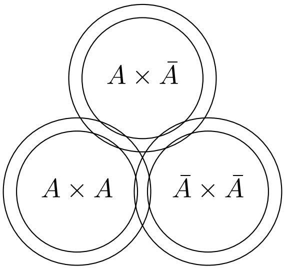
On Structural Similarities of Finite Automata and Turing Machine Enumerability Classes

Till Tantau



On Structural Similarities of Finite Automata and Turing Machine Enumerability Classes

vorgelegt von
Diplom-Informatiker und Diplom-Mathematiker

Till Tantau

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Bernd Mahr
Berichter:	Prof. Dr. Dirk Siefkes
Berichter:	Prof. Dr. Johannes Köbler

Tag der wissenschaftlichen Aussprache: 17. Februar 2003

Berlin 2003

D 83

Preface

This dissertation presents the lion's share of the research I did since I started working at the Technical University of Berlin at the end of 1999. Unfortunately, since I failed to focus my research on a single subject during the last three years, the results of several interesting research projects could not be included. If I had included them, the title would have had to be 'On a Bunch of Interesting, but Unrelated Theorems of Theoretical Computer Science'. When I went to my advisor Dirk Siefkes at the beginning of 2002, we pondered on which papers would be fit to form the basis of a dissertation. There were two alternatives: either two papers on a new concept, namely enumerability by finite automata; or different technical reports on reducibility to selective languages, prover-verifier protocols, and reachability problems. Reducibility classes of selective languages would have been more 'en vogue' and the results applicable in standard complexity theory and in practice (like a constant parallel time reachability algorithm for tournaments). In the end, the mathematical beauty of results that are presented in the following won over.

I have presented the central theorems of this dissertation at two conferences in 2002, namely at the 19th Symposium on Theoretical Aspects of Computer Science in Antibes–Juan les Pins, France, and at the 27th Symposium on Mathematical Foundations of Computer Science in Warsaw, Poland. This dissertation also includes results that were presented at other conferences and in technical reports, but these are not at the core of my topic. Compared to the two main conference papers, this dissertation focusses on demonstrating how the main results are part of broader contexts.

This text is best read front to back, but each chapter is as self-contained as possible. Especially the fifth chapter, which is a 'tutorial' on a new diagonalisation method, can be read independently of the other chapters. Notations and special terminology are explained directly preceding their first use and they are also explained in the list of notations.

There are many people whom I have to thank, since without them this dissertation would have been much worse. My 'being indebted' relation on them forms a partial ordering, but I decided to linearise this ordering to simplify its presentation, see theorems 2.38 and 2.39 for more details on linearisations. So, in alphabetical ordering I am deeply grateful to Sebastian Bab, Lane Hemaspaandra, Chris Homan, Johannes Köbler, Carsten Lenz, Arfst Nickelsen, Margit Russ, Birgit Schelm, Dirk Siefkes, Gerda Tantau, Karl Tantau, Leen Torenvliet, and Tux.

Till Tantau
Berlin, autumn of 2002

Abstract

There are different ways of measuring the complexity of functions that map words to words. Well-known measures are time and space complexity. *Enumerability* is another possible measure. It is used in recursion theory, where it plays a key rôle in bounded query theory, but also in resource-bounded complexity theory, especially in connection with nonuniform computations. This dissertation transfers enumerability to automata theory. It is shown that enumerability behaves similarly in recursion theory and in automata theory, but differently in complexity theory.

The enumerability of a function f is the smallest m such that there exists an m-enumerator for f. An *m-enumerator* is a machine that produces, for every input word w, a set of up to m possibilities for $f(w)$. By varying the parameter m and the class of allowed enumerators, different enumerability classes can be defined. In recursion theory, one allows arbitrary Turing machines as enumerators; in automata theory, only finite automata. A deep structural result that holds both for finite automata and for Turing machine enumerability is the following *cross product theorem*: if $f \times g$ is $(n+m)$-enumerable, then either f is n-enumerable or g is m-enumerable. In contrast, this theorem does not hold for polynomial-time enumerability.

Enumerability can be used to quantify the difficulty of a *language* A by asking how difficult it is to enumerate its n-fold characteristic function χ_A^n and cardinality function $\#_A^n$. A language is *(m,n)-verbose* if χ_A^n is m-enumerable. The inclusion structures of Turing machine and of finite automata verboseness classes are identical: all (m,n)-Turing-verbose languages are (h,k)-Turing-verbose iff all (m,n)-finite-automata-verbose languages are (h,k)-finite-automata-verbose. The structure of polynomial-time verboseness classes is different.

The enumerability of $\#_A^n$ has been studied in detail in recursion theory. Kummer's cardinality theorem states that if $\#_A^n$ is n-enumerable by a Turing machine, then A must be recursive. Evidence is gathered that this theorem also holds for finite automata: it is shown that the nonspeedup theorem, the cardinality theorem for two words, and the restricted cardinality theorem all hold for finite automata. The cardinality theorem does not hold for polynomial-time computations.

The central proofs rely on two proof techniques that promise to be applicable in other situations as well: generic proofs and branch diagonalisation. Generic proofs use elementary definitions, a concept from logic, to define enumerators in terms of other enumerators. They can be instantiated for all computational models that are closed under elementary definitions. Examples of such models are finite automata, but also Presburger arithmetic and ordinal number arithmetic. The second technique is a new diagonal-

isation method, where machines are tricked on codes of diagonalisation decision sequences, rather than on codes of machines. Branch diagonalisation is not applicable universally, but where it is applicable, it can be used to diagonalise against Turing machines, using only finite automata.

Results on enumerability classes have applications in unrelated areas, like finite automata protocol testing, classification problems where examples are provided, and separability. An intriguing example of such an application is the following theorem: if there exist regular supersets of $A \times A$, $A \times \bar{A}$, and $\bar{A} \times \bar{A}$ whose intersection is empty, then A is regular.

Die Komplexität von Funktionen, die Worte auf Worte abbilden, kann auf verschiedene Arten gemessen werden. Bekannte Maße sind Zeit- und Platzkomplexität. *Aufzählbarkeit* ist ein weiteres Komplexitätsmaß. Sie wird in der Rekursionstheorie eingesetzt, wo sie eine zentrale Rolle in der Theorie fragenbeschränkter Reduktionen spielt, sowie in der ressourcenbeschränkten Komplexitätstheorie, insbesondere in Verbindung mit nichtuniformen Berechnungen. In dieser Arbeit wird das Konzept der Aufzählbarkeit auf endliche Automaten übertragen. Es wird gezeigt, dass sich Aufzählbarkeit in der Rekursionstheorie und in der Automatentheorie gleichartig verhält, in der Komplexitätstheorie hingegen andersartig.

Die Aufzählbarkeit einer Funktion f ist die kleinste Zahl m, für die ein m-Aufzähler für f existiert. Ein *m-Aufzähler* ist eine Maschine, die bei Eingabe eines Wortes w eine Menge von höchstens m Möglichkeiten für $f(w)$ ausgibt. Verschiedene *Aufzählbarkeitsklassen* können durch Veränderung des Parameters m und der Art der erlaubten Maschinen definiert werden. In der Rekursionstheorie erlaubt man beliebige Turingmaschinen als Aufzähler, in der Automatentheorie lediglich endliche Automaten. Ein tiefliegendes strukturelles Resultat, das sowohl für Turingmaschinen als auch für endliche Automaten gilt, ist der folgende *Kreuzproduktsatz*: Ist $f \times g$ eine $(n+m)$-aufzählbare Funktion, so ist f eine n-aufzählbare oder g eine m-aufzählbare Funktion. Dieser Satz gilt nicht im Polynomialzeitfall.

Aufzählbarkeit kann auch benutzt werden, um die Komplexität von *Sprachen* zu quantifizieren. Dazu wird gefragt, wie schwierig es ist, die n-fache charakteristische Funktion χ_A^n und die n-fache Kardinalitätsfunktion $\#_A^n$ einer Sprache A aufzuzählen. Eine Sprache A ist *(m,n)-verbose*, falls χ_A^n m-aufzählbar ist. Die Inklusionsstrukturen der Verbosenessklassen von Turingmaschinen und der von endlichen Automaten sind gleich: alle (m,n)-Turing-verbosen Sprachen sind genau dann (h,k)-Turing-verbose, wenn alle in Bezug auf endliche Automaten (m,n)-verbosen Sprachen (h,k)-verbose sind. Dies gilt nicht im Polynomialzeitfall.

Die Aufzählbarkeit von $\#_A^n$ ist in der Rekursionstheorie wohluntersucht. Kummers Kardinalitätssatz besagt, dass A rekursiv ist, falls $\#_A^n$ von einer Turingmaschine n-aufgezählt werden kann. Vermutlich gilt dieser Satz auch für endliche Automaten: Zumindest der Nonspeedupsatz, der Kardinalitätssatz für zwei Worte und der eingeschränkte Kardinalitätssatz gelten für endliche Automaten. Der Kardinalitätssatz gilt nicht im Polynomialzeitfall.

Die Hauptbeweise in dieser Arbeit benutzen zwei Techniken, deren Einsatz auch in anderen Gebieten vielversprechend erscheint: generische Beweise und Astdiagonalisierung. Generische Beweise benutzen elementare

Definitionen, ein Konzept aus der Logik, um Aufzähler zu definieren. Solche Beweise lassen sich auf alle Berechnungsmodelle anwenden, die unter elementaren Definitionen abgeschlossen sind. Dies ist für endliche Automaten der Fall, aber auch für die Presburgerarithmetik und die Ordinalzahlarithmetik. Die zweite Technik ist eine neue Diagonalisierungsmethode, bei der Maschinen auf dem Kode der bisherigen Folge von Diagonalisierungsentscheidungen ausgetrickst werden und nicht auf ihrem eigenen Kode. Astdiagonalisierung ist nicht universell einsetzbar, aber wo sie eingesetzt werden kann, kann man mit ihrer Hilfe gegen Turingmaschinen mittels endlicher Automaten diagonalisieren.

Die Resultate über Aufzählbarkeitsklassen haben Anwendungen, so bei Protokolltests mittels endlicher Automaten, bei Klassifikationsproblemen mit Beispielen und bei Trennbarkeitsfragen. Ein schönes Beispiel einer solchen Anwendung lautet wie folgt: Existieren regulär Obermengen von $A \times A$, $A \times \bar{A}$ und $\bar{A} \times \bar{A}$, deren Schnitt leer ist, so ist A regulär.

Table of Contents

List of Notations

The notations are sorted alphabetically with Greek symbols inserted according to their English transliteration. Special symbols are put at the front. Modifiers on languages and alphabets, like the star in A^*, are listed at the letter A. Modifiers on words and bitstrings are listed at the letter W. Modifiers on relations are listed at the letter R.

$\lvert . \rvert$	The cardinality of a set. Also the length of a word.
$=_{\text{ae}}$	Denotes that two set are equal almost everywhere, that is, that their symmetric difference is finite.
$<$	Denotes irreflexive well-orderings.
$\leq_{\text{lex}}$, $\leq_{\text{std}}$	The lexicographical (dictionary) and standard (first lengthwise, then lexicographical) orderings of words.
$\sqsubseteq$	Prefix relation on words.
$\prec$, $\preceq$	The irreflexive and reflexive lengthwise preorderings of words; see example 2.35.
$\models$	The modelling relation; see page 39.
$\times$	The cross product of sets as in $A \times B$. Also the cross product of functions as in $f \times g$, which is defined by $(f \times g)(u,v) := \big(f(u), g(v)\big)$.
$\to_{\text{partial}}$	Denotes partial functions as in $f\colon A \to_{\text{partial}} B$.
$(a, b]$	The half-open interval $\{x \in \mathbb{R} \mid a < x \leq b\}$.
$(x_1, \dots, x_n)^{\text{t}}$	Column vector obtained by transposing the row vector $(x_1, \dots, x_n)$.
$\#^n_A$	The cardinality function of the language A, which is defined by $\#^n_A(w_1, \dots, w_n) := \big\lvert\{w_1, \dots, w_n\} \cap A\big\rvert$.
A	A language.
$\bar{A}$	The complement of A, that is, $\bar{A} := \Sigma^* \setminus A$.
A^+	The set of words that can be formed out of words in A, that is, $A^+ := \{w_1 \cdots w_n \mid n \geq 1, w_i \in A\}$.
A^*	The Kleene star of A, that is, $A^* := A^+ \cup \{\epsilon\}$.
A^n	Set of n-tuples of words in A.
$A^{(n)}$	Set of n-tuples of pairwise different words in A; see notation 5.22.
$A^{\binom{n}{k}}$	Set of n-tuples of pairwise different words such that exactly k of them are in A; see notation 5.22.
$A^{\langle n \rangle}$	The tupling alphabet for n words over the alphabet A; see definition 2.8.
α	An assignment; see definition 2.24.

$\operatorname{bin} G$	Binary representation of the graph G.
$\operatorname{bin} M$	Binary representation of the machine M.
$\operatorname{bin} n$	Binary representation of the natural number n.
$\operatorname{bin}_\ell n$	Binary representation of the natural number n padded with leading zeros to a minimum length of ℓ.
$c^{\mathcal{S}}$	The constant symbol c interpreted in the logical structure $\mathcal{S}$; see definition 2.19.
χ_A, χ_A^n	The characteristic function and the n-fold characteristic function of the language A. The ith bit of the bitstring $\chi_A^n(w_1, \dots, w_n)$ is 1 iff $w_i \in A$.
δ, $\hat{\delta}$	The transition function and the extended transition function of a DFA; see Definitions 2.1 and 2.2.
Δ	The transition relation of an NFA; see definition 2.5.
$\operatorname{distinct}(u_1, \dots, u_n)$	Logical formula that expresses that all u_i are pairwise distinct.
$\mathrm{DTIME}[f]$	The class of languages decidable by deterministic Turing machines in time f.
ϵ	The empty word.
$\mathrm{E}_r(C)$	The r-reduction closure of the class C; see page 124.
$\mathrm{EN}_C(n)$	The generic enumerability class of a class C of relations; see definition 3.12.
$\mathrm{EN}_{\mathrm{fa}}(n)$	The class of all n-fa-enumerable functions; see example 3.16.
$\mathrm{EN}_{\mathrm{On}}(n)$	The class of all functions that are n-enumerable in ordinal number arithmetic; see example 3.18.
$\mathrm{EN}_{\mathrm{Pa}}(n)$	The class of all functions that are n-enumerable in Presburger arithmetic; see example 3.17.
$\mathrm{EN}_{\mathrm{re}}(n)$	The class of all n-Turing-enumerable functions; see example 3.14.
F	The set of accepting states of a finite automaton; see definition 2.1.
f^n	A function symbol of arity n; see definition 2.18.
$f^{\mathcal{S}}$	The function symbol f interpreted in the logical structure $\mathcal{S}$; see definition 2.19.
$\mathrm{FDSPACE}[s]$	The class of functions computable by deterministic Turing machines in space s.
FL	The class of functions computable by deterministic Turing machines in logarithmic space.
FP	The class of functions computable by deterministic Turing machines in polynomial time.
$\mathcal{I}_\Sigma$	The index structure over the alphabet Σ; see definition 2.33.

$I_\sigma(u,v)$	True if the $\lvert v\rvert$-th letter of u is σ; see definition 2.33.
$I_\Box(u,v)$	True if the $\lvert v\rvert$-th letter of u 'is a blank'; see example 2.35.
K	The halting problem, that is, $\mathrm{K} = \{\operatorname{bin} M \mid M$ halts on input $\operatorname{bin} M\}$.
$\mathrm{L}(M)$	The language accepted by the automaton M; see definition 2.2. Also the language accepted by the Turing machine M.
$\mathrm{L}\big(M^{()}, X\big)$	The language accepted by the oracle Turing machine $M^{()}$ relative to the oracle X.
M	A deterministic finite automaton. Also a deterministic Turing machine.
$M^{()}$	An oracle Turing machine.
M^X	An oracle Turing machine whose queries are answered by the oracle X.
$M(w)$	The final output of the DFA M on input w; see definition 2.4. Also the output of the Turing machine M on input w.
N	A nondeterministic finite automaton.
$\mathbb{N}$	The set of natural numbers, that is, $\mathbb{N} = \{0, 1, 2, \ldots\}$.
NP	The class of languages that are decidable in polynomial time by nondeterministic Turing machines.
$O(f)$	The class of functions that are 'Big Oh of f'.
ODD^n_A	Set of n-tuples of words such that an odd number of them is in A; see definition 5.27.
P	The class of languages that are decidable in polynomial time by deterministic Turing machines.
P/f	Polynomial-time advice class with $f(\ell)$ advice bits for words of length ℓ; see definition 5.8.
P/k	Polynomial-time advice class with k advice bits per word length; see definition 5.8.
P/poly	Polynomial-time advice class with polynomially many advice bits per word length; see definition 5.8.
P-sel, P^X-sel	The class of sets that have a selector in FP, respectively FP^X; see definition 5.1.
ϕ, ψ	First-order or second-order formulæ.
$\phi(u_1, \ldots, u_n)$	A first-order or second-order formula in which the free variables are $(u_1, \ldots, u_n)$.
$\phi^{\mathcal{S}}(u_1, \ldots, u_n)$	The relation that is elementarily defined by ϕ in $\mathcal{S}$; see definition 2.25.
$\phi_\epsilon(v)$	True if v is the empty word; see example 2.34.

Q	A finite set of states; see definition 2.1.
Q_{initial}	The set of initial states of an NFA; see definition 2.5.
q_{initial}	The initial state of a DFA; see definition 2.1.
$Q_\sigma(i)$	True in a word structure $\mathcal{W}_w$ if the ith letter of w is σ; see definition 2.30.
R	A relation, that is, $R \subseteq U^n$ for some universe U.
$R[u_1, \dots, u_k]$	The set $\{(u_{k+1}, \dots, u_n) \in U^{n-k} \mid (u_1, \dots, u_n) \in R\}$ for an n-ary relation $R \subseteq U^n$; see notation 3.13.
R^n	A relation symbol of arity n; see definition 2.18.
$R^{\mathcal{S}}$	The relation symbol R interpreted in the logical structure $\mathcal{S}$; see definition 2.19.
$\mathrm{R}_r(C)$	The r-reduction closure of the class C; see page 124.
$\mathcal{S}$	A logical structure; see definition 2.19.
$\mathcal{S}_{C\|U}$	Structure for 'talking about' the relations in C that have universe U; see definition 3.20.
σ_*	A fixed element of an alphabet Σ.
Σ	An alphabet, that is, a nonempty finite set.
$\Sigma^{\langle n \rangle}$	The tupling alphabet for n words over the alphabet Σ; see definition 2.8.
SEMIREC	The class of semirecursive sets; see definition 5.1.
τ	A logical signature.
τ_Σ	Signature for word structures over the alphabet Σ; see definition 2.30.
U	Universe of a logical structure; see definition 2.19.
$u_1, u_2, u_3, \dots$	Variables for formal first-order variables.
$\mathrm{v}_1, \mathrm{v}_2, \mathrm{v}_3, \dots$	Formal first-order variables; see definition 2.21.
$\mathrm{V}_1^n, \mathrm{V}_2^n, \mathrm{V}_3^n, \dots$	Formal second-order variables of arity n; see definition 2.23.
$\mathrm{V}_C(m, n)$	The generic class of all (m, n)-verbose languages of a class C of relations; see definition 4.7.
$\mathrm{V}_{\text{fa}}(m, n)$	The class of all (m, n)-fa-verbose languages; see definition 4.7.
$\mathrm{V}_{\text{re}}(m, n)$	The class of all (m, n)-Turing-verbose languages; see definition 4.7.
w	A word, that is, an element of Σ^* for some alphabet Σ.
$w[i_1, \dots, i_k]$	The i_1th, … , i_kth letters of the word w.
w^{reversed}	The word w read backwards.
$\|w\|$	Length of the word w.
$\langle w_1, \dots, w_n \rangle$	The coded tuple of the words w_1, … , w_n; see definition 2.9.
$\mathcal{W}_w$	The word structure of the word w; see definition 2.30.

List of Figures

First Chapter

Introduction

Section 1.1

My Thesis

The enumerability and verboseness classes of finite automata on the one hand and of Turing machines on the other hand share numerous structural properties. They do not share these properties with intermediate resource-bounded computational models. Especially the finite automata versions of these classes have applications in areas unrelated to enumerability. Two methods that are used for the proofs of the structural similarities—elementary definitions of regular relations and branch diagonalisation—will be applicable to other proofs in automata, complexity, and recursion theory.

If you are already convinced of my thesis, you can stop reading now since you have attained my goal. However, I presume that you will accept my thesis (or any other) only after sufficient proof has been given. For this reason, this dissertation contains five chapters that try to provide ample evidence for my thesis. The distinction between 'my thesis' (by which I refer to the above claims) and 'my dissertation' (by which I refer to the whole dissertation essay) is admittedly rather old-fashioned, but hopefully useful: the thesis states succinctly what I try to *convince* you of, the dissertation is my not-so-succinct means of achieving this.

This introductory chapter is organised as follows. In section 1.2 an overview is given of the concepts treated in this dissertation. This overview is only intended to explain the intuition behind the concepts—detailed formal definitions are given in the main text. In section 1.3 the main results that I have obtained are listed. In section 1.4 an overview is given of the methods that are employed in the proofs of the main results. In section 1.5 the organisation of this dissertation is sketched. In section 1.6 I present my personal motivation and the *a priori* motivation for studying the main concepts. An *a posteriori* analysis of the actual relevance of the obtained results is given in the conclusion chapter.

Section 1.2

Concepts of this Dissertation

Three core concepts are treated in this dissertation: *enumerability*, *verboseness*, and *cardinality computations*. They are motivated below. Non-core (though by no means unimportant) concepts like separability, protocol testing, or classification with examples are explained at the beginnings of the chapters that treat them. The central *proof* concepts are explained in the methodology section.

Enumerability

Most results of this dissertation concern the *enumerability of functions.* Functions can be used to formalise problems: problems ask us to produce *solutions* (like the least cost of a sightseeing tour) for *problem instances* (like a city map together with a list of sightseeing points to be visited). The formal mapping of problem instances to solutions is a function. Unfortunately, many interesting functions turn out to be too difficult to compute within the time or space available. Fortunately, we often do not need to compute functions exactly, but require only some sort of *approximation* of the correct value.

An approximation is 'close' to the correct value. In classical approximation theory, see (Ausiello et al., 1999) for an introduction, approximations are within a constant factor of the correct value. For example, for the 'sightseeing tour problem', a good approximation of the cost of an optimal sightseeing tour is a number z that is within a small constant factor α of the cost of the optimal tour. If such a number z is output, the optimal cost is known to lie between z/α and z, that is, it is known to be an element of the set $\{\lceil z/\alpha\rceil, \dots, z\}$.

The idea behind enumerability is to allow more general sets. An *enumerator* for a function f outputs, for every input w, a small set that contains $f(w)$. This set need not be an interval. In particular, it can contain values that are far removed from the correct value. The only requirement is that the set of possibilities is small. For example, consider the function #SAT that maps (the code of) every propositional formula ϕ to (the code of) the number of satisfying assignments of this formula. On input of the formula $p \lor q \lor r$, an enumerator for #SAT might output $\{0, 1, 6, 7\}$, which contains the correct value 7. Another enumerator might produce the set $\{4, 5, 6, 7\}$ or any other set, as long as it contains the number 7. Enumerators are classified according to the size of the sets they enumerate, since a small set is a better 'approximation' than a larger set, and according to the computational resources they use. An enumerator that always outputs sets of size at most m is called an *m-enumerator.*

What functions are easy to enumerate? Certainly, functions having only a small range can easily be enumerated. For example, we can trivially 2-enumerate the characteristic function of every language and we can 4-enumerate the cross product of any two characteristic functions. However, many functions that arise in practice cannot be enumerated easily (possibly unless certain unlikely collapses of complexity classes occur). For example, Cai and Hemachandra (1989) have shown that #SAT cannot be poly-enumerated unless SAT $\in$ P. This means that unless P = NP, we cannot enumerate a set of size polynomial in the length of the input formula that contains the correct number of the formula's satisfying assignments. Beals et al. (1999) have shown that if the function that maps a graph to the number of its automorphisms is poly-enumerable, then the graph isomorphism problem is in RP. Mitsunori Ogihara and myself (2002) have shown that the graph automorphism problem is in P under the same assumption.

Enumerability is a relatively new concept, despite its simplicity and its range of applications. It was introduced thirteen years ago by Cai and Hemachandra (1989) in the context of resource-bounded computations, was transferred to resource-unbounded computations five years later by Kummer and Stephan (1994), and was transferred to finite automata only recently (Tantau, 2002A, B).

Verboseness

The second concept studied is the *verboseness of languages.* Verboseness is closely related to enumerability: the verboseness of a language A quantifies how difficult it is to enumerate the n-fold characteristic function χ_A^n of A. This function takes n words as input, n being some fixed number, and yields a bitstring as output whose ith bit is 1 iff the ith word is an element of A. Characteristic strings tell us exactly which words are in a language and which are not. If the n-fold characteristic function of a language is m-enumerable, the language is called (m,n)-*verbose.*

Studying verboseness means studying the enumerability of functions of a special kind, namely characteristic functions. Results obtained for enumerability classes thus apply directly to verboseness.

Verboseness was originally defined differently, namely in terms of bounded query classes, see (Beigel, 1987), (Gasarch, 1991), and (Gasarch and Martin, 1999). The relationship is the following: if one can compute the n-fold characteristic function of a language by asking k queries to some oracle X, one can also 2^k-enumerate the function without asking any queries; and conversely. Verboseness has been studied extensively for polynomial-time computations. An important result in this context was independently submitted by three research groups to the 1994 Structure in Complexity Theory Conference, see (Agrawal and Arvind, 1996), (Beigel et al., 1995A), and (Ogihara, 1995) for the journal versions. It states that no NP-hard problem can be polynomial-time $(2^n-1,n)$-verbose for any n, unless P = NP. The finite automata version of verboseness classes was first defined in (Tantau, 2002A).

Cardinality Computations

The third central concept studied is *cardinality computations.* In such a computation we do not try to determine *which* input words are in a language, but *how many.* We get n input words and are asked to count the number of words in A among these input words.

This counting problem, raised in its general form by Gasarch (1991), plays an important rôle in a variety of proofs, both in complexity theory (Mahaney, 1982; Immerman, 1988; Szelepcsényi, 1988; Hemachandra, 1989; Kadin, 1989) and recursion theory (Kummer, 1992; Kummer and Stephan, 1994; Beigel et al., 2000). To give just one example: the core idea behind the Immerman–Szelepcsényi theorem is to *decide* the reachability problem by first *counting* the number of reachable vertices in a graph.

The enumerability of cardinality functions is a fruitful subject. A profound result, due to Martin Kummer (1992), states that if the n-fold cardinality function of a language is n-enumerable, then the language must be recursive. This result is known as the cardinality theorem. Extensions in different directions were obtained by Kummer and Stephan (1994) and Nickelsen (1997). The study of the enumerability of cardinality functions by finite automata was started in (Tantau, 2002b).

The three concepts 'enumerability', 'verboseness', and 'cardinality computations' are studied by addressing the following questions:

1. What are the structural properties of enumerability classes for different computational models? That is, how are the classes of m-enumerable functions related?
2. What does the inclusion structure of verboseness classes look like for different computational models? That is, for which numbers m, n, h, and k are all (m, n)-verbose languages also (h, k)-verbose?
3. What can be said about languages whose cardinality function is enumerable by a finite automaton?
4. Which applications of enumerability exist in other areas?

Section 1.3

Results of this Dissertation

About sixty theorems, corollaries, and lemmas are proved in this dissertation. All of them are enlightening in one way or another—otherwise there would be no point in presenting them. In the following I point out only those results that directly support my thesis. Recall that it states that the structures of Turing machine and finite automata enumerability classes are similar, while they are different from the corresponding structure for resource-bounded computations.

1. I prove a purely structural theorem on enumerability classes, which holds both for Turing machines and for finite automata. It states that if the cross product (or, equivalently, the parallel application) of two functions is $(n+m)$-enumerable, then either the first function is n-enumerable or the second function is m-enumerable. Unlike previous results due to Beigel (1987) and Beigel et al. (1995B), this *cross product theorem* is true for *all* functions, not just for functions of a special kind. The theorem does not hold for resource-bounded computations.
2. I prove that, somewhat surprisingly, the intricate inclusion structure of verboseness classes is identical for Turing machines and for finite automata and different from the inclusion structure for resource-bounded Turing machines. This result is derived directly from the cross product theorem and from a branch diagonalisation argument (see below).
3. I prove that different weak forms of Kummer's recursion-theoretic cardinality theorem hold for finite automata. Recall that Kummer's theorem states that if the n-fold cardinality function of a language can be n-enumerated by a Turing machine, then the language must be recursive. I conjecture that not only the weak versions, but also the complete cardinality theorem holds for finite automata. Even the weak forms of the cardinality theorem do not hold for resource-bounded computations.

My thesis states that enumerability classes have applications in seemingly unrelated areas. The following results, which are shown in the sixth chapter, support this claim.

4.1 Enumerability and cardinality computations can be applied to the study of separability. For example, I show that the following statement is equivalent to Kummer's cardinality theorem: if A is a language for which there exist recursively enumerable supersets of $A^{\binom{n}{0}}$, $A^{\binom{n}{1}}$, … , $A^{\binom{n}{n}}$ whose intersection is empty, then A is recursive. Here $A^{\binom{n}{k}}$ is the set of all n-tuples of pairwise different words such that exactly k of them are in A. For finite automata I show that if A is a language for which there exist regular supersets of $A \times A$, $A \times \bar{A}$, and $\bar{A} \times \bar{A}$ whose intersection is empty, then A must be regular. A final example of a result of this type is a counterexample to an old theorem of Kinber (1976): I show that there exist disjoint $(3,5)$-fa-separable languages that are recursively inseparable. Kinber had claimed that such languages must be separable by finite automata.

4.2 Consider a finite automaton that monitors n high-speed data lines. It should tell us which lines are faulty, that is, on which lines some protocol is violated. I show that the automaton must receive at least $\lfloor \log_2 n \rfloor + 1$ bits from some external device in order to compute the set of faulty lines, if the protocol is not regular. This lower bound is tight.

4.3 I show that, for decision problems, 'examples do not help' Turing machines and finite automata, but examples do help in resource-bounded computations. An *n-example decision problem* asks us to decide whether an input w is in a certain language, but we get help in the form of n further examples of words that are also in the language if w is, and that are not in the language if w is not.

Section 1.4

Methodology of this Dissertation

The *way* the main results are proved is perhaps as important as the results themselves. Three techniques are used: the core results are proved in a 'unified' or 'generic' way; regular languages are constructed using elementary definitions; and branch diagonalisation is used to separate classes defined in terms of finite automata from classes defined in terms of Turing machines.

Generic Proofs are Applicable to Different Computational Models

My thesis states that the enumerability and verboseness classes of finite automata and Turing machines share numerous structural properties. This thesis is supported by theorems like theorem 5.19, which states that the inclusion structure of finite automata and Turing machine verboseness classes is *identical.* However, although such theorems tell us that similarities *exist*, they do not tell us *why* they exist. *A priori* there are two opposing possible explanations: finite automata and Turing machines might just happen to share these structures—but just 'by coincidence' and totally different proofs and techniques are needed to establish and prove these structures; or they might have the same structure because essentially the same proof works both for finite automata and for Turing machines.

At least in some cases, the latter is true. I formulate 'generic' theorems that leave the underlying computational model open. They have the form 'for every computational model with certain properties, the structure looks like this: ... '. Since finite automata and Turing machines satisfy these properties, the structures are the same for them. The generic proof pins down *why* this is the case.

For verboseness classes I formulate strong generic theorems that can be instantiated for finite automata and for Turing machines.

For cardinality computations, the situation is (currently) different. The generic theorems formulated for them impose stronger requirements on the computational model: the class of accepted relations must be closed under elementary definitions. While finite automata enjoy this property (see below), Turing machines do not. For this reason, my proofs for the weak forms of the cardinality theorem for finite automata are conceptually different from the proofs in the literature for Turing machines. It might be possible that the weak forms of the cardinality theorem 'happen' to hold for finite automata and for Turing machines, but for different reasons.

One might argue that the generic proof approach is unnecessary for the finite automata versions of the cardinality theorem, since it is not 'generic enough' to work also for Turing machines. There are two reasons why I believe that the generic approach is still useful here. First, there are various interesting computational models that satisfy the strong properties required in the proofs. They include Presburger arithmetic, which has recently had a renaissance in model checking; the arithmetical hierarchy, which is closely linked to Turing machine computations; and in a limited way even ordinal number arithmetic, which is a central tool in set theory. Second, there might be a generic proof of the complete cardinality theorem that works both for Turing machines and for finite automata.

Elementary Definitions of Regular Relations

In many proofs I define regular languages and relations in terms of existing ones and in terms of *first-order formulæ.* Using logic for the specification of languages and problems has a long tradition and numerous characterisations are known. Surveys have been written by Immerman (1999), who treats logical characterisations of complexity classes, or by Ebbinghaus and Flum (1995), who focus on regular languages. I have met theorists who claim that 'a complexity class deserves that name iff it has a logical characterisation'.

Often, the study of logical characterisations of language classes is motivated by the desire to understand the classes better. For example, consider the characterisation of the class P of problems decidable in polynomial time in terms of first-order logic augmented by least fixed point operators. This characterisation pins down the complexity of the class P rather nicely, but it is only very seldomly, if at all, *used* in writing programs. The same is true of Büchi's beautiful second-order characterisation of regular languages: regular languages are commonly defined in terms of finite automata or in terms of regular expressions, but almost never in terms of formulæ in monadic second-order logic. In these cases, the logical formalism is, at best, a theoretical foundation for real specification purposes.

Quite differently, my study of the first-order characterisation of regular languages was directly motivated by a need for an appropriate way of defining regular languages in terms of existing ones. A corollary of the characterisation provides such a way.

The first-order characterisation states that there exists a simple logical structure $\mathcal{I}_\Sigma$, whose universe is Σ^*, such that a language over Σ is regular iff there exists a first-order formula ϕ with one free variable with the following property: in order to 'test' whether a given word is an element of the language, we plug in this word for the free variable in ϕ and then check whether ϕ holds. This characterisation has been known for a long time: it was first reported by Büchi in 1962. However, his conceptually different characterisation of regular languages in terms of monadic second-order logic received much more attention. Nevertheless, work on first-order characterisations of regular languages did not stop, see for example Bruyère and Hansel (1997) for some recent results.

An important corollary of the characterisation states that every first-order formula, interpreted in any logical structure in which all relations are regular, defines a regular language. A relation is *regular* if it is accepted by a finite automaton that gets word tuples as input. The components of the tuple are put on separate tapes, which are read synchronously. For example, (the graph of) the addition function is regular and thus the corollary allows us to make claims like the following one: 'the language $A = \{\operatorname{bin} n \mid n \equiv 0 \mod 7\}$ is regular since it can be defined by $x \in A :\iff \exists y\,\bigl(y + y + y + y + y + y + y = x\bigr)$', where $\operatorname{bin} n$ denotes n's binary representation.

The method of 'defining regular relations in terms of other regular relations' is used in various proofs. The use of the logical formalism is never strictly necessary. Each time it could be replaced by a direct argument. Union, intersection, and complementation can 'simulate' logical disjunctions, conjunctions, and negations in formulæ. Quantifiers can be simulated by clever constructions of alternating automata. However, already for formulæ involving just one quantifier alternation, as in the proof of theorem 4.16, 'logic-free' arguments quickly become hopelessly involved.

The whole treatment of first-order formulæ was born out of a desire to simplify the presentation of the proofs of this dissertation. However, it has now become an integral part of the generic proof approach: in order to apply the generic theorems to finite automata, the closure of the class of regular relations under elementary definitions is vitally needed.

There is no reason to believe that the method cannot also be applied to other problems. One example is given at the end of the next chapter, where it is used to show that every tournament on words with a regular edge relation can be 'linearised' using a finite automaton. The corresponding statement for polynomial-time computations is still an open problem, see Hemaspaandra et al. (2001) for an informed guess.

The Branch Diagonalisation Method

I introduce a new diagonalisation technique that I call 'branch diagonalisation'. Except for one proof in a technical report by Kummer and Stephan (1991), where a similar but weaker argument is used, branch diagonalisation seems to be a new concept. The method is not universally applicable, but where it is, it yields extremely strong separations. Branch diagonalisation, like other diagonalisation methods, is a *method*, not a particular theorem.

Just as in any other diagonalisation we construct a language L by systematically tricking every machine that could witness that L has a certain property. For each machine M we make an appropriate *diagonalisation decision*, which means that we define the characteristic string of some words in such a way that M does witness that L has the property. The words for which we trick the machines are called *diagonalisation points.* By choosing the diagonalisation points appropriately, we can ensure that L still has certain desirable properties. Diagonalisation methods differ mainly in how the diagonalisation points are chosen.

Most diagonalisation methods follow a variant of a simple rule for choosing diagonalisation points: trick the ith machine on its own binary encoding. Advanced methods from recursion theory, namely finite and infinite injury arguments, also use this rule, but they reassign diagonalisation points if this turns out to be necessary. An advanced method from complexity theory, the super-sparse set technique, does not interpret words themselves as codes of machines, but rather the iterated logarithm of their length.

Branch diagonalisation uses a different rule for choosing diagonalisation points: trick each machine on the binary code of the string of all previous diagonalisation decisions. This rule ensures that every diagonalisation point encodes the whole previous diagonalisation process. Given two words that encode two different diagonalisation sequences, even a finite automaton can compute up to which point the diagonalisation sequences agree and it can compute what ‘happened’ when the sequences split. In certain situations this information suffices for showing that the diagonalisation language has a certain property, like being (m, n)-finite-automata-verbose.

To give a flavour of the strength of branch diagonalisation when compared to standard diagonalisation techniques, consider the classes $\mathrm{V_p}(m, n)$ and $\mathrm{V_p^X}(m, n)$. The first class contains all polynomial-time (m, n)-verbose languages, that is, all languages whose n-fold characteristic function is m-enumerable by a polynomially time-bounded Turing machine. The second class is the relativisation of $\mathrm{V_p}(m, n)$ to the oracle X.

Using a super-sparse set diagonalisation, Richard Beigel (1990; 1991) has shown that, for fixed n, the classes $\mathrm{V_p}(m, n)$ form a proper hierarchy:

$$\mathrm{V_p}(1, n) \subsetneq \mathrm{V_p}(2, n) \subsetneq \cdots \subsetneq \mathrm{V_p}(2^n - 1, n) \subsetneq \mathrm{V_p}(2^n, n).$$

This hierarchy is not the end of the story. By ‘thinning out’ the diagonalisation language, one can show that for every *recursive* oracle X we also have $\mathrm{V_p}(m + 1, n) \not\subseteq \mathrm{V_p^X}(m, n)$ for $m < 2^n$. This is no longer true for all $m < 2^n$ if we take a nonrecursive language as oracle X. For example $\mathrm{V_p}(n, n) \subseteq \mathrm{V_p^K}(n - 1, n)$, where K is the halting problem, by Beigel’s nonspeedup theorem (Beigel, 1987).

For some m we have $\mathrm{V_p}(m + 1, n) \not\subseteq \mathrm{V_p^X}(m, n)$ for *all* oracles X, regardless of whether they are recursive or not. Using branch diagonalisation we can identify all such m. A non-trivial example is n, but other examples also exist. This shows that in the hierarchy some non-inclusions are ‘more robust’ than others. The robust ones can only be shown by branch diagonalisation, the less robust ones by standard diagonalisation techniques.

Branch diagonalisation is used to prove one of the core results that support my thesis: the inclusion structures of Turing machine and finite automata verboseness classes are identical. I do not know of a way of proving this result without the use of branch diagonalisation.

Just as for the definition of regular relations in terms of existing ones, there is no reason to believe that branch diagonalisation can only be applied to the study of verboseness. There is a lot of evidence that the method is applicable to other problems. For example, using branch diagonalisation, I show that the intersection of two P-selective languages need not be semirecursive. This is a stronger statement than what was previously known: Hemaspaandra and Jiang (1995) have shown that for every recursive time bound t, the intersection of two P-selective languages need not have a selector that can be computed in time t. Their proof cannot be extended to prove the stronger statement established in this dissertation, since their proof is based on super-sparse sets. The branch diagonalisation proof is also simpler.

Section 1.5

Organisation of this Dissertation

This dissertation consists of five main chapters, plus the present introductory chapter and a conclusion chapter. As far as possible, each of the main chapters is devoted to a single subject. The second chapter and most of the third chapter introduce the machinery needed for the formulation and the proofs of the main results. These are proved at the end of the third chapter, in the fourth chapter, and in the fifth chapter. The sixth chapter discusses applications in areas unrelated to enumerability.

Second Chapter: The Class of Regular Relations and Its Closure Properties

In the second chapter regular relations are studied. Different concepts are called 'regular relations' in the literature. Although these concepts are similar, there are subtle differences that cause different properties. My definition yields the same class as the one studied by Büchi (1962), although his definition is more complicated. My definition has the following advantages: it is easy to formulate and to use; the class of regular relations is a robust class that is closed under many operations; and functions whose graphs are regular can be computed in logarithmic space and in linear time.

Apart from presenting numerous examples of regular relations, the second chapter also includes a proof of the main closure property of the class of regular relations: they are closed under elementary definitions. Since first-order and monadic second-order logic are needed for the proof of this closure property and since first-order logic is used in all sorts of proofs later on, the second chapter includes a short review of first-order and second-order logic.

Third Chapter: Enumerability

In the third chapter enumerability classes are introduced. First, enumerability classes that have previously been studied in the literature are reviewed. Since one of the aims of this dissertation is to give unified proofs of the main results whenever possible, I introduce a generic form of enumerability. In essence, a function is *m-enumerable with respect to some class C* if there exists a relation in C that contains the function's graph and is m-bounded. By varying the class C, previously studied enumerability classes can be obtained. Generic theorems can be formulated that hold for all enumerability classes defined in terms of a class C that satisfies certain closure properties. The generic definition of enumerability is general enough to capture 'computational models' like Presburger arithmetic and even ordinal number arithmetic. At the end of the third chapter the first generic theorem is proved, namely the cross product theorem. It is proved there and not alongside the other generic theorems in the fourth chapter, because it is a 'pure' structural theorem on enumerability classes.

Fourth Chapter: Towards a Cardinality Theorem for Finite Automata

In the fourth chapter generic proofs of weak forms of the cardinality theorem are presented. The driving force behind the whole chapter is my desire to prove the following conjecture: if the n-fold cardinality function of a language A can be n-enumerated by a finite automaton, then A is regular. I do not know whether this conjecture is true, but the weak forms of the cardinality theorem support it.

First, I prove a generic generalised nonspeedup theorem. It can be instantiated both for finite automata and for Turing machines, which yields a unified proof of these results. Second, I prove a generic cardinality theorem for two input words. This theorem cannot be instantiated for Turing machines, since the class of recursively enumerable languages is not closed under complement. However, it can be instantiated for finite automata and also for Presburger arithmetic. Thus the cardinality theorem *is true* for finite automata at least for $n = 2$. Interestingly, in recursion theory the cardinality theorem had also first been shown for two words, before Kummer succeeded in proving it for all n. Third, I prove a generic restricted cardinality theorem, which can also be instantiated for finite automata.

Together, these three results bring us as near to a proof of the cardinality conjecture for finite automata as did the results in recursion theory before Kummer's breakthrough proof of the cardinality theorem.

Fifth Chapter: The Branch Diagonalisation Method

The fifth chapter is a 'tutorial' on the branch diagonalisation method. The method is crucially needed in the proof that the inclusion structures of Turing machine and finite automata verboseness classes coincide. The chapter starts with an example of a 'simple' branch diagonalisation. Then I extract the core ideas of the proof and construct a framework for branch diagonalisations. In the main section of the chapter, this framework is applied to verboseness classes. In the following sections, branch diagonalisation is used to construct a counterexample to a theorem of Kinber and it is applied to the study of reduction closures of selective languages.

Sixth Chapter:
Applications of Enumerability and Cardinality Computations

In the sixth chapter results on enumerability are applied to unrelated settings. These are: separability of languages and relations, protocol checking using finite automata, and classification with examples.

Seventh Chapter:
Conclusion

In the conclusion chapter the ideas, concepts, and results of the five main chapters are regrouped and analysed 'in hindsight'. A summary is given of which results hold for which computational models; an appraisal is attempted of the relevance of the ideas, concepts, and results; and possible future work is outlined.

Section 1.6

My Motivation

Much of the research presented in this dissertation was initiated a little over two years ago by a talk given by Ulrich Hertrampf at a complexity theory workshop in Ilmenau. His talk, which was entitled 'Über (m,n)-reguläre Sprachen', treated frequency computations by finite automata. In a frequency computation a finite automaton gets n input words and outputs a bitstring that agrees on at least m positions with the characteristic string of the words.

Here in Berlin, we were also studying frequency computations, but only as part of the more general theory of 'partial information algorithms' due to Arfst Nickelsen (2001). We had not thought of applying our theory to finite automata. During his talk, Ulrich conjectured that if a frequency computation is possible for a language for some $m \in \{1, \dots, n\}$, then the language must be regular. Arfst and I managed to produce a counterexample to this conjecture on the spot, namely appropriate standard left cuts. This raised the interesting (and still open) question of what the inclusion structure of finite automata frequency classes looks like.

Frequency classes are related to verboseness classes: a frequency computation for $m = 1$ is possible iff the language is $(2^n - 1, n)$-verbose.

Thus it was only natural to define finite automata verboseness classes. We knew that nonregular $(3, 2)$-fa-verbose languages exist and, obviously, every $(1, 2)$-fa-verbose language is regular. This left the case of $(2, 2)$-fa-verbose languages open. What could be said about them?

On the way back from Ilmenau, which I still remember vividly since it was raining madly and the autobahn was packed with cars, I pondered on this question. For Turing machines, Beigel's nonspeedup theorem stated that $(2, 2)$-verbose languages are recursive. For polynomial-time computations, it was known that $(2, 2)$-verbose languages need not be polynomial-time computable. The reason was, roughly spoken, that polynomially time-bounded machines 'run out of time' when trying to perform a certain search procedure used in the proof of the nonspeedup theorem. Surely finite automata, being even less powerful than polynomial-time machines, should 'fail even more' when trying to perform a search.

I fully believed that there exist arbitrarily complex $(2, 2)$-fa-verbose languages. It was very surprising to me when a week later or so I stumbled across an argument that seemed to show that $(2, 2)$-fa-verbose languages are decidable at least in polynomial space. Excitedly, I rushed to Arfst, whose office is next door, and sketched the proof. Since my proof sketches tend to be a little, well, sketchy at times, he looked at me a little doubtfully, but could not find any obvious fault in the reasoning. I returned to my office and continued to ponder on the problem. About half an hour later I rushed into Arfst's office once more, even more excitedly, and explained an improvement: all $(2, 2)$-fa-verbose languages can be decided in linear time. Once more Arfst listened patiently. I returned to my office once more, only to rush back yet again half an hour later, this time with a proof that all (n, n)-fa-verbose languages are regular.

