Theoretical Computer Science, Vol. 194, pp. 137–161, 1998

Unambiguous Computations and Locally Definable Acceptance Types*

Rolf Niedermeier

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen,

Sand 13, D-72076 Tübingen, Fed. Rep. of Germany

Peter Rossmanith

Fakultät für Informatik, Technische Universität München,

Arcisstr. 21, D-80290 München, Fed. Rep. of Germany

October 31, 1996

*This paper is a completely revised and expanded version of a paper [27] presented at the "10th Annual Symposium on Theoretical Aspects of Computer Science."

Theoretical Computer Science, Vol. 194, pp. 137–161, 1998

Abstract

Hertrampf's locally definable acceptance types show that many complexity classes can be defined in terms of polynomial time bounded NTM's with simple local conditions on the nodes of its computation tree, rather than global concepts like number of accepting paths etc. We introduce a modification of Hertrampf's locally definable acceptance types which allows to get a larger number of characterizable complexity classes. Among others the newly characterizable classes are UP and $\textsc{Mod}Z_kP$. It is shown how different types of oracle access, e.g., guarded access, can be characterized by this model. This sheds new light on the discussion on how to access unambiguous computation. We present simple functions that describe precisely objects of current research as the unambiguous oracle, alternation, and promise hierarchies. We exhibit the new class UAP which seems to be an unambiguous analogue of Wagner's ∇P. UAP (and thus ∇P) contains *Few* and is currently the smallest class known with this property.

Theoretical Computer Science, Vol. 194, pp. 137–161, 1998

1 Introduction

Many classes of central interest in structural complexity theory are defined via polynomial time bounded, nondeterministic Turing machines (NTM's). A word w, for example, belongs to the language of a nondeterministic TM M, if M has at least one accepting path on input w. Many other acceptance mechanisms exist that are defined via numbers of accepting paths. More generally, *predicate classes* are defined via a predicate on computation trees of polynomial time bounded NTM's [5]. Hertrampf introduced an evaluation scheme for nondeterministic TM's (NTM's) that relies on evaluation of simple functions to be done locally in the nodes of a computation tree rather than to demand global conditions on it [19]. Using only OR as local functions yields NP, while AND yields Co-NP. Allowing both OR and AND results in the class $PSPACE$. This concept, called *locally definable acceptance type*, characterizes many important complexity classes, among them $\oplus P$, 1-NP, all levels of the polynomial hierarchy as well as all levels of the boolean hierarchy over NP (see [19, 18]). Examples for complexity classes that are predicate classes, but not known to be locally definable, are PP and BPP.

There exists, however, a large field of interesting classes that do not seem to be characterizable in terms of locally definable acceptance types or general predicates of computation trees, though they can intuitively be described by simple local conditions, among them the class unambiguous polynomial time UP [33]. Locally definable acceptance types were partially motivated by the wish to characterize as many complexity classes as possible in a uniform way. Demanding an outcome of 0 or 1 enables us to characterize more classes by local conditions, in particular unambiguous classes.

Borchert uses the concept of *promise classes* to characterize such complexity classes by a function with range $\{0, 1, \perp\}$ rather than a predicate [5]. In a similar way we introduce local functions on the nodes of a computation tree with the promise that the outcome of the function must be 0 or 1.

Unambiguous complexity classes are closely connected to the existence of one-way functions and public-key cryptology systems [15, 32]. Though they were always objects of central interest, just now there are attempts to examine them even more closely by analyzing different types of oracle and alternation hierarchies built on unambiguous classes, as well as negative and positive relativization results [10]. Hemaspaandra and Rothe investigate the Boolean, the nested difference, and the Hausdorff hierarchies over *UP* [17]. While the corresponding hierarchies over *NP* coincide, they present oracles providing several separations for the hierarchies over *UP*. They also investigate relations for the unambiguous polynomial and the unambiguous polynomial promise hierarchy. The levels of these hierarchies coincide with classes defined via CREW-PRAM's with exponentially many processors and unambiguous circuits of exponential size [26]. There is a tight relationship between these circuit characterizations and the presentation via locally definable acceptance types. A slightly modified characterization is also one key to the construction of separating oracles for the unambiguous polynomial hierarchy [30].

The approach to build hierarchies has proven to be fruitful in the case of *NP*. It is one aim of this paper to fully characterize all in an intuitive sense locally definable complexity classes. In order to do so, we modify Hertrampf's definition of locally definable acceptance types to capture also *UP* and related classes, as well as classes derived by different kinds of relativized unambiguous classes. In contrast to the classes of the polynomial time hierarchy, there are several reasonable possibilities for oracle access to unambiguous computations. We study guarded access [10] and robustly unambiguous computations [31].

We show that all these unambiguous classes can be characterized by local functions on the nodes of computation trees, if only $\{0, 1\}$ is allowed for the outcome at the root of the tree.

The class of characterizable classes in this sense, is closed under complement, polynomial length bounded existential and universal quantifications and all other closure properties of [19]. In addition it is also closed under some closure properties which were introduced by Hemaspaandra. These include the unambiguous versions ∃! and ∀! of existential and universal quantification. The newly characterizable classes include *UP*, Co-*UP*, $UP_{\leq k}$, $\mathrm{MODZ}_k P$ and all levels of the unambiguous alternation, oracle, and promise hierarchies. Please note that our concept is not more complicated than Hertrampf's one and that proofs are not harder. We claim that this concept captures exactly the intuitive notion of "being definable by simple local conditions."

Chandra, Kozen, and Stockmeyer introduced *alternation* [11]. If a state is *alternating*, it is accepting if and only if its successor is rejecting. Alternation is by definition a local function and fits thus well into the concept of locally definable acceptance types as we can see in Hertrampf's works. We combine the concepts of alternation and unambiguous computation to get the new complexity class *UAP*. This class consists of all languages accepted by a polynomial time bounded TM that is 1) unambiguous, i.e., each configuration has at most one accepting successor, and 2) may contain also alternating configurations. It is clear that *UAP* lies somewhere between *UP* and *PSPACE*, but where exactly depends obviously on how much alternation can help in an unambiguous computation.

Alternation does not help at all in the case of deterministic computations, but helps a lot for nondeterministic computations (getting from *NP* to *PSPACE*). In fact, it turns out that alternation *does* help even in the case of unambiguous computations: $UP \subsetneq UAP$ unless $UP = \mathrm{Few}P$. On the other hand, though deterministic, unambiguous, and nondeterministic polynomial space are all the same, *UAP* seems to be less than *PSPACE*: We show $UAP \subseteq SPP$.

We can look at *UAP* in still another way, i.e., as an unambiguous version of Wagner's ∇P [35]. The relationship between *UAP* and ∇P is just the same as between *UP* and 1-*NP*. What is forbidden for the left case, leads to rejection in the right case.

The remainder of the paper is organized as follows: In the next section we start defining the modified notion of locally definable acceptance types and explain the difference to Hertrampf's original definition. In the third section we provide a toolkit of basic functions for locally definable acceptance types, the propagation principle as an important tool in many proofs, and some normal form results. Subsequently, in Section 4 we present characterizations of *UP*, ModZ_kP, *UAP*, study relations of *UAP* to complexity classes like *SPP* [13], *Few* [9], and ∇P [35], and demonstrate the closure of our locally definable classes under several complexity theoretic operators and relativization. Finally, in Section 5 we investigate access to unambiguous computations within our framework and, in particular, study guarded access to unambiguous computations and robustly unambiguous computations.

2 Locally Definable Acceptance Types

Goldschlager and Parberry introduced so-called *extended Turing machines* in order to generalize the concept of alternating Turing machines [14]. They studied the power of nondeterministic, time-bounded Turing machines with an altered manner of acceptance. Instead of only labeling the states (and thus also the configurations) of the machine just with AND, OR, NOT, Accept or Reject, as it is done in alternating Turing machines, they allowed states to be labeled with a larger range of functions, in particular, any set of Boolean functions or, equivalently, two-valued logic. Recently, Hertrampf further generalized this concept by permitting any functions from *m-valued logic* for some fixed integer m rather than only from Boolean logic [19]. Goldschlager and Parberry's extended Turing machines accept on 1 and reject on 0, but in m-valued logic one has more than two values. Hertrampf decided that for m-valued logic a TM accepts on 1 and rejects on all other values. Hertrampf provided a complete case analysis when exactly one binary function from three-valued logic is allowed [18] (there are 19683 possibilities) and thus found 20 possibly different complexity classes. Among them are the complete second level of the polynomial time hierarchy and ∇P [35].

In this paper we investigate a different way of acceptance for m-valued logic. A TM accepts on 1 and rejects on 0 and guarantees that no other values occur as the result of the computation. We begin with defining the basic notions common to our and Hertrampf's work.

Definition 2.1 Let $m \geq 2$ be an integer. A *function f from m-valued logic* is a function $\{0, \ldots, m \perp 1\}^r \to \{0, \ldots, m \perp 1\}$ for $r \geq 0$.

The set $\{0, \ldots, m \perp 1\}$ is called the *base set* for the domain and range of f. Clearly, the restriction to natural numbers in the above definition is only a matter of convenience. Subsequently we will use the fact that arbitrary symbols can be encoded as numbers. In addition, we assume that always some particular number, which is different from 0 or 1, is identified with the special value $\perp$. Whenever a function of our m-valued logic has $\perp$ as an argument, the function evaluates to $\perp$. The meaning of $\perp$ can be understood as "undefined." All functions we use are "strict" with respect to $\perp$.

