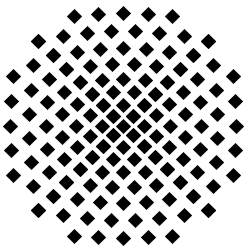


Universität Stuttgart
Fakultät Informatik

Institut für Informatik
Breitwiesenstraße 20–22
D–70565 Stuttgart

Acceptance by Transformation Monoids

(with an Application to
Local Self Reductions)

Ulrich Hertrampf

Report Nr. 1996/15

Abstract

We study the power of transformation monoids, which are used as an acceptance mechanism of nondeterministic polynomial time machines. Focussing our attention on four types of transformation monoids (including the monoids of all transformations on k elements) we obtain exact characterizations of all investigated polynomial time classes.

We apply these results to the cases of locally self reducible sets and of bottleneck Turing machines to obtain complete solutions to the formerly open problems related to these models. Especially, the complexity of k-locally self reducible sets for all numbers k, as well as the complexity of width-3 or width-4 bottleneck Turing machines are determined completely. Also for m-k-locally self reducible sets (i.e. k-locally self reducible sets, where the self reduction is given by a many-one reduction function) we determine the complexity exactly for all k.

1. Introduction

It has been a fruitful approach in structural complexity theory to define acceptance of nondeterministic polynomial time Turing machines by conditions posed on the ordered set of outcomes of the different computation paths of such a machine. Several models have been considered:

- "Counting classes" were defined by the number of accepting paths and a predicate which evaluated this number. As examples, see [PaZa83, CaHe89, GNW90].
- Polynomial time bit reductions [BCS92, HLSVW93], bottleneck Turing-machines [CaFu91], and serializable computations [Og94, HeOg95] can implicitly be interpreted as machine models where acceptance is defined by the sequence of outcomes of the computation paths of a nondeterministic polynomial time machine.
- Local self-reductions [BeSt95] also can be reformulated to fit in the above model. (See Section 4.)
- Locally definable acceptance types [He92a, He92b] were defined to generalize such an approach: not only the leaves, but every node of the computation tree influences the question of acceptance or rejection. However, in the case of an associative acceptance type [He94b], locally definable acceptance types are also one variant of the above described general phenomenon.
- The most explicit model of this kind is the leaf language model, which was introduced in [BCS91, BCS92] by Bovet, Crescenzi, and Silvestri. Moreover, Bovet et al. gave a mechanism, how to prove relativized inclusion or existence of separating oracles for the described complexity classes by investigating the respective leaf languages and their relations. This approach led to the solution of a very general separation problem in [He95b]; there, an algorithm was designed, which on input two classes of bounded counting type decides, whether there exists a separating oracle.

In most of these approaches, the vehicle to obtain results, i.e. mainly characterizations of so described complexity classes, was the monoid description. Thus for instance in [HLSVW93] it was shown that regular leaf languages whose syntactic monoid is solvable, describe complexity classes within MOD-PH.

In the current paper, we focus on those cases, where special kinds of transformation monoids are used as acceptance mechanism. A transformation monoid on k items is a monoid, whose elements are mappings from $\{1,\ldots,k\}$ into $\{1,\ldots,k\}$. Acceptance of a nondeterministic machine with leaf values from a given monoid usually is defined by a subset of the monoid: if the leaf string, evaluated in the monoid, yields an element of the given subset, the input is accepted, otherwise it is rejected. In the case of transformation monoids on k items we only allow special kinds of subsets by the following convention: The acceptance condition is given by a

subset of $\{1,\ldots,k\}$, i.e. the input is accepted if and only if the leaf string, evaluated in the monoid, yields an element which maps 1 into the given subset of $\{1,\ldots,k\}$.

There are mainly two motivations for this approach: It can be shown that k-local self reductions as introduced by Beigel and Straubing [BeSt95] are strongly related to the monoid of all transformations on 2^k items, and the many-one k-local self reductions are similarly related to a certain submonoid of the monoid of all transformations on k items. Secondly, the bottleneck Turing machines of width k, introduced by Cai and Furst [CaFu91] and further investigated by Ogihara [Og94] are strongly related to the full transformation monoid on k items.

One in some respect extreme special case of transformation monoids is the full permutation group S_k, which contains all bijective transformations. The power of finite groups as acceptance mechanisms was investigated in [He96] and in [He95a]. It is an easy exercise to find out that acceptance by the groups S_2, S_3, and S_4 characterizes the classes $\oplus$P, $\mathrm{MOD}_3 \cdot \oplus$P, and $\oplus \cdot \mathrm{MOD}_3 \cdot \oplus$P, respectively. Already in [HLSVW93] it was shown that S_k for $k \geq 5$ characterizes the class PSPACE. These groups were also considered in [Og94], where the corresponding classes were called *perm-SF*$_k$.

The current paper is organized as follows:

Section 2 will provide the necessary definitions and further preliminaries. In Section 3 we will state and prove our main results in their most general formulation. Then in Section 4 we can apply these results to solve the problems of (m-) k-locally self reducible sets and of width k bottleneck Turing machines.

2. Notations and Definitions

We assume the reader to be familiar with standard complexity theory notions, as can be found in [BDG88, BDG90, Pa94].

Definition. Let $k \geq 2$. Then we define the following two monoids:

M_k: The monoid M_k consists of all mappings f, such that $f\colon \{1,\ldots,k\} \to \{1,\ldots,k\}$. The monoid operation is composition of mappings. (This monoid has k^k elements.)

$\hat{M}_k$: The monoid $\hat{M}_k$ is the submonoid of M_k that consists of all non-bijective mappings plus the identity mapping. (This monoid has $k^k - (k!) + 1$ elements.)

Using these two monoids (for every k) we further define four complexity classes:

Definition. Let $k \geq 2$. The classes (M_k)P, (M_k')P, $(\hat{M}_k)$P, and $(\hat{M}_k')$P are defined as follows:

(M_k)P: A set L belongs to (M_k)P, if there is a polynomial time function f with values in $2^{\{1,\ldots,k\}}$, and a nondeterministic polynomial time machine N, which on every input x produces a leaf string $l_N(x)$ over M_k, such that $x \in L$ if and only if $l_N(x)$, evaluated in M_k and then (as a transformation) applied to 1 yields an element from $f(x)$.

(M'_k)P: This is the subclass of (M_k)P, where only machines are allowed, which on every input produce a leaf string $l_N(x)$ that does not yield a constant mapping when evaluated in M_k.

$(\hat{M}_k)$P: This is defined analogously to (M_k)P with the difference, that on all inputs the leaf string $l_N(x)$ has to be a string over $\hat{M}_k$.

$(\hat{M}'_k)$P: This is the subclass of $(\hat{M}_k)$P, where only machines are allowed, which on every input produce a leaf string $l_N(x)$ that does not yield a constant mapping when evaluated in $\hat{M}_k$.

We several times will need to deal with explicit transformations on 2, 3, 4, or 5 elements. To this end we introduce the following notational definition:

Definition. Let a and b be values from $\{1,2\}$. Then by $\frac{1\ 2}{a\ b}$ we denote the transformation t satisfying $t(1)=a$ and $t(2)=b$. Analogously the transformation $1\mapsto a$, $2\mapsto b$, $3\mapsto c$ on three items is denoted by $\frac{1\ 2\ 3}{a\ b\ c}$, and similarly for transformations on four elements $(\frac{1\ 2\ 3\ 4}{a\ b\ c\ d})$ and on five elements $(\frac{1\ 2\ 3\ 4\ 5}{a\ b\ c\ d\ e})$.

Now we would like to fix some notations regarding operators on complexity classes. The operators $\exists$ and $\forall$ should be known. In analogy to them the operators of the form MOD_k (for $k\geq 2$) are defined as follows: If $\mathcal{C}$ is a complexity class, then a set L belongs to $\mathrm{MOD}_k\cdot\mathcal{C}$, if and only if there is a set $B\in\mathcal{C}$ and a polynomial p, such that

$$x\in L \iff \#\{y \mid |y|\leq p(|x|)\wedge (x,y)\in B\}\not\equiv 0 \mod k.$$

The operator MOD_2 is usually denoted by $\oplus$. The dot between operator and class will often be omitted. Note that this does not cause problems, as for instance $\oplus\cdot\mathrm{P}$ and $\oplus\mathrm{P}$ indeed coincide. Similar equalities hold in all other cases considered in this paper.

For a class $\mathcal{C}$ the notation $\mathrm{P}^{\mathcal{C}}$ means the complexity class of languages accepted by a deterministic polynomial time Turing machine with access to an oracle from the class $\mathcal{C}$. If $\mathcal{C}_1$ and $\mathcal{C}_2$ are complexity classes, then by $\mathrm{P}^{\mathcal{C}_1:\mathcal{C}_2}$ (following [HHW95]) we mean the class of languages accepted by a deterministic polynomial time Turing machine with access to two oracles, one from $\mathcal{C}_1$ and the other one from $\mathcal{C}_2$, where the computation is done in two phases: in the first phase only oracle queries to the $\mathcal{C}_1$-oracle are allowed, and in the second phase only queries to the $\mathcal{C}_2$-oracle are allowed. It should be emphasized that classes and operators defined by query order are not at all artificial. Query order seems to be a very natural concept, as can be derived from the fact that it appears in a variety of contexts. E.g. see [HHW95, HHH96a, HHH96b].

If O is an operator and $\mathcal{C}$ is a complexity class, we will write $\mathrm{P}^{O}\cdot\mathcal{C}$ to denote $\mathrm{P}^{O\cdot\mathcal{C}}$. Similarly, if O_1 and O_2 are two operators, we write $\mathrm{P}^{O_1:O_2}\cdot\mathcal{C}$ to denote $\mathrm{P}^{O_1\cdot\mathcal{C}:O_2\cdot\mathcal{C}}$.

