Unambiguous Polynomial Hierarchies and Exponential Size [*]

Klaus-Jörn Lange Peter Rossmanith

Fakultät für Informatik, Technische Universität München
Arcisstr. 21, 80290 München, Fed. Rep. of Germany

Abstract

The classes NC^k and AC^k are defined by computational devices of polynomial size*, i.e., by devices using a polynomially bounded number of gates or processors. We consider here the case of* exponential size*, which results in classes between P and PSPACE. In this way we get new characterizations of P and UP. The resulting relations of nondeterminism, unambiguity, and determinism to several types of simultaneous write access to a shared memory resemble perfectly the polynomial case. A new phenomenon is the equivalence of concurrent read, exclusive read, and owner read for arbitrary types of write access in the case of exponential size.*

In the exponential case circuits of bounded depth characterize the polynomial hierachy. Using the notion of an unambiguous circuit we give a uniform framework to relate the various types of unambiguous polynomial hierarchies and to explain their differences.

1 Introduction

Between the two classes $DSPACE(\log n)$ and P there is a rich structure of complexity classes, e.g., the SC^k, NC^k, or AC^k classes. These are defined by sequential models like Turing machines and auxiliary push down automata or by parallel models like PRAMs (parallel random access machines), uniform circuit families, or alternation [18]. While the running time or depth of these parallel models is bounded by the logarithm of the length of the input or some fixed power of it, the size, i.e., the number of processors or of gates, is polynomially bounded. In this paper we consider these classes and their relationships for the case of exponential size.

Starting point of our investigations are the following equations which relate several types of PRAMs, circuit families, and auxiliary push down automata:

[*]work supported by the Deutsche Forschungsgemeinschaft in SFB 0342 A4 "KLARA"

Pol 1: [21]
$AC^k = \mathit{CRCW\text{-}TIME}(\log^k n)$ for $k \geq 1$,

Pol 2: [25]
$SAC^k = \mathit{NAuxPDA\text{-}TIME}(2^{O(\log^k n)})$ for $k \geq 1$,

Pol 3: [11]
$UAC^k = \mathit{CREW\text{-}TIME}(\log^k n)$ for $k \geq 1$,

Pol 4: [12, 16]
$USAC^1 = \mathit{StUAuxPDA\text{-}TIME}(pol)$, and

Pol 5: [7]
$\mathit{CROW\text{-}TIME}(\log n) = \mathit{DAuxPDA\text{-}TIME}(pol)$.

These relations are depicted in Figure 1.

$$\begin{tikzcd}
 & AC^1 = CRCW^1 = A\Sigma^{\mathcal{L}}_{\log} \arrow[dl, dash] \arrow[dr, dash] & \\
UAC^1 = CREW^1 \arrow[dr, dash] & & SAC^1 = NAuxPDA_{pt} \arrow[dl, dash] \\
 & USAC^1 = StUAuxPDA_{pt} \arrow[dl, dash] & \\
CROW^1 = DAuxPDA_{pt} & &
\end{tikzcd}$$

Figure 1: Classes of Polynomial Size

The classes $\mathit{NAuxPDA\text{-}TIME}(pol)$ and $\mathit{DAuxPDA\text{-}TIME}(pol)$ are well known to be the $\leq^{\mathcal{L}}_m$–closures of CFL and of $DCFL$ [23].

In the case of exponential size, the following is known:

Exp 1: Stockmeyer and Vishkin's relation between circuits of unbounded fan-in and $CRCW$–PRAMs is not restricted to polynomial size: $AC^k_{exp} = CRCW_{exp}\text{-}TIME(\log^k n)$ for $k \geq 1$ [21] and

Exp 2: *NP* is the exponential-size analogue of *NAuxPDA-TIME*(*pol*): $SAC^1_{exp} = NP$ [26].

Here the index *exp* denotes classes defined by circuits of exponential size or by PRAMs with an exponential number of processors.

Indeed, Venkateswaran showed that equation Exp 2 holds in a more general setting:

Exp 2′: $SAC^k_{exp} = NTISP(2^{O(\log^k n)}, pol)$ for $k \geq 1$.

In addition, equations Pol 3 and Pol 4 are easily extended to:

Exp 3: $UAC^k_{exp} = CREW_{exp}\text{-}TIME(\log^k n)$ for $k \geq 1$ and

Exp 4: $USAC^1_{exp} = UP$.

Again, the last equation can be generalized to:

Exp 4′: $USAC^k_{exp} = UTISP(2^{O(\log^k n)}, pol)$ for $k \geq 1$.

We will complete this picture by deriving the exponential analogue of equation Pol 5, which gives in particular a new characterization of the class *P:*

Exp 5: $P = CROW_{exp}\text{-}TIME(\log n)$.

We can generalize Exp 5, too:

Exp 5′: For $k \geq 1$:

$$CROW_{exp}\text{-}TIME(\log^k n) = DTISP(2^{O(\log^k n)}, pol).$$

Figure 2, depicting the equations Exp 1 to Exp 5, shows the similarity of the polynomial and of the exponential case.

For exponential size circuits and PRAMs there is a new phenomenon that does not occur in the polynomial case. All classes mentioned so far coincide with $USAC^1_{exp}$ if we choose a weaker uniformity condition. Especially, if $USAC^1_{exp}$ and its exponential-time uniform analogue turn out to be the same classes, then the exponential size AC hierarchy collapses. The reason for this is that exponential size, semi-unbounded fan-in circuit can already express every function, thus only the uniformity condition restricts $USAC^1_{exp}$. On the other hand, nonuniform polynomial size circuits do *not* recognize all languages.

