Extended Locally Definable Acceptance Types*

(Extend Abstract, Draft Version)

Rolf Niedermeier and Peter Rossmanith
Institut für Informatik, Technische Universität München
Arcisstr. 21, D-8000 München 2, Fed. Rep. of Germany
Topic: Computational Complexity

Abstract

Hertrampf's locally definable acceptance types showed that many complexity classes can be defined in terms of polynomially 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 *extended* locally definable acceptance types as a generalization of Hertrampf's concept in order to formally capture exactly all classes that are definable by simple local conditions in an intuitive sense. Among the new characterizable classes are UP and $\mathrm{MODZ}_k\mathrm{P}$. We show 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.

1 Introduction

Many classes of central interest in structural complexity theory are defined via polynomially time bounded Turing machines (TM's). A word w, for example, belongs to the language of a nondeterministic TM M, if M has on input w at least one accepting path. Many other acceptance mechanisms exist that are defined via numbers of accepting paths. Recently Hertrampf introduced an evaluation scheme for nondeterministic TM's (NTM's) which 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 [11]. Using only OR as local functions yields NP, while AND yields *Co*NP. Allowing both OR and AND results in the class PSPACE. This simple concept, called *locally definable acceptance type*, is capable to characterize many more 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 [10, 11]).

There exists, however, a large field of interesting classes which do not seem to be characterizable in terms of locally definable acceptance types, though they can intuitively be described by simple local conditions, among them the class UP defined via unambiguous TM's [16]. Unambiguous complexity classes are closely connected to the existence of one-way functions and public-key crypto systems [8, 14]. 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 [3]. The approach to build hierarchies has proven to be very 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 extend 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 a lot of reasonable possibilities for oracle access to unambiguous computations. Cai, Hemachandra, and Vyskoč [3] regard guarded and fault-tolerant access. A uniform concept that

*Research supported by DFG-SFB 0342 TP A4 "KLARA"

is able to characterize all of the classes derived in these ways and the corresponding concepts seems to be very helpful in further investigations in this area. We extend Hertrampf's locally definable acceptance types to *extended locally definable acceptance types*. We show that extended locally definable acceptance types can characterize at least as many classes as locally definable acceptance types. Moreover, the class of characterizable classes is closed under complement, polynomially bounded existential and universal quantifications and all other closure properties of [11]. In addition it is also closed under some closure properties which were introduced by Hemachandra. These include $\exists!$ and $\forall!$. The new characterizable classes include UP, *Co*UP, $\mathrm{UP}_{\leq k}$, $\mathrm{MODZ}_k\mathrm{P}$ 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 proves are not harder. We claim that extended locally definable acceptance types capture exactly the intuitive notion of "being definable by simple local conditions".

The remainder of the paper is organized as follows: In the next section we start defining extended locally definable acceptance types and explain the difference to Hertrampf's original definition. In the third section we provide characterizations of UP, $\mathrm{MODZ}_k\mathrm{P}$, UAP and demonstrate the closure of extended locally definable acceptance types under several complexity theoretic operators and relativization. Finally, in Section 4 we investigate access to unambiguous computations within the framework of extended locally definable acceptance types and, in particular, study guarded access to unambiguous computations.

We assume the reader to be familiar with standard notations of computational complexity theory. In particular, we denote by $L(M)$ the language accepted by a TM M and by $L(M, A)$ the language accepted by an oracle TM with access to oracle A. $||M(w)||$ denotes the number of accepting paths of machine M on input w. All further special notations will be introduced before they are used. Due to lack of space some results appear without proof. Mainly proofs are omitted for results where similar techniques as in Hertrampf's work [11] apply. A full paper containing all proofs is available.

2 Extended locally definable acceptance types

Goldschlager and Parberry [7] introduced so-called *extended Turing machines* in order to generalize the concept of alternating Turing machines [4]. They studied the power of nondeterministic, time-bounded Turing machines with an altered manner of acceptance. Instead of only labeling the states (and thus in a straightforward way also the configurations) of the machine just with AND (universal), OR (existential), NOT (negating), ACCEPT or REJECT, as it is done in alternating Turing machines, they allowed states to be labelled with a larger range of functions, in particular, the two-input Boolean functions. Recently, Hertrampf [11] further generalized this concept by permitting any functions from *m-valued logic* for some fixed integer m rather than only from Boolean logic as Goldschlager and Parberry did. In [10], Hertrampf gave a complete classification of the case when three-valued logic is allowed. He showed that with *locally definable acceptance types*, as he called the above concept, it is possible to characterize several important complexity classes (e.g. the whole polynomial time hierarchy [15]) in a unified framework.

In this paper we further extend Hertrampf's definition [11] of locally definable acceptance types for polynomial time machines in order to characterize even more complexity classes in a unified framework (e.g. UP [16] as a simple case) than Hertrampf did. We begin with defining the basic notions of this and Hertrampf's work.

Definition 1 Let m be an integer, $m \geq 2$. A *function from m-valued logic* is a function from $\{0, \ldots, m-1\}^r$ into $\{0, \ldots, m-1\}$ for some natural number r greater than 0.

The set $\{0, \ldots, m-1\}$ is called the *base set* for the domain and codamain of f. Clearly, the restriction to natural numbers in the above definition only has been done for reasons of convenience. Subsequently we will make extensively use of the fact that symbols not equal to numbers can be encoded as those in a straightforward way. In addition, we fix that always an extra value $\perp$ is available for m-valued functions, which is not explicitly mentioned as a member of the base set. (Nevertheless we speak of m-valued and not of $(m+1)$-valued functions.) Whenever a function of m-valued logic has an argument evaluating to $\perp$, then the function evaluates to $\perp$. The meaning of $\perp$ can be understood as "undefined". We say that all functions from m-valued logic are "strict" with respect to $\perp$.