Since then, my research on enumerability by finite automata has both been challenging and rewarding. Perhaps the most interesting spin-off of this research is the branch diagonalisation method, since, when applicable, it gives simple, elegant proofs of strong results. Although finite automata are such 'simple' devices and although finite automata have been studied for such a long time (at least compared to, say, polynomial-time computations), some questions that arose during my research seem to be as difficult as the corresponding problems for Turing machines. In particular, proving the cardinality conjecture for finite automata might well be as difficult as proving the cardinality theorem for Turing machines.

Despite all these exciting results, one may still ask whether it is worthwhile to study enumerability by finite automata. Some cynics claim that 'results in automata theory are either trivial or have been proved thirty years ago or both'. There are several reasons why the cynics are wrong.

First, the proofs in this dissertation are hardly trivial and there is good reason to believe that they *cannot* be proved in a much simpler way. For example, I shall demonstrate that there are 'difficult' regular languages (they can only be accepted by automata with a very large number of states) whose 2-fold cardinality function can be 2-enumerated by an automaton with only four states. This shows that *every* proof of the finite automata cardinality theorem for two words must turn 'simple' automata into 'arbitrarily complex' ones.

Second, some of the results obtained thirty years ago turn out to be wrong, as my counterexample to a claim of Kinber (1976) shows.

Third, the different applications in the sixth chapter show that results on finite automata enumerability are useful in other areas. They are arguably 'more useful' than the corresponding classical results from recursion theory: from a practical, and often also from a theoretical point of view, it is better to know that a language is regular than just knowing that it is recursive.

Fourth, the results obtained by branch diagonalisation are 'so strong' that even weak corollaries of these results are interesting in their own right. To give an example, from theorem 5.19 we can derive a condition on numbers n, m, h, and k such that for every oracle X there exists a polynomial-time (m,n)-verbose language that is not polynomial-time (h,k)-verbose, *not even relative to* X. The interesting aspect of this separation is that theorem 5.19 does not refer to polynomial-time computations at all. The theorem states a stronger separation that can be 'weakened' to this strong statement about polynomial-time computations. As another example, I show that there exist fa-selective languages whose intersection is not even semirecursive. An immediate corollary is the statement that the class of P-selective languages is not closed under intersection.

Second Chapter

The Class of Regular Relations and Its Closure Properties

Regular relations generalise regular languages. Instead of letting a finite automaton decide whether a *single* word is an element of a certain language, the automaton must decide whether a word *tuple* is an element of a certain relation. The different components of the word tuple are put on different, synchronously read input tapes. All heads on the input tapes advance exactly one symbol in each step. Equivalently, one may think of the words being fed to a single tape automaton in the following way: the first symbol on the input tape encodes the first symbols of the words, the second symbol encodes the second symbols, and so on. Shorter words are padded with blanks such that all words have the same length. Examples of regular relations are the lexicographical, standard, and prefix orderings of words, the graph of the addition function, and even the next-move relation on configuration graphs of fixed Turing machines. 'Difficult' relations, like the graph of the multiplication function, are not regular.

This particular definition of 'regular relation' yields the same class of relations as a definition of 'regular relation' due to Büchi (1962), although Büchi's definition is more complicated. In the literature, other classes of relations are also called 'regular' or 'rational'. Rabin and Scott (1959) consider two-tape automata where the heads may move at different speeds. Perrin (1990) considers automata with output and defines 'rational relations' as the graphs of the output functions of these automata.

Regular relations as defined here are useful both for the definition of more advanced concepts and as mere tools in proofs. Their usefulness stems from the fact that the class of regular relations inherits a great robustness from the class of regular languages. Nondeterminism is as powerful as determinism. It does not matter whether we read the words from left to right or from right to left. The class is closed under union, intersection, and complement. These latter closure properties are just special cases of the closure under elementary definitions. This means the following: Suppose a relational logical structure is given whose universe is Σ^* for some alphabet Σ and whose relations are all regular. Furthermore, a first-order formula over this structure is given that has certain free variables $u_1, \dots, u_n$. Then the set of all word tuples $(w_1, \dots, w_n)$ that make this formula true is regular.

The importance of this closure property for the proofs in later chapters is immense. All proofs of core results, including the finite automata versions of the generalised nonspeedup theorem, the cardinality theorem for two words, and the restricted cardinality theorem, make heavy use of the closure property. My earlier proofs of these theorems in (Tantau, 2002a,b), which use direct arguments, required a much more verbose reasoning. Basing proofs on closure properties of the class of regular relations allows an easy transfer to other computational models that share these closure properties. This gives deep insights into the structural similarities of Turing machine and finite automata enumerability classes.

In section 2.1 the notation and terminology are fixed for the different kinds of finite automata that will be used. I have not included an introduction to automata theory in general, which can be found in standard textbooks like the book of Hopcroft and Ullman (1979).

In section 2.2 regular relations are defined. The definition is accompanied by numerous examples. Most intuitively 'simple' relations are regular. Of course, no formal definition can faithfully capture a non-precise concept like 'simple' or 'efficient' as is well-discussed for the class P as representing the 'efficiently solvable' problems. Some intuitively simple relations, like the relation that relates a word and its reversal or the relation that relates a word and the doubled word, are not regular. A remedy would be to consider the class of relations that are accepted by multitape automata that read the tapes asynchronously, that is, considering the automata studied by Rabin and Scott. Unfortunately, this class lacks the crucial closure property of the class of regular relations: its closure under first-order definitions yields the complete arithmetical hierarchy since concatenation can be defined in this model. As demonstrated in the fifth chapter, my definition of regular relations is still general enough to allow their use in diagonalisation proofs.

In section 2.3 notations and terminology of first-order logic, second-order logic, and elementary definitions are reviewed. Two new theorems are presented that demonstrate the power of elementary definitions: the transitive closures of finite tournaments are elementarily definable; and the same is true for linearisations of tournaments on well-ordered sets.

In section 2.4 logical characterisations of the class of regular relations are formulated and proved. A corollary of the first-order characterisation is the main closure property of this class: every elementary definition in a regular structure defines a regular relation. At the end of the section, first applications are presented. One of them is a rephrasing of the closure property in the terminology of database theory: the class of regular structures is closed under first-order queries.

Section 2.1

Review of Finite Automata

This section fixes the notation and terminology for three kinds of finite automata: deterministic automata that recognise languages (Rabin–Scott recognisers), deterministic automata that have an output attached to each state (Moore automata), and nondeterministic automata. The only non-standard notion, introduced alongside Moore-type output, is what I call *final output*, see definition 2.4.

2.1 DEFINITION OF DETERMINISTIC FINITE AUTOMATA
A *deterministic finite automaton (DFA)* consists of an *input alphabet* Σ, a finite set Q of *states*, an *initial state* $q_{\text{initial}} \in Q$, a *transition function* $\delta\colon Q \times \Sigma \to Q$, and a set $F \subseteq Q$ of *accepting states.*

Deterministic finite automata are also called Rabin–Scott automata. Figure 2-1 on page 36 shows an example of how DFA's shall be depicted. Just like deterministic Turing machines, DFA's will be denoted by the letter M.

2.2 DEFINITION OF THE BEHAVIOUR OF DFA'S
The *extended transition function* $\hat{\delta}\colon Q \times \Sigma^* \to Q$ of a DFA is defined recursively as follows: let $\hat{\delta}(q, \epsilon) := q$; and for $\sigma \in \Sigma$ and $w \in \Sigma^*$ let $\hat{\delta}(q, \sigma w) := \hat{\delta}\bigl(\delta(q, \sigma), w\bigr)$. A DFA M *accepts* the language $\mathrm{L}(M) := \{w \in \Sigma^* \mid \hat{\delta}(q_{\text{initial}}, w) \in F\}$. A language is *regular* if it is accepted by a DFA.

2.3 DEFINITION OF DFA'S WITH (MOORE-TYPE) OUTPUT
A *DFA with output* consists of an *input alphabet* Σ, an *output alphabet* Γ, a finite set Q of *states*, an *initial state* $q_{\text{initial}} \in Q$, a *transition function* $\delta\colon Q \times \Sigma \to Q$, and an *output function* $\gamma\colon Q \to \Gamma$.

The output function γ generalises the usual set $F \subseteq Q$ of accepting states. When depicting DFA's with output, the output attached to a state is shown in the lower part of the state's circle, see figure 3-1 on page 64 for an example.

In the next definition, $w\,[i_1, \dots, i_k]$ denotes the symbols of the word w at positions i_1, ..., i_k. For example $abc\,[2, 2, 1] = bba$.

2.4 DEFINITION OF FINAL- AND MOORE-OUTPUT
Let M be a DFA with output and w an input word. The *final output* $M(w)$ of M is the value $\gamma\bigl(\hat{\delta}(q_{\text{initial}}, w)\bigr)$, that is, the output attached to the last state reached by M on input w. The *Moore output* of M on input w is the string $M\bigl(w\,[1]\bigr)\,M\bigl(w\,[1, 2]\bigr)\,M\bigl(w\,[1, 2, 3]\bigr)\cdots M(w)$.

2.5 DEFINITION OF NONDETERMINISTIC FINITE AUTOMATA
A *nondeterministic finite automaton with ϵ-moves (NFA with ϵ-moves)* consists of an *input alphabet* Σ, a finite set Q of *states*, a set $Q_{\text{initial}} \subseteq Q$ of *initial states*, a *transition relation* $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$, and a set $F \subseteq Q$ of *accepting states.* If the transition relation is a subset of $Q \times \Sigma \times Q$, the automaton is called an *NFA without ϵ-moves.*

Nondeterministic finite automata will be denoted by the letter N.

2.6 Definition of the Behaviour of nfa's
For every NFA $N = (\Sigma, Q, Q_{\text{initial}}, \Delta, F)$ with ϵ-moves, the pair (Q, Δ) forms a directed graph whose edges are labelled with symbols from Σ or, in case of the 'label' ϵ, unlabelled. The set of all strings of labels on paths that start in Q_{initial} and end in F is called the *language accepted by* N, written $\mathrm{L}(N)$.

2.7 Fact (Rabin and Scott, 1959)
For every NFA *N with ϵ-moves there exists a* DFA *M with* $\mathrm{L}(N) = \mathrm{L}(M)$.

Section 2.2

Definition of Regular Relations

Theorists usually ignore the difference between a set of words and a set of word tuples. Tuples can easily be converted to words and vice versa. You will only rarely find an explicit definition of, say, polynomial-time decidable relations, because it is assumed that you will infer their definitions from the definition of polynomial-time decidable languages and from the definition of your pet tupling function.

For the definition of regular relations more care has to be taken. Different reasonable tupling functions yield different notions of regular relations. For example, just writing the words alongside, separated by marker symbols, yields a boring class of regular relations. I propose to use the following tupling method: Given a word tuple $(w_1, \dots, w_n)$, image each word to be written on a new line on a sheet of squared paper. The first symbols of all words are now on top of each other, just like the second symbols, and so on. These 'columns of symbols' will be the individual symbols of the coded tuple $\langle w_1, \dots, w_n \rangle$, which is a word over the alphabet $\Sigma^{\langle n \rangle}$ of all possible columns of symbols. Since the words may have different lengths, some columns may contain blank squares. For example $\langle 123, 31 \rangle = \binom{1}{3}\binom{2}{1}\binom{3}{\square}$. This tupling method will be used throughout this dissertation since it can be used for finite automata, for polynomial-time computations, and for recursive computations.

The class of regular relations defined in terms of this tupling function has a number of useful properties:

1. It includes numerous relations that are intuitively 'simple', like the lexicographical ordering or the standard ordering of words.
2. It can be defined equivalently in terms of multi-tape automata with synchronously moving heads. These automata model finite memory online devices that are fed multiple data streams.
3. It enjoys strong closure properties.
4. It is large enough to be useful in diagonalisation proofs.

The class has previously been studied by Büchi (1962) and he also called its members 'regular relations'. However, his more complicated definition is based on monadic second-order logic. I believe that my definition is more intuitive and more easily applicable than Büchi's.

The multi-tape automata model that can be used for an alternative definition of regular relations has previously been studied by Kinber (1976) and recently by Austinat et al. (2000, 2003). It is defined as follows: An *n-tape automaton* has n input tapes instead of the usual single input tape. There is a head for each tape, but the heads must move *synchronously*. This means that in each step all heads advance exactly one symbol—no head may lag behind or dash ahead or turn around. One can think of the heads as a slot that moves over the tapes through which the automaton 'sees' exactly one symbol of each tape at a time. At the beginning of the computation each tape is initialised with one component of a word tuple $(w_1, \ldots, w_n)$. If the words have different lengths, shorter words are padded with blanks. At the end of the computation, when all heads hit the ends of the tapes, the automaton can accept or reject the tuple, depending on whether it has reached an accepting state or not. (More generally, a multi-tape automaton with output could now produce some final output.) Clearly, a relation is regular iff it is accepted by an appropriate multi-tape automaton.

In the following, formal definitions of the tupling function, of regular relations, and of regular functions (functions with a regular graph) are given. The definitions are accompanied by numerous examples. The closure properties of the class of regular relations are proved in section 2.4. Their usefulness for diagonalisation arguments is demonstrated in the fifth chapter.

2.8 Definition of Tupling Alphabets

Let Σ be an alphabet and let $\square \notin \Sigma$ be a special blank symbol. The *n-fold tupling alphabet* $\Sigma^{\langle n \rangle}$ is the following set of column vectors:

$$\Sigma^{\langle n \rangle} := \bigl(\Sigma \cup \{\square\}\bigr)^n \setminus \left\{ \begin{pmatrix} \square \\ \vdots \\ \square \end{pmatrix} \right\}.$$

For example, $\{0,1\}^{\langle 2 \rangle} = \bigl\{ \binom{0}{0}, \binom{0}{1}, \binom{1}{0}, \binom{1}{1}, \binom{\square}{0}, \binom{\square}{1}, \binom{0}{\square}, \binom{1}{\square} \bigr\}$.

2.9 Definition of Coded Tuples

Given words $w_1, \ldots, w_n \in \Sigma^*$, the *coded tuple* $\langle w_1, \ldots, w_n \rangle \in \bigl(\Sigma^{\langle n \rangle}\bigr)^*$ is the sequence of columns of a matrix with n rows and ℓ columns, where ℓ is the length of the longest w_i. The first row of the matrix contains the symbols of w_1, the second row contains the symbols of w_2, and so on, up to the nth row. Rows that are not completely filled by a word are filled up with $\square$.

2.10 Definition of Regular Relations
A relation $R \subseteq (\Sigma^*)^n$ is *regular* if the language

$$\bigl\{\langle w_1, \dots, w_n\rangle \bigm| (w_1, \dots, w_n) \in R\bigr\}$$

over the alphabet $\Sigma^{\langle n\rangle}$ is regular.

For a relation $R \subseteq U_1 \times \cdots \times U_n$ let us call $U_1 \cup \cdots \cup U_n$ the *universe of* R. By the above definition, the universe of a regular relation is always of the form Σ^* for some alphabet Σ. However, the question of whether a relation R is regular is also meaningful if the relation is a subset of $\Sigma_1^* \times \cdots \times \Sigma_n^*$ for different alphabets Σ_i. In this case, let us implicitly unite all the alphabets Σ_i, forming a big alphabet $\Sigma := \Sigma_1 \cup \cdots \cup \Sigma_n$, and consider R to be a subset of $(\Sigma^*)^n$. In other words, let us implicitly enlarge the universe of R from $\Sigma_1^* \cup \cdots \cup \Sigma_n^*$ to $(\Sigma_1 \cup \cdots \cup \Sigma_n)^*$.

2.11 Example: Lexicographical, Standard, and Prefix Orderings
For every alphabet Σ, the lexicographical ordering $\leq_{\text{lex}}$ of Σ^* (also known as dictionary ordering), the standard ordering $\leq_{\text{std}}$ (first lengthwise, then lexicographically), and the prefix ordering $\sqsubseteq$ are all regular. Figure 2-1 depicts an automaton that witnesses, for $\Sigma = \{0, 1\}$, that $\leq_{\text{lex}}$ is regular.

2.12 Example: Graph of a Final Output Function
Recall that the final output $M(w)$ of a DFA M is the output $\gamma\bigl(\hat\delta(q_{\text{initial}}, w)\bigr)$ attached to the last state reached by M on input w. For every DFA M the graph $\bigl\{\bigl(w, M(w)\bigr) \in \Sigma^* \times \Gamma \mid w \in \Sigma^*\bigr\} \subseteq \Sigma^* \times \Gamma^*$ of M's final output function is regular.

To see this, note that the second component of the first symbol of $\bigl\langle w, M(w)\bigr\rangle$ is $M(w)$. An automaton can accept the graph by first reading the first symbol, storing the second component in its state, and by then simulating M on the first components of the following symbols. Once the input has been completely read, the automaton accepts if $M(w)$ equals the stored value.

2.13 Example: Graph of a Moore Output Function
For every DFA M with output, the relation

$$\Bigl\{\bigl(w, M(w\texttt{[}1\texttt{]})\,M(w\texttt{[}1,2\texttt{]})\cdots M(w)\bigr) \in \Sigma^* \times \Gamma^* \Bigm| w \in \Sigma^*\Bigr\}$$

is regular. This relation is the graph of M's Moore output function. This shows that the relations called 'rational' by Perrin (1990) are regular in the sense of our definition.

In the next example, $\operatorname{bin} n$ denotes the canonical encoding of the natural number n as a bitstring. For example $\operatorname{bin} 6 = 110$. The notation $\operatorname{bin} X$ is similarly used to denote canonical binary encodings of other objects X, like graphs or machines. The reverse of a word w is denoted w^{reversed}.

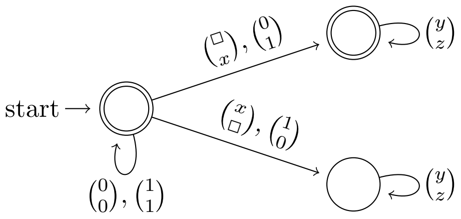

start
$\binom{\square}{x}, \binom{0}{1}$
$\binom{y}{z}$
$\binom{x}{\square}, \binom{1}{0}$
$\binom{0}{0}, \binom{1}{1}$
$\binom{y}{z}$

FIGURE 2-1
Given two words $u, v \in \{0,1\}^*$, the DFA accepts the coded word $\langle u, v\rangle$ iff $u \leq_{\text{lex}} v$. The variables $x \in \{0,1\}$ and $\binom{y}{z} \in \{0,1\}^{\langle 2\rangle}$ represent arbitrary values. Double circles around states indicate accepting states.

2.14 Example: Addition

The addition relation

$$\big\{\big((\operatorname{bin} m)^{\text{reversed}}, (\operatorname{bin} n)^{\text{reversed}}, \operatorname{bin}(m+n)^{\text{ reversed}}\big) \mid m, n \in \mathbb{N}\big\}$$

is regular, because it is the graph of a Moore output function.

2.15 Example: Next Configuration Relation

For a Turing machine M with state set Q, a disjoint tape alphabet Γ, and a single semi-infinite tape, let us code configurations (which are sometimes also called ‘instantaneous descriptions’) as follows: if $u \in \Gamma^*$ is the word before the head, $v \in \Gamma^*$ is the word starting at the head, and $q \in Q$ is the current state, we encode the configuration as $uqv \in (\Gamma \cup Q)^*$.

With this coding, M’s next-move relation on configurations is regular. Note that this is also true for other reasonable codings of configurations.

2.16 Counterexample: Nonregular Relations

The relation $\big\{(w, ww) \mid w \in \{0\}^*\big\}$ is not regular. Setting $a := \binom{0}{0}$ and $b := \binom{\square}{0}$, if it were regular, so would be the language $\{a^n b^n \mid n \in \mathbb{N}\}$. The relation $\big\{(w, w^{\text{reversed}}) \mid w \in \{0,1\}^*\big\}$ is not regular. If it were regular, so would be its intersection with the regular relation $\big\{(0^n 1 0^m, 0^n 1 0^m) \mid m, n \in \mathbb{N}\big\}$. However, $\big\{(0^n 1 0^n, 0^n 1 0^n) \mid n \in \mathbb{N}\big\}$ is not regular.

Since the most important special cases of relations are functions, it is only fitting to introduce a name for relations that are both regular and functions.

2.17 Definition of Regular Functions

A function $f\colon \Sigma^* \to \Delta^*$ is *regular* if its graph $\big\{\big(w, f(w)\big) \subseteq \Sigma^* \times \Delta^* \mid w \in \Sigma^*\big\}$ is regular.

By examples 2.12 and 2.13, the final and Moore output functions of DFA’s with output are regular functions.

A function need not be efficiently computable just because its graph is easily decidable. For example, the graph of the function that maps each word w to $2^{2^{|w|}}$ many 0’s is decidable in logarithmic space, but the function itself is not even computable in exponential time. Fortunately, regular functions have reasonably low complexity. They can be computed in logarithmic space and also, alternatively, in linear time, see theorem 2.41.

Section 2.3

Review of First-Order Logic and Second-Order Logic

In this section the notation and terminology for the syntax and semantics of first-order and second-order logic are fixed. The exposition follows the book of Ebbinghaus and Flum (1995). At the end of the section I present two new examples of powerful elementary definitions that have interesting applications, see (Nickelsen and Tantau, 2002) and (Hemaspaandra et al., 2001). Although these examples are not needed for proofs in other parts, they give a first flavour of the arguments that will be used later on.

2.18 Definition of Logical Signatures
A *logical signature* consists of a set of *symbols* and an *arity function.* Three types of symbols exist, namely *constant symbols*, *function symbols*, and *relation symbols.* The arity function assigns a positive integer to each function symbol and each relation symbol.

Typical logical signatures are finite or at least countable, but this is not required. As is customary, when writing down signatures, the arity of a symbol is written as a superscript. Constant symbols and function symbols are written in lower case, relation symbols in upper case. For special symbols it will be clear from context whether they are function symbols or relation symbols. For example, the logical signature $\tau = \{E^2, s, t\}$ consists of a binary relation symbol E and two constant symbols s and t. The logical signature $\tau' = \{0, <^2, +^2\}$ consists of a constant symbol 0, a binary relation symbol $<$, and a binary function symbol $+$.

2.19 Definition of Logical Structures
Given a logical signature τ, a *τ-structure* $\mathcal{S}$ consists of a nonempty set U, called the *universe* of $\mathcal{S}$, of an element $c^{\mathcal{S}} \in U$ for every constant symbol $c \in \tau$, of a function $f^{\mathcal{S}} \colon U^n \to U$ for every function symbol $f \in \tau$ of arity n, and of a relation $R^{\mathcal{S}} \subseteq U^n$ for every relation symbol $R \in \tau$ of arity n.

Logical structures in which all relations are regular will be particularly important.

2.20 Definition of Regular Structures
A logical structure $\mathcal{S}$ is *regular* if its universe is Σ^* for some alphabet Σ and if all its relations $R^{\mathcal{S}}$ and all its functions $f^{\mathcal{S}}$ are regular.

Examples of regular $\{R^2\}$-structures are $\bigl(\{0,1\}^*, \leq_{\text{lex}}\bigr)$ and $\bigl(\{a,b,c\}^*, \sqsubseteq\bigr)$.

2.21 Definition of Terms
Given a logical signature τ, the set of *τ-terms* is defined inductively as follows: every first-order variable, drawn from the fixed set $\{\mathrm{v}_1, \mathrm{v}_2, \mathrm{v}_3, \dots\}$, is a term; every constant symbol in τ is a term; and if $t_1, \dots, t_n$ are terms and $f \in \tau$ is a function symbol of arity n, then $f(t_1, \dots, t_n)$ is a term.

As is customary, to make terms more readable, the infix notation is used for special symbols. Thus $0 + \mathrm{v}_2$ should be understood as $+(0, \mathrm{v}_2)$. The variables v_i are 'formal' or 'syntactic' variables. In order to avoid too many subscripts, lower case letters set in italics, like 'x', are used as variables for formal variables. For example, if x represents the formal variable v_2, the term $f(x, \mathrm{v}_3)$ represents the term $f(\mathrm{v}_2, \mathrm{v}_3)$.

2.22 Definition of (Positive) First-Order formulæ
Given a logical signature τ, the set of *first-order τ-formulæ* is defined inductively as follows: if s and t are τ-terms, then $s = t$ is a first-order τ-formula; if $t_1, \ldots, t_n$ are τ-terms and $R \in \tau$ is a relation symbol of arity n, then $R(t_1, \ldots, t_n)$ is a first-order τ-formula; if ϕ is a first-order τ-formula, then so is $\neg\phi$; if ϕ and ψ are first-order τ-formulæ, then so are $(\phi \vee \psi)$ and $(\phi \wedge \psi)$; and if ϕ is a first-order τ-formula and v_i is a first-order variable, then $\exists \mathrm{v}_i\, \phi$ is a first-order τ-formula. A first-order formula is *positive* if it does not contain a negation.

The set of free variables in a formula is defined in the usual way. As is customary, in order to make formulæ more readable, I use the abbreviations $\rightarrow$, $\leftrightarrow$, and $\forall$. I use the 'dot notation' to denote bindings as in $\phi \rightarrow . \psi \wedge \rho$ for $\phi \rightarrow (\psi \wedge \rho)$. The exact meaning of the dot is: 'insert an opening bracket here and a closing bracket after the longest well-formed formula following the dot'. Parentheses are omitted if there is no risk of confusion.

2.23 Definition of (Monadic) Second-Order formulæ
Given a logical signature τ, the set of *second-order τ-formulæ* is defined inductively in the same way as first-order τ-formulæ, with two additions: if $t_1, \ldots, t_n$ are terms, then $\mathrm{V}_i^n(t_1, \ldots, t_n)$ is a second-order τ-formula for each $i \in \{1, 2, 3, \ldots\}$; and if ϕ is a second-order τ-formula, so is $\exists \mathrm{V}_i^n\, \phi$. The variables V_i^n are called *second-order variables.* A second-order τ-formula is *monadic* if it does not contain any occurrence of a variable V_i^n with $n \geq 2$.

As for first-order variables, capital letters set in italics, like 'X', are used as more readable substitutes for the variables V_i^n.

2.24 Definition of Assignments
Given a structure $\mathcal{S}$, a *first-order $\mathcal{S}$-assignment* is a function α that assigns an element $\alpha(\mathrm{v}_i) \in U$ to every first-order variable v_i. A *second-order $\mathcal{S}$-assignment* also assigns an n-ary relation $\alpha(\mathrm{V}_i^n) \subseteq U^n$ to each second-order variable V_i^n with $n, i \in \{1, 2, 3, \ldots\}$.

Given a τ-structure $\mathcal{S}$, an assignment α, and a τ-formula ϕ, the *modelling relation* $\mathcal{S} \models \phi[\alpha]$ is defined in the usual way for first-order and second-order formulæ. A *model* of a first-order or second-order formula ϕ is a structure $\mathcal{S}$ such that $\mathcal{S} \models \phi[\alpha]$ for every assignment α. For pairwise different first-order variables $u_1, \ldots, u_n$, let $\mathcal{S} \models \phi[u_1 = x_1, \ldots, u_n = x_n]$ denote that $\mathcal{S} \models \phi[\alpha]$ holds for every assignment α with $\alpha(u_1) = x_1, \ldots, \alpha(u_n) = x_n$. Let $\mathcal{S} \models \phi$ denote that $\mathcal{S} \models \phi[\alpha]$ holds for every assignment α.

2.25 Definition of Elementary Definitions
Given a τ-structure $\mathcal{S}$, a first-order τ-formula ϕ, and pairwise different variables u_1, ... , u_n, we define a relation $\phi^{\mathcal{S}}(u_1, \dots, u_n) \subseteq U^n$ by

$$\phi^{\mathcal{S}}(u_1, \dots, u_n) := \bigl\{(x_1, \dots, x_n) \in U^n \mid \\ \mathcal{S} \models \phi[u_1 = x_1, \dots, u_n = x_n]\bigr\}.$$

The relation $\phi^{\mathcal{S}}(u_1, \dots, u_n)$ is called *elementarily definable in $\mathcal{S}$*. If ϕ is a positive formula, the relation is called *positively elementarily definable in $\mathcal{S}$*. A function is *elementarily definable in $\mathcal{S}$* if its graph is. An element is *elementarily definable in $\mathcal{S}$* if its singleton set is.

Elementary definitions will be our prime vehicle for defining new relations out of existing ones. To give an example, consider an $\{R^2\}$-structure $\mathcal{S}$ and the formula $\phi := \exists y\, R(x, y)$. The relation $\phi^{\mathcal{S}}(x)$ is the domain of the relation $R^{\mathcal{S}}$. It will be convenient to have a handy notation for this situation. Instead of having to write sentences like 'the domain of a relation R is the set $\phi^{\mathcal{S}}(x)$, where $\phi = \exists y\, R(x, y)$ and where $\mathcal{S}$ is a structure over the signature $\{R^2\}$ in which $R^{\mathcal{S}} = R$', we can use the abbreviation 'the domain D of a relation R is defined by $x \in D :\iff \exists y\, R(x, y)$'. A slightly more complex example is the elementary definition $(x, y) \in R :\iff \exists z \mathbin{.} P(x, z) \land Q(z, y)$. It shows that $R = Q \circ P$ can be defined elementarily. In the following, two new examples of elementary definitions are presented.

2.26 Example: Has Sir Galahad Beaten Sir Lancelot?
A group of knights has gathered to hold a tournament. It consists of a series of jousts between every two knights. In each joust exactly one knight wins. After the tournament, Sir Galahad and Sir Lancelot meet. Sir Galahad exclaims, 'Bravely met Sir Lancelot! It is just that thou hast beaten me in our joust. Thou art more skillfully than I am.' Sir Lancelot replies, 'Bravely met indeed, Sir Lancelot! But I am not so sure. Methinks thou hast beaten a knight who hath beaten a knight, and so on, who hath beaten me. Is that not true?'

In order to answer Sir Lancelot's question, we must solve the reachability problem for tournaments. Tournaments are directed graphs in which there is exactly one directed edge between any two different vertices and in which there are no self-loops. In (Tantau, 2001), see (Nickelsen and Tantau, 2002) for a generalisation, it is shown that there exists a first-order formula ϕ over the signature $\{E^2, s, t\}$ of graphs with two designated vertices such that for every finite vertex set V we have $(V, E, s, t) \models \phi$ iff the following two conditions hold:

1. (V, E) is a tournament and
2. there exists a path from s to t in this tournament.

By replacing s and t in ϕ by the free variables v_1 and v_2, we obtain a first-order formula that defines the transitive closures of finite tournaments elementarily.

2.27 Counterexample: Arbitrary Graphs and Large Tournaments
Using the compactness theorem, it can be shown that the transitive closures of *arbitrary* finite graphs cannot be defined elementarily, see (Ebbinghaus and Flum, 1995) for a detailed proof. Arfst Nickelsen and myself (2002) have shown that the transitive closure of arbitrary tournaments (finite and infinite ones) cannot be defined elementarily.

2.28 Example: Linearisations of Tournaments
A *linearisation* of a directed graph $G = (V, E)$ is a linear ordering E' of V such that whenever $(u, v) \in E'$ there is a path from u to v in G. A *well-ordering* is a linear ordering such that every subset has a smallest element. The following theorem shows that we can elementarily define a linearisation of every tournament, provided we have access to an arbitrary well-ordering of the universe. The proof of the theorem transfers an idea of Nickelsen, see Hemaspaandra et al. (2001), to a logical setting. Nickelsen has used the proof idea to show that every P-selective language has an associative selector in $\mathrm{FP}^{\mathrm{NP}}$. Note that the theorem holds for tournaments of arbitrarily large cardinality, not just for finite ones.

2.29 Theorem
There exists a first-order formula ϕ such that for every tournament (V, E) and every irreflexive well-ordering $<$ of V, the relation $\phi^{(V,E,<)}(\mathrm{v}_1, \mathrm{v}_2)$ is a linearisation of (V, E).

Proof. Let $\{E^2, <^2\}$ be the signature of graphs together with a binary relation. The formula ϕ is build up from several other formulæ that are defined as follows (they are explained below):

$$\begin{aligned}
\phi_{\text{via}}(x, y, z) &:= \big(E(x, z) \lor x = z\big) \land \big(E(z, y) \lor z = y\big), \\
\phi_{\text{conn}}(x, y, z) &:= \phi_{\text{via}}(x, y, z) \lor \phi_{\text{via}}(y, x, z), \\
\phi_{\text{minconn}}(x, y, z) &:= \phi_{\text{conn}}(x, y, z) \land \forall z' \centerdot z' < z \to \neg\phi_{\text{conn}}(x, y, z'), \\
\phi &:= \neg\, \mathrm{v}_1 = \mathrm{v}_2 \land \\
&\qquad \exists z \centerdot \phi_{\text{minconn}}(\mathrm{v}_1, \mathrm{v}_2, z) \land \phi_{\text{via}}(\mathrm{v}_1, \mathrm{v}_2, z).
\end{aligned}$$

Let a structure $\mathcal{S} = (V, E, <)$ be given such that (V, E) is a tournament and $<$ is a well-ordering of V. The formula ϕ_{via} expresses that we can 'go from x to y via z'. For $u, v \in V$, let us call $c \in V$ a *connector of u and v*, if $(u, v, c) \in \phi^{\mathcal{S}}_{\text{conn}}(x, y, z)$. Note that u and v themselves are always connectors of u and v. The smallest connector c of u and v with respect to the well-ordering $<$ is the unique c for which $(u, v, c) \in \phi^{\mathcal{S}}_{\text{minconn}}(x, y, z)$

holds. Thus in the formula ϕ the existential quantifier refers exactly to this particular c. This shows that for $u \neq v$ we have $(u, v) \in \phi^{\mathcal{S}}(\mathrm{v}_1, \mathrm{v}_2)$ iff for the smallest connector c of u and v we can go from u to v via c. Let us abbreviate $\phi^{\mathcal{S}}(\mathrm{v}_1, \mathrm{v}_2)$ by E'.

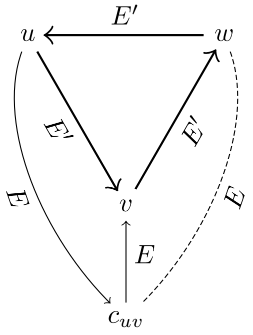
It remains to show that (V, E') is a linearisation of (V, E). First, $(u, v) \in E'$ implies that there is a path from u to v (via some c). Second, it is a linear ordering: It is clearly irreflexive. It is antisymmetric since for $u \neq v$, if we can go from u to v via c, we cannot also go from v to u via c. Nevertheless, we always have either $(u, v) \in E'$ or $(v, u) \in E'$. To prove transitivity, assume that we had a 'circle' in E', that is, $(u, v) \in E'$, $(v, w) \in E'$, and $(w, u) \in E'$ for distinct $u, v, w \in V$. Let c_{uv}, c_{vw}, and c_{wu} be the respective smallest connectors of the three pairs (u, v), (v, w), and (w, u). We may assume without loss of generality that c_{uv} is the smallest of these three connectors. This situation is depicted in figure 2-2. Then $(c_{uv}, w) \in E$ is impossible, since this would imply that c_{uv} is a connector of u and w no larger than c_{wu}, and thus $(u, w) \in E'$. But $(w, c_{uv}) \in E$ is also impossible, since this would imply that c_{uv} is a connector of w and v no larger than c_{vw}, and thus $(w, v) \in E'$. QED

Section 2.4

Logical Characterisations of the Class of Regular Relations

In this section we study how regular relations can be defined using formulæ. For regular languages, rather than relations, this study has a long tradition, dating back to the pioneering work of Richard Büchi (1960, 1962). As we shall see, the logical characterisations of regular languages also apply to regular relations. This is hardly surprising, since regular relations are defined in terms of regular languages. A corollary of the characterisations is the central closure property of the class of regular relations: every elementary definition in a regular structure defines a regular relation.

It is rather well-known that Büchi showed that all regular languages can be defined in terms of monadic second-order logic. It is less well-known that he also showed that regular languages can be defined in terms of first-order logic. Two different notions of 'definability' are involved here. Since most papers and textbooks deal with only one of these notions at a time, both are often just called 'definability'. The first is commonly used in descriptive complexity theory: by associating a structure with each word, the set of models of a formula defines a language. I call these definitions *model definitions*. The second notion, where the set of all words that make a certain formula true forms a language, is the classical logical concept of



$$\begin{tikzcd}
u \arrow[dr, "E'"] \arrow[ddr, bend right, "E"'] & & w \arrow[ll, "E'"'] \arrow[ddl, dashed, bend left, no head, "E"] \\
& v \arrow[ur, "E'"] & \\
& c_{uv} \arrow[u, "E"'] &
\end{tikzcd}$$

FIGURE 2-2
Impossible situation from the proof of theorem 2.29. If $u, v, w \in V$ were to form a circle with respect to the relation E' and if c_{uv} were the smallest of the connectors c_{uv}, c_{vw}, and c_{wu}, then both $(w, c_{uv}) \in E$ and $(c_{uv}, w) \in E$ would be impossible.

elementary definitions from the previous section. Note that elementary definitions implicitly always refer to first-order formulæ, whereas model definability can be applied to a large variety of logics.

Theorem 2.36 states that for each alphabet Σ that contains at least two letters there exists a logical structure $\mathcal{I}_\Sigma$ such that the following three statements are equivalent for relation on Σ^*:

1. It is regular.
2. It is model-definable in monadic second-order logic.
3. It is elementarily definable in $\mathcal{I}_\Sigma$.

Although Büchi has already given a proof of this theorem for a different structure, I present a partly new, complete proof for the above equivalences. I do so for three reasons.

First, showing a cyclic implication among the three statements gives a shorter proof than Büchi's, who shows the equivalence of the first two statements and then the equivalence of the last two statements.

Second, Büchi's universe is the set of natural numbers and operations on them are arithmetical operations. Words have to be coded as numbers in an awkward manner that gives rise to subtle problems like leading zeros and having to choose an appropriate base. In my proof, the universe consists of words, which are hence treated as 'first-order citizens' and for which these problems do not occur. A historically similar, though more difficult rephrasing of an arithmetical proof in terms of words is due to Quine (1946). He reproved Gödel's result that the first-order theory of arithmetic, that is, of the structure $(\mathbb{N}, +, \cdot)$, is undecidable by showing that the first-order theory of concatenation, that is, of the structure $\big(\{0,1\}^*, \circ\big)$, is undecidable.

Third, Büchi's original claim is wrong, since the structure $(\mathbb{N}, +, \mathrm{V}_2)$ that he uses does not do the trick. The monadic predicate V_2 is true for all powers of two. McNaughton (1963) corrected the mistake by keeping the addition function and replacing V_2 by a binary predicate e_2 on pairs (m, n) that is true if m is a power of two and the $(\log_2 m)$-th bit (from the right) of the binary representation of n is 1. For a recent study of arithmetical structures that can be used, especially for bases other than 2, see Bruyère et al. (1994) and Bruyère and Hansel (1997). The structure $\mathcal{I}_\Sigma$, which I propose to use instead, is defined as follows: its universe is Σ^*; and for each symbol $\sigma \in \Sigma$ it contains a binary relation I_σ that contains all word pairs (u, v) with $u\,[\![|v|]\!] = \sigma$, that is, where the $|v|$-th symbol of u is σ. For the binary alphabet it resembles McNaughton's structure, but misses the addition function and instead contains a dual of e_2 for positions having a 0.

Quite different logical characterisations of regular languages have also been studied. For example, much research has been devoted to axiomatic systems for regular languages based on equality formulæ, rather than on first-order formulæ. See (Salomaa, 1969) for an introduction.

In the following, the relevant definitions for the characterisations are presented first. Then the main characterisation theorem is proved and, as a corollary, we obtain the closure of the class of regular relations under elementary definitions. At the end of the section examples are given that employ the corollary. The two most important are: the class of regular structures is closed under first-order queries; and all regular functions are computable in logarithmic space and in linear time.

Terminology for the Monadic Second-Order Characterisation

For the characterisation of regular relations in terms of monadic second-order formulæ, a word is coded as a finite logical structure as follows: its universe is the set of the word's letter positions; and monadic relations indicate whether there is a certain letter at a given position or not.

2.30 Definition of Word Structures
For an alphabet Σ, let $\tau_\Sigma := \{Q^1_\sigma \mid \sigma \in \Sigma\} \cup \{<^2\}$. The *word structure* of a word $w \in \Sigma^+$ is the following τ_Σ-structure $\mathcal{W}_w$: its universe is the set $\{1, \dots, |w|\}$ of positions in w; the predicate $Q^{\mathcal{W}_w}_\sigma$ is true for a position i if $w[i] = \sigma$; and $<^{\mathcal{W}_w}$ is the standard ordering of $\{1, \dots, |w|\}$. The word structure of the empty word has the universe $\{1\}$ and all relations are empty.

In the literature, word structures are often misleadingly called 'word models'. The term is problematic since many word structures will *not* be models of a given formula.

The special rule for the empty word is necessary, because the universe of a structure may not be empty by definition. Note that the empty word structure is the only word structure that satisfies $\forall i \bigwedge_{\sigma\in\Sigma} \neg Q_\sigma(i)$. It is also the only structure that satisfies $\exists i \bigwedge_{\sigma\in\Sigma} \neg Q_\sigma(i)$. Only simple formulæ are needed to 'check' whether a given structure is the empty word structure.

2.31 Definition of Model-Definable Relations
Let Σ be an alphabet and let ϕ be a (first- or second-order) $\tau_{\Sigma^{\langle n\rangle}}$-formula. The formula *model-defines* the relation

$$\{(w_1, \dots, w_n) \in (\Sigma^*)^n \mid \mathcal{W}_{\langle w_1,\dots,w_n\rangle} \models \phi\}.$$

2.32 Example
Let $R \subseteq \{0,1\}^+ \times \{0,1\}^+$ be defined as follows: $(u, v) \in R$ if u and v have the same length and if u is the bitwise negation of v. This relation is

model-defined by the first-order formula

$$\forall i \centerdot Q_{\binom{0}{1}}(i) \vee Q_{\binom{1}{0}}(i).$$

In theorem 2.36 it is shown that exactly the regular relations are model-definable in monadic second-order logic.

Terminology for the First-Order Characterisation

For the characterisations of regular relations in terms of *first*-order formulæ we need a special structure, which I call the *index structure*. The only available operation is an indexing operation.

2.33 Definition of the Index Structure
The *index structure* $\mathcal{I}_\Sigma$ of an alphabet Σ is the following $\{I^2_\sigma \mid \sigma \in \Sigma\}$-structure: its universe is Σ^*; and $(u, v) \in I_\sigma^{\mathcal{I}_\Sigma}$ iff $1 \leq |v| \leq |u|$ and $u \llbracket |v| \rrbracket = \sigma$.

At first glance, only few relations seem to be definable elementarily in the index structure. Before reading on, I invite you to try to find a first-order formula ϕ such that $\phi^{\mathcal{I}_{\{0,1\}}}(v) = \{01\}$. The poverty of the structure is only superficial: theorem 2.36 states that all regular relations are elementarily definable in it. Before we prove this statement, let us have a look at some examples of easy relations that are definable elementarily in $\mathcal{I}_\Sigma$.

2.34 Example: How to Define the Empty Word
The empty word is the only word with the property that whatever index you try, there will not be a letter at that index. Thus the formula $\phi_\epsilon := \forall x \bigwedge_{\sigma\in\Sigma} \neg\, I_\sigma(v, x)$ has the property $\phi_\epsilon^{\mathcal{I}_\Sigma}(v) = \{\epsilon\}$.

2.35 Example: Lengthwise Preorderings
The formulæ $u \preceq v := \phi_\epsilon(u) \vee \bigvee_{\sigma\in\Sigma} I_\sigma(v, u)$ and $u \prec v := \neg\, v \preceq u$ express that u is not longer, respectively shorter, than v. Instead of $u \prec v$ I shall also write '$I_\square(u, v)$' since v is longer than u iff the '$|v|$-th symbol of u' is a 'blank'. Note, however, that $I_\square$ is not an element of $\mathcal{I}_\Sigma$'s signature and that $I_\square(u, v)$ is just an abbreviation for the lengthy formula $\neg\phi_\epsilon(v) \wedge \bigwedge_{\sigma\in\Sigma} \neg I_\sigma(u, v)$.

The Logical Characterisation Theorem

2.36 Logical Characterisation Theorem for Regular Relations
Let n be a positive integer and let Σ be an alphabet with $|\Sigma| \geq 2$. Then for every n-ary relation R on Σ^* the following statements are equivalent:

1. *R is regular.*
2. *R is model-definable by a monadic second-order $\tau_{\Sigma\langle n\rangle}$-formula.*
3. *R is elementarily definable in $\mathcal{I}_\Sigma$.*

Proof. The following proof establishes a cycling implication among the three statements. Let $\sigma_* \in \Sigma$ be a fixed symbol.

Statement 1 implies Statement 2.
Let R be regular via a DFA $M = (\Sigma, Q, q_{\text{initial}}, \delta, F)$. We must construct a monadic second-order formula ϕ that model-defines R. For simplicity, let us only consider the case where the coded input tuple is not the empty word. The following construction adapts the proof from the book of Straubing (1994) to relations.

On input of a coded tuple $t = \langle w_1, \dots, w_n \rangle$, the computation of M passes through a sequence $p_1, p_2, \dots, p_{|t|}, p_{|t|+1}$ of states with $p_1 = q_{\text{initial}}$. The computation is accepting if $p_{|t|+1} \in F$. Roughly spoken, the formula ϕ will say 'there exists a sequence of states that is the computation of M on input t and this sequence accepts'. To encode the sequence, for each state $q \in Q = \{q_1, \dots, q_{|Q|}\}$ let us introduce a special monadic second-order variable X_q. The idea is to ensure that the formula $X_q(i)$ is true for an index $i \in \{1, \dots, |t|\}$ iff $q = p_i$.

We first need a formula that expresses that the element p_i of the sequence is a specific state q: let $\psi_q(i) := X_q(i) \wedge \bigwedge_{q' \neq q} \neg X_{q'}(i)$.

The following formula expresses that the first state is q_{initial}:

$$\phi_{\text{start}} := \forall m \,.\, \underbrace{(\forall i \,\neg\, i < m)}_{m=1} \rightarrow \psi_{q_{\text{initial}}}(m).$$

Note that only the number 1 has the property $\forall i \; \neg\, i < 1$. Thus ϕ_{start} enforces $\psi_{q_{\text{initial}}}(1)$ and thus $p_1 = q_{\text{initial}}$.

