the same technique.

To proof Corollary 15 simply change the mode of acceptance in the proof of Theorem 14: Now M accepts iff the output gate of the simulated circuit C equals 0. □

As can be seen above (and in later theorems), it is of great importance to have an appropriate unit of measure for gates. That is, on the one hand, it should have as small values as possible (because of 'space reasons') and, on the other hand, it should enable an verification algorithm which works unambiguously for correctly guessed values. It will turn out that a decisive point for our proof technique is to find such suitable units of measure. In Lemma 16 it is shown that in the case of ambiguity bounded AuxPDA ambiguity is such an appropriate unit of measure.

Lemma 16 *Let C be a semi-unbounded circuit of depth $d(n)$, size $z(n)$, and ambiguity $a(n) = O(z(n))$ and let $m(n) = \max(2^{d(n)}, z(n))$. Then there exists an $m(n)^{O(1)}$ time and $\log(z(n))$ space bounded auxiliary pushdown automaton M which accepts the language $L = \{ (x, amb) \mid x \in \{0,1\}^n$ is an input to circuit C with output gate g, $amb = O(a(n))$, and $GAmb_C(g,x) \geq amb \}$.*

If even $GAmb_C(g,x) = amb$ holds, then M works unambiguously.

Proof. First, we show how M works. If $amb = 0$, then M accepts at once. Now let $amb > 0$. If g is an input gate of C, then M accepts iff g has value 1. If $g = AND(g_1, g_2)$ holds, then M guesses two numbers a_1 and a_2, checks whether $a_1 \cdot a_2 = amb$, and, finally, makes two recursive calls to verify that g_1 resp. g_2 have ambiguity not less than a_1 resp. a_2. If $g = OR(g_1, \ldots, g_k)$, then M guesses k numbers $a_1, \ldots, a_k$, checks whether $\sum_{i=1}^{k} a_i = amb$, and makes in the case of need recursive calls to verify that the g_i $(1 \leq i \leq k)$ have ambiguity greater than or equal to a_i.

The correctness and the space bound of the above algorithm are straightforward. To prove the time bound we need the following lemma.

Lemma 17 *The number of recursive calls of M for a fixed circuit C of depth d and a given number amb is bounded by $O(2^d \cdot amb)$.*

Proof. (idea) The proof is done per induction on the depth d of the circuit. □

Finally, we show unambiguity for M in the case of $GAmb_C(g,x) = amb$. We only give a idea of proof, which is formally done by induction on the circuit depth d.

For example, let $g = AND(g_1, g_2)$. W.l.o.g. assume that $a_1 \cdot a_2 = amb$ holds and we have guessed a too high ambiguity value for g_2, that is, $a_2 > GAmb_C(g_2, x)$ holds. But then the recursive call to verify that g_2 has ambiguity not less than a_2 will fail and M rejects by definition, because it will eventually reach an input gate which has ambiguity less than its guessed value. All the other cases are handled in a similar way. □

Eventually, it remains to give the proof of Theorem 8.

Proof. (of Theorem 8) In proof of Theorem 9, i.e., by means of Lemma 11 and Lemma 13, we have that the number of accepting computation paths exactly transfers to the ambiguity of the output gate; that means that the ambiguity of the output gate (which corresponds to some realizable node $x = (S, F, i)$ (where S is the starting configuration and F the unique accepting configuration) coincides with the number of accepting paths of the simulated AuxPDA.

Finally, with the help of Theorem 14 the ambiguity of the output gate is determined and such it suffices to modify the accepting condition according to the MOD_q-mechanism. □

3 Strongly Unambiguous Circuits and AuxPDA

In Section 3 and 4 we will deal with semi-unbounded circuit classes and, in particular, their unambiguous restrictions. In the beginning, we give the definitions of theses classes. The class SAC^k consists of all languages recognizes by polynomial size and $O(\log^k n)$ depth bounded circuits, which are composed of AND and OR gates of bounded fan-in and, moreover, OR gates of unbounded fan-in [Ven87]. Starting from this notion, Lange [Lan90] developed the two strongly unambiguous classes $UnambSAC^k$ and $UnambRAC^k$. For this purpose, he introduced the notion of vulnerable OR gates. For vulnerable OR gates multiple 1-input is forbidden. In this way $UnambSAC^k$ is defined as the class of all languages accepted by SAC^k circuits, where each unbounded OR gate may be replaced by a vulnerable one. Now, $UnambRAC^k$ is the subclass of SAC^k circuits, which consists of all languages accepted by circuits where even all OR-gates can be replaced by vulnerable ones.