We also use the binary operators $\wedge$ and $\vee$ for complexity classes, where $\mathcal{C}_1\wedge\mathcal{C}_2$ consists of all languages L, such that there are $L_1\in\mathcal{C}_1$ and $L_2\in\mathcal{C}_2$, satisfying $L=L_1\cap L_2$. Analogously, $\mathcal{C}_1\vee\mathcal{C}_2$ is defined via $L=L_1\cup L_2$. Again, by $(O_1\wedge O_2)\mathcal{C}$ we mean $O_1\mathcal{C}\wedge O_2\mathcal{C}$, and analogously for $\vee$.

3. The Acceptance Power of Transformation Monoids

In this section we will clarify the computational complexity inherent in the classes $(M_k)\mathrm{P}$, $(M'_k)\mathrm{P}$, $(\hat{M}_k)\mathrm{P}$, and $(\hat{M}'_k)\mathrm{P}$. We start with some easily seen bounds and relations:

3.1 Proposition.

(i) For all $k \geq 2$ we have:

$$\mathrm{P} \subseteq (\hat{M}'_k)\mathrm{P} \subseteq (M'_k)\mathrm{P} \cap (\hat{M}_k)\mathrm{P} \subseteq (M'_k)\mathrm{P} \cup (\hat{M}_k)\mathrm{P} \subseteq (M_k)\mathrm{P} \subseteq \mathrm{PSPACE}$$

(ii) For all $k \geq 2$ we have:

$$(M'_k)\mathrm{P} \subseteq (\hat{M}'_{k+1})\mathrm{P} \quad \textit{and} \quad (M_k)\mathrm{P} \subseteq (\hat{M}_{k+1})\mathrm{P}$$

This result seems to indicate that there might be an infinite hierarchy built by these classes. However, using the main result from [HLSVW93] we can show that this hierarchy collapses down to $(M'_5)\mathrm{P}$:

3.2 Theorem.

(i) $(M_5)\mathrm{P} = (M'_5)\mathrm{P} = \mathrm{PSPACE}$

(ii) For all $k > 5$ we have: $(M_k)\mathrm{P} = (M'_k)\mathrm{P} = (\hat{M}_k)\mathrm{P} = (\hat{M}'_k)\mathrm{P} = \mathrm{PSPACE}$

Proof. It suffices to show $(M'_5)\mathrm{P} = \mathrm{PSPACE}$, because by Proposition 3.1 we then can conclude that all the other classes from the theorem are between $(M'_5)\mathrm{P}$ and PSPACE.

Now for $(M'_5)\mathrm{P}$ note that the non-solvable group S_5 is a submonoid of M_5, and no product of group elements will ever yield a constant transformation. Thus $(S_5)\mathrm{P} \subseteq (M'_5)\mathrm{P} \subseteq \mathrm{PSPACE}$, but from [HLSVW93] we know that $(S_5)\mathrm{P} = \mathrm{PSPACE}$. This proves the theorem. □

Thus there are 14 classes left, whose complexity has to be determined, namely the four classes for $k = 2$, as well as those for $k = 3$ and $k = 4$, and two classes for $k = 5$, the latter ones being $(\hat{M}_5)\mathrm{P}$ and $(\hat{M}'_5)\mathrm{P}$. To this end we will develop a series of simulation results in Lemma 3.3.

Definition. We say, complexity class $\mathcal{C}$ can be simulated in $(M_k)\mathrm{P}$ (or in $(M'_k)\mathrm{P}$, $(\hat{M}_k)\mathrm{P}$, $(\hat{M}'_k)\mathrm{P}$, resp.) via $(\frac{1\ 2\ 3\ \ldots\ k}{a\ b\ c\ \ldots\ .}/\frac{1\ 2\ 3\ \ldots\ k}{d\ e\ f\ \ldots\ .})$, if for all $L \in \mathcal{C}$ there is a machine M, which is an $(M_k)\mathrm{P}$-machine (or $(M'_k)\mathrm{P}$-machine, ...) such that for all x, if $x \in L$ then M's leaf string on input x evaluates to $\frac{1\ 2\ 3\ \ldots\ k}{a\ b\ c\ \ldots\ .}$, but if $x \notin L$ then M's leaf string on input x evaluates to $\frac{1\ 2\ 3\ \ldots\ k}{d\ e\ f\ \ldots\ .}$.

3.3 Lemma. *The following simulations hold:*

a) P *can be simulated in* (M_2)P *via* $(\frac{1\ 2}{2\ 1}/\frac{1\ 2}{1\ 2})$*, as well as via* $(\frac{1\ 2}{1\ 1}/\frac{1\ 2}{1\ 2})$*.*

b) Δ_2^p *can be simulated in* $(\hat{M}_2)$P *via* $(\frac{1\ 2}{1\ 1}/\frac{1\ 2}{2\ 2})$*.*

c) $\forall$P *can be simulated in* $(\hat{M}_3)$P *via* $(\frac{1\ 2\ 3}{2\ 1\ 1}/\frac{1\ 2\ 3}{1\ 2\ 2})$*, as well as via* $(\frac{1\ 2\ 3}{1\ 1\ 2}/\frac{1\ 2\ 3}{1\ 2\ 2})$*.*

d) $(\forall \wedge \mathrm{MOD}_3\oplus)$P *can be simulated in* (M_3)P *via* $(\frac{1\ 2\ 3}{2\ 1\ 2}/\frac{1\ 2\ 3}{1\ 2\ 1})$*, as well as via* $(\frac{1\ 2\ 3}{1\ 1\ 1}/\frac{1\ 2\ 3}{1\ 2\ 1})$*.*

e) $(\forall \wedge \mathrm{MOD}_3\oplus)\exists$P *can be simulated in* $(\hat{M}_4)$P *via* $(\frac{1\ 2\ 3\ 4}{2\ 1\ 1\ 2}/\frac{1\ 2\ 3\ 4}{1\ 2\ 1\ 1})$*, as well as via* $(\frac{1\ 2\ 3\ 4}{1\ 1\ 2\ 2}/\frac{1\ 2\ 3\ 4}{1\ 2\ 1\ 1})$*.*

f) $(\forall\wedge\mathrm{MOD}_3\oplus)(\exists\vee\oplus\mathrm{MOD}_3\oplus)$P *can be simulated in* (M_4)P *via* $(\frac{1\ 2\ 3\ 4}{2\ 1\ 2\ 2}/\frac{1\ 2\ 3\ 4}{1\ 2\ 1\ 1})$*, as well as via* $(\frac{1\ 2\ 3\ 4}{1\ 1\ 1\ 1}/\frac{1\ 2\ 3\ 4}{1\ 2\ 1\ 1})$*.*

g) $(\forall\ \wedge\ \mathrm{MOD}_3\oplus)(\exists\ \vee\ \oplus\mathrm{MOD}_3\oplus)\forall$P *can be simulated in* $(\hat{M}_5)$P *via* $(\frac{1\ 2\ 3\ 4\ 5}{2\ 1\ 2\ 2\ 2}/\frac{1\ 2\ 3\ 4\ 5}{1\ 2\ 1\ 1\ 1})$*, as well as via* $(\frac{1\ 2\ 3\ 4\ 5}{1\ 1\ 1\ 1\ 1}/\frac{1\ 2\ 3\ 4\ 5}{1\ 2\ 1\ 1\ 1})$*.*

Proof.

a) Use a computation tree with only one leaf, and produce the leaf value as needed.

b) We use the following well known characterization of Δ_2^p: A set L belongs to Δ_2^p if and only if there is a nondeterministic machine N such that for all x we have, $x \in L$, if and only if the rightmost accepting path of N on input x is in an odd position. (Here we use only normalized machines, where the computation tree is a full binary tree, and without loss of generality we assume that the leftmost path is accepting.) Now, we simulate N, but instead of rejection in a leaf we produce the identity transformation; instead of acceptance in a leaf which is on a path in an even position, we produce the transformation $\frac{1\ 2}{2\ 2}$, and instead of acceptance in a leaf which is on a path in an odd position, we produce the transformation $\frac{1\ 2}{1\ 1}$. Clearly, the whole leaf string will evaluate to $\frac{1\ 2}{1\ 1}$, if the simulated computation accepts, and it will evaluate to $\frac{1\ 2}{2\ 2}$, otherwise.

c) We can, in a natural way, simulate $\forall$P in $(\hat{M}_3)$P via $(\frac{1\ 2\ 3}{1\ 2\ 3}/\frac{1\ 2\ 3}{1\ 3\ 3})$ as well as via $(\frac{1\ 2\ 3}{1\ 2\ 3}/\frac{1\ 2\ 3}{2\ 2\ 3})$ or via $(\frac{1\ 2\ 3}{1\ 2\ 3}/\frac{1\ 2\ 3}{1\ 2\ 1})$. Now, build a new simulation as follows: First branch to three possible configurations. In the leftmost one simulate via $(\frac{1\ 2\ 3}{1\ 2\ 3}/\frac{1\ 2\ 3}{1\ 3\ 3})$, in the middle one simulate via $(\frac{1\ 2\ 3}{1\ 2\ 3}/\frac{1\ 2\ 3}{2\ 2\ 3})$, and in the rightmost one simulate via $(\frac{1\ 2\ 3}{1\ 2\ 3}/\frac{1\ 2\ 3}{1\ 2\ 1})$. If the input was to be accepted, then all three parts lead to value $\frac{1\ 2\ 3}{1\ 2\ 3}$, thus also the whole tree evaluates to the same value. If the input was to be rejected, then the three parts evaluate to the values $\frac{1\ 2\ 3}{1\ 3\ 3}$, $\frac{1\ 2\ 3}{2\ 2\ 3}$, and $\frac{1\ 2\ 3}{1\ 2\ 1}$. Thus the whole tree evaluates to their product, i.e.

$\frac{1\ 2\ 3}{2\ 1\ 1}$. Now add one leaf of value $\frac{1\ 2\ 3}{2\ 1\ 1}$ to obtain $\frac{1\ 2\ 3}{2\ 1\ 1}$ in case of acceptance, and $\frac{1\ 2\ 3}{1\ 2\ 2}$ in case of rejection.