In order to ensure $p_{i+1} = \delta\big(p_i, t\texttt{[}i\texttt{]}\big)$ we use the following formula:

$$\phi_{\text{middle}} := \bigwedge_{q \in Q} \; \bigwedge_{\sigma \in \Sigma^{\langle n \rangle}} \forall i \,.\, \underbrace{\big(\psi_q(i) \wedge Q_\sigma(i)\big)}_{q = p_i \text{ and } \sigma = t\texttt{[}i\texttt{]}}$$
$$\rightarrow \Big[\forall i' \,.\, \underbrace{\big(i < i' \wedge \forall j \,.\, j < i \rightarrow \neg\, j < i'\big)}_{i' = i+1} \rightarrow \underbrace{\psi_{\delta(q,\sigma)}(i')}_{p_{i+1} = \delta(p_i, t\texttt{[}i\texttt{]})}\Big].$$

Finally, let $\delta^{-1}(F) := \big\{(q,\sigma) \in Q \times \Sigma^{\langle n \rangle} \mid \delta(q,\sigma) \in F\big\}$ be the set all state-symbol pairs that lead into an accepting state. Let

$$\phi_{\text{end}} := \forall m \,.\, \underbrace{(\forall i \,\neg\, m < i)}_{m = |t|} \rightarrow \bigvee_{(q,\sigma) \in \delta^{-1}(F)} \big(\psi_q(m) \wedge Q_\sigma(m)\big).$$

With these definitions, the formula $\phi := \exists X_{q_1} \cdots \exists X_{q_{|Q|}} \,.\, \phi_{\text{start}} \wedge \phi_{\text{middle}} \wedge \phi_{\text{end}}$ will be true iff the computation of M on input $\langle w_1, \dots, w_n \rangle$ is accepting.

Statement 2 implies Statement 3.
Let ϕ be a monadic second-order formula that model-defines R. We must transform this formula into a first-order formula ϕ' in such a way that $(\phi')^{\mathcal{I}_\Sigma}(u_1,\dots,u_n) = R$ holds. Thus for any tuple $(w_1,\dots,w_n) \in (\Sigma^*)^n \setminus \{(\epsilon,\dots,\epsilon)\}$ we have to ensure

$$\mathcal{W}_{\langle w_1,\dots,w_n\rangle} \models \phi \quad \text{iff} \quad \mathcal{I}_\Sigma \models \phi'[u_1 = w_1,\dots,u_n = w_n].$$

The core idea of the proof is the following: a monadic relation on a universe $\{1,\dots,\ell\}$ can be encoded as a word whose ith letter is σ_* iff the relation is true for i. Monadic second-order quantifications $\exists V\,\psi$ in ϕ are replaced by first-order quantifications $\exists \ddot{v}\,\psi'$ in ϕ', where $\ddot{v}$ is a fresh first-order variable. The *double* dot is intended to remind us that $\ddot{v}$ represents a *second*-order variable in ϕ. This idea is due to Büchi, although he formulated it in his arithmetical setting.

To give an example of the translation process, assume $\Sigma = \{a,b,c\}$ and $\sigma_* = a$. For $\ell = 4$, a monadic relation on the universe $\{1,2,3,4\}$ is a subset like $\{1,4\}$. This relation would be replaced by a word w of length four such that $w[i] = a$ iff $i \in \{1,4\}$. Examples of such words are *abba* or *abca*.

Using words to encode the second-order variables of ϕ raises the problem of how to encode its first-order variables. Since such a variable is interpreted as a number between 1 and ℓ, we can use the *length* of a word to encode such a value. A first-order quantification $\exists v\,\psi$ is transformed to $\exists \dot{v}\,\psi'$, where the *single* dot indicates that the fresh variable $\dot{v}$ represents a *first*-order variable. Whenever $\dot{v}$ is used inside ψ, we use only its length.

For example, if the first-order variable x with $\alpha(x) = 3$ is represented by the word $\dot{x} = aac$, which has length three as desired, and if the second-order variable V with $\alpha(V) = \{1,4\}$ is represented by the word *abca*, in order to check whether $V(x)$ holds we can check whether the $|aac|$-th symbol of *abca* is an a. Recall that a second-order $\mathcal{W}_{\langle w_1,\dots,w_n\rangle}$-assignment α assigns a number $\alpha(v) \in \{1,\dots,\ell\}$ to every first-order variable v and a set $\alpha(V) \subseteq \{1,\dots,\ell\}$ to every monadic second-order variable V.

Let us call a second-order $\mathcal{W}_{\langle w_1,\dots,w_n\rangle}$-assignment α *compatible* with a first-order $\mathcal{I}_\Sigma$-assignment α' if the following holds:

1. for all first-order variables v, the length of $\alpha'(\dot{v})$ is exactly $\alpha(v)$,
2. for all second-order variables V, the set $\{i \mid \alpha'(\ddot{v})[i] = \sigma_*\}$ is exactly $\alpha(V)$, and
3. $\alpha'(u_j) = w_j$ for all input variables u_j.

Our aim is to give an inductive definition of the formula ϕ' such that for all compatible assignments α and α' the following equivalence holds:

$$\mathcal{W}_{\langle w_1,\dots,w_n\rangle} \models \phi[\alpha] \quad \text{iff} \quad \mathcal{I}_\Sigma \models \phi'[\alpha']. \qquad (*)$$

We start with the simplest kind of formulæ ϕ, namely $\phi = (x = y)$. Let $(x = y)' := \dot{x} \preceq \dot{y} \wedge \dot{y} \preceq \dot{x}$. For this definition $(*)$ holds, since $\alpha(x) = \alpha(y)$ holds iff the lengths of $\alpha'(\dot{x})$ and $\alpha'(\dot{y})$ are equal. Likewise $(x < y)' := \dot{x} \prec \dot{y}$ also ensures that $(*)$ holds.

For the application $V(x)$ of a second-order variable V to a first-order variable x let $\big(V(x)\big)' := I_{\sigma_*}(\ddot{v}, \dot{x})$. The formula $V(x)$ is true iff $\alpha(x) \in \alpha(V)$. Since $I_{\sigma_*}(\ddot{v}, \dot{x})$ holds by definition iff the $|\alpha'(\dot{x})|$-th letter of $\alpha'(\ddot{v})$ is σ_*, condition $(*)$ holds.

The indicator predicates Q_ν are transformed as follows:

$$\Big(Q_{\left(\begin{smallmatrix}\sigma_1\\ \vdots \\ \sigma_n\end{smallmatrix}\right)}(x)\Big)' := I_{\sigma_1}(u_1, \dot{x}) \wedge \cdots \wedge I_{\sigma_n}(u_n, \dot{x}).$$

Recall that $Q_\nu(x)$ tests whether the $\alpha(x)$-th symbol of $\langle w_1, \ldots, w_n\rangle$ is the vector $\nu \in \Sigma^{\langle n\rangle}$. By definition of I_{σ_i}, for $\sigma_i \in \Sigma$, the test $I_{\sigma_i}(u_i, \dot{x})$ is true iff the $|\alpha'(\dot{x})|$-th letter of w_i is σ_i. For $\sigma_i = \Box$, the test $I_\Box(u_i, \dot{x})$ is true iff w_i does not have an $|\alpha'(\dot{x})|$-th letter.

The definitions $(\neg\psi)' := \neg\psi'$, $(\psi \wedge \zeta)' := \psi' \wedge \zeta'$, and $(\psi \vee \zeta)' := \psi' \vee \zeta'$ ensure that $(*)$ is satisfied for negations, disjunctions, and conjunctions.

For quantifications over first-order variables let

$$(\exists v\, \psi)' := \exists \dot{v} \,.\, \psi' \wedge \neg\phi_\epsilon(\dot{v}) \wedge \textstyle\bigvee_{j=1}^{n} \dot{v} \preceq u_j.$$

The formula $\neg\phi_\epsilon(\dot{v})$ ensures that $\alpha'(\dot{v})$ has length at least 1. The big disjunction is true if $\alpha'(\dot{v})$ is not longer than the longest input word. Thus $|\alpha'(\dot{v})| \in \{1, \ldots, \ell\}$, where ℓ is the length of $\langle w_1, \ldots, w_n\rangle$. Once more, $(*)$ is satisfied.

Finally, quantifications over second-order variables are transformed by

$$(\exists V\, \psi)' := \exists \ddot{v} \,.\, \psi' \wedge \textstyle\bigvee_{j=1}^{n} \ddot{v} \preceq u_j.$$

The disjunction ensures that $\alpha'(\ddot{v})$ is not longer than the longest input word. Hence the set of positions where $\alpha'(\ddot{v})$ has the letter σ_* is contained in $\{1, \ldots, \ell\}$. On the other hand, since our alphabet has at least two letters, for every subset of $\{1, \ldots, \ell\}$ there exists a word $\alpha'(\ddot{v})$ such that this subset is exactly the set of positions where $\alpha'(\ddot{v})$ has the letter σ_*.

Putting it all together, $(*)$ holds for all formulæ ϕ. Since $\mathcal{W}_{\langle w_1,\ldots,w_n\rangle} \models \phi$ iff $(w_1, \ldots, w_n) \in R$, condition $(*)$ ensures that $(w_1, \ldots, w_n) \in R$ holds iff $\mathcal{I}_\Sigma \models \phi'[u_1 = w_1, \ldots, u_n = w_n]$.

Statement 3 implies Statement 1.
This final part of the proof is new. Unlike the previous implication, the following arguments also hold for unary alphabets. This is not relevant for the present proof, but will play a rôle in corollary 2.37.

We prove, by induction on the structure of ϕ, that $\phi^{\mathcal{I}_\Sigma}(u_1, \dots, u_n)$ is regular for every first-order τ-formula ϕ. After possible renaming we may assume $u_i = \mathrm{v}_i$.

Let us start with atomic formulæ ϕ. If ϕ is of the form '$\mathrm{v}_i = \mathrm{v}_j$', then the relation $(\mathrm{v}_i = \mathrm{v}_j)^{\mathcal{I}_\Sigma}(\mathrm{v}_1, \dots, \mathrm{v}_n) = \big\{(w_1, \dots, w_n) \in (\Sigma^*)^n \mid w_i = w_j\big\}$ is clearly regular. Likewise, if ϕ is of the form '$I_\sigma(\mathrm{v}_i, \mathrm{v}_j)$', the relation $\big(I_\sigma(\mathrm{v}_i, \mathrm{v}_j)\big)^{\mathcal{I}_\Sigma}(\mathrm{v}_1, \dots, \mathrm{v}_n) = \big\{(w_1, \dots, w_n) \in (\Sigma^*)^n \mid (w_i, w_j) \in I_\sigma^{\mathcal{I}_\Sigma}\big\}$ is regular.

For non-atomic ϕ, the only interesting case is $\phi = \exists \mathrm{v}_i\, \psi$. For simplicity, let us assume $i = n + 1$. We have to show that if $\psi^{\mathcal{I}_\Sigma}(\mathrm{v}_1, \dots, \mathrm{v}_{n+1})$ is regular, so is

$$\begin{aligned}\phi^{\mathcal{I}_\Sigma}&(\mathrm{v}_1, \dots, \mathrm{v}_n) = \\ &\big\{(w_1, \dots, w_n) \in (\Sigma^*)^n \mid \text{there exists } w_{n+1} \in \Sigma^* \text{ such that} \\ &\qquad\qquad\qquad (w_1, \dots, w_{n+1}) \in \psi^{\mathcal{I}_\Sigma}(\mathrm{v}_1, \dots, \mathrm{v}_{n+1})\big\}.\end{aligned}$$

Let $M = \big(\Sigma^{\langle n+1\rangle}, Q, q_{\text{initial}}, \delta, F\big)$ be a DFA that accepts $\psi^{\mathcal{I}_\Sigma}(\mathrm{v}_1, \dots, \mathrm{v}_{n+1})$. We construct an NFA N with ϵ-moves that 'guesses' the word w_{n+1}. I first give a rough sketch of the construction, see also figure 2-3. The automaton N gets a coded tuple $\langle w_1, \dots, w_n\rangle$ as input. As its first action, N branches nondeterministically to all states that M would reach upon reading the first symbols of the words w_1 to w_n plus some arbitrary symbol in the last component of the input vector. In the next step, it branches to all states that M reaches upon reading the second symbols of the input words plus some arbitrary symbol in the last component, and so on. When the end of the input has been reached, the automaton may go on 'guessing' the word w_{n+1} using ϵ-moves. At any point N may nondeterministically decide that the end of the guessed word has been reached. From then on it simulates deterministically what M would do upon reading blank symbols in the last component.

In detail, the input alphabet of N is $\Sigma^{\langle n\rangle}$. The state set of N is $Q \times \{\text{before}, \text{after}\}$. The set $Q \times \{\text{before}\}$ corresponds to the states of M before the end of w_{n+1} has been guessed, the set $Q \times \{\text{after}\}$ corresponds to the states of M afterwards.

The set of initial states is $\{q_{\text{initial}}\} \times \{\text{before}\}$, the set of accepting states is $F \times \{\text{after}\}$. The state graph of N is defined as follows. Recall that the elements $(\sigma_1, \dots, \sigma_{n+1})^{\mathrm{t}}$ of $\Sigma^{\langle n+1\rangle}$ are column vectors, where the superscript 't' stands for 'transpose'. For every vector $(\sigma_1, \dots, \sigma_{n+1})^{\mathrm{t}} \in \Sigma^{\langle n+1\rangle}$ with $\delta\big(q, (\sigma_1, \dots, \sigma_{n+1})^{\mathrm{t}}\big) = q'$ there is an edge from (q, before) to (q', before), if $\sigma_{n+1} \neq \Box$; and there is an edge from (q, after) to (q', after), if $\sigma_{n+1} = \Box$. The label of the edge is $(\sigma_1, \dots, \sigma_n)^{\mathrm{t}}$; except if $\sigma_1 = \dots = \sigma_n = \Box$, where the label is ϵ instead. For each state $q \in Q$ there is an

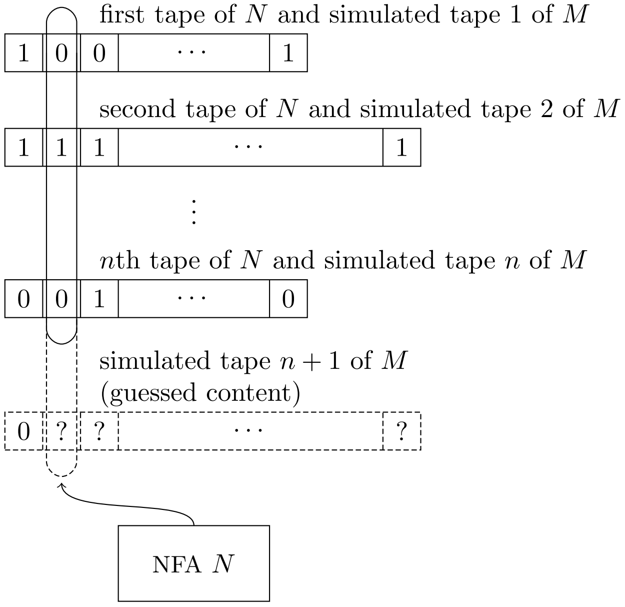

first tape of N and simulated tape 1 of M
1
0
0
···
1
second tape of N and simulated tape 2 of M
1
1
1
···
1
⋮
nth tape of N and simulated tape n of M
0
0
1
···
0
simulated tape $n+1$ of M
(guessed content)
0
?
?
···
?
NFA N

FIGURE 2-3
Tapes of the n-tape automata N from the third part of the proof of theorem 2.36. The dashed tape is a 'virtual' tape. It is a simulation of tape number $n+1$ of the $(n+1)$-tape automaton M. Its content is nondeterministically guessed by N while reading the input on the first n tapes.

ϵ-edge from (q, before) to (q, after), which corresponds to guessing the end of w_{n+1}.

To see that the construction ensures $\phi^{\mathcal{I}_\Sigma}(\mathrm{v}_1, \dots, \mathrm{v}_n) = \{(w_1, \dots, w_n) \in (\Sigma^*)^n \mid \langle w_1, \dots, w_n \rangle \in \mathrm{L}(N)\}$, let any coded word tuple $\langle w_1, \dots, w_n \rangle$ with $(w_1, \dots, w_n) \in \phi^{\mathcal{I}_\Sigma}(\mathrm{v}_1, \dots, \mathrm{v}_n)$ be given. Then there exists a word $w_{n+1} \in \Sigma^*$ such that $\langle w_1, \dots, w_{n+1} \rangle$ is accepted by M. For an appropriate nondeterministic path N will accept $\langle w_1, \dots, w_n \rangle$, namely on the path on which the word w_{n+1} is guessed. On the other hand, any accepting path of N induces a word w_{n+1} such that M accepts $\langle w_1, \dots, w_{n+1} \rangle$. Hence $(w_1, \dots, w_n) \in \phi^{\mathcal{I}_\Sigma}(\mathrm{v}_1, \dots, \mathrm{v}_n)$. QED

2.37 COROLLARY
Let $\mathcal{S}$ be a regular τ-structure, let ϕ be a first-order τ-formula, and let u_1, ... , u_n be first-order variables. Then $\phi^{\mathcal{S}}(u_1, \dots, u_n)$ is regular.

Proof. For regular structures over alphabets that contain at least two symbols, the claim can be deduced from theorem 2.36 as follows. Given $\mathcal{S}$ and ϕ, we can use theorem 2.36 to switch from the regular relations to first-order formulæ that provide elementary definitions of these relations in the index structure $\mathcal{I}_\Sigma$. Substituting the relation symbols in ϕ by these first-order formulæ yields a first-order formula once more. Then, again by theorem 2.36, but this time applied in the other direction, the defined relation is regular.

For *unary* alphabets we cannot use theorem 2.36 directly. However, we can repeat the third part of the proof of theorem 2.36, where it is shown that $\phi^{\mathcal{I}_\Sigma}(u_1, \dots, u_n)$ is regular for every first-order formula ϕ, but with $\mathcal{I}_\Sigma$ replaced by $\mathcal{S}$. This forces us to treat new atomic formulæ, but these are easily taken care of. The difficult part of the proof (the existential quantification) can be copied verbatim since, as pointed out in the proof, it also works for unary alphabets. QED

Applications of Elementary Definitions of Regular Relations

In the rest of this section, three applications of corollary 2.37 are presented. They are the first in a row of applications that continues in the next two chapters. The first application is a theorem on 'regular tournaments', which are tournaments on Σ^* whose edge relations are regular.

2.38 THEOREM
Every regular tournament has a regular linearisation.

Proof. Let (Σ^*, E) be a regular tournament. The well-ordering $<_{\text{std}}$ of Σ^* is regular. By theorem 2.29, there exists a linearisation of (Σ^*, E) that is elementarily definable in $(\Sigma^*, E, <_{\text{std}})$. By corollary 2.37, it is regular. QED

It is an interesting open problem whether every polynomial-time decidable tournament has a polynomial-time decidable linearisation, see (Hemaspaandra et al., 2001) for a detailed discussion of this question. Concerning *recursive* computations, the above proof would also work if every relation were recursive that is elementarily definable in a structure in which all relations are recursive. It is well-known that this is not the case, see counterexample 3.25. However, the class of recursive relations is closed under elementary definitions in which all quantifiers are *bounded.* Since a quick look at the defining formulæ in theorem 2.29 shows that they are bounded, we get a 'purely logical' proof of the below theorem and corollary. Jockusch (1968) attributes the corollary to Appel and McLaughlin. Semirecursive languages are defined in definition 5.1.

2.39 Theorem
Every recursive tournament has a recursive linearisation.

2.40 Corollary (Appel and McLaughlin in Jockusch, 1968)
A language is semirecursive iff it is an initial segment of a recursive linear ordering.

The second application is theorem 2.41 below, which belongs to section 2.2 conceptually. The theorem is proved here since part of the following proof is based on corollary 2.37. The proof demonstrates that corollary 2.37 can be applied in 'mundane' proof situations.

2.41 Theorem
Every regular function can be computed in logarithmic space and also in linear time (though perhaps not both at the same time).

Proof. Let $f\colon \Sigma^* \to \Delta^*$ be regular. Let $M = (\Sigma, Q, q_{\text{initial}}, \delta, F)$ be a DFA that accepts the graph of f. Let $R \subseteq Q \times \Sigma^* \times \Delta^*$ be the following relation: it contains a triple (q, u, v) if M accepts $\langle u, v\rangle$ when started in state q (instead of q_{initial}). Clearly, R is regular. Let $S \subseteq Q \times \Sigma^*$ be defined by $(q, u) \in S :\Longleftrightarrow \exists v\, R(q, u, v)$. It is regular by corollary 2.37.

The Function is Computable in Logarithmic Space
First, let us prove $f \in \text{FL}$ via some logarithmically space-bounded Turing machine M'. On input $w = \sigma_1 \cdots \sigma_n$ the machine M' computes the individual symbols of $\tilde{w} := f(w)$ in a stepwise fashion.

In order to compute the first symbol of $\tilde{w}$, the machine M' checks for which symbols $\tilde{\sigma}_1 \in \Sigma$ we have

$$\Bigl(\delta\bigl(q_{\text{initial}}, \tbinom{\sigma_1}{\tilde{\sigma}_1}\bigr), \sigma_2 \cdots \sigma_n\Bigr) \in S.$$

In other words, we check which first symbols on the second tape make M accept. If there is no such $\tilde{\sigma}_1$, then $\tilde{w} = \epsilon$ and we are done. Otherwise, there must be exactly one $\tilde{\sigma}_1 \in \Sigma$ with this property, since there is only one word $\tilde{w}$ for which $\langle w, \tilde{w} \rangle$ is accepted by M. We output this $\tilde{\sigma}_1$.

Let $q_1 := \delta\big(q_{\text{initial}}, \binom{\sigma_1}{\tilde{\sigma}_1}\big)$. We check for which symbol $\tilde{\sigma}_2 \in \Sigma$ we have

$$\Big(\delta\big(q_1, \tbinom{\sigma_2}{\tilde{\sigma}_2}\big), \sigma_3 \cdots \sigma_n\Big) \in S.$$

If there is no such $\tilde{\sigma}_2$, then $\tilde{w} = \tilde{\sigma}_1$ and we are done. Otherwise, there must exist exactly one possible $\tilde{\sigma}_2$, which we can output. Let $q_2 := \delta\big(q_1, \binom{\sigma_2}{\tilde{\sigma}_2}\big)$ and repeat the process. In case we reach the end of the input before $\tilde{w}$ has ended, we provide blank symbols instead of σ_i's.

The computation will stop at most $|Q|$ steps after the end of the input word and we will have produced the correct output $f(w)$. The only space used is needed for a counter for remembering to which position we must return after each check.

The Function is Computable in Linear Time

It remains to show that f can also be computed in linear time by some Turing machine M'' (the just given algorithm runs in quadratic time).

During the computation, M'' keeps track of a two lists L_i and K_i of pairs (q, u). The index i denotes the current stage. The pairs in the first list consist of a state q and a 'possible prefix' u of $f(w)$. The state q is the state reached by the automaton M after $|u|$ steps on input $\langle w, u \rangle$. This state could be recalculated easily from u, but in order to get a linear time algorithm we have to 'carry this information around'.

The following invariants are maintained over all steps i for the elements of the first list:

1. The 'prefix parts' of the list elements have length i.
2. One of them is the correct prefix of $f(w)$.

The second list also contains pairs (q, u), but here u is not a possible prefix of $f(w)$, but rather a possible value of $f(w)$ itself. For $(q, u) \in K_i$, the state q is the state reached by M on input $\langle w, u \rangle$ after i many steps. For this list we guarantee that the length of each u is at most i, that the correct value of $f(w)$ will enter this list at least once, and that once it has entered the list it will not leave the list anymore. Once the end of the input has been reached (extended by $|Q|$ blanks symbols to allow for the case that $f(w)$ is longer than w), we only have to check for which pair $(q, u) \in K_i$ we have $q \in F$. Then u will be the correct value of $f(w)$.

Let us start with the lists $L_0 := \{(q_{\text{initial}}, \epsilon)\}$ and $K_0 := \{(q_{\text{initial}}, \epsilon)\}$, which certainly have the desired properties. Having constructed a list L_i, we first construct a new list L'_{i+1} in a naïve fashion: we simply extend all prefixes in all possible ways, that is,

$$L'_{i+1} := \bigl\{\bigl(\delta(q,\sigma), u\sigma\bigr) \mid (q,u) \in L_i, \sigma \in \Sigma\bigr\}.$$

Each possible prefix in the newly created list L'_{i+1} is also a candidate for the value of $f(w)$. Setting $K'_{i+1} := K_i \cup L'_{i+1}$ ensures that we do not miss the correct value when it comes along. Note that we do not have to copy words when forming the elements of L'_{i+1}, which would take too long, but can store 'back pointers' that tell us which of the already stored words precedes the letter σ in the word $u\sigma$.

Setting $L_{i+1} := L'_{i+1}$ would maintain the invariants over the whole process, but the size of this list would grow horribly fast, namely exponentially. In order to achieve a linear time algorithm, we must prune the lists.

If for two different words u and u' we have both $(q,u) \in L'_{i+1}$ and $(q,u') \in L'_{i+1}$, then neither u nor u' is a prefix of $f(w)$. To see this, assume $uv = f(w)$. Then $\langle w, uv\rangle$ is accepted by M, but also $\langle w, u'v\rangle$ since u and u' lead to the same state q. Let L_{i+1} be the set of all $(q,u) \in L'_{i+1}$ for which $(q,u') \notin L'_{i+1}$ for all $u' \neq u$. The list L_{i+1} maintains the invariant and has at most $|Q|$ elements.

We must also prune the list K'_{i+1}, since we might otherwise insert a linear number of elements into this list. The same argument as for L'_{i+1} works here: if $(q,u) \in K'_{i+1}$ and $(q,u') \in K'_{i+1}$ with $u \neq u'$, then $f(w)$ is neither u nor u'. This allows us to keep the size of K_i bounded by $|Q|$.

The complete algorithm needs linear time—but also linear space. QED

The third 'application' of corollary 2.37 is just a rephrasing in the terminology of descriptive complexity theory and database theory. In relational database theory, databases can be modelled as relational logical structures. Operations on databases, called queries, transform databases into new databases. For example, we might 'query' which persons in the database live in Berlin. This query would map the database structure to a new, smaller structure containing only these persons.

2.42 Definition of Queries

Given two signatures τ and τ', a *query* is a mapping from the class of all τ-structures to the class of all τ'-structures.

A special kind of queries are queries that can be defined in terms of logical formulæ. First-order queries are especially important, both in database theory, see for example (Abiteboul et al., 1995) for an introduction, and in descriptive complexity theory, see (Immerman, 1999).

2.43 Definition of First-Order Queries

A query I from τ-structures to τ'-structures is called a *first-order query* if there exists a family $(\phi_s)_{s\in\tau'}$ of first-order formulæ with the following property: for every τ-structure $\mathcal{S}$ and every symbol $s \in \tau'$, the formula ϕ_s defines the constant, function, or relation $s^{I(\mathcal{S})}$ elementarily in $\mathcal{S}$.

In this terminology corollary 2.37 can be formulated in a compact way:

2.44 Theorem
The class of regular structures is closed under first-order queries.

Third Chapter

Enumerability

In this chapter enumerability by Turing machines, as introduced by Kummer and Stephan (1994), and by finite automata, as introduced by myself (2002a), is reviewed. Enumerability is generalised to arbitrary computational models. At the end of the chapter the cross product theorem is proved—one of the core results of this dissertation.

The class of functions $f\colon \Sigma^* \to \Sigma^*$ that are known to be computable efficiently, say in polynomial time or—even better—by a finite automaton, is frustratingly small. For example, the function #SAT mentioned in the introduction, which maps propositional formulæ to the number of their satisfying assignments, is presumably not computable in polynomial time. Theorists have proposed many ways of dealing with their frustration. One can weaken one's notion of 'efficiently computable'; for example by allowing randomised computations, by studying the problem's average-case complexity, its fixed parameter tractability, or all at the same time. Alternatively, one can weaken the requirement that the function must be computed *exactly.* Instead, one requires that there is an efficient way of approximating the value $f(w)$ for every w.

Enumerability is an abstract way of producing 'approximations'. An *enumerator* for a function f is a machine or device that outputs a small set of possibilities for $f(w)$ for every input word w. Such a set will be called a *pool* in the following.

Two resource measures are of interest for enumerators. First, we can measure the computational complexity of enumerators. For example, we can measure how much time or space is needed in order to produce a pool. Second, the size of pools is of importance. Intuitively, the smaller a pool, the better.

Regarding the computational complexity of enumerators, the first resource bound that has been studied was polynomial time, see (Cai and Hemachandra, 1989). I focus on the resource-unbounded case, that is, on enumerators that are Turing machines, and on the finite automata case. Regarding the pool size, I focus on constant bounds since the two central functions considered in later chapters, the n-fold characteristic function and the cardinality function, both have finite range. However, especially in the polynomial-time case, one can allow that the pool size varies according to the *length* of the input, see for example (Cai and Hemachandra, 1989); (Beals et al., 1999); and (Ogihara and Tantau, 2002) for interesting results in that direction.

According to my dissertation thesis, Turing machine enumerability and finite automata enumerability are linked. As we shall see, the links are surprisingly tight. Many arguments that work in the recursive setting can also be applied in the finite automata setting and vice versa. Proofs of recursion-theoretic theorems, like Beigel's nonspeedup theorem, need only to be modified slightly in order to prove analogous theorems for automata theory.

Previous papers that treat the links between Turing machine and finite automata enumerability use specialised arguments for the two models, see the proofs in (Tantau, 2002a), (Tantau, 2002b), and (Austinat et al., 2000). Using elementary definitions, I unify these proofs as far as possible. The generic theorems formulated in this chapter and the next apply to all computational models that satisfy certain properties. Finite automata satisfy these properties, Turing machines do so in several important cases. The parallel results for Turing machines and for finite automata are obtained by instantiating the generic theorems with these two computational models. We can also instantiate them with other models, including Presburger arithmetic, ordinal number arithmetic, and computation in the arithmetical hierarchy.

For the formulation of the unified proofs, I introduce the new concept of *generic enumerability.* Turing machine enumerability and finite automata enumerability are instantiations of generic enumerability.

In section 3.1 Turing machine enumerators are reviewed. Equivalent definitions of enumerability from the literature are presented. The equivalence of these definitions shows that enumerability is connected to bounded query complexity theory.

In section 3.2 the definition of finite automata enumerability from (Tantau, 2002a) is reviewed. That definition has the drawback that it is applicable only to functions whose range is finite. I present a new definition of finite automata enumerability that is also meaningful for functions having an infinite range (like the addition function) and show that the new definition is compatible with the old one.

In section 3.3 generic enumerability is defined. Turing machine enumerability, strong Turing machine enumerability (see below), and finite automata enumerability are instantiations of this concept.

In section 3.4 the generic cross product theorem is proved. Corollaries of this theorem include Beigel's nonspeedup theorem (1987), Beigel et al.'s generalised nonspeedup theorem (1995b), and the 'key lemma' of (Tantau, 2002a).

The proofs of other generic theorems are postponed to the next chapter, because they require the introduction of several new concepts for their formulation. By comparison, the cross product theorem is a 'pure' theorem on the structure of enumerability classes.

Section 3.1

Review of Turing Enumerability

A Turing machine can be used as an enumerator of a function in two different ways. The difference lies in whether we require the enumerator to halt on all inputs or not, see the following two definitions, which are due to Kummer and Stephan (1994). Two easy examples demonstrate that the notions differ. At the end of this section alternative definitions of Turing enumerability are reviewed. They show how enumerability is linked to other concepts, for example to bounded query complexity.

3.1 Definition of Turing Enumerators
Let m be a positive integer and let $f\colon \Sigma^* \to \Sigma^*$ be a function. A *Turing m-enumerator* for f is a deterministic Turing machine with an input tape, an output tape, and work tapes. For every input word $w \in \Sigma^*$, it starts its computation with the input tape initialised with w. Its computation may be finite or infinite. During its computation it may print words on the output tape, separated by marker symbols. At most m words may be printed and one of them must be $f(w)$.

3.2 Definition of Strong Turing Enumerator
A *strong Turing m-enumerator* is a Turing m-enumerator that halts on all inputs.

Functions for which there exists a (strong) Turing m-enumerator are called *(strongly) m-Turing-enumerable.* The word 'Turing' is omitted in textbooks on enumerability in recursion theory. Since I study different types of enumerability, adding 'Turing' helps to avoid confusion.

At first sight, every Turing enumerator seems to be transformable to a strong Turing enumerator: once m different words have been printed on the output tape, we can stop the enumerator since no more words will be enumerated. However, an enumerator could also output only $m-1$ words in total. In this case, after it has output these $m-1$ possibilities it could continue its computation infinitely long. We have no way of determining whether an mth word will be output at some time in the future. The following theorem, which is adapted from a slightly less general result in the book of Gasarch and Martin (1999), demonstrates this effect. In the theorem, χ_{K}^n denotes the n-fold characteristic function of the halting problem K. Theorem 3.4 gives an even tighter separation, using a simple new argument.

3.3 Theorem
Let n be a positive integer. Then χ_{K}^n is $(n+1)$-Turing-enumerable, but not strongly (2^n-1)-Turing-enumerable.

Proof. Consider the following Turing enumerator: on input $\langle \operatorname{bin} M_1, \dots, \operatorname{bin} M_n \rangle$ it first outputs the bitstring 0^n; then it starts a parallel simulation of the Turing machines M_i and every time one of them stops, it outputs the bitstring that is 1 exactly for those indices i for which the machine M_i has already stopped. The correct characteristic string will be output when the last machine that is going to stop at all has stopped. Clearly, at most $n+1$ many different outputs are produced.

For the sake of contradiction, assume that χ^n_{K} is strongly $(2^n - 1)$-Turing-enumerable via some M. Since M halts on all inputs, there exists a total Turing machine M' that on input $\langle \operatorname{bin} M_1, \dots, \operatorname{bin} M_n \rangle$ outputs a bitstring of length n that is *not* equal to $\chi^n_{\mathrm{K}}(\operatorname{bin} M_1, \dots, \operatorname{bin} M_n)$. Consider machines $M_1, \dots, M_n$ that do the following: independently of the input, M_i halts iff the ith bit of $M'\big(\langle \operatorname{bin} M_1, \dots, \operatorname{bin} M_n \rangle\big)$ is a 1. The existence of the machines M_i is ensured by the recursion theorem, see (Odifreddi, 1989). Then $\chi^n_{\mathrm{K}}(\operatorname{bin} M_1, \dots, \operatorname{bin} M_n) = M'\big(\langle \operatorname{bin} M_1, \dots, \operatorname{bin} M_n \rangle\big)$, which is a contradiction. QED

3.4 THEOREM
There exists a function that is 2-Turing-enumerable, but not strongly m-Turing-enumerable for any $m \geq 1$.

Proof. Define a function f as follows: for a Turing machine M, let $f\big(\operatorname{bin} M\big)$ be the number of steps that M makes on an empty input; if M never stops, let $f\big(\operatorname{bin} M\big) := 0$. The function f is 2-Turing-enumerable via a machine M' that first outputs 0, starts a simulation of its input, and outputs the number of simulation steps made if the simulated machine halts at some point. The function f is not strongly m-Turing-enumerable for any $m \geq 1$, since it is not bounded by any total recursive function while every strongly m-Turing-enumerable function is. QED

The next two theorems characterise (strong) Turing enumerability in terms of other notions. Only the third point is new, the other characterisations are proved in (Gasarch and Martin, 1999). A relation $R \subseteq U^2$ is *m-bounded* if for every $x \in U$ there exist at most m different $y \in U$ with $(x, y) \in R$.

3.5 CHARACTERISATION THEOREM FOR TURING ENUMERABILITY
Let m be a positive integer and $f\colon \Sigma^* \to \Sigma^*$ a function. Then the following statements are equivalent:

1. *The function f is m-Turing-enumerable via a machine M.*
2. *There exist partial recursive functions $f_1, \dots, f_m\colon \Sigma^* \to_{\text{partial}} \Sigma^*$ such that $f(w) \in \big\{ f_i(w) \mid i \in \{1, \dots, m\},\ f_i(w) \text{ is defined} \big\}$ for all $w \in \Sigma^*$.*
3. *There exists a recursively enumerable, m-bounded relation R that contains f's graph.*

If $\log_2 m$ *is an integer, the statements are furthermore equivalent to:*

4. *There exists an oracle* X *and an oracle Turing machine* $M^{()}$ *such that on every input* w *the machine* $M^{()}$ *computes* $f(w)$ *relative to the oracle* X *and asks at most* $\log_2 m$ *queries.*

Proof. The equivalence of statements 1, 2, and 4 is shown in (Gasarch and Martin, 1999). Let us concentrate on the first and third statement. Suppose the first statement holds. Let R be the relation that contains all pairs (w, v) such that v is one of the outputs of M on input w. Clearly, R is recursively enumerable, m-bounded, and f's graph is contained in R. For the other direction, let R be recursively enumerable via a Turing machine M'. Consider the following Turing machine M: on input w it starts an infinite dovetailed computation that simulates M' on all inputs (w, v). Whenever $(w, v) \in R$, the machine M outputs v. Since R is m-bounded, this machine will output at most m different v's. Since $\big(w, f(w)\big) \in R$, this machine will also output $f(w)$. QED

The following theorem shows how strong Turing enumerability can be defined equivalently. Once more, only the third statement is not treated in (Gasarch and Martin, 1999) and the missing equivalence can be proved similarly to the above proof. Note that, somewhat counter-intuitively, a relation can be recursive and m-bounded without being recursively bounded. For example, the graph of the function introduced in theorem 3.4 has this property.

3.6 Characterisation Theorem for Strong Turing Enumerability
Let m *be a positive integer and* $f\colon \Sigma^* \to \Sigma^*$ *a function. Then the following statements are equivalent:*

1. *The function* f *is strongly* m*-Turing-enumerable.*
2. *There exist (total) recursive functions* $f_1, \dots, f_m\colon \Sigma^* \to \Sigma^*$ *such that* $f(w) \in \big\{f_1(w), \dots, f_m(w)\big\}$ *for all* $w \in \Sigma^*$*.*
3. *There exists a recursive, recursively bounded,* m*-bounded relation* R *that contains* f*'s graph.*

If $\log_2 m$ *is an integer, the statements are furthermore equivalent to:*

4. *There exists an oracle* X *and an oracle Turing machine* $M^{()}$ *such that on every input* w *the machine* $M^{()}$ *computes* $f(w)$ *relative to the oracle* X*, asks at most* $\log_2 m$ *queries, and halts on all inputs relative to all oracles.*

Review of Finite Automata Enumerability

Finite automata enumerators were first proposed in (Tantau, 2002a), where they are defined as follows: for a positive integer m, a finite set X, and a function $f\colon (\Sigma^*)^n \to X$, a *finite automaton m-enumerator* for f is a DFA M whose final output on every input of the form $\langle w_1, \dots, w_n\rangle$ is a set of size at most m that contains $f(w_1, \dots, w_n)$. Such a function f is called *m-fa-enumerable.*

3.7 Example
Let $B = \{\epsilon, b_1, b_1 b_2, b_1 b_2 b_3, \dots\}$ with $b_i \in \{0,1\}$ for $i \in \{1,2,3,\dots\}$ be a branch in the 'tree' $\{0,1\}^*$. Consider the function $\chi_B^2\colon \{0,1\}^* \times \{0,1\}^* \to \{00, 01, 10, 11\}$. It maps a pair (u,v) of bitstrings to its characteristic string with respect to B. Trivially, this function is 4-fa-enumerable. The automaton in figure 3-1 shows that it is also 3-fa-enumerable, even if B is not regular. It is easily seen that χ_B^2 is 1-fa-enumerable iff B is regular.

This leaves one open case: for which B is χ_B^2 a 2-fa-enumerable function? A major result of the next chapter, the finite automata nonspeedup theorem, shows that χ_B^2 is 2-fa-enumerable iff B is regular.

3.8 Example
For a positive integer k, let A be the following regular language: $\{0^k 1 w \mid w \in \{0,1\}^*\}$. Let $\#_A^2\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1,2\}$ be the function that maps every pair (b,c) of bitstrings to $\bigl|\{b,c\} \cap A\bigr|$.

Since A is regular, $\#_A^2$ is 1-fa-enumerable via an appropriate automaton. However, any such automaton must have at least k states. Opposed to this, we can trivially construct a finite automaton 3-enumerator for $\#_A^2$ with just one state (we can do this for *every* language A).

Concerning 2-fa-enumerability, figure 3-2 shows that $\#_A^2$ can be 2-fa-enumerated by an automaton that has just four states. This observation is important, because we shall see in the next chapter that every language A for which $\#_A^2$ is 2-fa-enumerable must be regular. Thus, the four-state automaton from figure 3-2 'proves' that A is regular, although every automaton that accepts A must have at least k states.

A disadvantage of the above definition of finite automata enumerators is that we can only enumerate functions that have a finite range. In order to investigate the enumerability of functions like the addition function or the multiplication function, we need a more general definition of enumerability. Preferably, this general definition should be 'compatible' with the existing definition for finite ranges. The following definition achieves this.

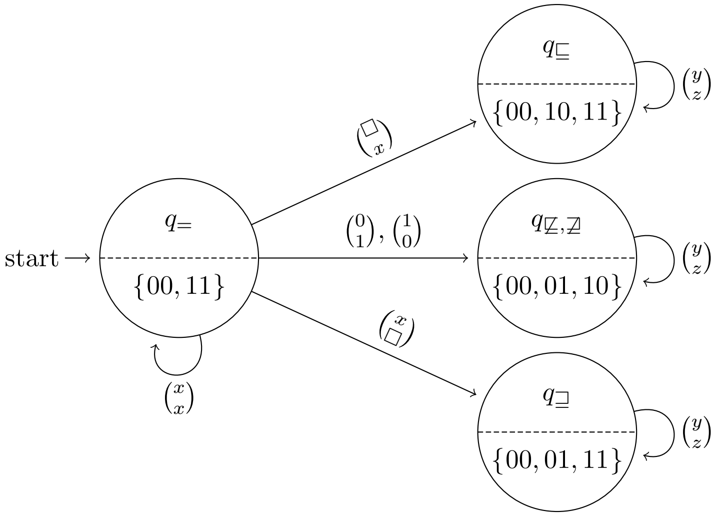


FIGURE 3-1
A DFA that witnesses that χ_B^2 is 3-fa-enumerable, if B is a branch in the tree $\{0,1\}^*$. Here $x \in \{0,1\}$ and $\binom{y}{z} \in \{0,1\}^{\langle 2 \rangle}$ denote arbitrary values.

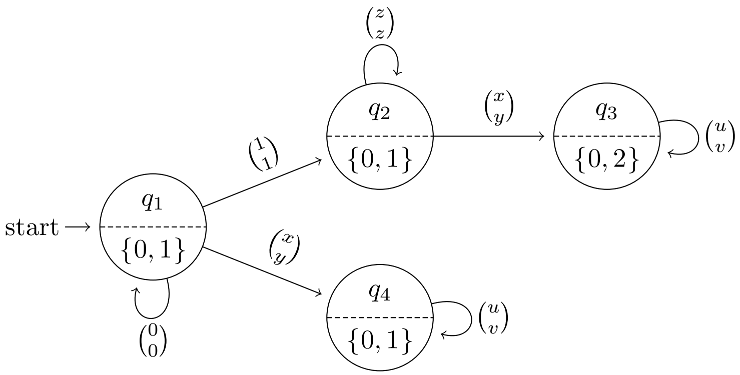


FIGURE 3-2
A DFA that witnesses that $\#_A^2$ is 2-fa-enumerable, where $A = \{0^k 1 w \mid w \in \{0,1\}^*\}$ for some fixed positive integer k. Here $x, y \in \{0, 1, \Box\}$ with $x \neq y$ denote arbitrary values, just like $z \in \{0,1\}$ and $\binom{u}{v} \in \{0,1\}^{\langle 2 \rangle}$. To see that this automaton works correctly, assume that two *different* words in A are given. The automaton will pass through the states q_1, q_2, and will end in q_3, where the output $\{0,2\}$ is produced. This output is correct since it contains the number 2. If only one of the words is in A or if they are identical, the state q_3 cannot be reached and the output $\{0,1\}$ is correct. If none of the words is in A, the output is correct since all output pools contain the number 0.

3.9 DEFINITION OF FINITE AUTOMATON ENUMERATORS
Let m be a positive integer and let $f\colon \Sigma^* \to \Sigma^*$ be a function. A *finite automaton m-enumerator* for f is a DFA M such that for all words $w \in \Sigma^*$

1. the coded pair $\langle w, f(w)\rangle$ is accepted by M and
2. there are at most m different u such that $\langle w, u\rangle$ is accepted by M.

We say that the function f is *m-fa-enumerable*.

The definition is easily transferred to functions $f\colon (\Sigma^*)^n \to (\Sigma^*)^k$ that take multiple words as input and produce multiple words as output.

The following theorem shows that the definition of finite automata enumerability given in (Tantau, 2002A) is a special case of definition 3.9.

3.10 THEOREM
Let m be a positive integer and X a finite set. A function $f\colon \Sigma^* \to X$ is m-fa-enumerable in the sense of (Tantau, 2002A) iff it is m-fa-enumerable in the sense of definition 3.9.

Proof. For the first direction, assume that f is m-fa-enumerable in the sense of (Tantau, 2002A) via some DFA M with output. A finite automaton m-enumerator M' for f in the sense of definition 3.9 works as follows: On input $\langle w, x\rangle$, it 'stores x in its state' and starts a simulation of M on input w. The input is accepted iff the final output of the simulated automaton contains x. For the other direction, on input w, we run M' in parallel on $\langle w, x\rangle$ for each $x \in X$ and output the set of all x for which $\langle w, x\rangle$ is accepted. QED

Just like m-Turing-enumerability, m-fa-enumerability can be defined alternatively. The following theorem is an analogue to theorems 3.5 and 3.6.

3.11 CHARACTERISATION THEOREM FOR FINITE AUTOMATA ENUMERABILITY
Let m be a positive integer and $f\colon \Sigma^* \to \Sigma^*$ a function. Then the following statements are equivalent:

1. *The function f is m-fa-enumerable via a DFA M.*
2. *There exist regular functions $f_1, \dots, f_m\colon \Sigma^* \to \Sigma^*$ such that $f(w) \in \{f_1(w), \dots, f_m(w)\}$ for all $w \in \Sigma^*$.*
3. *There exists a regular, m-bounded relation R that contains f's graph.*

Proof. The equivalence of the first and last statements follows directly from the definitions. Let us show that the second and third statements are equivalent. Suppose the second statement holds. Define a relation R as follows:

$$(w, v) \in R :\Longleftrightarrow v = f_1(w) \vee \dots \vee v = f_m(w).$$

By corollary 2.37, this elementary definition of R in terms of regular functions guarantees that R is regular. Clearly, R is m-bounded and contains f's graph.

