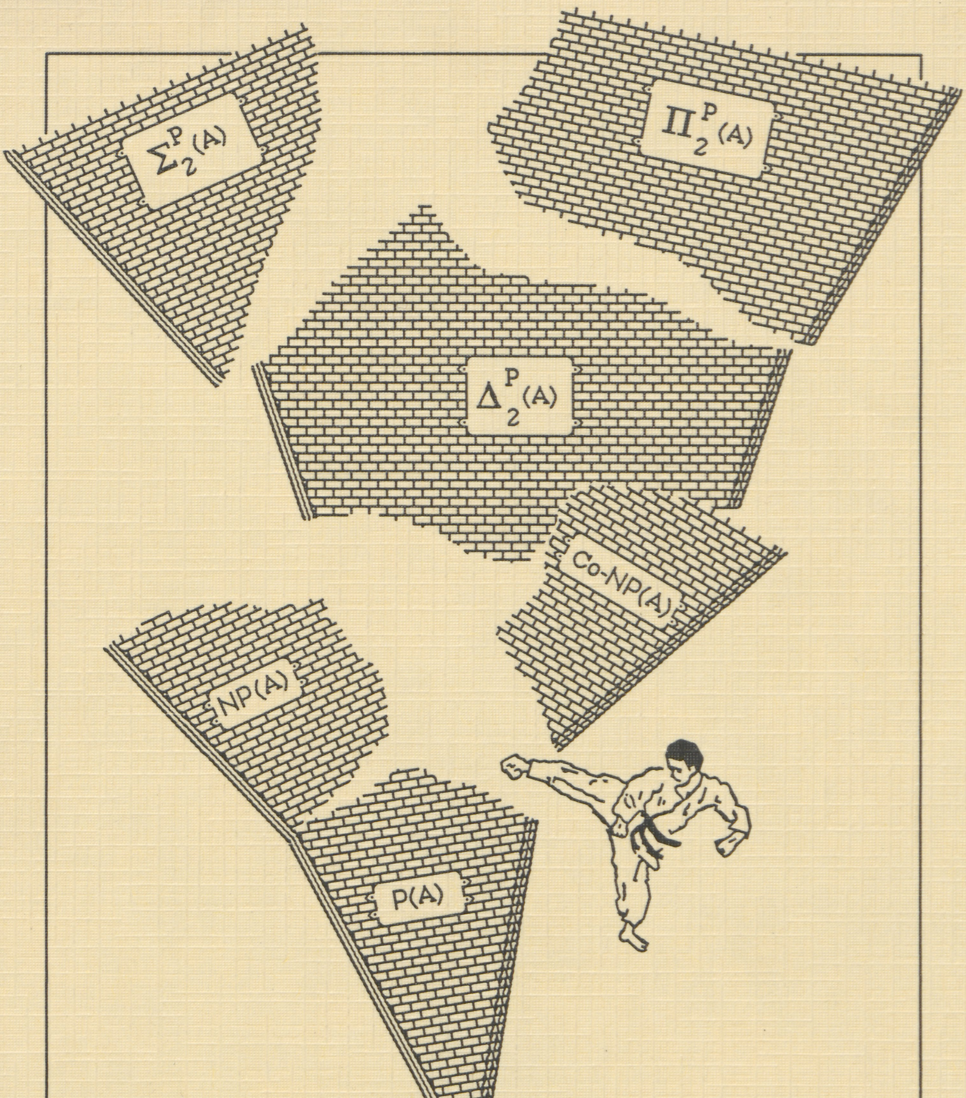


Structural Concepts in Relativised Hierarchies

Leen Torenvliet

STRUCTURAL CONCEPTS IN RELATIVISED HIERARCHIES

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van
doctor in de Wiskunde en Natuurwetenschappen
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
dr D.W. Bresters
hoogleraar in de Faculteit der Wiskunde en Natuurwetenschappen
in het openbaar te verdedigen in de Aula der Universiteit
(in de Lutherse kerk, ingang Singel 411, hoek Spui)
op woensdag 17 september 1986
des namiddags te 17.30 uur (precies)

door

Leendert Torenvliet

geboren te Amsterdam

1986

PROMOTOR: P. VAN EMDE BOAS

CONTENTS

PREFACE

The present thesis is based on three papers on the relativised Polynomial Time Hierarchy. This subject represents a specialised area of one of the central topics in theoretical computer science which is known as Computational Complexity Theory. The common theme of these papers is constituted by investigations after the nature of this infinite hierarchy of complexity classes which is believed to exist between polynomially bounded time and polynomially bounded space.

