Inaugural Dissertation

zur

Erlangung der Doktorwürde

der

Naturwissenschaftlich-Mathematischen Gesamtfakultät

der

Ruprecht-Karls-Universität

Heidelberg

vorgelegt von

Diplom-Mathematiker Bernd Borchert

aus Lünne im Emsland

1994

Für meine Eltern

Predicate Classes, Promise Classes, and the Acceptance Power of Regular Languages

Gutachter: Prof. Dr. Klaus Ambos-Spies, Universität Heidelberg
Prof. Dr. Klaus W. Wagner, Universität Würzburg

Tag der mündlichen Prüfung: 20. Dezember 1994

Preface

This Ph.D. thesis was written in the time from January 1991 until July 1994 and is submitted to the Department of Mathematics of the University of Heidelberg. It contributes some results to Structural Complexity Theory which is a subfield of Theoretical Computer Science.

First of all I have to thank my advisor Prof. Klaus Ambos-Spies for his continuing guidance and support. He also gave several crucial hints for the results of this thesis.

Also I have to thank Prof. Juris Hartmanis and his former students Richard Chang, Suresh Chari, Desh Ranjan, and Pankaj Rohatgi. The initial results leading to this thesis were observed while I was visiting Cornell University in spring 1992.

This is the right place to express gratitude to Prof. Steven Homer and Prof. Akihiro Kanamori from Boston University who six years ago led me to the interesting field of Structural Complexity Theory. Also I would like to thank Prof. Klaus Weihrauch, University of Hagen, for supervising my first diploma thesis, and Prof. Wolfgang Schönfeld, IBM Heidelberg, for his guidance while I was working in his group.

For helpful discussions I would like to thank Andreas Eisenblätter, Ulrich Hertrampf, Birgit Jenner, Klaus-Jörn Lange, Pierre McKenzie, Wolfgang Merkle, Andre Nies, Thomas Schwentick, Nikolai Vereshchagin, and Heribert Vollmer.

I am grateful to Prof. Klaus W. Wagner, University of Würzburg, who agreed to referee this thesis.

The results of Part I of the thesis were presented at the 9th Annual IEEE Conference of Structure in Complexity Theory 1994, see [Bo94b], a journal version will be submitted. The results of Part II were presented at the 11th Annual Symposium of Theoretical Aspects of Computer Science (STACS) 1994, see [Bo94a], a journal version will appear in *Theoretical Computer Science*.

Heidelberg, July 1994

Contents

1 Introduction

Part I of this thesis observes a close connection between two basic concepts of Structural Complexity Theory, both introduced by Karp in [Ka72]:

1. The concept of *polynomial time many-one reducibility* which since its definition was studied intensively, see for example [La75, AS85a].

2. The concept of *polynomial time nondeterministic computation*, in the slightly more general sence as it is used to define not only the class NP (like in the original paper) but also classes like $\oplus$P, PP, UP, BPP, and RP.

Part II of this thesis relates the concepts of Part I to the notion of a *regular language*.

More detailed outlines of the two parts and references to related work are given below.

1.1 Outline of Part I

Several complexity classes – like NP, $\oplus$P, and PP – are defined (say *accepted*) by a predicate on computation trees produced by polynomial time nondeterministic Turing machine computations. Such classes will be called *predicate classes.* For example NP is accepted by the predicate on computation trees which is 1 if and only if the tree contains a leaf with label 1. As another example, $\oplus$P is accepted by the predicate on computation trees which is 1 if and only if the tree contains an odd number of leaves with label 1. Call a class a *principal ideal* if with respect to polynomial time many-one reducibility it has a complete set and is closed downward. It is well known that the example classes NP, $\oplus$P, and PP are principal ideals. This observation can be generalized:

- The set of predicate classes is equal to the set of principal ideals.

After the preliminary definitions and observations in Chapter 2 this theorem will be shown in Chapter 3.

In Chapter 4 complexity classes like UP, BPP, and RP will be considered. These classes have in common that their original definition can be seen the following way: there is a $\{0, 1, \perp\}$-valued function – called *promise function* – on computation trees where it is presumed (=’promised’) for each machine accepting a language in the class that for each input the promise function is not $\perp$ for the corresponding computation tree. Such classes will be called *promise classes*. For example UP is defined (say *accepted*) by the promise function on computation trees which has the value 0 if the tree does not contain a leaf with label 1, which has the value 1 if the tree contains exactly one leaf with label 1, and which has the value $\perp$ if the tree contains more than one leaf with label 1. Call a class an *ideal* if with respect to polynomial time many-one reducibility it is closed downward and closed under join. It is easy to see that the example classes UP, BPP, and RP are countable ideals. Like before, this observation can be generalized:

- The set of promise classes is equal to the set of countable ideals.

The two characterizations of predicate classes and promise classes described above – and their corresponding versions for the recursive case – are the two main results of Part I of this thesis. In Chapter 5 analogous results for some other models of nondeterministic computation will be shown.

1.2 Ouline of Part II

In Part II predicates with a low complexity will be considered: the predicates which are determined by a regular language for the the yields of computation trees (the *yield* is the left-to-right concatenation of the leaf labels). For example, NP is accepted by the predicate determined by the regular language L which consists of the words containing at least one letter 1.

The main result of Part II will be that if the class determined by a (nontrivial) regular language L is not equal to P then the class contains at least one of the classes NP, co-NP and MOD$_p$P for p prime.

This will be interpreted as a non-density result in two ways: (1) on the assumption that the Polynomial Time Hierarchy does not collapse, and (2) for the relativized case.

Additionally, the analog of the main result for the log-space case is shown.

1.3 Related Work

Similar work like in Part I was done in Bovet, Crescenzi, and Silvestri in [BCS91, BCS92], by Vereshchagin in [Ve93], by Hertrampf, Lautemann, Schwentick, Vollmer, and Wagner in [HL*93], and by Jenner, McKenzie, and Thérien in [JMT94]. Like in Part I of this thesis also in these papers the definability of complexity classes with the help of nondeterministic computation models is investigated.

The classes determined by regular languages, see Part II, were first considered by Hertrampf, Lautemann, Schwentick, Vollmer, and Wagner in [HL*93]. These classes are a special case (namely the associative case) of the classes determined by *locally definable acceptance types* defined by Hertrampf in [Her92a, Her94b]. On the other hand the *mod-classes* and the classes determined by *finite acceptance types*, considered systematicly in [Her90, Bei91, BG92] and [GW87, Her94a], respectively, are classes which are by definition determined by regular languages.

Part I
Predicate Classes and Promise Classes

2 Preliminaries

First some standard order-theoretic notions will be defined in Section 2.1. The well-known concept of *recursively presentable classes* will be defined in Section 2.2. Then the *polynomial time many-one reducibility* and its notions of *degrees*, *principal ideals* and *ideals* are presented in the Sections 2.3 – 2.6. Section 2.7 introduces *computation trees*.

2.1 Order-Theoretic Notions

The following order-theoretic notions are standard, see for example [Gr78].

A *binary relation* R on a set S is a subset of $S \times S$. Only the infix notation will be used, i.e. xRy stands for $(x, y) \in R$. A binary relation R is *reflexive* if xRx for all $x \in S$, it is *transitive* if from xRy and yRz it follows xRz, it is *symmetric* if from xRy it follows yRx, and it is *antisymmetric* if from xRy and yRx it follows $x = y$. A *preorder* is a reflexive and transitive binary relation on a nonempty set. A *partial order* is a preorder which is antisymmetric, and an *equivalence relation* is a preorder which is symmetric. A binary relation R on a set S and a binary relation R' on a set S' are called *isomorphic* if there exists an isomorphism, i.e. a bijective mapping i from S to S' such that $xRy \iff i(x)R'i(y)$.

Let $\sqsubseteq$ be a preorder on a set S. An element $s \in S$ is called $\sqsubseteq$*-complete* for a subset $T \subseteq S$ if $s \in T$ and $t \sqsubseteq s$ holds for all $t \in T$. A $\sqsubseteq$*-minimum* ($\sqsubseteq$*-maximum*) is an element $s \in S$ such that $s \sqsubseteq t$ ($t \sqsubseteq s$) for all $t \in S$. For two elements $s, s' \in S$ a $\sqsubseteq$*-supremum* ($\sqsubseteq$*-infimum*) of s and s' is an element $t \in S$ such that $s \sqsubseteq t$ and $s' \sqsubseteq t$ ($t \sqsubseteq s$ and $t \sqsubseteq s'$) and if also for another element $t' \in S$ it holds that $s \sqsubseteq t'$ and $s' \sqsubseteq t'$ ($t' \sqsubseteq s$ and $t' \sqsubseteq s'$) then $t \sqsubseteq t'$ ($t' \sqsubseteq t$). If it is clear from the context that one is dealing with a preorder $\sqsubseteq$, a $\sqsubseteq$–supremum will just be called *supremum*, this will be done the same way for other order-theoretic notions.

Let $\sqsubseteq$ be a partial order on a set S. Note that for a partial order the supremum (infimum) of two elements, if it exists, is unique. The partial order $\sqsubseteq$ is called an *upper semi-lattice* if the supremum exists for every pair of elements, it is called a *lattice* if both supremum and infimum exist for every pair of elements. A binary relation $\sqsubseteq'$ on a set S' is called an (upper semi-) sublattice of an (upper semi-) lattice $\sqsubseteq$ on a set S if S' is a subset of S, $\sqsubseteq'$ is the restriction of $\sqsubseteq$ to $S' \times S'$, and S' is closed under $\sqsubseteq$-suprema (and $\sqsubseteq$-infima). Note that an (upper semi-) sublattice is an (upper semi-) lattice. An (upper semi-) lattice is called *distributive* if for all elements $t, a, b \in S$ the following holds: if $t \sqsubseteq s$, where s is the supremum of a and b, then there are elements $a', b' \in S$ such that $a' \sqsubseteq a$, $b' \sqsubseteq b$, and t is the supremum of a' and b'.

Let $\sqsubseteq$ be a partial order on a set S. Two elements $s, s' \in S$ are called $\sqsubseteq$-comparable if $s \sqsubseteq s'$ or $s' \sqsubseteq s$. A subset $S' \subseteq S$ is called a $\sqsubseteq$-*chain* if any two elements of S' are comparable, S' is called an $\sqsubseteq$-*antichain* if any two elements are incomparable. If the minimum $m \in S$ exists then an element $s \not\equiv m$ is called an $\sqsubseteq$-*atom* if no element $t \not\equiv m, s$ exists such that $t \sqsubseteq s$. The partial order $\sqsubseteq$ is called *atomic* if for every element $t \not\equiv m$ there is an atom s such that $s \sqsubseteq t$. A partial order $\sqsubseteq$ is called *dense* if for any two comparable but different elements there is an element properly between them, formally: for all s, s' for which $s \sqsubseteq s'$ but not $s' \sqsubseteq s$ there is a t such that $s \sqsubseteq t$ and $t \sqsubseteq s'$ but neither $t \sqsubseteq s$ nor $s' \sqsubseteq t$. A partial order which contains an atom is obviously not dense because there is no element properly between the minimum and the atom.

2.2 Recursively Presentable Classes

In this thesis a *language* will always be a set of words over the alphabet $\Sigma = \{0, 1\}$, for basic definitions like the one of *words* see for example [HU79]. A *class* is a set of languages.

Let z_i be the $(i + 1)$st word of Σ^* in the length-lexicographic order, see for example [AS89]. For a language A and an $i \in \mathbb{N}$ define $A^{(i)}$ to be the language $\{x \mid \langle z_i, x\rangle \in A\}$ where $\langle , \rangle$ is a usual bijective polynomial time computable pairing function, see for example [BDG88]. Call, like in [BDG88, AS89], a complexity class C *recursively presentable* if $C = \{A^{(i)} \mid i \in \mathbb{N}\}$ for some recursive A, for the notion of a *recursive language* see for example [HU79].

2.3 Polynomial Time Many-One Reducibility

Let FP denote the class of functions $\Sigma^* \rightarrow \Sigma^*$ which can be computed by a Turing machine running in deterministic polynomial time, see for example [HU79, BDG88] for a more detailed definition. Let $\leq_m^p$ denote the polynomial time many-one reducibility among languages, i.e. $A \leq_m^p B$ if there exists a function $f \in$ FP such that $x \in A \iff f(x) \in B$ for all words x. The original definition of this reducibility is from Karp in [Ka72]. It is easy to see that the binary relation $\leq_m^p$ is a preorder on the set of all languages. Define the *join* $A \oplus B$ of two languages A, B to be the language $0A \cup 1B$. The join $A \oplus B$ is a $\leq_m^p$-supremum of A and B: $A, B \leq_m^p A \oplus B$ and for all languages C: $A, B \leq_m^p C \iff A \oplus B \leq_m^p C$.

The following enumeration of FP will be useful. Let in some straightforward way the deterministic Turing machines which compute functions $\Sigma^* \rightarrow \Sigma^*$ be encoded by words. Define for every i the function $f_i^p \in$ FP to be the function computed by the following polynomial time deterministic Turing machine: on input x the machine simulates the computation of the deterministic Turing machine encoded by z_i and cancels the simulation – with output ϵ – if the simulated machine has not terminated after $|x|^i + i$ steps. It is easy to see that FP = $\{f_i^p \mid i \in \mathbb{N}\}$ and that the function which maps $\langle z_i, x\rangle$ to $f_i^p(x)$ is recursive.

2.4 Polynomial Time Many-One Degrees

For the notions of this section see for example [La75, AS85a]. Two languages A, B are called *polynomial time many-one equivalent*, in short $A \equiv_m^p B$, if $A \leq_m^p B$ and $B \leq_m^p A$. Note that $\equiv_m^p$ is an equivalence relation on the set of all languages. Let the *polynomial time many-one degree of a language* A, in short $\deg_m^p(A)$, be the set of languages polynomial time many-one equivalent to A, and let $\leq_{m,deg}^p$ denote the partial order on the $\leq_m^p$-degrees defined by

$$\deg_m^p(A) \leq_{m,deg}^p \deg_m^p(B) :\iff A \leq_m^p B.$$

Note that this definition does not depend on the choice of A and B.

The degree $\deg_m^p(A \oplus B)$ is the unique $\leq_{m,deg}^p$-supremum of $\deg_m^p(A)$ and $\deg_m^p(B)$. This shows that $\leq_{m,deg}^p$ is an upper semi-lattice on the set of all polynomial time many-one degrees.

The two degrees $\{\emptyset\}$ and $\{\Sigma^*\}$ are called the *trivial* degrees. Call a degree $\deg_m^p(A)$ *recursive* if A and therefore all languages in $\deg_m^p(A)$ are recursive.

Many results are known for the partial order $\leq^p_{m,deg}$. In this thesis only the following basic results about density and distributivity due to Ladner in [La75] and Ambos-Spies in [AS85a], respectively, will be considered.

Theorem 2.1 (Ladner 1975, Ambos-Spies 1985a) *The following partial orders are distributive upper semi-lattices, the one in (b) is an upper semi-sublattice of the one in (a).*
(a) The partial order $\leq^p_{m,deg}$ on the set of all nontrivial polynomial time many-one degrees.
(b) The partial order $\leq^p_{m,deg}$ on the set of the nontrivial recursive polynomial time many-one degrees. This upper semi-lattice is additionally dense.

2.5 Principal Ideals

Call a set I of languages a *principal $\leq^p_m$-ideal*, or simply *principal ideal*, if there exists a language A such that $I = \leq^p_m(A) := \{B \subseteq \Sigma^* \mid B \leq^p_m A\}$. There are several other names for the class $\leq^p_m(A)$, sometimes it is called *lower cone of A* or *downward closure of A*. For the choice of the name *principal ideal* see the next section. Note that the language A is $\leq^p_m$-complete for $\leq^p_m(A)$.

The following classes are examples of principal ideals.

- The class NP should be mentioned first as an example of a principal ideal. Complete languages for NP, like the problem SAT, were – for polynomial time Turing reducibility – first presented by Cook in [Co71], their $\leq^p_m$-completeness was shown in [Ka72, Le73]. For a list of $\leq^p_m$-complete problems for NP see [GJ79]. The fact that for example SAT is not only $\leq^p_m$-complete for NP but also every language $\leq^p_m$-reducible to SAT is in NP is easy to see.

- The two principal ideals $\{\emptyset\} = \leq^p_m(\emptyset)$, $\{\Sigma^*\} = \leq^p_m(\Sigma^*)$ will be called *trivial* principal ideals.

- The class P is a principal ideal $\leq^p_m(A)$ where A is any language in P $-\{\emptyset, \Sigma^*\}$. P contains the two trivial principal ideals and is contained in every nontrivial principal ideal, see Figure 1.