This section is organized as follows. In the first part, we present some complementation results for semi-unbounded circuits. In the second part, we give an simulation for $UnambSAC$-circuits by unambiguous AuxPDA.

3.1 Inductive Counting and Semi-unbounded Circuits

In the beginning, we present an application of our proof technique for an already known result. Borodin et al. [BCD+89] proved the closure under complementation of the circuit class SAC^k by directly constructing a 'complementary' circuit. In contrast to them, we make use of Venkateswaran's [Ven87] AuxPDA characterization of SAC^k to recognize the complement of an SAC^k-circuit by an AuxPDA. For this purpose, it is important to assume the simulated circuits to be leveled. The idea behind is that, on the one hand, circuits possess a regular and simple structure (especially if they are leveled), consist of only polynomially many gates (in opposite to the superpolynomial number of AuxPDA configurations), and, on the other hand, counting and recursive calls can easily be done on AuxPDA. By this, a variation of our inductive counting technique of Subsection 2.3 well applies.

Proposition 18 [Ven87] $SAC^k = NAPDA\text{-}TISP(c^{\log^k n}, \log n)$.

Proposition 18 states a result of Venkateswaran, which will be the 'verification tool' of Theorem 19.

Theorem 19 [BCD+89] $SAC^k = Co\text{-}SAC^k$.

Proof.(sketch) The proof is done in the following manner. For a given logspace-uniform circuit C we describe an auxiliary pushdown automaton M which accepts on input w iff C rejects. In this way, Theorem 19 follows by application of Proposition 18.

We use inductive counting technique to determine the number of gates which evaluate to 1 for each layer i of the given, leveled circuit C. Note that the number of layers of C coincides with its depth. We will ascertain the value of the output gate, which lies in the highest layer of C. M will accept iff the output gate evaluates to 0.

In layer 1, i.e., the layer of the input gates of C, the number $G1_1$ of gates with value 1 (1-gates for short) coincides by definition of SAC^k (each circuit is provided with the input word w and its negations) with n. Now suppose that we already know the number $G1_{i-1}$ of 1-gates in layer $i-1$. Here we show how to find out $G1_i$, i.e., the number of 1-gates in layer i. For each gate g in layer i its value is ascertained by considering all the gates in layer $i-1$ each time. In the course of this, M guesses the values of each gate h in layer $i-1$. If M guessed value 1 then it makes use of Venkateswaran's algorithm to verify this. If the verification succeeds then M additionally increases a counter U for the number of 1-gates in layer $i-1$ and checks, whether h is an input to g. If so, then the value for h will serve in a straightforward manner to compute the value of g (and, in this way, the value of $G1_i$). Finally, after M went through all gates in layer $i-1$ (what was done only to find out the value of one gate g of layer i), it checks whether the above counter U is equal to $G1_{i-1}$. If not (that is, U is less than $G1_{i-1}$), this means that M didn't make out all the 1-gates of layer $i-1$ and, therefore, M rejects. Otherwise, i.e., if $U = G1_{i-1}$, the computation proceeds by determining the value of the next gate in layer i in the same way as above. □

Above, we gave a new proof for the closure under complementation of SAC^k. Hereafter, we show one advantage of the proof of [BCD$^+$89], that is, the fact that it also serves to prove the closure under complement of $UnambSAC^k$ (even with exactly the same construction).

Theorem 20 *$UnambSAC^k = Co\text{-}UnambSAC^k$.*

Proof. As mentioned above, we exactly use the construction of [BCD$^+$89]. Therefore, we do not go into the details of the generation of the 'complementing' circuit of Borodin et al. It is only verified that the resulting circuit again is an $UnambSAC^k$-circuit. (The reader who wants to follow our explanations should have [BCD$^+$89] at hand and especially regard their Figure 2.)