Definition 2 An *(extended) 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. An (extended) locally definable acceptance type is called *finite* if F is a finite set. An *F-machine* M is a polynomially time bounded NTM, where each configuration with r successors is labeled by a function $f \in F \cup \{id\}$ with arity r, dependent only on the state of M. (Here id denotes the identity on the base set of F.) Leaves in the computation tree are labeled with an integer from $\{0, \dots, m-1\}$ dependent on the state. On input w to nodes in the computation tree *values* are assigned as follows: The value of a leaf is the integer, the leaf is labeled with. An inner node c, labeled with f and having successors $c_1, \dots, c_r$ with values $v_1, \dots, v_r$, has value $f(v_1, \dots, v_r)$. The value of the root of the computation tree is the *result* of M.

Definition 3 A language L is a member of the complexity class $[F]$P, iff there is an F-machine M such that the result of M on input w is 1 if $w \in L$ and 0 if $w \notin L$.

For the ease of notation we call a class of languages $\mathcal{C}$ *locally definable* iff $\mathcal{C} = [F]$P for some extended locally definable acceptance type F. To compare this to Hertrampf's locally definable acceptance types, we now give his definition. There seems to be hardly any difference. However, in the case of extended locally definable acceptance types, 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.

Definition 4 [11] A language L is a member of (F)P, iff there is an F-machine M such that the result of M on input w is 1 if $w \in L$ and the result is different from 1 if $w \notin L$.

Our definition seems to be a natural extension of Goldschlager's and Parberry's definition [7] for two-valued logic. In [7] a word is accepted iff the root of the computation tree evaluates to 1. If in m-valued logic the codomain may have m different values, why shouldn't we nevertheless demand that the root of the computation tree comes to a clear answer, i.e., saying "yes" (resp. accepting) or saying "no" (resp. rejecting a particular input word). By way of contrast, Hertrampf defines his locally definable acceptance types to accept iff the root evaluates to 1 and to reject if it evaluates to $0, 2, 3, \dots, m-1$. Moreover, the such defined *extended* locally definable acceptance types will prove to be as powerful as Hertrampf's locally definable acceptance types and, in addition, provide characterizations of several important complexity classes in the realm of unambiguous computations which seem not to be amenable by Hertrampf's definition. Naturally the question arises whether extended locally definable acceptance types for polynomial time machines really are more powerful than locally definable acceptance types, i.e., does there exist a class $\mathcal{C}$ which is characterizable by extended locally definable acceptance types, but not by locally definable acceptance types? This question is tightly connected to a major open problem in complexity theory [9]. Let $L_F = \{ \langle M, w \rangle \mid M \text{ is a } F\text{-TM and } w \in L(M) \}$. L_F is complete for (F)P, so each class characterizable by locally definable acceptance types has complete languages. We will see (Theorem 11) that UP is characterizable in terms of extended locally definable acceptance types. As a consequence, the following theorem gives strong evidence that extended locally definable acceptance types are a "proper extension".

Theorem 5 *Extended locally definable acceptance types are more powerful then locally definable acceptance types, unless* UP *has complete sets.*

Though we do not know, whether UP has complete sets (there are both positive and negative relativizations, see [9]), we cannot hope to find a characterization of UP in terms of a locally definable acceptance type, unless this question is solved. On the other hand, extended locally definable acceptance types (like locally definable acceptance types) for polynomial time machines are at most as powerful as PSPACE.

Proposition 6 *For all extended locally definable acceptance types F it holds that* $[F]\mathrm{P} \subseteq \mathrm{PSPACE}$.

Proof. The claim is a simple consequence of the fact that a polynomial space bounded machine M always can make a depth first search on the computation tree of a polynomial time machine. Thus it may compute the values of all nodes of the computation tree of an F-machine. □

In order to simplify the presentation of our results we make several agreements for the rest of the paper.

We will use extended locally definable acceptance types where different functions may have different base sets. For example, let $F = \{f_1, f_2\}$ and $f_1 : X_1^{k_1} \to X_1$ 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 $(1 \leq i \leq 2)$.

In Definition 7 we summarize several functions which will be used in the context with extended locally definable acceptance types. Most of the proofs in the paper can be done by making use of these "standard" functions.

Definition 7 Let $m \geq 2$ be a positive integer, let $a, b \in \{0, \ldots, m-1\}$ and let $A, B \subseteq \{0, \ldots, m-1\}$. Then we have the following functions from m-valued logic:

1. Transformation functions: $\textsc{Trans}^{A\mapsto a}_{B\mapsto b} : \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$, where

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

2. Boolean functions: $\textsc{Or}$, $\textsc{Or!}$, $\textsc{And}$, and $\textsc{And!} : \{0, \ldots, m-1\}^2 \to \{0, \ldots, m-1\}$, where

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

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

3. Counting functions: $\textsc{Add}_{\leq k}$, $\textsc{Mod}_k : \{0, \ldots, k\}^2 \to \{0, \ldots, k\}$, where

$$\textsc{Add}_{\leq k}(x,y) := \begin{cases} x + y & \text{if } x + y \leq k \\ \perp & \text{otherwise} \end{cases} \qquad \textsc{Mod}_k(x,y) := \begin{cases} 0 & \text{if } x = y = 0 \\ (x+y-1) \bmod k + 1 & \text{otherwise} \end{cases}$$

4. Function for modelling oracle access: $\textsc{Select} : \{0, \ldots, m-1\}^3 \to \{0, \ldots, m-1\}$, where

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

Next, we introduce the notion of underlined and overlined functions and numbers (e.g. we have $\overline{f}, \underline{f}$ or $\overline{0}, \underline{0}, \overline{x}, \underline{x}$ etc.). Let $f : \{0,\dots,m-1\}^k \to \{0,\dots,m-1\}$. Writing $\overline{f}$ means the function $\overline{f} : \{\overline{0},\dots,\overline{m-1}\}^k \to \{\overline{0},\dots,\overline{m-1}\}$ with $\overline{f}(\overline{x_1},\dots,\overline{x_k}) := \overline{f(x_1,\dots,x_k)}$ and $\overline{f}$ is undefined, otherwise. (Underlining is handled analogously.)