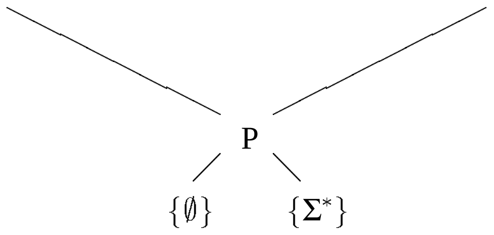


Figure 1: The two trivial principal ideals and P

- Not only P and NP but also all other classes Σ_n^p and Π_n^p of the Polynomial Time Hierarchy are principal ideals, the existence of $\leq_m^p$-complete languages was shown in [St77, Wr77].

- Let X be any language. Then the class P^X consisting of all languages computable in polynomial time with oracle X (see for example [Co71, BDG88] and also Section 5.5) is a principal (many-one) ideal according to the results in [AS86a], see also Corollary 5.8. As a special case, the classes Δ_n^p of the Polynomial Time Hierarchy are principal ideals.

- The classes NP(n) and co-NP(n) of the Boolean Hierarchy are principal ideals, the existence of $\leq_m^p$-complete languages was shown in [CG*88].

- Counting classes like PP, $\mathrm{C_{=}P}$, $\mathrm{MOD}_n\mathrm{P}$, $\oplus\mathrm{P} = \mathrm{MOD_2P}$, US = 1–NP are principal ideals, for the original definitions see [Gi77, Wa86b, BG92, PZ83, BGu82, GW87].

- The exponential time classes EXPTIME = DTIME(2^{poly}) and NEXPTIME = NTIME(2^{poly}) are principal ideals. More generally, the classes k-EXPTIME = DTIME($2^{\cdot^{\cdot^{\cdot^{2^{\mathrm{poly}}}}}}$) and k-NEXPTIME = NTIME($2^{\cdot^{\cdot^{\cdot^{2^{\mathrm{poly}}}}}}$), where in both cases the exponentiation tower has height k, can easily be shown to be principal ideals for every $k \geq 1$. For the exact definitions see for example [Jo90].

There is a strong connection between the inclusion order on the principal ideals and the partial order $\leq_m^p$ on the polynomial time many-one degrees.

Proposition 2.1 *For all languages A, B it holds:*

$$\leq_m^p(A) \subseteq \leq_m^p(B) \iff A \leq_m^p B \iff \mathrm{deg}_m^p(A) \leq_{m,deg}^p \mathrm{deg}_m^p(B).$$

Proof. Note that the second equivalence holds by the definition of $\leq_{m,deg}^p$. In order to see the first equivalence assume that $\leq_m^p(A) \subseteq \leq_m^p(B)$. Because $A \in \leq_m^p(A)$ it holds by the assumption that $A \in \leq_m^p(B)$, this shows $A \leq_m^p B$. If on the other side $A \leq_m^p B$ then for each $C \leq_m^p A$ it holds by the transitivity of $\leq_m^p$ that $C \leq_m^p B$, this shows $\leq_m^p(A) \subseteq \leq_m^p(B)$ □

In other words, the inclusion order on the principal ideals is isomorphic to the $\leq_m^p$-order on the polynomial time many-one degrees. It is clear by the proof that this isomorphism between the partial order on the degrees and the inclusion order on the principal ideals exists not only for $\leq_m^p$ but for every preorder.

The following corollary follows immediately from Proposition 2.1.

Corollary 2.1 *For all languages A, B it holds:*

$$\leq_m^p(A) = \leq_m^p(B) \iff A \equiv_m^p B \iff \mathrm{deg}_m^p(A) = \mathrm{deg}_m^p(B).$$

By the following proposition the property of being recursively presentable is determined for a principal ideal by any $\leq_m^p$-complete language.

Proposition 2.2 *Let A be a language. $\leq_m^p(A)$ is recursively presentable $\iff$ A is recursive $\iff$ $\mathrm{deg}_m^p(A)$ is recursive.*

Proof. Note that the second equivalence was already mentioned in the definition of the recursiveness of a $\leq_m^p$-degree.

In order to see the first equivalence assume that $\leq_m^p(A) = \{B^{(i)} \mid i \in \mathbb{N}\}$ for some recursive language B. Then $A = B^{(j)}$ for some $j \in \mathbb{N}$. But if B is recursive then also $B^{(j)} = A$ is.

For the other direction let a recursive language A be given. Define the language $C := \{\langle z_i, x\rangle \mid f_i^p(x) \in A\}$, the functions f_i^p were defined in Section 2.3. It is easy to see that C is recursive and that $\leq_m^p(A) = \{B \mid B \leq_m^p A\} = \{B \mid \exists i \in \mathbb{N} :$

$\forall x : x \in B \iff f_i^p(x) \in A\} = \{C^{(i)} \mid i \in \mathbb{N}\}$. This finishes the proof of the fact that the class $\leq_m^p(A)$ is recursively presentable if and only if A is recursive. □

By Propositions 2.1 and 2.2 the results for the polynomial time many-one degrees stated in Theorem 2.1 can be transfered to the principal ideals immediately.

Corollary 2.2 *The following partial orders are distributive upper semi-lattices. The one in (b) is an upper semi-sublattice of the one in (a).*

(a) The inclusion order on the set of all nontrivial principal ideals.

(b) The inclusion order on the set of all nontrivial recursively presentable principal ideals. This upper semi-lattice is additionally dense.

2.6 Ideals

A $\leq_m^p$*-ideal*, or simply *ideal*, is a nonempty set I of languages such that if languages A and B are in I then each language C with $C \leq_m^p A \oplus B$ is also in I. In other words, an ideal is a nonempty set of languages which is closed under join und closed downward. The name *ideal* follows the notation in Lattice Theory, see for example Grätzer [Gr78].

The following proposition shows the relation between between ideals and principal ideals.

Proposition 2.3 *(a) The principal ideals are exactly the ideals which have a $\leq_m^p$-complete language. (b) The recursively presentable principal ideals are exactly the ideals which have a recursive $\leq_m^p$-complete language.*

Proof. (a) Let a principal ideal $\leq_m^p(A)$ be given. A is $\leq_m^p$-complete for $\leq_m^p(A)$ because A is in $\leq_m^p(A)$ and by definition all languages in $\leq_m^p(A)$ are $\leq_m^p$-reducible to A. It remains to show that $\leq_m^p(A)$ is an ideal. Let $B, B' \in \leq_m^p(A)$ and $C \leq_m^p B \oplus B'$. Then $C \leq_m^p B \oplus B' \leq_m^p A$ by the supremum property of the join. By the transitivity of $\leq_m^p$ it follows that $C \in \leq_m^p(A)$. Therefore, $\leq_m^p(A)$ is an ideal.

For the other direction let an ideal I have a $\leq_m^p$-complete language A. It will be shown that $I = \leq_m^p(A)$. It holds $I \subseteq \leq_m^p(A)$ because every language in I is $\leq_m^p$-reducible to A. And it holds $\leq_m^p(A) \subseteq I$ because $A \in I$ and I is closed downward.

Part (b) follows from part (a) and Proposition 2.2. □

The two trivial principal ideals $\{\emptyset\}$ and $\{\Sigma^*\}$ will be also be called *trivial ideals*. Like in the case of principal ideals it is easy to see that the ideal P contains the two trivial ideals and every nontrivial ideal contains P.

Some examples of classes are given which are ideals but which are not principal ideals or not known to be principal ideals.

- Classes like UP (defined in [Va76]), BPP, RP (both defined in [Gi77]), FewP (defined as FNP in [Al86]), and AM (defined in [Ba85]), are easily shown to be (recursively presentable) ideals. These classes are not known to be principal ideals, see [Si82, Kow84, AS86b, HH88, Hem88, AS89].

- In Proposition 2.5 it will be shown that pairwise intersections of nontrivial (recursively presentable) ideals are nontrivial (recursively presentable) ideals, like $\Sigma_n^p \cap \Pi_n^p$ (for $n \geq 1$) or ZPP = RP $\cap$ co-RP. Generally, it is not known if such an intersection is a principal ideal. For a discussion of this question for the class NP $\cap$ co-NP see [Si82, Kow84, HI85, Hem88, AS89].

- (Effective) infinite unions of increasing sequences of (recursively presentable) ideals, like the class of languages of the Polynomial Time Hierarchy PH= $\bigcup_{n\in\mathbb{N}} \Sigma_n^p$ and the class of languages of the Boolean Hierarchy BH= $\bigcup_{n\in\mathbb{N}}$NP(n), are (recursively presentable) ideals which are in general not known to be principal ideals. For the original definitions of PH and BH see [St77, Wr77, CG*88].

- Let the class ELEMENTARY be the union $\bigcup_{k\geq 1}$ k-EXPTIME, for the definition of the classes k-EXPTIME for $k \geq 1$ see the examples of principal ideals in Section 2.5. The class ELEMENTARY is a (recursively presentable) ideal which is provably not a principal ideal because by the Time Hierarchy Theorem of [HS65] it can be shown for each $k \geq 1$ that k-EXPTIME is a proper subset of (k + 1)-EXPTIME.

- The class of all recursive languages is a countable ideal but neither a principal ideal nor a recursively presentable ideal.

- The class P/poly defined by Karp and Lipton in [KL80, KL82] can easily be shown to be an ideal. It is not countable, but for example P/poly $\cap$ NP is a countable ideal.

- The class of all languages is an ideal but not countable.

The following classes are not ideals.

- The class E = DTIME(2^{lin}) is recursively presentable, has a $\leq^p_m$-complete language, and is closed under join but is not an ideal because it is not closed downward.

- A polynomial time many-one degree is an ideal if and only if it is one of the two trivial degrees because otherwise it is not closed downward.

- For two $\leq^p_m$-incomparable recursive languages A, B the class $\leq^p_m(A) \cup \leq^p_m(B)$ is recursively presentable and closed downward but is not an ideal because it is not closed under join.

The relation of the different types of ideals introduced so far is described by the following Proposition 2.4, see also Figure 2.

Proposition 2.4 *(a) Every recursively presentable ideal is a countable ideal. (b) Every principal ideal is a countable ideal. (c) There is a recursively presentable ideal which is not a principal ideal. (d) There is a principal ideal which is not recursively presentable. (e) There is an ideal which is not countable.*

Proof. (a) Every recursively presentable class is by definition countable. (b) A principal ideal $\leq^p_m(A)$ is by Proposition 2.3 an ideal, and it is countable because there are at most countably many $\leq^p_m$-reductions. A witness for (c) is the class ELEMENTARY, see the examples above, and a witness for (d) is according to Propositions 2.2 and 2.3 any class $\leq^p_m(A)$ for a non-recursive A. A witness for (e) is the class P/poly, see the examples above. □

Proposition 2.5 *The following partial orders are distributive lattices, the one in (b) is a sublattice of the one in (a), and the one in (c) is a sublattice of the ones in (a) and (b).*

(a) The inclusion order on the set of all nontrivial ideals.

(b) The inclusion order on the set of all nontrivial countable ideals.

(c) The inclusion order on the set of all nontrivial recursively presentable ideals.

In (a), (b) and (c) the infimum of two nontrivial ideals I and J is given by their intersection, and the supremum is given by the smallest ideal containing both I and J.

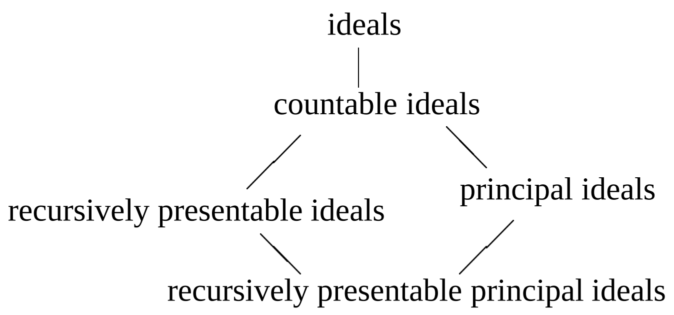

ideals
countable ideals
recursively presentable ideals
principal ideals
recursively presentable principal ideals

Figure 2: Types of Ideals

Proof. (a) Let two nontrivial ideals I and J be given. It will be shown that $I \cap J$ is a nontrivial ideal. Therefore, $I \cap J$ is the infimum of I and J. Because both I and J contain the class P also the class $I \cap J$ contains P and is not empty. Let $C \leq_m^p A \oplus B$ for two languages $A, B \in I \cap J$, then both I and J contain A and B. Therefore, they contain also C by their property of being an ideal. This shows that $I \cap J$ is a nontrivial ideal. The supremum of I and J is the class $H = \{C \mid \exists A \in I, \exists B \in J : C \leq_m^p A \oplus B\}$. It is easy to see that this class is an ideal, that it contains both I and J, and that it is contained in every ideal containing both I and J. For the distributivity let I, J, H be as above and let K be contained in H. Now it is shown that the supremum of $I' = K \cap I$ and $J' = K \cap J$ equals K. I' and J' and therefore also their supremum are contained in K. For the other direction it obviously suffices to show that for every language A in K the principal ideal $\leq_m^p(A)$ is the supremum of two principal ideals in I and J, respectively. By definition of H there exist sets $B \in I, C \in J$ such that $A \leq_m^p B \oplus C$, in other words $\leq_m^p(A)$ is contained in the supremum of $\leq_m^p(B)$ and $\leq_m^p(C)$. By the distributivity of the principal ideals there are languages $B' \in \leq_m^p(B), C' \in \leq_m^p(C)$ such that $\leq_m^p(A)$ is the supremum of $\leq_m^p(B')$ and $\leq_m^p(C')$.

(b) By the property of a sublattice to be a lattice it suffices to observe that for two given countable nontrivial ideals I and J the classes $I \cap J$ and H from (a) are countable. The distributivity follows like in (a).

(c) It suffices like in (b) to show that for two recursively presentable ideals $I = \{C^{(i)} \mid i \in \mathbb{N}\}$ and $J = \{D^{(j)} \mid j \in \mathbb{N}\}$ (for recursive languages C and D)

the classes $I \cap J$ and H from (a) are also recursively presentable. Define E to be the recursive language $\{\langle\langle z_i, z_j\rangle, x\rangle \mid x \in C^{(i)}$ and for all y with $|y| \leq |x|$ it holds that $\langle z_i, y\rangle \in C \iff \langle z_j, y\rangle \in D\}$. It will be shown that this construction guarantees that $I \cap J = \{E^{(i)} \mid i \in \mathbb{N}\}$. The inclusion from left to right is obvious. For the other direction consider a fixed $E^{(j)}$: if $E^{(j)}$ is infinite, then it is also an (infinite) language of both I and J, and if $E^{(j)}$ is finite then it is in P and therefore in $I \cap J$. This shows $I \cap J = \{E^{(i)} \mid i \in \mathbb{N}\}$. To represent the class H define F to be the language $\{\langle\langle\langle z_i, z_j\rangle, z_k\rangle, x\rangle \mid f_k^p(x) \in C^{(i)} \oplus D^{(j)}\}$, where f_k^p was defined in Section 2.3. It is easy to check that F is recursive, and by construction it holds that $H = \{F^{(i)} \mid i \in \mathbb{N}\}$. The distributivity follows like in (a). □

The following two results – called *exact pair theorems* – of Ambos-Spies in [AS86b] and Shinoda and Slaman in [ShS90] relate the notions of ideals and principal ideals more closely. The latter was shown to hold for the polynomial time Turing reducibility but the proof is also valid for the many-one case.

Theorem 2.2 (Ambos-Spies 1986) *For a recursively presentable ideal I there exist two recursive languages A and B such that $I = \leq_m^p(A) \cap \leq_m^p(B)$.*

Theorem 2.3 (Shinoda & Slaman 1990) *For a countable ideal I there exist two languages A and B such that $I = \leq_m^p(A) \cap \leq_m^p(B)$.*

For the nontrivial ideals the two Theorems 2.2 and 2.3 can by Proposition 2.5 be expressed the following way.

Proposition 2.6 *(a) The nontrivial countable ideals are exactly the pairwise intersections of the nontrivial principal ideals. (b) The nontrivial recursively presentable ideals are exactly the pairwise intersections of the nontrivial recursively presentable principal ideals.*

2.7 Computation Trees

Consider nondeterministic Turing machines as presented for example in [BDG88]. In this thesis it is additionally assumed for a Turing machine that for a state and a tupel of symbols read by the heads at most two transitions are specified by the transition function, and also it is assumed that the transition function is given as a linear list. This way it is guaranteed that if nondeterminism appears during a

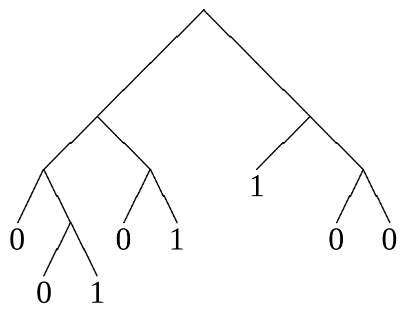


Figure 3: Computation tree

computation the computation branches into exactly two independent computations which can be distinguished as the left computation and the right computation.