Let C be some $UnambSAC^k$-circuit. Then the constructed 'complementing' circuit $\tilde{C}$ of [BCD$^+$89] mainly consists of two parts. First, C is transformed into some normal form (leveled, fixed width, strictly alternating) yielding circuit $\hat{C}$. $\hat{C}$ is easily seen to be again an $UnambSAC^k$-circuit. The second part is responsible for complementation. The only thing to show is that all unbounded OR-gates used in this part are vulnerable. But the inputs of these unbounded OR-gates are AND-gates, which themselves have so called '$COUNT$'-gates as inputs. There is always exactly one '$COUNT$'-gate evaluating to 1 and, because of this, at most one of the above AND-gates may have value 1. Observe that the '$THRESHOLD$'-gates additionally used in this part, may actually be built up with 'AKS' sorting networks [AKS83], which are NC^1-circuits and such clearly $UnambSAC^1$-circuits. Herein one should note that the AKS sorting networks do not need the negations of the bits of the given input word. By this means, we get a circuit $\tilde{C}$ only using vulnerable OR-gates, i.e., an $UnambSAC^k$-circuit $\tilde{C}$ recognizing the complement of the language of C. □

At this point, one should realize that this result could not be obtained by our proof technique, because we have no AuxPDA characterization of $UnambSAC^k$. But even if we had one, our technique probably wouldn't work for reasons discussed after Theorem 22.

Next, we are going to show the inclusion of the complement of strongly unambiguous AuxPDA classes in AuxPDA classes, where, informally spoken, the strong unambiguity is restricted to configurations reachable from the start configuration.

$ReachUnambAPDA_{pt}(\log n)$ is the class of languages recognized by logarithmically space and polynomially time bounded AuxPDA which satisfy for any input x and any configuration A that there is at most one path from the start configuration to A.

The proof of the above announced can be done by using the algorithm of Theorem 19 with two modifications. One of these concerns the circuit characterization of $StUnambAPDA_{pt}(\log n)$.

Proposition 21 [LR90] *$UnambRAC^1 = StUnambAPDA_{pt}(\log n)$.*

Proposition 21 plays the same role in the subsequent proof as Proposition 18 did in the proof of Theorem 19.

Theorem 22 *$Co\text{-}StUnambAPDA_{pt}(\log n) \subseteq ReachUnambAPDA_{pt}(\log n)$.*

Proof. (sketch) Like in the proof of Theorem 19 the number of 1-gates is counted for each circuit layer and the simulating AuxPDA M will use nearly the same algorithm. There are only two modifications: First, after nondeterministically guessing the value v of a gate, v is pushed on the store. Second, a guessed 1 isn't verified any longer with the help of the algorithm of Venkateswaran, but by means of the strongly unambiguous algorithm of Proposition 21. That is all the new of the construction.

So it remains to show the correctness of the above construction. First of all, observe that the pushdown store is only needed for the verification of 1-gates. Therefore, the pushed values v will not be popped until the accepting state is reached. (Remember that all our AuxPDA accept by empty pushdown store.) Because the contents of the pushdown store is part of the configuration, we will guarantee in such a way the unique reachability for the configurations of M.

The verification of guessed gate values is done strongly unambiguous according to Proposition 21. Moreover, because we can make use of the pushdown store to record the computation path, the computation is strongly unambiguous until M begins to empty the store (in the accepting case). In this way, it is clear that all computations starting in the initial configuration of M are strongly unambiguous (that is, in particular, the (accepting) end configuration of M is reached by exactly one path from the start configuration.

If M rejects then the protocol of the guessed gate values on the pushdown store guarantees the strong unambiguity (for reachable configurations). It is important to remark that none of these guessed values is popped unless M accepts. □

Perhaps one is tempted to assume that the above simulation even yields a strongly unambiguous AuxPDA and, with that, the closure under complementation of the class $StUnambAPDA_{pt}(\log n)$.

But, to this, let us regard the following situation. Suppose that the simulating AuxPDA M is in a configuration which assumes a too low value to be known for the sum of the values of layer $i-1$ and is going to determine this sum for layer i. Some simple considerations show that there now may be different ways to reach the accepting configuration (what stands in contradiction to strong unambiguity). To be brief, the proper reason for the failure is the case when we start in a configuration of M in which the normally by inductive counting ascertained number is pretended wrongly.

3.2 Simulating UnambSAC-Circuits by Unambiguous AuxPDA

To do an unambiguous simulation of $UnambSAC^1$-circuits by AuxPDA, we make use of the same technique as in Theorem 14 and Theorem 19. In particular, we are going to employ inductive counting over an appropriate unit of measurement for the gates of a leveled circuit. This measure again has to fulfill two important restrictions. On the one hand, it has to be polynomially bounded (so that it can be counted in logarithmic space) and, on the other hand, it must allow the unambiguous verification of correctly guessed values for this measure for any gate of the simulated circuit. In the simulation of ambiguity bounded circuits ambiguity was used as a measure. Here, a new measure is introduced (namely the saturation of a gate) because the ambiguity measure is not polynomially bounded for $UnambSAC^1$-circuits.