For the second simulation of ∀P take the easy simulation via $(\frac{1\ 2\ 3}{1\ 2\ 3}/\frac{1\ 2\ 3}{1\ 3\ 3})$ and add one leaf of value $\frac{1\ 2\ 3}{1\ 1\ 2}$ to obtain $\frac{1\ 2\ 3}{1\ 1\ 2}$ for acceptance and $\frac{1\ 2\ 3}{1\ 2\ 2}$ for rejection.

d) We use a simulation of ∀P via $(\frac{1\ 2\ 3}{1\ 2\ 3}/\frac{1\ 2\ 3}{2\ 1\ 1})$ as developed in the proof of part c). Moreover, it is easy to see that $\mathrm{MOD}_3\oplus\mathrm{P}$ can be simulated via $(\frac{1\ 2\ 3}{1\ 2\ 3}/\frac{1\ 2\ 3}{2\ 3\ 1})$. We call the first simulation A, and the second one B. Now consider a simulation consisting of seven parts, namely $xBAByAz$, where $x = \frac{1\ 2\ 3}{3\ 1\ 3}$, $y = \frac{1\ 2\ 3}{3\ 1\ 2}$, and $z = \frac{1\ 2\ 3}{1\ 2\ 1}$. In the case that both the ∀P-computation and the $\mathrm{MOD}_3\oplus\mathrm{P}$-computation are accepting, we obtain the leaf value $\frac{1\ 2\ 3}{3\ 1\ 3}\cdot\frac{1\ 2\ 3}{1\ 2\ 3}\cdot\frac{1\ 2\ 3}{1\ 2\ 3}\cdot\frac{1\ 2\ 3}{1\ 2\ 3}\cdot\frac{1\ 2\ 3}{3\ 1\ 2}\cdot\frac{1\ 2\ 3}{1\ 2\ 3}\cdot\frac{1\ 2\ 3}{1\ 2\ 1}$, which equals $\frac{1\ 2\ 3}{2\ 1\ 2}$. But if either A is replaced by $\frac{1\ 2\ 3}{2\ 1\ 1}$ or B is replaced by $\frac{1\ 2\ 3}{2\ 3\ 1}$ or both, then in every case we obtain the value $\frac{1\ 2\ 3}{1\ 2\ 1}$.

For the second statement, we have to use a different simulation, again consisting of seven parts, namely $wBAwBAz$, where A, B, and z are as above, and $w = \frac{1\ 2\ 3}{1\ 3\ 1}$. Then we obtain $\frac{1\ 2\ 3}{1\ 1\ 1}$, if A and B both equal $\frac{1\ 2\ 3}{1\ 2\ 3}$, and $\frac{1\ 2\ 3}{1\ 2\ 1}$ in all other cases.

e) Now we start by simulating ∀P in $(\hat{M}_4)$P like in part c), leaving additional value 4 untouched, i.e. via $(\frac{1\ 2\ 3\ 4}{2\ 1\ 1\ 4}/\frac{1\ 2\ 3\ 4}{1\ 2\ 2\ 4})$, or equivalently by exchanging the roles of values 3 and 4, we obtain a simulation via $(\frac{1\ 2\ 3\ 4}{2\ 1\ 3\ 1}/\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 2})$. Then it should be evident, how to simulate $\mathrm{MOD}_3\oplus\forall\mathrm{P}$ via $(\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 2}/\frac{1\ 2\ 3\ 4}{2\ 3\ 1\ 3})$. Recall that $\mathrm{MOD}_3\oplus\exists\mathrm{P} = \mathrm{MOD}_3\oplus\forall\mathrm{P}$.

Then we simulate ∃P in a natural way via $(\frac{1\ 2\ 3\ 4}{1\ 4\ 3\ 4}/\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 4})$. Adding a leaf of value $\frac{1\ 2\ 3\ 4}{1\ 3\ 3\ 2}$ on the right side, we obtain a simulation of ∃P via $(\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 2}/\frac{1\ 2\ 3\ 4}{1\ 3\ 3\ 2})$. This can be extended to a simulation of ∀∃P via $(\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 2}/\frac{1\ 2\ 3\ 4}{1\ 3\ 3\ 3})$. Analogously we obtain simulations of ∀∃P via $(\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 1}/\frac{1\ 2\ 3\ 4}{2\ 2\ 3\ 2})$ or via $(\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 3}/\frac{1\ 2\ 3\ 4}{1\ 2\ 1\ 1})$. Thus, as in part c), we obtain a simulation of ∀∃P via $(\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 2}/\frac{1\ 2\ 3\ 4}{2\ 1\ 1\ 1})$.

Now, the same arguments as in part d), just on four values instead of three (but with no meaning associated with the image of 4) yield both statements.

f) First note that $\mathrm{MOD}_3\oplus\mathrm{P}$ can be simulated via $(\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 4}/\frac{1\ 2\ 3\ 4}{2\ 3\ 1\ 4})$, as shown in part d) (just add an unused value 4). Call that simulation A. Then xAx, where $x = \frac{1\ 2\ 3\ 4}{2\ 2\ 3\ 4}$ yields a simulation of $\mathrm{MOD}_3\oplus\mathrm{P}$ via $(\frac{1\ 2\ 3\ 4}{2\ 2\ 3\ 4}/\frac{1\ 2\ 3\ 4}{3\ 3\ 2\ 4})$. This simulation can be used to show directly that also $\oplus\mathrm{MOD}_3\oplus\mathrm{P}$ can be simulated via $(\frac{1\ 2\ 3\ 4}{2\ 2\ 3\ 4}/\frac{1\ 2\ 3\ 4}{3\ 3\ 2\ 4})$. Call this simulation B.

On the other hand, again as in part d) by exchanging the roles of the variables 3 and 4, one can also simulate $\mathrm{MOD}_3\oplus\mathrm{P}$ via $(\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 4}/\frac{1\ 2\ 3\ 4}{2\ 4\ 3\ 1})$. Call this simulation C. Then AC simulates $\mathrm{MOD}_3\oplus\mathrm{P}$ via $(\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 4}/\frac{1\ 2\ 3\ 4}{4\ 3\ 2\ 1})$.

Since $\frac{1\ 2\ 3\ 4}{4\ 3\ 2\ 1}$ squared yields $\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 4}$, we can use this simulation also to simulate $\oplus\mathrm{MOD}_3\oplus\mathrm{P}$. Call this simulation D. Now, obviously BD simulates $\oplus\mathrm{MOD}_3\oplus\mathrm{P}$ via $(\frac{1\ 2\ 3\ 4}{2\ 2\ 3\ 4}/\frac{1\ 2\ 3\ 4}{2\ 2\ 3\ 1})$.

Clearly, $\forall\mathrm{P}$ can be simulated via $(\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 4}/\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 1})$. If we call this simulation E, then also $\forall\mathrm{P}\wedge\oplus\mathrm{MOD}_3\oplus\mathrm{P}$ can be simulated via $(\frac{1\ 2\ 3\ 4}{2\ 2\ 3\ 4}/\frac{1\ 2\ 3\ 4}{2\ 2\ 3\ 1})$ by taking BDE as the required simulation, where B and D simulate the $\oplus\mathrm{MOD}_3\oplus\mathrm{P}$-computation, and E simulates the $\forall\mathrm{P}$-computation. Taking complements, we obtain that $(\exists\vee\oplus\mathrm{MOD}_3\oplus)\mathrm{P}$ can be simulated via $(\frac{1\ 2\ 3\ 4}{2\ 2\ 3\ 1}/\frac{1\ 2\ 3\ 4}{2\ 2\ 3\ 4})$. Adding a leaf of value $\frac{1\ 2\ 3\ 4}{4\ 2\ 3\ 1}$, we obtain a simulation of $(\exists\vee\oplus\mathrm{MOD}_3\oplus)\mathrm{P}$ via $(\frac{1\ 2\ 3\ 4}{2\ 2\ 3\ 4}/\frac{1\ 2\ 3\ 4}{2\ 2\ 3\ 1})$. Thus $\forall(\exists\vee\oplus\mathrm{MOD}_3\oplus)\mathrm{P}$ can naturally be simulated via $(\frac{1\ 2\ 3\ 4}{2\ 2\ 3\ 4}/\frac{1\ 2\ 3\ 4}{2\ 2\ 3\ 2})$. This simulation works like a natural simulation of $\forall\mathrm{P}$ on three values 2, 3, 4, only the value 1 has got an image 2. By exchanging roles of variables, we can also simulate $\forall(\exists\vee\oplus\mathrm{MOD}_3\oplus)\mathrm{P}$ via $(\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 3}/\frac{1\ 2\ 3\ 4}{1\ 3\ 3\ 3})$ or via $(\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 2}/\frac{1\ 2\ 3\ 4}{2\ 2\ 3\ 2})$ or via $(\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 1}/\frac{1\ 2\ 3\ 4}{1\ 2\ 1\ 1})$. Thus, as in part c) we obtain a simulation via $(\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 3}/\frac{1\ 2\ 3\ 4}{2\ 1\ 1\ 1})$. Call this simulation F.

As to the $\mathrm{MOD}_3\oplus(\exists\vee\oplus\mathrm{MOD}_3\oplus)\mathrm{P}$ part, we use the fact that this equals $\mathrm{MOD}_3\oplus(\exists\vee\mathrm{MOD}_3\oplus)\mathrm{P}$, which can be easily shown using well known facts about parity and modulo computations. From part d) we obtain a simulation of $(\exists\vee\mathrm{MOD}_3\oplus)\mathrm{P}$ via $(\frac{1\ 2\ 3\ 4}{1\ 2\ 1\ 4}/\frac{1\ 2\ 3\ 4}{2\ 1\ 2\ 4})$ by adding an unused value 4. Now on values 1, 2, 4, this simulation acts like a parity computation. Thus we can simulate $\mathrm{MOD}_3\oplus(\exists\vee\mathrm{MOD}_3\oplus)\mathrm{P}$ via $(\frac{1\ 2\ 3\ 4}{1\ 2\ 4\ 4}/\frac{1\ 2\ 3\ 4}{2\ 4\ 1\ 1})$. Exchanging the roles of values 3 and 4 we obtain a simulation via $(\frac{1\ 2\ 3\ 4}{1\ 2\ 3\ 3}/\frac{1\ 2\ 3\ 4}{2\ 3\ 1\ 1})$. Call this simulation G.