Suppose the third statement holds. We must define the functions f_i. For a word w let $v_1 <_{\text{std}} \cdots <_{\text{std}} v_\ell$ be all words for which we have $(w, v_i) \in R$. Since R is m-bounded, $\ell \leq m$. Define $f_i(w)$ as follows: let $f_i(w) := v_i$ for $i \in \{1, \dots, \ell\}$ and let $f_i(w) := v_1$ for $i \in \{\ell + 1, \dots, m\}$. This ensures $f(w) \in \{f_1(w), \dots, f_m(w)\}$ for all $w \in \Sigma^*$. It remains to show that the functions f_i are regular.

The following formula $\psi_{\geq i}(w, v)$ is true if $v = v_j$ for some $j \geq i$. The formula $\psi_{=i}(w, v)$ is true if $v = v_i$.

$$\begin{aligned}
&\psi_{\geq i}(w, v) := R(w, v) \land \exists v_1 \cdots \exists v_{i-1} \,.\\
&\quad \text{distinct}(v_1, \dots, v_{i-1}) \land \textstyle\bigwedge_{j=1}^{i-1} v_j <_{\text{std}} v \land \bigwedge_{j=1}^{i-1} R(w, v_j).\\
&\psi_{=i}(w, v) := \psi_{\geq i}(w, v) \land \neg\, \psi_{\geq i+1}(w, v).
\end{aligned}$$

Here, $\text{distinct}(v_1, \dots, v_{i-1})$ is a shorthand for $\bigwedge_{1 \leq j < k \leq i-1} \neg\, v_j = v_k$. With this preparation, the functions f_i can be defined as follows:

$$f_i(w) = v \;:\!\iff\; \psi_{=i}(w, v) \lor \,.\, \psi_{=1}(w, v) \land \forall v' \, \neg\, \psi_{=i}(w, v').$$

The formula expresses that $f_i(w) = v$ if either v is the i-th word in standard ordering with $(w, v) \in R$ or, if there does not exists an ith such word, v is the first such word. By corollary 2.37, the functions f_i are regular. QED

Section 3.3

Definition of Generic Enumerability

In this section a new notion called *generic enumerability* is defined. By 'plugging in' different computational models, different notions of enumerability are obtained that have been studied in the literature. Generic enumerability is general enough to deal with functions that do not map words to words, but numbers to numbers or even ordinal numbers to ordinal numbers.

How should we define generic enumerability? Theorems 3.5, 3.6, and 3.11 from the previous two sections showed that Turing enumerability, strong Turing enumerability, and finite automata enumerability can all be characterised equivalently in similar ways. Both the second and the third statements of the characterisation theorems could be used for a generic definition of enumerability: in all three theorems, the statements differ only in the class of functions, respectively relations, that is used. By making these classes a parameter, we obtain generic enumerability. I choose the characterisation of enumerability in terms of bounded relations as the basis of my definition of generic enumerability. The reason is that classes of relations are technically easier to handle than classes of (possibly partial) functions.

3.12 Definition of Generic Enumerability
Let m be a positive integer. A function f is *m-enumerated by a relation R* if f's graph is contained in R and R is m-bounded. The class of all functions that are m-enumerated by some relation in a class C of relations is denoted $\mathrm{EN}_C(m)$.

It will be convenient to consider the set product $\times$ to be associative. With this convention, the graph of a function $f\colon U^n \to U^m$ can be a subset of a relation $R \subseteq U^{n+m}$. This will be useful whenever definition 3.12 needs to be applied to functions that take multiple input elements. The following notation is another convenience.

3.13 Notation
Let $n \geq k \geq 1$ and let $R \subseteq U^n$ be an n-ary relation. For $u_1, \ldots, u_k \in U$, let $R[u_1, \ldots, u_k] := \bigl\{(u_{k+1}, \ldots, u_n) \in U^{n-k} \mid (u_1, \ldots, u_n) \in R\bigr\}$. Let us call $R[u_1, \ldots, u_k]$ the *set enumerated by R on u_1, ... , u_k*. In formulæ, the notation '$(u_{k+1}, \ldots, u_n) \in R[u_1, \ldots, u_k]$' means '$R(u_1, \ldots, u_n)$'.

3.14 Example: Turing Enumerability
Let RE-RELATIONS denote the class of all recursively enumerable relations over arbitrary alphabets. By theorem 3.5, m-Turing-enumerability is an instantiation of definition 3.12 for $C = \text{RE-RELATIONS}$. Let us abbreviate $\mathrm{EN}_{\text{RE-RELATIONS}}(m)$ by $\mathrm{EN}_{\text{re}}(m)$. In the literature on recursion theory subscripts are usually omitted for this class since, there, Turing enumerability is the central notion of enumerability.

3.15 Example: Strong Turing Enumerability
Let REC-RELATIONS-BOUNDED denote the class of all recursive relations that are recursively bounded. By theorem 3.6, strong m-Turing-enumerability is captured by the class $\mathrm{EN}_{\text{REC-RELATIONS-BOUNDED}}(m)$. This class is denoted $\mathrm{EN}_S(m)$ in the book of Gasarch and Martin (1999).

3.16 Example: Finite Automata Enumerability
Let REGULAR-RELATIONS denote the class of all regular relations. By theorem 3.11, m-fa-enumerability is an instantiation of definition 3.12 for $C = \text{REGULAR-RELATIONS}$. Let $\mathrm{EN}_{\text{fa}}(m)$ abbreviate $\mathrm{EN}_{\text{REGULAR-RELATIONS}}(m)$.

3.17 Example: Enumerability in Presburger Arithmetic
Let presburger-arithmetic denote the class of relations that are elementarily definable in the structure $(\mathbb{N}, +)$. This class is a fragment of *Peano arithmetic*, due to Giuseppe Peano, where alongside the addition function the multiplication function is also available. While Kurt Gödel (1934), see also (Gödel, 1931), has shown that full Peano arithmetic is not decidable, Mojżesz Presburger (1929) has shown that the fragment is decidable. For this reason it is nowadays called *Presburger arithmetic*. Functions in $\text{EN}_{\text{PRESBURGER-ARITHMETIC}}(m)$ will be called *m-enumerable in Presburger arithmetic*. Let us abbreviate this class by $\text{EN}_{\text{Pa}}(m)$.

Despite its roots in pure logic, Presburger arithmetic has had a recent renaissance in model checking and protocol specification, see (Bultan et al., 1999) as a starting point for further references. Presburger arithmetic is useful in these applied areas since it is not only decidable, but also decidable in double exponential time (Oppen, 1978). In many practical situations it can even be decided much more efficiently, see for example the article by Wolper and Boigelot (2000) for details.

3.18 Example: Enumerability in Ordinal Number Arithmetic
Let ordinal-arithmetic denote the class of all relations that are elementarily definable in the structure $(\text{On}, +, \cdot)$. The universe is the class On of ordinal numbers, see for example (Jech, 1997) for an introduction, and the two binary functions $+, \cdot : \text{On} \times \text{On} \to \text{On}$ are interpreted as the usual ordinal addition and multiplication. That is, $\alpha + \beta$ is the order-type of the ordering α followed by the ordering β; and $\alpha \cdot \beta$ is the order-type of α many copies of β. Functions in $\text{EN}_{\text{ORDINAL-ARITHMETIC}}(m)$ will be called *m-enumerable in ordinal number arithmetic*. Let us abbreviate this class by $\text{EN}_{\text{On}}(m)$.

As the above examples show, generic enumerability can be instantiated in a variety of ways. You might fear that this very variety shows that the definition is so general that we cannot hope to prove anything useful about it. However, in the next section I prove a profound statement about generic enumerability—the generic cross product theorem—that holds for *all* of the above classes.

Section 3.4

The Generic Cross Product Theorem

In this section the first generic theorem is proved: the generic cross product theorem. Unlike the generic theorems of the next chapter, which concern the enumerability of special functions, the cross product theorem is a structural theorem on enumerability classes. It states that if the cross product $f \times g$ of two functions f and g is $(n+m)$-enumerable, then either f is n-enumerable or g is m-enumerable. This statement holds for all notions of enumerability that are defined in terms of classes of relations satisfying a relatively weak closure property: in essence, they must be closed under positive elementary definitions, see definition 3.21 for details. The class of regular relations and the class of recursively enumerable languages enjoy this closure property and the cross product theorem holds for them. Opposed to this, classes defined in terms of polynomial-time computations do not have this closure property and we shall see that the cross product theorem fails in the polynomial-time setting.

The statement of the theorem is deceptively innocent looking, but it implies Beigel's nonspeedup theorem (Beigel, 1987); Beigel et al.'s generalised nonspeedup theorem (Beigel et al., 1995B); and the 'key lemma' of (Tantau, 2002A).

This section starts with the definition of the weak closure property needed for the formulation of the generic cross product theorem. I also define a stronger closure property that will be used in the next chapter. Several examples of computational models are presented that enjoy these closure properties. The main part of this section is taken up by the proof of the generic cross product theorem. At the end of the section the generic theorem is instantiated for different computational models.

Definition of Closure Properties

The definitions of weak and strong closure properties refer to two simple concepts, namely single-valued refinements and the structure $\mathcal{S}_{U|C}$, which are defined first.

3.19 Definition of Single-Valued Refinements
A *single-valued refinement* of a relation $R \subseteq U^n$ is a relation $R' \subseteq R$ such that for every $(u_1, \dots, u_n) \in R$ there is exactly one $u' \in U$ such that $(u_1, \dots, u_{n-1}, u') \in R'$.

A single-valued refinement of a relation R is the graph of a partial 'choice function' f on R. It 'chooses' for any elements $u_1, \dots, u_{n-1} \in U$ an element $u_n := f(u_1, \dots, u_{n-1})$ such that $(u_1, \dots, u_n) \in R$. It is a partial function since it is undefined if such a choice is not possible. The name 'single-valued refinement' stems from complexity theory, more precisely from the study of NP-selective languages, see the survey entitled 'Much Ado about Functions' by Alan Selman (1996) for details.

The next definition makes classes C of relations accessible to logical formulæ by transforming them to logical structures $\mathcal{S}_{C|U}$. The structure $\mathcal{S}_{C|U}$ allows us to 'talk about every relation in C' in formulæ.

3.20 Definition
For a class C of relations and a set U, let $\mathcal{S}_{C|U}$ be the following structure: Its universe is U. Its signature includes an n-ary relation symbol for each n-ary relation $R \in C$ that has universe U. This symbol is interpreted as the relation R. Furthermore, its signature includes a constant symbol for every singleton set $\{r\} \in C$ with $r \in U$. This symbol is interpreted as r.

The constants are included only as a convenience: for every singleton set $R = \{r\} \in C$, every formula ϕ containing the symbol r can be replaced by the formula $\exists x \mathbin{.} R(x) \wedge \phi'$, where in ϕ' every occurrence of r is replaced by the fresh variable x. Note that the structure $\mathcal{S}_{C|U}$ does not necessarily contain a constant symbol for every element of U, but only those for which the singleton set is in C. Thus in order to use a constant in a formula over the signature of $\mathcal{S}_{C|U}$, we must first ascertain that its singleton set is in C.

An example is the structure $\mathcal{S}_{\text{RE-RELATIONS}\,|\{0,1\}^*}$. Its universe is the set $\{0,1\}^*$. It contains every recursively enumerable relation whose alphabet is $\{0,1\}$. It contains a constant for each bitstring since the singleton set of any bitstring is recursively enumerable.

3.21 Definition of Weakly Closed Relation Classes
A class C of relations is *weakly closed* if for every universe $U \in C$ of relations in C the following conditions are satisfied:

1. C contains an irreflexive well-ordering of U,
2. every relation that is positively elementarily definable in $\mathcal{S}_{C|U}$ is in C,
3. every finite relation that is elementarily definable in $\mathcal{S}_{C|U}$ is in C, and
4. every relation in C has a single-valued refinement in C.

Recall that a well-ordering is a linear ordering in which every subset has a smallest element. A more precise name for the above closure property would be 'positive elementary definition closed, elementarily definable finite relation closed, irreflexively well-ordered, single-valued refinable class', but that name has turned out to be too cumbersome for everyday use.

Note that the second condition refers to *positive* definitions, whereas the third condition refers to *all* elementary definitions. The third condition is automatically satisfied if every singleton set is in C.

3.22 Definition of Strongly Closed Relation Classes
A class C of relations is *strongly closed* if for every universe $U \in C$ of relations in C the following conditions are satisfied:

1. C contains an irreflexive well-ordering of U,
2. every relation that is elementarily definable in $\mathcal{S}_{C|U}$ is in C, and
3. every finite relation on U is in C.

The next lemma justifies the names 'strongly closed' and 'weakly closed'.

3.23 LEMMA
Every strongly closed class is weakly closed.

Proof. Let C be strongly closed. The first three properties of weakly closed classes follow trivially from the corresponding properties of strongly closed classes. To prove the existence of single-valued refinements, consider any relation $R \in C$ and let $<$ denote the irreflexive well-ordering of R's universe. A single-valued refinement R' of R can be defined by

$$\begin{aligned}&(x_1, \ldots, x_n) \in R' :\Longleftrightarrow \\ &\quad R(x_1, \ldots, x_n) \wedge \forall x' \centerdot x' < x_n \rightarrow \neg R(x_1, \ldots, x_{n-1}, x').\end{aligned}$$

QED

3.24 EXAMPLE: RECURSIVELY ENUMERABLE RELATIONS
The class of recursively enumerable relations over arbitrary alphabets is weakly closed. To see this, first note that the irreflexive standard ordering is a well-ordering. Second, the class is closed under positive elementary definitions since it is closed under union and intersection and since dovetailing can be used to 'search' for elements in an existential quantification. Third, every finite relation on words (elementarily definable or not) is recursively enumerable. Fourth, the class contains a single-valued refinement R' for every recursively enumerable relation $R \subseteq (\Sigma^*)^n$. The refinement can be obtained as follows: Suppose R is accepted by a Turing machine M. On input $\langle w_1, \ldots, w_n \rangle$ the refinement machine M' starts a dovetailed simulation of M on $\langle w_1, \ldots, w_{n-1}, w \rangle$ for all $w \in \Sigma^*$. For the first w for which this simulation accepts, M' checks whether $w = w_n$. If so, it accepts; otherwise, it rejects.

The class of recursively enumerable relations is not strongly closed since it is not closed under complement.

3.25 COUNTEREXAMPLE: POLYNOMIAL-TIME COMPUTATIONS
The class of relations that are accepted by deterministic, polynomially time-bounded Turing machines is not weakly closed. To see this, note that

$$\begin{aligned}R := \bigl\{&(\operatorname{bin} M, t) \in \{0,1\}^* \times \{0,1\}^* \mid \\ &M \text{ halts within } |t| \text{ steps on input } \operatorname{bin} M\bigr\}\end{aligned}$$

is an element of this class, while the set K defined by

$$\operatorname{bin} M \in \mathrm{K} :\Longleftrightarrow \exists t\, R(\operatorname{bin} M, t)$$

is exactly the halting problem. For the same reason, the class of recursive relations is not weakly closed.

3.26 Example: Regular Relations
The class of regular relations is strongly closed. First, the irreflexive strict standard ordering is regular. Second, by corollary 2.37 all relations that are elementarily definable in the regular structure $\mathcal{S}_{C|U}$ are regular. Third, every finite relation is regular.

3.27 Example: Presburger Arithmetic
The class of relations definable in Presburger arithmetic is strongly closed, since the well-ordering $<$ of $\mathbb{N}$ can be defined by $a < b :\Longleftrightarrow \neg\, a = b \wedge \exists c\; a + c = b$.

3.28 Example: Ordinal Number Arithmetic
The class of relations that are definable in ordinal number arithmetic is weakly closed, but not strongly closed. It enjoys the first two closure properties of strongly closed classes: First, we can define an irreflexive well-ordering of On by $\alpha < \beta :\Longleftrightarrow \neg\, \alpha = \beta \wedge \exists \gamma\; \alpha + \gamma = \beta$. A proof that this does, indeed, define a well-ordering can be found in the book of Jech (1997). Second, the class is closed under elementary definitions by definition. However, the third closure property is not satisfied, since there are finite relations on ordinal numbers that are not definable in ordinal number arithmetic. To see this, note that there are only countably many first-order formulæ over the signature of ordinal number arithmetic. Hence there are only countably many singleton sets that are definable in ordinal number arithmetic, but there are uncountably many ordinal numbers.

The Generic Cross Product Theorem

3.29 Generic Cross Product Theorem
Let n and m be positive integers, let C be weakly closed, and let $U \in C$ be a universe. Then for any two functions $f, g\colon U \to U$ the following holds: if $f \times g \in \mathrm{EN}_C(n+m)$, then $f \in \mathrm{EN}_C(n)$ or $g \in \mathrm{EN}_C(m)$.

Proof. Let $f \times g \in \mathrm{EN}_C(n+m)$ via a relation R. In the following, two relations F and G are constructed such that either $f \in \mathrm{EN}_C(n)$ via F or $g \in \mathrm{EN}_C(m)$ via G. The relations are constructed using positive elementary definitions and the closure properties of C ensure $F \in C$, respectively $G \in C$.

The construction of the relations F and G is based on an abstract form of *easy-hard arguments*. Easy-hard arguments have been used in complexity theory in different proofs, see for example (Kadin, 1989) or (Hemaspaandra et al., 1998). In such an argument one shows that either all words in Σ^*

are *easy* (in a sense to be defined), in which case a language is, well, easy; or there exists a *hard* word, which allows us to decide all *other* words, provided we know the characteristic value of the hard word.

Translated to the more abstract setting of this proof, 'easy' is a property of the elements of U. If all $u \in U$ are easy, then $f \in \mathrm{EN}_C(n)$ via F will hold. Otherwise, in case a hard element u_{hard} exists, $g \in \mathrm{EN}_C(m)$ via G will hold.

Before we proceed, let us fix some notations. Let $<$ be an irreflexive well-ordering of U that is contained in C. Recall that $R[u,v] = \bigl\{(x,y) \in U^2 \mid (u,v,x,y) \in R\bigr\}$ is the set of all pairs that are 'enumerated' by R for the pair (u,v). Recall also that in formulæ '$(x,y) \in R[u,v]$' means '$R(u,v,x,y)$'.

Definition of Easy Elements and Advisors

Let us call an element $u \in U$ *easy* if there exists a $v \in U$ such that in $R[u,v]$ at least $m+1$ pairs have the same first component x. Such a v will be called an *advisor for u*. The 'advisor relation' $A := \bigl\{(u,v) \mid v \text{ is an advisor for } u\bigr\}$ can be positively elementarily defined as follows:

$$(u,v) \in A \;:\Longleftrightarrow \\ \exists x \exists y_1 \cdots \exists y_{m+1} \,\centerdot\, y_1 < \cdots < y_{m+1} \wedge \textstyle\bigwedge_{i=1}^{m+1} (x, y_i) \in R[u,v].$$

Since this formula is positive, we have $A \in C$. The set of easy elements is also in C, since $\phi_{\text{easy}}(u) := \exists v\, A(u,v)$ is also a positive formula. However, the set of hard elements, defined by $\phi_{\text{hard}}(u) := \neg\phi_{\text{easy}}(u)$, need not be an element of C since its definition involves a negation.

Following the proof outline sketched at the beginning of the proof, let us consider two cases, depending on whether all elements are easy or not.

Case 1: A Hard Element Exists

Suppose there exists a hard element. Let $u_{\text{hard}} \in U$ be the smallest such element with respect to $<$. For the moment, let us assume that both the singleton sets $\{u_{\text{hard}}\}$ and $\bigl\{f(u_{\text{hard}})\bigr\}$ are elements of C. On this assumption, definition 3.20 allows us to use u_{hard} and $f(u_{\text{hard}})$ as constants in formulæ.

Let $y \in G[v] :\Longleftrightarrow \bigl(f(u_{\text{hard}}), y\bigr) \in R[u_{\text{hard}}, v]$. The graph of g is a subset of G since for all $v \in U$ we have $\bigl(f(u_{\text{hard}}), g(v)\bigr) \in R[u_{\text{hard}}, v]$. Since u_{hard} is hard, for all v the set $\bigl\{y \in U \mid \bigl(f(u_{\text{hard}}), y\bigr) \in R[u_{\text{hard}}, v]\bigr\}$ has size at most m, see also figure 3-3. Thus G is m-bounded and $g \in \mathrm{EN}_C(m)$ via G.

It remains to show that the assumption was correct. The first part is easy: since u_{hard} is elementarily definable, by the third closure property of C we have $\{u_{\text{hard}}\} \in C$. The second part is trickier since the function f could map u_{hard} to some 'unmentionable' element $f(u_{\text{hard}})$. Let us fix some element $v_* \in U$ whose singleton set is in C and consider the set $R[u_{\text{hard}}, v_*]$.

$$\begin{tikzcd}
\text{element } v \arrow[d, swap, "\textit{enumerate } R[u_{\text{hard}}, v]"] \\
R[u_{\text{hard}}, v] = \begin{array}{l} \{\dots, \\ \quad \big(f(u_{\text{hard}}), y_1\big), \\ \quad \big(f(u_{\text{hard}}), y_2\big), \\ \qquad \vdots \\ \quad \big(f(u_{\text{hard}}), y_m\big), \\ \quad \dots\} \end{array} \arrow[d, swap, "\begin{array}{r} \textit{consider second components of pairs} \\ \textit{with first component } f(u_{\text{hard}}) \end{array}"] \\
G[v] = \{y_1, y_2, \dots, y_m\}
\end{tikzcd}$$

FIGURE 3-3
Procedure from the proof of theorem 3.29 for enumerating a set that has size at most m and that contains $g(v)$, using the existence of a hard element u_{hard}. By definition, in the set $R[u_{\text{hard}}, v]$ there can be at most m pairs sharing the same first component. Thus there can be at most m different second components of pairs that have $f(u_{\text{hard}})$ as their first component.

It has size at most $n+m$ and we can thus elementarily define the first element of this set, the second element, and so on. Since one of these elements is $\bigl(f(u_{\text{hard}}), g(v_*)\bigr)$, say the ith one, we can construct a formula that singles out the 'first component of the ith element of $R[u_{\text{hard}}, v_*]$'. This formula defines $f(u_{\text{hard}})$ elementarily. By the third closure property of C, this implies $\{f(u_{\text{hard}})\} \in C$.

Case 2: All Elements are Easy
Suppose that all $u \in U$ are easy. Let $A' \in C$ be a single-valued refinement of the advisor relation A. Since all elements u are easy, they all have advisors. Thus A' is the graph of a (total) function that maps every element u to an advisor for u. Let

$$x \in F[u] :\iff \exists v \,.\, A'(u,v) \land \exists y\, (x,y) \in R[u,v].$$

The first part of the formula fixes v to be the advisor for u. In the set $R[u,v]$ at least $m+1$ pairs have the same first component (recall that this was the defining property of advisors). Thus there are at most $n+m-m=n$ different x with $(x,y) \in R[u,v]$, see also figure 3-4. Since the graph of f is a subset of F and since by the closure properties of C we have $F \in C$, we get $f \in \mathrm{EN}_C(n)$. QED

The theorem can also be seen as a lower bound on the enumerability of the cross product of two functions, since its contraposition states that if f and g are not n- and m-enumerable respectively, then $f \times g$ is not $(n+m)$-enumerable. Compare this to the trivial lower bound that $f \times g$ is not $(\max\{n,m\})$-enumerable.

Applying the theorem repeatedly yields the following generalisation:

3.30 Theorem
Let ℓ and $n_1, \dots, n_\ell$ be positive integers, let $f_1, \dots, f_\ell \colon U \to U$ be functions, and let C be weakly closed. If $f_1 \times \dots \times f_\ell \in \mathrm{EN}_C(n_1 + \dots + n_\ell)$, then $f_i \in \mathrm{EN}_C(n_i)$ for some $i \in \{1, \dots, \ell\}$.

Instantiations of the Cross Product Theorem

The following corollaries instantiate the cross product theorem for several computational models that are weakly closed. I also show that the theorem does *not* hold for polynomial-time computations.

3.31 Corollary (Tantau, 2002a, Key Lemma)
If $f \times g \in \mathrm{EN}_{\mathrm{re}}(n+m)$, then either $f \in \mathrm{EN}_{\mathrm{re}}(n)$ or $g \in \mathrm{EN}_{\mathrm{re}}(m)$.

3.32 Corollary (Tantau, 2002a, Key Lemma)
If $f \times g \in \mathrm{EN}_{\mathrm{fa}}(n+m)$, then either $f \in \mathrm{EN}_{\mathrm{fa}}(n)$ or $g \in \mathrm{EN}_{\mathrm{fa}}(m)$.

$$\begin{tikzcd}[row sep=large]
\text{easy element } u \arrow[d, swap, "\textit{use } A' \textit{ to find an advisor}"] \\
\text{advisor } v \arrow[d, swap, "\textit{enumerate } R[u,v]"] \\
R[u,v] = \begin{array}[t]{l} \big\{(x_1,y_1),(x_1,y_2),\dots,(x_1,y_{m+1}), \\ (x_2,y_{m+2}), \\ (x_3,y_{m+3}), \\ \qquad\vdots \\ (x_n,y_{m+n})\big\} \end{array} \arrow[d, swap, "\textit{consider only first components}"] \\
F[u] = \{x_1, x_2, \dots, x_n\}
\end{tikzcd}$$

FIGURE 3-4
Procedure from the proof of theorem 3.29 for enumerating a set of size at most n containing $f(u)$ for easy elements u. Using A', an advisor v is obtained for the easy element u. Since u is easy, $R[u,v]$ will contain $m+1$ pairs that have the same first component (like x_1 in the figure). After removing the second components, at most n possibilities for $f(u)$ are left.

3·33 Corollary
If $f \times g \in \mathrm{EN}_{\mathrm{Pa}}(n+m)$*, then either* $f \in \mathrm{EN}_{\mathrm{Pa}}(n)$ *or* $g \in \mathrm{EN}_{\mathrm{Pa}}(m)$*.*

3·34 Corollary
If $f \times g \in \mathrm{EN}_{\mathrm{On}}(n+m)$*, then either* $f \in \mathrm{EN}_{\mathrm{On}}(n)$ *or* $g \in \mathrm{EN}_{\mathrm{On}}(m)$*.*

The cross product theorem does not hold for polynomial-time computations. In order to show this, some results and terminology from partial information theory (Nickelsen, 2001) are needed.

3·35 Definition of Languages That are But One in P (Tantau, 1999)
A language A is in the class P-but-one if there exists a polynomially time-bounded Turing machine that on input of any number of pairwise different words outputs the characteristic values of all but one of these words.

In other words the machine may choose one word that it deems 'too complicated', but must decide all other words in time polynomial in the total length of the words. This class is also studied in (Nickelsen, 1997), where it has the slightly cryptic name $\mathrm{P}_{\mathrm{dist}}[\text{2-weakMIN}]$. Using a super-sparse set diagonalisation, Nickelsen has shown that there exists a language in P-but-one that is not in P. By increasing the height of the jumps between diagonalisation steps, one can use Nickelsen's argument to show the following stronger result:

3·36 Theorem
For every recursive function f *there exist a language in* P-but-one *that is not in* DTIME$[f]$*.*

For languages A in P-but-one that are not in P, the function $\chi_A \times \chi_A$ is still 2-enumerable via a polynomial-time machine, since on input of different words we can decide at least one word. However, neither $f = \chi_A$ nor $g = \chi_A$ is 1-enumerable in polynomial time since $A \notin \mathrm{P}$. This shows that the cross product theorem fails in the polynomial-time setting.

Fourth Chapter

Towards a Cardinality Theorem for Finite Automata

I conjecture that Kummer's recursion-theoretic cardinality theorem also holds for finite automata. In this chapter I gather evidence for this conjecture. Three theorems are presented that support it: the (generalised) nonspeedup theorem for finite automata, the cardinality theorem for finite automata for two words, and the restricted cardinality theorem for finite automata. Applications of these theorems, which are discussed in the sixth chapter, show that these theorems do not only support my conjecture, but that they are also of independent interest.

The classical recursion-theoretic cardinality theorem concerns the following question: how difficult is it to compute the *cardinality function* $\#_A^n$ for a given language A? The cardinality function takes n words as input and counts how many of them are in A. Formally, it maps $\langle w_1, \dots, w_n \rangle$ to $\bigl|\{w_1, \dots, w_n\} \cap A\bigr|$. Raised in its general form by William Gasarch (1991), this counting problem plays an important rôle in a variety of proofs both in complexity theory (Mahaney, 1982; Immerman, 1988; Szelepcsényi, 1988; Hemachandra, 1989; Kadin, 1989) and recursion theory (Kummer and Stephan, 1994; Beigel et al., 2000). For example, in the proof of the Immerman-Szelepcsényi theorem, which states that nondeterministic space is closed under complementation, the key idea is to *count* the number of reachable vertices in a graph in order to *decide* whether a certain vertex is reachable.

One way of quantifying the complexity of $\#_A^n$ is to consider its enumeration complexity, that is, the smallest m for which $\#_A^n$ is still m-Turing-enumerable. Intuitively, the larger m, the easier it should be to m-Turing-enumerate $\#_A^n$. This intuition is wrong, except for the trivial observation that $\#_A^n$ is $n+1$ enumerable, since its range is contained in $\{0, 1, 2, \dots, n\}$. Kummer's cardinality theorem states that even n-Turing-enumerating $\#_A^n$ is just as hard as deciding A. Intriguingly, the intuition *is* correct for polynomial-time computations: the work of Gasarch (1991); Hoene and Nickelsen (1993); and Nickelsen (1997) shows that a polynomial-time version of the cardinality theorem does not hold.

4.1 Cardinality Theorem (Kummer, 1992)
Let A be a language and $n \geq 1$. If $\#_A^n \in \mathrm{EN}_{\mathrm{re}}(n)$, then A is recursive.

Kummer's proof of the cardinality theorem combines ideas from different areas. Several less general results had already been proved when Kummer wrote his paper 'A Proof of Beigel's Cardinality Conjecture'. The title of Kummer's paper refers to the fact that Richard Beigel (1987) was the first to conjecture the cardinality theorem as a generalisation of his so-called nonspeedup theorem.

4.2 Nonspeedup Theorem (Beigel, 1987)
Let A be a language and $n \geq 1$. If $\chi_A^n \in \text{EN}_{\text{re}}(n)$, then A is recursive.

The premise of the nonspeedup theorem is much stronger than the premise of the cardinality theorem: in order to n-enumerate the function χ_A^n one must narrow the range of possibilities from 2^n possibilities to n possibilities, whereas for $\#_A^n$ one must narrow this range from only $n+1$ to n. The nonspeedup theorem is a consequence of the cardinality theorem: $\chi_A^n \in \text{EN}_{\text{re}}(n)$ implies $\#_A^n \in \text{EN}_{\text{re}}(n)$ since every possibility for $\chi_A^n(w_1, \ldots, w_n)$ induces one possibility for $\#_A^n(w_1, \ldots, w_n)$.

Two years after Beigel's dissertation had been published, James Owings wrote a paper in the Journal of Symbolic Logic entitled 'A Cardinality Version of Beigel's Nonspeedup Theorem'. He succeeded in proving the cardinality theorem for $n = 2$. For larger n he could only show that $\#_A^n \in \text{EN}_{\text{re}}(n)$ implies that A is recursive in the halting problem.

4.3 Cardinality Theorem for Two Words (Owings, 1989)
Let A be a language. If $\#_A^2 \in \text{EN}_{\text{re}}(2)$, then A is recursive.

4.4 Fact (Owings, 1989)
Let A be a language and $n \geq 1$. If $\#_A^n \in \text{EN}_{\text{re}}(n)$, then A is recursive in the halting problem.

Harizanov et al. (1992) have formulated the following 'restricted' cardinality theorem, whose proof is somewhat simpler than the proof of the full cardinality theorem.

4.5 Restricted Cardinality Theorem (Harizanov et al., 1992)
Let A be a language and $n \geq 1$. If $\#_A^n \in \text{EN}_{\text{re}}(n)$ via a Turing machine that never enumerates both 0 and n, then A is recursive.

As stated above, I conjecture that the cardinality theorem also holds for finite automata.

4.6 Conjecture
Let A be a language and $n \geq 1$. If $\#_A^n \in \text{EN}_{\text{fa}}(n)$, then A is regular.

The following three results support the conjecture. They are proved in sections 4.1, 4.2, and 4.3 respectively.

1. The (generalised) nonspeedup theorem holds for finite automata, see corollary 4.10.
2. The conjecture holds for $n = 2$, see corollary 4.17.
3. The restricted form of the conjecture holds for all n, see corollary 4.21.

Together, these results bring us as near to a proof of conjecture 4.6 as did the results in recursion theory before Kummer's breakthrough proof.

Similarly to the previous chapter, the proofs in this chapter are generic and can be applied to different notions of enumerability, provided the notion is defined in terms of a class of relations that has certain closure properties. The last two results require a stronger closure property than the one needed for the proof of the cross product theorem: instead of the classes's being *weakly* closed, it is necessary that they are *strongly* closed.

Since the class of recursively enumerable relations is not strongly closed (it is not closed under complement), the generic proofs of the cardinality theorem for two words and of the restricted cardinality theorem cannot be instantiated for Turing enumerability. In particular, we do not get new proofs of these theorems. However, we do get results similar to Owing's result: since negation can be 'simulated' by an oracle query to the halting problem, we get new proofs of the statements that if $\#_A^2 \in \mathrm{EN}_{\mathrm{re}}(2)$ or if $\#_A^n \in \mathrm{EN}_{\mathrm{re}}(n)$ via a Turing machines that never enumerates both 0 and n, then A must be recursive in the halting problem.

In section 4.4 we study which results of this section can be proved constructively (or, if you prefer, which are 'uniform'). Many results in automata, complexity, and recursion theory can be proved constructively. Consider a statement like 'the intersection of recursively enumerable languages is recursively enumerable'. A typical proof of this statement actually shows the stronger statement 'there exists an (effective) algorithm that gets two Turing machines M_1 and M_2 as input and outputs a Turing machine M such that $\mathrm{L}(M) = \mathrm{L}(M_1) \cap \mathrm{L}(M_2)$'. For this reason, statements like 'the intersection of recursively enumerable languages is recursively enumerable' are called *constructively provable.*

In the recursive setting, the results of this chapter are not constructively provable. For example, the proof of the nonspeedup theorem, which states 'if $\chi_A^n \in \mathrm{EN}_{\mathrm{re}}(n)$, then A is recursive', shows that A is decidable, but it provides no clue to a concrete decision procedure. Indeed, it can be shown that no constructive proof of the nonspeedup theorem is possible. Kaufmann and Kummer (1996) were even able to quantify its 'degree of nonconstructiveness', see fact 4.25. The situation is different for finite automata. I show that 'fair versions' of the three core theorems of this chapter can be formulated constructively.

Section 4.1

The Generic Generalised Nonspeedup Theorem

The first of the three results supporting conjecture 4.6 is the generalised nonspeedup theorem for finite automata, which is proved in this section.

The generalised nonspeedup theorem is a statement about inclusions of *verboseness classes*. These were originally defined in an effort to better understand the structure of undecidable problems. A language A is called (m,n)*-verbose* (Beigel et al., 1995B) if $\chi^n_A \in \mathrm{EN}_{\mathrm{re}}(m)$. The verboseness of a language expresses how difficult it is to enumerate the n-fold characteristic function of the language. The class of all (m,n)-verbose languages will be denoted $\mathrm{V}_{\mathrm{re}}(m,n)$.

All languages are in $\mathrm{V}_{\mathrm{re}}(2^n,n)$, whereas $\mathrm{V}_{\mathrm{re}}(1,n)$ contains exactly the recursive languages. The structure between these two extremes has been subject to thorough investigation. We have $\mathrm{V}_{\mathrm{re}}(n,n) = \mathrm{V}_{\mathrm{re}}(n-1,n) = \cdots = \mathrm{V}_{\mathrm{re}}(1,n)$ for all n by fact 4.2. Beigel et al. (1995B) have shown that all recursively enumerable and all semirecursive (Jockusch, 1966) languages are in $\mathrm{V}_{\mathrm{re}}(n+1,n)$, which equals $\mathrm{V}_{\mathrm{re}}(3,2)$ for $n \geq 2$. They also present a procedure, based on finite combinatorics, for deciding whether $\mathrm{V}_{\mathrm{re}}(m,n) \subseteq \mathrm{V}_{\mathrm{re}}(h,k)$ holds for given numbers m, n, h, and k.

Verboseness has also been studied extensively for the situation where the enumerating Turing machine is restricted to use only a polynomial amount of time. The inclusion structure of polynomial-time verboseness classes, denoted $\mathrm{V}_{\mathrm{p}}(m,n)$ in the following, is quite different from the structure in the recursive setting. For example $\mathrm{V}_{\mathrm{p}}(m,n) \subsetneq \mathrm{V}_{\mathrm{p}}(m+1,n)$ for all $m < 2^n$. Languages that are in $\mathrm{V}_{\mathrm{p}}(n,n)$ for some n are commonly called *cheatable* (Beigel, 1991), languages in the class $\mathrm{V}_{\mathrm{p}}(2^n-1,n)$ are called n*-approximable* (Beigel et al., 1995A) or n*-membership comparable* (Ogihara, 1995). A systematic comparison of polynomial-time verboseness classes with other notions of 'polynomial-time partial information classes' can be found in the dissertation of Arfst Nickelsen (2001) and in the survey (Nickelsen and Tantau, 2003).

In (Tantau, 2002A) finite automata verboseness classes have been defined in the obvious way by letting $A \in \mathrm{V}_{\mathrm{fa}}(m,n)$ if $\chi^n_A \in \mathrm{EN}_{\mathrm{fa}}(m)$. As in the recursive setting, all languages are in $\mathrm{V}_{\mathrm{fa}}(2^n,n)$ and $\mathrm{V}_{\mathrm{fa}}(1,n)$ contains exactly the regular languages. Austinat et al. (2003) have presented different examples of fa-verbose languages that lie between these extremes: for every infinite bitstring b, both the set of all words that are lexicographically smaller than b and the set of all finite prefixes of b are in $\mathrm{V}_{\mathrm{fa}}(3,2)$, see figure 3-1 on page 64. They show that $\mathrm{V}_{\mathrm{fa}}(3,2)$ contains context-sensitive languages that are not context-free and context-free languages that are not regular; but also, that infinite context-free languages lacking infinite regular subsets (like $\{a^ib^i \mid i \in \mathbb{N}\}$) lie outside $\mathrm{V}_{\mathrm{fa}}(2^n-1,n)$ for all n.

The generalised nonspeedup theorem, due to Beigel et al. (1995B), is a statement about the inclusion structure of verboseness classes. It states that $\mathrm{V}_{\mathrm{re}}(m+h,n+k) \subseteq \mathrm{V}_{\mathrm{re}}(m,n) \cup \mathrm{V}_{\mathrm{re}}(h,k)$ for all m, n, h, and k. In the following I present a generic proof of this theorem. Instantiated for

Turing machines and for finite automata, we get the original generalised nonspeedup theorem, respectively the finite automata version.

The generalised nonspeedup theorem is not the 'end of the story' concerning the inclusion structure of verboseness classes. The next step is the derivation of conditions for numbers n, m, h, and k for which $\mathrm{V}_{\mathrm{re}}(m,n) \subseteq \mathrm{V}_{\mathrm{re}}(h,k)$ holds. Such a condition can indeed be formulated: 'every (m,n)-good k-pool has size at most h', see below for the definition of 'good pools'. In the present chapter we still lack the necessary proof machinery for showing this condition to be *necessary.* This is fixed in the next chapter, which treats branch diagonalisation. Nevertheless, in this section we can at least show that the condition is *sufficient.*

Formulation and Proof of the Generalised Nonspeedup Theorem

4.7 Definition of Generic Verboseness
Let C be a class of relations and let m and n be positive integers. The class $\mathrm{V}_C(m,n)$ contains all sets A for which $\chi_A^n \in \mathrm{EN}_C(m)$.

4.8 Generic Generalised Nonspeedup Theorem
Let m, n, h, and k be positive integers and let C be weakly closed. Then

$$\mathrm{V}_C(m+h,n+k) \subseteq \mathrm{V}_C(m,n) \cup \mathrm{V}_C(h,k).$$

Proof. Suppose $A \in \mathrm{V}_C(m+h,n+k)$. Then $\chi_A^{n+k} \in \mathrm{EN}_C(m+h)$. Since $\chi_A^{n+k} = \chi_A^n \times \chi_A^k$, we can apply theorem 3.29 with $f = \chi_A^n$ and $g = \chi_A^k$. This yields that either $\chi_A^n \in \mathrm{EN}_C(m)$ or $\chi_A^k \in \mathrm{EN}_C(h)$ holds. In the first case, $A \in \mathrm{V}_C(m,n)$, and in the second case, $A \in \mathrm{V}_C(h,k)$. QED

4.9 Generic Nonspeedup Theorem
Let n be a positive integer and let C be weakly closed. Then

$$\mathrm{V}_C(n,n) = \mathrm{V}_C(1,1).$$

Proof. The generalised nonspeedup theorem yields

$$\mathrm{V}_C(n,n) \subseteq \mathrm{V}_C(n-1,n-1) \cup \mathrm{V}_C(1,1).$$

Iterating this inclusion yields $\mathrm{V}_C(n,n) \subseteq \mathrm{V}_C(1,1)$. QED

Since the classes of recursively enumerable relations and of regular relations are weakly closed, we get the following corollary:

4.10 Corollary (Generalised Nonspeedup Theorems)
Let m, n, h, and k be positive integers. Then

$$\mathrm{V}_{\mathrm{re}}(m+h,n+k) \subseteq \mathrm{V}_{\mathrm{re}}(m,n) \cup \mathrm{V}_{\mathrm{re}}(h,k),$$
$$\mathrm{V}_{\mathrm{fa}}(m+h,n+k) \subseteq \mathrm{V}_{\mathrm{fa}}(m,n) \cup \mathrm{V}_{\mathrm{fa}}(h,k).$$

4.11 Corollary (Nonspeedup Theorems)
Let A be a language and n a positive integer. If $A \in \mathrm{V}_{\mathrm{re}}(n,n)$, then A is recursive. If $A \in \mathrm{V}_{\mathrm{fa}}(n,n)$, then A is regular.

A Sufficient Condition for the Inclusion of Verboseness Classes

Theorem 4.8 can be used to prove inclusions of verboseness classes. An example is the proof of theorem 4.9, where we used the generic generalised nonspeedup theorem to show $\mathrm{V}_C(m+1,n+1) \subseteq \mathrm{V}_C(m,n)$. A systematic study of the derivable inclusions in the recursive setting has lead Beigel et al. (1995b) to a notion that they call (m,n)-*goodness*. For its definition a special notation is useful, which is also due to Beigel et al. (1995b). In the following, a *k-pool* is an arbitrary subset of $\{0,1\}^k$.

4.12 Notation
Let n and k be positive integers. Let P be a k-pool. Let $\mathrm{g}_P(n)$ denote the maximum cardinality of $\{b[i_1,\dots,i_n] \mid b \in P\}$, where the maximum is taken over all index tuples $(i_1,\dots,i_n) \in \{1,\dots,k\}^n$.

The intuition behind the value $\mathrm{g}_P(n)$ is the following: it is an upper bound on the size of a pool that we have to output for a selection of n words out of k words whose characteristic string is known to lie in P. More precisely, assume that for some language for some words $w_1, \dots, w_k$ we know that their characteristic string is contained in the pool P. Now suppose we have a selection $w_{i_1}, \dots, w_{i_n}$ of n words out of the words $w_1, \dots, w_k$ and we wish to output a minimal n-pool that is guaranteed to contain the characteristic string of these n words. Such an n-pool is given by the set $\{b[i_1,\dots,i_n] \mid b \in P\}$. The number $\mathrm{g}_P(n)$ is a tight upper bound on the size of this pool.

The following definition of goodness is essentially due to Beigel et al. (1995b), although I have modified it slightly by dropping the requirement that the indices must be sorted. This will simplify the proofs later on.

4.13 Definition of Good Pools (Beigel et al., 1995b)
Let m, n, and k be positive integers. A k-pool P is (m,n)-*good*, if for every $\ell \in \{1,\dots,n\}$ and every partition $n_1 + \dots + n_\ell = n$ with $n_1,\dots,n_\ell \in \{1,\dots,n\}$ we have

$$\mathrm{g}_P(n_1) + \dots + \mathrm{g}_P(n_\ell) - \ell + 1 \le m.$$

The notion of goodness generalises the intuition behind $\mathrm{g}_P(n)$. For $\ell = 1$, the definition requires that an (m,n)-good pool P must have the property $\mathrm{g}_P(n) \le m$. Thus if we are given a selection of n words out of k words for which we know that their characteristic string is contained in an (m,n)-good pool, then we can compute an n-pool of size m for them.

However, most of the time we will be given n words that are *not* chosen out of k words with such a nice property. Rather, n_1 many of the n words are among k words for which we know that the characteristic string is contained in P; the next n_2 many of the n words are among (different) k words for which we also know this to be the case; and so on. For the n words, which are scattered among the different blocks of k words, we wish to produce a pool that is as small as possible and that contains their characteristic string.

For the moment, assume that the pool is not only (m,n)-good, but that it also satisfies the requirement '$\mathrm{g}_P(n_1) \cdot \ldots \cdot \mathrm{g}_P(n_\ell) \leq m$'. Such a pool might be called *(m,n)-hyper-good*. For a hyper-good pool, we can easily produce a pool of maximum size m for the input words: simply output all combinations of the possible bitstrings for the first n_1 words, for the next n_2 words, and so on. Since there are at most $\mathrm{g}_P(n_1)$ possibilities for the characteristic string of the first n_1 words, at most $\mathrm{g}_P(n_2)$ possibilities for the characteristic string of the next n_2 words, and so on, there are at most $\mathrm{g}_P(n_1) \cdot \ldots \cdot \mathrm{g}_P(n_\ell)$ possibilities altogether, which would be bounded by m.

Unfortunately, good pools are not necessarily hyper-good. For good pools it is only required that the much smaller number $\mathrm{g}_P(n_1) + \cdots + \mathrm{g}_P(n_\ell) - \ell + 1$ is bounded by m. In order to output just m possibilities for the n input words, the language must have some extra structure that allows us to combine the bitstrings for the ℓ different word blocks in a more economic way. This 'economic way of combining' must ensure that each additional block of n_i words only produces $\mathrm{g}_P(n_i) - 1$ new possible bitstrings for *all* input words. Languages for which this is possible will be studied in the next chapter.

4.14 Theorem (Sufficient Condition for Inclusion)
Let m, n, h, and k be positive integers and let C be weakly closed. Let every (m,n)-good k-pool have size at most h. Then $\mathrm{V}_C(m,n) \subseteq \mathrm{V}_C(h,k)$.

Proof. Let $A \in \mathrm{V}_C(m,n)$ via a relation S. To prove $A \in \mathrm{V}_C(h,k)$ we show that there exists a relation $R \in C$ that contains the graph of χ_A^k for which $R[x_1,\ldots,x_k]$ is an (m,n)-good k-pool for all x_1, ..., x_k in the universe of S. By assumption, this will ensure that $R[x_1,\ldots,x_k]$ has size at most h. Thus R will be h-bounded.