Definition 2.2 An *m-valued locally definable acceptance type* is a set F of functions from m-valued logic. The base set of F is the union of the base sets of its functions. A locally definable acceptance type is called *finite* if F is a finite set. An *F-machine M* is a polynomial time bounded NTM, where each configuration with r successors is labeled by a function $f \in F \cup \{id\}$ with arity r, depending only on the state of M. The number of a configuration's successors must depend *only* on its state. (id denotes the identity on the base set of F.) Leaves in the computation tree are labeled with an integer from $\{0, \ldots, m \perp 1\}$ depending on the state. *Values* are assigned to nodes in the computation tree as follows: The value of a leaf is its integer label. An inner node c, labeled with f and having successors $c_1, \ldots, c_r$ with values $v_1, \ldots, v_r$, has value $f(v_1, \ldots, v_r)$. The value of the root of the computation tree is the *result* of M.

Hertrampf defined for each locally definable acceptance type F a complexity class $(F)P$ as follows [19]. Subsequently we introduce a possibly different class $[F]P$.

Definition 2.3 A language L is in $(F)P$ iff there is an F-machine M such that

$$w \in L \iff \text{the result of } M \text{ on input } w \text{ is } 1.$$

Definition 2.4 A language L is in $[F]P$, iff there is an F-machine M such that

$$\begin{aligned} w \in L &\iff \text{the result of } M \text{ on input } w \text{ is } 1, \\ w \notin L &\iff \text{the result of } M \text{ on input } w \text{ is } 0. \end{aligned}$$

For the ease of notation we call a class of languages $\mathcal{C}$ *locally definable* iff $\mathcal{C} = [F]P$ for some locally definable acceptance type F. There seems to be hardly any difference between Hertrampf's and our definition. In the case of $[F]P$, however, the F-machine must guarantee for all inputs w that the result is either 0 or 1, no other result is allowed. Thus it is no longer possible to diagonalize over F-machines, since it is undecidable whether an F-machine has this property. Also, some classes $[F]P$ may lack to have complete problems.

Borchert introduced predicate and promise classes [5]. Let $T(M, w)$ be the computation tree of a polynomial time NTM M on input w and let $\mathcal{T}$ be the set of computation trees. A language L is a *predicate class* if there is an M and an $f : \mathcal{T} \to \{0, 1\}$ with $w \in L \iff f(T(M, w)) = 1$.

A language L is a *promise class* if there is an M and an $f : \mathcal{T} \to \{0, 1, \perp\}$ with

$$\begin{aligned} w \in L &\Leftrightarrow f(T(M, w)) = 1, \text{ and} \\ w \notin L &\Leftrightarrow f(T(M, w)) = 0. \end{aligned}$$

Here, the function f is an arbitrary predicate on computation trees. While $(F)P$ are those predicate classes that are defined by *local* predicates, $[F]P$ are those promise classes that are defined by *local* predicates. Here a local predicate of a computation tree is a function from m-valued logic as defined in Definition 2.2.

3 Some Basic Concepts and First Results

Naturally the question arises whether there is a locally definable acceptance type F such that there is no locally definable acceptance type F' with $[F]P = (F')P$. This question is tightly connected to a major open problem in complexity theory [16]. Let $L_F = \{ \langle M, w \rangle \mid M \text{ is a timed } F\text{-TM and } w \in L(M) \}$. The language L_F is complete for $(F)P$, so each class characterizable by Hertrampf's locally definable acceptance types has complete languages. We will see (Subsection 3.2 and Theorem 4.1) that *UP* is characterizable in terms of our acceptance mechanism for locally definable acceptance types. As a consequence, not all $[F]P$ can be written as $(F')P$ unless *UP* has complete sets.

Though we do not know, whether *UP* has complete sets (there are both positive and negative relativizations, see [16]), we cannot hope to find a characterization of *UP* in terms of Hertrampf's locally definable acceptance type, unless this question is solved.

On the other hand, both kinds of locally definable acceptance types for polynomial time machines are at most as powerful as *PSPACE*. Hertrampf showed $(F)P \subseteq \mathit{PSPACE}$ for any F. The proof works for $[F]P$ too.

Proposition 3.1 *If F is a locally definable acceptance type, then $[F]P \subseteq \mathit{PSPACE}$.*

To simplify the presentation of our results we make several agreements for the rest of the paper. According to Definition 2.2 a locally definable acceptance type is a set F of functions from m-valued logic. For our version of locally definable acceptance types (Definition 2.4), where we have to guarantee that an F-machine either outputs 1 or 0, it will prove useful to allow F to consist of functions of different base sets. For example, consider $f_1: X_1^{k_1} \to X_1$ and $f_2: X_2^{k_2} \to X_2$ with $X_1 \neq X_2$. Then we settle that in fact f_1 and f_2 are extended to $\hat{f}_1, \hat{f}_2$ with base set $X := X_1 \cup X_2$ as follows: $\hat{f}_i: X^{k_i} \to X$, $\hat{f}_i(\vec{x}) := f_i(\vec{x})$ if $\vec{x} \in X_i^{k_i}$ and $\hat{f}_i(\vec{x}) := \perp$, otherwise. So we get the acceptance type $F = \{\hat{f}_1, \hat{f}_2\}$. For the ease of notation, we subsequently will always write $\{f_1, f_2\}$ when in fact meaning $\{\hat{f}_1, \hat{f}_2\}$.

In particular, to clearly separate different parts of the computation of an F-machine, the above technique is used for the special cases of underlined and overlined functions. For example, let $\textsc{Or} : \{0,1\}^2 \to \{0,1\}$ be defined in the natural way (also see the following subsection). Then $\overline{\textsc{Or}}$ is defined as the function from $\{\overline{0},\overline{1}\}^2$ onto $\{\overline{0},\overline{1}\}$, where $\overline{\textsc{Or}}(\overline{x},\overline{y})$ is the same as $\overline{\textsc{Or}(x,y)}$. Overlined functions as $\overline{\textsc{Or}}$ or values as $\overline{1}$ or $\overline{0}$ are simply new names. It is a notational convention, not complemention or anything. While $\overline{\textsc{Or}}$ is a new function, $\overline{0}$ ist a new value. This convention makes it possible to employ non-interfering $\textsc{Or}$-functions with different base sets in different parts of a computation tree of an F-machine. More generally, let $f : \{0,\ldots,m \perp 1\}^k \to \{0,\ldots,m \perp 1\}$. Writing $\overline{f}$ means the function $\overline{f} : \{\overline{0},\ldots,\overline{m \perp 1}\}^k \to \{\overline{0},\ldots,\overline{m \perp 1}\}$ with $\overline{f}(\overline{x_1},\ldots,\overline{x_k}) := \overline{f(x_1,\ldots,x_k)}$. Underlining is handled analogously.

3.1 A Toolkit of Basic Functions

To characterize a complexity class by locally definable acceptance types means, in the first place, to present a set of functions F, thus defining $[F]P$. It turns out that the set of functions we will need throughout the paper, often contain the same or quite similar functions. In the following we present a whole toolkit of such basic functions that will be used repeatedly in the forthcoming sections.

The toolkit can be classified in several drawers, containing functions whose purposes are similar. The first drawer contains logical functions.

3.1.1 Logical functions

The $\textsc{Or}$-function models existential nondeterminism and the $\textsc{And}$-function models universal nondeterminism, as already Hertrampf showed [19]: $NP = [\textsc{Or}]P$ and Co-$NP = [\textsc{And}]P$, where $\textsc{Or}$ and $\textsc{And}$ are defined in the usual way on base set $\{0,1\}$. In addition, we have *unambiguous* versions of these functions [25].

$$\textsc{Or!}(x,y) := \begin{cases} 0 & \text{if } x = y = 0, \\ 1 & \text{if } (x,y) \in \{(0,1),(1,0)\}, \\ \perp & \text{otherwise.} \end{cases} \qquad \textsc{And!}(x,y) := \begin{cases} 0 & \text{if } (x,y) \in \{(0,1),(1,0)\}, \\ 1 & \text{if } x = y = 1, \\ \perp & \text{otherwise.} \end{cases}$$

Just as OR and AND define *NP* and Co-*NP*, the unambiguous versions OR! and AND! define *UP* and Co-*UP*.

3.1.2 Functions that count

Many complexity classes are defined by counting and we introduce two types of functions that count. The domain of our functions is always finite and we cannot count in the group $\mathbf{Z}$. Instead we take a finite group $\mathbf{Z}_k$ and employ addition in this group as a function in our toolbox. We call this function $\textsc{Mod}_k\colon \{0,\ldots,k \perp 1\}^2 \to \{0,\ldots,k \perp 1\}$ because it adds modulo k: $\textsc{Mod}_k(x,y) = (x+y) \bmod k$.