A gate is considered as saturated if it is an input gate with value 1 or if it is an OR-gate with exactly two 1-inputs. The following definition generalizes this concept by making the saturation dependent on the saturation of its inputs.

Definition 23 Let C be an $UnambSAC$-circuit and g be any gate in C. Then the *saturation* of g is defined as follows:

1. If g is an input gate of C, then it has saturation 1 if $g = 1$ and 0, otherwise.
2. If $g = AND(g_1, g_2)$, then its saturation is the sum of the saturations of g_1 and g_2 if both these saturations are greater than 0 and the saturation of g is 0, otherwise.
3. If g is an OR-gate (of arbitrary fan-in) then we have to consider two cases. If g has at most one input with saturation greater than 0 (i.e., g has at most one input which evaluates to 1), then the saturation of g is defined as the saturation of this input gate. Otherwise, if g two 1-inputs, then g is a bounded OR-gate and the saturation of g is the sum over the saturations of all the inputs of g plus 1 (because g itself is a saturated OR-gate).

Note that the output gate of circuit C on input x has saturation greater than 0 iff C accepts x. Obviously, the saturation is polynomially bounded for all the gates of $UnambSAC^1$-circuits.

Now we present our 'verification lemma' which corresponds to Lemma 16 of the preceding section. Just as there, it is not necessary that the verification of a guessed value less than the actual value of the considered gate is unambiguous. It suffices that the verification is unambiguous if the guessed and the actual value coincide.

Lemma 24 *Let g be the output gate of an $UnambSAC^1$-circuit C and k be some polynomially bounded number. Then there exists a polynomially time and logarithmically space bounded AuxPDA M which accepts the language $L = \{\, (x,k) \mid x \in \{0,1\}^n$ is an input to C and the saturation of g is not less than $k\,\}$.*

If C accepts x and the saturation of g exactly is k then M works unambiguously on input (x,k).

Proof. First, we show how M works. If $k = 0$, then M accepts at once. Now let $k > 0$. There are several cases. If g is an input gate of the circuit, then M accepts iff $k = 1$ and g has value 1. If $g = AND(g_1, g_2)$, then M guesses two numbers k_1 and k_2 (both greater than 0), checks whether $k_1 + k_2 = k$ holds, and, finally, makes two recursive calls to verify that g_1 resp. g_2 have saturation greater or equal than k_1 and k_2, respectively. If g is a bounded OR-gate, then M first of all guesses whether g itself is a saturated gate (i.e., both the inputs have value 1). If so, then M guesses two numbers k_1 and k_2 greater than 0, checks whether $k_1 + k_2 = k - 1$, and, finally, makes two recursive calls like in the preceding case. If M guessed that g itself is not a saturated gate or if g is an unbounded OR-gate (and, therefore, must be vulnerable), then M guesses one input g_i of g and recursively verifies that g_i has saturation greater equal than k.

Second, we come to complexity bounds and correctness of M. In each recursive call M branches into at most two recursive calls. Because of the at most logarithmic depth of the circuit with output g, the polynomial time bound is immediate. The logarithmic space bound and the correctness are straightforward and the case of coincidence of guessed and actual saturation is treated in an analogous way like the corresponding case in Lemma 16. □

The following proposition is the equivalent to Theorem 14 of the preceding section and Theorem 19. In an fully analogous way to there we employ the inductive counting technique on leveled circuits in combination with the verification algorithm of Lemma 24. This proposition serves as fundamental constituent of our final result, the inclusion of $UnambSAC^k$ in $UnambAPDA^k$.

Proposition 25 *$UnambSAC^1 \subseteq UnambAPDA^1$.*

Proof. The proof results from a straightforward translation of the proof of Theorem 14. While there ambiguity was used as a unit of measure, we here make use of the saturation of gates. □

Finally, we are ready to state one of our main results. This theorem solves an open problem posed by Lange [Lan90] in generalized form.

Theorem 26 *$UnambSAC^k \subseteq UnambAPDA^k$.*

Proof. A leveled circuit of depth $\log^k n$ can be regarded as a circuit of $\log^{k-1} n$ circuit layers, each of depth $\log n$ (that is, each of these circuit layers is an $UnambSAC^1$-circuit). The essential trick is that we now do a simulation for each of these layers similar to that of Proposition 25. Here the problem arises that, when simulating such an $UnambSAC^1$-circuit layer by an AuxPDA M, in general we do not have automatically the values of the input gates at disposal. Therefore, M recursively computes those values each time they are needed. (Note that the simulation starts in the highest (that is, output-) $UnambSAC^1$-layer of the given circuit.)