Let a polynomial time nondeterministic Turing machine be a nondeterministic Turing machine M (of the special kind above) for which there is a polynomial p such that M computes for each input x on every computation path at most $p(|x|)$ steps.

Let a *computation tree* be a – not necessarily balanced – ordered binary tree (i. e. each inner node has a left subtree and a right subtree) where the inner nodes have no labels, and the leaves are labeled with 0 or 1. A formal definition of ordered trees is given for example in [HU79]. An example of a computation tree is shown in Figure 3.

It is clear that each polynomial time nondeterministic Turing machine M on an input x produces a computation tree $T(M,x)$ by starting at the root, adding a binary node – for the left and the right computation – each time a nondeterministic branching is encountered, and writing in the case of termination a 0 (for rejecting) or a 1 (for accepting) on the corresponding leaf.

Note that for a computation only the nondeterministic steps and the output bits are recorded in the computation tree, not the deterministic steps and also not any information about the configurations.

The following definition is similar to the definition of the functions f_i^p in Section 2.3. Let nondeterministic Turing machines (of the special kind above) be coded in some straightforward way by words of Σ^*. Therefore, each word z_i can be assumed to describe a nondeterministic Turing machine. Let M_i^p be the Turing machine which simulates on input x the machine described by z_i with the time bound $|x|^i + i$, i.e. it cancels – with a rejecting state – on every computation path the computation after $|x|^i + i$ steps if the computation has not already terminated on that path. Note that this enumeration M_i^p (for $i \in \mathbb{N}$) of nondeterministic Turing machines has the following properties: M_i^p is a polynomial time nondeterministic Turing machine for every i, and for every polynomial time nondeterministic Turing machine M there is an i such that $T(M, x) = T(M_i^p, x)$ for all x.

The following definition will be used only for examples, not for results. For a computation tree T let the rational number $\pi(T) \in [0, 1]$ be the probability to reach a leaf with label 1 if one moves from the root of T to a leaf, tossing a coin on every inner node. For example, the computation tree in Figure 3 has the π-value $\frac{7}{16}$.

3 Predicate Classes

In this chapter the notion of a *predicate class* will be defined and will be shown to be equivalent to the notion of a principal ideal.

3.1 The Definition of Predicate Classes

Let a *predicate* (for computation trees) be a function from the set of computation trees to the set $\{0, 1\}$. Note that a predicate could also be considered as a tree language.

Definition 3.1 *For a predicate F and a polynomial time nondeterministic Turing machine M define the language $L_F(M)$ by*

$$x \in L_F(M) \iff F(T(M, x)) = 1.$$

Let the predicate class accepted by F, *short* F – P, *be the set of languages $L_F(M)$ for which M is a polynomial time nondeterministic Turing machine.*

In other words, membership of x in $L_F(M)$ is decided the following way: construct the computation tree produced by M on input x, and let x be in $L_F(M)$ if and only if the F-value of the computation tree is 1.

The following examples, especially the first, may clarify the definition above.

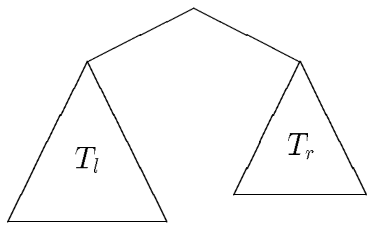


Figure 4: The left and the right subtree

- NP is the set of languages L for which there is a polynomial time nondeterministic Turing machine M such that $x \in L$ if and only if there is an accepting computation path of M on input x, or equivalently, $x \in L$ if and only if in $T(M, x)$ there is a leaf with label 1. Therefore, NP is accepted by the predicate F_1 which is 1 for a computation tree T if and only if T has a leaf with label 1, in other words, NP = F_1 – P.

- The class co-NP is the set of complements of languages in NP. Therefore, co-NP is accepted by the predicate which is 1 for a computation tree T if and only if all leaves in T have label 0.

- P can easily be shown to be accepted by the predicate F_l for computation trees which is 1 if and only if the leftmost leaf in the tree has label 1.

- By definition of the class $\oplus$P in [PZ83] it holds $\oplus$P = F_{odd} – P where $F_{odd}(T)$ = 1 if and only if for the tree T the number of leaves with label 1 is odd.

- By definition in [Gi77] the class PP is accepted by the predicate which is 1 for a tree T if and only if $\pi(T) > \frac{1}{2}$, for the definition of the function π see Section 2.7. It is easy to show that PP is also accepted by the (different) predicate F_{maj} which is 1 for a tree T if and only if there are more leaves with label 1 than leaves with label 0.

- By the alternation characterization of PSPACE in [CKS81] one has PSPACE = F_a – P, where $F_a(T)$ is the value of the Boolean evaluation of the formula given by T for which the inner nodes are alternatingly interpreted as conjunction and disjunction gates.

- The class D^P = NP(2), originally defined in [PY82], is the class consisting of the languages which are an intersection of a language in NP and a language in co-NP. The class D^P can easily be shown to be accepted by the following predicate F_D: for a single-leaf tree F_D has the (arbitrary) value 0, and for a non-single-leaf tree T F_D has the value 1 if and only if the left subtree T_l – see Figure 4 – has a leaf with label 1 and the right subtree T_r does not have a leaf with label 1.

- The constructions in [Her92a] imply definitions for many predicates which accept well-known complexity classes, for example the Σ_n^p-, Π_n^p- and Δ_n^p-classes of the Polynomial Time Hierarchy.

By a usual encoding of trees into words one can consider a recursive function on words to be a recursive function on computation trees and vice versa. Let the set of *recursive predicates* be the set of the recursive functions from computation trees to $\{0, 1\}$ and call a predicate class *recursive* if it is accepted by some recursive predicate. All examples of predicate classes given above are recursive.

3.2 The Characterization of Predicate Classes

Call a predicate class *trivial* if it is accepted by one of the two constant predicates.

Proposition 3.1 *The following sets of classes are equal: (a) the trivial predicate classes, (b) the trivial principal ideals, (c) the trivial recursive predicate classes, (d) the trivial recursively presentable principal ideals.*

Proof. If F is the constant-0 predicate then for a every polynomial time nondeterministic Turing machine M the language $L_F(M)$ is empty. Therefore, F – P = $\{\emptyset\}$ = $\leq_m^p(\emptyset)$. Likewise, F – P = $\{\Sigma^*\}$ = $\leq_m^p(\Sigma^*)$ if F is the constant-1 predicate. This shows already the equality of the sets in (a) and (b). For (c) and (d) it suffices to observe that the two trivial predicate classes are recursive and that the two trivial principal ideals are recursively presentable. $\square$

The first main result is stated.

Theorem 3.1 *(a) The predicate classes are exactly the principal ideals. (b) The recursive predicate classes are exactly the recursively presentable principal ideals.*

Note that the direction from left to right of part (a) of the theorem says that every predicate of any recursive or non-recursive complexity accepts a complexity class which has the 'nice' properties of a principal ideal: with respect to $\leq_m^p$ it has a complete set and is closed downward.

Proof. First part (a) with its two directions will be proven, part (b) will follow easily.
Proof of part (a), direction $\subseteq$: *Every predicate class is a principal ideal.*

Fix a predicate F. By Proposition 3.1 it can be assumed w.l.o.g. that F is not constant-1. By the properties of the machines M_i^p, see Section 2.7, one has the following enumeration of $F-\mathrm{P}$:

$$F-\mathrm{P} = \{L_F(M_i^p) \mid i \in \mathbb{N}\}.$$

For the predicate F a language K_F will be defined the following way (like this was done for NP in [BGS75, Har78, BDG88]):

$$K_F := \{\langle z_i, x, 0^t\rangle \mid t = |x|^i + i \text{ and } F(T(M_i^p, x)) = 1\}.$$

It will be shown that for all F

$$F-\mathrm{P} = \leq_m^p(K_F).$$

It will be observed first that K_F is an element of $F-\mathrm{P}$: consider the following polynomial time nondeterministic Turing machine M_u : for an input y M_u first checks if y encodes a triple $\langle z_i, x, 0^t\rangle$ with $t = |x|^i + i$; if this is not the case then M_u produces by nondeterminism a computation tree T with $F(T) = 0$, otherwise it simulates the computation of M_i^p on input x for t steps, branching each time M_i^p branches. By construction the computation tree $T(M_u, \langle z_i, x, 0^t\rangle)$ is identical to the computation tree $T(M_i^p, x)$. Therefore, $\langle z_i, x, 0^t\rangle \in L_F(M_u) \iff F(T(M_u, \langle z_i, x, 0^t\rangle)) = 1 \iff F(T(M_i^p, x)) = 1 \iff \langle z_i, x, 0^t\rangle \in K_F$ by definition of K_F. This means $L_F(M_u) = K_F$ and therefore $K_F \in F-\mathrm{P}$. Now the above equality $F-\mathrm{P} = \{L_F(M_i^p) \mid i \in \mathbb{N}\} = \leq_m^p(K_F)$ is easy to see: Let $L_F(M_i^p)$ be a language in $F-\mathrm{P}$, and remember that the running time of M_i^p on

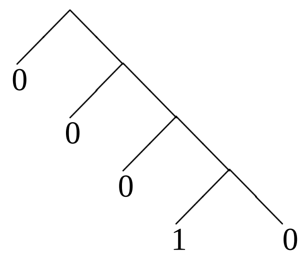


Figure 5: Comb, encoding 0001

input x is bounded by $|x|^i + i$. Consider the function g which for an input x computes in polynomial time the tripel $g(x) := \langle z_i, x, 0^{|x|^i+i}\rangle$. By the definition of K_F the function g is a polynomial time many-one reduction from $L_F(M_i^p)$ to K_F. Therefore, $L_F(M_i^p) \in \leq_m^p(K_F)$. Let for the other direction of the above equation $B \in \leq_m^p(K_F)$, i.e. $B \leq_m^p K_F$ via a function $f \in$ FP. Then the polynomial time nondeterministic Turing machine M_f which for an input x first computes $f(x)$ and then runs M_u on input $f(x)$ shows that $B \in F-$P because by construction $x \in B \iff f(x) \in K_F \iff x \in L_F(M_f)$. This finishes the proof of the above equation.

Proof of part (a), direction $\supseteq$: *Every principal ideal is a predicate class.*

Define a *comb* to be a computation tree which has the special form that the left successor of each inner node is a leaf. The *word encoded by a comb* is the word consisting of the sequence of leaf labels of these left successors of inner nodes, starting at the top, see Figure 5.

For a language A let G_A be the predicate which is 1 for a tree if and only if the tree is a comb which encodes a word from A.

To prove that every principal ideal is a predicate class it suffices to show that for all languages $A \neq \Sigma^*$

$$\leq_m^p(A) = G_A - \mathrm{P}.$$

Note that the case $A = \Sigma^*$ is already covered by Proposition 3.1.

In order to show the inclusion from left to right of the above equation let $B \in \leq_m^p(A)$ be given, i.e. B is many-one reducible to A via a function $g \in$ FP. Now let

M_g be the polynomial time nondeterministic Turing machine which on an input x first computes $g(x)$ and then by nondeterminism produces a comb which encodes $g(x)$. Now of course $x \in B \iff g(x) \in A \iff x \in L_{G_A}(M_g)$, this means $B \in G_A - \mathrm{P}$. For the other direction let a language $L_{G_A}(M) \in G_A - \mathrm{P}$ for a polynomial time nondeterministic Turing machine M be given. A function $g \in \mathrm{FP}$ will be constructed such that $x \in L_{G_A}(M) \iff g(x) \in A$. Let g be the function computed by the following polynomial time deterministic Turing machine: on input x it checks if M produces a comb, note that this can be done in deterministic polynomial time; if M does not produce a comb then the machine outputs a word not in A, otherwise it outputs the word encoded by the comb. By construction $x \in L_{G_A}(M) \iff g(x) \in A$. Therefore, $L_{G_A}(M) \in \leq_m^p(A)$. This shows the above equation.

Proof of part (b).

By Propositions 2.2 and 3.1 it suffices to observe that the two constructions in the proof of (a) keep recursiveness, i.e. the language K_F is recursive if the predicate F is, and the predicate G_A is recursive if the language A is. $\square$

With Theorem 3.1 (and Proposition 3.1) one can immediately transfer Corollary 2.2 to the predicate classes.

Corollary 3.1 *The following partial orders are distributive upper semi-lattices, the one in (b) is an upper semi-sublattice of the one in (a).*

(a) The inclusion order on the set of all nontrivial predicate classes.

(b) The inclusion order on the set of all nontrivial recursive predicate classes. This upper semi-lattice is additionally dense.

4 Promise Classes

In this chapter the notion of a *promise class* will be defined and will be shown to be equivalent to the notion of a countable ideal.

4.1 The Definition of Promise Classes

Extend the notion of a predicate to that of a partial predicate which will be called *promise function* here: let a *promise function* be a function from the set of computation trees to the 3-element set $\{0, 1, \perp\}$, the constant-$\perp$ function is excluded.

This definition of a promise function can be considered as a special case of the general concept *promise problem* defined in [ESY84] and [Se88].

Definition 4.1 *For a promise function F and a polynomial time nondeterministic Turing machine M say that M* respects *F if $F(T(M,x)) \neq \perp$ for all words x. In the case that M respects F define the language $L_F(M)$ by*

$$x \in L_F(M) \iff F(T(M,x)) = 1.$$

For every promise function F define the promise class accepted by F*, in short* F – P*, to be the set of languages $L_F(M)$ for which M is a polynomial time nondeterministic Turing machine which respects F. Call a promise class* recursive *if it is accepted by some recursive promise function.*

The examples below may clarify Definition 4.1. Note that if one considers a predicate F as a promise function not having $\perp$ in its image then the two definitions of F – P in Definitions 3.1 and 4.1 coincide. Therefore, the following Proposition 4.1 holds.

Proposition 4.1 *(a) Every predicate class is a promise class. (b) Every recursive predicate class is a recursive promise class.*

The following examples of recursive promise classes are not known to be predicate classes.

- UP is the set of languages L for which there is a polynomial time nondeterministic Turing machine M such that for every input x there is at most one accepting path of M on input x and (if M fulfills this condition) $x \in L$ if and only if there is exactly one accepting path of M on input x. This means that UP = F_u – P for the promise function F_u which has for a computation tree T the value 0 if T does not have a leaf with label 1, which has the value 1 if T has exactly one leaf with label 1, and which has the value $\perp$ otherwise. See also [NR93] for this promise function accpeting UP.

- BPP is equal to F_{BPP} – P where $F_{\text{BPP}}(T)$ has the value $0, 1$, or $\perp$, depending if the value of $\pi(T)$ is in the interval $[0, \frac{1}{4}]$, $[\frac{3}{4}, 1]$, or $]\frac{1}{4}, \frac{3}{4}[$, respectively.

- Let H be the promise function which has the value $0, 1,$ or $\perp$, depending if the the quotient of the number of the leaves with label 1 and the total number of leaves in the tree is in the interval $[0, \frac{1}{4}]$, $[\frac{3}{4}, 1]$, or $]\frac{1}{4}, \frac{3}{4}[$, respectively. Then H accepts the class BPP_{path} defined in [HHT92]. In that paper it is shown that it is unlikely that BPP_{path} equals BPP.

- RP is equal to F_{RP} – P where $F_{\text{RP}}(T)$ has the value $0, 1,$ or $\perp$, depending if the value of $\pi(T)$ is 0, in the interval $[\frac{3}{4}, 1]$, or in the interval $]0, \frac{3}{4}[$, respectively.

- FewP, defined as FNP in [Al86], is the class of languages in NP with at most polynomially many accepting paths. FewP can be easily shown to be accepted by the promise function which for a tree T has the value 0 if there are no leaves with label 1 in T, which has the value 1 if the number of leaves with label 1 is ≥ 1 but does not exceed the depth of T, and which has the value $\perp$ otherwise.

- Finite intersections of nontrivial promise classes like NP $\cap$ co-NP and ZPP = RP $\cap$ co-RP will be shown to be promise classes in the following Lemma 4.1.

4.2 The Characterization of the Promise Classes

Call a promise class *trivial* if it is accepted by a promise function which has not both 0 and 1 in its image.