Now we can proceed as in part d), taking $xGFGyFz$ for the simulation via $(\frac{1\ 2\ 3\ 4}{2\ 1\ 2\ 2}/\frac{1\ 2\ 3\ 4}{1\ 2\ 1\ 1})$, and $wGFwGFz$ for the simulation via $(\frac{1\ 2\ 3\ 4}{1\ 1\ 1\ 1}/\frac{1\ 2\ 3\ 4}{1\ 2\ 1\ 1})$, where $x=\frac{1\ 2\ 3\ 4}{3\ 1\ 3\ 4}$, $y=\frac{1\ 2\ 3\ 4}{3\ 1\ 2\ 4}$, $z=\frac{1\ 2\ 3\ 4}{1\ 2\ 1\ 4}$, and $w=\frac{1\ 2\ 3\ 4}{1\ 3\ 1\ 4}$.

g) The difference between f) and g) is exactly as the one between d) and e). We will use the fifth available value to perform the inner $\forall$-operator and change it to a parity-like behavior for the $\oplus\mathrm{MOD}_3\oplus\forall\mathrm{P}$-part, or construct an $\exists\forall\mathrm{P}$-simulation for the second part. We leave the details to the reader.

□

The next lemma allows us to use the results of the previous lemma in order to obtain upper bounds for the optimization classes obtained by applying the $\mathrm{P}^{\exists:\oplus}$-operator. By dots in the mappings of the form $\frac{1\ 2\ 3\ \ldots\ k}{2\ 1\ .\ \ldots\ .}$, we mean that any value is allowed as image in this place.

3.4 Lemma. *Let $\mathcal{C}$ be a complexity class, which is closed under polynomial time conjunctive truth-table reduction. If, for $k\geq 2$, $\mathcal{C}$ can be simulated in $(M_k)\mathrm{P}$ via $(\frac{1\ 2\ 3\ \ldots\ k}{2\ 1\ .\ \ldots\ .}/\frac{1\ 2\ 3\ \ldots\ k}{1\ 2\ .\ \ldots\ .})$ as well as via $(\frac{1\ 2\ 3\ \ldots\ k}{1\ 1\ .\ \ldots\ .}/\frac{1\ 2\ 3\ \ldots\ k}{1\ 2\ .\ \ldots\ .})$, then $\mathrm{P}^{\exists:\oplus}\cdot\mathcal{C}$ can be simulated in $(M_k)\mathrm{P}$ via $(\frac{1\ 2\ 3\ \ldots\ k}{1\ 1\ 1\ \ldots\ 1}/\frac{1\ 2\ 3\ \ldots\ k}{2\ 2\ 2\ \ldots\ 2})$.*

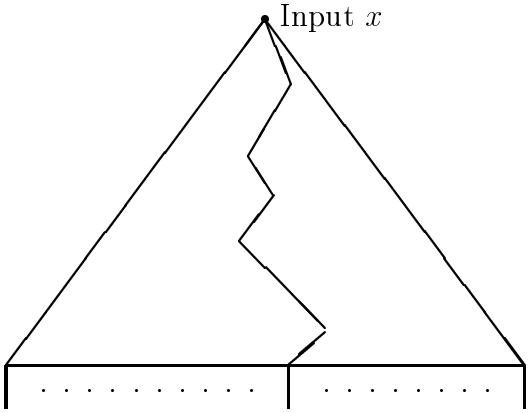
If $k\geq 3$, then the same statement holds for $(\hat{M}_k)\mathrm{P}$ instead of $(M_k)\mathrm{P}$.

Proof. Let $L \in \mathrm{P}^{\exists \cdot \oplus} \cdot \mathcal{C}$ via machine N. By the assumed closure property of the class $\mathcal{C}$ we obtain closure of $\oplus\mathcal{C}$ under complementation as well as under conjunctive and disjunctive truth-table reductions. Thus we may assume that N in every computation asks exactly one query to the $\oplus\mathcal{C}$-oracle, and N accepts if and only if that query is answered positively. We use a well-known technique to have a nondeterministic machine guess the oracle answers of the $\exists\mathcal{C}$-oracle in such a way that when only the YES-answers are checked, on the rightmost path where all the checks are successful, even the NO-answers have been correct. The algorithm that achieves this is the following:

$i := 1$;
LOOP
    Compute i-th oracle query $q(i)$;
    Guess oracle answer $a(i)$ on $q(i)$:
        In left subtree, let $a(i) :=$ NO;
        In right subtree, let $a(i) :=$ YES;
    IF no more queries to NP-oracle THEN
        EXIT LOOP
    ELSE $i := i + 1$;
END OF LOOP

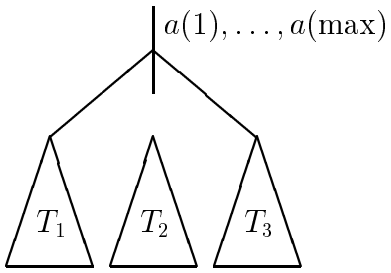
The reader may check that the computation tree produced by this algorithm in fact has the desired property.

We obtain a subtree which looks as follows:



Here, on every path of the subtree we have a specific sequence of oracle answers $a(1), \ldots, a(\max)$.

Now, we split into three subtrees (for each path):



The subtrees T_1 and T_3 are identical; both simulate the final $\oplus\mathcal{C}$-oracle query (depending on $a(1), \ldots, a(\max)$), using the simulation via $(\frac{1\ 2\ 3\ \ldots\ k}{2\ 1\ .\ \ldots\ .}/\frac{1\ 2\ 3\ \ldots\ k}{1\ 2\ .\ \ldots\ .})$, which can be obtained directly from the simulation of $\mathcal{C}$ via $(\frac{1\ 2\ 3\ \ldots\ k}{2\ 1\ .\ \ldots\ .}/\frac{1\ 2\ 3\ \ldots\ k}{1\ 2\ .\ \ldots\ .})$. Thus, the evaluated leaf string in each of these subtrees yields $\frac{1\ 2\ 3\ \ldots\ k}{2\ 1\ .\ \ldots\ .}$, if the $\oplus\mathcal{C}$-oracle answers positively, and $\frac{1\ 2\ 3\ \ldots\ k}{1\ 2\ .\ \ldots\ .}$, otherwise. In the middle subtree T_2, however, we check by one $\exists\mathcal{C}$-simulation, using the simulation via $(\frac{1\ 2\ 3\ \ldots\ k}{1\ 1\ .\ \ldots\ .}/\frac{1\ 2\ 3\ \ldots\ k}{1\ 2\ .\ \ldots\ .})$, which can be obtained directly from the simulation of $\mathcal{C}$ via $(\frac{1\ 2\ 3\ \ldots\ k}{1\ 1\ .\ \ldots\ .}/\frac{1\ 2\ 3\ \ldots\ k}{1\ 2\ .\ \ldots\ .})$, whether all the guessed YES-answers on the $\exists\mathcal{C}$-oracle queries were correct. We do that by producing leaf value $\frac{1\ 2\ 3\ \ldots\ k}{1\ 1\ .\ \ldots\ .}$ on accepting paths, and $\frac{1\ 2\ 3\ \ldots\ k}{1\ 2\ .\ \ldots\ .}$ on rejecting paths.

We analyze the behavior of this machine: If a wrong YES-answer is in the list $a(1), \ldots, a(\max)$, then T_2 has only rejecting paths. Thus, the leaf string of this subtree evaluates to $\frac{1\ 2\ 3\ \ldots\ k}{1\ 2\ .\ \ldots\ .}$. Then, T_1 and T_3, which either both have value $\frac{1\ 2\ 3\ \ldots\ k}{2\ 1\ .\ \ldots\ .}$, or both have value $\frac{1\ 2\ 3\ \ldots\ k}{1\ 2\ .\ \ldots\ .}$, will together yield $\frac{1\ 2\ 3\ \ldots\ k}{1\ 2\ .\ \ldots\ .}$ in any case. But, if all YES-answers were correct, then T_2 will yield $\frac{1\ 2\ 3\ \ldots\ k}{1\ 1\ .\ \ldots\ .}$, and so the combination of T_1, T_2, and T_3 will yield $\frac{1\ 2\ 3\ \ldots\ k}{1\ 1\ .\ \ldots\ .}$, if T_3 rejects, or $\frac{1\ 2\ 3\ \ldots\ k}{2\ 2\ .\ \ldots\ .}$, if T_3 accepts. Thus, the whole computation tree on input x will yield value $\frac{1\ 2\ 3\ \ldots\ k}{2\ 2\ .\ \ldots\ .}$, if the rightmost guess without wrong YES-answers leads to acceptance, and it will yield value $\frac{1\ 2\ 3\ \ldots\ k}{1\ 1\ .\ \ldots\ .}$, otherwise. Now, adding one single leaf of value $\frac{1\ 2\ 3\ \ldots\ k}{2\ 1\ .\ \ldots\ .}$ on the right, and one single leaf of value $\frac{1\ 2\ 3\ \ldots\ k}{1\ 1\ 1\ \ldots\ 1}$ on the left, will yield the desired simulation.