Observe that at the transition from one circuit layer to another we forget in some respect information. That is, to compute the value of an input gate g of some circuit layer i, M begins a (re)computation which actually provides the saturation of g. But then M is only interested in whether g has saturation greater than 0 (i.e., g has value 1) or g has saturation 0 (i.e., g has value

0). This is necessary because, otherwise, the saturation values were not any longer polynomially bounded. (About this, especially observe the subsequent Remark 27.)

Nevertheless, the whole simulation obviously remains unambiguous. The logarithmic space bound and the correctness of the simulation are straightforward and such the only interesting thing which remains to be shown is the time bound $c^{\log^k n}$ $(= n^{O(\log^{k-1} n)})$.

For each *UnambSAC*^{1}-circuit layer only polynomially many recursive calls are performed by M due to the polynomial time bound of the simulation for *UnambSAC*^{1}-circuits of Proposition 25. Thus, the recursion depth of $\log^{k-1} n$ yields a total running time of $n^{O(\log^{k-1} n)}$. □

Remark 27 We separated an *UnambSAC*^{k}-circuit in layers of depth $O(\log n)$. This is the only possibility we had because if the layers were chosen 'thicker' than $O(\log n)$, then the space of the simulating AuxPDA M would become greater than $O(\log n)$ and if the layers were chosen 'thinner' than $O(\log n)$, then the simulation time would become greater than $2^{O(\log^k n)}$.

Assume that we separate C in circuit layers of arbitrary depth D. Let $T(D)$ resp. $S(D)$ denote the time resp. space that are needed for the simulation of such a circuit layer with the help of the techniques of Proposition 25. Then clearly the following holds. (Observe that for $D > 0$ we have to assume a polynomial size for each of those circuit layers.) $T(D) = \max(d^D, n^{O(1)})$ for some constant d and $S(D) = \max(O(D), O(\log n))$. Obviously, $D = O(\log n)$ is optimal to gain a simulation in $c^{\log^k n}$time and logarithmic space.

For the running time $t(n)$ of the whole simulation analogous to Theorem 26 it holds
$t(n) = T(D)^{\frac{\log^k n}{D}} \geq n^{O(\frac{\log^k n}{D})} = c^{\frac{\log^{k+1} n}{D}}$.
Consequently, if $D = o(\log n)$, then $t(n) = \omega(c^{\log^k n})$.

4 Other applications

In this section, we will utilize the mutual characterization of AuxPDA and circuits especially for unambiguous classes. First, we deal with the restriction of pushdown heights of unambiguous AuxPDA and, second, we introduce the notion 'oblivious' for AuxPDA and show that in the most interesting cases it is no restriction to demand obliviousness. In addition, oblivious and unambiguous AuxPDA classes will prove to conincide with *WeakUnambRAC*^{k} and *UnambRAC*^{k} for arbitrary k. In this way, we extend a result of [LR90], where only a characterization for $k = 1$ was given.

4.1 AuxPDA with restricted pushdown height

For Turing machines there is a great interest in simultaneous resource bounds, i.e., restricting time and space bounds at the same time. As far as AuxPDA are concerned, one most of the time deals with simultaneous bounds on running-time and working space. But what is about the unlimited pushdown store? There has also been a lot of research to restrict the size of the pushdown store. Harju [Har79] showed (also see [Ruz80] for an alternative proof) that deterministic AuxPDA with polynomial running-time and logarithmic working-tape can be simulated by deterministic AuxPDA with logarithmic space and $O(\log^2 n)$ pushdown-height. However, the simulation yields a superpolynomial running-time. But later on, Dymond and Ruzzo [DR86] could prove the above result where even the polynomial running-time can be preserved. The dual result for nondeterministic AuxPDA (with also preservation of the polynomial running-time) could be shown earlier by Ruzzo [Ruz80]. This result can be generalized: $s(n)$ space and $r(n)$ reversal bounded nondeterministic AuxPDA can be simulated within same space and reversal bounds and with a pushdown height bounded by $O(s(n) \cdot \log(r(n)))$ [Kin81, BH89]. Herein, the reversal bound refers to the restriction of the number of pushdown reversals. Observe that according to [BH89] for polynomial reversal and logarithmic space bounded AuxPDA we get a polynomial time bound. From this, we can deduce Ruzzo's result.