Proposition 4.2 *The following sets of classes are equal: (a) the trivial promise classes, (b) the trivial ideals, (c) the trivial recursive promise classes, (d) the trivial recursively presentable ideals.*

Proof. If a promise function F does not contain 1 in its image then each language $L_F(M)$ for a polynomial time nondeterministic Turing machine M which respects F is empty. Such a machine M always exists because it can be chosen to be the one which computes for every input a fixed computation tree T for which $F(T) = 0$, remember that the constant-$\perp$ function was excluded to be a promise function. Therefore, F – P = $\{\emptyset\}$. Likewise, F – P = $\{\Sigma^*\}$ if the promise function F does not contain 0 in its image. This shows already the equality of the sets in (a) and (b). For (c) and (d) it suffices to observe that the two trivial promise classes are recursive and that the two trivial ideals are recursively presentable. □

Note that the four sets in Proposition 4.2 above coincide with the four sets in Proposition 3.1.

The next Proposition 4.3 is used in the proof of Lemma 4.1 and in the proof of Theorem 4.1.

Proposition 4.3 *Let F be a promise function. F – P is trivial if and only if F does not have both* 0 *and* 1 *in its image.*

Proof. The direction from right to left holds by the definition of trivial promise classes. In order to prove the other direction it will be shown that if F has both 0 and 1 in its image then it contains P: chose two computation trees T_0 and T_1 such that $F(T_0) = 0$ and $F(T_1) = 1$ and consider a polynomial time deterministic Turing machine D. Let M be the polynomial time nondeterministic Turing machine M which on input x simulates D, and if D terminates with an accepting (rejecting) state M produces by nondeterminism the computation tree T_1 (T_0). The language $L_F(M)$ is by construction equal to the language accepted by D. This shows that P is contained in F – P. Therefore, F – P is not a trivial ideal by Proposition 4.2. □

Before coming to the general characterization of promise classes the following Lemma 4.1 is shown.

Lemma 4.1 *(a) The intersection of two nontrivial promise classes is a nontrivial promise class. (b) The intersection of two nontrivial recursive promise classes is a nontrivial recursive promise class.*

Proof. (a) Let two promise functions D and E be given which both have both 0 and 1 in their image. Define the following promise function $F_{D,E}$: it has the value $\perp$ for the two single-leaf computation trees und is determined for a non-single-leaf tree T by the left and right subtrees T_l, T_r of T (see Figure 4):

$$F_{D,E}(T) := \begin{cases} D(T_l) & \text{if } D(T_l) = E(T_r) \\ \perp & \text{otherwise} \end{cases}$$

For the definition of $F_{D,E}$ see also Figure 6.

		$E(T_r)$		
		0	1	$\perp$
$D(T_l)$	0	0	$\perp$	$\perp$
	1	$\perp$	1	$\perp$
	$\perp$	$\perp$	$\perp$	$\perp$

Figure 6: Definition of $F_{D,E}(T)$

Both D and E have both 0 and 1 in their image, i.e. there exist computation trees T_1, T_2, T_3, T_4 for which $D(T_1) = E(T_2) = 0$ and $D(T_3) = E(T_4) = 1$. Then the computation tree whose left subtree is T_1 (T_3) and whose right subtree is T_2 (T_4) has the $F_{D,E}$-value 0 (1). This means that $F_{D,E}$ has both 0 and 1 in its image, especially it is not the constant-$\perp$ function. Therefore, by Proposition 4.3, $F_{D,E} - \text{P}$ is a nontrivial promise class.

It will be shown that the definition of $F_{D,E}$ guarantees that

$$F_{D,E} - \text{P} = D - \text{P} \cap E - \text{P}.$$

For the inclusion from left to right it will first be shown that each language in $F_{D,E} - \text{P}$ is a language in $D - \text{P}$. Let a polynomial time nondeterministic Turing machine M respect $F_{D,E}$. Define M_l to be the polynomial time nondeterministic Turing machine which on input x simulates the computation of M on input x besides that it ignores the first branching (which exists because M respects $F_{D,E}$) and only simulates the left computation. Note that the computation tree $T(M_l, x)$ is the left subtree of $T(M, x)$. Because M respects $F_{D,E}$ it can by the definition of $F_{D,E}$ be concluded that $D(T(M_l, x)) \neq \perp$ and – moreover – $D(T(M_l, x)) = 1 \iff F_{D,E}(T(M, x)) = 1$. This means that M_l respects D and $L_D(M_l) = L_{F_{D,E}}(M)$. Therefore, the language $L_{F_{D,E}}(M)$ is an element of $D - \text{P}$. The same way it is shown that each language in $F_{D,E} - \text{P}$ is a language in $E - \text{P}$.

For the other direction of the equation above let a language $L \in D - \text{P} \cap E - \text{P}$

be given. This means that there are two polynomial time nondeterministic Turing machines M_d and M_e respecting D and E, respectively, such that $L = L_D(M_d) = L_E(M_e)$. Let $M_{d,e}$ be the polynomial time nondeterministic Turing machine which on input x first branches and then simulates M_d on input x in the left computation and M_e on input x in the right computation. By construction $M_{d,e}$ respects $F_{D,E}$ and $L = L_{F_{D,E}}(M_{d,e})$. This shows that every language $L \in D - \mathrm{P} \cap E - \mathrm{P}$ is in $F_{D,E} - \mathrm{P}$ and finishes the proof of the above equation.

For part (b) it suffices to observe that if in the proof of (a) the promise functions D and E are recursive then also $F_{D,E}$ is recursive. □

The following theorem characterizes the promise classes.

Theorem 4.1 *(a) The promise classes are exactly the countable ideals. (b) The recursive promise classes are exactly the recursively presentable ideals.*

Proof. First part (a) with its two directions will be proven.

Proof of part (a), direction $\subseteq$: *Every promise class is a countable ideal.*

Let F be a promise function. By Proposition 4.2 it can w.l.o.g. be assumed that F containes 0 in its image. It will be shown that with respect to $\leq_m^p$-reducibility $F - \mathrm{P}$ is closed downward and closed under join. To show that $F - \mathrm{P}$ is closed downward let a set A be polynomial time many-one reducible via a reduction function f to a language $L_F(M)$ where M is a polynomial time nondeterministic Turing machine which respects F. Then also A is in $F - \mathrm{P}$ because $A = L_F(M_f)$, where M_f is the machine which for an input x first computes $f(x)$ and then simulates M on $f(x)$, note that also M_f respects F. Therefore, $F - \mathrm{P}$ is closed downward with respect to $\leq_m^p$-reducibility.

The closure under join is also easy to see: let M_a, M_b be two polynomial time nondeterministic Turing machines which respect F. Then the following machine M_c also respects F: on input $0x$ M_c simulates M_a on input x, on input $1x$ it simulates M_b on input x, and on the empty word as input it produces a tree T for which $F(T) = 0$. By construction $L_F(M_c) = L_F(M_a) \oplus L_F(M_b)$. This shows that the join of any two languages in $F - \mathrm{P}$ is also a language in $F - \mathrm{P}$.

$F - \mathrm{P}$ is countable because there are only countably many polynomial time nondeterministic Turing machines.

Proof of part (a), direction $\supseteq$: *Every countable ideal is a promise class.*

For the two trivial ideals the statement holds by 4.2. For a given nontrivial countable ideal I let A, B be the two languages for I from the Theorem 2.3 of

Slaman and Shinoda, i.e. $I = \leq_m^p(A) \cap \leq_m^p(B)$. By Theorem 3.1 there are two predicates G_A and G_B such that G_A – P = $\leq_m^p(A)$ and G_B – P = $\leq_m^p(B)$. Because I is nontrivial the two promise classes G_A – P and G_B – P are nontrivial promise classes. Now by Lemma 4.1(a) also $I = \leq_m^p(A) \cap \leq_m^p(B) = G_A$ – P $\cap$ G_B – P is a (nontrivial) promise class.

Proof of part (b), direction $\subseteq$: *Every recursive promise class is a recursively presentable ideal.*

Given a recursive promise function F it is shown in part (a) that F – P is an ideal, it remains to show that F – P is recursively presentable. If F does not have both 0 and 1 in its image then by Proposition 4.2 F – P is a trivial recursively presentable ideal. So it can w.l.o.g. be assumed that F has both 0 and 1 in its image. This implies by Proposition 4.3 that F – P is not a trivial ideal, therefore P $\subseteq F$ – P.

First note that

$$F - \text{P} = \{L_F(M_i^p) \mid i \in \mathbb{N} \text{ and } M_i^p \text{ respects } F\}.$$

Construct – like this was done for RP and UP in [Ad78] and [AS89], respectively – the following recursive language A_F for which it will be shown that $\{A_F^{(i)} \mid i \in \mathbb{N}\} = F$ – P.

$$A_F := \{\langle z_i, x\rangle \mid F(T(M_i^p, x)) = 1 \text{ and for all } y \text{ with } |y| \leq |x| : F(T(M_i^p, y)) \not\equiv \bot\}.$$

It suffices to show that

$$\{A_F^{(i)} \mid i \in \mathbb{N}\} = \{L_F(M_i^p) \mid i \in \mathbb{N} \text{ and } M_i^p \text{ respects } F\}.$$

For every number $i \in \mathbb{N}$: if M_i^p respects F then $A_F^{(i)} = L_F(M_i^p)$ by construction of A_F, otherwise $A_F^{(i)}$ is finite and therefore an element of P $\subseteq F$ – P = $\{L_F(M_i^p) \mid i \in \mathbb{N} \text{ and } M_i^p \text{ respects } F\}$. This shows that F – P = $\{A_F^{(i)} \mid i \in \mathbb{N}\}$ and finishes the proof of part (b), direction $\subseteq$.

Proof of part (b), direction $\supseteq$: *Every recursively presentable ideal is a recursive promise class.*

The proof is analog to the one for part (a), direction $\supseteq$, besides that Theorem 2.2 of Ambos-Spies (instead of Theorem 2.3) and Lemma 4.1(b) (instead of Lemma 4.1(a)) are used. $\square$

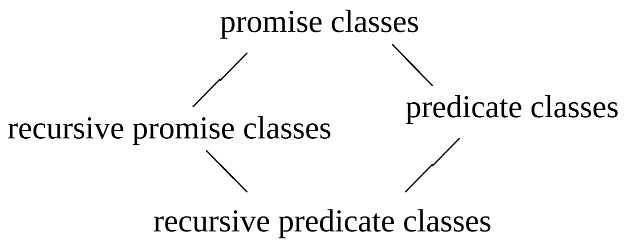


Figure 7: The relation of predicate classes and promise classes

4.3 Consequences of the Characterization of the Promise Classes

The following Corollary 4.1 combines Propositions 2.5 and 2.6 with Propositions 3.1 and 4.2 and Theorems 3.1 and 4.1.

Corollary 4.1 *(a) The nontrivial promise classes are exactly the pairwise intersections of nontrivial predicate classes. The inclusion order on the set of nontrivial promise classes is a distributive lattice. (b) The nontrivial recursive promise classes are exactly the pairwise intersections of nontrivial recursive predicate classes. The inclusion order on the set of nontrivial recursive promise classes is a distributive lattice.*

The following Corollary 4.2 combines Proposition 2.4 with Theorems 3.1 and 4.1, see Figure 7.

Corollary 4.2 *(a) The set of recursive predicate classes is a proper subset of the set of recursive promise classes. (b) The set of recursive predicate classes is a proper subset of the set of predicate classes. (c) The set of recursive promise classes is a proper subset of the set of promise classes. (d) The set of predicate classes is a proper subset of the set of promise classes. (e) There is a predicate class which is not a recursive promise class. (f) There is a recursive promise class which is not a predicate class.*

In [Si82, Kow84, HI85, HH88, AS89] it was investigated whether promise classes like UP, RP, BPP, and NP $\cap$ co-NP have $\leq_m^p$-complete languages. The following consequence of Theorems 3.1 and 4.1 and Propositions 2.2 and 2.3 states that this is the case if and only if the 'promise' can be eliminated.

Corollary 4.3 *(a) A promise class has a $\leq_m^p$-complete language if and only if it is a predicate class. (b) A recursive promise class has a $\leq_m^p$-complete language if and only if it is a recursive predicate class.*

The next corollary follows from Theorem 4.1 and the facts stated before that PH and BH are recursively presentable ideals and that E = DTIME(2^{lin}) is recursively presentable but not an ideal.

Corollary 4.4 PH *and* BH *are recursive promise classes.* E *is not a promise class.*

5 Analogous Results for Other Nondeterministic Computation Models

The two main results of this paper were shown in the preceeding Chapters 3 and 4. In this chapter some other models of nondeterministic computation will be considered. For each model the notion of a predicate class is defined and an analog of Theorem 3.1(a) is stated. The analoga will be called *corollaries* because their proofs are similar to that of Theorem 3.1(a).

Remark. For the models of Sections 5.1, 5.2, and 5.3 also notions of *recursive predicate classes, promise classes,* and *recursive promise classes* could easily be defined in the obvious way, and analoga of Theorems 3.1(b), 4.1(a), and 4.1(b) could be proven for each model.

5.1 Balanced Polynomial Time Turing Machines

Call a polynomial time nondeterministic Turing machine M *balanced* if for every input x the computation tree $T(M,x)$ is balanced, i.e. all paths from the root of the tree to a leaf have the same length. Note that also for this model the deterministic steps are not recorded in the computation tree. Consider a predicate F for balanced computation trees. Note that F can be characterized by a language of words of length 2^i for $i \geq 1$. Let the *balanced predicate class accepted by F* be the set of all languages $L_F(M)$ such that M is a balanced polynomial time nondeterministic Turing machine.

Corollary 5.1 *The balanced predicate classes are exactly the principal ideals.*

Sketch of proof. Let $M_i^{\mathrm{p,bal}}$ be the nondeterministic Turing machine which on input x simulates the machine M_i^p in the following way: it first computes the length l of the path from the root to the leftmost leaf of $T(M_i^p, x)$ and then it simulates M_i^p on input x with the following two additional features: if M_i^p terminates with result r on some path with length smaller than l it extends by nondeterminism the computation so that every extended path has length l and result r; if on the other side the computation is already on level l of the computation tree then only the leftmost extending computation path is simulated. The enumeration $M_i^{\mathrm{p,bal}}$ has the property that each $M_i^{\mathrm{p,bal}}$ is a balanced nondeterministic polynomial time Turing machine, and that for each balanced nondeterministic polynomial time Turing machine M there is an i such that $T(M, x) = T(M_i^{\mathrm{p,bal}}, x)$ for all x.

For a given predicate F for balanced computation trees, which is not constant-1, define the language $K_F^{\mathrm{bal}} := \{\langle z_i, x, 0^t\rangle \mid t = |x|^i + i \text{ and } F(T(M_i^{\mathrm{p,bal}}, x)) = 1\}$, and show – like in Theorem 3.1 – that the balanced predicate class accepted by F is equal to $\leq_m^p(K_F^{\mathrm{bal}})$. For the other direction define for a given language $A \neq \Sigma^*$ the predicate G_A^{bal} on balanced computation trees characterized by the language $\{xy \mid x \in A \text{ and } |y| = 2^{|x|+1} - |x|\}$, and show that $\leq_m^p(A)$ is equal to the balanced predicate class accepted by G_A^{bal}. □

5.2 Polynomial Time Bit-Reducibility

Predicates on balanced computation trees can be identified with languages consisting of words of length 2^i for $i \geq 1$. In [HL*93, HVW94, JMT94] a certain more general concept of balanced computation trees was introduced for which there is a one-one correspondence between languages and predicates on balanced computation trees of that more general type. It was shown that this approach is equivalent to the following approach of looking at *bit-reducibility closures*.

The following definition is equivalent to the one in [HL*93]. A language A is *polynomial time bit-reducible* to a language B if there exist two functions $f, g \in \mathrm{FP}$ such that

$$x \in A \iff [g(x, z_0)][g(x, z_1)] \ldots [g(x, f(x))] \in B,$$

where $[g(x, z_i)]$ is defined to be the letter 0 if $g(\langle x, z_i\rangle) = \epsilon$ and the letter 1 otherwise. Let $\mathcal{R}_m^{\text{p,bit}}(B)$ be the set of all languages polynomial time bit-reducible to B, and call $\mathcal{R}_m^{\text{p,bit}}(B)$ the *bit-reducibility closure of* B. A characterization analogous to the one in Theorem 3.1 is obtained.