For each $i \in \{1,\ldots,n\}$ let m_i be the smallest number such that $A \in \mathrm{V}_C(m_i,i)$ via some relation $R_i \in C$. Applying theorem 4.8 to the class $\mathrm{V}_C(m_{i+j},i+j)$ yields

$$A \in \mathrm{V}_C(m_{i+j},i+j) \subseteq \mathrm{V}_C(m_i-1,i) \cup \mathrm{V}_C\big(m_{i+j}-(m_i-1),j\big).$$

Since A is not an element of $\mathrm{V}_C(m_i-1,i)$ by the minimality of m_i, it must lie in $\mathrm{V}_C(m_{i+j}-m_i+1,j)$. Because of the minimality of m_j, this yields

$m_j \le m_{i+j} - m_i + 1$ and thus $m_i + m_j \le m_{i+j} + 1$.

The relation R is defined as follows:

$$b \in R[x_1, \dots, x_k] \;:\Longleftrightarrow$$
$$\bigwedge_{j=1}^{n} \bigwedge_{i_1=1}^{k} \cdots \bigwedge_{i_j=1}^{k} b\,[i_1, \dots, i_j] \in R_j\bigl[x_{i_1}, \dots, x_{i_j}\bigr].$$

The formula is not entirely legal, since '$b\,[i_1, \dots, i_j]$' is not a legal first-order term. This abuse of notation could be avoided by replacing for example $b[5,4] \in R_2[x_5, x_4]$ by the more verbose formula $\bigvee_{b' \in \{0,1\}^k} . \, b = b' \wedge b'[5,4] \in R_2[x_5, x_4]$.

The definition of R ensures $\chi_A^k(x_1, \dots, x_k) \in P := R[x_1, \dots, x_k]$. It remains to show that P is (m,n)-good for all x_1, … , x_k in S's universe. To see this, let $n_1 + \dots + n_\ell = n$ be any partition with $\ell \in \{1, \dots, n\}$ and $n_1, \dots, n_\ell \in \{1, \dots, n\}$. We have $\mathrm{g}_P(j) \le m_j$, since for any indices $i_1, \dots, i_j \in \{1, \dots, k\}$ the set $R_j\bigl[x_{i_1}, \dots, x_{i_j}\bigr]$ has size at most m_j. Hence for every partition $n_1 + \dots + n_\ell = n$ with $\ell \in \{1, \dots, n\}$ and $n_1, \dots, n_\ell \in \{1, \dots, n\}$,

$$\begin{aligned} \mathrm{g}_P(n_1) + \dots + \mathrm{g}_P(n_\ell) - \ell + 1 &\le m_{n_1} + \dots + m_{n_\ell} - \ell + 1 \\ &\le m_n \le m. \end{aligned}$$

This follows from the inequality $m_i + m_j \le m_{i+j} + 1$ established above and the trivial inequality $m_n \le m$. QED

4.15 Corollary
Let m, n, h, and k be positive integers. Let every (m,n)-good k-pool have size at most h. Then $\mathrm{V}_{\mathrm{re}}(m,n) \subseteq \mathrm{V}_{\mathrm{re}}(h,k)$ and $\mathrm{V}_{\mathrm{fa}}(m,n) \subseteq \mathrm{V}_{\mathrm{fa}}(h,k)$.

Section 4.2

The Generic Cardinality Theorem for Two Input Words

The aim of this section is to prove the finite automata cardinality conjecture for $n = 2$. As before, we start with a generic version of the claim. Unlike the generic theorems of the previous section, the following theorem is formulated only for classes of relations that are *strongly* closed.

4.16 Theorem
Let C be strongly closed and let A be a set. If $\#_A^2 \in \mathrm{EN}_C(2)$, then $A \in C$.

Proof. Suppose $\#_A^2 \in \mathrm{EN}_C(2)$ via a relation $R \in C$. Our first aim is to switch from the cardinality function $\#_A^2$ to the characteristic function χ_A^2.

Ideally, if we could show $\chi_A^2 \in \mathrm{EN}_C(2)$, then theorem 3.29 would yield the claim. Unfortunately, if R enumerates both the numbers 0 and 1 on input (x, y) with $x \neq y$, we only know $\chi_A^2(x, y) \in \{00, 01, 10\}$; and if R enumerates both the numbers 1 and 2, we only know $\chi_A^2(x, y) \in \{01, 10, 11\}$. Thus, as first step, we only show $\chi_A^2 \in \mathrm{EN}_C(3)$.

Let $C_2 \in C$ be the ternary relation that is defined as follows:

$$\begin{aligned} b \in C_2[x, y] \;:\Longleftrightarrow \quad & \big(b = 00 && \to\!\!\textbf{.}\; 0 \in R[x, y] \lor x = y\big) \\ \land\, & \big(b = 01 \lor b = 10 && \to\!\!\textbf{.}\; 1 \in R[x, y] \land \neg\, x = y\big) \\ \land\, & \big(b = 11 && \to\!\!\textbf{.}\; 2 \in R[x, y] \lor x = y\big). \end{aligned}$$

The graph of χ_A^2 is contained in C_2 (hence the name) and $C_2[x, y]$ is always a subset of one of the following pools: $\{00, 01, 10\}$, $\{00, 11\}$, and $\{01, 10, 11\}$.

You may have noticed that, unlike 0, 1, and 2, the constants 00, 01, 10, and 11 are not necessarily in the universe of the relation R. Thus we might be unable to refer to these constants in the formula that defines the relation C_2. However, these constants are only used ‘internally’ and we can pick any four distinct elements of R's universe and interpret them as 00, 01, 10, and 11 respectively. (If R's universe has less than four elements, the claim is trivial since the universe is ordered and *all* of its subsets can be defined elementarily.)

The second aim is to enumerate pools of minimal size for χ_A^3, that is, for any *three* input elements. This is achieved by a relation C_3 that is defined by $b \in C_3[x, y, z] :\Longleftrightarrow b\texttt{[1,2]} \in C_2[x, y] \land b\texttt{[1,3]} \in C_2[x, z] \land b\texttt{[2,3]} \in C_2[y, z]$. The formula expresses that the bitstring $b \in \{0, 1\}^3$ is consistent with the sets enumerated by C_2 on every selection of two elements. In particular, $\chi_A^3(x, y, z) \in C_3[x, y, z]$. As in the proof of theorem 4.14, the formula could be legalised if desired.

The next step is to employ an easy-hard argument similar to the argument used in the proof of theorem 3.29. This time, let us call a *pair* (x, y) of elements *easy* if there exists an element z such that $\big\{b\texttt{[1,2]} \mid b \in C_3[x, y, z]\big\}$ has size at most 2. The element z will be called an *advisor* for (x, y). The advisor relation, denoted B in this proof in order to avoid a name clash with the language A, is the following ternary relation:

$$\begin{aligned} &(x, y, z) \in B \;:\Longleftrightarrow \\ &\qquad \neg \bigvee_{\substack{b,c,d \in \{0,1\}^2,\\ b,c,d \text{ distinct}}} \textbf{.} \quad \begin{aligned} &\big(b0 \in C_3[x, y, z] \lor b1 \in C_3[x, y, z]\big) \\ \land\, &\big(c0 \in C_3[x, y, z] \lor c1 \in C_3[x, y, z]\big) \\ \land\, &\big(d0 \in C_3[x, y, z] \lor d1 \in C_3[x, y, z]\big). \end{aligned} \end{aligned}$$

The formula $\phi_{\text{easy}}(x, y) := \exists z\; B(x, y, z)$ is true exactly for easy pairs (x, y).

Case 1: Existence of a Hard Pair that is Partly In and Out
Suppose there exists a hard pair $(x_{\text{hard}}, y_{\text{hard}})$ with $\chi_A(x_{\text{hard}}) \neq \chi_A(y_{\text{hard}})$, that is, $\chi_A^2(x_{\text{hard}}, y_{\text{hard}}) = 01$ or $\chi_A^2(x_{\text{hard}}, y_{\text{hard}}) = 10$. We only need to consider the case $\chi_A^2(x_{\text{hard}}, y_{\text{hard}}) = 01$ since the other case is symmetric. We can freely use x_{hard} and y_{hard} in formulæ in the following, because all singleton sets are elements of C by the third closure property of strongly closed classes.

I claim that $z \in A$ holds iff $011 \in C_3[x_{\text{hard}}, y_{\text{hard}}, z]$. To prove this, we show that there exists at most one bitstring in $P := C_3[x_{\text{hard}}, y_{\text{hard}}, z]$ that starts with 01. Suppose we had both $010 \in P$ and $011 \in P$. Then $000 \notin P$, since otherwise $\{b[2,3] \mid b \in P\} \supseteq \{10, 11, 00\}$, contradicting the assumption that one possibility has been excluded for $\#_A^2(y_{\text{hard}}, z)$. Likewise, $101 \notin P$ and also $111 \notin P$, since otherwise $\{b[1,3] \mid b \in P\} \supseteq \{00, 01, 11\}$.

Since $(x_{\text{hard}}, y_{\text{hard}})$ is a hard pair, we have either $\{b[1,2] \mid b \in P\} = \{00, 01, 10\}$ or $\{b[1,2] \mid b \in P\} = \{01, 10, 11\}$. In the first case, since $000 \notin P$ and $00 \in \{b[1,2] \mid b \in P\}$, we must have $001 \in P$. Likewise, since $101 \notin P$ and $10 \in \{b[1,2] \mid b \in P\}$, we must have $100 \in P$. But then $P \supseteq \{010, 011, 001, 100\}$ and thus $\{b[2,3] \mid b \in P\} \supseteq \{10, 11, 01, 00\}$, a contradiction. Similarly, in the second case we must have $100 \in P$ and $110 \in P$ and thus $P \supseteq \{010, 011, 100, 110\}$, which yields $\{b[2,3] \mid b \in P\} \supseteq \{10, 11, 00\}$, also a contradiction. This shows that P contains only one bitstring starting with 01.

Case 2: All Hard Pairs are Either In or Out
For this case, assume that $\chi_A(x_{\text{hard}}) = \chi_A(y_{\text{hard}})$ holds for every hard pair $(x_{\text{hard}}, y_{\text{hard}})$. The aim is to show $\chi_A^2 \in \mathrm{EN}_C(2)$, which implies the claim by theorem 3.29. The rough idea is as follows. On input of two elements x and y, we first check whether the pair (x, y) is hard, using the formula $\neg\phi_{\text{easy}}$. If so, by assumption we know that $\chi_A(x) = \chi_A(y)$ and we can output the pool $\{00, 11\}$. Otherwise the pair is easy. In this case we know that there exists an advisor z such that $\big\{b[1,2] \mid b \in C_3[x, y, z]\big\}$ has size at most 2. Once we have fixed such an advisor, we can output the set.

In detail, the construction is as follows. Let $B' \in C$ denote a single-valued refinement of the advisor relation B. The relation B' is the graph of a partial function that maps every easy pair (x, y) to an advisor for it and that is undefined for all hard pairs. Consider the following relation S:

$$\begin{aligned} b \in S[x,y] \;:\Longleftrightarrow\quad & \big(\neg\phi_{\text{easy}}(x,y) \to \mathbin{.} b = 00 \lor b = 11\big) \\ & \land \Big(\phi_{\text{easy}}(x,y) \to \exists z \mathbin{.} B'(x,y,z) \\ & \qquad \land \big(b0 \in C_3[x,y,z] \lor b1 \in C_3[x,y,z]\big)\Big). \end{aligned}$$

The first line ensures that S enumerates $\{00, 11\}$ if (x, y) is a hard pair. If it is easy, the second line first fixes z such that it is an advisor and then outputs all bitstrings in the set $C_3[x, y, z]$ with the last bit removed. Since (x, y) is easy, this set will have size at most 2. Thus $\chi^2_A \in \mathrm{EN}_C(2)$ via S. QED

4.17 Corollary
Let A be a language. If $\#^2_A \in \mathrm{EN}_{\mathrm{fa}}(2)$, then A is regular.

4.18 Corollary
Let $A \subseteq \mathbb{N}$. If $\#^2_A \in \mathrm{EN}_{\mathrm{Pa}}(2)$, then A is definable in Presburger arithmetic.

Note that we do *not* obtain the corollary 'if $\#^2_A \in \mathrm{EN}_{\mathrm{re}}(2)$, then A is recursive'. There are two reasons for this. First, the theorem would only claim that A is recursively enumerable, not that it is recursive. But this problem is easily taken care of, since $\#^2_A \in \mathrm{EN}_{\mathrm{re}}(2)$ iff $\#^2_{\bar{A}} \in \mathrm{EN}_{\mathrm{re}}(2)$. The second reason is more profound: the class of recursively enumerable relations is not closed under universal quantification, but such a quantification is used for the definition of the relation S. The statement 'if $\#^2_A \in \mathrm{EN}_{\mathrm{re}}(2)$, then A is recursive' is nevertheless true by fact 4.1, whose proof is quite different from the proof of theorem 4.16.

We do not obtain the corollary 'if $\#^2_A \in \mathrm{EN}_{\mathrm{On}}(2)$, then A is definable in ordinal number arithmetic' either. This time, the reason is that the class of relations definable in ordinal number arithmetic does not contain all singletons, see example 3.28. The following theorem shows that, in contrast to the recursive setting, this claim also cannot be proved by other means.

4.19 Theorem
There is a set $A \subseteq \mathrm{On}$ that is not elementarily definable in ordinal number arithmetic, but for which $\#^2_A \in \mathrm{EN}_{\mathrm{On}}(2)$ (even via a relation that never enumerates both 0 and 2).

Proof. Let $A := \{\alpha\}$ such that $\alpha \in \mathrm{On}$ is not definable in ordinal number arithmetic. Such an ordinal exists, as argued in example 3.28. We have $\#^2_A \in \mathrm{EN}_{\mathrm{On}}(2)$ via the relation $\mathrm{On} \times \mathrm{On} \times \{0, 1\}$, which is elementarily definable in ordinal number arithmetic and never enumerates 2. QED

Section 4.3

The Generic Restricted Cardinality Theorem

In this section I prove that the restricted cardinality theorem holds for finite automata. A central idea of the proof, namely the use of a tuple $(y_1, \dots, y_n)$ in the definition of easy tuples, is due to Austinat et al. (2000).

4.20 Theorem
Let n be a positive integer, let C be strongly closed, and let $A \subseteq U$ be a set. If $\#_A^n \in \mathrm{EN}_C(n)$ via a relation R for which $R[x_1, \dots, x_n]$ never contains both 0 and n for any $x_1, \dots, x_n \in U$, then $A \in C$.

Proof. We prove the claim by induction on n. For $n = 1$ the claim is true. So suppose the claim has already been shown for $n - 1$.

Let $\#_A^n \in \mathrm{EN}_C(n)$ via a relation R such that $R[x_1, \dots, x_n]$ never contains both 0 and n for any $x_i \in U$. As in the previous proofs, we define easy elements, based on a notion of advisors. Let us call a tuple $(y_1, \dots, y_n) \in U^n$ an *advisor* for a tuple $(x_1, \dots, x_{n-1}) \in U^{n-1}$, if it satisfies the following relation:

$$\begin{aligned}(x_1, \dots, x_{n-1}, y_1, \dots, y_n) \in B \;:\Longleftrightarrow\; & \\ & \mathrm{distinct}(x_1, \dots, x_{n-1}, y_1, \dots, y_n) \\ & \wedge 0 \in R[y_1, \dots, y_n] \wedge \textstyle\bigwedge_{i=1}^{n} n \in R[x_1, \dots, x_{n-1}, y_i].\end{aligned}$$

Note that an advisor tuple can only, but need not, exist if at least one x_i is in A. Let us call a tuple $(x_1, \dots, x_{n-1})$ of pairwise different elements *easy* if

1. at least one x_i is not in A or
2. there exists an advisor for it.

A tuple $(x_1, \dots, x_{n-1})$ of pairwise different elements is *hard* if it is not easy.

Case 1: Existence of a Hard Tuple
Suppose that there exists a hard tuple $(x_1^{\mathrm{hard}}, \dots, x_{n-1}^{\mathrm{hard}})$. Since the class C contains all singletons, we can freely use the x_i^{hard} in formulæ in the following. Let

$$y \in \hat{A} \;:\Longleftrightarrow\; n \in R\big[x_1^{\mathrm{hard}}, \dots, x_{n-1}^{\mathrm{hard}}, y\big] \vee \textstyle\bigvee_{i=1}^{n-1} y = x_i^{\mathrm{hard}}.$$

I claim $\hat{A} =_{\mathrm{ae}} A$. This means that A and $\hat{A}$ are equal almost everywhere, that is, that their symmetric difference is finite. This will prove $A \in C$.

Since condition 1 does not hold for hard tuples, all x_i^{hard} are in A. For $y \in A \setminus \{x_1^{\mathrm{hard}}, \dots, x_{n-1}^{\mathrm{hard}}\}$ we thus have $\#_A^n(x_1, \dots, x_{n-1}, y) = n$, which implies $n \in R[x_1^{\mathrm{hard}}, \dots, x_{n-1}^{\mathrm{hard}}, y]$. Thus for all $y \in A$ we have $y \in \hat{A}$.

For $y \notin A$, we can have $n \in R[x_1^{\mathrm{hard}}, \dots, x_{n-1}^{\mathrm{hard}}, y]$ for at most $n - 1$ different y's, since any such y's would form an advisor for $(x_1^{\mathrm{hard}}, \dots, x_{n-1}^{\mathrm{hard}})$, contradicting the assumption that condition 2 does not hold. Thus $y \notin \hat{A}$ whenever $y \notin A$, except for these finitely many exceptions.

Case 2: All Tuples are Easy
Suppose all tuples of pairwise different elements are easy. We argue that

$\#_A^{n-1} \in \mathrm{EN}_C(n-1)$ via a relation S for which $S[x_1, \dots, x_{n-1}]$ never contains both 0 and $n-1$ for any x_i. This yields the claim by the induction hypothesis. For the definition of S, first consider the following relation $\tilde{S}$, which 'works' only for distinct x_i:

$$\begin{aligned} &k \in \tilde{S}[x_1, \dots, x_{n-1}] \ :\Longleftrightarrow \\ &\qquad \bigl[\bigl(\ \exists y_1 \cdots \exists y_n \ B(x_1, \dots, x_{n-1}, y_1, \dots, y_n)\bigr) \to \textstyle\bigvee_{i=1}^{n-1} k = i\bigr] \\ &\quad \wedge \bigl[\bigl(\neg\exists y_1 \cdots \exists y_n \ B(x_1, \dots, x_{n-1}, y_1, \dots, y_n)\bigr) \to \textstyle\bigvee_{i=0}^{n-2} k = i\bigr]. \end{aligned}$$

For distinct x_i, if there exists an advisor tuple for $(x_1, \dots, x_{n-1})$, the very existence of the advisor tuple ensures that for at least one x_i we have $x_i \in A$. Thus $\#_A^{n-1}(x_1, \dots, x_{n-1}) > 0$. If there does not exist an advisor tuple, which can only happen if condition 1 holds, at least one x_i is not in A. Thus $\#_A^{n-1}(x_1, \dots, x_{n-1}) < n-1$.

The desired relation S that works for all x_i, not just for distinct x_i, can be obtained from $\tilde{S}$ as follows:

$$\begin{aligned} &k \in S[x_1, \dots, x_{n-1}] \ :\Longleftrightarrow \\ &\qquad \bigl(\ \ \mathrm{distinct}(x_1, \dots, x_{n-1}) \to k \in \tilde{S}[x_1, \dots, x_{n-1}]\bigr) \\ &\quad \wedge \bigl(\neg \, \mathrm{distinct}(x_1, \dots, x_{n-1}) \to \textstyle\bigvee_{i=0}^{n-2} k = i\bigr). \end{aligned}$$

QED

4.21 Corollary
Let n be a positive integer and A a language. If $\#_A^n \in \mathrm{EN}_{\mathrm{fa}}(n)$ via an automaton that never enumerates both 0 and n, then A is regular.

4.22 Corollary
Let n be a positive integer and $A \subseteq \mathbb{N}$. If $\#_A^n \in \mathrm{EN}_{\mathrm{Pa}}(n)$ via a relation that never enumerates both 0 and n, then A is definable in Presburger arithmetic.

Section 4.4

Constructiveness of the Generic Theorems

This section addresses the question of whether the results of the previous sections can be proved in a constructive way. As we shall see, this depends on the computational model: for Turing machines the answer is negative, for finite automata it is positive (at least for a fair version of the question).

Both the nonspeedup theorem and the cardinality theorem for two words rely on the cross product theorem. In order to investigate whether these theorems can be proved constructively, let us revisit the proof of the cross product theorem, paying close attention to its constructiveness. The theorem tells us that if $f \times g$ is $(n+m)$-enumerable, then there exists an n-enumerator for f or there exists an m-enumerator for g. The proof provides us with a construction of these enumerators, starting with the enumerator for $f \times g$ and using the closure properties of the class C.

Unfortunately, the closure properties themselves are not constructive for all computational models. For the class of recursively enumerable languages the third closure property of weakly closed classes, see definition 3.21, is highly nonconstructive: the closure property amounts to the statement 'every singleton set in the arithmetical hierarchy is recursively enumerable'. This statement is certainly true since *every* singleton set is trivially recursively enumerable, but there is no way of computing the element of the singleton set from the code of a machine that witnesses the singleton's membership in the arithmetical hierarchy. When the cross product theorem for Turing machines is proved directly, as done in (Tantau, 2001), the 'hard elements' u_{hard} appear magically and must be hardwired into machines.

A conceptually different source of nonconstructiveness is the value of $f(u_{\text{hard}})$. We know that it is among a set of at most $n+m$ possibilities, but we cannot 'construct' the correct one. Instead, we must hardwire the index of the correct choice into the formulæ.

These sources of nonconstructiveness are not just peculiarities of my proof. For Turing enumerability, the work of Beigel et al. (1993) shows that it is an integral part of the cross product theorem: they show that every proof of the nonspeedup theorem, which is a direct corollary of the cross product theorem, must be nonconstructive. More precisely, there is no algorithm that gets as input (the code of) a Turing machine witnessing $A \in \mathrm{V}_{\text{re}}(n,n)$ and yields as output (the code of) a Turing machine deciding A.

To be fair, a machine M that witnesses $A \in \mathrm{V}_{\text{re}}(n,n)$ typically also witnesses $B \in \mathrm{V}_{\text{re}}(n,n)$ for different languages B. It is hence impossible for any algorithm to 'output' exactly A on input M. A fair version of this construction problem, which is called 'search problem' by Kaufmann and Kummer (1996), is formulated next.

4.23 Definition of Fair Construction Problems for Turing Machines
A Turing machine *solves the construction problem for the nonspeedup theorem*, respectively *for the restricted cardinality theorem*, if it has the following properties:

1. As input, it gets the code of a machine M_{witness} that witnesses $\chi_A^n \in \mathrm{EN}_{\text{re}}(n)$ for some language A, respectively $\#_A^n \in \mathrm{EN}_{\text{re}}(n)$ such that 0 and n are never enumerated both.
2. As output, it yields the code of a machine M.

3. The Turing machine M_{witness} witnesses $\chi^n_{\mathrm{L}(M)} \in \mathrm{EN}_{\text{re}}(n)$, respectively $\#^n_{\mathrm{L}(M)} \in \mathrm{EN}_{\text{re}}(n)$ such that 0 and n are never enumerated both.

Even the fair version of the construction problem cannot be solved. That is, no Turing machine M solves the construction problem for the nonspeedup theorem or for the restricted cardinality theorem. In a detailed study, Kaufmann and Kummer (1996) were able to 'quantify' the 'degree of nonconstructiveness'. The following definition and theorem explain what is meant by this.

4.24 Definition of the Weak Construction Problem
Let k be a positive integer. A Turing machine *solves the weak k-construction problem for the nonspeedup theorem*, respectively *for the restricted cardinality theorem*, if it has the following properties:

1. As input, it gets the code of a machine M_{witness} that witnesses $\chi^n_A \in \mathrm{EN}_{\text{re}}(n)$ for some language A, respectively $\#^n_A \in \mathrm{EN}_{\text{re}}(n)$ such that 0 and n are never enumerated both.
2. As output, it yields the codes of k machines $M_1, \dots, M_k$.
3. For at least one machine M_i we have $\mathrm{L}(M_i) =_{\text{ae}} B$ for some language B for which M_{witness} witnesses $\chi^n_B \in \mathrm{EN}_{\text{re}}(n)$, respectively $\#^n_B \in \mathrm{EN}_{\text{re}}(n)$ such that 0 and n are never enumerated both.

A solver for the weak construction problem yields only somewhat crude constructive approximations of the language A. The following facts show that even these crude approximations are hard to come by.

4.25 Fact (Kaufmann and Kummer, 1996)
The weak k-construction problem for the nonspeedup theorem is solvable exactly for $k \geq 2n-1$.

4.26 Fact (Kaufmann and Kummer, 1996)
The weak k-construction problem for the restricted cardinality theorem is solvable exactly for $k \geq (n+1)/2$.

The situation is quite different for finite automata. Since the class of regular relations is strongly closed in a uniform way, the first source of nonuniformity in the cross product theorem disappears for finite automata: the set of hard words is regular and we can nicely refer to all its elements. We can even effectively check whether hard words exist at all, by checking whether a language specified by a finite automaton is empty or not. The second source of nonuniformity, namely the unknown value of $f(u_{\text{hard}})$, is a persisting problem. Before we investigate how this problem can be dealt with, let us define the problem we wish to solve.

4.27 Definition of Fair Construction Problems for Finite Automata
A Turing machine *solves the finite automata construction problem for the nonspeedup theorem*, respectively *for the cardinality theorem for two words*, respectively *for the restricted cardinality theorem*, if it has the following properties:

1. As input, it gets the code of a DFA M_{witness} that witnesses $\chi_A^n \in \mathrm{EN}_{\text{fa}}(n)$ for some language A, respectively $\#_A^2 \in \mathrm{EN}_{\text{fa}}(2)$, respectively $\#_A^n \in \mathrm{EN}_{\text{fa}}(n)$ such that 0 and n are never enumerated both.
2. As output, it yields the code of a DFA M.
3. The DFA M_{witness} witnesses $\chi_{\mathrm{L}(M)}^n \in \mathrm{EN}_{\text{fa}}(n)$, respectively $\#_{\mathrm{L}(M)}^2 \in \mathrm{EN}_{\text{fa}}(2)$, respectively $\#_{\mathrm{L}(M)}^n \in \mathrm{EN}_{\text{fa}}(n)$ such that 0 and n are never enumerated both.

4.28 Theorem
There exists a Turing machine that solves the finite automata construction problem for the nonspeedup theorem.

Proof. In case $n = 1$, the claim is trivial and we are done. So assume $n > 1$. Let A be a language for which M_{witness} witnesses $A \in \mathrm{V}_{\text{fa}}(n, n)$.

Consider the proof of the cross product theorem for $f = \chi_A^{n-1}$ and $g = \chi_A$. In the proof, two cases are distinguished. For the second case (all word tuples are easy), the DFA M_{witness} is *constructively* turned into a DFA M'_{witness} that witnesses $A \in \mathrm{V}_{\text{fa}}(n-1, n-1)$ and we can repeat the argument. Note that checking whether all word tuples are easy amounts to checking whether a relation specified by a finite automaton is $(\Sigma^*)^{n-1}$. For the first case (a hard word tuple exists), the machine M_{witness} is constructively turned into an automaton that accepts A, *provided* the characteristic string of a certain word tuple is known. Although we do not know this value, we can 'try' all 2^{n-1} possible values. Each value yields a candidate for a DFA that accepts A and (at least) one of them will be correct. For each candidate we can check (effectively) whether M_{witness} witnesses membership in $\mathrm{V}_{\text{fa}}(n, n)$ for them and output the DFA for which this is the case. QED

For the solution of the other two construction problems, the following lemma is helpful.

4.29 Lemma
Let n and k be positive integers. There is a Turing machine that works as follows: It gets as input the codes of two DFA's that accept a $(k+1)$-ary relation R and an $(n+1)$-ary relation R_{witness}. If there exists a word tuple $(x_1, \dots, x_k)$ for which R_{witness} witnesses $\#_{R[x_1,\dots,x_k]}^n \in \mathrm{EN}_{\text{fa}}(n)$, the machine yields such a tuple as output. If no such tuple exists, a special output is produced.

Proof. Using the formula $\forall z_1 \cdots \forall z_n \neg \exists i_1 \cdots \exists i_{n+1} \mathbin{.} \text{distinct}(i_1, \dots, i_{n+1}) \wedge \bigwedge_{j=1}^{n+1} R_{\text{witness}}(z_1, \dots, z_n, i_j)$ we first check whether R_{witness} is n-bounded. If this is the case, we consider the following relation:

$$\begin{aligned}&(x_1, \dots, x_k) \in S :\Longleftrightarrow \\ &\quad \forall z_1 \cdots \forall z_n \; \#^n_{R[x_1,\dots,x_k]}(z_1, \dots, z_n) \in R_{\text{witness}}[z_1, \dots, z_n].\end{aligned}$$

The term '$\#^n_{R[x_1,\dots,x_k]}(z_1, \dots, z_n)$' is not legal, but it could easily be replaced by a more verbose legal version. The relation S is true for a tuple $(x_1, \dots, x_k)$ if the language $R[x_1, \dots, x_k]$ is 'consistent' with the witness relation R_{witness}. Thus the smallest tuple for which this relation is true is the sought tuple. This tuple can be obtained constructively, since its definition is based on the closure properties of the class of regular relations. Checking whether the set is empty can also be done effectively. QED

4.30 Theorem
There exists a Turing machine that solves the finite automata construction problem for the cardinality theorem for two words.

Proof. As in the proof of the previous theorem, most steps of the proof of the finite automata cardinality theorem for two words are constructive. The only exception is the beginning of the first case: a hard pair whose components have differing characteristic values appears 'magically' and there is no way to avoid this. However, we can invoke the above lemma for the relation $(x, y, z) \in R :\Longleftrightarrow 011 \in C_3[x, y, z]$. If the first case of theorem 4.16 is the 'right' case, for an appropriate pair (x, y) the language $R[x, y]$ will be consistent with the witness machine. By lemma 4.29, we can obtain such a pair constructively. If the second case is 'right', the proof of theorem 4.16 invokes the finite automata nonspeedup theorem. Since the fair construction problem for this theorem can also be solved by theorem 4.28, we get the claim. QED

4.31 Theorem
There exists a Turing machine that solves the finite automata construction problem for the restricted cardinality theorem.

Proof. Once more, most steps of the proof of theorem 4.20 are constructive. This time the nonconstructive element in the proof is the fact that the constructed language $\hat{A}$ might differ on up to $n-1$ positions from A. As in the previous proof, we get around this nonconstructive part by defining all languages that differ from $\hat{A}$ in at most $n-1$ places in a parametric way and then invoke lemma 4.29 to find the 'right' $n-1$ places. QED

There exists a much simpler algorithm that solves the three construction problems discussed above: on input of a witnessing automaton M_{witness}, systematically check for *all* DFA's whether they accept a language for which M_{witness} is a witnessing automaton. If we find such an automaton, we output it. This straight-forward construction method has two severe drawbacks: first, if the automaton M_{witness} is not a witnessing machine after all, this will not be noticed and the algorithm loops endlessly; second, the brute-force search is computationally more expensive than the transformation algorithms used in the theorems.

Fifth Chapter

The Branch Diagonalisation Method

This chapter is a tutorial to a new diagonalisation method, which I call 'branch diagonalisation'. Unlike other diagonalisation techniques it is not only applicable to Turing machines, but also to finite automata. The method is not universally applicable; for example one of the involved classes must be uncountable. But when it is applicable it yields extremely strong separations. An example is the separation of verboseness classes: using a finite injury argument, Beigel et al. (1995b) were able to show that $\mathrm{V}_{\mathrm{re}}(m,n)$ is not contained in $\mathrm{V}_{\mathrm{re}}(h,k)$ for certain numbers m, n, h, and k; using branch diagonalisation, I show that for the same numbers the much smaller class $\mathrm{V}_{\mathrm{fa}}(m,n)$ is not even contained in the much bigger class $\mathrm{V}_{\mathrm{re}}^{X}(h,k)$ for any oracle X. The class $\mathrm{V}_{\mathrm{re}}^{X}(h,k)$ contains all languages that are (m,n)-verbose via a Turing machine that has oracle access to X.

The first use of diagonalisation dates back to Georg Cantor's famous proof (Cantor, 1874) that the continuum is 'larger' than the set of natural numbers. Diagonalisation was first used in computer science by Alan Turing (1936) at a time when computer science did not even exist as a discipline. Since then, diagonalisation has evolved and is now used extensively both in recursion theory, see the tenth chapter of (Odifreddi, 1999) for an overview, and in complexity theory, see the recent survey article by Fortnow (2000) for the current state of the art.

All diagonalisation methods, including branch diagonalisation, follow the same pattern: We start with a countable set of, say, Turing machines that witness that a language has a certain property. For example, we might start with the set of (clocked) polynomially time-bounded Turing machines that witness that a language is in the class P. We then construct a language that does not have this property by systematically tricking all machines. This is ensured by defining the characteristic values of some appropriate words in such a way that the first machine cannot witness that the language has the property. Then, for some other words, we ensure that the second machine is tricked, and so on. The words for which we trick the machines will be called *diagonalisation points.* By carefully choosing the diagonalisation points we can ensure that the resulting language still has certain desirable properties, like being decidable in exponential time. Diagonalisation methods differ mainly in how the diagonalisation points are chosen.

The simplest way of choosing them is to trick each machine on its own binary encoding. For example, let us construct a language L that is not recursively enumerable: Take the first Turing machine M_1 that could prove this. In order to ensure that M_1 does not accept L, we study the behaviour of M_1 on its own binary encoding $\operatorname{bin} M_1$. If M_1 halts (and, for the purposes of this example, accepts by definition) on input bin M_1 we do not put bin M_1 into L, otherwise we do. In the same way we trick M_2 on input bin M_2, and so on. The language L constructed in this way is surely not recursively enumerable. Note that $L = \{\text{bin}\, M \mid M \text{ does not halt on input } \text{bin}\, M\}$ is exactly the complement of the halting problem. Thus we have recovered Turing's result that the halting problem is not co-recursively enumerable. This choice of diagonalisation points, namely machine codes (possibly augmented by an input or a number of steps), is also used in numerous proofs in complexity theory: the space and time hierarchy theorems use this method.

More advanced diagonalisation arguments use a more complicated approach. For the super-sparse set technique, first used by Kurtz (1983) according to Hemaspaandra and Jiang (1995), we do not directly interpret words as codes of machines, but rather the iterated logarithm of their length. This causes the diagonalisation points to be spaced extremely far apart: the ith machine is tricked on words of length $\text{tow}(i)$, where $\text{tow}(0) = 1$ and $\text{tow}(n+1) = 2^{\text{tow}(n)}$.

The most important diagonalisation techniques in recursion theory are finite and infinite injury methods. These methods have in common that diagonalisation points 'move around'. 'Urgent problems' during a later stage of the diagonalisation process can make it necessary to reassign a diagonalisation point.

All these methods have one thing in common: they are not applicable to finite automata. Finite automata lack the ability to decode the code of another finite automaton, let alone the ability to keep track of the reassignments performed during a finite injury argument. The branch diagonalisation method was born out of a need to diagonalise using finite automata. This method chooses the diagonalisation points in such a way that even a finite automaton can 'work with them'.

We start with a finite set Q of 'diagonalisation choices' or 'diagonalisation actions'. Each element of Q represents a possible action taken in a diagonalisation stage. For example, for the halting problem diagonalisation above, $Q = \{d_0, d_1\}$ where d_0 is the decision 'leave the word out, since the machine halts' and d_1 is the decision 'put it in, since the machine does not halt'. Consider the tree Q^*, whose predecessor relation is the prefix relation. Each branch of this tree represents an infinite sequence of diagonalisation decisions. During a diagonalisation such a branch is incrementally constructed: the way we trick the first machine determines the first node of the branch, the way we trick the second machine determines the second node, and so on. For example, for the set Q for the halting problem the diagonalisation might result in a branch of Q^* like $\{\epsilon, d_0, d_0d_0, d_0d_0d_1, d_0d_0d_1d_0, \dots\}$.

The key idea of the branch diagonalisation method is to *use a binary en-*

coding of the nodes of the diagonalisation branch as diagonalisation points. For example, if we encode d_0 by 0 and d_1 by 1, for the above branch we diagonalise against the first Turing machine on the word ϵ. We diagonalise against the second machine on the word 0, against the third machine on 00, against the fourth on 001, and so on. In more complicated settings we may need more than one word for each diagonalisation stage, say k many. We obtain these words be appending k different tags to the binary encoding of the current diagonalisation node.

In a branch diagonalisation every diagonalisation point encodes the whole previous diagonalisation process. Given two words that encode two different diagonalisation sequences, even a finite automaton can compute up to what point the diagonalisation sequences agree and it can compute what 'happened' when the sequences split. In certain situations this information suffices for showing that the diagonalisation language has a certain property, like being (m, n)-fa-verbose.

Branch diagonalisation, which was called 'structural diagonalisation' in (Tantau, 2001), has been used before. In a technical report, Kummer and Stephan (1991) introduced *k-branches*, which are a special case of the branches considered in this dissertation. They use k-branches in an ad hoc fashion for a separation of frequency classes, but do not use or propose their diagonalisation as a general method. In the tenth chapter of the book of Odifreddi (1999), a special finite injury argument called 'tree diagonalisation' is introduced. This method has in common with branch diagonalisation that diagonalisation choices drive the construction of a branch in a tree of diagonalisation choices. Odifreddi traces the roots of tree diagonalisation back to an article of Lachlan (1975). Both k-branches and tree diagonalisation are defined in such a way in the literature that they apply only to Turing machines. As I shall demonstrate, branch diagonalisation gives especially powerful results when used in conjunction with finite automata.

Section 5.1, entitled 'The Art of Branch Diagonalisation', introduces the branch diagonalisation method. First, a simple example is presented. Then the essential ideas are extracted from the proof. This leads to a formal framework for branch diagonalisation, which is developed in the course of the section. The remaining sections present theorems whose proofs employ a branch diagonalisation.

In section 5.2 a beautiful theorem is proved: the separation theorem for verboseness classes. A consequence of this theorem is that the inclusion structures of finite automata and Turing machine verboseness classes coincide: $\mathrm{V}_{\mathrm{fa}}(m, n) \subseteq \mathrm{V}_{\mathrm{fa}}(h, k)$ iff $\mathrm{V}_{\mathrm{re}}(m, n) \subseteq \mathrm{V}_{\mathrm{re}}(h, k)$.

Section 5.3 studies relations that are separable or inseparable by regular relations. Separation by regular relations is defined analogously to separation by recursive or polynomial-time computable sets: two sets A and B are *fa-separable* if there exists a regular set C with $A \subseteq C \subseteq \bar{B}$. Using branch diagonalisation, I prove that two languages A and B can be disjoint and recursively inseparable, while the closely related relations $A^{(2)}$ and $B^{(2)}$ are fa-separable. A surprising result of this section is theorem 5.24, which provides a counterexample to a theorem of Kinber (1976).

In section 5.4 branch diagonalisation is used to separate reduction closures of selectivity classes. In the recursive setting these reduction closures play a key rôle in the solution of Post's problem, in the polynomial-time setting they have applications in the study of problems having small circuits. A strong separation is shown for the bounded queries reduction closures of selective languages: for every k, there exists an fa-selective language whose parallel $(k+1)$-queries equivalence closure is not contained in the parallel k-queries reduction closure of *any semirecursive language.* Apart from a branch diagonalisation, the proof of the separation uses a combinatorial result on walks on hypercubes. Although the obtained results are stronger than some previously known results, they are not quite as strong as the results presented in (Beigel et al., 2000) and (Tantau, 2000). I have not included proofs of these stronger results, which are proved differently, since this chapter focusses on 'branch diagonalisation in action'.

Section 5.1

The Art of Branch Diagonalisation

In this section branch diagonalisation is introduced and formalised. A theorem is presented whose proof uses a simple branch diagonalisation. It states that the intersection of P-selective languages need not be semirecursive, which improves an earlier result due to Hemaspaandra and Jiang (1995). The proof is reviewed and the 'essence' of the ideas is extracted. Building on the analysis, I develop a formal framework for branch diagonalisation.

Although a general 'branch diagonalisation theorem' is formulated at the end of this section, see theorem 5.16, branch diagonalisation is still a *method.* It cannot be completely described by a single theorem. In this respect it resembles other methods like, say, induction. In many textbooks an 'induction theorem' is formulated that states, for example, that if a set S of natural numbers contains 0 and contains together with each number n also $n+1$, then S is the set of all natural numbers. However, such theorems are rarely applied directly and the 'induction method' often needs to be adapted to specific situations.

We start with a simple proof that uses a branch diagonalisation. The theorem refers to P-selective and semirecursive languages. These notions are due to Selman (1979) and Jockusch (1966) respectively.

5.1 Definition of Selective Languages
A *selector* for a language $A \subseteq \Sigma^*$ is a function $f\colon \Sigma^* \times \Sigma^* \to \Sigma^*$ such that $f(u,v) \in \{u,v\}$ for all words $u,v \in \Sigma^*$ and such that $f(u,v) \in A$ whenever $u \in A$ or $v \in A$. A language is P-*selective* if it has a selector that is computable in polynomial time, it is *semirecursive* if it has a recursive selector, and it is *fa-selective* if it has a regular selector.

Hardly surprisingly, branch diagonalisation proofs refer to *trees* and *branches*. For our purposes, they can be defined as follows. Note that branches are required to be infinite.

5.2 Definition of Trees and Branches
A *tree* is a language that is closed under prefix. Its alphabet is called the *tree alphabet*. The elements of a tree are called *nodes*. The empty word is the *root node*. A node u is a *descendant* of a node v if v is a proper prefix of u. A descendant is a *successor* of a node if it is exactly one symbol longer. A *branch* of a tree T is an infinite set $\{u_1, u_2, u_3, \ldots\} \subseteq T$ such that u_1 is the root node and each u_i is a successor of u_{i-1}.

5.3 Theorem
There exist two P*-selective sets whose intersection is not semirecursive.*

Proof. I present this proof in more detail than customary in order to make its analysis easier later on. For a crisp presentation of this proof see theorem 5.5 of the book of Hemaspaandra and Torenvliet (2002).

Preparation
Our aim is to construct a language A that is not semirecursive. The special way we do this will ensure that A is the intersection of two P-selective languages B and C.

Let M_1, M_2, M_3, ... be an enumeration of all Turing machines that compute recursive selector functions. Note that this enumeration need not be effective (indeed, it cannot be effective). For each machine M_i, let f_i denote the selector function computed by M_i and let D_i denote the class of all languages for which f_i is a selector. Thus $L \in D_i$ if for all words $u,v \in \Sigma^*$ with $u \in L$ or $v \in L$ we have $f_i(u,v) \in L$. Since the sets D_i cover the class of semirecursive languages, for the construction of the diagonalisation language A it suffices to ensure that A is not an element of any D_i.

Construction of the Diagonalisation Branch

A requirement like '$A \notin D_i$' can be satisfied as follows: given any word u, consider the 'behaviour' of f_i on the words $u0$ and $u1$. Either $f_i(u0, u1) = u0$ or $f_i(u0, u1) = u1$. In the first case, all languages $L \in D_i$ have the property that $u1 \in L$ enforces $u0 \in L$. Setting $\chi^2_A(u0, u1) := 01$ ensures $A \notin D_i$. In the second case, $A \notin D_i$ is ensured by setting $\chi^2_A(u0, u1) := 10$.

The two cases correspond to two possible 'diagonalisation decisions', which will be called d_1 and d_2. The decision d_1 means 'put $u1$ into A and leave out $u0$'; whereas d_2 means 'put $u0$ into A and leave out $u1$'. Let $Q := \{d_1, d_2\}$ and let us encode d_1 by the symbol 1 and d_2 by the symbol 0. For a node $u \in Q^*$, let $\operatorname{bin} u$ denote its binary encoding; for example $\operatorname{bin} d_1 d_2 d_1 = 101$.

Recall that the key idea of the branch diagonalisation method is to trick each machine on the code of the previous diagonalisation decisions. For the present proof this means the following: The first diagonalisation point u_1 is the empty node $\epsilon \in Q^*$ (no decisions have been made, yet) and we trick M_1 on the words $\operatorname{bin} u_1\, 0 = 0$ and $\operatorname{bin} u_1\, 1 = 1$. If $f_1(0, 1) = 0$, the first diagonalisation decision is d_1, otherwise it is d_2. We set $\chi^2_A(0, 1) = 01$ or $\chi^2_A(1, 0) = 10$ accordingly. For a node $u \in Q^*$, let us call the words $\operatorname{bin} u\, 0$ and $\operatorname{bin} u\, 1$ *associated* with u.

The second diagonalisation node u_2 is either $u_1 d_1$ or $u_1 d_2$, depending on the first diagonalisation decision. We trick M_2 on the words $\operatorname{bin} u_2\, 0$ and $\operatorname{bin} u_2\, 1$ that are associated with u_2. This results in a specific value of $\chi^2_A(\operatorname{bin} u_2\, 0, \operatorname{bin} u_2\, 1)$, a second diagonalisation decision, and a new node u_3. We trick M_3 on the words associated with u_3, which yields a third diagonalisation decision and a node u_4; and so on.

As we trick all machines, a branch $\{u_1, u_2, u_3, \dots\} \subseteq Q^*$ is constructed. The characteristic values of the words associated with the nodes on this branch are fixed during the diagonalisation process. For all other words, no matter how we choose their characteristic values, the language A will not be an element of any D_i. Let us put no other words into A, except for the empty word (this is for æsthetic reasons, see below).

The Diagonalised Set is the Intersection of P-Selective Sets

It remains to show that the language A is the intersection of two P-selective languages. Before we construct them, let us first scrutinise A's structure.

I claim that the language A itself forms a branch in the full binary tree $\{0, 1\}^*$. First, the root node is an element of A by definition. Second, for any node u_i of the diagonalisation branch either the bitstring $\operatorname{bin} u_i\, 0$ or the bitstring $\operatorname{bin} u_i\, 1$ is an element of A. By definition of u_{i+1}, these nodes 'happen' to be $\operatorname{bin} u_{i+1}$. Thus A is exactly the set $\{\operatorname{bin} u_1, \operatorname{bin} u_2, \operatorname{bin} u_3, \dots\}$. (This is a peculiarity of this particular proof. In other branch diagonalisations the language A is more complicated.)