If the assumed simulations of $\mathcal{C}$ are possible even in $(\hat{M}_k)$P, then also the constructed simulation of $\mathrm{P}^{\exists:\oplus}\cdot\mathcal{C}$ will be possible in $(\hat{M}_k)$P, provided the mappings of the form $\frac{1\ 2\ 3\ \ldots\ k}{2\ 1\ .\ \ldots\ .}$, which were needed in the proof, may be used in $(\hat{M}_k)$P, which can be achieved in the case $k > 2$ by choosing 1 as the image of 3. This completes the proof. □

Now we are ready to give upper bounds also for classes of the form $\mathrm{P}^{\exists:\oplus}\cdot\mathcal{C}$ for certain classes $\mathcal{C}$.

3.5 Lemma. *The following simulations hold:*

a) $\mathrm{P}^{\exists:\oplus}\cdot\mathrm{P}$ *can be simulated in* (M_2)P *via* $(\frac{1\ 2}{1\ 1}/\frac{1\ 2}{2\ 2})$.

b) $\mathrm{P}^{\exists:\oplus}\cdot\forall\mathrm{P}$ *can be simulated in* $(\hat{M}_3)$P *via* $(\frac{1\ 2\ 3}{1\ 1\ 1}/\frac{1\ 2\ 3}{2\ 2\ 2})$.

c) $\mathrm{P}^{\exists:\oplus}\cdot(\forall\wedge\mathrm{MOD}_3\oplus)\mathrm{P}$ *can be simulated in* (M_3)P *via* $(\frac{1\ 2\ 3}{1\ 1\ 1}/\frac{1\ 2\ 3}{2\ 2\ 2})$.

d) $\mathrm{P}^{\exists:\oplus}\cdot(\forall\wedge\mathrm{MOD}_3\oplus)\exists\mathrm{P}$ *can be simulated in* $(\hat{M}_4)\mathrm{P}$ *via* $(\frac{1\ 2\ 3\ 4}{1\ 1\ 1\ 1}/\frac{1\ 2\ 3\ 4}{2\ 2\ 2\ 2})$.

e) $\mathrm{P}^{\exists:\oplus}\cdot(\forall\wedge\mathrm{MOD}_3\oplus)(\exists\vee\oplus\mathrm{MOD}_3\oplus)\mathrm{P}$ *can be simulated in* $(M_4)\mathrm{P}$ *via* $(\frac{1\ 2\ 3\ 4}{1\ 1\ 1\ 1}/\frac{1\ 2\ 3\ 4}{2\ 2\ 2\ 2})$.

f) $\mathrm{P}^{\exists:\oplus}\cdot(\forall\wedge\mathrm{MOD}_3\oplus)(\exists\vee\oplus\mathrm{MOD}_3\oplus)\forall\mathrm{P}$ *can be simulated via* $(\frac{1\ 2\ 3\ 4\ 5}{1\ 1\ 1\ 1\ 1}/\frac{1\ 2\ 3\ 4\ 5}{2\ 2\ 2\ 2\ 2})$ *in* $(\hat{M}_5)\mathrm{P}$.

Proof. All the statements follow if we apply Lemma 3.4 to the suitable statements of Lemma 3.3. □

Now we are ready to prove our main results:

3.6 Theorem. $(\hat{M}_2')\mathrm{P}=\mathrm{P}$ $\quad$ $(M_2')\mathrm{P}=\oplus\mathrm{P}$

$(\hat{M}_2)\mathrm{P}=\Delta_2^p$ $\quad$ $(M_2)\mathrm{P}=\mathrm{P}^{\exists:\oplus}\cdot\mathrm{P}$

Proof. We prove the four statements:

$(\hat{M}_2')\mathrm{P}$: The monoid $\hat{M}_2$ consists of the two constant mappings and the identity mapping. Now let L be a set in $(\hat{M}_2')\mathrm{P}$. Then there is a nondeterministic Turing machine M, which in every leaf outputs the identity mapping, and a polynomial time computable function f, such that the values of f are subsets of $\{1,2\}$, and $x\in L$, if and only if $1\in f(x)$. Thus L is clearly in P.

The reverse direction is Lemma 3.3 a).

$(M_2')\mathrm{P}$: Now in addition to the identity mapping we may also use the mapping $\frac{1\ 2}{2\ 1}$. So, obviously, $(M_2')\mathrm{P}$ coincides with $\oplus\mathrm{P}$.

$(\hat{M}_2)\mathrm{P}$: In this case we may use both constant mappings and the identity mapping. Let L be in $(\hat{M}_2)\mathrm{P}$. Then there is a nondeterministic machine, which on every x produces a leaf string $l(x)$ consisting of these transformations. Further we have a polynomial time computable function f, whose value is a subset of $\{1,2\}$. Our simulation first computes $f(x)$. Now, if $f(x)$ is the empty set, we reject. If $f(x)=\{1,2\}$, we accept. In the remaining cases $f(x)$ is either $\{1\}$ or $\{2\}$, and we have to check whether the product of the leaf string evaluated in $\hat{M}_2$ maps 1 to the single value in $f(x)$. But by queries to an NP oracle we can certainly find out, which monoid element in the leaf string is the last one, which is not the identity mapping (if any). Once we know that, we also know the value of the whole product, which suffices to check, whether $x\in L$. Thus L belongs to P^{NP}.

The reverse direction is Lemma 3.3 b).

$(M_2)\mathrm{P}$: Let L be a set in $(M_2)\mathrm{P}$, and let f and N be the according function and machine. Then, N on every input x produces a leaf string $l_N(x)$ from M_2^*, and $x\in L$ if and only if $l_N(x)$, evaluated in M_2, belongs to $f(x)$. A

$\mathrm{P}^{\exists:\oplus}\cdot\mathrm{P}$-machine simulates N as follows: First, compute $f(x)$. Then, by queries to an oracle from NP, find the rightmost letter in $l_N(x)$ which is a constant mapping. By one question to a $\oplus$P-oracle find out the result of $l_N(x)$ in M_2. This information, together with the value of $f(x)$ clearly suffices to check, whether $x \in L$.

The reverse direction is Lemma 3.5 a).

□

Now we will prove the analogous theorem for the case $k = 3$.

3.7 Theorem. $(\hat{M}_3')\mathrm{P} = \oplus\forall\mathrm{P}$ $\quad$ $(M_3')\mathrm{P} = \oplus(\forall \wedge \mathrm{MOD}_3\oplus)\mathrm{P}$

$(\hat{M}_3)\mathrm{P} = \mathrm{P}^{\exists:\oplus}\cdot\forall\mathrm{P}$ $\quad$ $(M_3)\mathrm{P} = \mathrm{P}^{\exists:\oplus}\cdot(\forall \wedge \mathrm{MOD}_3\oplus)\mathrm{P}$

Proof. We prove the four statements:

$(\hat{M}_3')\mathrm{P}$: The monoid $\hat{M}_3$ consists of all those mappings on $\{1,2,3\}$, which have at most two different values, and the identity. Now let L be a set in $(\hat{M}_3')\mathrm{P}$. Then there is a nondeterministic Turing machine N, which in every leaf either outputs the identity mapping, or a mapping with two different values. Moreover, two consecutive non-identity leaves have to fit to each other in the following sense: If the first one, m_1, has values a and b satisfying $a < b$, then the second one, m_2, satisfies $m_2(a) \neq m_2(b)$. If $m_2(a) > m_2(b)$, then we say that m_2 is an inverting element. (Note that m_2's property of being inverting depends on m_2 and on m_1.) Here, by consecutive non-identity leaves we mean two leaves with the property that in the computation tree m_1 appears to the left of m_2, and moreover, between these two leaves there are only leaves whose attached monoid element is the identity. Without loss of generality we can assume that the leftmost leaf is $\frac{1\ 2\ 3}{1\ 2\ 2}$, and that the rightmost leaf is $\frac{1\ 2\ 3}{1\ 2\ 2}$. We want to check whether value 1 is fixed under the action of the leaf string.

Now, a parity-computation on input x can do the following: Let $l = m_1 \ldots m_r$ be the leaf string of N on input x. Branch to all pairs p, q of values such that $1 \leq p < q \leq r$. Check that neither m_p nor m_q is an identity leaf, and that all m_i, where $p < i < q$ are identity leaves. The latter check is possible in $\forall$P, the others even in P. If one of the checks is unsuccessful, reject. Otherwise accept, if and only if m_q is an inverting leaf. This simulation will have an odd number of accepting computations, if and only if the leaf string as a whole is an inverting mapping, i.e. if and only if it maps 1 to 2 and 2 to 1. So we can check in $\oplus\forall$P, whether 1 is mapped to 1. Similarly we can check whether a is mapped to b for all $a, b \in \{1,2,3\}$. As $\oplus\forall$P is closed under complement, union and intersection, this proves the inclusion $(\hat{M}_3')\mathrm{P} \subseteq \oplus\forall\mathrm{P}$.

The reverse direction follows directly from the first statement in Lemma 3.3 c).

(M_3')P: Now we may also use nontrivial permutation elements of M_3. We proceed as in the claim for $(\hat{M}_3')$P, but we branch on all triples p, q, σ, where $1 \leq p < q \leq r$, and σ is an element of S_3. Then we check that neither m_p nor m_q is a permutation leaf, and that all m_i, where $p < i < q$ are permutation leaves, and additionally that the product $m_{p+1} \dots m_{q-1} = \sigma$. These checks can be done in $\forall\mathrm{P} \wedge \mathrm{MOD}_3\oplus\mathrm{P}$ (because $(S_3)\mathrm{P} = \mathrm{MOD}_3\oplus\mathrm{P}$). Now accept, if and only if all checks are successful, and σm_q is an inverting element. This proves that $(M_3')\mathrm{P} \subseteq \oplus(\forall \wedge \mathrm{MOD}_3\oplus)\mathrm{P}$.

The reverse direction follows directly from the first statement in Lemma 3.3 d).