In general, complexity classes improve on robustness when going from polynomial size to exponential size. For example, we show that concurrent, exclusive, and owner read access restrictions all are equivalent when using an exponential number of processors. This holds for every type of write access restriction. For a polynomial number of processors, we only know the equivalence of exclusive read and owner read in the cases of concurrent write and of exclusive write [17]. All these equivalences mentioned so far pertain to the case of a logarithmic running time.

$$\begin{tikzcd}
 & & AC^1_{exp} = CRCW^1_{exp} = A\Sigma^P_{\log} \arrow[dl, dash] \arrow[dr, dash] & \\
 & UAC^1_{exp} = CREW^1_{exp} \arrow[dr, dash] & & SAC^1_{exp} = NP \arrow[dl, dash] \\
 & & USAC^1_{exp} = UP \arrow[dl, dash] & \\
 & CROW^1_{exp} = P & &
\end{tikzcd}$$

Figure 2: Classes of Exponential Size

Another example is the equivalence of weak and strong unambiguity for semi-unbounded circuits. The difference between these two notions of unambiguity is that in the case of weak unambiguity, for each input there must not exist two different accepting subtrees, i.e.: There is at most one way for the circuit to accept its input. On the other hand, in the case of strong unambiguity, we put this restriction on every gate of the circuit and not on the output gate, only. The gain of robustness is mainly caused by the bigger robustness of the family of exponential functions. Barrington [13] pointed out that also quasipolynomial size already implies a gain in robustness.

Despite the similarity of Figures 1 and 2 the polynomial case and the exponential one differ when considering circuits of bounded depth. While AC^0 is a proper subclass of NC^1 and hence also of *CROW-TIME*$(O(\log n))$, the class AC^0_{exp} coincides with the polynomial hierarchy:

$$SAC^1_{exp} = NP \subseteq AC^0_{exp} = PH \subseteq AC^1_{exp}.$$

An investigation of unambiguous hierarchies between *P* and *PSPACE* comprises the second part of this paper. The polynomial hierarchy has characterizations via oracles, bounded alternation, polynomially bounded quantifiers, and oracles restricted to one question [6, 19, 20, 22, 27]. Instead of using nondeterministic TM's we can also build polynomial hierarchies working with unambiguous computations. The many possibilities to define hierarchies then no longer seem to be equivalent. The largest hierarchy is an oracle hierarchy based on promise access to an unambiguous oracle. Promise access was investigated by Cai, Hemachandra, and Vyskoč [5]. For promise access it is sufficient that the oracle machine behaves unambiguously on queried words, but not necessarily on all possible words. An apparently weaker hierarchy is the unambiguous oracle hierarchy, followed by the unambiguous alternation hierarchy. The latter coincides with hierarchies built by unambiguous quantifiers and the one-query unambiguous oracle hierarchy.

We find a uniform framework that characterizes all these hierarchies in terms of circuits. If the circuits are weakly unambiguous, we get the unambiguous promise hierarchy, while strongly unambiguous circuits characterize the unambiguous oracle hierarchy. If the circuit is in a normal form such that the unbounded fan-in gates have to be at the top of the circuit, then we finally get the unambiguous alternation or quantifier hierarchy.

2 Preliminaries

The reader is assumed to be familiar with the basic facts of complexity theory as they are contained in [2] or [9]. So we will use without explanation the classes *DSPACE*$(\log n)$, *NSPACE*$(\log n)$, *P*, *NP*, and *PSPACE*. Throughout the paper "exponential" means bounded by a function in $2^{n^{O(1)}}$.

There are two results that are particularly important for this paper. First, the equivalence of unbounded fan-in circuits and CRCW-PRAMs due to Stockmeyer and Vishkin [21]. Their theorem is most often used for polynomial size circuits, but holds as well for exponential size ones. Second, the characterization by Venkateswaran of the polynomial hierarchy by constant depth, unbounded fan-in, exponential size circuits together with the characterization of *NP* by logarithmic depth, semi-unbounded fan-in, exponential size circuits [26].

We use the following notation:

- $XRYW_{exp}$-*TIME*$(t(n))$ is the class of languages accepted by $XRYW$-PRAMs with exponentially many processors in $O(t(n))$ steps.
- AC^k_{exp} and SAC^k_{exp} is the class of languages accepted by unbounded (semi-unbounded) fan-in circuit of exponential size and $O(\log^k n)$ depth.
- AC^k and SAC^k is the class of languages accepted by unbounded (semi-unbounded) fan-in circuit of polynomial size and $O(\log^k n)$ depth.
- *NAuxPDA-TIME*$(f(n))$ denotes the class of languages accepted by nondeterministic f(n)-time bounded auxiliary push down automata. Here, auxiliary means both two-way access to the input and augmentation with a logarithmically space bounded working tape. *DAuxPDA-TIME*(pol) denotes the corresponding deterministic class.
- *NTISP*(f, g) is the class of all langauges accpeted by Turing machines which are simultaneously f-time and g-space bounded. *DTISP*(f, g) denotes the corresponding deterministic class.

All other special notation will be introduced where needed.