Subsequently, we will restrict pushdown size for unambiguous and strongly unambiguous AuxPDA. For this purpose, we make use of the characterization of AuxPDA by circuits and vice versa. This is done in the following way. Assume that we have a circuit C (which only has bounded AND-gates) of depth $d(n)$ and size $z(n)$ simulating an AuxPDA M. Then we again simulate C by an AuxPDA N in the usual way (for example, cf. [Ven87]): Starting at the output gate, for AND-gates we examine both children and for OR-gates only one guessed child. Because we only need to store constant many parameters (with a space requirement of $O(\log(z(n))))$ of the recursive calls, a pushdown-height of $O(d(n) \cdot \log(z(n)))$ is immediate. Because of the equality of the considered AuxPDA and circuit class, we clearly have that N has the same time and space complexity as M. Our first application of the described technique yields a generalization of the above mentioned result of Ruzzo [Ruz80].

Theorem 28 *L is accepted by a NAuxPDA in $\log n$ space and $2^{O(\log^k n)}$ time iff L is accepted by such a machine which, furthermore, uses at most $O(\log^{k+1} n)$ pushdown-height.*

Proof. Just make use of Venkateswaran's result, given in Proposition 18. □

Making use of two main results of [LR90], we further gain the proposed restriction of the pushdown heights for unambiguous and strongly unambiguous AuxPDA. Unfortunately, we have such a result only for polynomial time AuxPDA.

Theorem 29 *L is accepted by an unambiguous resp. strongly unambiguous AuxPDA in $\log n$ space and polynomial time iff L is accepted by such a machine which, furthermore, uses at most $O(\log^2 n)$ pushdown-height.*

Proof. Only make use of the equalities $WeakUnambRAC^1 = UnambAPDA^1$ resp. $UnambRAC^1 = StUnambAPDA^1$ to be found in [LR90]. □

Note that in [BHS90] a small pushdown size was obtained for a special case of unambiguous AuxPDA. The above technique now yields small pushdown sizes even for an arbitrary, unambiguous AuxPDA.

4.2 Obliviousness for AuxPDA

An automaton is called *oblivious* if the movements of all its heads are independent from the input except its length. This property is easily achieved for space bounded Turing machines. Roughly speaking, we just move the heads to and fro the two ends of the respective tape contents.

For AuxPDA obliviousness is not so easy to attain because of the pushdown store head. But here the characterization of AuxPDA by circuits and vice versa helps. The main idea behind again is to simulate a circuit by an AuxPDA. If the circuit has a regular structure, namely strictly alternating (i.e., for all $i \geq 0$, all gates on level $2i+1$ are OR-gates and all gates on level $2i+2$ are AND-gates) and leveled, then the shape of the diagram which plots pushdown height versus time (subsequently called pushdown-diagrams) will also be very regular. This special shape of a pushdown-diagrams is called *W-cycle*[‡] (cf. Figure 1.). Note that the above mentioned normal form for circuits is easily achieved by an only doubled circuit depth and a polynomially enlarged circuit size. (See [BCD$^+$89] for details.) Because we only consider logspace and at least polynomially time bounded AuxPDA, for the input and working tapes of the AuxPDA we can use the above mentioned technique for space bounded Turing machines and, therefore, we altogether get an oblivious AuxPDA.

[‡]This name is taken from numerical mathematics, where it is used in the theory of multigrid methods.

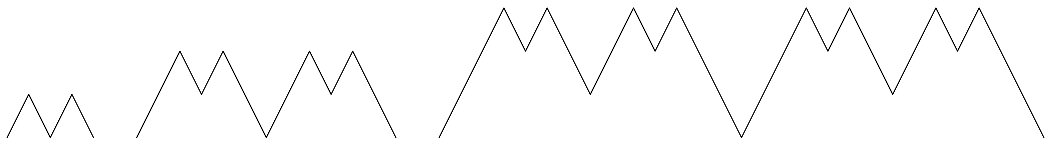

Figure 1: W-cycles

To simulate an circuit C (which itself simulates an given AuxPDA M), we employ nearly the same technique as in the preceding subsection. The only difference is that when we evaluate an AND-gate, we do this in a slightly modified way. First, we push the left gate on the store, then we evaluate it, afterwards we pop it from the store, and, finally, we compute the right input of the AND-gate and do the analogous computation for the right input. (Note that we only need the store for the evaluation of AND-gates.) Because of the symmetry of both the subcircuits of the AND-gate this altogether yields a pushdown-diagram in which the following holds. If we divide it into two equal parts (left and right) both are symmetric to each other and this also holds for a recursive division of these parts. In this way, we gain pushdown-diagrams in W-cycle form, i.e., a special case of obliviousness.