We also define a variant of counting modulo k, where we keep track whether the result represents an "absolute" 0 or a 0 modulo k. We call this function $\textsc{Mod}\mathrm{Z}_k\colon \{\mathbf{0}^!, 0,\ldots,k \perp 1\}^2 \to \{\mathbf{0}^!, 0,\ldots,k \perp 1\}$, where the Z reflects that zeros are specially treated: There are two symbols for zero, the heavy zero, $\mathbf{0}^!$, and the light zero, 0. The only way to preserve a $\mathbf{0}^!$ is to add two $\mathbf{0}^!$'s. For the following definition remember that with respect to "+" and "mod" both zeros just have the numeric value zero. Their "heaviness" is just an attribute that plays no rôle in numeric computations.

$$\textsc{Mod}\mathrm{Z}_k(x,y) = \begin{cases} \mathbf{0}^! & \text{if } x = y = \mathbf{0}^!, \\ (x+y) \bmod k & \text{otherwise.} \end{cases}$$

There is another method to add numbers if the domain is limited. Instead of counting modulo k, we can also identify all numbers that are greater than some finite limiting number. In this way we get the function $\textsc{Add}_{\leq k}\colon \{0,\ldots,k \perp 1\}^2 \to$

$\{0, \ldots, k \perp 1\}$.

$$\textsc{Add}_{\leq k}(x, y) = \begin{cases} x + y & \text{if } x + y \leq k, \\ \perp & \text{otherwise.} \end{cases}$$

3.1.3 Functions that combine other functions

We need a monadic function that translates or encodes values in any way we like. We can use such a function, e.g., to identify 0 and $\mathbf{0}^!$, which are possible results of $\textsc{ModZ}_k$. We call such a function a *transformation function*, and in most cases its range consists only of two different values. That is, the transformation function partitions its domain into two sets. Let A and B be two sets and a and b two symbols. Then we write

$$\textsc{Trans}^{A \mapsto a}_{B \mapsto b}(x) := \begin{cases} a & \text{if } x \in A, \\ b & \text{if } x \in B, \\ \perp & \text{otherwise.} \end{cases}$$

The last item in our toolkit of standard functions is a function with three arguments. The value of the function is the first or the second argument according to the third argument. The third argument selects the first or the second argument and so we call this function SELECT. It is formally defined as

$$\textsc{Select}(x, y, z) = \begin{cases} x & \text{if } z = \overline{1} \text{ and } x, y \neq \perp, \\ y & \text{if } z = \overline{0} \text{ and } x, y \neq \perp, \\ \perp & \text{otherwise.} \end{cases}$$

The SELECT-function is essential for modeling oracle access. The first two arguments get the results from the two alternative branches of the computation, while the third argument gets the result of the oracle. The two branches and the oracle computation use different domains, that is, the third argument is overlined, to help distinguishing them. In the next subsection it will become clear how this can be exploited.

3.2 The Propagation Principle

Before we come to some of our first results, we present a principle to be applied to F-TM's, playing a central rôle in the whole work for characterizations via $[F]P$. It is called the *propagation principle.* Due to Definition 2.4 each root node of a computation tree of an F-TM must evaluate to 0 or 1, but not to any other value. Thus by definition not every computation tree is possible for an F-TM. If we choose F wisely, only computation trees of a very special kind are possible. We will often try to prove that some F-TM can accept only languages in some given complexity class. Here it helps if we know that all computation trees are of a simple structure—it makes proofs simpler.

As an example consider the case where $F = \{\text{Or!}\}$. To keep away value $\perp$ from the root, it is necessary that at most one leaf node of the tree has value 1 and all the other leaves are 0. Thus all computation trees of an Or!-TM have at most one leaf that evaluates to 1 and all other leaves evaluate to 0. We can prove this formally by induction on the depth of a computation tree. We prove that all reachable leaves of a root that evaluates to 0 evaluate to 0 and that among the leaves reachable from a root that evaluates to 1 there is exactly one that evaluates to 1, while all others evaluate to 0. If the root is a leaf, the assumption is trivially true. If the root is not a leaf itself, it has some children. All children must evaluate to 0 or at most one of the children evaluates to 1, since otherwise the configuration itself would evaluate to $\perp$, which is not the case by the definition of an F-TM. By induction hypothesis the subtree of the child labeled 0 has no leaf labeled 1 and the subtree of the child labeled 1 has exactly one leaf labeled 1.

Using this knowledge about the structure of computation trees of an Or!-TM, we can show that $[\text{Or!}]P \subseteq UP$. In the following we will use the propagation principle on the fly instead of making a formal induction proof as above.

For example, for $F = \{\overline{f}, \text{Trans}^{\overline{1}\mapsto 0}_{A\mapsto 1}\}$, where $A = \{\overline{0}, \overline{2}, \ldots, \overline{m \perp 1}\}$ and f is some function from $\{0, \ldots, m \perp 1\}^k$ into $\{0, \ldots, m \perp 1\}$, the propagation principle implies that non-trivial computation trees of an F-TM have the following shape: The root of the computation tree is labeled with $\textsc{Trans}^{\overline{1}\mapsto 0}_{A\mapsto 1}$ and is connected (by a deterministic step) to a computation tree where all inner nodes are labeled $\overline{f}$ and the leaf nodes are labeled by constants from $\{\overline{0}, \overline{1}, \ldots, \overline{m \perp 1}\}$. Otherwise, the root clearly would not evaluate to 0 or 1.

3.3 Normal Forms

This subsection presents some normal forms for $[F]P$, similar to Hertrampf's normal forms for $(F)P$. We can replace F by a single binary function. This helps making later proofs simpler.

Theorem 3.2 *Let F be a finite, locally definable acceptance type. Then there exists one binary function g such that $[F]P = [\{g\}]P$.*

Proof. The proof is similar to Hertrampf's $(F)P$ proof [19]. □

The next results relate $(F)P$ and $[F]P$ or transfer results known for $(F)P$ to $[F]P$.

Theorem 3.3 *For all locally definable acceptance types F there exists a locally definable acceptance type F' such that $(F)P = [F']P$.*

Proof. Let $\mathcal{C} = (F)P$. Then there exists a binary function $f\colon \{0, \ldots, m \perp 1\}^2 \to \{0, \ldots, m \perp 1\}$ such that $\mathcal{C} = (\{f\})P$ [19]. We claim that $F' = \{\overline{f}, \textsc{Trans}^{\overline{1}\mapsto 1}_{\overline{0},\overline{2},\ldots,\overline{m-1}\mapsto 0}\}$ shows $\mathcal{C} = [F']P$. This is an easy exercise. □

Lemma 3.4 *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be locally definable. Then the classes* Co-$\mathcal{C}_1, \mathcal{C}_1 \wedge \mathcal{C}_2 := \{ A \cap B \mid A \in \mathcal{C}_1, B \in \mathcal{C}_2\}$, *and* $\mathcal{C}_1 \vee \mathcal{C}_2 := \{A \cup B \mid A \in \mathcal{C}_1, B \in \mathcal{C}_2\}$ *are also locally definable.*

Proof. W.l.o.g. (Theorem 3.2) let $\mathcal{C}_1 = [\{f_1\}]P$ and $\mathcal{C}_2 = [\{f_2\}]P$. We claim that Co-$\mathcal{C}_1 = [\{\overline{f_1}, \textsc{Trans}^{\overline{1}\mapsto 0}_{\overline{0}\mapsto 1}\}]P$ and $\mathcal{C}_1 \vee \mathcal{C}_2 = [\{\textsc{Or}(\overline{\cdot}, \underline{\cdot}), \overline{f_1}, \underline{f_2}\}]P$. Herein, $\textsc{Or}(\overline{\cdot}, \underline{\cdot})$ is defined according to

$$\textsc{Or}(\overline{\cdot}, \underline{\cdot})(x, y) = \begin{cases} \textsc{Or}(u, v) & \text{if } x \in \{\,\overline{0}, \overline{1}\,\},\ y \in \{\,\underline{0}, \underline{1}\,\} \text{ and } \overline{u} := x,\ \underline{v} := y, \\ \perp & \text{otherwise.} \end{cases}$$

The claim for $\mathcal{C}_1 \wedge \mathcal{C}_2$ then follows by the closure under complementation and De Morgan's laws. Again the proof is straightforward. □

4 Unambiguous Complexity Classes

In this section we deal with a large spectrum of unambiguous complexity classes and characterize them via locally definable acceptance types. In addition, we introduce in a natural way the complexity class *unambiguous alternating polynomial time UAP*, and show its relations to known classes like *SPP*, ∇P, and *Few*. So *UAP* is the smallest known class that contains *Few*. We also study complexity classes defined by several operators like, e.g., unambiguous existential and universal quantification and unambiguous hierarchies.

4.1 Characterizations of $UP_{\leq k}$ and $\textsc{ModZ}_k P$

For a TM M we denote by $Accept(M, w)$ the number of accepting paths of M on input w. The class *UP* was defined by Valiant via unambiguous polynomial time bounded TM's [33]. Formally, a language L is a member of *UP*, if there exists a polynomial time bounded NTM M such that

$$Accept(M, w) = \begin{cases} 1 & \text{if } w \in L, \\ 0 & \text{if } w \notin L. \end{cases}$$

Unambiguity plays an important rôle in complexity theory (cf. [8, 25, 28] for some recent results) and, in particular, in cryptography where it is shown that the question for existence of one-way functions is equivalent to the question whether

$P = UP$ [15, 32]. Cai, Hemachandra, and Vyskoč [10] examined $UP_{\le k}$ as a generalization of UP. Here, for some constant number k, up to k accepting paths are allowed rather than only one:

$$\begin{aligned} 0 < \mathit{Accept}(M, w) &\le k \quad \text{if } w \in L, \\ \mathit{Accept}(M, w) &= 0 \quad \text{if } w \notin L. \end{aligned}$$

Theorem 4.1 *$UP_{\le k}$ is locally definable.*

Proof. We claim that $UP_{\le k} = [F]P$, where $F = \{\overline{\textsc{Add}_{\le k}}, \textsc{Trans}^{\bar{0}\mapsto 0}_{\bar{1},\ldots,\bar{k}\mapsto 1}\}$. The proof is straightforward using the propagation principle. □

Beigel, Gill, and Hertrampf [4] introduced $\textsc{ModZ}_k P$ as a variation of $\textsc{Mod}_k P$ [9]. Here, $L \in \textsc{ModZ}_k P$, if there is a polynomial time bounded NTM M, such that

$$\begin{aligned} &\mathit{Accept}(M, x) \not\equiv 0 \pmod{k} && \text{if} \quad x \in L, \\ &\mathit{Accept}(M, x) = 0 && \text{if} \quad x \notin L. \end{aligned}$$

In contrast to $\textsc{Mod}_k P$, for $\textsc{ModZ}_k P$ rejection by mk accepting paths is not allowed for $m \ge 1$. We have $SPP \subseteq \textsc{ModZ}_k P \subseteq \textsc{Mod}_k P \cap NP$ and $\textsc{ModZ}_k P$ has some very interesting closure properties [4, 13]. (SPP is formally introduced in the next subsection, Definition 4.4.)