3 Exponential Parallelism

In this section we investigate complexity classes defined by PRAMs with an exponential number of processors. It is well known that PRAMs using an exponential number of processors recognize in polynomial time exactly all languages in *PSPACE* regardless which type of memory access we allow. In the following, we investigate fixed powers of the logarithm function as running times, in particular $O(\log n)$ steps. As long as we allow polylogarithmic running time and a polynomial number of processors, there is no difference between PRAMs with different reading and writing ability from a complexity theoretic point of view, since a OROW-PRAM can simulate a CRCW-PRAM with only a logarithmic factor of loss in time. All models define together the class *NC*.

Working with an exponential number of processors, life is harder at first glance, since even a polylogarithmic overhead of the running time does not guarantee equivalence of the OROW and the CRCW model. Nevertheless, in fact life is *easier* in the exponential world as we will see shortly. Many classes collapse whose cousins with polynomial size withstood any attack to show their equality by now.

Venkateswaran showed in [26] the equivalence of semi-unbounded circuits of exponential size and nondeterministic polynomially space bounded machines:

Theorem 1

$$SAC^k_{exp} = NTISP(c^{\log^k n}, pol) \textit{ for } k \geq 1.$$

In particular this yields $NP = SAC^1_{exp}$.

Let us start by deriving a comparable result for $DTISP(f, g)$-classes. Our characterization will not work with circuits, but with CROW-PRAMs with an exponential number of processors and logarithmic running time. In the polynomial world we have $CROW\text{-}TIME(\log n) = LOG(DCFL)$. The deterministic flavor of CROW memory access also manifests itself when working with an exponential number of processors. The following result is only stated for polynomial space and time $2^{O(\log^k n)}$, but holds in a more general setting.

Theorem 2 *For $k \geq 1$:*

$$DTISP(2^{O(\log^k n)}, pol) = CROW_{exp}\text{-}TIME(\log^k n).$$

Proof: "$\subseteq$": The configuration graph of a polynomially space bounded computation contains at most an exponential number of nodes, since there are at most exponentially many different configurations. Since we consider only deterministic computations, we know also that this graph is in fact a forest, where each tree is of depth $O(c^{\log^k n})$. We further know that the degree of all nodes is constant since a configuration has at most a constant number of predecessors. A CROW-PRAM constructs this configuration forest in its memory by assigning one processor to each possible configuration. Each processor computes "its" successor and "its" predecessors in the forest and writes pointers in the appropriate locations in the global memory. In this way the global memory contains a representation of the computation graph in question. We can find out whether the initial and final configuration are in the same tree by simply performing pointer jumping.

"$\supseteq$": We adopt Dymond and Ruzzo's technique to simulate CROW-PRAMs deterministically [7]. They showed $CROW\text{-}TIME(\log n) \subseteq LOG(DCFL)$ by recursively evaluating functions that describe the contents of memory cells and states of processors at a given time. Let $G(m, t)$ be the contents of the mth memory cell after the tth step and $S(p, t)$ be the state of the pth processor after step t. The state of a processor includes its instruction counter as well as the contents of all its local memory cells. We can assume without loss of generality that there are only a constant number of local memory cells, so polynomial space suffices to encode a processor's state as well as a memory cell's contents.

In polynomial time q(n) we can compute $S(p, t)$ from $S(p, t-1)$ and $G(m, t-1)$ where m is the memory cell read when in state $S(p, t-1)$. Similarly, we can compute in time q(n) $G(m, t)$ from $G(m, t-1)$ and $S(p, t-1)$ where p is the owner of memory cell m. The recursion depth is $O(\log^k n)$, so time $O(q(n)2^{O(\log^k n)}) = 2^{O(\log^k n)}$ suffices for the whole recursion. $\square$

Corollary 1

$$P = CROW_{exp}\text{-}TIME(\log n).$$

In the proof of $P \subseteq CROW_{exp}\text{-}TIME(\log n)$ concurrent read seems to be absolutely necessary. Not so! It is true that it isn't possible to copy some information to many places in small time, but there is a trick to avoid it. For every possible input we create one instance of the simulating algorithm, that is, we make 2^n copies of the algorithm and each copy pretends to act on a different input. It is not necessary at all that a processor reads from the actual input. After termination of all algorithms we are confronted with 2^n possibly different results located in 2^n global memory cells. In $O(\log n)$ steps n processors can collect the whole input in the naïve binary tree way. A single processor can then use the whole input as an index to the 2^n possible results and access the correct one.

We can even prove a more general result. Concurrent, exclusive, and owner read access lead to the same complexity classes for an exponential number of processors.

Theorem 3 *For $k \geq 1$ we have:*

1. $$\begin{aligned} &CRCW_{exp}\text{-}TIME(\log^k n) \\ &\quad = ERCW_{exp}\text{-}TIME(\log^k n) \\ &\quad = ORCW_{exp}\text{-}TIME(\log^k n), \end{aligned}$$
2. $$\begin{aligned} &CREW_{exp}\text{-}TIME(\log^k n) \\ &\quad = EREW_{exp}\text{-}TIME(\log^k n) \\ &\quad = OREW_{exp}\text{-}TIME(\log^k n), \end{aligned}$$
3. $$\begin{aligned} &CROW_{exp}\text{-}TIME(\log^k n) \\ &\quad = EROW_{exp}\text{-}TIME(\log^k n) \\ &\quad = OROW_{exp}\text{-}TIME(\log^k n). \end{aligned}$$

Proof: The idea of the following construction is to resolve simultaneous read access by multiple computating all values computed in a computation.