Corollary 5.2 *The bit-reducibility closures are exactly the principal ideals.*

Sketch of proof. First it is indicated that every bit-reducibility closure is principal ideal, this direction of the corollary was already shown in [BCS92]. For a given language $B \neq \Sigma^*, \Sigma^* - \{\epsilon\}$ let K_B^{bit} be the language $\{\langle z_i, z_j, x, 0^t\rangle \mid t = (|x|^i + i + |x|)^j + j$ and $[f_j^p(x, z_0)][f_j^p(x, z_1)] \ldots [f_j^p(x, f_i^p(x))] \in B\}$, remember from Section 2.3 that FP $= \{f_i^p \mid i \in \mathbb{N}\}$. Like in the proof of Theorem 3.1 show that

$$\leq_m^p(K_B^{\text{bit}}) = \mathcal{R}_m^{\text{p,bit}}(B).$$

In order to see for example that K_B^{bit} is in $\mathcal{R}_m^{\text{p,bit}}(B)$ let f be the function which maps an input of the form $\langle z_i, z_j, x, 0^t\rangle$ where $t = (|x|^i + i + |x|)^j + j$ to $f_i^p(x)$. And let g be the function which maps an input of the form $\langle\langle z_i, z_j, x, 0^t\rangle, y\rangle$ where $t = (|x|^i + i + |x|)^j + j$ to $g_j(\langle x, y\rangle)$ if $|y|$ is not greater than $|x|^i + i$, and to ϵ otherwise. If the input w is not of that form assumed in the two definitions above, f and g can be defined such that $[g(w, z_0)][g(w, z_1)] \ldots [g(w, f(w))]$ is a word not in B. It is easy to see that both f and g are in FP and that K_B^{bit} is polynomial time bit-reducible to B via f and g.

In order to see that every principal ideal is a bit-reducibility closure define for a given language $A \neq \Sigma^*$ the language $G_A^{\text{bit}} := \{xy \mid x \in A, |y| = 2^{|x|}\}$. It will be shown that

$$\leq_m^p(A) = \mathcal{R}_m^{\text{p,bit}}(G_A^{\text{bit}}).$$

Let B be $\leq_m^p$-reducible to A via $h \in$ FP. Define f to be the function which maps a word x to z_i where $i = |h(x)| + 2^{|h(x)|}$. And define g to be the function which maps a pair $\langle x, z_j\rangle$ to a (fixed) word $\neq \epsilon$ if $j \leq |h(x)|$ and the jth bit of $h(x)$ is 1, and to ϵ otherwise. It is easy to see that both f and g are in FP and that B is polynomial time bit-reducible to G_A^{bit} via f and g. This shows the inclusion from left to right of the above equation. In order to see the other inclusion let a language C be polynomial time bit-reducible to G_A^{bit} via two functions $f', g' \in$ FP. Then C is $\leq_m^p$-reducible to A via the following function in FP: on input x check if $f'(x)$ is equal to z_i for an i of the form $j + 2^j$; if this is not the case output a word which is not in A; otherwise output the word $[g'(x, z_0)] \cdots [g'(x, z_j)]$. □

5.3 Polynomial Time Nondeterministic Transducers

Call the following kind of polynomial time nondeterministic Turing machine a *polynomial time nondeterministic transducer*: it is a polynomial time nondeterministic Turing machine of the kind described in the introduction besides that it outputs on each computation path not only 0 or 1 but a whole word. The computation trees $T(M, x)$ of nondeterministic transducers are binary trees with words as leaf labels. Call these trees *transducer computation trees*.

Consider a predicate F on transducer computation trees, and let for a polynomial time nondeterministic transducer M the language $L_F(M)$ be defined by $x \in L_F(M) \iff F(T(M, x)) = 1$. Let the *transducer predicate class accepted by F* be the class of languages $L_F(M)$ for which M is a polynomial time nondeterministic transducer.

Examples. Let F (F') be the predicate which interprets for a transducer computation tree the leaf labels as binary numbers and is 1 if and only if the largest of them is odd (if and only if the largest of them appears only once in the tree). Then F – P = F' – P = Δ_2^p by the results in [Wa87, Kr88] and [Pa84], respectively. Likewise for the predicate F'' which is 1 if and only if the length of the longest leaf label in the transducer computation tree is odd it is easy to see that F'' – P = Θ_2^p by the results in [Wa87, Kr88, Wa90].

Note that transducer predicate classes are a generalization of predicate classes: identify all the words $\neq \epsilon$. Then every predicate induces a transducer predicate – accepting the same class – by reading ϵ as 0 and all other words as 1.

This shows already one direction of the following corollary, the other is proven with basically the same proof as for Theorem 3.1.

Corollary 5.3 *The transducer predicate classes are exactly the principal ideals.*

Let at this point the following corollary summarize the results of Theorem 3.1(a) and Corollaries 5.1, 5.2, and 5.3.

Corollary 5.4 *The following sets of classes are equal:*
(a) the set of principal ideals,
(b) the set of predicate classes,
(c) the set of balanced predicate classes,

(d) the set of bit-reducibility closures,
(e) the set of transducer predicate classes.

5.4 Polynomial Time Function Classes

Fix any nonemtpty set S, for example $S = \{0,1\}$, $S = \Sigma^*$, $S = \mathbb{N}$, or $S = \mathbf{Z}$, and consider the set S^{Σ^*} of the functions from Σ^* to S. Define the polynomial time many-one reducibility $\leq^p_{m,S}$ among these functions, i.e. for $r, t \in S^{\Sigma^*}$ let $r \leq^p_{m,S} t$ if there exists a function $f \in$ FP such that for all $x \in \Sigma^*$:

$$r(x) = t(f(x)),$$

see for example [Wa86a] and also [Vo94a, Vo94b] where this reducibility is called $\leq^{FP}_m$. It is easy to see that $\leq^p_{m,S}$ is a preorder. For $t \in S^{\Sigma^*}$ let $\leq^p_{m,S}(t)$ be the set $\{r \in S^{\Sigma^*} \mid r \leq^p_{m,S} t\}$ and call $\leq^p_{m,S}(t)$ a *principal* $\leq^p_{m,S}$*-ideal*. Note that t is $\leq^p_{m,S}$-complete for $\leq^p_{m,S}(t)$.

Consider a function F from the computation trees to S. For a polynomial time nondeterministic Turing machine M let $s_F(M) \in S^{\Sigma^*}$ be the function which maps x to $F(T(M,x))$, and let the S*-function class accepted by* F, in short $F-$P, be the set of functions $s_F(M)$ such that M is a polynomial time nondeterministic Turing machine.

Examples. For $S = \{0,1\}$ one has exactly the case of Chapter 3. Therefore, the concept of S-function classes is a generalization of the concept of predicate classes. For $S = \mathbb{N}$ let F be the function which maps a computation tree to the number of 1's in the tree, then $F-$P = #P according to the definition in [Va79]. For $S = \mathbf{Z}$ (the set of integers) let G be the function which maps a computation tree to the difference of the number of 1's and the number of 0's in the tree, then $G-$P = GapP according to the definition in [FFK94].

With a nearly identical proof like the one for Theorem 3.1 one has for every nonempty set S the following theorem.

Corollary 5.5 *The S-function classes are exactly the principal $\leq^p_{m,S}$-ideals.*

Sketch of proof. For a function F from the computation trees to S define the function K_F^S which maps an input of the form $\langle z_i, x, 0^t \rangle$ where $t = |x|^i + i$ to $F(T(M_i^p, x))$, and which maps all other inputs to a fixed value $a \in S$ which is in the image of F otherwise. It can be shown like in the proof Theorem 3.1 that F – P = $\leq_{m,S}^p(K_F^S)$.

For a principal ideal $\leq_{m,S}^p$-ideal $\leq_{m,S}^p(t)$ let G_t^S be the function which maps a computation tree T to $t(x)$ if T is a comb which encodes x, and maps T to a fixed value $a \in S$ which is in the image of t otherwise. Now it can be shown like in the proof of Theorem 3.1 that $\leq_{m,S}^p(t)$ = G_t^S – P. □

The function class notion can be extended in the obvious way to the nondeterministic transducers, see the previous section. Several well-known complexity classes are ℕ-function transducer classes, for example the function classes OptP and OptP[$O(\log n)$] from Krentel [Kr88]: let H be the function which interprets for a transducer computation tree the leaf labels as binary numbers and maps the tree to the largest number of them. Then H – P = OptP by definition of OptP (for the maximization problems), and let H' be the function which maps a transducer tree to the length of the longest leaf label in the tree, then it is easy to see that H' – P = OptP[$O(\log n)$]. As another example let D be the function which maps the tree to the number of its different leaf labels. Then D – P = Span-P according to the definition in [KST89]. In Vollmer's thesis [Vo94b] several other ℕ-function transducer classes are investigated. In order to get an example of a Σ^*-function transducer class let F_l be the function which maps a transducer computation tree to the leaf label of the leftmost path. Then F_l – P = FP.

Again, one has the following characterization (for every nonempty set S).

Corollary 5.6 *The S-function transducer classes are exactly the principal $\leq_{m,S}^p$-ideals.*

5.5 Relativized Predicate Classes

Consider the well-known concept of (nondeterministic) oracle Turing machines as described for example in [BDG88]. Let X be a language, X will be called in the following context an *oracle*. A *polynomial time nondeterministic oracle-X Turing machine* $M^{\text{rel},X}$ is a nondeterministic oracle Turing machine M^{rel} equipped with the oracle X whose running time on every path is bounded by a polynomial in the input length (the oracle questions are counted as one step). For a computation of a polynomial time nondeterministic oracle-X Turing machine $M^{\text{rel},X}$ on an input x the computation tree $T(M^{\text{rel},X}, x)$ is defined like in the unrelativized case, the oracle questions are not recorded in $T(M^{\text{rel},X}, x)$. Given a predicate F on computation trees, let $L_F(M^{\text{rel},X})$ be the language defined by $x \in L_F(M^{\text{rel},X}) \iff F(T(M^{\text{rel},X}, x)) = 1$, and let the *predicate class accepted by F relative to oracle X*, in short $F-\mathrm{P}^X$, be the set of languages $L_F(M^{\text{rel},X})$ for which $M^{\text{rel},X}$ is a polynomial time nondeterministic oracle-X Turing machine.

Example. Let F_1 be the predicate accepting NP, see the first example in Section 2.5. The class Σ_2^p is by definition the predicate class accepted by F_1 relative to oracle SAT, in other words $\Sigma_2^p = F_1 - \mathrm{P}^{\mathrm{SAT}}$.

Corollary 5.7 *Let X be any language. Every predicate class relative to oracle X is a principal ideal.*

Sketch of proof. Let the nondeterministic oracle Turing machines be encoded by words. Let M_i^{rel} be the nondeterministic oracle Turing machine which simulates on input x the nondeterministic oracle Turing machine encoded by z_i with the time bound of $|x|^i + i$ steps, note that also the oracle questions are simulated. Let X be any oracle. Like in the unrelativized case it is easy to see that $F-\mathrm{P}^X = \{L_F(M_i^{\text{rel},X}) \mid i \in \mathbb{N}\}$. For a non-constant predicate F define like in the proof of Theorem 3.1 the language

$$K_F^X := \{\langle z_i, x, 0^t\rangle \mid t = |x|^i + i \text{ and } F(T(M_i^{\text{rel},X}, x)) = 1\},$$

and show that $F-\mathrm{P}^X = \leq_m^p(K_F^X)$. Note that the reduction $x \to \langle z_i, x, 0^{|x|^i+i}\rangle$ from a language $L_F(M_i^{\text{rel},X})$ to K_F^X does not need the oracle X. $\square$

Note that the opposite direction of the statement of the above Corollary 5.7 holds if and only if the oracle X is in P: If $X \in \mathrm{P}$ then $F-\mathrm{P} = F-\mathrm{P}^X$ for all predicates F, so the opposite direction holds by Theorem 3.1. If X does not belong to P then every predicate class relative to oracle X is either trivial or it contains the language X, so it cannot be the class P. But P is a principal ideal.

Let F_l be the predicate from the examples in Section 2.5 which for a computation tree has the value 1 if and only if the leftmost leaf in T has label 1. It is easy to see that $\mathrm{P}^X = F_l - \mathrm{P}^X$ for every oracle X, where P^X is the set of languages which can be computed in deterministic polynomial time with oracle X. Therefore, the above Corollary 5.7 implies as a special case the following result of Ambos-Spies in [AS86a] mentioned before.

Corollary 5.8 (Ambos-Spies 1986) *For every oracle X the class* P^X *is a principal ideal.*

Part II

On the Acceptance Power of Regular Languages

In this part of the thesis predicate classes will be considered which are accepted by a predicate of very low complexity: the predicates determined by a regular language for the yields of computation trees.

The basic definitions and observations are presented in Chapter 6. Chapter 7 leads to a lemma about regular languages which is used in Chapter 8 to prove the main result and its corollaries.

6 Predicate Classes Accepted by Regular Languages

In Section 6.1 it will be shown how – in an obvious way – any language determines a predicate on computation trees and therefore determines a predicate class. After the definition of *regular languages* in Section 6.2 some basic results about predicate classes determined by regular languages are presented in Section 6.3.

6.1 Predicate Classes Accepted by Languages

For a computation tree T let the *yield of* T, formally yield(T), be the word which is the concatenation of the labels of the leaves of T, read from left to right. For example, the yield of the computation tree in Figure 3 is the word 00101100.

Given any language A, one can consider A as a predicate Y_A for computation trees by the definition

$$Y_A(T) = 1 :\iff \text{yield}(T) \in A.$$

In other words, given a language A, the predicate Y_A is determined for a computation tree T by looking at the yield of T: if the yield is a word from A then the predicate has the value 1, otherwise it has the value 0.

For simplicity the predicate class $Y_A - \mathrm{P}$ will just be denoted as $A - \mathrm{P}$, this should not cause confusion. Say that A *accepts* $A - \mathrm{P}$. Likewise denote the language $L_{Y_A}(M)$ (for a polynomial time nondeterministic Turing machine M) just by $L_A(M)$.

Examples.

- Let <NP> be the language which consists of the words which contain at least one letter 1. Then obviously $Y_{<\text{NP}>} = F_1$ where F_1 is the predicate from Section 3.1 which accepts NP. In other words, <NP> – P = NP.

- Likewise for the language L_{maj} which consists of the words which have more 1's than 0's it holds that $Y_{L_{\text{maj}}} = F_{\text{maj}}$, where F_{maj} was one of the predicates from Section 3.1 accepting PP. In other words, L_{maj} – P = PP.

6.2 The Definition of Regular Languages

In Part II of this thesis the predicate classes accepted by regular languages will be considered. Regular languages were introduced by McCulloch and Pitts in [MP43] and Kleene in [Kl56]. There are many equivalent characterizations of regular languages, see for example [HU79], here they will be defined to be the languages which are accepted by finite automata.

It follows the definition of finite automata and regular languages. To this formal definition will only be refered in the proof of Lemma 7.3.

Define – like in [HU79] – a *finite automaton* to be a quintuple $A = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of *states*, Σ is the alphabet $\{0, 1\}$, $\delta : Q \times \Sigma \to Q$ is the *transition function*, $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ is the set of *accepting states*.

For every word $w \in \Sigma^*$ a function $\delta_w : Q \to Q$ is defined the following inductive way. Let δ_ϵ denote the identity function, and let δ_{w0} and δ_{w1} be defined by $\delta_{w0}(q) := \delta(\delta_w(q), 0)$ and $\delta_{w1}(q) := \delta(\delta_w(q), 1)$, respectively. The definition reflects the idea that δ_w is the function which in the finite automaton A starts with a state q and then follows the letters of w, stopping in state $\delta_w(q)$.

For a finite automaton $A = (Q, \Sigma, \delta, q_0, F)$ call $\{w \in \Sigma^* \mid \delta_w(q_0) \in F\}$ the *language accepted by A*. A language is called *regular* if it is accepted by a finite automaton.

6.3 Predicate Classes Accepted by Regular Languages

First some examples of predicate classes accepted by regular languages will be given:

- The language $<$NP$>$ consisting of the words which contain the letter 1 (defined already in the examples of Section 6.1) is regular, i.e. NP is accepted by the regular language $<$NP$>$. Note that the second example language from Section 6.1 L_{maj} is not regular.
- Define $<$co-NP$>$ to be the complement of $<$NP$>$, i.e. the regular language consisting of the words which only contain 0's. By definition of co-NP the language $<$co-NP$>$ accepts co-NP.
- Let $<$P$>$ be the regular language which consists of the words starting with letter 1. Then obviously $Y_{<\text{P}>} = F_l$, where F_l is the predicate from Section 3.1 which accepts P. This means $<$P$>$ – P = P.
- Call the languages which cannot distinguish any yields of computation trees *trivial*, these are the four (regular) languages $\emptyset, \{\epsilon\}, \Sigma^*$, and $\Sigma^* - \{\epsilon\}$, note that the yield of a computation tree has at least length 1. It is easy to see that these four languages are exactly the languages which accept the trivial predicate classes.
- In [HL*93] it is mentioned that for every $i \in \mathbb{N}$ there exist regular languages accepting the classes Σ_i^p and Π_i^p of the Polynomial Time Hierarchy. Also there the existence of a regular language accepting the class PSPACE is shown.
- For a number $k \geq 2$ and a subset $S \subseteq \{0, \ldots, k-1\}$ let $<S, \{0, \ldots, k-1\}>$ be the regular language consisting of the words for which the number of 1's is equal modulo k to an element of S. Then, by definition of MOD$_k$P, see [BG92], the language $<\{1, \ldots, k-1\}, \{0, \ldots, k-1\}>$ accepts the predicate class MOD$_k$P. As a special case, the language $<\{1\}, \{0, 1\}>$ accepts by definition $\oplus$P.