Before we come to our first results, we present a principle to be applied to F-TM's, playing a central role in the whole work. Due to Definition 3 each root node of a computation tree of an F-TM must evaluate to 0 or 1 (and nothing else is allowed). In this way, the functions which are contained in F force the computation tree to be of a special structure in order to avoid values different from 0 or 1 at its root. This is illustrated in the following two examples. First, consider the case where $F = \{\textsc{Or}!\}$ holds. To keep away from the root having e.g. value $\perp$, it is necessary that at most one leaf node of the tree has value 1 and all the other leaves are 0. Second, if it holds $F = \{\overline{f}, \textsc{Trans}^{\overline{1}\mapsto 0}_{A\mapsto 1}\}$, where $A = \{\overline{0}, \overline{2}, \dots, \overline{m-1}\}$ and f is some function from $\{0,\dots,m-1\}^k$ into $\{0,\dots,m-1\}$, then the following tree structure (and node labelling) is imperative: The root node of the computation tree of the F-TM is labelled with the transformation function and is connected (by a deterministic step) to a computation tree where all inner nodes are labelled $\overline{f}$ (and the leaf nodes are labelled by a constant from A). Otherwise, the root clearly wouldn't evaluate to 0 or 1. The above observation will be called the *propagation principle* in the following. This name is motivated by the fact that e.g. the "undefined value" $\perp$ propagates through the whole computation tree to the root. Thus the coming into being of a value $\perp$ always is forbidden in a computation of a F-TM.

We start with stating (without proof due to the lack of space) that for extended acceptance types the same normal form theorem as for Hertrampf's acceptance types holds.

Theorem 8 *Let F be a finite, extended locally definable acceptance type. Then there exists one binary function g such that $[F]\mathrm{P} = [\{g\}]\mathrm{P}$.*

In what follows, we will succinctly show that extended locally definable acceptance types are at least as powerful as their non-extended counterparts of Hertrampf. In addition, we indicate the closure under complementation and under the Boolean operations. Note that these results already have been established for locally definable acceptance types by Hertrampf [11] and that the proofs are quite similar for our extended version.

Theorem 9 *For all locally definable acceptance types F there exists an extended locally definable acceptance type F' such that it holds $(F)\mathrm{P} = [F']\mathrm{P}$.*

Proof. Let $\mathcal{C}$ be a class characterizable by some locally definable acceptance type F. Then there exists a binary function $f : \{0,\dots,m-1\}^2 \to \{0,\dots,m-1\}$ such that $\mathcal{C} = (\{f\})\mathrm{P}$ [11]. We claim that $F' = \{\overline{f}, \textsc{Trans}^{\overline{1}\mapsto 1}_{\overline{0},\overline{2},\dots,\overline{m-1}\mapsto 0}\}$ serves the purpose. (The details are contained in the full paper.) $\Box$

Finally, we establish the closure under complementation and the closure under the Boolean operations for extended locally definable types. Again this is a result which also holds for Hertrampf's acceptance types.

Lemma 10 *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 8) let $\mathcal{C} = [\{f\}]\mathrm{P}$, $\mathcal{C}_1 = [\{f_1\}]\mathrm{P}$, and $\mathcal{C}_2 = [\{f_2\}]\mathrm{P}$. We claim that $co - \mathcal{C} = [\{\overline{f}, \textsc{Trans}^{\overline{1}\mapsto 0}_{\overline{0}\mapsto 1}\}]\mathrm{P}$ and $\mathcal{C}_1 \vee \mathcal{C}_2 = [\{\textsc{Or}(\bar{\cdot}, \underline{\cdot}), \overline{f_1}, \underline{f_2}\}]\mathrm{P}$ hold. Herein, $\textsc{Or}(\bar{\cdot}, \underline{\cdot})$ is defined according to

$$\textsc{Or}(\bar{\cdot}, \underline{\cdot})(x,y) = \begin{cases} \textsc{Or}(u,v) & \text{if } \overline{u} = x,\ \underline{v} = y \\ \perp & \text{otherwise} \end{cases}$$

The claim for $\mathcal{C}_1 \wedge \mathcal{C}_2$ then follows by the closure under complementation and De Morgan's laws. (See again the full paper for details.) $\Box$

3 New Characterizations

We start with showing that UP, $\mathrm{UP}_{\leq k}$, $\mathrm{MODZ}_k\mathrm{P}$, and UAP are all locally definable. Note that for all these classes it does not seem to be possible to get a characterization in terms of a plain locally definable acceptance type. For a TM M we denote by $\mathrm{ACCEPT}(M,x)$ the number of accepting paths of M on input x. The class UP was defined by Valiant [16] via unambiguous polynomially time bounded TM's. Formally, a language L is a member of UP, if there exists a polynomially time bounded NTM M such that $\mathrm{ACCEPT}(M,x) = 1$ if $x \in L$ and $\mathrm{ACCEPT}(M,x) = 0$ if $x \notin L$.

Unambiguity plays an important role in complexity theory (e.g. cf. [2, 12, 13] for some recent results) and, in particular, in cryptography where the question for existence of one-way functions is shown to be equivalent with the question whether P = UP holds [8, 14]. $\mathrm{UP}_{\leq k}$ is a generalization of UP defined by Cai, Hemachandra, and Vyskoč [3]. Here up to k (for some constant k) accepting paths are allowed rather than only one.