Given a CRCW, CREW, or CROW algorithm we create $2^n P^T$ instances of this algorithm that will execute in parallel. Here P denotes the number of processors and T the number of steps in total. The $2^n P^T$ instances are arranged in 2^n blocks—one block for each possible input. All instances of algorithms in a block pretend to work on a fixed input, so no concurrent reads to the inputs are necessary. Each block contains P^T copies of the original algorithm. All of them simulate the first step of the original algorithm, but only $2^n P^{T-t+1}$ of them the tth step.

Let us assume that at the beginning of the tth step P^{T-t} copies of the algorithm have an exact copy of the original algorithm's memory contents in their own global memory. Then each processor reads the correct value—the same as in the original algorithm. If the processor writes, it also writes the right value. In the next step P times fewer instances of the algorithm will be alive.

For each instance that will be alive in the next step, there are $P-1$ instances that will be dead. At this point we can avoid concurrent reading by letting the pth processor in the instance that will be alive read from a memory cell in the pth instance that will be dead. We arrange that the pth processor in the living instance is the read-owner of all memory cells in the pth dying instance.

At the end, n processors determine in $\log n$ steps that block that worked on the correct input. $\square$

Corollary 2 *For $k \geq 1$ we have:*

1. $$\begin{aligned} &\mathit{DTISP}(2^{O(\log^k n)}, \mathit{pol}) = \\ &\quad = \mathit{CROW}_{\mathit{exp}}\text{-}\mathit{TIME}(\log^k n) \\ &\quad = \mathit{EROW}_{\mathit{exp}}\text{-}\mathit{TIME}(\log^k n) \\ &\quad = \mathit{OROW}_{\mathit{exp}}\text{-}\mathit{TIME}(\log^k n), \end{aligned}$$

2. $$\begin{aligned} P &= \mathit{CROW}_{\mathit{exp}}\text{-}\mathit{TIME}(\log n) \\ &= \mathit{EROW}_{\mathit{exp}}\text{-}\mathit{TIME}(\log n) \\ &= \mathit{OROW}_{\mathit{exp}}\text{-}\mathit{TIME}(\log n). \end{aligned}$$

4 Unambiguous Circuits

Unambiguous circuits are an analogue to unambiguous Turing machines [11]. There are two kinds of unambiguous circuits: weakly unambiguous and strongly unambiguous ones. A Turing machine or an auxiliary PDA is strongly unambiguous, if for all pairs of configurations there exist at most one valid computation connecting these configurations. This leads to the complexity classes *StUP* and *StUAuxPDA-TIME*(*pol*) [4]. Strong unambiguity is the more natural one in the world of circuits. A strongly unambiguous circuit consists of *robust* gates of bounded fan-in and *vulnerable* gates of unbounded fan-in. While there is no restriction for robust gates, vulnerable OR-gates must receive at most one 1 from their predecessors, and vulnerable AND-gates must receive at most one 0. Otherwise, a vulnerable gate is destroyed and outputs a value that causes vulnerable gates that receive this value to be destroyed, too. In an unambiguous circuit no vulnerable gate must be destroyed. This restriction must be fulfilled for all possible input values. In the following, we assume that all unambiguous circuits are layered, that is, all paths from a gate to an input have the same length. In this way we come to the classes UAC^k for $k > 0$. Strongly unambiguous circuits show tight relations to exclusive write PRAMs [11]: As an unambiguous analogue of $\mathit{CRCW}\text{-}\mathit{TIME}(\log^k n) = AC^k$ we have $\mathit{CREW}\text{-}\mathit{TIME}(\log^k n) = UAC^k$ [10]. To define unambiguous circuits of semi-unbounded fan-in, it turns out to be reasonable to forbid robust OR-gates of bounded fan-in. Thus an $USAC^k$-circuit is an unambiguous circuit consisting of vulnerable OR-gates of unbounded fan-in and robust AND-gates of bounded fan-in.

We have $\mathit{StUAuxPDA}\text{-}\mathit{TIME}(\mathit{pol}) = USAC^1$ [12] as an strongly unambiguous analogue of $\mathit{NAuxPDA}\text{-}\mathit{TIME}(2^{O(\log^k n)}) = SAC^k$.[1]

A Turing machine or an auxiliary PDA is (weakly) unambiguous if for each input there exists at most one accepting computation. This concept leads to the complexity classes *UP* and *UAuxPDA-TIME*(*pol*). We define *weakly unambiguous* circuits analogously to unambiguous Turing machines. A circuit is weakly unambiguous, if there is at most one accepting subtree for each possible input. We call the respective weakly unambiguous circuit classes $WUSAC^1$ and $WUAC^1$.

As an weakly unambiguous analogue of *NAuxPDA*-TIME$(2^{O(\log^k n)}) = SAC^k$ we have $\mathit{UAuxPDA}^1 = \mathit{WUSAC}^1$ [12]. For more results in this field we refer to [3, 4, 14]. In the following we consider circuit families of exponential size.

We denote by $USAC^1_{exp}$, $WUSAC^1_{exp}$, and UAC^1_{exp} the classes of languages corresponding to the classes $USAC^1$, $WUSAC^1$, and UAC^1 if we replace the polynomial size bound by an exponential one.