Theorem 4.2 *$\textsc{ModZ}_k P$ is locally definable.*

Proof. The proof is similar to the proof of Theorem 4.1. As there the number of accepting computations has to be counted (but now modulo k). We claim that $\textsc{ModZ}_k P = [F]P$, where $F = \{\overline{\textsc{ModZ}_k}, \textsc{Trans}^{\overline{\mathbf{0}'}\mapsto 0}_{\bar{1},\ldots,\overline{k-1}\mapsto 1}\}$. (See 3.1.2 and 3.1.3 for the definitions of both functions.) For both directions we can proceed as in the proof of Theorem 4.1. □

4.2 Unambiguous Alternating Polynomial Time

Definition 4.3 We call an alternating TM *unambiguous*, if for all inputs its computation tree contains neither existential nodes with more than one accepting successor nor universal nodes with more than one rejecting successor. We call the class of languages accepted by this type of polynomial time bounded TM's *UAP*, *unambiguous alternating polynomial time.* The number of alternations is not bounded.

Fenner, Fortnow, and Kurtz [13] introduced a natural family of counting classes as a generalization of previously studied counting classes. Herein, the acceptance mechanism is defined via the difference between the number of accepting and the number of rejecting computations. In particular, they considered *SPP*, the smallest reasonable class of this family. They showed that *SPP* is the so-called gap analog of *UP* and that *SPP* languages are low for any gap-definable class, i.e., if $\mathcal{C}$ is gap-definable, then $\mathcal{C} = \mathcal{C}^{SPP}$.

Definition 4.4 Let $\text{gap}_M(w)$ denote the difference between the number of accepting and rejecting computations of a polynomial time NTM M on input w. Then *SPP* is the class of languages L such that

$$\begin{aligned} w \in L &\Rightarrow \text{gap}_M(w) = 1, \text{ and} \\ w \notin L &\Rightarrow \text{gap}_M(w) = 0. \end{aligned}$$

for some polynomial time bounded NTM M.

It is fairly easy to see that $UP \subseteq SPP \subseteq \oplus P$ [13], where $\oplus P$ is the same as Mod_2P. We can extend the first inclusion to *UAP*.

Theorem 4.5 *$UAP \subseteq SPP$.*

Proof. Let $L \in UAP$. Then there exists a polynomial time bounded unambiguous, alternating Turing machine M such that $L = L(M)$. Without loss of generality we may assume that each configuration of M is either final or has exactly two successors. In order to prove $L \in SPP$ we have to find a nondeterministic

TM M' such that $\mathrm{gap}_{M'}(w) = 0$ iff $w \notin L$ and $\mathrm{gap}_{M'}(w) = 1$ iff $w \in L$. We construct M' by taking M and replacing all existential and universal states by nondeterministic states. In the case of a universal state we add one rejecting state that is reachable only from this state. If a final state is accepting, the NTM has an identical state. In the case of a rejecting final state, the corresponding state of the NTM branches nondeterministically to an accepting and a rejecting final state, thus making its gap 0.

We say a configuration of a nondeterministic TM has gap g if the difference between reachable accepting and rejecting final configurations is g. We prove by induction on the heights of subtrees in the computation tree that a configuration of M' has gap 1 iff the corresponding configuration of M is an accepting one and that the configuration of M' has gap 0 otherwise.

The claim is by construction true for the leaves of the configuration tree. For an inner node c we distinguish two cases. If the configuration is an originally existential one and has children c_1 and c_2, then $\mathrm{gap}_{M'}(c) = \mathrm{gap}_{M'}(c_1) + \mathrm{gap}_{M'}(c_2)$. If c is accepting then either c_1 is accepting and c_2 rejecting and thus by induction hypotheses $\mathrm{gap}_{M'}(c_1) = 1$ and $\mathrm{gap}_{M'}(c_2) = 0$ or vice versa. In either case $\mathrm{gap}_{M'}(c) = \mathrm{gap}_{M'}(c_1) + \mathrm{gap}_{M'}(c_2) = 1$. If c is rejecting then both c_1 and c_2 are rejecting and $\mathrm{gap}_{M'}(c) = \mathrm{gap}_{M'}(c_1) + \mathrm{gap}_{M'}(c_2) = 0 + 0 = 0$.

The second case is that c is an originally universal configuration. Then c is accepting means that the children c_1 and c_2 are both accepting and thus $\mathrm{gap}_{M'}(c) = \mathrm{gap}_{M'}(c_1) + \mathrm{gap}_{M'}(c_2) \perp 1 = 1 + 1 \perp 1 = 1$. The $\perp 1$ comes from the additional rejecting child added in the construction of M'. If c is rejecting then either c_1 is accepting and c_2 rejecting or vice versa. Anyway, $\mathrm{gap}_{M'}(c) = \mathrm{gap}_{M'}(c_1) + \mathrm{gap}_{M'}(c_2) = 1 + 0 \perp 1$ or $0 + 1 \perp 1 = 0$. □

Wagner [35, 18] introduced some type of weak alternating machine, leading to the class ∇P. Our class UAP appears to be the unambiguous analog of ∇P.

Definition 4.6 The class ∇P is the set of languages accepted by some type of weak alternating TM's where existential configurations have at most one accepting successor and universal configurations at most one rejecting successor, and if this condition is violated the machine is said to reject by undefined behavior.

Now we show that UAP appears to be weaker than ∇P. Note that it is unknown whether ∇P is closed under complement, whereas UAP obviously is.

Proposition 4.7 *$UAP \subseteq \nabla P \cap \text{Co-}\nabla P$.*

Proof. In the same way as for alternating Turing machine classes it can be seen by application of De Morgan's laws that UAP is closed under complement. In addition, by definition it holds that $UAP \subseteq \nabla P$ and so the claim follows. □

Allender [1] introduced Few$P$, a subclass of NP and a generalization of $UP_{\leq k}$. Cai and Hemachandra [9] proved that Few$P \subseteq \oplus P$. The class Few$P$ is the set of languages recognized by polynomial time NTM's for which the number of accepting computations is bounded by a fixed polynomial in the input size. We now improve Few$P \subseteq \oplus P$ to Few$P \subseteq UAP$.

Theorem 4.8 *Few$P \subseteq UAP$.*

Proof. Let M be an NTM with at most $p(n)$ accepting paths, where p is some polynomial. Let L_m be the set of all words w with at least m accepting computation paths, that is, $L_m := \{ w \mid \mathit{Accept}(M, w) \geq m \}$. Clearly, $L_{p(n)+1} = \emptyset$.

For $0 \leq m \leq p(n)$ an unambiguous, alternating TM can compute L_m as follows: Nondeterministically choose method (i) or (ii).

(i) Determine, whether $w \in L_{m+1}$ and accept if this is indeed the case.

(ii) Guess m different computation paths $(p_1, \ldots, p_m)$ of M. If there are rejecting paths among $p_1, \ldots, p_m$ then reject. If all of them are accepting paths, then accept if $w \notin L_{m+1}$.

Induction shows that this computation is correct and unambiguous. □

Due to $UP_{\leq k} \subseteq \text{Few}P$ we immediately get the corollary $UP_{\leq k} \subseteq UAP$.

From Proposition 4.7 and Theorem 4.8 we also get the following new inclusion.

Corollary 4.9 $\text{Few}P \subseteq \nabla P \cap \text{Co-}\nabla P$.

The subsequently introduced class *Few-NP* stands in the same relation to Few$P$ as ∇P does to UAP or as 1-NP (also called US) to UP.

Definition 4.10 The class *Few-NP* is the set of languages recognized by polynomial time NTM's that accept a word iff they have more than zero and less than $p(n)$ accepting paths for some polynomial p and reject, otherwise.

We can also find *Few-NP* $\subseteq \nabla P$ as an analog of Theorem 4.8.

Corollary 4.11 *Few-NP* $\subseteq \nabla P$.

Proof. If $L \in$ *Few-NP* then there is some polynomial p and an NTM M such that

$$w \in L \Leftrightarrow \quad 1 \leq \mathit{Accept}(M, w) \leq p(n).$$

First guess $p(n) + 1$ different computation paths of M and check whether all of them are accepting. If they are, then reject "by undefined behavior," i.e., generate two accepting paths. If they are not all accepting proceed as in Theorem 4.8. Now if $\mathit{Accept}(M, w) > p(n)$ the algorithm rejects. If $\mathit{Accept}(M, w) \leq p(n)$ then there are at most $p(n)$ accepting paths and we can safely proceed as in the case of Few$P$. □

Cai and Hemachandra [9] introduced *Few* as a generalization of Few$P$. The class *Few* is the set of languages recognized by polynomial time NTM's for which the number of accepting computations is bounded by a fixed polynomial in the input size *and* there is polynomial time computable predicate that depending on the input word w and the number of accepting computations accepts or rejects w. In contrast to Few$P$, which is clearly contained in NP, that is unlikely for *Few*. We can, however, show the inclusion in UAP.