Theorem 11 UP *and* $\mathrm{UP}_{\leq k}$ *are locally definable.*

Proof. We give the characterization for $\mathrm{UP}_{\leq k}$. Clearly, from this we also have a characterization for UP since this simply means the special case $k = 1$. We show that it holds $\mathrm{UP}_{\leq k} = [\{\overline{\textsc{Add}_{\leq k}}, \textsc{Trans}^{\overline{0}\mapsto 0}_{\overline{1},\ldots,\overline{k}\mapsto 1}\}]\mathrm{P}$.

For the inclusion "$\subseteq$" analogous to previous proofs we augment the computation tree of the $\mathrm{UP}_{\leq k}$-machine with a new root and a deterministic step to the root of the original tree. The new root is labeled with the transformation function all inner nodes of the original tree are labeled with $\overline{\textsc{Add}_{\leq k}}$, and its leaves are labeled with values $\overline{0}$ or $\overline{1}$. Due to the definition of $\overline{\textsc{Add}_{\leq k}}$ and the propagation principle the claim immediately follows.

To prove "$\supseteq$" we again apply the propagation principle. From this we have that the computation tree of an $\{\overline{\textsc{Add}_{\leq k}}, \textsc{Trans}^{\overline{0}\mapsto 0}_{\overline{1},\ldots,\overline{k}\mapsto 1}\}$-machine has to be of the above discussed form (that is the root is labeled with the transformation function and the remaining inner nodes have label $\overline{\textsc{Add}_{\leq k}}$). Obviously, such a tree can be simulated by a $\mathrm{UP}_{\leq k}$-machine. □

Beigel, Gill, and Hertrampf introduced the class $\mathrm{MODZ}_k\mathrm{P}$ [1]. $L \in \mathrm{MODZ}_k\mathrm{P}$, if there is a polynomially time bounded NTM M, such that

$$\begin{array}{llll} \mathrm{ACCEPT}(M,x) \not\equiv 0 & (\mathrm{mod}\ k) & \text{if} & x \in L, \\ \mathrm{ACCEPT}(M,x) = 0 & & \text{if} & x \notin L. \end{array}$$

In contrast to $\textsc{Mod}_k\mathrm{P}$, for $\mathrm{MODZ}_k\mathrm{P}$ rejection by $m * k$ accepting paths is not allowed for $m \geq 1$. We have FewP $\subseteq$ SPP $\subseteq$ $\textsc{Mod}_k\mathrm{P}$ and $\mathrm{MODZ}_k\mathrm{P}$ has some very interesting closure properties (see [1, 6]).

Theorem 12 $\mathrm{MODZ}_k\mathrm{P}$ *is locally definable.*

Proof. The proof is similar to the proof for the characterization of $\mathrm{UP}_{\leq k}$. Like there the number of accepting computations has to be counted (but now modulo k). It can be shown that $\mathrm{MODZ}_k\mathrm{P} = [\{\overline{\textsc{Mod}_k}, \textsc{Trans}^{\overline{0}\mapsto 0}_{\overline{1},\ldots,\overline{k-1}\mapsto 1}\}]\mathrm{P}$. □

Combining the concepts of unambiguity and alternation yields the class UAP. We call an alternating TM *unambiguous*, if for all inputs the computation tree does contain 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 TM's UAP. UAP has some interesting properties, which we list without proof: $\mathrm{UP}_{\leq k} \subseteq$ UAP, UAP $\subseteq$ SPP $\subseteq \oplus$P, and UAP $\subseteq \nabla\mathrm{P} \cap \mathit{Co}\nabla\mathrm{P}$. For a definition of Wagner's class ∇P see [11] or [18], for a definition of SPP see [6]. The next theorem shows another surprising property of UAP.

Theorem 13 FewP $\subseteq$ UAP.

Proof. Let M be an NTM with at most $p(n)$ accepting paths, where p is some polynomial. Let $L_m := \{ w \mid ||N_M(w)|| \geq m \}$. Clearly, $L_{p(n)+1} = \emptyset$ and $L_1 = L(M)$.

For $1 \leq m \leq p(n)$ an unambiguous, alternating TM can compute L_m as follows: Nondeterministically choose method (i) or (ii).

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

(ii) Accept iff $w \in L_{m+1}$.

$w \in L_{m+1}$ or $w \notin L_{m+1}$ is determined by the same method. A simple induction shows that this computation is correct and unambiguous (including the choice (i) and (ii)). Since the computation ends, when m reaches $p(n) + 1$, it is also polynomially time bounded. □

From Theorem 13 we also get the new inclusion FewP $\in \nabla$P. UAP is in some way an "unambiguous analogue" of ∇P. The following table compares relationships between ∇P and other complexity classes (see [18]) to results about UAP.

1-NP $\subseteq \nabla$P	UP $\subseteq$ UAP
∇P $\subseteq \forall \oplus$ P	UAP $\subseteq \oplus$P
∇P $\subseteq C_=$P	UAP $\subseteq$ SPP

Here $C_=$P is defined in [17] and SPP in [6]. We can also find Few-NP $\subseteq \nabla$P as an analogue of Theorem 13. (Few-NP is defined by machines that accept a word, iff they have more than zero and less than $p(n)$ accepting paths and reject, otherwise.) The proof is a little variation of the proof of Theorem 13. Another simple variation of the above proof shows even *Few* $\subseteq$ UAP. The known relations *Few* $\subseteq$ SPP and *Few* $\subseteq$ Mod_kP follow from this. To our best knowledge UAP is the smallest class containing *Few*, though our proof seems to be substantially easier than the proof of *Few* $\subseteq$ SPP [6].