All results about polynomial size circuits transfer to corresponding relations in the exponential world. In most cases, however, these are rather odd ones, or we can find better results with a little bit more effort. We start with unbounded fan-in circuits:

Theorem 4 *For $k \geq 1$:*

$$UAC^k_{exp} = \mathit{CREW}_{\mathit{exp}}\text{-}\mathit{TIME}(\log^k n).$$

[1]Observe that $SAC^1 = LOG(CFL)$ does not seem to carry over to $USAC^1$ and $LOG(UCFL)$.

Proof: "$\subseteq$" We assign one memory cell to each gate and one processor to each wire. These processors compute the boolean functions of a gate and write the result into the gate's memory cell, which in the beginning has a 1 in the case of an OR gate and a 0 in the case of an AND gate. There are no write conflicts—at most one processor writes for a vulnerable gate, and the writes for robust gates can be carried out one after the other.

"$\supseteq$" For the following construction it is crucial, that our PRAMs have a polynomially bounded word length. We assume that there is no indirect access to local memory. Since the program is finite, each processor accesses only a constant number of local memory cells. Altogether a polynomial number of bits suffices to describe the state of a processor. We denote the state of processor p in step t by $\mathit{State}(p,t)$ and the contents of the mth memory cell in step t by $\mathit{Global}(m,t)$. We construct a circuit with gates $\langle \mathit{State}(p,t)=s\rangle$ and $\langle \mathit{Global}(m,t)=a\rangle$ for all possible states s and memory contents a. Gate $\langle \mathit{State}(p,t)=s\rangle$ computes whether $\mathit{State}(p,t)=s$. Its interconnection is

$$\langle \mathit{State}(p,t)=s\rangle \equiv \\ \exists_{(s',m,a)}\langle \mathit{State}(p,t-1)=s'\rangle \wedge \langle \mathit{Global}(m,t-1)=a\rangle,$$

where we quantify over (s',m,a) such that a processor reads from memory cell m when in state s', and enters state s if it reads value a. Gate $\langle \mathit{Global}(m,t)=a\rangle$ computes whether $\mathit{Global}(m,t)=a$. Its interconnection is

$$\langle \mathit{Global}(m,t)=a\rangle \equiv \\ \langle \mathit{Global}(m,t-1)=a\rangle \wedge \forall_{(p,s)}\neg\langle \mathit{State}(p,t-1)=s\rangle \\ \vee \exists_{(p,s)}\langle \mathit{State}(p,t-1)=s\rangle.$$

The universal quantification runs over (p,s) such that a processor in state s writes into memory cell m. The existential quantification runs over (p,s) such that a processor in state s writes a into memory cell m. The subformulae before and after the OR reflect the cases that (i) no processor writes into m, but it contained already a, and (ii) some processor writes a into memory cell m. Since at every step the state of a processor as well as the content of a memory cell are uniquely determined and since the simulated PRAM obeys the exclusive write restriction, the construed circuit is unambiguous. $\square$

By Theorem 3 we have:

Corollary 3 *$\mathit{UAC}^k_{\mathit{exp}} = \mathit{XREW}_{\mathit{exp}}\text{-}\mathit{TIME}(\log^k n)$ for $X \in \{C,E,O\}$ and $k \geq 1$.*

The unambiguous variant of *NP* is the well-known class *UP* introduced by Valiant [24] as the class of languages accepted in polynomial time with at most one accepting path. We defined the class *StUP* (strongly unambiguous polynomial time) as the class of languages accepted by polynomial time bounded Turing machines that have at most one computation path between two arbitrary configurations and are unambiguous.

Since $\mathit{USAC}^1 = \mathit{StUAuxPDA}^1$ and $\mathit{WUSAC}^1 = \mathit{UAuxPDA}^1$, we obviously expect $\mathit{USAC}^1_{\mathit{exp}} = \mathit{StUP}$ and $\mathit{WUSAC}^1_{\mathit{exp}} = \mathit{UP}$, and this is indeed the case. In contrast to polynomial size, however, we can show that weak and strong unambiguity defines the same complexity classes, if the size is exponential.

Theorem 5
$\mathit{USAC}^1_{\mathit{exp}} = \mathit{StUP} = \mathit{UP} = \mathit{WUSAC}^1_{\mathit{exp}}$.

Proof: A nondeterministic multi-tape Turing machine without space bound can write a protocol of its computation onto a separate tape. In this way its computation graph becomes a forest: No configuration has two predecessors. From this $\mathit{StUP} = \mathit{UP}$ follows.

Next, we show $\mathit{WUSAC}^1_{\mathit{exp}} \subseteq \mathit{UP}$. An accepting subtree of an semi-unbounded circuit has only polynomial size and can be guessed in polynomial time. The guessing Turing machine works unambiguously, since there is at most one accepting subtree by definition.

Finally, we show $\mathit{StUP} \subseteq \mathit{WUSAC}^1$. We use gates $\langle c_1, c_2, t\rangle$ that compute whether configuration c_2 is reachable from configuration c_1 in exactly t steps.

$$\langle c_1,c_2,t\rangle \equiv \exists_{c_3}\langle c_1,c_3,\lfloor t/2\rfloor\rangle \wedge \langle c_3,c_2,\lceil t/2\rceil\rangle$$

The circuit guesses the middle of the path. Since there is only one middle configuration, all gates are vulnerable and the circuit is unambiguous. $\square$

Observe, that this is the trick that also proves the equivalence of nondeterministic and symmetric time. In a more general setting, it is possible to show the following relations, which we state without proof:

Theorem 6 *For each $k \geq 1$ we have:*

$$\mathit{WUSAC}^k_{\mathit{exp}} = \mathit{UTISP}(c^{\log^k n}, \mathit{pol}) \text{ and}$$

$$\mathit{USAC}^k_{\mathit{exp}} = \mathit{StUTISP}(c^{\log^k n}, \mathit{pol}).$$

5 Bounded Depth

In this section we look at the various unambiguous polynomial hierarchies. We will relate them to *UAC*-circuits of bounded depth. In this way we will get new aspects of classes defined by bounded ambiguities.