Let $\mathcal{R}$ be the set of all nontrivial regular languages, and let $\mathcal{R}$ – P be the set of classes $\{L – \text{P} \mid L \in \mathcal{R}\}$, i.e. the set of all predicate classes which are accepted by a nontrivial regular language.

The following Theorem 6.1 is due to Hertrampf, Lautemann, Schwentick, Vollmer and Wagner in [HL*93].

Theorem 6.1 ([HL*93]) *P is the minimum of the inclusion order on* $\mathcal{R}-\mathrm{P}$*, and* PSPACE *is its maximum .*

Remark. Note that it is not known whether P = PSPACE. In that case $\mathcal{R}-\mathrm{P}$ would – by the above theorem – only consist of the class P, and the following Proposition 6.1 and even Theorem 8.1 in Chapter 8 would hold for trivial reasons.

Proposition 6.1 *The inclusion order on* $\mathcal{R}-\mathrm{P}$ *is an upper semi-lattice.*

Proof. Given two languages A, B, let $A \oplus B$ be the language $0A \cup 1B$. It will be shown that $A \oplus B-\mathrm{P}$ is the smallest class containing both $A-\mathrm{P}$ and $B-\mathrm{P}$.

In order to show that $A-\mathrm{P} \subseteq A \oplus B-\mathrm{P}$ let a polynomial time nondeterministic Turing machine M be given. Define M_0 to be the polynomial time nondeterministic Turing machine which on input x first produces by nondeterminism a 0 in the leftmost path, and then simulates M on input x. By construction yield$(T(M,x)) \in A \iff$ yield$(T(M_0,x)) \in 0A \iff$ yield$(T(M_0,x)) \in A \oplus B$ for all inputs x. In other words, for every polynomial time nondeterministic Turing machine M there is polynomial time nondeterministic Turing machine M_0 such that $L_A(M) = L_{A\oplus B}(M_0)$. Therefore, $A-\mathrm{P} \subseteq A \oplus B-\mathrm{P}$. The inclusion $B-\mathrm{P} \subseteq A \oplus B-\mathrm{P}$ holds similarly.

In order to show that $A\oplus B-\mathrm{P}$ is the smallest class among the predicate classes accepted by nontrivial regular languages containing both $A-\mathrm{P}$ and $B-\mathrm{P}$ let $C-\mathrm{P}$ be another nontrivial (regular) language containing $A-\mathrm{P}$ and $B-\mathrm{P}$. It will be shown that for every polynomial time nondeterministic Turing machine M there is polynomial time nondeterministic Turing machine M_0 such that $L_{A\oplus B}(M) = L_C(M_0)$. Because C is not trivial one can choose two words v, w with length ≥ 1 such that $v \in C$ and $w \notin C$. Given a polynomial time nondeterministic Turing machine M, define M_1 (M_2) to be the polynomial time nondeterministic Turing machine which for an input x first checks if M on input x has more than one computation path. If yes then M_1 (M_2) simulates M besides that it does not compute the leftmost path; otherwise it produces by nondeterminism a computation tree with yield v if $\epsilon \in A$ ($\epsilon \in B$) and a computation tree with yield w if

$\epsilon \notin A$ ($\epsilon \notin B$). Because C – P contains both A – P and B – P there exist polynomial time nondeterministic Turing machines M_3 and M_4 such that yield(M_1, x) $\in A \iff$ yield(M_3, x) $\in C$, and yield(M_2, x) $\in B \iff$ yield(M_4, x) $\in C$, respectively. Now let M_0 be the polynomial time nondeterministic Turing machine which for an input first looks if the bit of the leftmost path of the computation of M on the input is 0 or 1 and then simulates M_3 or M_4, respectively. By construction is yield(M, x) $\in A \oplus B \iff$ yield(M_0, x) $\in C$. In other words, for every polynomial time nondeterministic Turing machine M there is polynomial time nondeterministic Turing machine M_0 such that $L_{A\oplus B}(M) = L_C(M_0)$. This shows $A \oplus B$ – P $\subseteq C$ – P. □

It will be shown in Chapter 8 that if a nontrivial regular language R does not accept P then at least one of the classes NP, co-NP or MOD$_p$P for p prime is contained in R – P. For the proof the following detour to formal languages will be made.

7 A Lemma about Regular Languages

In Section 7.1 a reducibility among languages will be introduced which implies the inclusion of the corresponding accepted predicate classes. It will be shown in Section 7.3 that for the regular languages this reducibility is related to the concept of *generalized definite languages* which will be defined in Section 7.2 .

7.1 o-h-Reducibility

Let an *ϵ-free homomorphism* be a mapping h which maps the letters 0 and 1 to non-empty words. An (ϵ-free) homomorphism is extended to words inductively by $h(\epsilon) := \epsilon$ and $h(ax) := h(a)h(x)$ for a letter a and a word x, see also [HU79].

For two languages A and B the *o-h-reducibility* will be defined, the name stands for *offset–homomorphism*. It is not known to the author whether the concept is defined in the literature.

Definition 7.1 *Let A, B be two languages. A is* o-h-reducible *to B if there exist two words y, z, called the* offsets, *and an ϵ-free homomorphism h such that for all words x*

$$x \in A \iff yh(x)z \in B.$$

Proposition 7.1 *The o-h-reducibility on the set of all languages is a preorder.*

Proof. A language is o-h-reducible to itself via two empty offsets and the homomorphism id with $id(0) = 0$ and $id(1) = 1$. This shows the reflexivity of the relation, the transitivity is also shown in a straightforward way as follows. Let a language A be o-h-reducible to a language B via the offsets y_1 and z_1 and the homomorphism determined by h_1, and let B be o-h-reducible to a language C via the offsets y_2 and z_2 and the homomorphism h_2. Then A is o-h-reducible to C via the offsets $y_2h_2(y_1)$ and $h_2(z_1)z_2$ and the homomorphism h with $h(0) = h_2(h_1(0))$ and $h(1) = h_2(h_1(1))$ because it holds $x \in A \iff y_1h_1(x)z_1 \in B \iff y_2h_2(y_1)h_2(h_1(x))h_2(z_1)z_2 \in C$. Finally note that h is ϵ-free because h_1 and h_2 are. $\Box$

The following easy lemma motivates the definition o-h-reducibility.

Lemma 7.1 *Let A, B be languages. If A is o-h-reducible to B then A*– P $\subseteq$ *B*– P.

Proof. Let A be o-h-reducible to B via the offsets y and z and the homomorphism h. For a given polynomial time nondeterministic Turing machine M a polynomial time nondeterministic Turing machine M_0 will be constructed such that $L_A(M) = L_B(M_0)$. On input x M_0 simulates the computation of M on input x, producing everytime M rejects or accepts by nondeterminism a computation tree whose yield is h(0) or h(1), respectively. And if y (z) is not the empty word, M_0 produces additionally in the leftmost (rightmost) computation path a tree whose yield is y (z). By construction yield($T(M, x)$) $\in A \iff$ yield($T(M_0, x)$) $\in B$. This shows A– P $\subseteq B$– P. $\Box$

Example. Let, like in [BGu82, GW87], 1–NP (2–NP) be the class accepted by the regular language L_1 (L_2) which consists of the words which contain exactly one 1 (exactly two 1's). The lemma above shows that 1–NP $\subseteq$ 2–NP because L_1 is o-h-reducible to L_2 via the homomorphism id with $id(0) = 0, id(1) = 1$ and the offsets $y = 1$ and $z = \epsilon$. The languages L_1 and L_2 also show that the opposite direction of the Lemma 7.1 above does not hold because it is easy to see that L_2 is not o-h-reducible to L_1 but 1–NP = 2–NP was shown in [GW87].

Remember that for a number $k \geq 2$ and a subset $S \subseteq \{0, \dots, k-1\}$ the language $<S, \{0, \dots, k-1\}>$ was defined (in the last example of Section 6.3) to be the regular language consisting of the words for which the number of 1's is equal modulo k to an element of S. Note that if S is empty or equal to $\{0,\dots,k-1\}$ then $<S,\{0,\dots,k-1\}>$ is trivial. The following Lemma 7.2 gives some examples of o-h-reducibility among the languages of the type $<S,\{0,\dots,k-1\}>$ where S is a nonempty and proper subset of $\{0,\dots,k-1\}$ for some $k \geq 2$. The relation of being a proper subset will be expressed in the following text by $\subset$.

Lemma 7.2 *For a nonemtpty set $S \subset \{0,\dots,k-1\}$ for some $k \geq 2$ there exists a prime p and a nonempty set $Q \subset \{0,\dots,p-1\}$ such that $<Q,\{0,\dots,p-1\}>$ is o-h-reducible to $<S,\{0,\dots,k-1\}>$.*

Proof. The proof is by induction on the factorization length of k. If k is a prime then take $Q := S$. If $k = mn$ for $1 < m,n < k$ then consider the sets $U_0 := \{0,n,2n,\dots,(m-1)n\}, U_1 := \{1,n+1,2n+1,\dots,(m-1)n+1,\},\dots,U_{n-1} := \{n-1,2n-1,3n-1,\dots,mn-1\}$. Two cases (1) and (2) are distinguished:

(1) Assume that for some one $i \in \{0,\dots n-1\}$ the set $S \cap U_i$ is neither empty nor equal to U_i. Define the nonempty set $Q' \subset \{0,\dots,m-1\}$ by $Q' := \{j \mid jn+i \in S \cap U_i\}$. Now the language $<Q',\{0,\dots,m-1\}>$ is o-h-reducible to the language $<S,\{0,\dots,k-1\}>$ via the homomorphism h determined by $h(0) := 0, h(1) := 1^n$ and the offsets 1^i and ϵ: it is easy to see that $1^i h(x) \in\ <U_i,\{0,\dots,k-1\}>$ for all x, and that $x \in\ <Q',\{0,\dots,m-1\}> \iff 1^i h(x) \in\ <S \cap U_i,\{0,\dots,k-1\}>$, this means that $x \in\ <Q',\{0,\dots,m-1\}> \iff 1^i h(x) \in\ <S,\{0,\dots,k-1\}>$.

(2) Assume that for all $i \in \{0,\dots n-1\}$ the set $S \cap U_i$ is either empty or equal to U_i. Then for all numbers $j \in \mathbb{N}$ it holds: j modulo k is equal to a number in S if and only if $j+n$ modulo k is equal to a number in S. Let Q' be the set $S \cap \{0,\dots,n-1\}$. Note that Q' is a nonempty and proper subset of$\{0,\dots,n-1\}$ because S is a nonempty and proper subset of $\{0,\dots,k-1\}$. Now it is easy to see that for all numbers $j \in \mathbb{N}$ it holds that j modulo n is equal to a number in Q' if and only if j modulo k is equal to a number in S, in other words: $<Q',\{0,\dots,n-1\}>\ =\ <S,\{0,\dots,k-1\}>$. Therefore, $<Q',\{0,\dots,n-1\}>$ is o-h-reducible to $<S,\{0,\dots,k-1\}>$ by the reflexivity of the o-h-reducibility.

In both cases (1) and (2) there exists by the induction assumption and by the transitivity of the o-h-reducibility a prime p and a nonempty set $Q \subset \{0,\dots,p-1\}$ such that $<Q,\{0,\dots,p-1\}>$ is o-h-reducible to $<S,\{0,\dots,k-1\}>$. □

7.2 Generalized Definite Languages

The following concept is defined implicitly in Eilenberg [Ei76], see also [Heu89].

Definition 7.2 (Eilenberg 1976) *Call a language L* generalized definite *if there is a natural number n such that for all words x, y of length n and for all words v, w (of any length) it holds*

$$xvy \in L \iff xwy \in L.$$

In other words, a language L is generalized definite if and only if there is a number n such that the membership in L for a word z which has length $\geq 2n$ depends only on the prefix and the suffix of z of length n.

The finite and the cofinite languages are examples of generalized definite languages. The language <P>, which was defined to consist of the words starting with letter 1, is an example of a generalized definite language which is neither finite nor cofinite. Note that a generalized definite language is a regular language, but for example none of the regular languages <NP>, <co-NP> and $<S, \{0, \ldots, k-1\}>$ for a nonempty set $S \subset \{0, \ldots, k-1\}$ for $k \geq 2$ is generalized definite.

7.3 The Main Lemma

The following Lemma 7.3 is a lemma about regular languages, independent of questions about polynomial time computations.

Lemma 7.3 *A regular language R is generalized definite if and only if none of the languages* <NP>, <co-NP>, *and* $<Q, \{0, \ldots, p-1\}>$ *for a nonempty set $Q \subset \{0, \ldots, p-1\}$ for a prime p is o-h-reducible to R.*

Proof. To see the direction $\Longrightarrow$ let <NP> be o-h-reducible to a (regular) language R via two offsets z, z' and an ϵ-free homomorphism h determined by $h(0) = w_0$ and $h(1) = w_2$. Given $n \in \mathbb{N}$, consider the words 0^n00^n and 0^n10^n. Because the first word is not in <NP> and the second is in <NP> one has by the o-h-reducibility that $zh(0^n00^n)z' = zh(0^n)w_0h(0^n)z'$ is not in R and $zh(0^n)w_1h(0^n)z'$ is in R. But the length of $zh(0^n)$ and the length of $h(0^n)z'$ are both $\geq n$. Because this holds for every $n \in \mathbb{N}$ R is not generalized definite. For <co-NP> and

$<Q, \{0, \ldots, p-1\}>$ for a nonempty set $Q \subset \{0, \ldots, p-1\}$ with p prime the proof is analog.

For the other direction of the lemma assume that a regular language R is not generalized definite. It will be shown that one of the languages <NP>, <co-NP>, or $<Q, \{0, \ldots, p-1\}>$ for a nonempty set $Q \subset \{0, \ldots, p-1\}$ for a prime p is o-h-reducible to R.

Let R be accepted by the finite automaton $(Q, \{0, 1\}, \delta, q_0, F)$. For the definition of a finite automaton and the definition of the function $\delta_w : Q \to Q$ (for every word w) see Section 6.2. Assume w.l.o.g. that every state is reachable from q_0, i.e. for every state $q \in Q$ there is a word w such that $\delta_w(q_0) = q$.

Because Q is finite, for every word w and every state q the iteration of δ_w starting in state q has to run into a cycle sometime, more formally: for every word w and every state q there exist two numbers $1 \leq m \leq n$ such that $c_1, \ldots, c_m, \ldots, c_n$ are different states, $c_1 = q$, $c_{i+1} = \delta_w(c_i)$ for $1 \leq i < n$ and $c_m = \delta_w(c_n)$. Assume that for some other word z the set $\{\delta_z(c_m), \ldots, \delta_z(c_n)\}$ has elements from both F and $Q \setminus F$. It is shown that in this case a language of the type $<Q, \{0, \ldots, p-1\}>$ for a nonempty set $Q \subset \{0, \ldots, p-1\}$ for some prime p is o-h-reducible to R: let $k := 1 + n - m$ and define the nonempty set $S \subset \{0, \ldots, k-1\}$ to be the set $\{j - m \mid m \leq j \leq n \text{ and } \delta_z(c_j) \in F\}$. Take a word z' for which $\delta_{z'}(q_0) = c_m$. Define the homomorphism h by $h(0) = w^k$ and $h(1) = w$. Now it is clear that for every word x: $x \in <S, \{0, \ldots, k-1\}> \iff z'h(x)z \in R$, this means that the language $<S, \{0, \ldots, k-1\}>$ is o-h-reducible to R, and by Lemma 7.2 and the transitivity of the o-h-reducibility also a language $<Q, \{0, \ldots, p-1\}>$ for a nonempty set $Q \subset \{0, \ldots, p-1\}$ for some prime p is o-h-reducible to R.

From now on assume that for all states q and for all words w, z like above the set $\{\delta_z(c_m), \ldots, \delta_z(c_n)\}$ consists of states which are either all in F or all in $Q \setminus F$.

Because R is not generalized definite there exist words r, s, t, t' such that r and s have length $|Q|^{|Q|}$ and $rts \in R$ but $rt's \notin R$. There are at most $|Q|^{|Q|}$ mappings $Q \to Q$. Therefore, there exist words s_1, s_2, s_3 such that $s = s_1s_2s_3$, $s_2 \neq \epsilon$ and $\delta_{s_1s_2} = \delta_{s_1}$, i.e. δ_{s_2} is the identity function on the set $\{q' \in Q \mid \exists q \in Q : q' = \delta_{s_1}(q)\}$ - the set of states *reachable* by δ_{s_1}.