$(\hat{M}_3)$P: This is very similar to the case $(\hat{M}_3')$P, but as now it is not forbidden to have a constant as value of the leaf string, we have to find the rightmost position q, such that $m_1 \dots m_q = c$ for some constant mapping c. Now, a constant mapping can be given either directly, or it can be forced by a sequence of two consecutive non fitting non-identity leaves. In any case we can find the rightmost occurrence of such a case by questions to an oracle in $\exists\forall\mathrm{P}$ (binary search), and then apply the $\oplus\forall\mathrm{P}$ simulation on the rest of the leaf string. This proves that $(\hat{M}_3)\mathrm{P} \subseteq \mathrm{P}^{\exists:\oplus} \cdot \forall\mathrm{P}$.

The reverse direction is Lemma 3.5 b).

(M_3)P: A combination of the techniques of the previous cases shows that $(M_3)\mathrm{P} \subseteq \mathrm{P}^{\exists:\oplus} \cdot (\forall \wedge \mathrm{MOD}_3\oplus)\mathrm{P}$.

The reverse direction is Lemma 3.5 c).

□

We proceed with $k = 4$:

3.8 Theorem.
$$(\hat{M}_4')\mathrm{P} = \oplus(\forall \wedge \mathrm{MOD}_3\oplus)\exists\mathrm{P}$$
$$(M_4')\mathrm{P} = \oplus(\forall \wedge \mathrm{MOD}_3\oplus)(\exists \vee \oplus\mathrm{MOD}_3\oplus)\mathrm{P}$$
$$(\hat{M}_4)\mathrm{P} = \mathrm{P}^{\exists:\oplus} \cdot (\forall \wedge \mathrm{MOD}_3\oplus)\exists\mathrm{P}$$
$$(M_4)\mathrm{P} = \mathrm{P}^{\exists:\oplus} \cdot (\forall \wedge \mathrm{MOD}_3\oplus)(\exists \vee \oplus\mathrm{MOD}_3\oplus)\mathrm{P}$$

Proof. The lower bounds follow directly from Lemma 3.3 e) and f) in the cases of the prime classes, and they are proven in Lemma 3.5 d) and e) for the other two classes. As to the upper bounds, we only provide the first result in detail. The others can be obtained combining the technique of this case and of the cases investigated in Theorem 3.7.

So let L be in $(\hat{M}_4')\mathrm{P}$ via machine N. We have to show that L is in $\oplus(\forall \wedge \mathrm{MOD}_3\oplus)\exists\mathrm{P}$. Assume that N's leaf string on input x is of the form $m_0' e e \dots e m_1 m_1' e e \dots e m_2 \dots$, where e is the identity mapping on four elements, there is at least one occurrence of e between m_{i-1}' and m_i, and the m_i and m_i' are mappings with exactly three different values, such that $m_i m_i' = m_i$ for all $i \geq 1$. This can easily be achieved. Now branch on all 5-tuples (p, q, p', q', σ) and accept if

and only if p and q are pointers to some m'_{i-1} and m_i, p' and q' are pointers to some m'_{j-1} and m_j, such that $i < j$, and moreover $m'_{i-1}m_i$ as well as $m'_{j-1}m_j$ are mappings with only two values, such that the whole substring of the leaf string starting at m'_i up to m_j is inverting in the sense of the proof of Theorem 3.7. Finally, the parity of the number of such inversions will tell us again, whether value 1 is fixed by the leaf string. It remains to show that all the checks can be carried out in $(\forall \wedge \mathrm{MOD}_3\oplus)\exists\mathrm{P}$. First we mention that the fact that p and q as well as p' and q' point to one group of the leaf string each, can be checked in $\forall\mathrm{P}$, as it is enough to check that all leaves in between p and q are identities, and the same for p' and q'. Then we have to check that each of these pairs of pointers defines a 2-value group, i.e. that the mappings $m'_{i-1}m_i$ and $m'_{j-1}m_j$ have two values each. This can be done deterministically in polynomial time. Third we have to check that there is no other 2-value group in between. For that we need a check in $\forall\exists\mathrm{P}$, because we have to check for all pairs p'' and q'', such that $q < p'' < q'' < p'$ the condition: if the according leaves would combine to a 2-value element of M_4, then there is another non-identity leaf between them. This is a typical Π_2^p-question. Now it remains to check that the product of all groups between the p, q group and the p', q' group viewed as a permutation on three elements has the guessed value σ: We branch to all pairs of pointers into this substring in an order-preserving way. Then we check, which element of S_3 the second pointer would realize, if it would be consecutive to the first one. Then we assume this element as value if all elements in between are identity, or we assume identity otherwise. This can be achieved by a $\mathrm{MOD}_3\oplus$-computation (for the elements from S_3), applied to a $\forall\mathrm{P}$ set (for the identity check). Thus all checks can be done either in $\forall\exists\mathrm{P}$ or in $\mathrm{MOD}_3\oplus\forall\mathrm{P}$, which equals $\mathrm{MOD}_3\oplus\exists\mathrm{P}$. The whole check then is possible in $(\forall \wedge \mathrm{MOD}_3\oplus)\exists\mathrm{P}$. □

3.9 Theorem. $(\hat{M}'_5)\mathrm{P} = \oplus(\forall \wedge \mathrm{MOD}_3\oplus)(\exists \vee \oplus\mathrm{MOD}_3\oplus)\forall\mathrm{P}$

$(\hat{M}_5)\mathrm{P} = \mathrm{P}^{\exists:\oplus}\cdot(\forall \wedge \mathrm{MOD}_3\oplus)(\exists \vee \oplus\mathrm{MOD}_3\oplus)\forall\mathrm{P}$

Proof. The lower bound for $(\hat{M}'_5)\mathrm{P}$ follows directly from Lemma 3.3 g), the one for $(\hat{M}_5)\mathrm{P}$ is Lemma 3.5 f). The upper bound proofs are identical to those given in the above cases for $k = 3$ and $k = 4$, only lifted up to another level. □

4. Applications

In this section we will apply the results of Section 3 to the cases of local self-reducibility and bottleneck Turing-machines. First we recall the definition of a k-local self-reducible set from [BeSt95].

Definition. Let $A \subseteq I\!N$, and let $k \geq 1$. A is called k-local self-reducible, if there is a number $n_0 \geq k$ and a polynomial-time computable function $f\colon I\!N^{\geq n_0} \to \{0,1\}^{(\{0,1\}^k)}$ such that

$$\forall n \in I\!N^{\geq n_0} \qquad \chi_A(n) = f(n)(\chi_A(n-k), \ldots, \chi_A(n-1)).$$

To obtain a smooth class we take the closure of the class of all k-local self-reducible sets under many-one reducibility:

Definition. The class k-LSR is defined by

$$k - \mathrm{LSR} = \{L \mid \exists A\colon L \leq^{\mathrm{p}}_m A \text{ and } A \text{ is } k\text{-local self-reducible}\}.$$

Besides the k-local self-reductions, in [BeSt95] also a many-one type of k-local self-reductions is introduced:

Definition. Let $A \subseteq I\!N$, and let $k \geq 2$. A is called m-k-local self-reducible, if there is a number $n_0 \geq k$ and a polynomial-time computable function $f\colon I\!N^{\geq n_0} \to \{1, \ldots, k\}$ such that

$$\forall n \in I\!N^{\geq n_0} \qquad \chi_A(n) = \chi_A(n - f(n)).$$

In analogy to the definition of k-LSR we introduce a smooth class for the many-one case, too:

Definition. The class k-$\mathrm{LSR_m}$ is defined by

$$k - \mathrm{LSR_m} = \{L \mid \exists A\colon L \leq^{\mathrm{p}}_m A \text{ and } A \text{ is m-}k\text{-local self-reducible}\}.$$

Remark.

1) We only require the respective relations for $n \geq n_0$ to make sure that the property of being locally self-reducible is robust against finite variation of the set.

2) In [BeSt95] results are of the form: all k-local self-reducible sets are in class $\mathcal{C}$, and there is a $\leq^{\mathrm{p}}_m$-complete set for $\mathcal{C}$ which is k-local self-reducible. In our setting this just reads k-LSR $= \mathcal{C}$.

3) The function f in the definition of k-local self-reducibility corresponds to a k-tt reduction, while the according function in the definition of m-k-local self-reducibility corresponds to a many-one reduction. Thus in the first case the value $f(n)$ has to be a k-ary boolean function, but in the second case a pointer to one of the k previous numbers suffices.

Now we relate these definitions to the classes (M_k)P and $(\hat{M}'_k)$P from Section 3:

4.1 Theorem. *k-LSR $= (M_{2^k})$P*

Proof. To show k-LSR $\subseteq (M_{2^k})$P it suffices to prove that all k-local self-reducible sets are in (M_{2^k})P, since trivially the latter class is closed under $\leq^{\mathrm{p}}_m$-reductions.

Now let A be k-local self-reducible via $n_0 \in \mathbb{N}$ and function f. Let c_i $(1 \leq i \leq n_0 - 1)$, the values of $\chi_A(i)$, be computed once and for all. On input n, an (M_{2^k})P-machine and a function f, witnessing $A \in (M_{2^k})$P can work as follows:

If $n < n_0$, then: Produce a trivial tree with one leaf, whose value is the identity function. If $c_n = 1$ then let $g(n) = \{1\}$, otherwise let $g(n) = \{2\}$.

If $n \geq n_0$, then: Produce a tree with $n - n_0 + 2$ leaves, one for each value i such that $n_0 \leq i \leq n$ (starting with n_0 on the left, ending with n on the right), and one more leaf to the left, which is called the "initial leaf".

Let every leaf produce a transformation on 2^k values, which are coded as k-bit strings, as its leaf value. The initial leaf produces the constant mapping d_0, satisfying $d_0(\alpha_1 \ldots \alpha_k) = \beta_1 \ldots \beta_k$, where $\beta_1 = \chi_A(n_0 - k), \ldots, \beta_k = \chi_A(n_0 - 1)$. For every $i \in \{n_0, n_0 + 1, \ldots, n\}$ let the according leaf produce the mapping d_i, where

$$d_i(\alpha_1 \ldots \alpha_k) = \alpha_2 \ldots \alpha_k \beta,$$

satisfying $\beta = f(i)(\alpha_1, \ldots, \alpha_k)$.