Theorem 14 UAP *is locally definable.*

Proof. Again the proof essentially is a variation of the proof method used in Theorem 11. We claim that it holds UAP = $[\{\text{And}!, \text{Or}!\}]$P. □

By Theorem 9 all classes characterizable by locally definable acceptance types are also characterizable by extended locally definable acceptance types. Hertrampf showed that if a class $\mathcal{C}$ is characterizable , also $\exists\mathcal{C}$, $\forall\mathcal{C}$, $\oplus\mathcal{C}$, and $\text{MOD}_k\,\mathcal{C}$ are characterizable. We will now prove that extended locally definable acceptance types are also closed under these operations. Theorem 9 is not sufficient for this purpose, see for example $\forall$UP. Moreover, we show closure of extended locally definable acceptance types also under the operations $\exists!$ and $\forall!$ introduced by Hemachandra, as well as under MODZ_k which is defined analogously to the MOD_k operator in [11].

Definition 15

1. $L \in \exists!\,\mathcal{C}$ iff there exists a language $A \in \mathcal{C}$ and a polynomial p such that for all x, y, z such that $|y|, |z| \leq p(|x|)$, $\langle x, y\rangle, \langle x, z\rangle \in A$ we have $y = z$, and

$$x \in L \iff \exists y : |y| \leq p(|x|) \wedge \langle x, y\rangle \in A,$$

2. $\forall!\,\mathcal{C} = Co\,\exists!\,Co\mathcal{C}$.

Theorem 16 *If $\mathcal{C}$ is locally definable, then $\exists\mathcal{C}$, $\forall\mathcal{C}$, $\exists!\,\mathcal{C}$, $\forall!\,\mathcal{C}$,* $\text{MOD}_k\,\mathcal{C}$*, and* $\text{MODZ}_k\,\mathcal{C}$ *are also locally definable.*

Proof. Let $\mathcal{C} = [\{f\}]$P. We establish the following equalities:

1. $\exists\mathcal{C} = [\{\overline{f}, \textsc{Trans}^{\overline{0}\mapsto 0}_{\overline{1}\mapsto 1}, \textsc{Or}\}]\mathrm{P}$,
2. $\forall\mathcal{C} = [\{\overline{f}, \textsc{Trans}^{\overline{0}\mapsto 0}_{\overline{1}\mapsto 1}, \textsc{And}\}]\mathrm{P}$,
3. $\exists!\,\mathcal{C} = [\{\overline{f}, \textsc{Trans}^{\overline{0}\mapsto 0}_{\overline{1}\mapsto 1}, \textsc{Or}!\}]\mathrm{P}$,
4. $\forall!\,\mathcal{C} = [\{\overline{f}, \textsc{Trans}^{\overline{0}\mapsto 0}_{\overline{1}\mapsto 1}, \textsc{And}!\}]\mathrm{P}$,
5. $\mathrm{MOD}_k\,\mathcal{C} = [\{\overline{f}, \textsc{Trans}^{\overline{0}\mapsto \underline{0}}_{\overline{1}\mapsto \underline{1}}, \underline{\textsc{Mod}_k}, \textsc{Trans}^{\underline{0},\underline{k}\mapsto 0}_{\underline{1},\ldots,\underline{k-1}\mapsto 1}\}]\mathrm{P}$,
6. $\mathrm{MODZ}_k\,\mathcal{C} = [\{\overline{f}, \textsc{Trans}^{\overline{0}\mapsto \underline{0}}_{\overline{1}\mapsto \underline{1}}, \underline{\textsc{Mod}_k}, \textsc{Trans}^{\underline{0}\mapsto 0}_{\underline{1},\ldots,\underline{k-1}\mapsto 1}\}]\mathrm{P}$.

The first four points all lead to computation trees of the same form: First we have a tree (including the root node) where nodes are labeled with $\textsc{Or}$ ($\textsc{And}$, $\textsc{Or}!$, $\textsc{And}!$, respectively), then each leaf of this tree is connected to some node labeled with the transformation function, and finally each of these nodes has a subtree where all inner nodes are labeled with $\overline{f}$. Note that this shape of computation trees of F-machines is imperative due to the propagation principle. Clearly, here and in the following e.g. the $\overline{f}$-part of the above tree has to be understood as a tree which has values from $\{\overline{0}, \ldots, \overline{m-1}\}$ at its leaves and outputs $\overline{0}$ or $\overline{1}$. In the most trivial case, this means that such an $\overline{f}$-part may only consist of constant labels $\overline{0}$ or $\overline{1}$ (without nodes labeled $\overline{f}$).

Now we take a closer look at the characterization of $\exists\mathcal{C}$. The inclusion "$\subseteq$" can be seen in the following way. By definition of the $\exists$ operator it holds that $L \in \mathcal{C}$ iff there exists a language $A \in \mathcal{C}$ and a polynomial p such that $x \in L \iff \exists y : |y| \leq p(|x|) \wedge \langle x, y\rangle \in A$. This mechanism can be simulated by an $\{\overline{f}, \textsc{Trans}^{\overline{0}\mapsto 0}_{\overline{1}\mapsto 1}, \textsc{Or}\}$-machine as follows. We use a computation tree as described above. The $\textsc{Or}$-part is used to guess the string y and the $\overline{f}$-part is used to simulate the $\{f\}$-machine for $\mathcal{C}$ on input $\langle x, y\rangle$. Here we check whether it holds $\langle x, y\rangle \in A$. The transformation function in the above tree is just needed to have a clear seperation between the (existential) guessing of y and the simulation of the machine for $\mathcal{C}$ by $\overline{f}$.