UP-hierarchies

Stockmeyer [22] defined the polynomial hierarchy by Turing reductions. There are also characterizations of the polynomial hierarchy in terms of bounded depth alternation [6], polynomial length bounded existential and universal quantification of P [20, 22, 27], and by one-query access to an oracle [19, section 5].

If we shift our interest from nondeterministic to unambiguous computation, we are faced with these four approaches to define a hierarchy. For unambiguous computations, the resulting unambiguous hierarchies do not longer seem to coincide. There are at least three possibilities:

1. There is an unambiguous base machine that queries an unambiguous oracle. If the base machine must be unambiguous for every possible oracle, the resulting complexity classes are rather weak [5]. We therefore require the base machine to be unambiguous for only one particular oracle.

   If the oracle language is accepted by a Turing machine that behaves unambiguously on at least all queried inputs (but not necessarily on all inputs) we get the unambiguous promise hierarchy $\mathcal{UPH}$ with levels $\Sigma_k^{\mathcal{UP}}$.

2. The alternative is to require that the oracle is accepted by an unambiguous Turing machine that is unambiguous on all inputs (i.e., the usual definition), and not only on those that are queried by the base machine. This leads to the unambiguous oracle hierarchy UPH with levels Σ_k^{UP}.

3. The levels of the unambiguous one-query hierarchy coincide with levels defined via unambiguous quantification or bounded alternation of an unambiguous Turing machine. We call the resulting classes $A\Sigma_k^{UP}$ and the hierarchy is $AUPH$.

From these characterizations we immediately get the following relationship:

Theorem 7 $A\Sigma_k^{UP} \subseteq \Sigma_k^{UP} \subseteq \Sigma_k^{\mathcal{UP}}$.

Unambiguous AC^0_{exp}-circuits

From the work of Venkateswaran [26] it follows immediately that $PH = AC^0_{exp}$. We define $UAC^{0,k}_{exp}$ as follows. A language belongs to $UAC^{0,k}_{exp}$, if it is accepted by an unambiguous circuit of unambiguous circuit of exponential size such that at most k unbounded fan-in vulnerable gates are on each path. Additionally, the circuit may contain bounded fan-in robust gates, but at most $O(\log n)$ on each path. The corresponding weakly unambiguous class is called $WUAC^{0,k}_{exp}$ and we define

$$UAC^0_{exp} := \bigcup_k UAC^{0,k}_{exp} \text{ and}$$
$$WUAC^0_{exp} := \bigcup_k WUAC^{0,k}_{exp}.$$

Furthermore we define the classes $UAC^{0,k}_{exp}$-NF and $WUAC^{0,k}_{exp}-NF$ via the additional restriction for the underlying circuits that all vulnerable gates are at the top of the circuit, that is, on every path there are vulnerable gates followed by robust gates. For this normal form, weak and strong unambiguity coincide, i.e.,

$$WUAC^{0,k}_{exp}\text{-}NF = UAC^{0,k}_{exp}\text{-}NF.$$

The reason is simple: There is no difference between weak and strong unambiguity unless there are robust gates that get as their inputs the outputs of vulnerable gates.

The constant-depth unambiguous AC^0_{exp}-circuit classes provide us with a uniform setting for the three possibilities of unambiguous polynomial hierarchies.

Theorem 8

1. *$A\Sigma_k^{UP} = UAC^{0,k}_{exp}$-NF,*

2. *$\Sigma_k^{UP} = UAC^{0,k}_{exp}$,*

3. *$\Sigma_k^{\mathcal{UP}} = WUAC^{0,k}_{exp}$.*

Idea of proof: 1) To simulate an unambiguous alternating machine by UAC^0_{exp}-circuits in normal form, we express each "layer" of alternation by a vulnerable gate of unbounded fan-in; existential layers by OR gates and universal layers by AND gates. At the ends of the last layer we check the validity of the path constructed so far, which can be done by an NC^1-subcircuit. The uniformity of this curcuit can be fulfilled by assuming the simulated machine to be in some normal form; e.g., all layers are of the same depth, the machine first performs its alternations recording the nondeterministic decisions, and then deterministically checkes its decisions.[2] The simulation of an unambiguous circuit in normal form by an unambiguously alternating machine is quite obvious.

2) and 3) The representation of languages in Σ_k^{UP} by $UAC^{0,k}_{exp}$-circuits is done by induction over k. For this, we essentially use query-replacement of the base machine. Guessed negative answers directly lead to oracle queries, while positive guesses lead to a simulation of the base machine on level 1 and so on. This can be refined to a circuit of exponential size consisting of

[2]Thus, we essentially simulate here the unambiguous quantification of an NC^1-predicate.

one vulnerable OR gate of unbounded fan-in, many robust AND's of bounded fan-in and many $UAC^{0,k-1}_{exp}$-subcircuits with a vulnerable AND-gate on top. In total this results in an $UAC^{0,k}_{exp}$-circuit.