For showing that A is the intersection of two P-selective languages, the only relevant property of A is its being a branch. It will not be important how this branch is formed exactly or how many times it veers to the left or right. Rather, *every* branch $\{x_0, x_1, x_2, \ldots\} \subseteq \{0,1\}^*$ is the intersection of two P-selective languages: let

$$B := \bigl\{w \in \{0,1\}^* \mid w \leq_{\text{lex}} x_{|w|}\bigr\} \text{ and}$$
$$C := \bigl\{w \in \{0,1\}^* \mid w \geq_{\text{lex}} x_{|w|}\bigr\}.$$

These languages are P-selective via the following two selector functions: f_B maps a pair (w_1, w_2) to the lexicographically smaller one of the two words, f_C maps the pair to the larger one. These functions are even regular, and thus in particular polynomial-time computable. Clearly, $A = B \cap C$. QED

5.4 Corollary (Hemaspaandra and Jiang, 1995)
The class of P*-selective languages is not closed under intersection.*

The branch diagonalisation proof of theorem 5.3 gives a stronger result than the original proof of Hemaspaandra and Jiang (1995) of corollary 5.4. Their proof uses a super-sparse set diagonalisation. Although it can be used to show that for arbitrary recursive functions f there exist two P-selective languages whose intersection does not have a selector that is computable in time f, it cannot be used to establish the claim of theorem 5.3. The reason is, roughly spoken, that their diagonalisation involves a *simulation* of selectors for earlier stages of the diagonalisation. The proof of theorem 5.3 avoids such a simulation and yields a much stronger result. It can be recycled to show an even stronger statement:

5.5 Theorem
For every oracle X, there exist two fa-selective languages whose intersection does not have a selector that is recursive in X.

A Formalisation of the Branch Diagonalisation Method

In the remainder of this section the branch diagonalisation method is formalised. The aim is to formulate a 'branch diagonalisation theorem', see theorem 5.16, that encapsulates the core idea of the method. The framework is developed by analysing the proof of theorem 5.3.

The first step of the proof was the introduction of machines M_i that served as possible witnesses for showing that the diagonalisation language A is semirecursive. The exact details of these machines turned out to be irrelevant. It was only important that the language A is not an element of any of the classes D_i. The class D_i contained all languages L for which the selector computed by M_i witnesses that L is semirecursive.

The first building block of the framework is an abstraction of the first proof step. It provides an abstract way of talking about sets of languages for which devices (like Turing machines) witness membership in a class of languages (like the class of semirecursive languages).

5.6 Definition of Countable Coverings
Let C be a class of languages. A *countable covering of* C is a countable set of subclasses of C that cover C.

In other words, a countable covering V of C has the property that for each language $L \in C$ there exists a class $D \in V$ with $L \in D$. For example, the set $\{D_1, D_2, D_3, \ldots\}$ from the proof of theorem 5.3 forms a countable covering of the class of semirecursive languages.

5.7 Example: Countable Language Classes
Let C be any countable class of languages. Then $V := \big\{\{L\} \mid L \in C\big\}$ is a trivial countable covering of C.

The next example refers to *advice classes*, which are defined as follows.

5.8 Definition of Advice (Karp and Lipton, 1980)
Let $f\colon \mathbb{N} \to \mathbb{N}$ be a function, which will be called *advice bound.* The class P/f contains all languages A for which there exists a polynomially time-bounded Turing machine M and an *advice function* $h\colon \mathbb{N} \to \{0,1\}^*$, with $|h(n)| = f(n)$ for all n, such that $A = \big\{w \in \Sigma^* \mid \big\langle w, h(|w|)\big\rangle \in \mathrm{L}(M)\big\}$.

As is customary, if k is a constant, P/k denotes the class P/f with $f(n) = k$ for all n. The class P/poly contains all languages that are in P/f for some polynomial f.

5.9 Example: Advice Classes
For every constant k, a countable covering V of P/k that can be used in branch diagonalisations can be defined as follows: for each polynomially time-bounded machine M let V contain the class of all languages L for which there exists an advice function $h\colon \mathbb{N} \to \{0,1\}^k$ such that $L = \big\{w \in \Sigma^* \mid \big\langle w, h(|w|)\big\rangle \in \mathrm{L}(M)\big\}$.

At first sight, the definition of countable coverings is too broad to be useful for an abstract notion of diagonalisation. Indeed, *every* class C has a countable covering: just take $V = \{C\}$, which corresponds to the rather silly machine model of a single device that witnesses membership in C for every language $L \in C$. In order to use countable coverings in a branch diagonalisation we have to restrict ourselves to coverings that have a special property: the property of being Q*-diagonalisable*, which is introduced next.

The second step in the proof of theorem 5.3 was the construction of the language A. We systematically ensured that $A \notin D_i$ holds for all

$i \in \{1, 2, 3, \ldots\}$. To achieve this, for each i, two words $\operatorname{bin} u_i\, 0$ and $\operatorname{bin} u_i\, 1$ were chosen and the characteristic values of these words with respect to A were defined in such a way that $A \notin D_i$. In a more abstract setting, instead of two words we allow a finite number n of words $w_1, \ldots, w_n$ where we diagonalise. Instead of just two diagonalisation decisions d_1 and d_2, we allow Q to contain a finite number of possible diagonalisation decisions.

Each diagonalisation decision corresponds to a way of defining the characteristic string of the words $w_1, \ldots, w_n$. Although a diagonalisation decision is conceptually different from the corresponding bitstring, the notation can be simplified by identifying them. For this reason, let us require $Q \subseteq \{0, 1\}^n$, that is, Q is a set of possible choices for characteristic strings. Recall that subsets of $\{0, 1\}^n$ are called *n-pools*.

5.10 Definition of Diagonalisable Classes

Let Q be an n-pool. A language class C is *Q-diagonalisable* if there exists a countable covering V of C with the following property: for all classes $D \in V$ and for all pairwise distinct words $w_1, \ldots, w_n \in \Sigma^*$ of the same length we have $Q \not\subseteq \bigl\{\chi_L^n(w_1, \ldots, w_n) \mid L \in D\bigr\}$.

5.11 Example: Semirecursive Languages

The class of semirecursive languages is Q-diagonalisable for $Q = \{01, 10\}$: the countable covering is given by the covering $\{D_1, D_2, D_3, \ldots\}$ from the proof of theorem 5.3; for each class D_i, for any two words w_1 and w_2 the set $\bigl\{\chi_L^2(w_1, w_2) \mid L \in D_i\bigr\}$ is either $\{00, 10, 11\}$, namely if f_i picks w_1; or $\{00, 01, 11\}$, if it picks w_2. In either case, the set does not contain $Q = \{01, 10\}$.

5.12 Example: Countable Language Classes

Every countable class is Q-diagonalisable for all Q that contain at least two different bitstrings. To see this, just take the countable covering from example 5.7 where each class D contains just one language. Then for any n words $w_1, \ldots, w_n$ the set $\bigl\{\chi_L^n(w_1, \ldots, w_n) \mid L \in C\bigr\}$ has just one element and is thus not a superset of Q.

Note that every language class that is defined through some computational formalism, like 'is accepted by machines of a certain type' or 'is generated by a grammar of a certain type', is countable and thus Q-diagonalisable for all Q of size at least two.

5.13 Example: Advice Classes

The class P/k is Q-diagonalisable for all Q that contain at least $2^k + 1$ different bitstrings. To see this, consider the countable covering V from example 5.9. Each D contains all languages L that are accepted by a specific polynomially time-bounded Turing machine M for some advice.

For any pairwise different words $w_1, \dots, w_n \in \Sigma^\ell$ of the same length ℓ, the set $\{\chi^n_L(w_1, \dots, w_n) \mid L \in D\}$ contains at most 2^k different bitstrings: one bitstring for each advice for words of length ℓ. This set cannot contain all of Q.

The third step of the proof of theorem 5.3 was the construction of a branch that dictated the diagonalisation points. Since we identified diagonalisation decisions with their encodings, each node in the diagonalisation tree can be interpreted as a long bitstring. We no longer have two words that are associated with a node, but n words if Q is an n-pool. In the proof of theorem 5.3 the two words associated with a node $u \in \{0,1\}^*$ were obtained by appending 0 and 1, respectively. For a node $u \in Q^*$, we similarly associate the words $u \operatorname{bin}_n 1$, $u \operatorname{bin}_n 2$, … , $u \operatorname{bin}_n n$. Here $\operatorname{bin}_n t$ denotes the binary encoding of t, filled up with enough leading zeros to ensure that its length is exactly n. Instead of $\operatorname{bin}_n t$, one could also use the more economical tag $\operatorname{bin}_{\lceil \log_2 n \rceil + 1} t$, but that would be too long to write down.

5.14 Definition of Associated Words
Let Q be an n-pool. The *diagonalisation tree* for this pool is the tree Q^*. For a node $u \in Q^*$, the words $u \operatorname{bin}_n 1, \dots, u \operatorname{bin}_n n \in \{0,1\}^*$ are *associated* with u.

For example, if $n = 3$ and $Q = \{000, 001, 101, 110\}$, the three words associated with the node $101\,001 \in Q^*$ are 101001001, 101001010, and 101001011, see figure 5-1. Note that every bitstring $b \in \{0,1\}^*$ can be decomposed in at most one way in the form $u \operatorname{bin}_n t$ with $u \in Q^*$ and $t \in \{1, \dots, n\}$.

Just as in the proof of theorem 5.3, during the diagonalisation process a branch $\{u_1, u_2, u_3, \dots\}$ in the tree Q^* is constructed. The characteristic values of the words associated with the nodes on this branch become fixed during the diagonalisation. For the other words we are free to choose their characteristic values in any way that suits us. Figure 5-1 depicts this situation.

5.15 Definition of Diagonalisation along a Branch
Let Q be an n-pool and let $B = \{u_1, u_2, u_3, \dots\}$ be a branch of Q^*. Let $u_{i+1} = u_i b_i$ with $b_i \in Q$. A language $A \subseteq \{0,1\}^*$ *diagonalises along B* if for all $i \in \{1, 2, 3, \dots\}$ we have $\chi^n_A\bigl(u_i \operatorname{bin}_n 1, \dots, u_i \operatorname{bin}_n n\bigr) = b_i$.

We now have the necessary machinery for the formulation of the branch diagonalisation theorem.

5.16 Branch Diagonalisation Theorem
Let n be a positive integer and let C and C' be language classes. Suppose

there exists an n-pool Q such that

1. *C is Q-diagonalisable and*
2. *for every branch of Q^* there exists a language in C' that diagonalises along this branch.*

Then $C' \nsubseteq C$.

Proof. Let $V = \{D_1, D_2, D_3, \ldots\}$ be a countable covering for which a Q-diagonalisation of C is possible. We construct a branch $B = \{u_1, u_2, u_3, \ldots\}$ of Q^* such that no language that diagonalises along this branch is an element of any $D_i \in V$. It is thus not an element of C, but, by assumption, there exists a language in C' that diagonalises along this branch.

The branch is defined inductively by $u_1 := \epsilon$ and $u_{i+1} := u_i b_i$, where for each $i \in \{1, 2, 3, \ldots\}$ we choose the bitstring $b_i \in Q$ in such a way that $\chi_L^n\big(u_i \operatorname{bin}_n 1, \ldots, u_i \operatorname{bin}_n n\big) \neq b_i$ for all languages $L \in D_i$. Each time, such a bitstring must exist since C is Q-diagonalisable. No language that diagonalises along B will be an element of any D_i. QED

Section 5.2

Branch Diagonalisation and Separation of Verboseness Classes

In this section the branch diagonalisation method is applied to verboseness classes. Recall that $\mathrm{V}_{\mathrm{re}}(m, n)$ and $\mathrm{V}_{\mathrm{fa}}(m, n)$ were defined as the classes of all languages whose n-fold characteristic function is m-Turing-enumerable, respectively m-fa-enumerable. Using branch diagonalisation we can completely characterise the inclusion structure of these verboseness classes. The main result is that the two structures are identical.

Theorem 4.14 from the previous chapter gives a sufficient condition for the inclusions $\mathrm{V}_{\mathrm{re}}(m, n) \subseteq \mathrm{V}_{\mathrm{re}}(h, k)$ and $\mathrm{V}_{\mathrm{fa}}(m, n) \subseteq \mathrm{V}_{\mathrm{fa}}(h, k)$: it suffices that all (m, n)-good k-pools have size at most h. For the recursive case Beigel et al. (1995b) have shown that this is also a *necessary* condition, which reduces the inclusion and also the equality problem for Turing verboseness classes to finite combinatorics. In order to check whether $\mathrm{V}_{\mathrm{re}}(m, n) \subseteq \mathrm{V}_{\mathrm{re}}(h, k)$ holds, we just have to check whether all (m, n)-good k-pools have size at most h.

In the following we shall see that the condition is *also necessary* for finite automata: if there exists an (m, n)-good k-pool of size at least $h + 1$, then there exists a language in $\mathrm{V}_{\mathrm{fa}}(m, n)$ that is not an element of $\mathrm{V}_{\mathrm{fa}}(h, k)$. The proof technique for this result must necessarily be different from the finite injury argument used by Beigel et al. (1995b), since this technique cannot

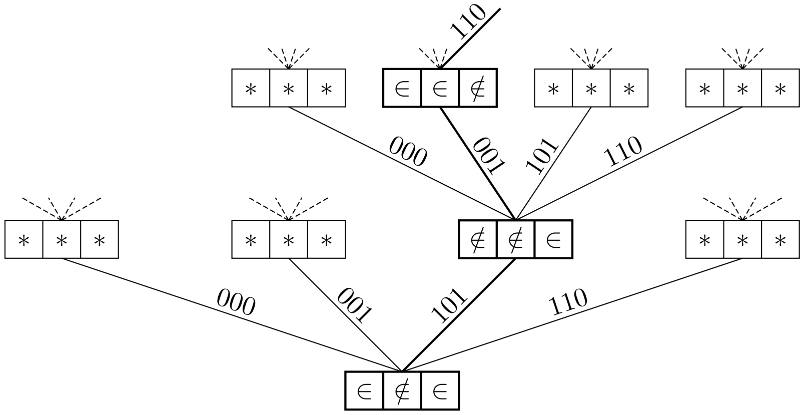


Figure 5-1
Visualisation of the diagonalisation process for $Q = \{000, 001, 101, 110\}$ and $n = 3$. A diagonalisation branch $B = \{\epsilon, 101, 101\,001, 101\,001\,110, \ldots\}$ is depicted in bold. The boxes at the tree nodes represent the words that are associated with these nodes. For example, the two boxes in the top row containing the symbol $\in$ represent the words $101\,001\,\mathrm{bin}_3\,1 = 101001001$ and $101\,001\,\mathrm{bin}_3\,2 = 101001010$. For languages that diagonalise along this branch, boxes containing the symbol $\in$ must be in the language, boxes containing the symbol $\notin$ may not be in the language, and boxes containing a 'don't-care-star' may or may not be in the language.

be applied to finite automata. Instead, I use branch diagonalisation. As promised in the introduction to this chapter, branch diagonalisation yields powerful results, provided it is applicable. Here it yields that there exists a language in $\mathrm{V}_{\mathrm{fa}}(m,n)$ that is not even an element of the much larger class $\mathrm{V}_{\mathrm{re}}(h,k)$. A corollary is the strong class separation of theorem 5.19. We also get a new, finite-injury-free proof of the result of Beigel et al. (1995B).

We must show that $\mathrm{V}_{\mathrm{fa}}(m,n) \not\subseteq \mathrm{V}_{\mathrm{re}}(h,k)$ if there exists an (m,n)-good k-pool Q of size $h+1$. The first step is to show that the class $\mathrm{V}_{\mathrm{re}}(h,k)$ is Q-diagonalisable. As the following theorem shows, this is even true for the class $\mathrm{V}_{\mathrm{re}}^{X}(h,k)$ of languages A for which χ_A^k is h-Turing-enumerable relative to the oracle X.

5.17 Theorem
Let h and k be positive integers. Let Q be a k-pool of size at least $h+1$. Then $\mathrm{V}_{\mathrm{re}}^{X}(h,k)$ is Q-diagonalisable for every oracle X.

Proof. Let $M_1^{()}$, $M_2^{()}$, $M_3^{()}$, ... be an enumeration of all oracle Turing machines that h-enumerate functions relative to the oracle X. Let D_i denote the class of languages L for which χ_L^k is h-enumerated by $M_i^{()}$ relative to X. The D_i form a countable covering of $\mathrm{V}_{\mathrm{re}}^{X}(h,k)$ and given any k pairwise different words $w_1,\dots,w_k \in \Sigma^*$ the set $\{\chi_L^k(w_1,\dots,w_k) \mid L \in D_i\}$ has size at most h. Thus Q is not a subset of this set. QED

To complete the branch diagonalisation, we must now show that for every branch in Q^* there exists a language in $\mathrm{V}_{\mathrm{fa}}(m,n)$ that diagonalises along this branch. At this point, for a given branch of Q^* we are still free to assign the characteristic values of the words that are not associated with the nodes on the branch. We make only scant use of this freedom: words not associated with the branch are not in the language.

5.18 Theorem
Let m, n, h, and k be positive integers. Let Q be an (m,n)-good k-pool with $0^k \in Q$. Then for every branch of Q^* there exists a language in $\mathrm{V}_{\mathrm{fa}}(m,n)$ that diagonalises along this branch.

Proof. Let $\{u_1,u_2,u_3,\dots\}$ be any branch of Q^*. Let $A \subseteq \{0,1\}^*$ be the smallest language that diagonalises along this branch, that is, let A contain no words that are not associated with a node on the branch.

In order to show $A \in \mathrm{V}_{\mathrm{fa}}(m,n)$, we construct a finite automaton that produces on input of any n words w_1, ..., w_n an n-pool P as its final output that has size at most m and that contains $\chi_A^n(w_1,\dots,w_n)$.

For the definition of the automaton we define a series of regular functions that 'preprocess' the input words. (Recall that a function is regular if its graph is regular.) Since the composition of regular functions is regular, the preprocessing can be performed by a finite automaton.

First Step: Replacement of Bothersome Words

The first function f_1 replaces 'bothersome' input words by 'nice' fixed input words. An input is *bothersome* if it is not of the form $u \operatorname{bin}_n t$ for some node $u \in Q^*$ and some $t \in \{1, \dots, n\}$. Bothersome words are not elements of A. We replace them by a fixed word of the correct form, such that in the following we can assume that all input words have correct form.

Let u_* be a fixed node and let $t_* \in \{1, \dots, n\}$ be a fixed number such that $u_* \operatorname{bin}_n t_* \notin A$. The function f_1 is defined as follows: it maps an input tuple $(w_1, \dots, w_n)$ to a tuple $(w'_1, \dots, w'_n)$, where $w'_i = w_i$ if w_i is not bothersome, and $w'_i = u_* \operatorname{bin}_n t_*$ if it is. The graph of f_1 is clearly regular. The output of f_1 does not contain any bothersome words and it has the same characteristic string as its input.

Second Step: Decomposition into Node Part and Tag Part

The second preprocessor function f_2 decomposes the input words $w_i = u \operatorname{bin}_n t$ into their 'node part' u and their 'tag part' t. Formally, the function maps a tuple $(w_1, \dots, w_n)$ to a long tuple $(v_1, \dots, v_n, t_1, \dots, t_n)$, where $w_i = v_i \operatorname{bin}_n t_i$, $v_i \in Q^*$, and $t_i \in \{1, \dots, n\}$. This function is also regular.

Third Step: Which Nodes are Ancestors of other Nodes?

The third function f_3 takes a tuple $(v_1, \dots, v_n, t_1, \dots, t_n)$ as input. Its task is, roughly spoken, to discern how the nodes v_i are related with respect to the ancestor relation.

The tuple $(v_1, \dots, v_n)$ may contain a node v_i several times, since several input words may be associated with the same node. Let $\ell \leq n$ be the cardinality of the set $\{v_1, \dots, v_n\}$ and let $\{y_1, \dots, y_\ell\} = \{v_1, \dots, v_n\}$. For $j \in \{1, \dots, \ell\}$ let n_j denote the number of input words that are associated with y_j, that is, the number of indices i for which $v_i = y_j$.

The output of f_3 equals its input extended by the following graph: Its vertex set is $\{1, \dots, \ell\}$. There is an edge from a vertex i to another vertex j with the label $b \in Q$ if y_i is a proper prefix of y_j and if the path through y_i to y_j 'heads in the direction b at y_i'. More precisely, there is such an edge if $y_i b \sqsubseteq y_j$ with $b \in Q$. Since there are only finitely many possible edges and since checking whether $y_i b \sqsubseteq y_j$ holds can be done by a finite automaton, the function f_3 is regular.

Fourth Step: Computation of the Branching Sequences

The function f_4 gets a tuple $(v_1, \dots, v_n, t_1, \dots, t_n, G)$ as input, where G is the graph produced by f_3. The task of f_4 is to replace G by a set of *branching sequences*, which is an ad hoc concept that I introduce only for the purposes of this proof. For a branch $\{\tilde{u}_1, \tilde{u}_2, \tilde{u}_3, \dots\}$ of Q^*, the first element of the corresponding branching sequence is a pair (j_1, b_1) where j_1 is the index of the first y_j through which the branch heads and where $b_1 \in Q$ is the 'direction' in which the branch heads at y_j, that is, $y_{j_1} b_1 \sqsubseteq \tilde{u}_{k_1}$ for a minimal index k_1. The next element of the sequence is the pair (j_2, b_2) where j_2 is the index of the second y_j through which the branch heads and where $b_2 \in Q$ is the corresponding direction, that is, $y_{j_2} b_2 \sqsubseteq \tilde{u}_{k_2}$ for a minimal $k_2 > k_1$. In this way a sequence of pairs is defined. Note that the maximum length of such a sequence is ℓ. Thus there are only finitely many such sequences.

The set output by f_4 contains all branching sequences that are induced by the branches of Q^*. At first sight, this set might seem a little hard to compute since Q^* has uncountably many branches. However, we just need to check for every given sequence, of which there are only finitely many, whether it is induced by some branch. This can be done by checking for a given sequence whether it respects the ancestor relation stored in the graph G: nodes later in the branching sequence must be ancestors of all previous nodes and no node may be 'skipped' by the sequence. Formally, a sequence $\bigl((j_1, b_1), \dots, (j_p, b_p)\bigr)$ *respects* the relation if $j_1 \rightarrow_{b_1} j_2 \rightarrow_{b_2} \cdots \rightarrow_{b_{p-1}} j_p$ is a path of maximal length to j_p in G. This shows that the computation of the branching sequences can be implemented by a big lookup table. Thus it can trivially be performed by a finite automaton.

Fifth Step: Characteristic Values from Branching Sequences

The final step is to turn the branching sequences into characteristic strings for the input words. For each branching sequence $\bigl((j_1, b_1), \dots, (j_p, b_p)\bigr)$ we output a possible characteristic string for the input words. It is defined as follows: for input words $v \operatorname{bin}_n t$ where v is not one of the nodes $y_{j_1}, \dots, y_{j_p}$ we output 0; for input words $y_{j_i} \operatorname{bin}_n t$ with $i \in \{1, \dots, p\}$ we output the tth bit of b_i. The important observation here is that *for the branching sequence of the 'right' branch $\{u_1, u_2, u_3, \dots\}$ the just defined bitstring will be the characteristic string of the input words with respect to the language A.* Thus the correct characteristic value of the input words will be output by this procedure. Since this final step can also be implemented by a table lookup, the whole enumeration process can be performed by a finite automaton.

The Produced Pool is Good

It remains to show that the produced set P of bitstrings has size at most m. To prove this, let us first summarise what bitstrings are enumerated: P is the set of characteristic strings of the input words with respect to minimal languages that diagonalise along branches of Q^*. Let us count how many such bitstrings exist.

Consider a branch that goes through *none* of the nodes v_i. Let L be the minimal language that diagonalises along this branch. The characteristic string $\chi_L^n(w_1, \dots, w_n)$ is 0^n. This will be the first bitstring found in P.

Now consider a node v_i that is not a descendant of another node v_j and consider all branches going through v_i, but going through no other node v_j.

For words not associated with v_i, the characteristic values will once more be 0. However, the characteristic values of words associated with v_i can both be 0 and 1, depending on the 'direction' in which the branches head at v_i. If the tags of the input words associated with v_i are $t_1, \dots, t_{n_i}$, then all bitstrings in $\{b[t_1, \dots, t_{n_i}] \mid b \in Q\}$ are possible characteristic strings of these words.

Recall from notation 4.12 that the number $\mathrm{g}_Q(n_i)$ is an upper bound on the size of $\{b[t_1, \dots, t_{n_i}] \mid b \in Q\}$. Thus the branches going through v_i, but going through no other node v_j, induce at most $\mathrm{g}_Q(n_i)$ bitstrings in P. However, since the branches going through v_i that head off in the 'direction' $0^k \in Q$ all induce the bitstring 0^n once more, the branches solely going through v_i actually induce at most $\mathrm{g}_Q(n_i) - 1$ *new* bitstrings in P apart from 0^n.

Now consider a node v_i that has no ancestor in the set $\{v_1, \dots, v_n\}$ and consider a direct descendant v_j of v_i. 'Direct' means that there exists no node in the set $\{v_1, \dots, v_n\}$ that is a descendant of v_i and an ancestor of v_j. Consider the branches going exactly through v_j and v_i. The branches induce $\mathrm{g}_Q(n_j)$ many bitstrings in P, but once more one of them will already be found in P, namely the one induced by branches that go through v_j and v_i and head off in the direction 0^k in v_j. Using structural induction one can now show that *for each node v_i, the branches going exactly through v_i and its ancestors induce at most $\mathrm{g}_Q(n_i) - 1$ new bitstrings in Q.* In total we get

$$\begin{aligned}|P| &\le 1 + \bigl(\mathrm{g}_Q(n_1) - 1\bigr) + \bigl(\mathrm{g}_Q(n_2) - 1\bigr) + \cdots + \bigl(\mathrm{g}_Q(n_\ell) - 1\bigr) \\ &= \mathrm{g}_Q(n_1) + \cdots + \mathrm{g}_Q(n_\ell) - \ell + 1 \le m,\end{aligned}$$

where the (m, n)-goodness of Q was used in the last step. QED

5.19 Verboseness Class Inclusion Theorem
Let m, n, h, and k be positive integers. Let X be any oracle. Then the following statements are equivalent:

1. *All (m, n)-good k-pools have size at most h.*
2. $\mathrm{V_{re}}(m, n) \subseteq \mathrm{V_{re}}(h, k)$.
3. $\mathrm{V_{fa}}(m, n) \subseteq \mathrm{V_{fa}}(h, k)$.
4. $\mathrm{V_{fa}}(m, n) \subseteq \mathrm{V}^X_{\mathrm{re}}(h, k)$.

Proof. Statement 1 implies both statements 2 and 3 by corollary 4.15. Since the inclusions $\mathrm{V_{fa}}(m, n) \subseteq \mathrm{V_{re}}(m, n) \subseteq \mathrm{V}^X_{\mathrm{re}}(m, n)$ hold trivially for all m and n, both statements 2 and 3 imply statement 4. To show that statement 4 implies statement 1, let us argue by contraposition. If there exists an (m, n)-good k-pool of size $h + 1$, there also exists one containing 0^k (if necessary, toggle the bits in all bitstrings at appropriate positions). For this pool Q, the branch diagonalisation theorem, theorem 5.16, can be applied to the classes $C' := \mathrm{V}_{\mathrm{fa}}(m,n)$ and $C := \mathrm{V}^X_{\mathrm{re}}(h,k)$. The two conditions from the branch diagonalisation theorem are met by theorems 5.17 and 5.18. QED

Theorem 5.19 reduces not only the decision problem for the inclusion of finite automata and Turing machine verboseness classes to finite combinatorics, but also the decision problem for equality. We have $\mathrm{V}_{\mathrm{fa}}(m,n) = \mathrm{V}_{\mathrm{fa}}(h,k)$ iff $\mathrm{V}_{\mathrm{re}}(m,n) = \mathrm{V}_{\mathrm{re}}(h,k)$ iff the following combinatorial property holds: all (m,n)-good k-pools have size at most h and all (h,k)-good n-pools have size at most m. In a complicated analysis Beigel et al. (1995B) were able to simplify this combinatorial property, see the following theorem, where $(a,b] := \{x \in \mathbb{R} \mid a < x \leq b\}$.

5.20 Verboseness Class Equality Theorem
Let n and k be positive integers. Let $m \in \{1,\dots,2^n\}$ and $h \in \{1,\dots,2^k\}$. Then $\mathrm{V}_{\mathrm{fa}}(m,n) = \mathrm{V}_{\mathrm{fa}}(h,k)$ iff $\mathrm{V}_{\mathrm{re}}(m,n) = \mathrm{V}_{\mathrm{re}}(h,k)$, which in turn holds iff

1. *$m = h$ and $n = k$, or*
2. *the quotients m/n and h/k both lie in the same of the following intervals:*

$$(0,1];\ (1,\tfrac{3}{2}];\ (\tfrac{3}{2},\tfrac{5}{3}];\ (\tfrac{5}{3},\tfrac{7}{4}];\ (\tfrac{7}{4},\tfrac{9}{5}];\ (\tfrac{9}{5},\tfrac{11}{6}];\ (\tfrac{11}{6},\tfrac{13}{7}];\dots.$$

The first of these intervals represents the nonspeedup theorem: if both $m/n \in (0,1]$ and $h/k \in (0,1]$, then $m \leq n$ and $h \leq k$ and hence, by the nonspeedup theorem, $\mathrm{V}_{\mathrm{re}}(m,n)$ and $\mathrm{V}_{\mathrm{re}}(h,k)$ both contain exactly the recursive languages.

Section 5.3

Branch Diagonalisation and Separable Sets

This section starts the study of sets that are separable by finite automata. This study is continued in the next chapter. Using branch diagonalisation, we show that there exist languages A and B that are inseparable by finite automata, but for which the closely related relations $\{(x,y) \in A^2 \mid x \neq y\}$ and $\{(x,y) \in B^2 \mid x \neq y\}$ are separable by a finite automaton. Once more, the branch diagonalisation method allows us to derive a stronger result: there even exist *recursively* inseparable languages with this property. The result can also be extended in a different direction, which allows the construction of a counterexample to an old result of Kinber (1976, theorem 2).

Two sets A and B are *separable* by a set C if A is a subset of C and B is a subset of C's complement. For elements in C we know that they cannot be elements of B and for elements not in C we know that they cannot be elements of A. A graphic way of envisioning separability is the following: think of the sets A and B as two pieces of cake and think of the separating set as a cheese-cover. In order to separate the sets A and B, the cheese-cover must completely cover the first piece of cake, while the second piece of cake must lie completely outside the cover.

Given two sets A and B, one usually seeks a separating set C that is as simple as possible. It can happen that two difficult sets A and B can be separated by a very simple set C. For example, if A is an arbitrary subset of $\{0\}^+$ and B is a subset of $\{1\}^+$, then A and B can be separated by the *regular* language $\{0\}^*$. This is true even if A and B are *nonrecursive*. On the other hand, a language A can be separated from its complement $\bar{A}$ only by A itself. In particular, it cannot be separated from its complement by any set that is simpler than A.

Sets that can be separated by a recursive set are commonly called *recursively separable*. Sets that can be separated by a set that can be decided in polynomial time are called *polynomially separable*. I propose calling sets that can be separated by a regular set *fa-separable*.

5.21 Example: Polynomially Separable Languages
An example of supposedly difficult, but still polynomially separable sets are the languages k-clique and $(k-1)$-colourable. The first language contains all coded pairs $\langle \operatorname{bin} G, \operatorname{bin} k\rangle$ such that G is an undirected finite graph that contains a clique of size k. The second language contains all coded pairs $\langle \operatorname{bin} G, \operatorname{bin} k\rangle$ such that G can be coloured with $k-1$ colours. The languages are disjoint: any colouring of a graph that contains a clique of size k has to assign a different colour to each vertex of the clique; thus at least k different colours are needed. It is much harder to see that the languages are polynomially separable. The proof is based on the 'sandwich theorem' due to Lovász (1979), see (Knuth, 1994) for an overview. It states that the Lovász number $\vartheta(G)$ of a graph G is sandwiched between the clique number and the chromatic number of G. Since Grötschel et al. (1981) have shown that the Lovász number can be computed in polynomial time, we get a polynomial-time separation algorithm. Further examples of polynomially separable languages are presented in an article of Pudlák (2001).

The first result on separable sets uses the following notation.

5.22 Notation
Let $A \subseteq U$ be a subset of a universe U. For a positive integer n, let

$A^{(n)}$ denote the set of all n-tuples of pairwise different elements of A. For $k \in \{0, \dots, n\}$, let $A^{\binom{n}{k}}$ denote the set of all n-tuples of pairwise different elements in U such that exactly k of them are in A.

The set $A^{(n)}$ is 'almost' the n-fold Cartesian product A^n of A, only tuples that contain the same component twice are missing. Note that $A^{(2)} = \bigl\{(x, y) \in A^2 \mid x \neq y\bigr\}$ is the 'inequality relation' on A. For elements of $A^{\binom{n}{k}}$, we 'choose' exactly k elements in A out of n elements in total. Note that $A^{\binom{n}{n}} = A^{(n)}$ and $A^{\binom{n}{0}} = \bar{A}^{(n)}$.

The following proof employs the branch diagonalisation method, but it does not appeal to the branch diagonalisation theorem since the basic objects involved are pairs of (separable or inseparable) languages rather than individual languages. It would be possible to rephrase the branch diagonalisation theorem for this more general setting, but the following proof demonstrates nicely how branch diagonalisation can be used as a *method* without referring to a particular theorem.

5.23 THEOREM
For every oracle X, there exist disjoint languages A and B that are not separable by any language that is recursive in X, but for which $A^{(n)}$ and $B^{(n)}$ are fa-separable for all $n \geq 2$.

Proof. For a branch $Z = \{u_1, u_2, u_3, \dots\}$ of the binary tree $\{0, 1\}^*$, let $A_Z := \{u_2, u_3, u_4, \dots\}$ denote the nodes on this branch except for the root, and let B_Z denote the set of all elements in A_Z with the last bit toggled, that is, with 0 replaced by 1 and vice versa.

The first part of the branch diagonalisation is the construction of a branch Z such that A_Z and B_Z are recursively inseparable relative to the oracle X. This branch is constructed as follows: let $M_1^{()}$, $M_2^{()}$, $M_3^{()}$, ... be an enumeration of all oracle Turing machines that could witness such a separation relative to X. As always, the enumeration need not be effective. In stage i we guarantee that the set $L_i := \mathrm{L}\bigl(M_i^{()}, X\bigr)$ of words accepted by $M_i^{()}$ relative to the oracle X does not separate A_Z and B_Z, that is, either $A_Z \not\subseteq L_i$ or $B_Z \not\subseteq \overline{L_i}$.

Suppose we have already constructed u_i and must now decide how to define u_{i+1}. We check whether both $u_i0 \in L_i$ and $u_i1 \notin L_i$ hold. If this is the case, let $u_{i+1} := u_i1$, which will ensure both $A_Z \not\subseteq L_i$ and $B_Z \not\subseteq \overline{L_i}$. If it is not the case, let $u_{i+1} := u_i0$, which will ensure either $A_Z \not\subseteq L_i$ or $B_Z \not\subseteq \overline{L_i}$. In either case we guarantee that L_i does not separate A_Z and B_Z.

The relations $A_Z^{(n)}$ and $B_Z^{(n)}$ are fa-separable for every branch Z: Any two words in A_Z are comparable with respect to the prefix ordering, but no two different words in B_Z are comparable with respect to the prefix ordering. Thus for every branch Z the relation $A_Z^{(2)}$ is a subset of $\sqsubseteq$, whereas $B_Z^{(2)}$ is a subset of the complement of $\sqsubseteq$. In particular, $A_Z^{(n)}$ and $B_Z^{(n)}$ are fa-separable for every Z and all $n \geq 2$. QED

The proof of the above theorem was a by-product of a futile attempt to prove a strengthened version of a theorem of Kinber (1976). I ran into problems that, ultimately, lead to the formulation of theorem 5.23, whose claim is almost the exact opposite of (Kinber, 1976, theorem 2). The 'opposite' of theorem 5.23 above would be the following claim: 'if $A^{(n)}$ and $B^{(n)}$ are fa-separable, then A and B are fa-separable'. Kinber did not quite claim this. Rather, he claimed that A and B are fa-separable under the slightly stronger assumption that A and B are (m, n)-fa-separable for $m > n/2$. In order to refute Kinber's claim, one thus has to construct fa-inseparable languages A and B that are nevertheless (m, n)-fa-separable for some $m > n/2$. This is possible as the argument of theorem 5.24 below shows. Before we prove this theorem, let us first review Kinber's notion of (m, n)-fa-separability, which is a generalisation of fa-separability.

For two languages A and B let us call a pair (w, b), consisting of a word $w \in \Sigma^*$ and a bit $b \in \{0, 1\}$, *bad* if either $b = 1$ and $w \in B$ or if $b = 0$ and $w \in A$. Two disjoint languages A and B are called *recursively* (m, n)*-separable*, respectively (m, n)*-fa-separable*, if there exists a recursive, respectively regular, n-ary function f that on input of any n pairwise different words $w_1, \dots, w_n$ outputs a bitstring $b \in \{0, 1\}^n$ such that at most $n - m$ of the pairs $\big(w_1, b\texttt{[}1\texttt{]}\big), \dots, \big(w_n, b\texttt{[}n\texttt{]}\big)$ are bad. The intuition behind this definition is that an (m, n)-separating function must output 1 for words in A and 0 for words in B and it may make up to $n - m$ mistakes. Words that are neither in A nor in B play no rôle. Note that languages are $(1, 1)$-fa-separable iff they are fa-separable.

5.24 Theorem

For every oracle X, there exist disjoint $(3, 5)$*-fa-separable languages that are not recursively separable relative to X.*

Proof. For the purposes of this proof, a *forest* is partial ordering $(F, \leq)$ such that for every *node* $y \in F$ the set $\{x \mid x \leq y\}$, called the *path to y*, is well-ordered by $\leq$.

I show that for every branch Z of the tree $\{0, 1\}^*$ the languages $A := A_Z$ and $B := B_Z$ constructed in the proof of theorem 5.23 are $(3, 5)$-fa-separable. To prove this, we must construct a DFA M that on input of any five words $w_1, \dots, w_5 \in \{0, 1\}^*$ will claim '$w_i \in A$' or '$w_i \in B$' for each i such that among the claims for words in $A \cup B$ at most two claims are wrong.

Let $W := \{w_1, \dots, w_5\}$. Let y_i denote the word w_i without the last bit (if w_i is the empty string, then $w_i \notin A \cup B$ and we can ignore it). Let us say that w_i is *associated* with y_i. Let us call two words w_i and w_j *siblings* if they are associated with the same word $y_i = y_j$. Let $Y := \{y_1, \dots, y_5\}$.

Similarly to the proof of theorem 5.18, the automaton scans the forest structure of $(Y, \sqsubseteq)$. This means that for each pair (i, j) it determines whether $y_i \sqsubseteq y_j$ holds. Then it considers all paths in the forest $(Y, \sqsubseteq)$ for which at least three words are associated with the nodes on this path. Given such a path, leading to a node y, the automaton assigns outputs to some of the input words according to the following rule: for each $i \in \{1, \dots, 5\}$, if y_i is a proper prefix of y and $w_i \sqsubseteq y$ we claim '$w_i \in A$'; and if y_i is a proper prefix of y and $w_i \not\sqsubseteq y$ we claim '$w_i \in B$'. Since a word may be associated with a node that lies on more than one path, the just given rule may assign conflicting outputs to a word w_i. Also, it may not assign any output to w_i at all. In either case the automaton outputs '$w_i \in A$'. Note that in both cases, if w_i has a sibling w_j, the automaton also outputs '$w_j \in A$'. See figure 5-2 for an example.

According to the construction, the output of the automaton for a word $w_i \in A \cup B$ can be *incorrect* only if y_i is not a proper prefix of the last node of any of the above-mentioned paths or if two of these paths 'split' exactly at y_i. Note that if $w_i \in A \cup B$ has a sibling, at least one output will be correct for the sibling pair.

I now argue that the described procedure $(3, 5)$-fa-separates A and B. Let $W' := W \cap (A \cup B)$ be the words for which our algorithm must produce a correct output with an error margin of 2. Since for $|W'| \leq 2$ we can output anything, the interesting cases are $|W'| \in \{3, 4, 5\}$.

For $|W'| = 5$, there can only be a mistake for one word associated with the top node. Since there cannot be any splits, we make at most one mistake.

For $|W'| = 4$, a mistake is possible for one word associated with the top node, and there can be another mistake caused by a split earlier on the path to which the words in W' are associated. In total, we can make at most two mistakes.

For $|W'| = 3$, if a sibling pair is associated with any node on the path, at least one output is correct and we are done. Otherwise, one mistake is again possible for the word associated with the top node. If there is no split at the root node, we make at most one additional mistake at the 'middle' node. So assume that there is a split at the root node. Then *two* additional input words must be associated with the path leading away in the wrong direction from the root (since we considered only paths to which at least three words are associated). But then there cannot be another split at the middle node of our main path and the output for it must be correct. QED

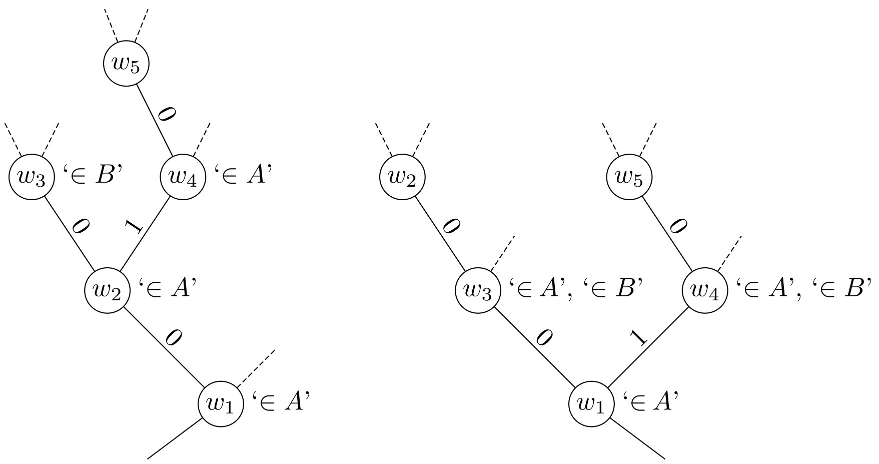


FIGURE 5-2
Two possible arrangements of input words and the claims made for these words in the proof of theorem 5.24. In both examples, w_3 and w_4 are siblings. In the left example, the path leading to w_3 causes the claims '$w_1 \in A$' and '$w_2 \in A$' to be made. The path leading to w_5 also causes these claims to be made, plus the additional claims '$w_3 \in B$' and '$w_4 \in A$'. No claim is made for w_5 and hence, in the end, '$w_5 \in A$' would be claimed. In the right example, the path leading to w_2 causes the claims '$w_1 \in A$', '$w_3 \in A$', and '$w_4 \in B$' to be made. The path leading to w_5 causes the claims '$w_1 \in A$', '$w_3 \in B$', and '$w_4 \in A$' to be made. The conflicting claims and the missing claims for the top nodes are resolved by claiming membership in A for all of these nodes. Hence in the end, '$w_i \in A$' is claimed for all nodes.

The above proof is specific for the numbers 3 and 5 and it is not clear how it could be adapted to different numbers. I conjecture that, as in the recursive setting, the claim is true for all $m < n$.

5.25 Conjecture
Let m and n be positive integers with $m < n$. Then there exist disjoint (m, n)-fa-separable languages that are recursively inseparable.

Although theorem 2 of Kinber's paper fails, corollaries of this theorem can still be true. For example, Kinber's claim is true if instead of arbitrary disjoint sets A and B we consider A and $\bar{A}$: if a set and its complement are (m, n)-fa-separable for $m > n/2$, then it is regular. Austinat et al. (2000) were the first to give a (correct) proof of this corollary. In the next chapter we shall see that it can also be derived from the restricted cardinality theorem for finite automata.

To conclude this section on separability and branch diagonalisation, I present a final example of a language is difficult and closely related relations are fa-separable. As we shall see in the next chapter, things change drastically if either $A\times\bar{A}$ or $\bar{A}\times A$ is removed from the claim of the theorem.

5.26 Theorem
There exists a language A that is not semirecursive, but for which there exist regular supersets of $A \times A$, $A \times \bar{A}$, $\bar{A} \times A$, and $\bar{A} \times \bar{A}$ whose intersection is empty.

Proof. Consider the language A constructed in the proof of theorem 5.3. The language is a branch $\{u_1, u_2, u_3, \dots\}$ of the tree $\{0, 1\}^*$ that is not semirecursive. As pointed out in the caption of figure 3-1 on page 64, every branch of $\{0, 1\}^*$ is $(3, 2)$-fa-verbose. Lemma 6.3 from the next chapter states that for $(3, 2)$-fa-verbose languages there exist regular supersets of $A \times A$, $A \times \bar{A}$, $\bar{A} \times A$, and $\bar{A} \times \bar{A}$ whose intersection is empty. (The proof of lemma 6.3 does not use the present theorem, big promise.) QED

Section 5.4

Branch Diagonalisation and the Complexity of Odd Languages

Branch diagonalisation can be used to separate bounded queries reduction closures of selective languages. Recall from definition 5.1 that a language is *selective* if a selector function can select from any two words one word that is 'more likely' to be in the language.

Bounded queries reductions are reductions in which only a fixed number of queries may be posed to the oracle. They are often motivated as follows:

Suppose we wish to solve some difficult problem A, but we have insufficient resources to do so. For this reason, we introduce an oracle B to which we pose questions. The oracle models an expensive resource like a database or a computationally difficult problem for which we have a special purpose software or hardware. We wish to minimise and bound the number of times we have to invoke the expensive oracle in order to solve our problem A. The number of times we have to invoke the oracle is the *query complexity of A relative to B*. For a recent overview of the known recursion-theoretic results and applications see the book 'Bounded Queries in Recursion Theory' by Gasarch and Martin (1999).

The study of the structure of reduction closures of selective languages has a long tradition. First, and most importantly, the reduction closures of semirecursive languages are one of the key ingredients of the solution of Post's problem (Post, 1944). Post asked whether there are nonrecursive, recursively enumerable languages that are not Turing-equivalent to the halting problem. Informally speaking, he asked whether there exist problems that are 'easier' than the halting problem, but that cannot be solved by any computer. The answer to Post's problem is positive. The original proof uses reduction closures (though not with a bounded number of queries) of semirecursive languages, see the book of (Odifreddi, 1989) for details.

Recently, bounded queries reduction closures of semirecursive languages have been studied by Beigel et al. (2000). In an article in the Journal of Symbolic Logic entitled 'The Complexity of $\textsc{odd}_n^A$' they show that, for each k, asking k parallel queries to any nonrecursive, semirecursive language allows one to decide a language that cannot be decided asking only $k-1$ parallel queries to *any* semirecursive language. In particular, the parallel k-queries reduction closure of the class SEMIREC is larger than its parallel $(k-1)$-queries reduction closure.