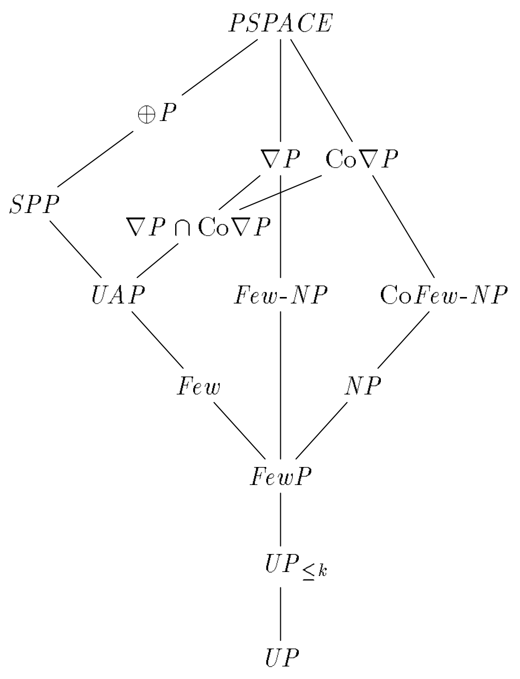


Figure 4.1: Inclusions around UAP

Theorem 4.12 *$Few \subseteq UAP$.*

Proof. Look at the proof of Theorem 4.8. There we explicitly computed the number of accepting computations of an NTM by a UAP-machine. Knowing the number of accepting computations, a UAP-machine can clearly compute the polynomial time computable predicate as given in the definition of Few. □

We summarize all the inclusions around UAP as derived so far in Figure 4.2.

Due to the tight links between UAP and ∇P it might be of special interest to compare relationships between ∇P and other complexity classes to results about UAP. The following table gives three results for ∇P [35] and the corresponding ones for UAP.

$1\text{-}NP \subseteq \nabla P$	$UP \subseteq UAP$
$\nabla P \subseteq \forall\oplus P$	$UAP \subseteq \oplus P$
$\nabla P \subseteq C_{=}P$	$UAP \subseteq SPP$

Here $1\text{-}NP$ is the subclass of $Few\text{-}NP$ with $p(n) = 1$ and $C_{=}P$ is the class of languages recognized by polynomial time NTM's that accept a word w iff the number of accepting computations coincides with some given polynomial time computable function on w [34]. For a definition of complexity classes defined by operators as $\forall\oplus P$ we refer to [36] and the subsequent subsection.

Finally, it remains to be shown that UAP is locally definable.

Theorem 4.13 *UAP is locally definable.*

Proof. We claim that $UAP = [F]P$, where $F = \{\textsc{And}!, \textsc{Or}!\}$. The proof follows directly from the definition of UAP. □

It seems to be likely that the inclusion $UAP \subseteq SPP$ is strict since UAP does not seem to be gap-definable and SPP does not seem to be locally definable.

4.3 Complexity Classes Defined by Operators

By Theorem 3.3 all classes characterizable by locally definable acceptance types as $(F)P$ are also characterizable by $[F']P$ for some F'. Hertrampf [19] showed that if a class $\mathcal{C}$ is characterizable as $\mathcal{C} = (F)P$, then $\exists\mathcal{C}$, $\forall\mathcal{C}$, $\oplus\mathcal{C}$, and $\mathrm{MOD}_k\mathcal{C}$ are, too. We will now prove that characterizations via $[F]P$ are also closed under these operations. Theorem 3.3 is not sufficient for this purpose, since, for example, $\forall\, UP$ is not covered by this. Moreover, we show closure of locally definable classes also under the operations $\exists!$ and $\forall!$ invented by Hemaspaandra, as well as under the operator MODZ_k which is defined analogously to the MOD_k operator in [19]. For the sake of completeness, we define all operators in the following.

Definition 4.14 Let $\mathcal{C}$ be a class of languages.

1. $L \in \exists\mathcal{C}$ iff there exists a language $A \in \mathcal{C}$ and a polynomial p such that

$$w \in L \iff \exists y\colon |y| \le p(|w|) \wedge \langle w, y\rangle \in A.$$

Here $\langle w, y\rangle$ denotes the pairing of w and y.

Analogous definitions exist for $\forall\mathcal{C}$ and $\mathrm{MOD}_k\mathcal{C}$. Clearly, $\forall\mathcal{C} = \text{Co-}\exists\text{Co-}\mathcal{C}$.

2. $L \in \exists!\mathcal{C}$ iff there exists a language $A \in \mathcal{C}$ and a polynomial p such that $w \in L$ iff for each w there is at most one y with $\langle w, y\rangle \in A$ *and*

$$w \in L \iff \exists y\colon |y| \le p(|w|) \wedge \langle w, y\rangle \in A.$$

   Analogous definitions hold for $\forall!\mathcal{C}$ and $\mathrm{MODZ}_k\,\mathcal{C}$ (see also 3.1.2). Again $\forall!\mathcal{C} = \text{Co-}\exists!\,\text{Co-}\mathcal{C}$.

The following theorem summarizes the announced closure under the various complexity theoretic operators as given in Definition 4.14.

Theorem 4.15 *If $\mathcal{C}$ is locally definable, then $\exists\mathcal{C}$, $\forall\mathcal{C}$, $\exists!\mathcal{C}$, $\forall!\mathcal{C}$, $\mathrm{MOD}_k\,\mathcal{C}$, and $\mathrm{MODZ}_k\,\mathcal{C}$ are locally definable, too.*

Proof. Let $\mathcal{C} = [\{f\}]P$. The following equalities hold:
1) $\exists\mathcal{C} = [\{\overline{f}, \textsc{Trans}^{\bar{0}\mapsto 0}_{\bar{1}\mapsto 1}, \textsc{Or}\}]P$, 2) $\forall\mathcal{C} = [\{\overline{f}, \textsc{Trans}^{\bar{0}\mapsto 0}_{\bar{1}\mapsto 1}, \textsc{And}\}]P$,
3) $\exists!\mathcal{C} = [\{\overline{f}, \textsc{Trans}^{\bar{0}\mapsto 0}_{\bar{1}\mapsto 1}, \textsc{Or!}\}]P$, 4) $\forall!\mathcal{C} = [\{\overline{f}, \textsc{Trans}^{\bar{0}\mapsto 0}_{\bar{1}\mapsto 1}, \textsc{And!}\}]P$,
5) $\mathrm{MOD}_k\,\mathcal{C} = [\{\overline{f}, \textsc{Trans}^{\bar{0}\mapsto \underline{0}}_{\bar{1}\mapsto \underline{1}}, \underline{\textsc{Mod}_k}, \textsc{Trans}^{\underline{0}\mapsto 0}_{\underline{1},\dots,\underline{k-1}\mapsto 1}\}]P$,
6) $\mathrm{MODZ}_k\,\mathcal{C} = [\{\overline{f}, \textsc{Trans}^{\bar{0}\mapsto \underline{\mathbf{0}}'}_{\bar{1}\mapsto \underline{1}}, \underline{\textsc{ModZ}_k}, \textsc{Trans}^{\underline{\mathbf{0}}'\mapsto 0}_{\underline{1},\dots,\underline{k-1}\mapsto 1}\}]P$.

We omit the fairly straightforward proofs. (Use the propagation principle.) □

4.4 Unambiguous Hierarchies

Definition 4.16 The levels of the *unambiguous alternation hierarchy* are defined as follows. The level $AU\Sigma^p_k$, $k \ge 1$, is the set of languages accepted by UAP-machines with at most $k \perp 1$ alternations between existential and universal configurations, starting with an existential configuration. The level $AU\Pi^p_k$ is defined analogously, just starting with a universal configuration. Especially, $AU\Sigma^p_0 = AU\Pi^p_0 = P$.

Hemaspaandra [unpublished] showed that these levels coincide with classes obtained from UP by iteratively applying unambiguous existential and universal polynomial length bounded quantification, so, e.g.,

$$AU\Sigma_2^p = \forall!\,\exists!\,P = \forall!\,UP \text{ and } AU\Pi_2^p = \exists!\,\forall!\,P.$$

In the case of normal existential and universal, polynomial length bounded quantifiers one gets the levels of the polynomial hierarchy, which coincide with the levels of the alternation hierarchy. For unambiguous computation, however, the oracle and alternation hierarchies do *not* seem to coincide. (In fact, there are relativized worlds in which $AU\Sigma_k^p \subsetneq U\Sigma_k^p$ [30].)

Definition 4.17 The levels of the *unambiguous polynomial hierarchy* are defined inductively as follows. $U\Sigma_0^p = U\Pi_0^p = U\Delta_0^p = P$ and for $k > 0$

$$U\Sigma_k^p = UP(U\Sigma_{k-1}^p), \qquad U\Pi_k^p = \text{Co-}U\Sigma_k^p, \qquad U\Delta_k^p = P(U\Sigma_{k-1}^p).$$

The unambiguous polynomial hierarchy (or *UP* oracle hierarchy) itself is defined as $UPH = \bigcup_{k\geq 0} U\Sigma_k^p$.

Hemaspaandra [unpublished] proved that the levels of the unambiguous alternation hierarchy coincide with levels of the so-called *unambiguous one-query oracle hierarchy.* Herein, the unambiguous one-query oracle hierarchy is defined in the same line as the unambiguous polynomial hierarchy with the additional restriction that the oracle may only be asked once. As a corollary to Theorem 4.15 we now can characterize all levels of the unambiguous one-query hierarchy in terms of locally definable acceptance types, since all levels of the unambiguous alternation hierarchy are obviously characterizable.