For the reverse direction we show that an $\{\overline{f}, \textsc{Trans}^{\overline{0}\mapsto 0}_{\overline{1}\mapsto 1}, \textsc{Or}\}$-machine with computation tree of the above described shape (which is the only one possible due to the propagation principle) can be simulated by $\exists\mathcal{C}$. First, we have to existentially guess a path y through the $\textsc{Or}$-part of the above tree. Then this y is used to check whether $\langle x, y\rangle \in \mathcal{C}$ holds (where x denotes the input word). This obviously is the analogue to the $\overline{f}$-part of the above tree. Then the simulation leads to an acceptance iff the root of an $\overline{f}$-part evaluates to $\overline{1}$, which again has the consequence that the root (which is labeled $\textsc{Or}$) of the whole computation tree evaluates to 1. (Note that $\overline{1}$ will be mapped to 1 by the transformation function.)

Thus the claim for the first characterization follows. The claims for $\forall\mathcal{C}$, $\exists!\,\mathcal{C}$, and $\forall!\,\mathcal{C}$ are proven in a similar way and, therefore, are omitted.

The proofs for $\mathrm{MOD}_k\,\mathcal{C}$ and MODZ_k are quite similar. In both cases due to the propagation principle the computation tree of an F-machine has to be of the following shape. The root node is labeled $\textsc{Trans}^{\underline{0},\underline{k}\mapsto 0}_{\underline{1},\ldots,\underline{k-1}\mapsto 1}$ (resp. $\textsc{Trans}^{\underline{0}\mapsto 0}_{\underline{1},\ldots,\underline{k-1}\mapsto 1}$), then we have a subtree where nodes are labeled with $\underline{\textsc{Mod}_k}$, then each leaf of this $\underline{\textsc{Mod}_k}$-part is connected to some node labeled $\textsc{Trans}^{\overline{0}\mapsto \underline{0}}_{\overline{1}\mapsto \underline{1}}$, and finally each of these nodes has a subtree where all inner nodes are labeled with $\overline{f}$. The final leaf nodes are labeled $\overline{0}, \ldots, \overline{k}$. Note that the only difference between $\mathrm{MOD}_k\,\mathcal{C}$ and $\mathrm{MODZ}_k\,\mathcal{C}$ lies in the transformation function applied at the root of the whole computation tree. Now the proof is done analogously to that for $\exists\mathcal{C}$. We only need a second transformation function since the outcome of a $\textsc{Mod}_k$-part not necessarily is restricted to two values as is demanded for F-machines. $\square$

UAP is defined by unambiguous, alternating TM's. If we restrict the maximum number of alternations to some constant we get the levels of the so-called unambiguous alternation hierarchy. Hemachandra showed that these levels coincide with classes obtained from UP by iteratively applying unambiguous existential and universal polynomially length bounded quantification, so e.g. $A\Sigma_2^{\mathrm{UP}} = \forall!\,\exists!\,\mathrm{P} = \forall!\,\mathrm{UP}$, $A\Pi_2^{\mathrm{UP}} = \exists!\,\forall!\,\mathrm{P}$.

In the case of normal existential and universal polynomially length bounded quantifiers we get the levels of the polynomial time hierarchy [15] which coincide with the levels of the alternation hierarchy [19]. For unambiguous computation, however, the oracle and alternation hierarchies do *not* seem to coincide. Hemachandra could prove that all levels of the unambiguous alternation hierarchy coincide with levels of the unambiguous one-query oracle hierarchy.

As a corollary of the last theorem we therefore can characterize all levels of the unambiguous one-query hierarchy in terms of extended locally definable acceptance types.

To show that all levels of the polynomial time hierarchy are characterizable by locally definable acceptance types, Hertrampf showed closure of locally definable acceptance types under two operations which he calls Δ' and Σ'. For our situation it seems better to prove the following, even more extensive theorem directly:

Theorem 17 *If $\mathcal{C}$ is locally definable, then $\mathrm{P}^{\mathcal{C}}, \mathrm{UP}^{\mathcal{C}}$, and $\mathrm{NP}^{\mathcal{C}}$ are locally definable, too.*

Proof. Theorem 17 is a special case of the more general Theorem 24. For a direct proof one could use the following functions. Let $\mathcal{C} = [\{f\}]\mathrm{P}$. We claim that $\mathrm{P}^{\mathcal{C}} = [\{\overline{f}, \textsc{Select}\}]\mathrm{P}$, $\mathrm{UP}^{\mathcal{C}} = [\{\overline{f}, \textsc{Select}, \textsc{Or}{*}\}]\mathrm{P}$, and $\mathrm{NP}^{\mathcal{C}} = [\{\overline{f}, \textsc{Select}, \textsc{Or}\}]\mathrm{P}$. Herein, $\textsc{Or}{*}$ is defined as follows:

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

□

Observe that by Theorem 17 we are able to characterize all levels of the UP oracle hierarchy. Thus we have both unambiguous hierarchies characterized in the unified framework of extended locally definable acceptance types.

4 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 $L \in \mathrm{P}^{\mathrm{UP}}$, there is an oracle TM (OTM) M and an oracle $A \in \mathrm{UP}$, such that $L = L(M, A)$. A is the database and $A \in \mathrm{UP}$ means that all entries in the database are computed by an unambiguous polynomially time bounded NTM.

Cai, Hemachandra, and Vyskoč [3] introduced the class $\mathrm{P}^{\mathcal{UP}}$ which can be interpreted in a subroutine access way. Here $L \in \mathrm{P}^{\mathcal{UP}}$ iff $L = L(M, A)$, $A \in \mathcal{UP}$, but 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 claim that this notion of access to unambiguous computation (called *guarded access*), is more natural than the database view.