In a different line of research, Hemaspaandra et al. (1996) have studied bounded queries reduction closures of P-selective languages. As in the recursive setting, it turns out that k parallel queries to the class of P-selective languages are more powerful than $k-1$ parallel queries. In the polynomial-time setting, the research is mainly motivated by the fact that the *Turing* reduction closure of the class P-sel of P-selective languages is the class P/poly of languages having small circuits. Thus, the bounded queries reduction closures of P-sel are part of the fine structure of P/poly. This study has been remarkably fruitful. For example, Agrawal and Arvind (1996); Beigel et al. (1995A); and Ogihara (1995) have independently shown that if the satisfiability problem is bounded queries reducible to a P-selective language, then P = NP. It is not known whether SAT $\in$ P/poly has the same consequence.

In the following branch diagonalisation is used to prove a strong separation of the bounded reduction closures of selective languages. Both of the above-mentioned results of Beigel et al. (2000) and of Hemaspaandra et al. (1996) concerning the k-queries reduction closures of SEMIREC, respectively P-sel, are corollaries of this separation. In essence, I show that there exists an fa-selective language A such that the language ODD^k_A, which contains all k-tuples of words such that an odd number of them is in A, is not reducible to any semirecursive language asking only $k-1$ parallel queries.

The focus of this section is on the employment of a branch diagonalisation to obtain the results—both Beigel et al. and Hemaspaandra et al. also show different or stronger results that do not follow from the results proved in this section. A general treatment of reduction closures of selective languages can be found in the technical report (Tantau, 2000).

The following notations are used in this section. Let SEMIREC^X denote the class of all languages that have a selector that is recursive in X. Let P^X-sel denote the class of all languages that have a selector that is polynomial-time computable with (arbitrary) oracle access to X. Let $A \leq^X_{k\text{-tt}} B$ denote that A is reducible to B with k parallel queries via a Turing machine that has (arbitrary) oracle access to X. Let $A \leq^{\text{P}^X}_{k\text{-tt}} B$ denote that the Turing machine is furthermore polynomially time-bounded. For a reduction $\leq_r$ and a class C of languages, let $\text{R}_r(C) := \{A \mid A \leq_r B \in C\}$ denote the *reduction closure of* C and let $\text{E}_r(C) := \{A \mid A \leq_r B,\ B \leq_r A,\ B \in C\}$ denote the *equivalence closure of* C.

5.27 Definition of Odd Languages

For a language $A \subseteq \Sigma^*$ and a positive integer n let

$$\text{ODD}^n_A := \bigl\{\langle w_1, \dots, w_n\rangle \mid w_1, \dots, w_n \in \Sigma^*,\ \#^n_A(w_1, \dots, w_n) \text{ is odd}\bigr\}.$$

5.28 Theorem

For every oracle X, there exists an fa-selective language A such that for all positive integers k

$$\text{ODD}^{k+1}_A \notin \text{R}^X_{k\text{-tt}}\bigl(\text{SEMIREC}^X\bigr).$$

Proof. Before we plunge into the details of the branch diagonalisation, a little preparation is helpful. We begin with some simple combinatorics. Then we construct a pool Q such that the class $\text{R}^X_{k\text{-tt}}\bigl(\text{SEMIREC}^X\bigr)$ is Q-diagonalisable. Once this is done, we show that for every branch of Q^* there exists an fa-selective language A such that a language A' in the many-one reduction closure of ODD^{k+1}_A diagonalises along this branch. The languages A and A' are constructed in tandem with the diagonalisation branch.

Walks and Transition Counts

A *walk on the n-dimensional hypercube* or just an *n-walk* is a sequence $(b_1, \dots, b_\ell)$ of bitstrings of length n such that from one bitstring to the next exactly one bit changes. An example of a 3-walk is the sequence $(000, 010, 011, 010, 110)$. For a position $p \in \{1, \dots, n\}$, the *transition set* T_p is the set of indices i with $b_i[p] \neq b_{i-1}[p]$. The *transition count* of a position p is the cardinality of T_p. For example, the transition sets of the above walk are $\mathrm{T}_1 = \{5\}$, $\mathrm{T}_2 = \{2\}$, and $\mathrm{T}_3 = \{3, 4\}$; its transition counts are 1, 1, and 2.

The length of an n-walk having transition counts at most k is bounded by $nk + 1$. For any positive integers n and k with $nk + 1 < 2^n$ there exists an n-walk $(b_1, \dots, b_\ell)$ with transition counts at most $k + 1$ such that no n-walk $(b'_1, \dots, b'_m)$ with transition counts at most k visits all bitstrings in the set $\{b_1, \dots, b_\ell\}$. To see this, consider a walk $(b'_1, \dots, b'_m)$ visiting a maximum number of bitstrings among all n-walks that have transition counts at most k. Since $nk + 1 < 2^n$, there exists a bitstring $b' \in \{0, 1\}^n$ that is not visited. Extend the walk $(b'_1, \dots, b'_m)$ to a walk $(b'_1, \dots, b'_m, b'_{m+1}, \dots, b')$ as follows: from b'_m to b'_{m+1} flip the first bit where b'_m and b' differ, from b'_{m+1} to b'_{m+2} flip the second bit where they differ, and so on. This will increase all transition counts by at most one and the new walk will visit at least one bitstring more than any walk having transition counts at most k.

The Reduction Closure is Q-Diagonalisable

Let n be chosen large enough such that $nk + 1 < 2^n$. Let $Q = \{b_1, \dots, b_\ell\}$, where $(b_1, \dots, b_\ell)$ is an n-walk with transition counts at most $k + 1$ that is not covered by any n-walk having transition counts at most k. We may assume $b_1 = 0^n$.

I claim that the class $\mathrm{R}^X_{k\text{-tt}}(\textsc{semirec}^X)$ is Q-diagonalisable. Let $M^{()}_1$, $M^{()}_2$, $M^{()}_3$, … be an enumeration of all Turing machines that compute selectors relative to the oracle X. Let $R^{()}_1$, $R^{()}_2$, $R^{()}_3$, … be an enumeration of all Turing reduction machines that ask k parallel queries on every input relative to the oracle X. Let $D_{m,r}$ be the set of all languages L that are reduced by R^X_r to a language for which M^X_m computes a selector. Clearly, the sets $D_{m,r}$ form a countable covering of $\mathrm{R}^X_{k\text{-tt}}(\textsc{semirec}^X)$.

We have to show that for any n different words w_1, … , w_n we have $Q \not\subseteq \bigl\{\chi^n_L(w_1, \dots, w_n) \mid L \in D_{m,r}\bigr\}$. To prove this, we show that the elements of the set $\bigl\{\chi^n_L(w_1, \dots, w_n) \mid L \in D_{m,r}\bigr\}$ are visited by an n-walk with transition counts at most k.

The reduction machine R^X_r maps each word w_i to queries q^1_i, … , q^k_i. The queries are posed to an oracle Y for which M^X_m computes a selector. Let $S := \{q^1_1, \dots, q^k_n\}$ be the set of all queries produced for the words. There exist different 'scenarios' that correspond to the possible characteristic strings of these queries with respect to Y. In other words, they correspond to different possible answers to these queries.

The sought walk is constructed as follows: it starts at the characteristic string that is induced for the input words in the scenario where all queries are answered 'no'. The walk then leads directly to the characteristic string that is induced in the scenario where only one query is answered 'yes' and all other queries are answered 'no'. This one query is the query that is 'most likely' to be in Y according to M_m^X. The walk then leads directly to the bitstring that is induced in the scenario where another query is answered 'yes', namely the 'second most likely' query according to M_m^X. This way, the complete walk is constructed.

For every $L \in D_{m,r}$ one scenario is correct. Thus the walk visits the correct value of $\chi_L^n(w_1, \dots, w_n)$. To see that the transition counts are bounded by k, note that the value of $\chi_L(w_i)$ depends only on the value of $\chi_Y^k(q_i^1, \dots, q_i^k)$. The ith bit of the walk can only be toggled if the answer for one of the queries q_i^1, ... , q_i^k changes from 'no' to 'yes'. Since there are only k queries, only k changes are possible for each position.

Construction of the Odd Language

It remains to show how that for every branch $B = \{u_1, u_2, u_3, \dots\}$ of Q^* there exist languages A and A' such that

1. A is fa-selective,
2. A' many-one reduces to ODD_A^{k+1}, and
3. A' diagonalises along B.

Let $u_{i+1} = u_i c_i$ with $c_i \in Q$. Let $\Sigma := \{1, \dots, \ell\}$ be an alphabet that contains a symbol for each index of the walk $(b_1, \dots, b_\ell)$. Let $s\colon Q \to \Sigma$ be a mapping that assigns an index $i \in \Sigma$ with $b = b_i$ to each bitstring $b \in Q$. (Several such mappings exist if the walk visits the same bitstring twice.) Let $s^*\colon Q^* \to \Sigma^*$ denote the extension of s to words. Let

$$A := \bigl\{ v \in \Sigma^* \mid v \leq_{\text{lex}} s^*\bigl(u_{|v|+1}\bigr) \bigr\}.$$

The language A is clearly fa-selective and the first requirement is hence satisfied. Let

$$A' := \bigl\{ u \, \text{bin}_n \, p \mid \\ u \in Q^*, \, p \in \{1, \dots, n\}, \, \bigl| A \cap \bigl\{ s^*(u)\, j \mid j \in \mathrm{T}_p \bigr\} \bigr| \text{ is odd} \bigr\}.$$

Recall that T_p denotes the set of indices i in the walk $(b_1, \dots, b_\ell)$ where there is a bitflip at the pth position from b_{i-1} to b_i. Figure 5-3 visualises how the languages A and A' are related.

$$\begin{tikzcd}
4\,3\,1 \in A & 010\,011\,\mathrm{bin}_3\,1 \notin A' \\
4\,3\,2 \in A \arrow[r] & 010\,011\,\mathrm{bin}_3\,2 \in A' \\
4\,3\,3 \in A \arrow[r] & 010\,011\,\mathrm{bin}_3\,3 \notin A' \\
4\,3\,4 \in A \arrow[ur] & \\
4\,3\,5 \notin A \arrow[uuuur] &
\end{tikzcd}$$

FIGURE 5-3
Example situation for the proof of theorem 5.28. The diagonalisation set is $Q = \{000, 010, 011, 110\}$. The diagonalisation branch $\{u_1, u_2, u_3, \ldots\} \subseteq Q^*$ passes the node $u_4 = 010\,011\,010$. A walk that visits all elements of Q is $(000, 010, 011, 010, 110)$. Note that 010 is visited twice. The alphabet Σ is the set $\{1, 2, 3, 4, 5\}$ and the mapping $s\colon Q \to \Sigma$ is given by $s(000) = 1$, $s(010) = 4$, $s(011) = 3$, and $s(110) = 5$. On the right hand side, words associated with the node $u_3 = 010\,011$ are shown. The arrows indicate which words in Σ^* determine, via their parity with respect to A, the membership of words in A'. Since $u_4 = 010\,011\,010$ is mapped to $4\,3\,4$ by s^*, the language A contains all words in Σ^3 that are lexicographically smaller or equal to $4\,3\,4$. This yields $\chi^3_{A'}(u_3\,\mathrm{bin}_3\,1, u_3\,\mathrm{bin}_3\,2, u_3\,\mathrm{bin}_3\,3) = 010$, which is exactly what is required of languages that diagonalise along the branch, since the branch heads in the 'direction' 010 in u_3.

The language A' is easily many-one reducible to $\textsc{odd}_A^{k+1}$, since the sizes of the T_p are bounded by $k+1$. Thus the second requirement is satisfied. To prove the third requirement, for positive integers i we must show

$$\chi_{A'}^n\bigl(u_i \operatorname{bin}_n 1, \dots, u_i \operatorname{bin}_n n\bigr) = c_i.$$

By definition, $\chi_{A'}\bigl(u_i \operatorname{bin}_n p\bigr)$ is the parity of the cardinality of the set $A \cap \bigl\{s^*(u_i)\, j \mid j \in \mathrm{T}_p\bigr\}$. By definition of A, we have $s^*(u_i)\, j \in A$ iff $s^*(u_i)\, j \leq_{\text{lex}} s^*(u_{i+1})$, which is in turn equivalent to $j \leq s\bigl(c_i\bigr)$. The number of j's in T_p that satisfy this *is exactly the number of times there has been a bitflip at position p before the walk reaches c_i*. Thus the parity of this number equals the pth bit of c_i. QED

5.29 Corollary (Tantau, 1999)
For every oracle X, for every positive integer k we have

$$\mathrm{E}^{\mathrm{P}}_{(k+1)\text{-tt}}(\text{P-sel}) \not\subseteq \mathrm{R}^{\mathrm{P}^X}_{k\text{-tt}}\bigl(\mathrm{P}^X\text{-sel}\bigr).$$

Proof. Just note $\textsc{odd}_A^{k+1} \in \mathrm{E}^{\mathrm{P}}_{(k+1)\text{-tt}}(A)$. QED

For $X = \emptyset$, the above corollary is due to Hemaspaandra et al. (1996).

5.30 Corollary (Beigel et al., 2000)

$$\mathrm{R}_{1\text{-tt}}(\textsc{semirec}) \subsetneq \mathrm{R}_{2\text{-tt}}(\textsc{semirec}) \subsetneq \mathrm{R}_{3\text{-tt}}(\textsc{semirec}) \subsetneq \dots .$$

As mentioned earlier, a stronger result than theorem 5.28 can be shown. Beigel et al. (2000), see also Tantau (2000), show that the claim does not only hold for *some* fa-selective language A, but for *every* semirecursive language that is not recursive in X.

Sixth Chapter

Applications of Enumerability and Cardinality Computations

The aim of this chapter is to show that the results of the previous chapters are not isolated theorems formulated in an artificial setting called 'finite automata enumerability and cardinality computations'. Rather, they have applications in areas that are at first glance quite unrelated—which is, at least in my opinion, a desirable property of any theoretical result. Indeed, the most intriguing aspect of the results of the present chapter is their *lack* of any mentioning of cardinality computations and enumeration. The finite automata versions of the cardinality theorem and of the nonspeedup theorem have 'practical' applications that their recursive counterparts do not have.

Fact 4.1, Kummer's cardinality theorem for Turing machines, is a beautiful theorem. It can be formulated in a nice and short way. Its proof combines ideas from different areas in an elegant way. However, just like its predecessor, the nonspeedup theorem, it suffers a severe drawback: it concerns languages that are recursive, but generally computationally extremely complex.

This is rather unfortunate, since at first sight the cardinality theorem might be used to make the description and specification of algorithms more economic: for a given problem, one might be tempted to give an algorithm that n-Turing-enumerates the function $\#_A^n$. By the cardinality theorem, this algorithm can always be transformed to an algorithm that *decides* the language A.

There are two problems with this idea. First, as discussed in section 4.4, the transformation is not constructive. The cardinality theorem and even the nonspeedup theorem only show that there *exists* a decision algorithm for A, they do not explicitly *construct* this algorithm. Rather, 'hard words' and their characteristic values appear magically in the proofs. Second, there is no correspondence between the computational complexity of an enumerator for $\#_A^n$ and the computational complexity of a decision algorithm for A. By 'thinning out' the diagonalisation language of the proof of theorem 2.29 of (Nickelsen, 1997), one can show that for every recursive function f there exists a language $A \notin \text{DTIME}[f]$ for which $\#_A^n$ is polynomial-time n-enumerable. Thus although the cardinality theorem states that A must be recursive if $\#_A^n \in \text{EN}_{\text{re}}(n)$, its computational complexity can become arbitrarily large, even if the enumerator is efficiently computable.

In the finite automata setting the situation is much more favourable—at least for $n = 2$. First, by theorem 4.30 the finite automata cardinality theorem is *constructive*. More precisely, its fair construction problem can be solved effectively. Second, it states that A must be *regular* if $\#_A^2 \in \text{EN}_{\text{fa}}(2)$.

From a practical (but also from a theoretical) point of view it is often more 'useful' to know that a language is regular than knowing that it is recursive—especially if there exists a constructive algorithm for obtaining a finite automaton that decides the language.

Each of the three sections of this chapter presents a different situation (or 'setting') in which the previous chapter's results can be applied. The focus is on the finite automata versions of these results, but other computational models are also considered. While in the last two sections 'practical' settings are discussed, the first section argues that results on cardinality computations are 'separability results in disguise'.

The first setting, introduced in section 6.1, rephrases the finite automata cardinality theorem for two words in terms of disjoint supersets of regular relations. It is shown that a language A must be regular if there exist disjoint regular supersets of the relations $A \times A$, $A \times \bar{A}$, and $\bar{A} \times \bar{A}$. 'Disjoint' means that the intersection of the three sets is empty. The restricted cardinality theorem can also be rephrased in terms of fa-separability: A must be regular if the sets $A^{(n)}$ and $\bar{A}^{(n)}$ are fa-separable. Recall that $A^{(n)}$ is the set of all n-tuples of pairwise different words in A. Kummer's original cardinality theorem can also be rephrased in an equivalent way, based on disjoint recursively enumerable supersets.

In the second setting, introduced in section 6.2, a finite automaton is used to monitor n data lines. The class of protocols that can be checked in such a way is severely restricted by the limited capabilities of finite automata. For example, since finite automata cannot count, they are unable to monitor even a simple protocol in which the number of 'send' tokens must match the number of 'acknowledge' tokens. In order to increase the number of checkable protocols, a special purpose hardware is allowed to 'help' the automaton. An example of such a hardware might be a counter. For this setting, it is discussed how much information must flow from the special purpose hardware to the automaton for nonregular protocols. Using the nonspeedup theorem for finite automata, corollary 4.11, I show that $\lfloor \log_2 n \rfloor + 1$ bits is a lower bound on the amount of this information.

The third setting, introduced in section 6.3, concerns the new notion of *classification with the help of examples.* We are given objects (formalised as words) and are asked to classify these objects according to some partition of the universe of objects. For a complicated partition this classification may be too difficult to perform in the computational model under consideration. For example a finite automaton will only be able to classify words if every equivalence class of the partition is regular. Likewise, a Turing machine will only be able to classify words if every equivalence class is recursive. In order to facilitate the classification procedure, together with the input object we provide n further (different) objects that are known to have the same classification as the input object. This problem will be called the 'classification problem with n examples'. In the recursive setting additional examples do not help. In the finite automata setting additional examples do not help if there are only two equivalence classes (for three and more equivalence classes the problem is left open). In the polynomial-time setting additional examples *do* help and allow us to decide arbitrarily complex languages.

Section 6.1

Cardinality Computations and Separable Sets

I claim that results on cardinality computations are separability results in disguise. This section explains what is meant by this. The cardinality theorem, the cardinality conjecture, and weaker forms of the cardinality theorem are rephrased as separability theorems. The rephrasings no longer refer either to enumerability or to cardinality computations, but only to regular or recursively enumerable relations and separability. To give an example: the cardinality theorem is equivalent to the statement that A must be recursive if there exist recursively enumerable supersets of $A^{\binom{n}{0}}$, … , $A^{\binom{n}{n}}$ whose intersection is empty. Recall that $A^{\binom{n}{k}}$ is the set of all n tuples of pairwise different words such that exactly k of them are in A.

Separability by finite automata was studied already in section 5.3, where we studied whether for difficult languages there exist closely related fa-separable relations. In the present section we study whether the fa-separability of relations can enforce the separability or regularity of related languages. For example, it is shown that if $A^{(n)}$ and $\bar{A}^{(n)}$ are fa-separable, then A must be regular. Compare this with the result from section 5.3 that $A^{(n)}$ and $B^{(n)}$ can be fa-separable without A and B being fa-separable.

In the following, two notions of separability for *multiple* sets are introduced. For pairs of sets these notions coincide with the usual notion of separability of sets. Lemma 6.3 shows how the enumerability of the cardinality function is linked to separability. The rest of the section lists rephrasings of results of previous chapters in terms of separability.

Separability for Multiple Sets

We start with a generalisation of the notion of separability to multiple sets A_1, … , A_k. There are two different ways of defining such a generalisation. First, one can require that there exist pairwise disjoint supersets of the sets A_i. For example, any three subsets of the sets $\{0\}^+$, $\{1\}^+$, and $\{2\}^+$ are *separable by pairwise disjoint regular sets.*

A second, less restrictive way of defining separability of multiple sets is the following: let us once more consider supersets of the sets A_i, but we only require that these sets are disjoint, but not necessarily pairwise disjoint. Recall that multiple sets are called 'disjoint' if their intersection is empty. In this case, the sets A_i are called *separable by disjoint regular sets.* This is a weak condition: if two sets A_1 and A_2 are separable, then A_1, A_2, A_3, ... , A_n are separable by disjoint sets for all sets A_3, ... , A_n. In particular, sets that are separable by disjoint sets must be disjoint themselves, but not necessarily pairwise disjoint.

6.1 Definition of Separability by (Pairwise) Disjoint Sets
Let C be a class of sets and let $k \geq 2$. Sets $A_1, \dots, A_k$ are *separable by disjoint sets in C* if there exist supersets $B_i \supseteq A_i$ with $B_i \in C$ for $i \in \{1, \dots, k\}$ such that $B_1 \cap \cdots \cap B_k = \emptyset$. If, in addition, the sets B_i are pairwise disjoint, the sets A_i are *separable by pairwise disjoint sets in C.*

Two languages are recursively separable iff they are separable by disjoint recursive sets in the sense of the above definition. It is well-known that there exist disjoint recursively enumerable languages that are recursively inseparable, see (Rosser, 1936). Indeed, this is true for any number of sets as the following theorem shows, whose proof is adapted from the proof in (Odifreddi, 1989) for two languages. Note that the theorem claims that there exist sets that cannot be separated by *disjoint* sets, which is much stronger than claiming that the sets cannot be separated by *pairwise disjoint* sets. The theorem is intended to demonstrate the usefulness of the concept of separability by disjoint sets. It will not be used in other proofs.

6.2 Theorem
For every $k \geq 2$ there exist pairwise disjoint, recursively enumerable languages A_1, ... , A_k that are not separable by disjoint recursive sets.

Proof. For $i \in \{1, \dots, k\}$ let A_i contain the codes $\operatorname{bin} M$ of all Turing machines M that halt on input $\operatorname{bin} M$ and output i. Clearly, these sets are recursively enumerable and pairwise disjoint. Suppose there exist recursive sets $B_i \supseteq A_i$ whose intersection is empty. Define a function f by $f(w) := \min\bigl\{i \in \{1, \dots, k\} \mid w \notin B_i\bigr\}$. Since the intersection of the B_i's is empty, this is a well-defined recursive function.

Let M be a Turing machine that computes f and consider the output i of this machine M on input $\operatorname{bin} M$. This output is the smallest number i such that $\operatorname{bin} M \notin B_i$. In particular, $\operatorname{bin} M \notin A_i$. By definition, this means that M does not output i on input $\operatorname{bin} M$, which is a contradiction. QED

The following lemma relates cardinality computations to separability. The rephrasing of the cardinality theorem and the cardinality conjecture in terms of separability is based on this lemma.

6.3 Lemma
Let n be a positive integer and let C be weakly closed. Then for every set A the following equivalences hold:

1. $\#_A^2 \in \mathrm{EN}_C(2)$ *iff $A \times A$, $A \times \bar{A}$, and $\bar{A} \times \bar{A}$ are separable by disjoint relations in C.*
2. $\chi_A^2 \in \mathrm{EN}_C(3)$ *iff $A \times A$, $A \times \bar{A}$, $\bar{A} \times A$, and $\bar{A} \times \bar{A}$ are separable by disjoint relations in C.*
3. $\#_A^n \in \mathrm{EN}_C(n)$ *via a relation that never enumerates both 0 and n iff $A^{(n)}$ and $\bar{A}^{(n)}$ are separable by disjoint relations in C.*
4. $\#_A^n \in \mathrm{EN}_C(n)$ *iff $A^{\binom{n}{0}}, \dots, A^{\binom{n}{n}}$ are separable by disjoint relations in C.*

Proof. For the proofs of the first, third, and fourth equivalence we can focus on tuples of pairwise distinct elements. The reason is that, given a relation $\tilde{R}$ that n-enumerates $\#_A^n$ for pairwise different elements, we can turn it into a relation R that n-enumerates $\#_A^n$ for all elements as follows:

$$\begin{aligned} i \in R[x_1, \dots, x_n] :\iff \\ & \big[\big(\textstyle\bigvee_{i \neq j} x_i = x_j\big) \wedge .\, i = 0 \vee \dots \vee i = n-1\big] \\ \vee\ & \big[\big(\textstyle\bigwedge_{i \neq j} x_i \neq x_j\big) \wedge i \in \tilde{R}[x_1, \dots, x_n]\big]. \end{aligned}$$

The formula '$x_i \neq x_j$' is an abbreviation for the positive formula '$x_i < x_j \vee x_j < x_i$', where $<$ is an irreflexive well-ordering in C.

The First Equivalence
Assume that there exist disjoint supersets $B_2 \supseteq A \times A$, $B_1 \supseteq A \times \bar{A}$, and $B_0 \supseteq \bar{A} \times \bar{A}$ with $B_0, B_1, B_2 \in C$. Consider the following relation $\tilde{R} \in C$:

$$\begin{aligned} i \in \tilde{R}[x, y] :\iff \quad & \big(i = 0 \wedge .\, B_0(x, y) \wedge B_0(y, x)\big) \\ \vee\ & \big(i = 1 \wedge .\, B_1(x, y) \vee B_1(y, x)\big) \\ \vee\ & \big(i = 2 \wedge .\, B_2(x, y) \wedge B_2(y, x)\big). \end{aligned}$$

The cardinality of $\tilde{R}[x, y]$ is always bounded by 2, since $\tilde{R}[x, y] = \{0, 1, 2\}$ would imply either $(x, y) \in B_0 \cap B_1 \cap B_2$ or $(y, x) \in B_0 \cap B_1 \cap B_2$. Both are impossible by the assumption that the sets B_i are disjoint. Next, we have $\#_A^2(x, y) \in \tilde{R}[x, y]$ for all elements x and y with $x \neq y$: If $x, y \in \bar{A}$, then both $(x,y) \in B_0$ and $(y,x) \in B_0$ and thus $0 \in \tilde{R}[x,y]$. Likewise, if $x, y \in A$, then $2 \in \tilde{R}[x,y]$. If exactly one of x and y is in A, then either $(x,y) \in B_1$ or $(y,x) \in B_1$ and thus $1 \in \tilde{R}[x,y]$.

For the other direction let $\#_A^2 \in \mathrm{EN}_C(2)$ via R. Then the following sets are disjoint supersets of $A \times A$, $A \times \bar{A}$, and $\bar{A} \times \bar{A}$, respectively:

$$\begin{aligned}
(x,y) \in B_2 &:\Longleftrightarrow 2 \in R[x,y] \vee x = y,\\
(x,y) \in B_1 &:\Longleftrightarrow 1 \in R[x,y] \wedge x \neq y,\\
(x,y) \in B_0 &:\Longleftrightarrow 0 \in R[x,y] \vee x = y.
\end{aligned}$$

The Second Equivalence

Assume $B_{00} \supseteq \bar{A} \times \bar{A}$, $B_{01} \supseteq \bar{A} \times A$, $B_{10} \supseteq A \times \bar{A}$, and $B_{11} \supseteq A \times A$ for disjoint $B_b \in C$ with $b \in \{00, 01, 10, 11\}$. Then $\chi_A^2 \in \mathrm{EN}_C(3)$ via the relation R defined by

$$b \in R[x,y] :\Longleftrightarrow \bigvee_{b' \in \{0,1\}^2} . \, b' = b \wedge B_{b'}(x,y).$$

For the other direction assume $\chi_A^2 \in \mathrm{EN}_C(3)$ via a relation R. For $b \in \{00, 01, 10, 11\}$ define B_b by $(x,y) \in B_b :\Longleftrightarrow b \in R[x,y]$. The B_b are disjoint since R is 3-bounded, and $B_{00} \supseteq \bar{A} \times \bar{A}$, $B_{01} \supseteq \bar{A} \times A$, $B_{10} \supseteq A \times \bar{A}$, and $B_{11} \supseteq A \times A$.

The Third Equivalence

Assume $A^{(n)} \subseteq B \in C$, $\bar{A}^{(n)} \subseteq B' \in C$, and $B \cap B' = \emptyset$. Then '$\#_A^n \in \mathrm{EN}_C(n)$ for pairwise different elements' via the following relation:

$$\begin{aligned}
i \in \tilde{R}[x_1, \dots, x_n] :\Longleftrightarrow \quad & \big(i = 0 \wedge B'(x_1, \dots, x_n)\big)\\
\vee & \big(i = 1 \vee \dots \vee i = n-1\big)\\
\vee & \big(i = n \wedge B(x_1, \dots, x_n)\big).
\end{aligned}$$

The defining formula states: 'enumerate $\{1, \dots, n-1\}$ plus the numbers 0 and n, depending on whether $(x_1, \dots, x_n)$ is an element of B', respectively B'. Note that 0 and n are never both elements of $\tilde{R}[x_1, \dots, x_n]$.

For the other direction assume $\#_A^n \in \mathrm{EN}_C(n)$ via a relation $R \in C$ that never enumerates both 0 and n. The separating disjoint sets $B, B' \in C$ with $A^{(n)} \subseteq B$ and $\bar{A}^{(n)} \subseteq B'$ can be defined as follows:

$$\begin{aligned}
(x_1, \dots, x_n) \in B &:\Longleftrightarrow n \in R[x_1, \dots, x_n],\\
(x_1, \dots, x_n) \in B' &:\Longleftrightarrow 0 \in R[x_1, \dots, x_n].
\end{aligned}$$

The Fourth Equivalence

Assume that disjoint sets $B_0, \dots, B_n \in C$ are given with $B_i \supseteq A^{\binom{n}{i}}$. Then '$\#_A^n \in \mathrm{EN}_C(n)$ for pairwise different elements' via the relation

$$i \in \tilde{R}[x_1, \dots, x_n] :\Longleftrightarrow \bigvee_{i' \in \{0, \dots, n\}} . \, i' = i \wedge B_{i'}(x_1, \dots, x_n).$$

The relation $\tilde{R}$ never enumerates more than n different numbers, because the B_i are disjoint. We have $\#^n_A(x_1, \dots, x_n) \in \tilde{R}[x_1, \dots, x_n]$ for all pairwise different elements, since $\#^n_A(x_1, \dots, x_n) = i$ implies $(x_1, \dots, x_n) \in A^{\binom{n}{i}} \subseteq B_i$ and thus $i \in \tilde{R}[x_1, \dots, x_n]$.

Finally, assume $\#^n_A \in \mathrm{EN}_C(n)$ via a relation R. Define sets B_i for $i \in \{0, \dots, n\}$ by $(x_1, \dots, x_n) \in B_i :\Longleftrightarrow i \in R[x_1, \dots, x_n]$. These relations do the trick. QED

The equivalences of the above lemma allow us to rephrase numerous results on cardinality computations as separability results. The following theorems and conjecture are rephrasings of, in order, fact 4.1, corollary 4.17, corollary 4.21, and conjecture 4.6.

6.4 Cardinality Theorem (Separability Version)
Let n be a positive integer and let A be a language. If $A^{\binom{n}{0}}$, ..., $A^{\binom{n}{n}}$ are separable by disjoint recursively enumerable sets, then A is recursive.

6.5 Cardinality Theorem for Two Words (Separability Version)
Let A be a language. If $A \times A$, $A \times \bar{A}$, and $\bar{A} \times \bar{A}$ are separable by disjoint regular sets, then A is regular.

6.6 Restricted Cardinality Theorem (Separability Version)
Let n be a positive integer and let A be a language. If $A^{(n)}$ and $\bar{A}^{(n)}$ are fa-separable, then A is regular.

6.7 Cardinality Conjecture (Separability Version)
Let n be a positive integer and let A be a language. If $A^{\binom{n}{0}}$, ..., $A^{\binom{n}{n}}$ are separable by disjoint regular sets, then A is regular.

The next theorems combine lemma 6.3 with the results on Presburger arithmetic. They are rephrasings of corollaries 4.22 and 4.18.

6.8 Theorem
Let $A \subseteq \mathbb{N}$ and let ϕ and ψ be formulæ in Presburger arithmetic with the same n free variables. Let $\phi \wedge \psi$ be unsatisfiable, but let $\phi(x_1, \dots, x_n)$ be true for all pairwise different $x_i \in A$, and let $\psi(x_1, \dots, x_n)$ be true for all pairwise different $x_i \notin A$. Then A is definable in Presburger arithmetic.

6.9 Theorem
Let $A \subseteq \mathbb{N}$ and let ϕ, ψ, and ρ be formulæ in Presburger arithmetic with two free variables x and y. Let $\phi \wedge \psi \wedge \rho$ be unsatisfiable, but let $\phi(x, y)$ be true for $x, y \in A$, let $\psi(x, y)$ be true for $x \in A$ and $y \in \bar{A}$, and let $\rho(x, y)$ be true for $x, y \in \bar{A}$. Then A is definable in Presburger arithmetic.

The separating sets used in the separability version of the cardinality theorem, theorem 6.4 above, are recursively enumerable. Clearly, if we require these sets to be even recursive, the theorem is also true. But what happens if we allow them to be co-recursively enumerable? It turns out that the theorem is also true in this case. The proof is based on an old idea that, according to Odifreddi (1989), is due to Sierpinski (1924) and Laventrieff (1925).

6.10 THEOREM
Let n be a positive integer and let A be a language.If $A^{\binom{n}{0}}, \dots, A^{\binom{n}{n}}$ are separable by disjoint co-recursively enumerable sets, then A is recursive.

Proof. It suffices to show that if there exist disjoint co-recursively enumerable supersets B_i of any sets $A_1, \dots, A_k$, then there also exist disjoint *recursive* supersets C_i of $A_1, \dots, A_k$.

The sets C_i are defined as follows: On input w, run the 'co-recursive enumerators' for the sets B_j for $j \in \{1, \dots, k\}$. Sooner or later, one of these enumerators must reject the word w since the B_j are disjoint. If the co-enumerator for B_i is the first to reject the word, let $w \notin C_i$, if some other enumerator rejects it first, let $w \in C_i$.

The sets C_i are clearly recursive. To see that they are disjoint, consider any word w. Let j be the index of the set B_j that is the first for which w is rejected by the co-enumerator for B_j. Then $w \notin C_j$. To see $A_i \subseteq C_i$, consider any word $w \in A_i$. Since $A_i \subseteq B_i$, the word will *never* be rejected by the co-enumerator for B_i, and thus $w \notin C_i$ is impossible. QED

SECTION 6.2

Finite Automata Protocol Monitors

Suppose we are asked to construct a testing device that monitors n signal lines. The device is synchronously fed as input the n symbols currently transported on the different lines. It should check whether all symbol streams are *valid* with respect to some protocol. When the streams end, the device should tell us on which lines the protocol was not adhered to.

A simple protocol might be 'the stream may not contain the same symbol four times in succession'. If the symbols on the streams represent voltages, this protocol might be used to verify that the signal lines are free of direct current. A more complicated protocol is 'the stream consists of valid CD-ROM sectors'. It might be used to decide for which sectors a time-consuming error correction is necessary. The following definition formalises the setting.

6.11 DEFINITION OF THE PROTOCOL MONITORING FUNCTION
A *token* is an indivisible entity that is transported on a data line. The *token alphabet* is the set of all possible tokens that can be transported. A *stream* is

a word over the token alphabet. A *protocol* is a set of valid streams, that is, a language over the token alphabet. The *protocol monitoring function for n streams* maps any n words over the token alphabet to their characteristic string with respect to the protocol.

6.12 EXAMPLE: MANCHESTER CODE
The Manchester code, see for example (Illingworth, 1983), encodes a sequence of bits in such a way that at most two bits in succession have the same value and that 0's and 1's appear equally often. This is achieved by coding 0 as 01 and 1 as 10. For example, 00110 is encoded as 0101101001. The protocol of the Manchester code is the set of all bitstrings that are valid Manchester encodings of bitstrings. For example, 0101101001 is an element of this protocol, whereas the streams 0111 and 101 are not.

The advantage of encoding a bitstring in Manchester code is that the encoded stream is *self-clocked* or *self-synchronised* and it is free of direct current. To illustrate the self-clocking property, imagine that you wish to transfer once 1 000 000 zeros at one megahertz and once 1 000 001 zeros. If the zeros are transmitted directly without reencoding, the receiver measures a low voltage for one second in the first case and a low voltage for one second and one millionth of a second in the second case. In order to tell the difference, the receiver and the sender have to synchronise their clocks extremely well. Opposed to this, for the Manchester code, in the first case the receiver measures 1 000 000 times a switch from low voltage to high voltage, directly followed by a switch back; whereas in the second case the receiver measures this double switch 1 000 001 times. The clock of the receiver must now only be able to differentiate between one zero and two zeros, not between 1 000 000 zeros and 1 000 001 zeros.

We can use a DFA with output as introduced in definition 2.3 to compute the Manchester code monitoring function for n streams. For complicated protocols, where the set of valid streams is not regular, we cannot hope to use a finite automaton for the monitoring. This is rather unfortunate since the high speed used on most signal lines typically forces the use of a simple online device—like a finite automaton. Examples of more complicated protocols are the Internet protocol (Postel, 1981) and the set of all valid HTML streams (Raggett et al., 1999).

Salvage by Advice Bits

To salvage the idea of using finite automata in such situations, one might attempt to employ a mixed strategy: we use a finite automaton *plus another simple special purpose hardware that also monitors the signal lines.* For example, such a special purpose hardware might be a counter that is increased every time a ‘send’ tag is transported on some line, and decreased every time an ‘acknowledge’ tag is transported. At some point, the special hardware device could communicate some information to the finite automaton, which should then decide which signal lines were faulty. The special hardware might tell the automaton whether the ‘send’ and ‘acknowledge’ symbols paired up correctly, or it could tell the automaton the index of a line where this was not the case. The following definition formalises this ‘mixed strategy’.

6.13 Definition
Let FC be a class of functions. A function f can be *FC-computed with k bits of advice per word* if there is a function $g \in FC$ such that for all words w there exists a bitstring $b \in \{0,1\}^k$ with $f(w) = g\bigl(\langle w, b\rangle\bigr)$.

Surely a special hardware should allow us to compute the protocol monitoring function of some nonregular protocols. However, the finite automata nonspeedup theorem tells us *that the special hardware must communicate at least $\lfloor \log_2 n\rfloor + 1$ bits to the automaton.* In particular, even getting the index of one (possibly faulty) line does not suffice to compute the monitoring function of *any* nonregular protocol.

6.14 Logarithmic Lower Communication Bound Theorem
Let A be a nonregular protocol and n a positive integer. Then A’s protocol monitoring function for n streams cannot be computed by finite automata with only $\lfloor \log_2 n\rfloor$ bits of advice per tuple.

Proof. Suppose we could compute χ_A^n with $\lfloor \log_2 n\rfloor$ bits of advice per tuple. Then $\chi_A^n \in \mathrm{EN}_{\mathrm{fa}}\bigl(2^{\lfloor \log_2 n\rfloor}\bigr) \subseteq \mathrm{EN}_{\mathrm{fa}}(n)$. By corollary 4.11 this implies that A is regular, contrary to the assumption. QED

Salvage by Massive Failure Detection

A different way of trying to salvage the idea of using finite automata is to weaken the requirement that the automaton must output the exact set of faulty lines. For example, we might require that the automaton must only detect *massive failures.* This means that on input of any n distinct streams, the automaton must accept if all streams are valid and must reject if all streams are invalid. If some streams are valid and some invalid or if some streams are identical, the automaton may accept or reject. Thus, the automaton is only required to detect a *massive* failure. The following theorem shows that massive failure detection is not easier than individual failure detection.

6.15 Massive Failure Detection Theorem
Let A be a protocol and let n be a positive integer. Suppose there exists an automaton that accepts all n tuples of pairwise different streams in A and

that rejects all n tuples of pairwise different streams not in A. Then A is regular.

Proof. Suppose such an automaton M exists for a protocol A. Then the relation accepted by M separates $A^{(n)}$ and $\bar{A}^{(n)}$. Thus, by theorem 6.6, A is regular. QED

Section 6.3

Classification with Examples

Suppose we are given objects and are asked to classify these objects according to some property. For example, we might be asked to classify objects according to their colour. On input of an object we would then have to output a colour classification like 'red' or 'green' or 'black'.

In this section we investigate whether this classification problem gets any simpler if we are provided, together with the input object, further objects that are guaranteed to be classified the same way. This is not the case for Turing machines and, if there are only two possible classifications, neither for finite automata. Opposed to this, for polynomially time-bounded Turing machines extra examples allow the classification of problems that cannot be classified without. These problems can even become arbitrarily complex computationally. The same is true for multitape automata that may move their heads at different speeds.

A formal definition of the classification problem is given below. Objects are modelled as words. The classification is performed according to a partition of Σ^*.

6.16 Definition of Classification Problems
A *classification problem* is a partition $A_1, \dots, A_k$ of Σ^*. The associated *classification function* maps every word $w \in \Sigma^*$ to the index i for which $w \in A_i$ holds.

A simple classification problem is given by $\{w \in \{0,1\}^* \mid w \text{ contains no } 0\}$, $\{w \in \{0,1\}^* \mid w \text{ contains exactly one } 0\}$, and $\{w \in \{0,1\}^* \mid w \text{ contains at least two } 0\text{'s}\}$. A more complicated classification problem is the classification of bitstrings according to whether they contain more, equally many, or less 0's than 1's.

Classification problems are, roughly spoken, 'as difficult as languages'. For example, a finite automaton can compute the classification function of a classification problem $A_1, \dots, A_k$ iff all A_i are regular. Analogously, a Turing machine can compute the classification function iff all A_i are recursive. Thus we cannot hope to solve a classification problem if some of the involved partitions are not regular, respectively not recursive.

As in the previous section, we allow some sort of 'external help' that is intended to help us in solving the classification problem for, say, nonregular sets using a finite automaton. The idea is to provide, together with the input object, n further (different) objects that have the same classification as the input object. For the above example of classifying objects according to their colour, on input of a red ball, some external agent would provide further examples of red objects like a red cube and a red hat. The automaton sees these three red objects on its tapes and should then produce the output 'red'. If we provide an incorrect input, like a red ball together with a black hat and a black cat, the automaton is allowed to get confused and may produce an arbitrary output.

6.17 Definition of Example Classification Functions
Given a classification problem $A_1, \ldots, A_k$, an *n-example classification function* is a function $f\colon (\Sigma^*)^{n+1} \to \{1, \ldots, k\}$ with the following property: for every $i \in \{1, \ldots, k\}$ and every $n+1$ pairwise different words $w_1, \ldots, w_{n+1} \in A_i$ we have $f(w_1, \ldots, w_{n+1}) = i$.

An especially important and interesting special case of the classification problem with examples is the *decision problem with examples.* An *n-example decision function* for a language A is an n-example classification function for the classification problem $A_1 = A$ and $A_2 = \bar{A}$.

In the following we study, for different computational models, whether there exist classification problems for which an n-example classification function can be computed, but whose classification function (without examples) cannot. The presentation is sorted according to increasing power of the computational models. We start with finite automata that read the tapes synchronously, that is, with the standard model studied up to now.

Classification by Finite Automata with Synchronously Moving Heads

The first theorem of this section shows that for finite automata that read their tapes synchronously examples *do not help* if there are only two equivalence classes, that is, they do not help for the decision problem. For a larger number of equivalence classes I conjecture that extra example also do not help. This conjecture is motivated by the observation that in the recursive setting, which is studied at the end of this section, extra examples do not help for any number of equivalence classes, see theorem 6.27.

6.18 Theorem
Let n be a positive integer and let A be a language. If A has a regular n-example decision function, then A is regular.

Proof. We show that $A^{(n+1)}$ and $\bar{A}^{(n+1)}$ are fa-separable, which shows that A is regular by theorem 6.6. Let f be the n-example decision function. Consider the set $B := \{(w_1, \dots, w_{n+1}) \in (\Sigma^*)^{n+1} \mid f(w_1, \dots, w_{n+1}) = 1\}$. It is regular and it has the properties $A^{(n+1)} \subseteq B$ and $\bar{A}^{(n+1)} \subseteq \bar{B}$. QED

6.19 CLASSIFICATION CONJECTURE
Let n and k be positive integers. Let $A_1, \dots, A_k$ be a classification problem that has a regular n-example classification function. Then all A_i are regular.

For recursive computations the above conjecture holds, see theorem 6.27. Unfortunately, it is not clear how its proof might be transferred to finite automata.

Classification by Finite Automata with Asynchronously Moving Heads

Before turning our attention on recursive computations, let us first increase the computational power of finite automata only a little bit: let us allow the automata to move their heads at different speeds and in different directions.

Formally, we now study deterministic, multitape, space-bounded, offline Turing machines. For a space bound $s\colon \mathbb{N} \to \mathbb{N}$, such a machine has the following components: it has n read-only input tapes on which it finds n input words; it has a work tape that stores bits and may both be read and written, but only on $s(\ell)$ cells, where ℓ is the length of the longest input word; and it has a write-only output tape on which it writes its output. Let FDSPACE$[s]$ denote the class of all functions that can be computed by such a machine with space bound s. The sought formalisation of functions computable by finite automata that may move their heads arbitrarily is FDSPACE$[0]$.

The following example presents a nonregular language that has a one-example decision function in FDSPACE$[0]$.

6.20 EXAMPLE: THE ARITHMETIC PROGRESSION
Let PROGRESSION $:= \{\operatorname{bin} 1; \operatorname{bin} 2; \operatorname{bin} 3; \operatorname{bin} 4; \dots; \operatorname{bin} n \mid n \in \mathbb{N}\}$ denote the *arithmetic progression language* (the semicolon is a marker symbol). This language can be decided in double logarithmic space, but it cannot be decided in less space, since it is not regular and since Hartmanis et al. (1965) have shown that any language decidable in less than double logarithmic space is regular. Thus the characteristic function of this language is in FDSPACE$\big[O(\log\log n)\big]$, but not in FDSPACE$[0]$.

The following theorem shows that there *is* a one-example decision function for PROGRESSION in FDSPACE$[0]$. The proof is loosely based on an idea that Frank Stephan told me about in a personal communication.

6.21 Theorem

There is a one-example decision function for PROGRESSION *in* FDSPACE[0].

Proof. Let u be an input word on the first tape that is to be decided and let v be an example put on the second tape that is classified the same way as u. Then either $u, v \in$ PROGRESSION or $u, v \notin$ PROGRESSION. Note that if v is not classified the same way as u then the following algorithm may produce a wrong output (which we are allowed to do).

We may assume that the input word u is shorter than the example v, since otherwise we can just exchange the rôles of u and v. The decision algorithm works as follows: First, it checks whether u is a prefix of v. If this is not the case, then it is impossible that both u and v are elements of PROGRESSION and u is classified as 'u is not an element of PROGRESSION'.