Corollary 4.18 *All levels of the unambiguous alternation hierarchy, which are the same as the levels of the unambiguous one-query oracle hierarchy, are locally definable.*

Theorem 4.19 *If $\mathcal{C}$ is locally definable, then $P^{\mathcal{C}}$, $UP^{\mathcal{C}}$, and $NP^{\mathcal{C}}$ are locally definable, too.*

Proof. Theorem 4.19 is a special case of the more general Theorem 5.8. For a direct proof, however, one can establish the following equalities.

$$\begin{aligned} P^{\mathcal{C}} &= [\{\overline{f}, \textsc{Select}\}]P, \\ UP^{\mathcal{C}} &= [\{\overline{f}, \textsc{Select}, \textsc{Or}*\}]P, \text{and} \\ NP^{\mathcal{C}} &= [\{\overline{f}, \textsc{Select}, \textsc{Or}\}]P, \end{aligned}$$

where $\mathcal{C} = [\{f\}]P$. Herein, $\textsc{Or}*\colon \{0, 1, *\}^2 \mapsto \{0, 1, *\}$ is defined as follows.

$$\textsc{Or}*(x, y) := \begin{cases} * & \text{if } x = y = 1 \text{ or } x = * \text{ or } y = *, \text{ and } x, y \neq \perp, \\ \textsc{Or}(x, y) & \text{otherwise.} \end{cases}$$

□

By Theorem 4.19 we directly get characterizations of the polynomial hierarchy and of the UP oracle hierarchy in terms of locally definable acceptance types.

Corollary 4.20 *All levels of the polynomial hierarchy and the unambiguous polynomial hierarchy UPH are locally definable.*

5 Access to Unambiguous Computations

The computation of a TM with access to an oracle can be interpreted in different ways. One possibility is to interpret the oracle as a database, a second one to see accesses to the oracle as subroutine calls or even remote procedure calls on some different computer. In the database case, all answers "are there," while in the subroutine view answers are computed if and only if they are queried.

The class P^{UP}, for example, is obtained in a database like way. For each language $L \in P^{UP}$, there is an oracle TM (OTM) M and an oracle $A \in UP$, such that $L = L(M, A)$. The set A is the database and $A \in UP$ means that all entries in the database are computable by an unambiguous polynomial time bounded NTM.

Cai, Hemachandra, and Vyskoč [10] introduced the class $P^{\mathcal{UP}}$ that can be interpreted in a subroutine access way. Here $L \in P^{\mathcal{UP}}$ iff $L = L(M, A)$, and A needs not necessarily be accepted by an unambiguous TM M_A; it suffices that M_A is unambiguous *only on all queries posted by M*. The overall computation (computation of M together with subroutine computations of M_A) remains unambiguous. Cai, Hemachandra, and Vyskoč propose that this notion of access to unambiguous computation (called *guarded access*) is more natural than the database view.

5.1 Guarded Access

In this subsection we will show that guarded access is characterizable by locally definable acceptance types. Our contribution, however, goes deeper than this. We provide new insight into the problem of how to access unambiguous computations. Since locally definable acceptance types do not use oracles or other global concepts like number of accepting paths, they are a good means for an "overall computation" model. Locally definable acceptance types show which concepts can be realized by local conditions in a computation model. The characterization of guarded oracle access supports the conjecture of Cai, Hemachandra, and Vyskoč that guarded access to unambiguous computation is a natural notion. Surprisingly, however, the unrestricted ("database") access is characterizable by locally definable acceptance types, too.

In the remainder of this subsection we will formalize these ideas and state the results. To formalize guarded access, Cai, Hemachandra, and Vyskoč used the notion of *promise problems* introduced by Even, Selman, and Yacobi [12].

Definition 5.1 A *promise problem* is a pair of predicates (Q, R), where Q is called the *promise* and R the *property.*

In this paper we are interested in promise classes related to unambiguous computations. This leads to *promise UP.*

Definition 5.2 Let N_1, N_2, ... be a standard enumeration of nondeterministic polynomial-time Turing machines, $Q_i := \{ w \mid Accept(N_i, w) \leq 1 \}$, and $R_i :=$

$\{w \mid Accept(N_i, w) \geq 1\}$. Then $\mathcal{UP}$ ("promise UP") is the following class of promise problems: $\{\,(Q_i,\, R_i) \mid i \geq 1\,\}$.

Definition 5.3 Let $\mathcal{A} = (Q,\, R)$ be a promise problem. We say that $L \in P^{\mathcal{A}}$ if there is a deterministic polynomial-time oracle Turing machine M such that:

1. $L = L(M, R)$, and
2. for every string x, in the computation of $M^R(x)$ every query z made to the oracle satisfies $z \in Q$.

$UP^{\mathcal{A}}$ and $NP^{\mathcal{A}}$ are defined analogously. The following theorem shows that oracle access to $\mathcal{UP}$ is characterizable by locally definable acceptance types.

Theorem 5.4 *$P^{\mathcal{UP}}$, $UP^{\mathcal{UP}}$, and $NP^{\mathcal{UP}}$ are locally definable.*

We omit the proof of Theorem 5.4, because it is an immediate consequence of a more general result (Theorem 5.8). For the proofs in this section we need the following technical notion:

Definition 5.5 Let F be a locally definable acceptance type. We write $(A, B) \in \langle F \rangle P$ iff there is an NTM M and an evaluation scheme as in Definition 2.2, but

- the root of M evaluates to one of $(0,0)$, $(0,1)$, $(1,0)$, or $(1,1)$ for all inputs,
- at least one configuration of each computation tree of M must be labeled by a function from F,
- $w \in A$ iff the root of M evaluates either to $(1,0)$ or $(1,1)$ on input w,
- $w \in B$ iff the root of M evaluates either to $(0,1)$ or $(1,1)$ on input w.

By $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$ we understand comfortable names of integers.

For $\mathcal{UP}$ we have the following characterization. The proof (by usual techniques) is omitted.

Lemma 5.6 *$\mathcal{UP} = \langle F \rangle P$, where $F = \{\text{Or}*, \text{Trans}_{1\mapsto(1,1)}^{0\mapsto(1,0),*\mapsto(0,1)}\}$.*

In order to prove the next theorem we need a technical lemma that states that the closure of a class $\langle F \rangle P$ under polynomial time reductions stays the same if we add $\{(0,0),(0,1),(1,0),(1,1)\}$ to the base set of F. For a class $\langle F \rangle P$ itself this claim is in general false. For example, $(\emptyset,\emptyset) \notin \mathcal{UP}$, but $(\emptyset,\emptyset) \in \langle F' \rangle P$, where $F' = \{\text{Or}*, \text{Trans}_{1\mapsto(1,1),}^{0\mapsto(1,0),*\mapsto(0,1)}, id\}$, where *id* simply denotes the identity over base set $\{(0,0),(0,1),(1,0),(1,1)\}$. Here the base set of F is $\{0,1,*,(1,0),(0,1),(1,1)\}$ and that of F' is $\{0,1,*,(0,0),(1,0),(0,1),(1,1)\}$. The base set of $F = \{\text{Or}*, \text{Trans}_{1\mapsto(1,1)}^{0\mapsto(1,0),*\mapsto(0,1)}\}$ does not contain $(0,0)$ and thus no computation tree of an F-machine contains nodes with value $(0,0)$.

Lemma 5.7 *Define $id\colon \{(0,0),(0,1),(1,0),(1,1)\} \to \{(0,0),(0,1),(1,0),(1,1)\}$, $x \mapsto x$ and $F' = F \cup \{id\}$, where F is a locally definable acceptance type. Then $P^{\langle F \rangle P} = P^{\langle F' \rangle P}$, $UP^{\langle F \rangle P} = UP^{\langle F' \rangle P}$, and $NP^{\langle F \rangle P} = NP^{\langle F' \rangle P}$.*

Proof. The inclusions from left to right are trivial. For the reverse direction assume that $(A',B') \in \langle F' \rangle P$ and $L \in P^{(A',B')}$. Let S be the base set of F and define

$$\tau(w) = \begin{cases} (0,0) & \text{if } w \notin A' \text{ and } w \notin B', \\ (0,1) & \text{if } w \notin A' \text{ and } w \in B', \\ (1,0) & \text{if } w \in A' \text{ and } w \notin B', \\ (1,1) & \text{if } w \in A' \text{ and } w \in B'. \end{cases}$$

If S is empty, the claim is trivially fulfilled, since then $< F' > P = P$. Otherwise, choose some fixed $s \in S$. In what follows we construct an oracle (A,B) that only provides answers from S. For queries w where the answer of the original oracle (A',B') is contained in S, oracle (A,B) behaves in exactly the same way as (A',B'). If for a query w the original oracle (A',B') should provide an answer not in S, then (A,B) simply answers s on query w. In this case, the answer of (A',B') can also be computed deterministically by the querying machine. Formally, define the sets A and B as follows. If $\tau(w) \in S$ then $w \in A$ iff $w \in A'$ and $w \in B$ iff $w \in B'$. Otherwise, $w \in A$ iff $s \in \{(1,0),(1,1)\}$ and $w \in B$ iff $s \in \{(0,0),(0,1)\}$.