The following classes with preceding 'W-cycle' are defined in the intuitive way.

Theorem 30 *1. $NAPDA^k$ = (W-cycle)$NAPDA^k$.*

2. $UnambAPDA_{pt}(\log(n))$ = (W-cycle)$UnambAPDA_{pt}(\log(n))$.

3. $StUnambAPDA_{pt}(\log(n))$ = (W-cycle)$StUnambAPDA_{pt}(\log(n))$.

The proof of Theorem 30 is similar to the proofs of Theorem 28 and 29. In [LR90] the questions whether $UnambAPDA^k = WeakUnambRAC^k$ and whether $StUnambAPDA^k = UnambRAC^k$ hold for $k > 1$ remained open. We cannot fully answer this question, but if we confirm the consideration to W-cycle-oblivious AuxPDA classes, we get the desired equality.

Theorem 31 *1. (W-cycle)$UnambAPDA^k = WeakUnambRAC^k$.*

2. (W-cycle)$StUnambAPDA^k = UnambRAC^k$.

Proof.(sketch) The '$\subseteq$'-direction was implicitely proven in [LR90] respectively follows from the proof of Theorem 29.

To prove the reverse direction, we make use of the 'totally symmetric' shape of the pushdown-diagrams for W-cycle-oblivious AuxPDA. Again, we consider pairs of surface configurations (cf. Subsection 2.2), but the structure of the constructed circuit is simpler now. We mainly need gates of shape $\langle A, B\rangle$, which compute whether there exists a computation from surface configuration A to B, where the level of the pushdown store is the same for A and B and does not go below this level during the computation (cf. Figure 2). These gates are defined as

$$\langle A, B\rangle \quad \equiv \exists_{C,\ldots,G}\langle C, D\rangle \wedge \langle F, G\rangle \wedge \langle A \rightarrow C, E \rightarrow D\rangle \wedge \langle E \rightarrow F, G \rightarrow B\rangle,$$

where, for example, $\langle A \rightarrow C, E \rightarrow D\rangle$ computes whether there is an one-push-step from A to C and an one-pop-step from E to D (where first the symbol a is pushed and then popped).

The correctness and the weak resp. strong unambiguity of such defined circuit are straightforward. Because there are only polynomially many surface configurations and we recursively divide computation paths in two equal sized paths, the depth $O(\log^{k+1} n)$ and the polynomial size of the circuit are immediate. □

.

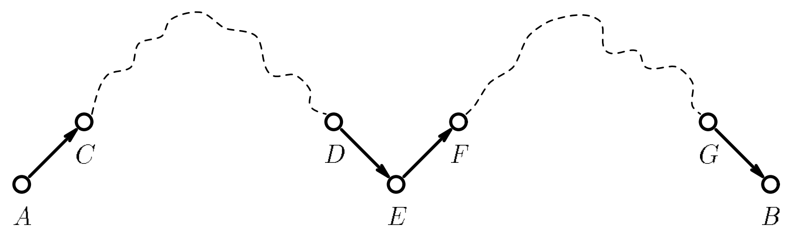

Figure 2:

5 Conclusion

This paper contains three sections. In the first section we showed how nondeterministic auxiliary pushdown automata whose ambiguity is low can be simulated by unambiguous ones. The second section contains results about the circuit classes $UnambSAC^k$, for which in contrast to $UnambRAC^k$ no nontrivial properties were known by now. In the third section certain normal forms of auxiliary pushdown automata are considered.

The first two sections use the same proof techniques: Inductive counting is combined with simulations between circuits and auxiliry pushdown automata. Often circuits played only a role in proofs to show a result for AuxPDA, while sometimes results about circuits themselves were proved. This proof technique works very well for semi-unbounded fan-in circuits and auxiliary pushdown automata, but it does not seem to be able to prove new results for space bounded classes. Nevertheless, it can be applied here, too, if we use the characterization of NL in terms of skew-circuits (see [Ven88]). In this way, we can even find a uniform proof for the closure under complementation for both NL and $LOGCFL$. Finally, the third section makes use of already known, mutual characterizations of AuxPDA and circuits to gain results about pushdown-height and obliviousness of AuxPDA. The essential trick here is to simulate the (poly)logarithmic depth-bounded and regularly structured circuits by AuxPDA. An interesting question is whether also time bounded, deterministic AuxPDA can be made oblivious.