Since f is polynomial-time computable, also d_i is for given $i \in \{n_0, \ldots, n\}$, and moreover d_i can be viewed as an element of M_{2^k} for all $i \in \{n_0, \ldots, n\} \cup \{0\}$. Then the whole machine produces a computation tree with $n - n_0 + 2$ leaves, each of which is equipped with a transformation from M_{2^k}, and it is an easy exercise to check that $n \in A$ if and only if the sequence $d = d_0 d_{n_0} d_{n_0+1} \ldots d_n$, which obviously as a product in M_{2^k} yields a constant mapping, satisfies

$$d(\alpha_1 \ldots \alpha_k) = \gamma_1 \ldots \gamma_{k-1} 1$$

for each $\alpha_1, \ldots, \alpha_k$ and arbitrary values $\gamma_1, \ldots, \gamma_{k-1}$. Now define a subset of M_{2^k} by $B = \{c \mid c = \gamma_1 \ldots \gamma_k \text{ is a constant mapping, and } \gamma_k = 1\}$, then $n \in A \iff d \in B$, so choosing $g(x) = B$ we complete the proof that $A \in (M_{2^k})$P.

For the reverse direction, let $A \in (M_{2^k})$P. We have to construct a k-local self-reducible set B such that $A \leq^{\mathrm{p}}_m B$. Here, one difficulty arises from the fact that we have an independent computation tree for every input to the (M_{2^k})P-machine M deciding A, but we may only construct one universal characteristic sequence defining B. Thus we will proceed in stages: we choose a value for n_0, then a section of the characteristic sequence, say $\chi_B(n_0), \ldots, \chi_B(n_1 - 1)$ for the "first" string (the empty string), another section, say $\chi_B(n_1), \ldots, \chi_B(n_2 - 1)$ for the next string (string "0"), and so on. Thus on input x we virtually concatenate exponentially long sequences of the exponentially many (measured in the input size) strings that are less than x in lexicographic ordering, which still results in an exponentially long sequence. We only have to make sure that the sections for all strings of a given length m have an easily computable fixed length (like 2^m or 2^{2m}). Each section will start with a sequence of k mappings each of which has value 0. Thus the characteristic sequence in the beginning of the section for the i-th string x is $\chi_B(n_{i-1}) = \chi_B(n_{i-1}+1) = \cdots = \chi_B(n_{i-1}+k-1) = 0$, and we will view 0^k as the code for number 1 (in the coding of $\{1, \ldots, 2^k\}$ by k-bit strings). Let further $0^{k-1}1$ be the

code for number 2 and assume w.l.o.g. that the result of the leaf string of M on input x is either the constant 1 or the constant 2. Now we would like to map the whole leaf string $b_1, \ldots, b_r$ into the characteristic sequence directly. But unfortunately we obviously can only simulate mappings of the form $\alpha_1 \ldots \alpha_k \mapsto \alpha_2 \ldots \alpha_k \beta$ directly, but the b_μ $(1 \leq \mu \leq r)$ can be any element of M_{2^k}. Thus it remains to prove that all elements of M_{2^k} can be simulated by finite sequences of mappings of the form $\alpha_1 \ldots \alpha_k \mapsto \alpha_2 \ldots \alpha_k \beta$. For this we remark that M_{2^k} is generated by the following 2^k elements:

a) t_j $(1 \leq j \leq 2^k - 1)$,
where $t_j(j) = j + 1$, $t_j(j + 1) = j$, and $t_j(j') = j'$ if $j' \notin \{j, j + 1\}$.

b) s, where $s(j) = \max(j, 2)$ for all j.

It is clear that s can be simulated:

$$s = s_1 s_0 s_0 \ldots s_0$$

(with $k - 1$ times s_0), where $s_1(0^k) = 0^{k-1}1$, and $s_1(\alpha_1 \ldots \alpha_k) = \alpha_2 \ldots \alpha_k \alpha_1$, otherwise, and $s_0(\alpha_1 \ldots \alpha_k) = \alpha_2 \ldots \alpha_k \alpha_1$, in all cases.

For the t_j we assume that our coding of $\{1, \ldots, 2^k\}$ in k-bit strings is a Gray-code, i.e. two consecutive numbers differ only in one place. With the above example of s it should be clear how to simulate t_j.

Now we can build the values $b_1, \ldots, b_r$ into the characteristic sequence, each b_μ by a number $k \cdot l$ of characteristic bits, where l depends on the number of steps needed to simulate b_μ by elements t_j and s. We will always use the maximum l (inserting dummy elements, if necessary) to obtain a fixed a priori computable length of $r \cdot k \cdot l$. Then, if $n_{i+1} = n_i + r \cdot k \cdot l$, it is easy to compute $n_{i+1} - 1$, and obviously the input x belongs to A, if and only if $n_{i+1} - 1$ belongs to B, the set whose characteristic sequence is implicitly constructed in this proof, and which is obviously in k-LSR. This completes the proof of the theorem. □

Similarly we can show:

4.2 Theorem. $k\text{-}LSR_m = (\hat{M}'_k)\mathrm{P}$

Proof. The proof is quite analogous to the previous one. We only give a sketch of where the differences come from:

1) The self-reduction in the first case was a k-tt reduction with 2^k different values, which leads to M_{2^k}. Now our many-one reduction has only k possible places to refer to, this leads to M_k.

2) A constant mapping would mean that k consecutive elements all depend on the same one of the k previous elements. But then trivially all elements beyond these k elements either all are in the set, or all are not. Thus the considered set would be finite or co-finite, and thus in P. So it is sufficient (and necessary) to only allow leaf strings that do not evaluate to constant mappings, thus explaining the prime in $(\hat{M}'_k)\mathrm{P}$.

3) The implicit mapping $\alpha_1 \ldots \alpha_k \mapsto \alpha_2 \ldots \alpha_k \beta$ in a many-one fashion can only keep all the information if $\beta = \alpha_1$. Thus after k such steps we would obtain the same sequence $\alpha_1 \ldots \alpha_k$ again. This means, essentially the only allowed permutations are identity permutations, thus explaining the hat in $(\hat{M}'_k)$P.

The reader will be able to fill in the details. □

Combining these two theorems with the results in Section 3 we obtain the following corollaries:

4.3 Corollary.

a)
$$\begin{aligned}
\textit{1-LSR} &= \mathrm{P}^{\exists:\oplus}\cdot\mathrm{P}\\
\textit{2-LSR} &= \mathrm{P}^{\exists:\oplus}\cdot(\forall \wedge \mathrm{MOD}_3\oplus)(\exists \vee \oplus\mathrm{MOD}_3\oplus)\mathrm{P}\\
k\textit{-LSR} &= \mathrm{PSPACE}, \qquad (k \geq 3)
\end{aligned}$$

b)
$$\begin{aligned}
\textit{2-LSR}_m &= \mathrm{P}\\
\textit{3-LSR}_m &= \oplus\forall\mathrm{P}\\
\textit{4-LSR}_m &= \oplus(\forall \wedge \mathrm{MOD}_3\oplus)\exists\mathrm{P}\\
\textit{5-LSR}_m &= \oplus(\forall \wedge \mathrm{MOD}_3\oplus)(\exists \vee \oplus\mathrm{MOD}_3\oplus)\forall\mathrm{P}\\
k\textit{-LSR}_m &= \mathrm{PSPACE}, \qquad (k \geq 6)
\end{aligned}$$

Note that the equalities for k-LSR ($k \neq 2$) and k-LSR$_\mathrm{m}$ ($k \geq 6$) were already proven in [BeSt95].

It can easily be seen that the definitions of the classes SF_k and $shr\text{-}SF_k$ from [CaFu91] and [Og94] coincide with our classes (M_k)P and $(\hat{M}_k)$P. Thus we obtain:

4.4 Corollary.

a)
$$\begin{aligned}
SF_2 &= \mathrm{P}^{\exists:\oplus}\cdot\mathrm{P}\\
SF_3 &= \mathrm{P}^{\exists:\oplus}\cdot(\forall \wedge \mathrm{MOD}_3\oplus)\mathrm{P}\\
SF_4 &= \mathrm{P}^{\exists:\oplus}\cdot(\forall \wedge \mathrm{MOD}_3\oplus)(\exists \vee \oplus\mathrm{MOD}_3\oplus)\mathrm{P}\\
SF_k &= \mathrm{PSPACE}, \qquad (k \geq 5)
\end{aligned}$$

b)
$$\begin{aligned}
shr\text{-}SF_2 &= \Delta_2^p\\
shr\text{-}SF_3 &= \mathrm{P}^{\exists:\oplus}\cdot\forall\mathrm{P}\\
shr\text{-}SF_4 &= \mathrm{P}^{\exists:\oplus}\cdot(\forall \wedge \mathrm{MOD}_3\oplus)\exists\mathrm{P}\\
shr\text{-}SF_5 &= \mathrm{P}^{\exists:\oplus}\cdot(\forall \wedge \mathrm{MOD}_3\oplus)(\exists \vee \oplus\mathrm{MOD}_3\oplus)\forall\mathrm{P}\\
shr\text{-}SF_k &= \mathrm{PSPACE}, \qquad (k \geq 6)
\end{aligned}$$

Note that the equalities for SF_2, $shr\text{-}SF_2$, SF_k ($k \geq 5$), and $shr\text{-}SF_k$ ($k \geq 6$) have already been proven in [CaFu91] and [Og94], resp.

The classes $\mathrm{P}^{\exists:\oplus}\cdot(\forall \wedge \mathrm{MOD}_3\oplus)\mathrm{P}$ and $\mathrm{P}^{\exists:\oplus}\cdot(\forall \wedge \mathrm{MOD}_3\oplus)(\exists \vee \oplus\mathrm{MOD}_3\oplus)\mathrm{P}$ have nice closure properties, especially they are closed under boolean operations. Thus we can answer a question posed in [HeOg95]:

4.5 Corollary. *SF_3 and SF_4, and generally all SF_k and shr-SF_k classes, are closed under complement, union, and intersection.*

5. Conclusion, Extensions, and Problems

We completely characterized all classes of the form (M)P and (M')P, where M is one of the monoids $\hat{M}_k$ or M_k ($k \geq 2$). We solved all open questions concerning local self-reductions and many-one local self-reductions. We also obtained exact characterizations for the classes defined via so-called bottleneck Turing machines.