To show the reverse we simulate an $UAC^{0,k}_{exp}$-circuit top-down. A vulnerable OR is processed by (unambiguously) guessing the predecessor delivering the value true. Vulnerable AND's simply lead to an oracle query. Robust gates of bounded fan-in are simulated sequentially. Since the depth of robust gates is logarithmically bounded, this results in a polynomial running time. □

Corollary 4

1. *$AUPH = UAC^0_{exp}$-NF,*
2. *$UPH = UAC^0_{exp}$,*
3. *$\mathcal{UPH} = WUAC^0_{exp}$.*

From the above theorem we get immediately an alternative proof of Theorem 7. Though we know inclusions between the levels of the three hierarchies, we do not know whether the three hierarchies intertwine, or if some hierarchy is contained in a fixed level of one of the other hierarchies. Another possibility is that some (or all) of these hierarchies collapse to a fixed level. We will see later that a collapse would be hard to prove.

As another consequence we get the containment of all three hierarchies in *SPP*, the smallest gap-definable class [8].

Theorem 9 *$AUPH \subseteq UPH \subseteq \mathcal{UPH} \subseteq SPP$.*

Idea of proof: Let C be a weakly unambiguous UAC^0_{exp}-circuit. Without loss of generality we may assume C' not to contain robust OR-gates of bounded fan-in by replacing $A \vee B$ by $A \vee (\neg A \wedge B)$. The simulating *SPP* machine M will simulate C' top-down. A logical output 'true' of a gate will be simulated by M having gap 1 at that point of its subcomputation. The output 'false' will be represented by M having gap 0. If a vulnerable gate is destroyed, which may occur in a weakly unambiguous circuit, this may result in M having an arbitrary gap. A vulnerable OR gate x with m predecessors is processed by M by guessing a number $1 \le i \le m$ and recursively processing predecessor i. If at most one predecessor has gap 1 and all other have gap 0, this will lead to a subcomputation with gap 1, if the gate x outputs 'true', and with gap 0, if x outputs 'false'. If more than one of the predecessors carry a 'true' or if any predecessor carries an undefined value, M probably will will have a gap different from 0 or 1. A vulnerable AND gate x with m predecessors is again processed by guessing a number $1 \le i \le k$ and recursively processing the ith predecessor of x. But in addition, we add $m-1$ rejecting paths at this part of the computation of M. Assume x is not destroyed during the computation of C, i.e., it receives at least $m-1$ inputs corresponding to a gap 1 and probably one input corresponding to gap 0. But this means that M will have gap 0 if x evaluates to 'false' and will have gap 1 if x evaluates to 'true'. Again M can have gap $\neq 1$ or $\neq 0$, if x would be destroyed.

A robust AND gate is simulated sequentially by M, which leads to a multiplication of the incomming gaps. Thus the process of 'undefined $\wedge$ false $\mapsto$ false' is simulated by the multiplication of arbitrary gap with gap 0 resulting in gap 0. □

Bounded ambiguities

Let $UP_{\le d}$ denote the class of all languages that are recognizable by polynomially time bounded machines that accept their inputs in at most d different ways. The class *UP* and *FewP* [1] are special cases of $UP_{\le d}$:

$$UP = UP_{\le 1} \text{ and } FewP = UP_{\le pol}.$$

Our main result concerning $UP_{\le d}$ is that up to d accepting paths can be simulated by an unambiguous Turing machine with d alternations (see also Theorem 11 in [15]):

Theorem 10 *$UP_{\le d} \subseteq A\Sigma^{UP}_d$.*

The special case $d = n^{O(1)}$ leads immediately to $FewP \subseteq A\Sigma^{UP}_{-} =: AUP$ [15]. Following Theorem 8.1, we can say equivalently that *FewP* is contained in $UAC^{0,pol}_{exp}$-*NF*, since AUP is recognized by circuits of exponential size, consisting of a top level of polynomial depth of vulnerable gates of unbounded fan-in and an input level of logarithmic depth of robust gates of bounded fan-in. From this follows also that a CREW-PRAM with exponentially many processors could recognize *FewP* in polynomial time. But this is not very thrilling, since $CREW_{exp}$-$TIME(pol) = PSPACE$. In this connection, it is interesting to note that AUP is contained in SPP.

Theorem 11 [15] *$AUP \subseteq SPP$.*

On the other hand, it is easy to include *FewP* inside of $P^{\mathcal{UP}}$, and hence in $\Sigma^{\mathcal{UP}}_2$, or equivalently in $WUAC^{0,2}_{exp}$. Observe that by Theorem 9 this upper bound of *FewP* is contained in SPP, too.

6 Discussion and Open Questions

A central point of this work is the comparison of unambiguity to determinism and nondeterminism. It turned out that the case of polynomial time, or equivalently, of exponential size[3], resulted in a pattern of relations nearly identical to the case of logarithmic space (or of polynomial size). Thus unambiguity resembles nondeterminism and not determinism with respect to structural behavior. This is different from the view point of computational power, in particular with the case of logarithmic space, where there are relations indicating that unambiguity is rather related to determinism. While we have the equivalence of *UP* and of *StUP*, nothing like that is known for $USPACE(\log n)$ and $StUSPACE(\log n)$. In the latter case these classes ought to be different, as it is indicated by [4], where $StUSPACE(\log n) \subseteq DAuxPDA$-TIME(*pol*) was shown. Indeed, very similar methods yield also the closure under complementation of $StUSPACE(\log n)$ [4]. Observe that this question of closure under complementis still open for $UP = StUP$ and $USPACE(\log n)$.