Then $L \in P^{(A,B)}$ by the following simulation: An oracle TM M with access to (A,B) simulates an oracle TM M' that witnesses $L \in P^{(A',B')}$ as follows. If M' asks the question w to the oracle, then M first checks deterministically, whether $\tau(w) \notin S$. This is possible since a path in the computation of an $\langle F'\rangle$-machine that results in a constant not in S leads deterministically to a leaf, because all inner nodes on such a path have to be labeled with id. So to determine whether $\tau(w) \in S$, M starts to simulate an $\langle F'\rangle$-machine that computes (A',B') on input w. It follows the computation path of M' as long as nodes are labeled by id, if the next node is labeled by a function from F then it evaluates to some value from S, so $\tau(w) \in S$. If the next node is a leaf, then $\tau(w)$ is the label of the leaf and M checks whether this label is in S. In this way M gets the answer of the oracle query. If, however, $\tau(w) \in S$, the oracle (A,B) yields the same answer as (A',B') does. By construction and the above mentioned deterministic checkability, $(A,B) \in \langle F\rangle P$. The proof for $UP^{\langle F\rangle P}$ and $NP^{\langle F\rangle P}$ is identical. □

Theorem 5.8 *Let F be a locally definable acceptance type. Then $P^{\langle F\rangle P}$, $UP^{\langle F\rangle P}$, and $NP^{\langle F\rangle P}$ are locally definable.*

Proof. It can be shown by the same techniques as in Theorem 3.2 that there is a binary function f such that $\langle F\rangle P = \langle\{f\}\rangle P$. Let $F_1 = \{\overline{f}, \text{Choose}\}$, $F_2 = \{\overline{f}, \text{Choose}, \text{Or}*\}$, and $F_3 = \{\overline{f}, \text{Choose}, \text{Or}\}$. We show $[F_1]P = P^{\langle F\rangle P}$, $[F_2]P = UP^{\langle F\rangle P}$, and $[F_3]P = NP^{\langle F\rangle P}$. Choose is defined as

$$\text{Choose}(x,y,z) := \begin{cases} x & \text{if } z = \overline{(1,1)},\ y \neq \perp, \\ y & \text{if } z = \overline{(1,0)},\ x \neq \perp, \\ * & \text{if } z = \overline{(0,0)} \text{ or } z = \overline{(0,1)},\ x \neq \perp,\ y \neq \perp, \\ \perp & \text{otherwise.} \end{cases}$$

"$\subseteq$" The propagation principle tells us that a computation tree of an F_i-machine M consists of nodes labeled Choose with an $\overline{f}$-subtree on the right.

The leaves outside of $\overline{f}$-subtrees are labeled 0 or 1. In the case of F_2 (resp. F_3) there may be also OR∗'s (resp. OR's) outside of $\overline{f}$-subtrees. Also due to the propagation principle right successors of CHOOSE-nodes have always value $\overline{(0,0)}$, $\overline{(0,1)}$, $\overline{(1,0)}$ or $\overline{(1,1)}$. Let us call in the case of $\overline{(1,0)}$ the left successor "forbidden" and in the case of $\overline{(1,1)}$ the middle successor forbidden. For the remaining cases both the left and middle successor are forbidden. By induction it can be seen that the result of an F_i-machine is 1, iff there exists a path from the root to some leaf in the computation tree of M, such that all nodes on the path have value 1 and there are no forbidden nodes among them. In the case of F_1 and F_2 there can be at most one such path (for F_2 we again have to apply the propagation principle).

For F_1 a DTM can find this path by avoiding forbidden nodes as follows: By queries to a suitable oracle $(A, B) \in \langle F \cup \{ id \} \rangle P$, where id is the identity on $\{ (0,0), (0,1), (1,0), (1,1) \}$, the DTM finds out the value of the correct successor of a CHOOSE-node on the path and thus knows which successor is forbidden. The oracle is defined as follows: $p \in A$, iff p encodes a computation path of M starting at the root of the computation tree and ending on top of an $\overline{f}$-subtree. B consists of those $p \in A$, such that the encoded path ends at a top of an $\overline{f}$-tree with value $\overline{(1,1)}$. Note that in this way no promise breaking questions are asked and $(A, B) \in \langle F \cup \{ id \} \rangle P$. (Check deterministically whether p encodes such a path and if yes simulate the $\overline{f}$-subtree.) By Lemma 5.7 we also know $P^{(A,B)} \subseteq P^{\langle F \rangle P}$.

For F_2 and F_3 an NTM can guess the path and verify with the help of oracle (A, B) that no forbidden nodes are on the path. This has to be done beginning from the root to the leaf in order to not ask promise breaking questions. In the case of F_2 the NTM works unambiguously, since there is at most one such path.

"$\supseteq$" Let $L \in P^{\langle F \rangle P}$ (resp. $L \in UP^{\langle F \rangle P}$, $L \in NP^{\langle F \rangle P}$) and M be a witnessing OTM with oracle $(A, B) \in \langle F \rangle P$, i.e., $L = L(M, (A, B))$. M can be simulated by an F_i-machine as follows: Nondeterministic steps of M are simulated by OR- (resp. OR∗-steps), oracle queries by a CHOOSE-configuration, where the rightmost (third) successor is an $\overline{f}$-subcomputation, computing $\overline{(0,1)}$ or $\overline{(1,1)}$ for a positive and $\overline{(0,0)}$ or $\overline{(1,0)}$ for a negative oracle answer. The leftmost (first) successor is the root of a subcomputation simulating M after a positive oracle answer, the middle successor after a negative answer. Let us call the successor corresponding to the correct answer of the oracle the correct one. Paths in the computation tree of the F_i-machine traversing only correct successors of CHOOSE-nodes correspond to computation paths of M. On such paths no promise breaking questions are asked and the values of $\overline{f}$-subtrees adjacent to this path are $\overline{(1,0)}$ or $\overline{(1,1)}$. This means that no values $*$ are created and thus the leaf values at the end of those paths reach the root (possibly composed by OR or OR$*$ nodes). The result is 1, if there exists an accepting path of M, and 0 otherwise.

In the whole computation tree no $\perp$ is generated; $*$ is generated only in parts of the tree corresponding to subcomputations after wrong oracle answers. These $*$'s are "absorbed" by CHOOSE-nodes and never affect the result. □

Hemaspaandra and Rothe [17] introduced the unambiguous promise hierarchy:

Definition 5.9 The *unambiguous promise hierarchy* consists of the classes $\mathcal{U}\Sigma_k^p$, $\mathcal{U}\Pi_k^p$, and $\mathcal{U}\Delta_k^p$ for $k \geq 0$. These classes are defined as $\mathcal{U}\Sigma_0^p = P$ and $L \in \mathcal{U}\Sigma_k^p$ if and only if there are nondeterministic, polynomial time bounded oracle TM's $N_1, \ldots, N_k$ such that $L = L(N_1, A_1)$ and $A_i = L(N_{i+1}, A_{i+1})$, for $i < k$ and $A_k = \emptyset$. Let Q_1 be the set of queries asked by N_1 with oracle A_1 on all possible inputs and Q_i the set of queries asked by N_i with oracle A_i on all inputs in Q_{i-1}. Then N_i with oracle A_i must have at most one accepting path for all inputs in Q_{i-1}, where $Q_0 = \Sigma^*$. The classes $\mathcal{U}\Pi_k^p$ and $\mathcal{U}\Delta_k^p$ are defined similarly.

Generalizing the methods of this subsection it can be shown that all levels of the unambiguous promise hierarchy are locally definable.

5.2 Robustly unambiguous computations

An OTM has a property *robustly* if it has the property relative to every oracle. The most important property is, of course, accepting some fixed language, which was investigated by Schöning [31]. We are interested in *robustly unambiguous* OTM's—OTM's that behave unambiguously for every oracle. Hartmanis and Hemachandra showed that if a polynomial time bounded OTM is unambiguous for all oracles, then it accepts a language in $P^{NP \oplus A}$ relative to all oracles A.

For a class $\mathcal{L}$ we denote by $\mathcal{UP}^{\mathcal{L}}$ the class of languages recognized by robustly unambiguous OTM's with an oracle in $\mathcal{L}$, i.e.,

$$\mathcal{UP}^{\mathcal{L}} = \{\, L(M,A) \mid A \in \mathcal{L},\ M \text{ is robustly unambiguous and polynomial time bounded} \,\}.$$

Don't confuse $\mathcal{UP}^{\mathcal{L}}$ with a promise class that occurs only as an oracle. We show that $\mathcal{UP}^{\mathcal{L}}$ is locally definable if only $\mathcal{L}$ itself is locally definable. The property of being robustly unambiguous can thus be also expressed by local conditions on the computation tree.

Theorem 5.10 *$\mathcal{UP}^{[F]P}$ is locally definable.*

Proof. Let $[F]P = [\{f\}]P$ according to Theorem 3.2. We show that $[\{\textsc{Select}, \textsc{Or!}, \overline{f}\}]P = \mathcal{UP}^{[F]P}$.

"$\subseteq$" Let $L \in [\{\textsc{Select}, \textsc{Or!}, \overline{f}\}]P$ and M be a $\{\textsc{Select}, \textsc{Or!}, \overline{f}\}$-machine witnessing this fact. By the propagation principle a computation tree of M contains Or! and Select-nodes. The rightmost children of all Select-nodes are $\overline{f}$-subtrees that evaluate to $\overline{1}$ or $\overline{0}$. The children of all Or!-nodes evaluate to 0 or 1, and at most one child evaluates to 1.

We construct an oracle A as follows: A contains computation paths p starting with the initial configuration on input w such that p ends at a Select-node whose rightmost child evaluates to $\overline{1}$. With the help of A we construct an OTM M' such that $L(M', A) = L$. The machine M' simulates M by traversing a path of its computation tree starting at M's root. At a configuration labeled Or!, M' guesses one child and continues its simulation there. At a Select-node M' asks its oracle whether the rightmost child of this configuration evaluates to $\overline{1}$. If yes,

M' continues at the leftmost child, otherwise at the middle child. Reaching a leaf, M' accepts if the leaf is labeled 1. Otherwise M' rejects. Obviously, M' has an accepting computation iff M accepts, i.e., its root evaluates to 1.

We show furthermore that M' works robustly unambiguous: There is at most one accepting path for M' with some arbitrary oracle B since *every* Or!-node has at most one child labeled with 1. (Observe that Select is strict with respect to $\perp$.) To conclude $[\{\text{Select}, \text{Or}!, \overline{f}\}]P \subseteq \mathcal{UP}^{[F]P}$ we have to prove $A \in [F]P$, which is quite obvious.