The results can even be slightly strengthened: As already mentioned, the classes k-LSR could be denoted by k-LSR$_{k\text{-tt}}$, because the actually occurring reduction type is a k-truth-table reduction. Thus we know from [BeSt95] (and from Corollary 4.3) that 3-LSR$_{3\text{-tt}}$ = PSPACE. In fact we can show:

5.1 Theorem. 3-$LSR_{2\text{-}tt}$ = PSPACE

Proof. It suffices to show that the permutation group S_5 can be simulated by 2-tt mappings in a 3-local self reduction. Let three consecutive bits of the characteristic sequence of a set code the values 1, 2, 3, 4, 5 as follows: 1 = 111, 2 = 110, 3 = 100, 4 = 001, 5 = 011. It is an easy exercise to show that the group S_5 is generated by the elements $\frac{1\ 2\ 3\ 4\ 5}{1\ 3\ 2\ 4\ 5}$, $\frac{1\ 2\ 3\ 4\ 5}{1\ 2\ 4\ 3\ 5}$, $\frac{1\ 2\ 3\ 4\ 5}{1\ 2\ 3\ 5\ 4}$, and $\frac{1\ 2\ 3\ 4\ 5}{3\ 1\ 2\ 4\ 5}$. We use the following 2-tt mappings: c_i for $i \in \{1,2,3\}$ maps the sequence $a_3a_2a_1$ to $a_2a_1a_i$. $e_{i,j}$ for $i,j \in \{1,2,3\}$ maps the sequence $a_3a_2a_1$ to a_2a_1x, where $x = 1$, if $a_i = a_j$, and $x = 0$, otherwise. $d_{i,j}$ for $i,j \in \{1,2,3\}$ maps the sequence $a_3a_2a_1$ to a_2a_1x, where $x = 1$, if $a_i \neq a_j$, and $x = 0$, otherwise. $v_{i,j}$ for $i,j \in \{1,2,3\}$ maps the sequence $a_3a_2a_1$ to a_2a_1x, where $x = 0$, if $a_i = a_j = 0$, and $x = 1$, otherwise. $u_{i,j}$ for $i,j \in \{1,2,3\}$ maps the sequence $a_3a_2a_1$ to a_2a_1x, where $x = 0$, if $a_i = 1$ and $a_j = 0$, and $x = 1$, otherwise. One can easily check that the sequence $c_3e_{2,3}c_3$ maps 111 to 111, 110 to 100, 100 to 110, 001 to 001, and 011 to 011. Thus this sequence realizes the generator $\frac{1\ 2\ 3\ 4\ 5}{1\ 3\ 2\ 4\ 5}$. Similarly, the sequence $e_{2,3}c_3e_{1,3}$ realizes the generator $\frac{1\ 2\ 3\ 4\ 5}{1\ 2\ 4\ 3\ 5}$, the sequence $c_3e_{1,3}c_3$ realizes the generator $\frac{1\ 2\ 3\ 4\ 5}{1\ 2\ 3\ 5\ 4}$, and finally $c_3d_{1,3}e_{1,3}c_3v_{2,3}u_{2,3}$ realizes the generator $\frac{1\ 2\ 3\ 4\ 5}{3\ 1\ 2\ 4\ 5}$. □

What do we know about 1-tt reductions? Certainly 6-LSR$_{1\text{-tt}}$ = PSPACE, and 1-LSR$_{1\text{-tt}}$ = 1-LSR = $\mathrm{P}^{\exists:\oplus}\cdot\mathrm{P}$. As an open question we would like to ask for a characterization of k-LSR$_{1\text{-tt}}$ for $k \in \{2,3,4,5\}$.

Another open question is the relation between (M_k)P and $(\hat{M}'_{k+1})$P. Certainly for $k = 2$ and $k \geq 5$ we have inclusion from left to right. But is that also true for $k = 3$ and $k = 4$? Note that (M_4)P contains $\exists\forall\oplus$P, and thus by Toda's result [To91] the whole polynomial time hierarchy. However, it would be surprising, if $(\hat{M}'_5)$P, which equals $\oplus(\forall \wedge \mathrm{MOD}_3\oplus)(\exists \vee \oplus\mathrm{MOD}_3\oplus)\forall$P, would contain the PH.

References

[BDG88] J. L. Balcázar, J. Díaz, J. Gabarró, *Structural Complexity I;* Springer (Berlin – Heidelberg – New York, 1988).

[BDG90] J. L. Balcázar, J. Díaz, J. Gabarró, *Structural Complexity II;* Springer (Berlin – Heidelberg – New York, 1990).

[Ba89] D. A. Barrington, Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1; *J. Comput. Syst. Sci.* **38** (1989), pp. 150–164.

[BaTh88] D. A. Mix Barrington, D. Thérien, Finite monoids and the fine structure of NC^1; *J.ACM* **35** (1988), pp. 941–952.

[BGH90] R. Beigel, J. Gill, U. Hertrampf, Counting classes: thresholds, parity, mods, and fewness; *Proceedings of the 7th Symp. on Theoretical Aspects of Computer Science* (1990), LNCS 415, pp. 49–57.

[BeSt95] R. Beigel, H. Straubing, The power of local self-reductions; *Proceedings of the 10th Structure in Complexity Theory Conference* (1995), pp. 277–285.

[BCS91] D. P. Bovet, P. Crescenzi, R. Silvestri, Complexity classes and sparse oracles; *Proceedings of the 6th Structure in Complexity Theory Conference* (1991), pp. 102–108.

[BCS92] D. P. Bovet, P. Crescenzi, R. Silvestri, A uniform approach to define complexity classes; *Theoretical Computer Science* **104** (1992), 263–283.

[CaFu91] J.-Y. Cai, M. Furst, PSPACE survives constant-width bottlenecks; *International Journal of Foundations of Computer Science* **2** (1991), pp. 67–76.

[CaHe89] J.-Y. Cai, L. Hemachandra, On the power of parity polynomial time; *Proceedings of the 6th Symp. on Theoretical Aspects of Computer Science* (1989), LNCS 349, pp. 229–239.

[GNW90] T. Gundermann, N.A. Nasser, G. Wechsung, A survey on counting classes; *Proceedings of the 5th Structure in Complexity Theory Conference* (1990), pp. 140–153.

[HeHo91] L. Hemachandra, A. Hoene, On sets with efficient implicit membership tests; *SIAM Journal on Computing* **20** (1991), pp. 1148–1156.

[HHH96a] E. Hemaspaandra, L. Hemaspaandra, H. Hempel, Query order in the polynomial hierarchy; Technical Report TR-634, University of Rochester, Dept. of CS., 1996.

[HHH96b] E. Hemaspaandra, L. Hemaspaandra, H. Hempel, $R^{SN}_{1-tt}(NP)$ distinguishes robust many-one and Turing completeness; Technical Report, University of Rochester, Dept. of CS., 1996.

[HHW95] L. A. Hemaspaandra, H. Hempel, G. Wechsung, Query order and self-specifying machines; Technical Report Math/95/13, Fak. f. Mathematik und Informatik, Friedrich-Schiller-Universität Jena, Germany (1995).

[HeOg95] L. A. Hemaspaandra, M. Ogihara, Universally serializable computation; Technical Report 520, Dept. of Computer Science, University of Rochester (1994).

[He90] U. Hertrampf, Relations among MOD-classes; *Theoretical Computer Science* **74** (1990), 325–328.

[He92a] U. Hertrampf, Locally definable acceptance types for polynomial time machines; *Proceedings of the 9th Symp. on Theoretical Aspects of Computer Science* (1992), LNCS 577, pp. 199–207.

[He92b] U. Hertrampf, Locally definable acceptance types – the three-valued case; *Proceedings of the 1st Latin American Symp. on Theoretical Informatics* (1992), LNCS 583, pp. 262–271.

[He94a] U. Hertrampf, Complexity classes with finite acceptance types; *Proceedings of the 11th Symp. on Theoretical Aspects of Computer Science* (1994), LNCS 775, pp. 543–553.

[He94b] U. Hertrampf, Complexity classes defined via k-valued functions; *Proceedings of the 9th Structure in Complexity Theory Conference* (1994), pp. 224–234.

[He95a] U. Hertrampf, Über Komplexitätsklassen, die mit Hilfe von k-wertigen Funktionen definiert werden; *Habilitationsschrift* Universität Würzburg (1995).

[He95b] U. Hertrampf, Classes of bounded counting type and their inclusion relations; *Proceedings of the 12th Symp. on Theoretical Aspects of Computer Science* (1995), LNCS 900, pp. 60–70.

[He96] U. Hertrampf, On the acceptance power of groups and semigroups; Technical Report 96-07, Fachbereich IV – Informatik, Universität Trier, Germany (1996).

[HLSVW93] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, K.W. Wagner, On the power of polynomial time bit reductions; *Proceedings of the 8th Structure in Complexity Theory Conference* (1993), pp. 200–207.

[Og94] M. Ogihara, On serializable languages; *International Journal of Foundations of Computer Science* **5** (1994), pp. 303–318.

[PaZa83] C. H. Papadimitriou, S. K. Zachos, Two remarks on the complexity of counting; *Proceedings of the 6th GI Conference on Theoretical Computer Science* (1983), LNCS 145, pp. 269–276.

[Pa94] C. H. Papadimitriou, *Computational Complexity,* Addison-Wesley, Reading, Mass. (1994).

[To91] S. Toda, PP is as hard as the polynomial-time hierarchy; *SIAM Journal on Computing* **20** (1991), pp. 865–877.