In this section we will show that guarded access is characterizable by extended 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 extended 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. Extended 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 extended locally definable acceptance types, too.

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

Definition 18 [5] A *promise problem* is a pair of predicates (Q, R). Q is called the *promise* and R the *property.*

Definition 19 [3] We define $\mathcal{UP}$ ("promise UP") as the following class of promise problems: $\{(Q_i,\, R_i) \mid i \geq 1\}$, where $N_1,\, N_2,\, \ldots$ is a standard enumeration of nondeterministic polynomial-time Turing machines with $Q_i = \{w \mid ||N_i(w)|| \leq 1\}$, and $R_i = \{w \mid ||N_i(w)|| \geq 1\}$.

Definition 20 Let $A = (Q,\, R)$ be a promise problem. We say that $L \in \mathrm{P}^A$ if there is a deterministic polynomial-time Turing machine M such that $L = L(M, R)$, and for every string x, in the computation of M every query z made to the oracle satisfies $z \in Q$. UP^A and NP^A are defined analogously.

For the proofs in this section we need the following technical notion:

Definition 21 Let F be an extended locally definable acceptance type. $(A, B) \in \langle F\rangle P$ iff there is an NTM M and an evaluation scheme as in Definition 2, but

- the root of M evaluates to one of $(0,0)$, $(0,1)$, $(1,0)$, or $(1,1)$ for all possible inputs,
- $w \in A$ 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, e.g., 0, 1, 2, and 3.

Lemma 22 *$\mathcal{UP} = \langle\{h\}\rangle\mathrm{P}$, where $h : \{(0,1),(1,0),(1,1)\}^2 \rightarrow \{(0,1),(1,0),(1,1)\}$ is defined as*

h	$(0,1)$	$(1,0)$	$(1,1)$
$(0,1)$	$(0,1)$	$(0,1)$	$(0,1)$
$(1,0)$	$(0,1)$	$(1,0)$	$(1,1)$
$(1,1)$	$(0,1)$	$(1,1)$	$(0,1)$

Proof. "$\subseteq$:" Let M be a polynomially time bounded NTM. Label all accepting final configurations of the computation tree of M by $(1,1)$ and all rejecting ones by $(1,0)$. If all inner configurations are labeled by h, then the root evaluates to $(0,1)$ iff there are more than one accepting paths, to $(1,1)$ if there is exactly one, and to $(1,0)$ if there is no accepting path at all. This means $(A, B) \in \langle\{f\}\rangle\mathrm{P}$, where $A = \{\, w \mid ||M(w) \leq 1 \,\}$, $B = \{\, w \mid ||M(w) \geq 1 \,\}$.

"$\supseteq$:" Let $(A, B) \in \langle\{f\}\rangle\mathrm{P}$. We have to show that there is a polynomial time bounded NTM N_i such that $A = \{\, w \mid ||N_i(w)|| \leq 1 \,\}$ and $B = \{\, w \mid ||N_i(w)|| \geq 1 \,\}$.

Since $(A, B) \in \langle\{f\}\rangle\mathrm{P}$, there is a witnessing TM M, i.e., the root of M's computation tree on input w evaluates to $(1,0)$ or $(1,1)$, if $w \in A$ and to $(0,1)$ or $(1,1)$, if $w \in B$. Let M' be an NTM which works as follows: M' simulates M using its nondeterminism to simulate branches of M labeled by f. If M' reaches a final configuration of M labeled by $(1,0)$ it rejects. If it is labeled $(1,1)$, it accepts and if it is labeled $(0,1)$, M' nondeterministically chooses two accepting configurations. Note that (by induction) M' has more than one accepting path, iff the root of M evaluates to $(0,1)$, it has exactly one accepting path, iff the root of M evaluates to $(1,1)$, and it has no accepting path at all, iff the root of M evaluates to $(1,0)$. Note that no leaves are labeled by $(0,0)$ since $(0,0)$ does not occur in the base set of h. $\square$

In order to prove the next theorem we need a technical lemma which states that the closure of a class $\langle F\rangle P$ under polynomial time reductions stays the same, if the base set of F is completed with $\{(0,0),(0,1)\}$. For a class $\langle F\rangle P$ itself this claim is in general false. For example, $(\emptyset,\emptyset)\notin \mathcal{UP}=\langle\{h\}\rangle\mathrm{P}$, but $(\emptyset,\emptyset)\in\langle\{h'\}\rangle\mathrm{P}$, where h' is defined as h, but additionally $h'((0,0)):=(0,0)$.

Lemma 23 *Let* $id:\{(0,0),(0,1)\}\to\{(0,0),(0,1)\}, x\mapsto x$ *and* $F'=F\cup\{id\}$, *where* F *is an extended locally definable acceptance type. Then* $\mathrm{P}^{\langle F\rangle\mathrm{P}}=\mathrm{P}^{\langle F'\rangle\mathrm{P}}$, $\mathrm{UP}^{\langle F\rangle\mathrm{P}}=\mathrm{UP}^{\langle F'\rangle\mathrm{P}}$, *and* $\mathrm{NP}^{\langle F\rangle\mathrm{P}}=\mathrm{NP}^{\langle F'\rangle\mathrm{P}}$.