If u is a prefix of v, the heads are returned to the beginning of the inputs. On the second tape, the machine checks whether the first number before the first semicolon is 1. If so, it places the head for the second tape on the beginning of the first digit of the second number of this tape (which must be $\operatorname{bin} 2 = 10$ if both words are in the language). Then the machine enters a loop during which it verifies that the current number on the first tape is exactly one less than the current number on the second tape. Then it advances to the next numbers on both tapes. If the first tape ends and everything 'went fine', the machine outputs 'u is an element of PROGRESSION'; otherwise it outputs 'u is not an element of PROGRESSION'.

QED

The above theorem shows that even a single example allows us to solve a classification problem in zero extra space that we cannot solve without such an extra example. This raises the question of how powerful extra examples are for zero extra space. The following theorem states that we can 'trade space for examples'. A key idea of the proof, namely to encode logarithmically many bits of the work tape as the position of a head on a tape, is due to Hartmanis (1972).

6.22 Theorem (Trading Space for Examples)

Let n and k be positive integers. Let $A_1, \dots, A_k$ be a classification problem that has an n-example classification function in FDSPACE$[s]$. *Then there exists a number n' such that the classification problem has an n'-example classification function in* FDSPACE$[s']$ *where* $s'(\ell) = \max\{s(\ell) - \log_2 \ell, 0\}$.

Proof. Let M be a machine that uses n examples and space s. We wish to construct a machine M' that saves $\log_2 \ell$ space on inputs of maximum length ℓ by using a larger number n' of examples.

The machine M' works in two stages. In the first stage, it sorts the $n'+1$ input words according to their length. Naturally, since the input tapes are read-only, no actual reordering takes place. Rather, the reordering is done 'virtually' in the state of M': it scans all input words; keeps track of their relative lengths in its state; returns to the beginning of all the words; and from then on treats the shortest word as if it were on tape 1, the second shortest word as if it were on tape 2, and so on.

During the second stage, M' simulates M on the first $n+1$ words. The tapes $n+2$ through to $n'+1$ will be called *auxiliary tapes.* The input words on the auxiliary tapes will not be considered at all. However, the *positions of the heads on these tapes* will be used to encode the missing $\log_2 \ell$ bits of the work tape, where ℓ is the longest of the first $n+1$ words. Note that all words on the auxiliary tapes have at least length ℓ.

In detail, the simulation of M by M' works as follows: the machine M' keeps only the last $\max\{s(\ell) - \log_2 \ell, 0\}$ many bits of M's work tape on its own work tape. Whenever M reads or writes the contents of a bit stored in one of the first $\log_2 \ell$ many cells, which are 'missing' on the tape of M', the machine M' extracts this information from the head positions of the auxiliary tapes. How this information is extracted is explained below, after we have argued that the auxiliary tapes can be treated like the registers of a random access machine.

The position of the head on the ith auxiliary tape, that is, the number of cells from the left end to the current head position, is a number between 0 and at least ℓ. These numbers form an array. It is easily seen that, given enough auxiliary tapes, the machine M can perform some basic operations on this array: it can copy a number from one position in the array to another position, it can compare numbers, it can add and subtract numbers, and it can multiply and divide numbers by constants. Thus we can perform all basic operations of random access machines with a fixed number of registers that can hold the maximum number ℓ.

The information of the missing $\log_2 \ell$ bits can be stored as follows: one 'register' holds the current head position if the head is on one of the missing bits; another register holds the bits before this head, coded as a binary number; and one register holds the bits starting at the head position, also coded as a binary number, but in reverse. For example, if the contents of the missing part of the tape is $110{\rhd}01011$, then the register for the head position stores the number 4, the register for the bits left of the head stores 6 since $\operatorname{bin} 6 = 110$, and the register for the bits starting at the head position stores 26 since $(\operatorname{bin} 26)^{\text{reversed}} = 11010^{\text{reversed}} = 01011$.

With this coding, moving the head in the missing part can be simulated by doubling and halving the numbers in the registers for the tape contents of the missing part. The machine M' also notices when it reaches the left or right end of the missing part. Thus it can perform a simulation of M in space $\max\{s(\ell) - \log_2 \ell, 0\}$. QED

By repeatedly applying the above theorem we get the following corollary.

6.23 Corollary
Let n be a positive integer. If a classification problem has an n-example classification function in FDSPACE$\big[O(\log n)\big]$*, then it has an n'-example classification function in* FDSPACE$[0]$ *for some suitable n'.*

The above corollary shows that finite automata with asynchronously moving heads are just as powerful as logarithmically space-bounded Turing machines with respect to classification problems in the presence of examples. As a final example of the power of examples, I show that there exist arbitrarily complex languages that can be decided in zero space when examples are given.

6.24 Example: But-One Logarithmically Space-Bounded Languages
Let us define a class L-but-one analogously to the class P-but-one from definition 3.35 as the class of languages that are 'but one in L'. Nickelsen's proof of theorem 3.36 can be adapted to show that for every recursive function f there exists a language in L-but-one that is not in DTIME$[f]$. However, every language in L-but-one is decidable with one example: on input of u, together with an example v, decide one of them and then output the result. Together with corollary 6.23, this proves the following theorem.

6.25 Theorem
For every recursive function f, there exists a language $A \notin$ DTIME$[f]$ *and a positive integer n such that A has an n-example decision function in* FDSPACE$[0]$.

Classification by Turing Machines

I now show that conjecture 6.19 is true in the recursive setting. The proof is based on Boris Trakhtenbrot's *co-recursively enumerable tree lemma*, see lemma 6.26 below. I present a proof of this lemma since you may find it worthwhile to compare its proof with the proof of theorem 6.18 for the finite automata case. Recall that definition 5.2, where trees and branches are defined, requires that branches are always infinite.

6.26 Co-Recursively Enumerable Tree Lemma (Trakhtenbrot, 1963)
Let T be a co-recursively enumerable tree that has at least one and at most countably many branches. Then T has a recursive branch.

Proof. Let T' denote the subtree of T that contains all nodes that have infinitely many descendants. Let us call a node $u \in T'$ *good* if there is only one branch in T' that contain u. There must exist a good node: otherwise T' would either be empty, which is forbidden by assumption, or every node would have two incomparable descendants and T' would hence have uncountably many branches.

Consider a good node u and the branch B through it. Without loss of generality we can assume that u is the root, that is, the empty word. I claim that B is recursive. A word is in B iff it is in T and has infinitely many descendants. To check this, we run the following algorithm: run the co-recursive enumeration algorithm for T in a dovetailed fashion. Every time an element of $\bar{T}$ is enumerated, we mark this word as 'dead'. Furthermore, if all successors of a node are dead, the node also dies. No word in B will ever die and all words in $\bar{B}$ will die sooner or later. Thus, we can reject input words that die at some point and accept input words for which all other words of the same length die at some point. QED

6.27 Theorem
Let n and k be positive integers. Let $A_1, \ldots, A_k$ be a classification problem that has a recursive n-example classification function. Then all A_i are recursive.

Proof. Let $A_1, \ldots, A_k$ be a partition of the set Σ^*. Let $w_1, w_2, w_3, \ldots$ denote the words in Σ^* in standard ordering. Let $f\colon (\Sigma^*)^{n+1} \to \{1, \ldots, k\}$ be the n-example classification function. Define a tree T as follows:

1. The tree alphabet is the set $\Gamma := \{1, \ldots, k\}$.
2. A node $x_1 \cdots x_\ell \in \Gamma^*$ with $x_i \in \Gamma$ is in T, iff for all pairwise different indices $i_1, \ldots, i_{n+1} \in \{1, \ldots, \ell\}$ with $x_{i_1} = \cdots = x_{i_{n+1}} = j$ we have $f(w_{i_1}, \ldots, w_{i_{n+1}}) = j$.

Intuitively, the branches of the tree correspond exactly to the partitions of Σ^* for which f is an n-example classification function. More precisely, a branch $\{u_1, u_2, u_3, \ldots\}$ of T corresponds to the partition in which the word w_i is in the equivalence class A_j, where j is the last letter of u_{i+1}. The 'direction' in which a branch 'heads' at the ith node tells us in which equivalence class the ith word of Σ^* lies. The definition ensures that the branch that corresponds to the classification problem $A_1, \ldots, A_k$ is a branch of T.

I claim that there is a fixed number r such that the letters of any two nodes in T of the same length differ on at most r positions. To see this, consider any two nodes $x_1 \cdots x_\ell \in T$ and $y_1 \cdots y_\ell \in T$. Consider a graph with multiple edges whose vertex set is Γ. For each $i \in \{1, \ldots, \ell\}$ with $x_i \neq y_i$ let there be an edge from the vertex $x_i \in \Gamma$ to the vertex $y_i \in \Gamma$ with label i. Then for any two different vertices p and q of this graph there can be at most n edges going from p to q: otherwise the labels on the n edges would be indices of words for which f outputs both p and q, which is impossible. In total, there can be at most $r := nk(k-1)$ edges in the graph. This number bounds the number of positions where any two nodes in T of the same length can differ.

The tree T has at most countably many branches. To see this, fix one branch. Then every other branch is obtained by changing at most r letters at the same positions in all nodes. By lemma 6.26, this shows that T has a recursive branch. But then all other branches are also recursive, including the branch corresponding to the partition $A_1, \ldots, A_k$. This shows that each A_j is recursive, since we can decide whether $w_i \in A_j$ holds by tracing the branch up to the $(i+1)$-th node and checking whether the last letter is j. QED

Seventh Chapter

Conclusion

Please reread the thesis stated at the beginning of this dissertation. If I have not been able to convince you of this thesis, I would like to apologise for having wasted your time. In any case, I would like to thank you for having read this dissertation.

This conclusion regroups and analyses the ideas, concepts, and results of the five main chapters 'in hindsight'. First, a summary is given of which results hold for which computational models. Second, an appraisal is attempted of the relevance of the ideas, concepts, and results with respect to both theory and practice. Third, possible future work is outlined.

Section 7.1

Which Results Hold for Which Models?

Which of the main results of this dissertation hold for which computational models? The body text does not always answer this question directly, since it is organised according to proof methods, not according to computational models. Proofs that a certain theorem holds for one model and does not hold for another model are sometimes presented in different chapters. In the following I summarise which results hold for which computational models. When repeating results, only the core statement is repeated. For example, restrictions like 'n must be a positive integer' are omitted; please see the main text for detailed statements of the theorems.

I do not mention most of the results obtained in the sixth chapter on applications. This is not due to a lack of faith in their importance. Rather, these are treated as part of the next section's discussion of the relevance of the results of this dissertation.

The Cross Product Theorem

The first core result was the cross product theorem. To my knowledge, this theorem is the only known purely *structural* result on enumerability classes. (Except for the simple observation that enumerability classes form a proper hierarchy for reasonable computational models.)

3.29 Generic Cross Product Theorem
For weakly closed C, if $f \times g \in \mathrm{EN}_C(n+m)$, then either $f \in \mathrm{EN}_C(n)$ or $g \in \mathrm{EN}_C(m)$.

Since the class of recursively enumerable languages, the class of regular relations, Presburger arithmetic, first-order arithmetic, and ordinal number arithmetic are all weakly closed, the theorem holds for all of these models. Resource-bounded computations like polynomial-time computations are not weakly closed and the theorem makes no claims for them. By the discussion following theorem 3.36, the cross product theorem does not hold for resource-bounded computational models.

The cross product theorem

· holds for Presburger arithmetic,
· holds for finite automata,
· does not hold for polynomial-time computations,
· holds for recursive computations,
· holds for first-order arithmetic,
· holds for ordinal number arithmetic.

An immediate corollary of the cross product theorem was the following generic version of the generalised nonspeedup theorem.

4.8 Generic Generalised Nonspeedup Theorem
For weakly closed C we have $\mathrm{V}_C(m+h, n+k) \subseteq \mathrm{V}_C(m,n) \cup \mathrm{V}_C(h,k)$.

The generalised nonspeedup theorem is just the restriction of the cross product theorem to functions f and g that are both of the form χ_A^n, respectively χ_A^m, for some fixed language A. As the cross product theorem shows, this restriction is unnecessary. Indeed, proving the theorem for arbitrary functions f and g makes the proof even *simpler*, since one avoids having to handle big tuples of words. The generic approach adds a bit of complication to the proof, but comparing the one paragraph proof in (Tantau, 2001) for the cross product theorem for Turing machines with the longer proof given in (Beigel et al., 1995b) demonstrates this point.

The counterexample to the cross product theorem for polynomial-time computations is also a counterexample to the polynomial-time version of the generalised nonspeedup theorem. Concerning the computational models listed above, the generalised nonspeedup theorem and the cross product theorem hold for exactly the same models.

The Cardinality Theorem

The next central topic was the question of whether Kummer's cardinality theorem holds for other computational models, in particular, whether it holds for finite automata. This questions was not answered, but strong evidence was collected that suggests a positive answer.

4.1 Kummer's Cardinality Theorem
If $\#_A^n \in \mathrm{EN}_{\mathrm{re}}(n)$*, then A is recursive.*

I have shown that Kummer's cardinality theorem can be rephrased equivalently in terms of separability. This rephrasing allows some interesting modifications, like replacing the recursively enumerable supersets by co-recursively enumerable supersets, see the following two theorems.

6.4 Kummer's Cardinality Theorem (Separability Version)
If $A^{\binom{n}{0}}, \dots, A^{\binom{n}{n}}$ are separable by disjoint recursively enumerable sets, then A is recursive.

6.10 Theorem
If $A^{\binom{n}{0}}, \dots, A^{\binom{n}{n}}$ are separable by disjoint co-recursively enumerable sets, then A is recursive.

Since the nonspeedup theorem is a direct consequence of the cardinality theorem, we know that the cardinality theorem does not hold for polynomial-time computations. However, for finite automata the situation is unclear.

The cardinality theorem

· might or might not hold for Presburger arithmetic,
· might or might not hold for finite automata,
· does not hold for polynomial-time computations,
· holds for recursive computations,
· holds for first-order arithmetic,
· does not hold for ordinal number arithmetic.

Weak Cardinality Theorems

The *weak* forms of the cardinality theorem can all be proved using the generic proof method. The three weak forms are: the nonspeedup theorem, the cardinality theorem for two words, and the restricted cardinality theorem.

4.9 Generic Nonspeedup Theorem
For weakly closed C we have $\mathrm{V}_C(n,n) = \mathrm{V}_C(1,1)$.

4.16 Generic Cardinality Theorem for Two Words
For strongly closed C, if $\#_A^2 \in \mathrm{EN}_C(2)$, then $A \in C$.

4.20 Generic Restricted Cardinality Theorem
For strongly closed C, if $\#_A^n \in \mathrm{EN}_C(n)$ via a relation that never enumerates both 0 and n, then $A \in C$.

The nonspeedup theorem, the cardinality theorem for two words, and the restricted cardinality theorem

· hold for Presburger arithmetic,
· hold for finite automata,

· do not hold for polynomial-time computations,
· hold for recursive computations,
· hold for first-order arithmetic,
· do not hold for ordinal number arithmetic, except for the nonspeedup theorem, which does hold.

The rôle of ordinal number arithmetic is somewhat peculiar. The restricted cardinality theorem fails for it, because there exist ordinal numbers that are not definable in ordinal number arithmetic. However, this 'deficiency' is not a problem for the nonspeedup theorem.

For finite automata and for Turing machines, the proofs of the cardinality theorem for two words and of the restricted cardinality theorem are quite different. My proofs for finite automata are based on first-order formulæ that involve negation. The classic proofs for Turing machines are based on Kummer's recursively enumerable tree lemma (1992) and on Trakhtenbrot's co-recursively enumerable tree lemma, see lemma 6.26.

A natural question in this context is: can we give a proof of the cardinality theorem (at least for two words) that works both for finite automata and for Turing machines? I do not know whether this is the case.

Closure of Regular Relations under Elementary Definitions

Elementary definitions of regular relations are the first of the two central proof methods used in this dissertation.

2.37 Corollary
Let $\mathcal{S}$ be a regular τ-structure and ϕ a first-order τ-formula. Then the relation $\phi^{\mathcal{S}}(u_1, \dots, u_n)$ is regular.

This theorem allows us to use the formalism of first-order logic for the definition of regular relations in terms of existing ones. I made use of this method throughout the text. All instantiations of the generic theorems for finite automata depend on the above closure property of the class of regular relations.

· Presburger arithmetic is closed under elementary definitions.
· The class of regular relations is closed under elementary definitions.
· The class of polynomial-time decidable relations is not closed under elementary definitions, not even under positive elementary definitions.
· The class of recursively enumerable relations is not closed under elementary definitions, but it is closed under positive elementary definitions.

· First-order arithmetic is closed under elementary definitions.
· Ordinal number arithmetic is closed under elementary definitions, but some ordinal numbers are not elementarily definable.

The Branch Diagonalisation Method

The second central proof method of this dissertation is branch diagonalisation. It is not universally applicable (for example it can only be applied to uncountable classes), but if it is applicable it yields strong separations. A main result that was proved using branch diagonalisation is the following:

5.19 Verboseness Class Inclusion Theorem
The following statements are equivalent for all oracles X:

1. $\mathrm{V}_{\mathrm{re}}(m,n) \subseteq \mathrm{V}_{\mathrm{re}}(h,k)$.
2. $\mathrm{V}_{\mathrm{fa}}(m,n) \subseteq \mathrm{V}_{\mathrm{fa}}(h,k)$.
3. $\mathrm{V}_{\mathrm{fa}}(m,n) \subseteq \mathrm{V}^{X}_{\mathrm{re}}(h,k)$.

Once more, the theorem cannot be extended to polynomial-time computations: there exist polynomial-time $(2,2)$-verbose languages outside P, for example the languages in P-but-one.

At first sight, branch diagonalisation might seem useful only for showing separations of classes defined in terms of finite automata and perhaps Turing machines. This impression is wrong, since separations for polynomial-time computations can be obtained as simple corollaries from the strong separation results obtained by branch diagonalisation: for example, if $\mathrm{V}_{\mathrm{fa}}(m,n) \not\subseteq \mathrm{V}^{X}_{\mathrm{re}}(h,k)$, then also the much larger class $\mathrm{V}_{\mathrm{p}}(m,n) \supseteq \mathrm{V}_{\mathrm{fa}}(m,n)$ is not contained in the much smaller class $\mathrm{V}^{X}_{\mathrm{p}}(h,k) \subseteq \mathrm{V}^{X}_{\mathrm{re}}(h,k)$.

Separability Results

'Results on cardinality computations are separability results in disguise.' Two pairs of theorems support this claim, which was made at the beginning of section 6.1. While the two parts of each pair are proved in different chapters in the main text since their proofs require different proof techniques, in this conclusion I present these results as pairs. The second part of each pair shows that the first part is optimal in a certain sense.

In the main text, the results are formulated in an even more general way, namely relative to an arbitrary oracle X. For the purposes of this conclusion this extra generality seems more distracting than enlightening.

6.5 Theorem
If $A \times A$, $A \times \bar{A}$, and $\bar{A} \times \bar{A}$ are separable by disjoint regular sets, then A is regular.

5.26 THEOREM
There exists a language A that is not semirecursive, but for which $A \times A$, $A \times \bar{A}$, $\bar{A} \times A$, and $\bar{A} \times \bar{A}$ are separable by disjoint regular sets.

The beauty of the above theorems lies in the fact that they neither refer to enumerability nor to cardinality computations. Separability results are much easier to explain to 'non-specialists' since only standard terminology is used in their formulation. I taught a first-year undergraduate course in 2001 where I explained (the claim of) the above results to my students and several of them were startled and intrigued by the theorems. (Several other were quite indifferent, but that might have had to do with their lack of interest in theoretical computer science in general and automata theory in particular.)

The next two theorems form the second pair.

6.6 THEOREM
If $A^{(n)}$ and $\bar{A}^{(n)}$ are fa-separable, then A is regular.

5.23 THEOREM
There exist disjoint recursively inseparable languages A and B for which $A^{(n)}$ and $B^{(n)}$ are fa-separable for all $n \geq 2$.

SECTION 7.2

Relevance of the Main Results

A popular question asked on many lecture evaluation forms is: 'How do you rate the relevance of what was taught?' Phrased this way, the question is difficult to answer for students attending courses on theoretical computer science, because there are different forms of 'relevance' of theorems: practical relevance to programming and program specification; relevance to proofs of other, unrelated theorems; and relevance with respect to proof techniques. In the following, I try to appraise the relevance of the results obtained in this dissertation, differentiating between these different forms.

Practical Relevance

Theoretical results can be relevant to 'real programming' insofar as they propose algorithms or specification methods. For example, Kleene's fundamental result that one can turn every regular expression into a deterministic finite automaton is a theoretical result of immense practical relevance. It offers a way of turning a simple specification into a highly effective algorithm.

The first result of this dissertation that I think might have a 'practical' application is the cardinality theorem for finite automata for two words. As argued in the introduction to the sixth chapter, this theorem allows us to 'specify' a regular language using automata that are arbitrarily smaller than the smallest deterministic or nondeterministic automaton that can decide the language.

The practical relevance of this 'specification method' should certainly not be overestimated. It is, in its current form, only applicable to a narrow class of languages. As a matter of fact, it takes some pondering to come up with a language for which such a specification is possible—in the main text I give just one example, namely $\{0^k 1w \mid w \in \Sigma^*\}$ for fixed k.

One result of this dissertation refers to practical settings directly: theorem 6.14 states that for nonregular protocols the n stream protocol monitoring function cannot be computed with less than $\lfloor \log_2 n \rfloor$ bits of advice from some external source. This is a practical result, albeit a negative one. It just tells us that certain protocol checkers do not exist. Negative results only help in a rather indirect way in the construction of devices or algorithms: they inform us that trying to come up with a certain kind of algorithm is futile. This can be used as a guide for the construction, but not as a tool.

While the setting of theorem 6.14, and also of the related theorem 6.15 on massive failure detection, is practical, one can question its relevance. That is, is it really realistic that one would perform a protocol check using a finite automaton coupled with a special purpose hardware? Unfortunately, there is a good reason to believe that this is possible only in very specialised situations: The whole setting only makes sense if one transmits at most $n-1$ bits for n data lines. Thus all protocols for which the setting might be applicable are $(2^{n-1}, n)$-fa-verbose. Austinat et al. (2003) have shown that no inherently context-free language can be $(2^n - 1, n)$-fa-verbose for any n. Thus, even for simple languages like $\{0^n 1^n \mid n \in \mathbb{N}\}$ the setting cannot be used. On the other hand, this is an interesting negative result in itself. Furthermore, Austinat et al. have also shown that there exist, for example, $(2^{n-1}, n)$-fa-verbose context-sensitive languages that are not context-free.

A third practical result is theorem 6.18. It states that if a language can be decided with n examples by a finite automaton, then the language is regular. There are numerous situations, both in theory and in practice, where we are given a bunch of objects that are known to be classified in the same way. For example, a biometric access device might make numerous measurements of a fingerprint or a voice and compute a set of short signatures from them. All signatures belong to the same person and should all be classified the same way, namely ‘grant access’ or ‘deny access’.

Once more, theorem 6.18 is a negative result. It just tells us that finite automata are useless for the decision of nonregular languages, even in the presence of examples. An intriguing open problem in this context is the classification conjecture, see conjecture 6.19.

For polynomial-time computations (and even for finite automata that may move their heads arbitrarily) the situation is different. Here we can decide arbitrarily difficult problems if we have access to enough examples. However, the practical relevance of this result is once more diminished by the fact that examples can only help for polynomial-time $(2^n - 1, n)$-verbose languages, so-called *non-p-superterse languages.* It is known that many natural problems, including all NP-complete problems, the graph isomorphism problem, and the graph automorphism problem, are either in P or they are p-superterse. Thus for these natural problems examples do not help. Once more, this is an interesting negative result in itself.

To sum up, I believe that the main results have practical applications only in certain specialised situations.

Relevance to Other Proofs

Theoretical results can be relevant to other parts of theory. For example, the celebrated PCP-theorem on polynomially checkable proofs has beautiful applications in non-approximability proofs.

The nature of mathematical progress makes it hard to predict whether some, or any, of the results proved in this dissertation will be used in proofs in unrelated areas. In order to apply results from one area in another area a ‘terminological chasm’ has to be bridged. If the chasm is too great, the bridge will never be build.

There are two reasons why I think my results on finite automata enumerability classes and cardinality computations will be useful in other areas.

First, the separability results are already an example of the application of the main theorems to a setting that has nothing to do with enumerability or cardinality computations. Indeed, I found these applications only some time after the main theorems had already been proved.

Second, in the recursive setting it took some time before the cardinality theorem was used in other proofs. An example is the proof of Beigel et al. (2000) that for every nonrecursive, semirecursive languages A the language ODD_A^{k+1} is not weakly k-truth-table reducible to any semirecursive language. The proof given by Beigel et al. relies on the cardinality theorem.

Theoretical results can have an impact not so much because they have numerous corollaries, but because the proof method can be used in other situations or because the proof offers new insights into a field. An example are proofs that show that there are oracles relative to which the P-NP-problem is answered affirmatively, respectively negatively. These results are not really 'useful' since they tell us nothing about the status of the P-NP-problem 'in the real world' nor do they bring us nearer to a proof of $\mathrm{P} \neq \mathrm{NP}$. (One might argue, ironically, that they bring us further away from such a proof.) Nevertheless, oracles and relativisations are certainly relevant to theoretical computer science and have provided us with a new way of approaching problems and proofs.

I believe that the two proof techniques 'elementary definitions of regular relations' and 'branch diagonalisation' will be applicable in new situations, including topics totally unrelated to this dissertation.

In both cases my main reason for this belief is that both methods arose out of necessity, not out of curiosity. I had results on the enumerability of cardinality functions using finite automata that were difficult to prove and whose proofs were even more difficult to write down. The search for a consistent and elegant way for writing down these proofs lead me to the proof of the main closure property of the class of regular relations. Indeed, it was only afterwards that I noticed that Büchi had already (implicitly) proved this closure property. This also explains why the proof of theorem 2.36 uses a logical structure that is different from the structures previously used in the literature: it is the structure that I used before Dirk Siefkes brought Büchi's proof to my attention.

The branch diagonalisation method also arose out of necessity. I needed a way of transferring a diagonalisation proof from the recursive setting to finite automata. This seemed impossible since all standard diagonalisation methods involve some kind of simulation, which is impossible to do using only finite automata. Fortunately, at that time I was also studying the relationship of different partial information classes, a concept due to Arfst Nickelsen (2001), in the recursive setting. In particular, I was interested in the question of whether there exists a partial information class that contains the union of any n branches, but that does not contain the union of appropriately chosen $n+1$ branches (such classes exist). The proofs involved arguments that were an early form of branch diagonalisation. These arguments also worked for finite automata, but noticing this took one of those rare moments of clairvoyance.

What convinced me that branch diagonalisation is a method whose applications are not restricted to the study of verboseness was a discussion

I had with Leen Torenvliet in Rochester, New York, in September 2001. Together with Lane Hemaspaandra, Leen was busy adding the finishing touches to a book on semi-feasible algorithms; or so they thought—it took over a year before the book finally went into print (Hemaspaandra and Torenvliet, 2002). They had asked several other graduate students and myself to do some proofreading. The manuscript included a proof that the class of P-selective languages is not closed under intersection. When I read their proof, I recalled a different proof of this result, which is now presented as theorem 5.3 in this dissertation. I had found this different proof during my study of the branch diagonalisation method. Assuming that my proof was well-known since it was 'so simple' and assuming that it was the standard, obvious way of proving this, I sketched my proof on the backside as a 'correction'. (What I wrote on the backside, you can now read on page 82 of their book.) When Leen turned the page and read the backside, his startled expression made me realise that branch diagonalisation is a method that has a surprising range of applications.

Section 7.3

Outlook

In the following I list problems not solved in this dissertation that I would like to suggest for further research. In all cases I believe that even a partial solution would be valuable not only for the study of enumerability, but also for the study of other concepts of automata, complexity, and recursion theory.

Does the Cardinality Theorem Hold for Finite Automata?

For me, the most intriguing question raised in this dissertation is whether the cardinality theorem holds for finite automata. That is, I would like to know whether the following conjecture is true:

4.6 Cardinality Conjecture for Finite Automata
If $\#_A^n \in \mathrm{EN}_{\mathrm{fa}}(n)$, *then A is regular.*

Most likely, a proof of this conjecture would increase also our understanding of the cardinality theorem for Turing machines. Two possible routes to a proof seem possible to me: either, one might try to extend the proof method I used in the proof of the conjecture for $n = 2$, or one might try to adapt Kummer's proof from the recursive setting to finite automata. Unfortunately, both routes appear to be quite stony.

Can all Main Theorems be Derived from One Theorem?

I now formulate a conjecture that would imply all of the following theorems: the sum theorem (see below), the cardinality theorem, the cross product theorem, the generalised nonspeedup theorem, and the nonspeedup theorem. I formulate this conjecture once for Turing machines and once for finite automata.

Kummer and Stephan (1994) have generalised the cardinality theorem as follows: their 'sum theorem' states that a function $f\colon \mathbb{N} \to \mathbb{N}$ must be recursive if $\Sigma_f^n \in \mathrm{EN}_{\mathrm{re}}(n)$, where $\Sigma_f^n(x_1, \dots, x_n) := f(x_1) + \dots + f(x_n)$.

Recall that the nonspeedup theorem, which states that $\chi_A^n \in \mathrm{EN}_{\mathrm{re}}(n)$ implies that A is recursive, follows from the cross product theorem, which states that $f \times g \in \mathrm{EN}_{\mathrm{re}}(n+m)$ implies $f \in \mathrm{EN}_{\mathrm{re}}(n)$ or $g \in \mathrm{EN}_{\mathrm{re}}(m)$. The cross product theorem is the 'pure core' of the nonspeedup theorem. Perhaps, the sum theorem also has a 'pure core', namely the following claim: 'if $f * g \in \mathrm{EN}_{\mathrm{re}}(n+m)$ then either $f \in \mathrm{EN}_{\mathrm{re}}(n)$ or $g \in \mathrm{EN}_{\mathrm{re}}(m)$'. The star operator, which is a mixture of the addition operator and the cross product operator, is defined by $(f * g)(x, y) := f(x) + g(y)$. I have not been able to find a counterexample to this statement and believe that the following two conjectures hold:

Conjecture
If $f * g \in \mathrm{EN}_{\mathrm{re}}(n+m)$ then either $f \in \mathrm{EN}_{\mathrm{re}}(n)$ or $g \in \mathrm{EN}_{\mathrm{re}}(m)$.

Conjecture
If $f * g \in \mathrm{EN}_{\mathrm{fa}}(n+m)$ then either $f \in \mathrm{EN}_{\mathrm{fa}}(n)$ or $g \in \mathrm{EN}_{\mathrm{fa}}(m)$.

In what Situations Can We Use Branch Diagonalisation?

In this dissertation, branch diagonalisation was applied to uncountable classes defined in terms of finite automata. One might try to make branch diagonalisation applicable to a broader range of situations. First, the method might be useful also for computational models below finite automata. For example, we may ask whether we can use it for star-free languages (languages obtained from regular expressions that do not include the Kleene-star) or for diagonalisation proofs in Presburger arithmetic. If this works, one might also be able to prove new results for low complexity circuit classes like AC^0 or TC^0. Second, the method might be adaptable to countable classes by imposing further requirements on the branches. In this way, it might be possible to construct branch diagonalisations between standard complexity classes.

Bibliography

Serge Abiteboul, Richard Hull, and Victor Vianu.
Foundations of Databases.
Addison-Wesley, 1995.

Manindra Agrawal and Vikraman Arvind.
Quasi-linear truth-table reductions to P-selective sets.
Theoretical Computer Science, 158(1–2):361–370, 1996.

Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi.
Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties.
Springer-Verlag, 1999.

Holger Austinat, Volker Diekert, and Ulrich Hertrampf.
A structural property of regular frequency computations.
Theoretical Computer Science, 292(1):33–43, 2003.

Holger Austinat, Volker Diekert, Ulrich Hertrampf, and Holger Petersen.
Regular frequency computations.
In *Proceedings of the RIMS Symposium on Algebraic Systems, Formal Languages and Computation*, volume 1166 of RIMS Kokyuroku, pages 35–42. Research Institute for Mathematical Sciences, Kyoto University, Japan, 2000.

Robert Beals, Richard Chang, William I. Gasarch, and Jacobo Torán.
On finding the number of graph automorphisms.
Chicago Journal on Theoretical Computer Science, 1999(1):1–28, 1999.

Richard Beigel.
Query-Limited Reducibilities.
PhD thesis, Stanford University, USA, 1987.

Richard Beigel.
Bi-immunity results for cheatable sets.
Theoretical Computer Science, 73(3):249–263, 1990.

Richard Beigel.
Bounded queries to SAT and the Boolean hierarchy.
Theoretical Computer Science, 84(2):199–223, 1991.

Richard Beigel, William I. Gasarch, John Gill, and James C. Owings, Jr.
Terse, superterse, and verbose sets.
Information and Computation, 103(1):68–85, 1993.

Richard Beigel, William I. Gasarch, Martin Kummer, Georgia Martin, Timothy McNicholl, and Frank Stephan.
The complexity of ODD_n^A.
Journal of Symbolic Logic, 65(1):1–18, 2000.

Richard Beigel, Martin Kummer, and Frank Stephan.
Approximable sets.
Information and Computation, 120(2):304–314, 1995a.

Richard Beigel, Martin Kummer, and Frank Stephan.
Quantifying the amount of verboseness.
Information and Computation, 118(1):73–90, 1995b.

Véronique Bruyère and Georges Hansel.
Bertrand numeration systems and recognizability.
Theoretical Computer Science, 181(1):17–43, 1997.

Véronique Bruyère, Georges Hansel, Christian Michaux, and Roger Villemaire.
Logic and p-recognizable sets of integers.
Bulletin of the Belgian Mathematical Society, 1(2):191–238, 1994.

Julius Richard Büchi.
Weak second-order arithmetic and finite automata.
Zeitschrift für mathematische Logik und Grundlagen der Mathematik, 6: 66–92, 1960.

Julius Richard Büchi.
On a decision method in restricted second-order arithmetic.
In Ernest Nagel, Patrick Suppes, and Alfred Tarski, editors, *Proceedings of the 1960 International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

Tevfik Bultan, Richard Gerber, and William Pugh.
Model checking concurrent systems with unbounded integer variables: Symbolic representations, approximations, and experimental results.
ACM Transactions on Programming Languages and Systems, 21(4):747–789, 1999.

Jin-Yi Cai and Lane A. Hemachandra.
Enumerative counting is hard.
Information and Computation, 82(1):34–44, 1989.

Georg Cantor.
Über eine Eigenschaft des Inbegriffs aller reellen algebraischen Zahlen.
Journal für die reine und angewandte Mathematik, 77:258–262, 1874.

Martin Davis, editor.
The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems, and Computable Functions.
Raven Press, 1965.

Heinz-Dieter Ebbinghaus and Jörg Flum.
Finite Model Theory.
Perspectives in Mathematical Logic. Springer-Verlag, 1995.

Lance Fortnow.
The computational complexity column: Diagonalization.
Bulletin of the European Association for Theoretical Computer Science, 71:102–112, 2000.

William I. Gasarch.
Bounded queries in recursion theory: A survey.
In *Proceedings of the Sixth Annual Structure in Complexity Theory Conference*, pages 62–78. IEEE Computer Society Press, 1991.

William I. Gasarch and Georgia A. Martin.
Bounded Queries in Recursion Theory, volume 16 of Progress in Computer Science and Applied Logic.
Birkhäuser, 1999.

Kurt Gödel.
Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I.
Monatshefte für Mathematik und Physik, 38:173–198, 1931.

Kurt Gödel.
On undecidable propositions of formal mathematical systems.
Mimeographed notes by Stephen C. Kleene and Barkley Rosser of lectures at the Institute for Advanced Study, Princeton, New Jersey, 1934.
See Davis (1965) for a reprint.

Martin Grötschel, László Lovász, and Alexander Schrijver.
The ellipsoid method and its consequences in combinatorial optimization.
Combinatorica, 1(2):169–197, 1981.

Valentina Harizanov, Martin Kummer, and Jim Owings.
Frequency computations and the cardinality theorem.
Journal of Symbolic Logic, 52(2):682–687, 1992.

Juris Hartmanis.
On non-determinancy in simple computing devices.
Acta Informatica, 1:336–344, 1972.

Juris Hartmanis, Philip M. Lewis, and Richard E. Stearns.
Hierarchies of memory-limited computations.
In *Proceedings of the Sixth Annual IEEE Symposium on Switching Circuit Theory and Logic Design*, pages 179–190. IEEE Computer Society Press, 1965.

Lane A. Hemachandra.
The strong exponential hierarchy collapses.
Journal of Computer and System Sciences, 39(3):299–322, 1989.

Edith Hemaspaandra, Lane A. Hemaspaandra, and Harald Hempel.
A downward collapse within the polynomial hierarchy.
SIAM Journal on Computing, 28(2):383–393, 1998.

Lane A. Hemaspaandra, Harald Hempel, and Arfst Nickelsen.
Algebraic properties for P-selectivity.
In Jie Wang, editor, *Proceedings of the Seventh Annual International Computing and Combinatorics Conference*, volume 2108 of Lecture Notes on Computer Science, pages 49–58. Springer-Verlag, 2001.

Lane A. Hemaspaandra, Albrecht Hoene, and Mitsunori Ogihara.
Reducibility classes of P-selective sets.
Theoretical Computer Science, 155(2):447–457, 1996.

Lane A. Hemaspaandra and Zhigen Jiang.
P-selectivity: Intersections and indices.
Theoretical Computer Science, 145(1–2):371–380, 1995.

Lane A. Hemaspaandra and Leen Torenvliet.
Theory of Semi-Feasible Algorithms.
Monographs in Theoretical Computer Science. Springer-Verlag, 2002.

Albrecht Hoene and Arfst Nickelsen.
Counting, selecting, and sorting by query-bounded machines.
In Patrice Enjalbert, Alain Finkel, and Klaus W. Wagner, editors, *Proceedings of the Tenth International Symposium on Theoretical Aspects of Computer Science*, volume 665 of Lecture Notes on Computer Science, pages 196–205. Springer-Verlag, 1993.

John E. Hopcroft and Jeffrey D. Ullman.
Introduction to Automata Theory, Languages, and Computation.
Addison-Wesley, 1979.

Valerie Illingworth, editor.
Dictionary of Computing.
Oxford University Press, 1983.

Neil Immerman.
Nondeterministic space is closed under complementation.
SIAM Journal on Computing, 17(5):935–938, 1988.

Neil Immerman.
Descriptive Complexity.
Graduate Texts in Computer Science. Springer-Verlag, 1999.

Thomas Jech.
Set Theory, volume 97 of Perspectives in Mathematical Logic.
Springer-Verlag, second edition, 1997.

Carl G. Jockusch, Jr.
Reducibilities in Recursive Function Theory.
PhD thesis, Massachusetts Institute of Technology, USA, 1966.

Carl G. Jockusch, Jr.
Semirecursive sets and positive reducibility.
Transactions of the American Mathematical Society, 131:420–436, 1968.

Jim Kadin.
$\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ and sparse Turing-complete sets for NP.
Journal of Computer and System Sciences, 39(3):282–298, 1989.

Richard M. Karp and Richard J. Lipton.
Some connections between uniform and non-uniform complexity classes.
In *Proceedings of the Twelveth Annual ACM Symposium on Theory of Computing*, pages 302–309. ACM Press, 1980.

Susanne Kaufmann and Martin Kummer.
On a quantitative notion of uniformity.
Fundamenta Informaticæ, 25(1):59–78, 1996.

Efim B. Kinber.
Frequency computations in finite automata.
Cybernetics, 2:179–187, 1976.

Donald E. Knuth.
The sandwich theorem.
The Electronic Journal of Combinatorics, 1(A1):1–48, 1994.

Martin Kummer.
A proof of Beigel's cardinality conjecture.
Journal of Symbolic Logic, 57(2):677–681, 1992.

Martin Kummer and Frank Stephan.
Some aspects of frequency computation.
Technical Report 21/91, Universität Karlsruhe, Fakultät für Informatik, Germany, 1991.

Martin Kummer and Frank Stephan.
Effecitive search problems.
Mathematical Logic Quarterly, 40(2):224–236, 1994.

Stuart A. Kurtz.
A relativized failure of the Berman-Hartmanis conjecture.
Technical Report 83-001, Department of Computer Science, University of Chicago, 1983.

Alistair H. Lachlan.
A recursively enumerable degree which will not split over all lesser ones.
Annals of Mathematical Logic, 9:307–365, 1975.

M. Laventrieff.
Sur les sous-classes de la classification de M. Baire.
Comptes Rendus de l'Académie des Sciences, 180:111–114, 1925.

László Lovász.
On the Shannon capacity of a graph.
IEEE Transactions on Information Theory, 25(1):1–7, 1979.

Stephen R. Mahaney.
Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis.
Journal of Computer and System Sciences, 25(2):130–143, 1982.

Richard McNaughton.
Review of (Büchi, 1962).
Journal of Symbolic Logic, 28(1):100–102, 1963.

Arfst Nickelsen.
On polynomially $\mathcal{D}$-verbose sets.
In Rüdiger Reischuk and Michel Morvan, editors, *Proceedings of the 14th International Symposium on Theoretical Aspects of Computer Science*, volume 1200 of Lecture Notes on Computer Science, pages 307–318. Springer-Verlag, 1997.

Arfst Nickelsen.
Polynomial Time Partial Information Classes.
Wissenschaft und Technik Verlag, 2001.
Dissertation, Technische Universität Berlin, 1999.

Arfst Nickelsen and Till Tantau.
On reachability in graphs with bounded independence number.
In Oscar H. Ibarra and Louxin Zhang, editors, *Proceedings of the Eighth Annual International Computing and Combinatorics Conference*, volume 2387 of Lecture Notes on Computer Science, pages 554–563. Springer-Verlag, 2002.

Arfst Nickelsen and Till Tantau.
Partial information classes.
SIGACT News, 34(1):32–46, 2003.

Piergiorgio Odifreddi.
Classical Recursion Theory I, volume 125 of Studies in Logic and the Foundations of Mathematics.
North-Holland, 1989.

Piergiorgio Odifreddi.
Classical Recursion Theory II, volume 143 of Studies in Logic and the Foundations of Mathematics.
North-Holland, 1999.

Mitsunori Ogihara.
Polynomial-time membership comparable sets.
SIAM Journal on Computing, 24(5):1068–1081, 1995.

Mitsunori Ogihara and Till Tantau.
On the reducibility of sets inside NP to sets with low information content.
Technical Report 2002-782, Computer Science Department, University of Rochester, 2002.

Derek C. Oppen.
A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic.
Journal of Computer and System Sciences, 16(3):323–332, 1978.

James C. Owings, Jr.
A cardinality version of Beigel's nonspeedup theorem.
Journal of Symbolic Logic, 54(3):761–767, 1989.

Dominique Perrin.
Finite automata.
In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 1, pages 1–57. Elsevier, 1990.

Emil L. Post.
Recursively enumerable sets of positive integers and their decision problems.
Bulletin of the American Mathematical Society, 50:284–316, 1944.

Jon Postel.
Internet protocol—DARPA internet program protocol specification.
Technical report, Information Sciences Institute, University of Southern California, 1981.

Mojżesz Presburger.
Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt.
In *Comptes Rendus du Premier Congrès des Mathématiciens des Pays Slaves*, pages 92–101, Warsaw, Poland, 1929.

Pavel Pudlák.
On reducibility and symmetry of disjoint NP-pairs.
In Jiří Sgall, Aleš Pultr, and Petr Kolman, editors, *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science*, volume 2136 of Lecture Notes on Computer Science, pages 621–632. Springer-Verlag, 2001.

Willard von Orman Quine.
Concatenation as a basis for arithmetic.
Journal of Symbolic Logic, 11(4):104–114, 1946.

Michael O. Rabin and Dana Scott.
Finite automata and their decision problems.
IBM Journal of Research and Development, 3(2):114–125, 1959.

Dave Raggett, Arnaud Le Hors, and Ian Jacobs, editors.
HTML 4.01 Specification, W3C Recommendation, www.w3c.org, 24th December 1999. World Wide Web Consortium.

Barkley J. Rosser.
Extensions of some theorems of Gödel and Church.
Journal of Symbolic Logic, 1(1):87–91, 1936.

Arto Salomaa.
Theory of Automata, volume 100 of Monographs in Pure and Applied Mathematics.
Pergamon Press, 1969.

Alan L. Selman.
P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP.
Mathematical Systems Theory, 13:55–65, 1979.

Alan L. Selman.
Much ado about functions.
In *Proceedings of the Eleventh Annual IEEE Conference on Computational Complexity*, pages 198–212. IEEE Computer Society Press, 1996.

Waclaw Sierpinski.
Sur une propriété des ensembles ambigus.
Fundamenta Mathematicæ, 6:1–5, 1924.

Howard Straubing.
Finite Automata, Formal Logic, and Circuit Complexity.
Progress in Theoretical Computer Science. Birkhäuser, 1994.

Robert Szelepcsényi.
The method of forced enumeration for nondeterministic automata.
Acta Informatica, 23(3):279–284, 1988.

Till Tantau.
Combinatorial representations of partial information classes and their truth-table closures.
Diploma thesis, Technische Universität Berlin, Germany, 1999.

Till Tantau.
On the power of extra queries to selective languages.
Technical Report 00-077, Electronic Colloquium on Computational Complexity, www.eccc.uni-trier.de/eccc, 2000.

Till Tantau.
A note on the complexity of the reachability problem for tournaments.
Technical Report 01-092, Electronic Colloquium on Computational Complexity, www.eccc.uni-trier.de/eccc, 2001.

Till Tantau.
Comparing verboseness for finite automata and Turing machines.
In Helmut Alt and Afonso Ferreira, editors, *Proceedings of the 19th International Symposium on Theoretical Aspects of Computer Science*, volume 2285 of Lecture Notes on Computer Science, pages 465–476. Springer-Verlag, 2002A.

Till Tantau.
Towards a cardinality theorem for finite automata.
In Krzysztof Diks and Wojciech Rytter, editors, *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science*, volume 2420 of Lecture Notes on Computer Science, pages 625–636. Springer-Verlag, 2002B.

Boris A. Trakhtenbrot.
On the frequency computation of functions.
Algebra i Logika, 2:25–32, 1963.
In Russian.

Alan M. Turing.
On computable numbers with an application to the Entscheidungsproblem.
Proceedings of the London Mathematical Society, 42:230–265, 1936.

Pierre Wolper and Bernard Boigelot.
On the construction of automata from linear arithmetic constraints.
In Susanne Graf and Michael Schwartzbach, editors, *Proceedings of the Sixth International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of Lecture Notes on Computer Science, pages 1–19. Springer-Verlag, 2000.