"$\supseteq$" Now let $L \in \mathcal{UP}^{[F]P}$. Let $L = L(M, A)$, where M is a robustly unambiguous OTM and $A \in [F]P$. We furthermore assume that on each path and for every oracle B the machine $M(B)$ asks every question to the oracle *at most once.* This is in fact not a restriction, since M can store all questions and answers and check whether it asked a question before and if it did use the stored answer to decide how to proceed instead of using an oracle query.

A $\{\overline{f}, \text{Select}, \text{Or}!\}$-machine simulates M as in Theorem 4.19. The difference is that we have Or!-nodes instead of Or$*$-nodes. We can show that nowhere in the computation tree of M' an Or!-node receives two 1's, which means that everywhere Or! and Or$*$ compute the same results. Theorem 4.19 then shows that the simulation is correct. Or!-nodes never receive two 1's since otherwise there would be an oracle B relative to which M does not work unambiguously. We can construct B by looking at the path that leads to the Or!-node with two 1-children. This path corresponds to a possible path of M with the suitable oracle B, which is well-defined since M asks every query at most once. $\square$

6 Conclusion

We introduced the complexity class *unambiguous alternating polynomial time,* which combines the well-known concepts of alternation and unambiguity. As it turned out, this class contains *Few* and is contained in *SPP*. We also investigated how several unambiguous polynomial hierarchies can be defined.

All these and many more classes were analyzed within the framework of locally definable acceptance types, which are part of a whole spectrum of acceptance mechanisms defined via computation trees of nondeterministic polynomial time machines (see [20] for a survey). We can roughly distinguish three mechanisms: predicate classes, where the acceptance condition can depend on the complete tree [5]; leaf languages, where the acceptance condition only depends on the leaf word [7, 21]; and locally definable acceptance types, where the acceptance condition only depends on local, k-valued functions in the tree [18, 19]. Clearly, the second and the third mechanisms are special cases of the first one.

In all cases, however, in principle there are two possible agreements with respect to the acceptance mechanism. Either one only allows answers of type "yes" or "no," or one further admits the possibility that there might be "forbidden" values. Borchert studied both cases for predicate classes [5], Hertrampf studied the yes/no case for locally definable acceptance types [18, 19], whereas we studied the yes/no/forbidden case for them, thus obtaining strong relations to unambiguous computation. For leaf languages only the yes/no case was studied [21], whereas the yes/no/forbidden case still remains open there. On the other hand, note that the concept of leaf languages was also applied to logarithmic space and NC^1 [23].

As a whole, this paper provides an analysis of the power of locally definable acceptance types. It inspired and is complemented by further recent work on unambiguous computation [17, 26, 30].

Acknowledgment We are grateful to Uli Hertrampf for several fruitful remarks on the conference version of this paper and to the anonymous referees for numerous insightful comments.

References

[1] E. W. Allender. The complexity of sparse sets in P. In *Proceedings of the 1st IEEE Symposium on Structure in Complexity*, pages 1–11, 1986.

[2] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity Theory I.* Springer, 1988.

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

[4] R. Beigel, J. Gill, and U. Hertrampf. Counting classes: Threshold, parity, mods, and fewness. In C. Choffrut and T. Lengauer, editors, *Proceedings of the 7th Symposium on Theoretical Aspects of Computer Science*, number 415 in Lecture Notes in Computer Science, pages 49–57. Springer-Verlag, 1990.

[5] B. Borchert. Predicate classes and promise classes. In *Proceedings of the 9th IEEE Symposium on Structure in Complexity*, pages 235–241, 1994.

[6] D. P. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity.* Prentice Hall, 1994.

[7] Daniel P. Bovet, Pierluigi Crescenzi, and Riccardo Silvestri. A uniform approach to define complexity classes. *Theoretical Computer Science*, 104(2):263–283, 1992.

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

[9] J. Cai and L. A. Hemachandra. On the power of parity polynomial time. *Math. Systems Theory*, 23:95–106, 1990.

[10] J.-Y. Cai, L. Hemachandra, and J. Vyskoč. Promise problems and access to unambiguous computation. In I. Havel, editor, *Proceedings of the 17th Conference on Mathematical Foundations of Computer Science*, number 629 in Lecture Notes in Computer Science, pages 162–171, Prague, Czechoslavakia, August 1992. Springer-Verlag.

[11] A. K. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *Journal of the ACM*, 28:114–133, 1981.

[12] S. Even, A. Selman, and Y. Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61(2):159–173, 1984.

[13] S. A. Fenner, L. J. Fortnow, and S. A. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48:116–148, 1994.

[14] L. Goldschlager and I. Parberry. On the construction of parallel computers from various bases of boolean functions. *Theoretical Computer Science*, 43:43–58, 1986.

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

[16] J. Hartmanis and L. A. Hemachandra. Complexity classes without machines: On complete languages for UP. *Theoretical Computer Science*, 58:129–142, 1988.

[17] L. A. Hemaspaandra and J. Rothe. Unambiguous computation: Boolean hierarchies and turing-complete sets. Technical Report 483, University of Rochester, 1994.

[18] U. Hertrampf. Locally definable acceptance types—the three valued case. In I. Simon, editor, *Proceedings of the 1st Symposium on Latin American*

Theoretical Informatics, number 583 in Lecture Notes in Computer Science, pages 262–271, São Paulo, Brazil, April 1992. Springer-Verlag.

[19] U. Hertrampf. Locally definable acceptance types for polynomial time machines. In *Proceedings of the 9th Symposium on Theoretical Aspects of Computer Science*, number 577 in Lecture Notes in Computer Science, pages 199–207. Springer-Verlag, 1992.

[20] U. Hertrampf. Complexity classes defined via k-valued functions. In *Proceedings of the 9th IEEE Symposium on Structure in Complexity*, pages 224–234, 1994.

[21] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. W. Wagner. On the power of polynomial time bit-reductions. In *Proceedings of the 8th IEEE Symposium on Structure in Complexity*, pages 200–207, 1993.

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

[23] B. Jenner, P. McKenzie, and D. Thérien. Logspace and logtime leaf languages. In *Proceedings of the 9th IEEE Symposium on Structure in Complexity*, pages 242–254, 1994.

[24] D. S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Algorithms and Complexity*, volume A of *Handbook of Theoretical Computer Science*, chapter 2, pages 67–161. Elsevier, 1990.

[25] K.-J. Lange. Unambiguity of circuits. *Theoretical Computer Science*, 107:77–94, 1993.

[26] K.-J. Lange and P. Rossmanith. Unambiguous polynomial hierarchies and exponential size. In *Proceedings of the 9th IEEE Symposium on Structure in Complexity*, pages 106–115, 1994.

[27] R. Niedermeier and P. Rossmanith. Extended locally definable acceptance types. In P. Enjalbert, A. Finkel, and K. W. Wagner, editors, *Proceedings of the 10th Symposium on Theoretical Aspects of Computer Science*, number 665 in Lecture Notes in Computer Science, pages 473–483. Springer-Verlag, 1993.

[28] R. Niedermeier and P. Rossmanith. Unambiguous auxiliary pushdown automata and semi-unbounded fan-in circuits. *Information and Computation*, 118(2):227–245, May 1995.

[29] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[30] P. Rossmanith. Separating the unambiguous polynomial time hierarchy by oracles. Submitted for publication, November 1995.

[31] U. Schöning. Robust algorithms: a different approach to oracles. *Theoretical Computer Science*, 40:57–66, 1985.

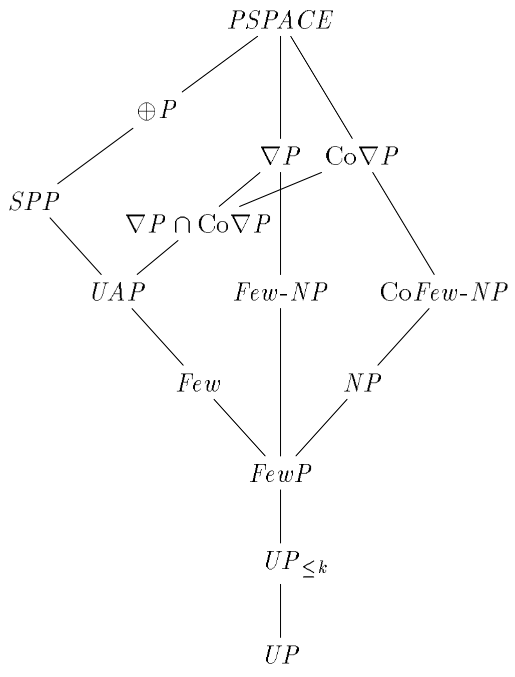
[32] A. L. Selman. A survey of one-way functions in complexity theory. *Mathematical Systems Theory*, 25(3):203–221, 1992.

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

[34] K. W. Wagner. Compact descriptions and the counting polynomial time hierarchy. *Acta Informatica*, 23:325–356, 1986.

[35] K. W. Wagner. Alternating machines using partially defined “AND” and “OR”. Technical Report 39, Institut für Informatik, Universität Würzburg, January 1992.

[36] K. W. Wagner and G. Wechsung. *Computational complexity.* Reidel Verlag, Dordrecht and VEB Deutscher Verlag der Wissenschaften, Berlin, 1986.


$PSPACE$
$\oplus P$
∇P
Co∇P
SPP
$\nabla P \cap$ Co∇P
UAP
Few-NP
CoFew-NP
Few
NP
$FewP$
$UP_{\leq k}$
UP