Theorem 24 *Let* F *be an extended locally definable acceptance type. Then* $\mathrm{P}^{\langle F\rangle\mathrm{P}}$, $\mathrm{UP}^{\langle F\rangle\mathrm{P}}$, *and* $\mathrm{NP}^{\langle F\rangle\mathrm{P}}$ *are locally definable.*

Proof. We claim without proof that there is a single boolean function f such that $\langle F\rangle\mathrm{P}=\langle\{f\}\rangle\mathrm{P}$. (It can be shown by the same techniques as in Theorem 8.) Let $F_1=\{\overline{f},\textsc{Choose}\}$, $F_2=\{\overline{f},\textsc{Choose},\textsc{Or}*\}$, and $F_3=\{\overline{f},\textsc{Choose},\textsc{Or}\}$, where $\textsc{Choose}$ is defined as

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

We will show $[F_1]\mathrm{P}=\mathrm{P}^{\langle F\rangle\mathrm{P}}$, $[F_2]\mathrm{P}=\mathrm{UP}^{\langle F\rangle\mathrm{P}}$, and $[F_3]\mathrm{P}=\mathrm{NP}^{\langle F\rangle\mathrm{P}}$.

"$\subseteq$:" The propagation principle tells us that a computation tree of an F_i-machine M consists of nodes labeled $\textsc{Choose}$ with an $\overline{f}$-subtree on the right. In the case of F_2 (resp. F_3) there may be also $\textsc{Or}*$'s (resp. $\textsc{Or}$'s) outside of $\overline{f}$-subtrees. Also due to the propagation principle right successors of $\textsc{Choose}$-nodes have always value $\overline{(1,0)}$, $\overline{(1,1)}$, $\overline{(0,0)}$, or $\overline{(0,1)}$. Let us call in the case of $\overline{(1,0)}$ or $\overline{(0,0)}$ the left successor "forbidden", in the case of $\overline{(1,1)}$ or $\overline{0,0)}$ the middle successor is forbidden. By induction it can be seen that the result of an F_i-machine is 1, iff there exists an 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 to show this). Along such a path all $\textsc{Choose}$-nodes get only values $\overline{(1,0)}$ or $\overline{(1,1)}$ from their right successor (propagation principle!).

For F_1 a DTM can find this path (there is at most one) by avoiding forbidden nodes as follows: By queries to a suitable oracle (A,B), the DTM finds out the value of the right successor of a $\textsc{Choose}$-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 which has value $\overline{(1,0)}$ or $\overline{(1,1)}$. B consists of those p, such that the encoded path ends at a top of an $\overline{f}$-tree with value $\overline{(0,1)}$ or $\overline{(1,1)}$. Note that in this way no promise breaking questions are asked and $(A,B)\in\langle F\cup\{id\}\rangle\mathrm{P}$. For F_2 and F_3 an NTM can guess the path and verify with the help of an $\langle\{f\}\rangle\mathrm{P}$ oracle 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. We have shown $[F_1]\mathrm{P}\subseteq\mathrm{P}^{\langle F\cup\{id\}\rangle}$, $[F_2]\mathrm{P}\subseteq\mathrm{UP}^{\langle F\cup\{id\}\rangle}$, and $[F_3]\mathrm{P}\subseteq\mathrm{NP}^{\langle F\cup\{id\}\rangle}$. Lemma 23 completes the proof.

"$\supseteq$:" Let $L\in\mathrm{P}^{\langle F\rangle\mathrm{P}}$ (resp. $L\in\mathrm{UP}^{\langle F\rangle\mathrm{P}}$, $L\in\mathrm{NP}^{\langle F\rangle\mathrm{P}}$) and M be a witnessing OTM with oracle $(A,B)\in\langle F\rangle\mathrm{P}$, 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 $\textsc{Or}$- (resp. $\textsc{Or}*$-steps), oracle queries by a $\textsc{Choose}$-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 right 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 such 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. □

Corollary 25 *All levels of the unambiguous promise hierarchy and especially* $\mathrm{P}^{\mathcal{UP}}$, $\mathrm{UP}^{\mathcal{UP}}$, *and* $\mathrm{NP}^{\mathcal{UP}}$ *are locally definable.*

References

[1] R. Beigel, J. Gill, and U. Hertrampf. Counting classes: Threshold, parity, mods, and fewness. In *Proc. of 7th STACS*, number 415 in LNCS, pages 49–57. Springer, 1990.

[2] G. Buntrock, L. A. Hemachandra, and D. Siefkes. Using inductive counting to simulate nondeterministic computation. In *Proc. of 15th MFCS*, number 452 in LNCS, pages 187–194. Springer, 1990. (to appear in *Information and Computation*).

[3] J.-Y. Cai, L. Hemachandra, and J. Vyskoč. Promise problems and access to unambiguous computation. to appear in Proc. of MFCS'92, April 1992.

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

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

[6] S. A. Fenner, L. J. Fortnow, and S. A. Kurtz. Gap-definable counting classes. In *Proc. of 6th Conference on Structure in Complexity Theory*, pages 30–42, 1991.

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

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

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

[10] U. Hertrampf. Locally definable acceptance types—the three valued case. In *Proc. of LATIN'92*, number 583 in LNCS, pages 262–271. Springer, 1992.

[11] U. Hertrampf. Locally definable acceptance types for polynomial time machines. In *Proc. of 9th STACS*, number 577 in LNCS, pages 199–207. Springer, 1992.

[12] K. Lange. Unambiguity of circuits. In *Proc. of 5th Conference on Structure in Complexity Theory*, pages 130–137, 1990. (to appear in TCS).

[13] R. Niedermeier and P. Rossmanith. Unambiguous simulations of auxiliary pushdown automata and circuits. In *Proc. of LATIN'92*, number 583 in LNCS, pages 387–400. Springer, 1992.

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

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

[16] L. Valiant. The relative complexity of checking and evaluating. *Inform. Proc. Letters*, 5:20–23, 1976.

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

[18] K. W. Wagner. Alternating machines using partially defined "AND" and "OR". July 1992.

[19] C. Wrathall. Complete sets of the polynomial hierarchie. *Theoretical Comput. Sci.*, 3:23–33, 1977.