For strong unambiguous circuit and AuxPDA classes many results are known now. However, many of them only hold for $O(\log n)$ depth circuits, respectively polynomially time bounded auxiliary pushdown automata. Some open questions that arise in this context are: Is $StUnambAPDA^k$ contained in $UnambRAC^k$ and thus $StUnambAPDA^k = UnambRAC^k$ for $k > 1$?. This question seems very hard to answer, but maybe at least $StUnambAPDA^k \subseteq UnambSAC^k$ can be obtained.

A similar question arises for the restriction of pushdown store height. While it is easy to see that nondeterministic auxiliary pushdown automata with running time bounded by $c^{\log^k n}$ use w.l.o.g. only $O(\log^{k+1} n)$ pushdown store space for arbitrary k, the same result was proved for deterministic, unambiguous, and strongly unambiguous AuxPDA only for $k = 1$.

References

[AKS83] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3:1–19, 1983.

[All86] E. W. Allender. The complexity of sparse sets in P. In *Proc. of 1st Structure in Complexity Conf.*, volume 223 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 1986.

[BCD$^+$89] A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo, and M. Tompa. Two applications of inductive counting for complementation problems. *SIAM Journal on Computing*, 18(3):559–578, 1989.

[BGH90] G. Beigel, J. Gill, and U. Hertrampf. Counting classes: Threshold, parity, mods, and fewness. In *Proc. of 7th Symposium on Theoretical Aspects of Computer Science*, number 415 in Lecture Notes in Computer Science, pages 49–57. Springer, 1990.

[BH89] G. Buntrock and A. Hoene. Reversals and alternation. In *Proc. of 6th Symposium on Theoretical Aspects of Computer Science*, number 349 in Lecture Notes in Computer Science, pages 218–228. Springer, 1989.

[BHS90] G. Buntrock, L. A. Hemachandra, and D. Siefkes. Using inductive counting to simulate nondeterministic computation. In *Proc. of 15th Symposium on Mathematical Foundations of Computer Science*, number 452 in Lecture Notes in Computer Science, pages 187–194. Springer, 1990. (to appear in *Information and Computation*).

[BJLR90] G. Buntrock, B. Jenner, K.-J. Lange, and P. Rossmanith. Unambiguousness and fewness for logarithmic space. Forthcoming paper, December 1990.

[Coo71] S. A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *Journal of the ACM*, 18:4–18, 1971.

[DR86] P. Dymond and W. L. Ruzzo. Parallel RAMs with owned global memory and deterministic language recognition. In *Proc. of 13th International Colloquium on Automata, Languages and Programming*, number 226 in Lecture Notes in Computer Science, pages 95–104. Springer, 1986.

[GS88] J. Grollmann and A. Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17:309–335, 1988.

[Har79] T. Harju. A simulation result for the auxiliary pushdown automaton. *Journal of Computer and System Sciences*, 19:119–132, 1979.

[Kin81] N. K. King. Measures of parallelism in alternating computation trees. In *Proc. of 13th ACM Symposium on Theory of Computing*, pages 189–201, 1981.

[Lan90] K.-J. Lange. Unambiguity of circuits. In *Proc. of 5th Structure in Complexity Conf.*, pages 130–137, 1990.

[LR90] K.-J. Lange and P. Rossmanith. Characterizing unambiguous augmented pushdown automata by circuits. In *Proc. of 15th Symposium on Mathematical Foundations of Computer Science*, number 452 in LNCS, pages 399–406. Springer, 1990.

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

[Ruz81] W. L. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22:365–383, 1981.

[Ryt87] W. Rytter. Parallel time $O(\log n)$ recognition of unambiguous context-free languages. *Information and Computation*, 73:75–86, 1987.

[Sni82] M. Snir. On parallel searching. In *Proc. SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 242–253, Ottawa, Canada, 1982.

[SV85] S. Skyum and L. G. Valiant. A complexity theory based on boolean algebra. *Journal of the ACM*, 32:484–502, 1985.

[Val76] L. Valiant. The relative complexity of checking and evaluating. *Information Processing Letters*, 5:20–23, 1976.

[Ven87] H. Venkateswaran. Properties that characterize LOGCFL. In *Proc. of 19th ACM Symposium on Theory of Computing*, pages 141–150, 1987.

[Ven88] H. Venkateswaran. Circuit definitions of nondeterministic complexity classes. In *Proc. of 8th Annual Conference FST & TCS*, number 338 in Lecture Notes in Computer Science, pages 175–192. Springer, 1988.