Consider for some word u and a state q reachable by δ_{s_1} the set of states $\{c_1, \ldots, c_m, \ldots, c_n\}$ of the iteration of δ_{us_1} starting with $q = c_1$. By assumption the states $\delta_{s_3}(c_m), \ldots, \delta_{s_3}(c_n)$ do belong either all to F or all to $Q \setminus F$. Consider the first case and assume that for some $c_j \in \{c_1, \ldots, c_{m-1}\}$ the state $\delta_{s_3}(c_j)$ is not in F. Then, taking a word z for which $\delta_z(q_0) = c_j$ and defining a homomorphism h by $h(0) = s_2$

and $h(1) = \{us_1\}^m$, it is easy to see that $x \in {<}\text{NP}{>} \iff zh(x)s_3 \in R$, i.e. ${<}\text{NP}{>}$ is o-h-reducible to R. Likewise, ${<}\text{co-NP}{>}$ is o-h-reducible to R if none of the states $\delta_{s_3}(c_m), \ldots, \delta_{s_3}(c_n)$ is in F and there is some $c_j \in \{c_1, \ldots, c_{m-1}\}$ such that $\delta_{s_3}(c_j)$ is in F.

So the only case left is that for each word u and each state q reachable by δ_{s_1} the following holds: $\delta_{s_3}(q) \in F \iff \delta_{us_1s_3}(q) \in F$.

Take the word r from above for which $rts \in R$ but $rt's \notin R$. Because r has length $\geq |Q|$ there exist three words r_1, r_2, r_3 such that $r = r_1r_2r_3$, $r_2 \neq \epsilon$ and $\delta_{r_1}(q_0) = \delta_{r_1r_2}(q_0)$. Define the homomorphism h by $h(0) = r_2$ and $h(1) = r_3ts_1$. It will be shown that $x \in {<}\text{NP}{>} \iff r_1h(x)r_3t's \in R$, i.e. ${<}\text{NP}{>}$ is o-h–reducible to R. The implication $\Leftarrow$ is obvious, and for $\Rightarrow$ consider a word $x \in {<}\text{NP}{>}$, i.e. $x = 0^a1y$ for some $a \in \mathbb{N}$. Then the state $\delta_{r_1h(0^a1)}(q_0)$ is reachable by δ_{s_1} and $\delta_{r_1h(0^a1)s_3}(q_0) = \delta_{r_1r_3ts_1s_3}(q_0) = \delta_{rts}(q_0) \in F$. Therefore, by the above assumption applied to $u = h(y)r_3t'$ and $q = \delta_{r_1h(0^a1)}(q_0)$ also $\delta_{r_1h(0^a1y)r_3t's}(q_0) = \delta_{r_1h(0^a1)h(y)r_3t's_1s_3}(q_0) = \delta_{h(y)r_3t's_1s_3}(\delta_{r_1h(0^a1)}(q_0)) \in F$. □

8 A Result for Classes Accepted by Regular Languages

In Section 8.1 the main result of Part II is presented. The Sections 8.2 and 8.3 will interpret the main result as a non-density result. In Section 8.4 the analog of the main result is shown for the log-space case.

8.1 The Main Result

First the following Lemma 8.1 is shown which can be considered as an easy consequence of the results and methods of Beigel and Gill in [BG92].

Lemma 8.1 (Beigel & Gill 1992) *For a prime p and a nonempty set $Q \subset \{0, \ldots, p-1\}$ the language $<Q, \{0, \ldots, p-1\}>$ accepts* MOD$_p$P.

Proof. The results and methods of [BG92] are applied. Fix a prime p and a nonempty set $Q \subset \{0, \ldots, p-1\}$. Because MOD$_p$P is by definition equal to $<\{1, \ldots p-1\}, \{0, \ldots, p-1\}>$ – P it needs to be proven:

$$<Q, \{0, \ldots, p-1\}> – \text{P} = <\{1, \ldots p-1\}, \{0, \ldots, p-1\}> – \text{P}$$

To show the inclusion from left to right let $\{i_1, \ldots, i_n\}$ be the numbers in $\{0, \ldots, p-1\}$ which are not in Q. Given a nondeterministic machine M construct by Property 2.2. of [BG92] the machine M_1 which for an input x has $(a+p-i_1)(a+p-i_2)\ldots(a+p-i_n)$ accepting paths if M has a accepting paths on input x. Because p is prime for every input x the number of accepting paths of M is equal modulo p to an element of Q iff the number of accepting paths of M_1 is not equal modulo p to 0. Therefore, $<Q, \{0, \ldots, p-1\}>$ – P $\subseteq$ $<\{1, \ldots p-1\}, \{0, \ldots, p-1\}>$ – P.

For the inclusion from right to left let $i \in Q$ and $j \in \{0, \ldots, p-1\} \backslash Q$. Then by Theorem 6.3. of [BG92] there is for every machine M a machine M_2 such that the number of accepting paths of M_2 on an input x is always equal modulo p to either i or j and it is equal to i if and only if the number of accepting paths of M on input x is not equal modulo p to 0. Therefore, $<\{1, \ldots p-1\}, \{0, \ldots, p-1\}>$ – P $\subseteq$ $<Q, \{0, \ldots, p-1\}>$ – P. □

Now the main result is stated.

Theorem 8.1 *Let A be a nontrivial regular language. If A is generalized definite then A* – P = P, *otherwise A* – P *contains at least one of the classes* NP, co-NP, *or* MOD$_p$P *for p prime.*

Proof. Consider a nontrivial regular language A. Assume that A is generalized definite, and let the number n be the constant for A from Definition 7.2, i.e. for a word with length $\geq 2n$ membership in A depends only on its prefix and its suffix of length n. It will be shown that A accepts P. P is of course a subset of A – P, and in order to see that A – P is a subset of P let a polynomial time nondeterministic Turing machine M be given. It suffices to show that $L_A(M)$ is in P. Consider the following deterministic Turing machine D which works in polynomial time. Because A is fixed and n is constant it can be assumed that D has a list of all words in A of length $\leq 2n$. For an input x the machine D first visits by a left traversal (see for example [AHU74]) deterministicly the $2n$ leftmost leaves of $T(M, x)$, note that the left traversal of $T(M, x)$ can be done by simulating M. If D recognizes that the yield of $T(M, x)$ has length $< 2n$ then it terminates, and it terminates with an accepting state if and only if it finds the yield of $T(M, x)$ in its list of words belonging to A. If D recognizes that the yield of $T(M, x)$ has length $\geq 2n$ it memorizes the prefix v of length n of the yield of $T(M, x)$, and visits with a right traversal the n rightmost leaves of $T(M, x)$ in order to find the suffix w of length n of the yield of $T(M, x)$. Finally, D looks up in its list whether the concatenation vw

belongs to A, and accepts if and only if this is the case. By construction and by the property of A to be generalized definite it follows that D accepts the input x if and only if the yield of $T(M, x)$ is in A. In other words, D accepts $L_A(M)$. This shows that $L_A(M)$ is in P. Because this holds for every polynomial time nondeterministic Turing machine M the class A – P is a subset of P.

If on the other hand A is not generalized definite then by Lemma 7.3 at least one of the languages <NP>, <co-NP> or $<Q, \{0, \ldots, p-1\}>$ for a nonempty set $Q \subset \{0, \ldots, p-1\}$ for a p prime is o-h-reducible to A. Therefore, by Lemma 7.1 at least one of the classes <NP> – P = NP, <co-NP> – P = co-NP or $<Q, \{0, \ldots, p-1\}>$ – P for a nonempty set $Q \subset \{0, \ldots, p-1\}$ for a p prime is contained in A – P. By Lemma 8.1, at least one of the classes NP, co-NP, or MOD$_p$P is contained in A – P. □

The theorem can be stated in the following weaker form in which the notion *generalized definite* is not used.

Corollary 8.1 *Let A be a nontrivial regular language. If A –* P *is not equal to* P *then A –* P *contains at least one of the classes* NP, co-NP, *or* MOD$_p$P *for p prime.*

8.2 A Non-Density Result on the Assumption that PH does not Collapse

Remember that PH is the union of the classes Σ_i^p of the Polynomial Time Hierarchy. Say that PH *collapses to* Σ_i^p if PH = Σ_i^p, and say that PH *collapses* if there is some $i \in \mathbb{N}$ such that PH collapses to Σ_i^p.

The following Lemma 8.2 can be seen as an easy consequence of the results of Toda in [To91].

Lemma 8.2 (Toda 1991) *If* MOD$_p$P *for some prime p is contained in* NP *or* co-NP *then PH collapses to Σ_2^p.*

Sketch of proof. Consider the case $p = 2$. Assuming $\oplus$P $\subseteq$ NP one has with the notation of [KST93] (BP$\cdot$ is the operator corresponding to BPP, i.e. BP $\cdot$ P = BPP):

$$\text{PH} \subseteq \text{BP} \cdot \oplus\text{P} \subseteq \text{BP} \cdot \text{NP} \subseteq \Pi_2^p.$$

The first inclusion holds by a result in [To91], the second inclusion holds by the assumption, and the third inclusion holds by a result in [Ba85]. Therefore, PH = $\Pi_2^p = \Sigma_2^p$. The same argumentation goes through for primes $p \neq 2$, see [TO92]. Because MOD$_p$P is closed under complements, see [BG92], the lemma also holds for the assumption $\oplus$P $\subseteq$ co-NP. □

The assumption that PH does not collapse is stronger than the assumption P $\neq$ NP but still can be considered reasonable. The next corollary states that a nondensity-result would follow as a consequence.

Corollary 8.2 *If PH does not collapse (to Σ_2^p) then* NP *and* co-NP *are two atoms of the inclusion order on* $\mathcal{R}$ – P.

Proof. First note that if PH does not collapse (to Σ_2^p) then P, NP, and co-NP are different from each other. Now assume that a class L – P for a language $L \in \mathcal{R}$ is properly between P and NP. Because in that case L – P is not equal to P the class L – P contains, by the previous Theorem 8.1, one of the classes NP, co-NP, MOD$_p$P for p prime. By the assumption L – P cannot contain NP, and also it cannot contain co-NP, because then NP would properly contain its set of complements, a set-theoretic contradiction. So the only case left is that L – P contains a class MOD$_p$P for p prime. But then also NP contains MOD$_p$P, and PH collapses to Σ_2^p by the previous Lemma 8.2. This shows that if PH does not collapse (to Σ_2^p) then there cannot be a class in $\mathcal{R}$ – P properly between P and NP. The same argumentation holds for co-NP. □

8.3 A Non-Density Result for the Relativized Case

Consider the relativized versions of the classes Σ_i^p and Σ_{i+1}^p of the Polynomial Time Hierarchy. For all oracles the first class is a subset of the second but there is an oracle for which this inclusion is proper, see [BGS75, St77, Has86]. The same holds for the relativized versions of many pairs of complexity classes. This concept of comparing complexity classes was formalized by Zachos in [Za88] to define a partial order on relativizable complexity classes which expresses that an inclusion is oracle independent. This concept will be presented now.

Obviously the definitions of Section 6.1 can be relativized for every oracle X, see Section 5.5. This way for each language A and each oracle X the class A – P^X is defined. Let a *family* be a mapping from the set of oracles to classes. Families will be indicated by parenthesis around the oracle variable, for example the family which maps an oracle X to the class NP^X will just be denoted by NP$^{(X)}$. This way each language A defines the family $A-\mathrm{P}^{(X)}$, say that A *accepts the family* $A-\mathrm{P}^{(X)}$

On the set of families accepted by nontrivial regular languages the partial order $\rightarrow$ will be defined. Let A, B be two languages, define $A-\mathrm{P}^{(X)} \rightarrow B-\mathrm{P}^{(X)}$ if $A-\mathrm{P}^X \subseteq B-\mathrm{P}^X$ holds for every oracle X. The partial order $\rightarrow$ corresponds to the idea of oracle independent inclusion of relativizable complexity classes. The concept and the symbol $\rightarrow$ is the same as the one of Zachos in [Za88] though here the definition is for families instead of classes.

Let $\mathcal{R}-\mathrm{P}^{(X)}$ be the set of families which are accepted by a nontrivial regular language.

Proposition 8.1 *The partial order $\rightarrow$ on $\mathcal{R}-\mathrm{P}^{(X)}$ is an upper semi–lattice which has a minimum, a maximum, an infinite chain and an infinite antichain.*

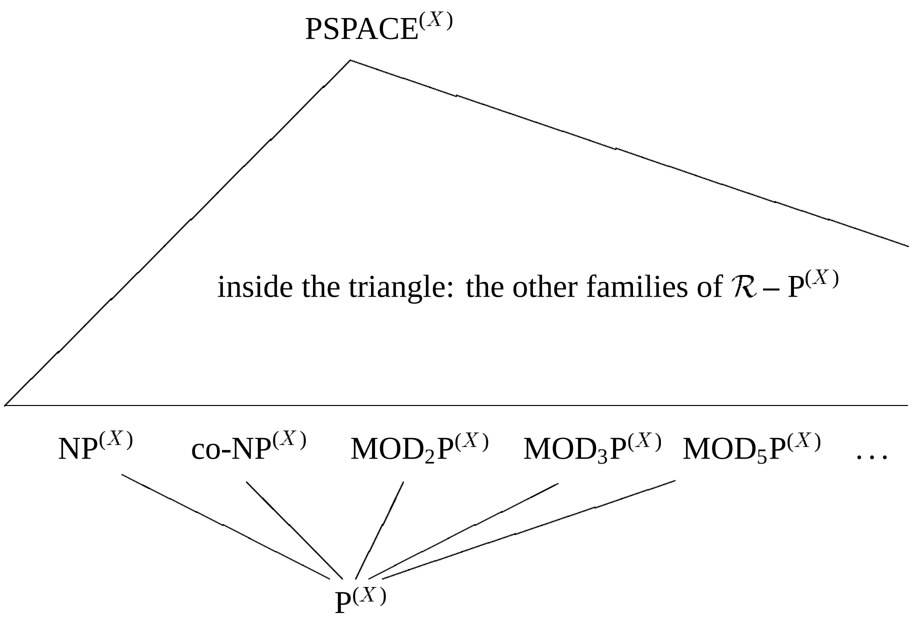
Proof. The upper semi-lattice part holds by the oracle-independent construction of $A \oplus B$ in the proof of Proposition 6.1. The minimum and maximum are P$^{(X)}$ and PSPACE$^{(X)}$, respectively, by Theorem 6.1 which is relativizable. The chain is given by the families $\Sigma_i^{p,(X)}$ because $\Sigma_i^{p,(X)} \rightarrow \Sigma_{i+1}^{p,(X)}$ was shown for every $i \in \mathbb{N}$ in [St77], and $\Sigma_i^{p,(X)} \not\rightarrow \Sigma_{i+1}^{p,(X)}$ by the results in [BGS75, Has86]. To obtain an antichain consider the families MOD$_p$P$^{(X)}$ for p prime: by a result in [BG92] there exists for any two primes $p \neq q$ an oracle X such that MOD$_p$P^X is not a subset of MOD$_q$P^X, what is another way of saying that the families MOD$_p$P$^{(X)}$ for p prime are pairwise $\rightarrow$–incomparable. $\square$

A natural question for a given partial order is to ask about density, see for example [La75]. The following result says that the partial order $\rightarrow$ on $\mathcal{R}-\mathrm{P}^{(X)}$ is atomic and therefore not dense, see Figure 8.3. The result is called *corollary*$^\circ$ because its proof is nearly the same as the proof of Theorem 8.1.

Corollary 8.3 *The upper semi-lattice $\rightarrow$ on $\mathcal{R}-\mathrm{P}^{(X)}$ is atomic. The atoms are the pairwise different families* NP$^{(X)}$, co-NP$^{(X)}$ *and* MOD$_p$P$^{(X)}$ *for p prime.*

Proof. By the results in [BGS75, Yao85, Bei91, Tor91] the families NP$^{(X)}$, co-NP$^{(X)}$ and MOD$_p$P$^{(X)}$ for p prime are pairwise incomparable and therefore they are different from P$^{(X)}$. The corollary is now proven by a relativized version of the proof of Theorem 8.1. $\square$

Figure 8: $\rightarrow$ shown as a diagram


$\mathrm{PSPACE}^{(X)}$
inside the triangle: the other families of $\mathcal{R}-\mathrm{P}^{(X)}$
$\mathrm{NP}^{(X)}$
$\text{co-NP}^{(X)}$
$\mathrm{MOD}_2\mathrm{P}^{(X)}$
$\mathrm{MOD}_3\mathrm{P}^{(X)}$
$\mathrm{MOD}_5\mathrm{P}^{(X)}$
...
$\mathrm{P}^{(X)}$

8.4 An Analogous Result for the Log-Space Case

Let a *log-space nondeterministic Turing machine* be a nondeterministic Turing machine (of the kind described in the introduction) for which there is a constant c such that for an input x the computation terminates on every path and does not use more than $c \cdot \log_2(|x|)$ cells of the working tape on every path. Because every log-space nondeterministic Turing machine M is a polynomial time one, the computation tree $T(M, x)$ for an input x and the language $L_F(M)$ for a predicate F on computation trees is already defined. Let the *log-space predicate class accepted by* F, in short $F-\mathrm{L}$, be the set of languages $L_F(M)$ such that M is a log-space nondeterministic Turing machine. Let $A-\mathrm{L}$ be the log-space predicate class accepted by Y_A.