These subtleties of unambiguous classes showed up again during the consideration of unambiguous polynomial hierarchies. The various definitions of polynomial hierarchies, which coincide in the case of nondeterminism, fall apart for unambiguous classes. It is convincing how the concept of unambiguity of circuits explained these differences and put them into a unifying frame. The relation between these hierarchies are still unknown. For instance, it is easy to see that the closure of *UP* under union would imply $UPH = A\Sigma_2^{UP}$, but what would be the consequences, if we could prove $A\Sigma_k^{UP}$ to be closed under union for some $k \geq 2$.

Acknowledgement

We thank Lane Hemaspaandra for the stimulating lecture and many ideas he gave us during his visit. Our special thanks go to Rolf Niedermeier for fruitful discussions on the topic of unambiguous circuits.

References

[1] E. W. Allender. *Invertible Functions.* PhD thesis, Georgia Institute of Technology, 1985.

[2] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity Theory I and II.* Springer, 1990.

[3] G. Buntrock, L. A. Hemachandra, and D. Siefkes. Using inductive counting to simulate nondeterministic computation. *Information and Computation*, 102:102–117, 1993.

[4] G. Buntrock, B. Jenner, K.-J. Lange, and P. Rossmanith. Unambiguity and fewness for logarithmic space. In L. Budach, editor, *Proc. 8th FCT*, number 529 in LNCS, pages 168–179, Gosen, Sept. 1991.

[5] J.-Y. Cai, L. Hemachandra, and J. Vyskoč. Promise problems and access to unambiguous computation. In I. Havel, editor, *Proc. 17th MFCS*, number 629 in LNCS, pages 162–171, Prague, Czechoslavakia, Aug. 1992.

[6] A. K. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *J. ACM*, 28:114–133, 1981.

[7] P. W. Dymond and W. L. Ruzzo. Parallel RAMs with owned global memory and deterministic language recognition. In *Proc. 13th ICALP*, number 226 in LNCS, pages 95–104. 1986.

[8] S. A. Fenner, L. J. Fortnow, and S. A. Kurtz. Gap-definable counting classes. In *Proc. 6th Structures*, pages 30–42, 1991.

[9] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[10] K.-J. Lange. Complexity and structure in formal language theory. In *Proc. 8th Structures*, pages 224–238, 1993.

[11] K.-J. Lange. Unambiguity of circuits. *Theoretical Comput. Sci.*, 107:77–94, 1993.

[12] K.-J. Lange and P. Rossmanith. Characterizing unambiguous augmented pushdown automata by circuits. In B. Rovan, editor, *Proc. 15th MFCS*, number 452 in LNCS, pages 399–406, Banská Bystrica, Czechoslovakia, Aug. 1990.

[13] D. A. Mix Barrington. Quasipolynomial size circuit classes. In *Proc. 7th Structures*, pages 86–93, 1992.

[14] R. Niedermeier and P. Rossmanith. Unambiguous simulations of auxiliary pushdown automata and circuits. In I. Simon, editor, *Proc. 1st LATIN*, number 583 in LNCS, pages 387–400, São Paulo, Brazil, Apr. 1992.

[3]which includes the case of an exponential number of processors

[15] R. Niedermeier and P. Rossmanith. Extended locally definable acceptance types. In P. Enjalbert, A. Finkel, and K. W. Wagner, editors, *Proc. 10th STACS*, number 665 in LNCS, pages 473–483. 1993.

[16] R. Niedermeier and P. Rossmanith. Unambiguous auxiliary pushdown automata and semi-unbounded fan-in circuits. 1993. To appear in *Information and Computation*.

[17] P. Rossmanith. The owner concept for PRAMs. In C. Choffrut and M. Jantzen, editors, *Proc. 8th STACS*, number 480 in LNCS, pages 172–183, Hamburg, Feb. 1991.

[18] W. L. Ruzzo. On uniform circuit complexity. *J. Comput. Syst. Sci.*, 22:365–383, 1981.

[19] W. L. Ruzzo, J. Simon, and M. Tompa. Space-bounded hierarchies and probabilistic computations. *J. Comput. Syst. Sci.*, 28:216–230, 1984.

[20] L. Stockmeyer and A. Meyer. Word problems requiring exponential time: preliminary report. In *Proc. 5th STOC*, pages 1–9, 1973.

[21] L. Stockmeyer and U. Vishkin. Simulation of parallel random access machines by circuits. *SIAM J. Comput.*, 13(2):409–422, May 1984.

[22] L. J. Stockmeyer. The polynomial time hierarchy. *Theoretical Comput. Sci.*, 3:1–22, 1977.

[23] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *J. ACM*, 25:405–414, 1978.

[24] L. Valiant. The relative complexity of checking and evaluating. *Inf. Process. Lett.*, 5:20–23, 1976.

[25] H. Venkateswaran. Properties that characterize LOGCFL. *J. Comput. Syst. Sci.*, 43:380–404, 1991.

[26] H. Venkateswaran. Circuit definitions of nondeterministic complexity classes. *SIAM J. Comput.*, 21(4):655–670, 1992.

[27] C. Wrathall. Complete sets and the polynomial hierarchy. *Theoretical Comput. Sci.*, 3:23–33, 1977.