With identical proofs the Lemmata 7.1 and 8.1 have their following analoga 8.3 and 8.4 for the log-space case. For the proof of Lemma 8.4 results from [BD*92] (instead from [BG92]) are applied.

Lemma 8.3 *Let A, B be two languages. If A is o-h-reducible to B then* A– L $\subseteq$ B– L .

Lemma 8.4 *For a nonempty set $Q \subset \{0, \ldots, p-1\}$ for a prime p the language* $<Q, \{0, \ldots, p-1\}>$ *accepts* MOD_pL.

The following corollary is the log-space analog of Theorem 8.1, it is stated in the form of Corollary 8.1.

Corollary 8.4 *Let A be a nontrivial regular language. If* A– L *is not equal to* L *then* A– L *contains at least one of the classes* NL *or* MOD_pL *for p prime.*

Sketch of proof. The proof is basically the same as the one for 8.1. If A is generalized definite then it follows A– L = L with the analog argumentation like in the proof of Theorem 8.1, besides that the left and the right traversal algorithm have to be done with a look-ahead of $2n$ and n nodes, respectively.

If A is not generalized definite then the same argumentation with Lemma 7.3 applies like in the proof of Theorem 8.1, using Lemmata 8.3 and 8.4. Additionally it is known from [Im88, Sz88] that NL = co-NL. □

References

[Ad78] L. Adleman. *Two theorems on random polynomial time*, Proc. 19th IEEE Symp. on Foundations of Computer Science, 1978, pp. 75–83

[AHU74] A. V. Aho, J. E. Hopcroft, J. D. Ullman. *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974

[Al86] E. W. Allender. *The complexity of sparse sets in P*, 1st Structure in Complexity Theory Conference, Lecture Notes in Computer Science 223, Springer Verlag, 1986, pp. 1-11

[AS85a] K. Ambos-Spies. *On the structure of the polynomial time degrees of recursive sets*, Habilitationsschrift, Universität Dortmund, 1985

[AS85b] K. Ambos-Spies. *Sublattices of the polynomial time degrees*, Information and Control **65**, 1985, pp. 63–84

[AS86a] K. Ambos-Spies. *A note on complete problems for complexity classes*, Information Processing Letters **23**, 1986, pp. 227–230

[AS86b] K. Ambos-Spies. *Minimal pairs for polynomial time reducibilities*, Computation Theory and Logic, Lecture Notes in Computer Science 270, Springer Verlag, 1986, pp. 1–13

[AS89] K. Ambos-Spies. *On the relative complexity of hard problems for complexity classes without complete problems*, Theoretical Computer Science **63**, 1989, pp. 43–61

[Ba85] L. Babai. *Trading group theory for randomness*, Proceedings of the 17th ACM Symposion on Theory of Computing, 1985, pp. 421–429

[BGS75] T. Baker, J. Gill, R. Solovay. *Relativizations of the P=NP? question*, SIAM Journal of Computing **4**, 1975, pp. 431–442

[BDG88] J. Balcazar, J. Diaz, J. Gabarro. *Structural Complexity I*, Springer Verlag, 1988

[Bei91] R.Beigel. *Relativized counting classes: relations among thresholds, parity and mods*, Journal of Computer and System Science **42**, 1991, pp. 76–96

[BG92] R. Beigel, J. Gill. *Counting Classes: thresholds parity, mods, and fewness,* Theoretical Computer Science **103**, 1992, pp. 3–23

[BGu82] A. Blass, Y. Gurevich. *On the unique satisfiability problem*, Information and Control **55**, 1982, pp. 80–88

[Bo94a] B. Borchert. *On the acceptance power of regular languages*, Proc. 11th Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Computer Science 775, Springer Verlag, 1994, pp. 533–542

[Bo94b] B. Borchert. *Predicate classes and promise classes*, Proc. 9th Structure in Complexity Theory Conference, 1994, pp. 235–241

[BCS91] D. P. Bovet, P. Crescenzi, R. Silvestri. *Complexity classes and sparse oracles,* Proc. 6th IEEE 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, pp. 263–283

[BD*92] G. Buntrock, C. Damm, U. Hertrampf, C. Meinel. *Structure and importance of logspace-MOD classes*, Mathematical Systems Theory **25**, 1992, pp. 223–237

[CG*88] J. Y. Cai, Th. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, G. Wechsung. *The Boolean Hierarchy 1: structural properties,* SIAM Journal on Computing **17**, No. 6, 1988, pp. 1232–1252

[CKS81] A. K. Chandra, D. C. Kozen, L. J. Stockmeyer. *Alternation*, Journal of the ACM **28**, 1981, pp. 114–133

[Co71] S. A. Cook. *The complexity of theorem proving procedures*, Proc. 3rd Annual ACM Symposium on the Theory of Computing (STOC), 1971, pp. 151–158

[Ei76] S. Eilenberg. *Automata, languages, and machines,* Volume B, Academic Press, New York, 1976

[ESY84] S. Even, A. Selman, Y. Yacobi. *The complexity of promise problems with applications to public-key crytography*, Information and Control **61(2)**, 1984, pp. 159–173

[FFK94] S. A. Fenner, L. J. Fortnow, S. A. Kurtz. *Gap-definable counting classes*, Journal of Computer and System Sciences **48**, 1994, pp. 116–148

[GJ79] M. R. Garey, D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, San Francisco, 1979

[Gi77] J. Gill. *Computational complexity of probabilistic Turing machines*, SIAM Journal of Computing **6**, 1977, pp. 675–695

[Gr78] G. Grätzer. *General Lattice Theory*, Birkhäuser Verlag, Basel, 1978

[GW87] T. Gundermann, G. Wechsung. *Counting Classes with Finite Acceptance Types*, Computers and Artificial Intelligence **6** No. 5, 1987, pp. 395–409

[HHT92] Y. Han, L. Hemachandra, T. Thierauf. *Threshold computation and cryptographic security*, Technical Report No. 443, Department of Computer Science, University of Rochester, 1992

[Har78] J. Hartmanis. *Feasable Computations and Provable Complexity Properties*, CBMS-NSF Regional Conference Series in Applied Mathematics, Society for Industrial and Applied Mathematics, Philadelphia, 1978

[HH88] J. Hartmanis, L. A. Hemachandra. *Complexity classes without machines: on complete languages for UP*, Theoretical Computer Science **58**, 1988, pp. 129–142

[HI85] J. Hartmanis, N. Immerman. *On complete problems for NP ∩ co-NP*, 12th International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Computer Science 194, Springer Verlag, 1985, pp. 250–259

[HS65] J. Hartmanis, R. E. Stearns. *On the computational complexity of algorithms*, Transactions of the American Mathematical Society **117**, 1965, pp. 285–306

[Has86] J. Hastad. *Almost Optimal Lower Bounds for Small Depth Circuits*, Proceedings of the 18th ACM Symposium on Theory of Computing (STOC), 1986, pp. 6–20

[Hem88] L. A. Hemachandra. *Structure of complexity classes: separations, collapses and completeness*, Proceedings Mathematical Foundations of Computer Science (MFCS), Lecture Notes in Computer Science 324, Springer Verlag, 1988, pp.59–72

[Her90] U. Hertrampf. *Relations among mod–classes*, Theoretical Computer Science **74**, 1990, pp. 325–328

[Her92a] U. Hertrampf. *Locally definable acceptance types for polynomial time machines*, Proc. 9th Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Computer Science 577, Springer Verlag, 1992, pp. 199–207

[Her94a] U. Hertrampf. *Complexity classes with finite acceptance types* Proc. 11th Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Computer Science 775, Springer Verlag, 1994, pp. 543–553

[Her94b] U. Hertrampf. *Complexity classes defined via k-valued functions*, Proc. 9th Structure in Complexity Theory Conference, 1994, pp. 224–234

[HL*93] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, K. Wagner. *On the power of polynomial time bit-computations*, Proc. 8th Structure in Complexity Theory Conference, 1993, pp. 200–207

[HVW94] U. Hertrampf, H. Vollmer, K. W. Wagner. *On balanced vs. unbalanced computation trees*, Technical Report No. 82, Institut für Informatik, Universität Würzburg, 1994

[Heu89] U. Heuter. Generalized definite tree languages, Proc. Conference on Mathematical Foundations of Computer Science (MFCS), Lecture Notes in Computer Science 379, Springer Verlag, 1989, pp. 270–280

[HU79] J. Hopcroft, J. Ullman. *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979

[Im88] N. Immerman. *Nondeterministic space is closed under complementation*, SIAM Journal of Computing **17**, 1988, pp. 935–938.

[JMT94] B. Jenner, P. McKenzie, D. Thérien. *Logspace and logtime leaf languages*, Proc. 9th Structure in Complexity Theory Conference, 1994

[Jo90] D. S. Johnson. *A catalog of complexity classes*, in: J. van Leeuwen, ed., Handbook of Theoretical Computer Science, Volume A, North-Holland, Amsterdam 1990

[Ka72] R. Karp. *Reducibility among combinatorial problems*, in: R. E. Miller and J. W. Thatcher, eds., Complexity of Computer Computations, Plenum, New York, 1972, pp. 85–103

[KL80] R. M. Karp, R. J. Lipton. *Some connections between nonuniform and uniform complexity classes*, Proc. 12th Annual ACM Symposium on Theory of Computing (STOC), 1980, pp. 302–309

[KL82] R. M. Karp, R. J. Lipton. *Turing machines that take advice*, Enseignement Mathématique **28**, 1982, pp. 191–209

[Kl56] S. C. Kleene. *Representation of events in nerve nets and finite automata*, Automata Studies, Princeton University Press, Princeton, 1956, pp. 3-42

[KST89] J. Köbler, U. Schöning, J.Torán. On counting and approximation, Acta Informatica **26**, 1989, pp. 363–379

[KST93] J. Köbler, U. Schöning, J.Torán. *The Graph Isomorphism Problem: Its Structural Complexity*, Birkhäuser Verlag, 1993

[Kow84] W. Kowalczyk. *Some connections between representability of complexity classes and the power of formal systems of reasoning*, Proc. Conference on Mathematical Foundations of Computer Science (MFCS), Lecture Notes in Computer Science 176, Springer Verlag, 1984, pp. 364-369

[Kr88] M. W. Krentel. *The complexity of optimization problems*, Journal of Computer and System Sciences **36**, 1988, pp 490–509

[La75] R. Ladner. *On the structure of polynomial-time reducibilities*, Journal of the ACM **22**, 1975, pp. 155–171

[Le73] L. Levin. *Universal sequential search problems*, Problems of Information Transmission **9**, 1973, pp. 265-266

[MP43] W. S. McCulloch, W. Pitts. *A logical calculus of the ideas immanent in nervous activity*, Bull. Math. Biophysics **5**, 1943, pp. 115–133

[NR93] R. Niedermeier, P.Rossmanith. *Extended locally definable acceptance types*, 10th Symposium on Theoretical Aspects of Computer Science (STACS), Notes in Computer Science 665, Springer Verlag, 1993, pp. 473–483

[PY82] C. H. Papadimitriou, M. Yannakakis. *The complexity of facets (and some facets of complexity)*, Proc. 14th Annual ACM Symposium on the Theory of Computing (STOC), 1982, pp. 255–260

[PZ83] C.H. Papadimitriou, S.K. Zachos. *Two remarks on the power of counting*, 6th GI Conference on Theoretical Computer Science, Lecture Notes in Computer Science 145, Springer Verlag, 1983, pp. 269–276

[Pa84] C. H. Papadimitriou. *On the complexity of unique solutions*, Journal of the ACM **31** No. 2, 1984, pp. 392–400

[Se88] A. L. Selman. *Promise Problems Complete for Complexity Classes*, Information and Computation **78**, 1988, pp. 87–98

[ShS90] J. Shinoda, T. A. Slaman. *On the theory of the PTIME degrees of the recursive sets*, Journal of Computer and System Sciences **41**, 1990, pp. 321–366

[Si82] M. Sipser. *On relativization and the existence of complete sets*, 9th International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Computer Science 140, Springer Verlag, 1982, pp. 523–531

[St77] L. Stockmeyer. *The polynomial-time Hierarchy*, Theoretical Computer Science **3**, 1977, pp. 23–33

[Sz88] R. Szelepcsenyi. *The method of forced enumeration for nondetermistic automata*, Acta Informatica **26**, 1988, pp. 279-284

[To91] S. Toda. *PP is as hard as the Polynomial Time Hierarchy*, SIAM Journal on Computing **20**, 1991, pp. 865–877

[TO92] S. Toda, M. Ogiwara. *Counting classes are at least as hard as the Polynomial Time Hierarchy*, SIAM Journal of Computing **21**, 1992, pp. 316–328

[Tor91] J. Toran. *Complexity classes defined by counting quantifiers*, Journal of the ACM **38**, 1991, pp. 753–774

[Va76] L. G. Valiant. *The relative complexity of checking and evaluating*, Information Processing Letters **5**, 1976, pp. 20–23

[Va79] L. G. Valiant. *The complexity of computing the permanent*, Theoretical Computer Science **8**, 1979, pp. 189–201

[Ve93] N. K. Vereshchagin. *Relativizable and nonrelativizable theorems in the polynomial theory of algorithms* (Russian), Izvestija Rossijskoj Akademii Nauk **57** No. 2, 1993, pp. 51–90 (an English translation is available as a manuscript and is to appear in 1994)

[Vo94a] H. Vollmer. *On different reducibility notions for function classes*, Proc. 11th Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Computer Science 775, Springer Verlag, 1994, pp. 449–460

[Vo94b] H. Vollmer. *Komplexitätsklassen von Funktionen*, Dissertation (Ph.D. Thesis), Universität Würzburg, 1994

[Wa86a] K. W. Wagner. *Some observations on the connection between counting and recursion*, Theoretical Computer Science **47**, 1986, pp. 131–147

[Wa86b] K. W. Wagner. *The complexity of combinatorial problems with succinct input representation*, Acta Informatica **23**, 1986, pp. 325–356

[Wa87] K. W. Wagner. *More complicated questions about maxima and minima, and some closures of NP*, Theoretical Computer Science **51**, 1987, pp. 53–80.

[Wa90] K. W. Wagner. *Bounded query classes*, SIAM Journal of Computing **19**, No. 5, 1990, pp. 833-846

[Wr77] C. Wrathall. *Complete sets and the Polymomial-Time Hierachy*, Theoretical Computer Science **3**, 1977, pp. 23–33

[Yao85] A. Yao. *Separating the Polynomial Time Hierarchy by Oracles*, Proc. 26th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1985, pp. 1-10

[Za88] S. Zachos. *Probabilistic Quantifiers and Games*, Journal of Computer and System Sciences **36**, 1988, pp. 433-451

Subject Index

Symbol Index

Index of Classes

Lebenslauf

Name	Hermann Bernd Borchert
Geburtsdatum	geboren am 21. August 1962 in Thuine/Emsland, Niedersachsen
Eltern	Bernhard Borchert, Maschinenbau-Ingenieur, und Paula Borchert
Staatsangehörigkeit	deutsch
Familienstand	ledig
Mai 1982	Abitur am Gymnasium Georgianum Lingen/Ems
Juli 1982 – Sept. 1983	Wehrdienst
Okt. 1982 – Dez. 1990	Studium der Mathematik und Informatik in Hagen, Münster, München, Boston und Heidelberg
Oktober 1984	Vordiplom in Mathematik
Mai 1988	Master of Arts in Computer Science, Boston University
Dezember 1988	Diplom in Informatik, FernUniversität Hagen
Dezember 1990	Diplom in Mathematik, Universität Heidelberg
Mai 1989 – Sept. 1991	Freier Mitarbeiter bei IBM Heidelberg
Okt. 1991 – Sept. 1992	LGFG Stipendium des Landes Baden-Württemberg
Jan. 1992 – Juni 1992	Aufenthalt an der Cornell University, Ithaca, New York
seit Okt. 1992	Assistent am Lehrstuhl für Logik am Mathematischen Institut der Universität Heidelberg