Many Years of Great Effort by famous scientists have learned that the nature of this hierarchy itself is far too difficult to allow for absolute assertions on its structure to be proved, and form a rewarding (in the sense of paper producing) object of study.

Therefore, instead of looking at the real world, we create different parallel universes, in which we realise potential properties of this hierarchy. Limiting our world of interest to surroundings in which this hierarchy exists at least partially, we derive theorems on the way these classes in the hierarchy can be separated. We are interested especially in creating worlds in which the hierarchy is fairly stretched at bottom levels.

Our work was initiated in the summer of 1983 when we acquired by coincidence, a draft paper of José Balcázar titled "Simplicity for Relativised Complexity Classes", which later appeared in SIAM Journal of Computing under the title "Simplicity, Relativisations and Nondeterminism". In this paper the question was posed whether there could exist an oracle relative to which NP could have a language both simple and P-immune. At first sight it seemed that there should be an as straightforward answer to this question, as to other problems which had been solved before. But numerous discussions I had with P. van Emde Boas during the subsequent period (some of which are still going on) gave evidence to the conjecture that it may not be as straightforward as it seemed then.

It was not until the summer of 1984 that we found a construction of an oracle which provided a positive answer to the question and could start to write the paper "Combined Simplicity and Immunity for Relativised NP" which we submitted (and was accepted) for STACS85 held at Saarbrücken in January 1985. The full paper was accepted as a submission for Information and Control, and will appear in this journal at a later time.

In the spring of 1985 we extended the concept of Immunity and Simplicity to the second level of the Polynomial Time Hierarchy inspired by questions posed by Homer and Gasarch in their paper on the relation between relativised NP and exponential time bounded complexity classes, and again by Balcázar in his invited lecture at the conference on Mathematical Foundations of Computer Science.

Most of the techniques needed for the constructions of the oracles were developed in an electronic mail correspondence I had with P. van Emde Boas, who at that time was on sabbatical leave working at IBM Research Center in San José. The results were accumulated in the two papers "Towards a Strong P-Time Hierarchy", and "Simplicity for Relativised NP", which have appeared as reports in the series of the Computer Science department at the University of Amsterdam. The combination of these results establishes strong separations between language classes in the first two levels of the Relativised Polynomial Time Hierarchy. It seems that by similar methods strong separations above the first levels can not be obtained, and hence that (apart from a nagging open problem mentioned in Chapter 3), this thesis covers all separation results which can be achieved by constructive methods.

VOORWOORD VOOR DE LEEK

In het voor u liggende proefschrift worden een aantal algoritmen beschreven die in het vakgebied bekend staan onder de naam diagonalisaties. Een diagonalisatie is wellicht het best te vergelijken met een verkiezingsprogamma dat tot doel heeft (en erin slaagt) *geen enkele* stem te verwerven. Omdat de in dit proefschrift gebruikte wiskundige termen geen betekenis hebben voor iemand die niet dagelijks deze termen hanteert, en gekweld wordt door de hierachter verborgen vragen, wil ik proberen door het schetsen van een analogie, het soort problemen dat in dit proefschrift wordt aangevallen te verduidelijken.

De door mij beoogde situatie is vol van discriminatie op grond van ras, geloof en levensovertuiging, en van ongelijke behandeling tussen mannen en vrouwen, ja zelfs van ongelijke behandeling tussen mannen onderling. Het lijkt mij daarom verreweg het veiligst, de situatie te plaatsen in een ver land in opper Mongolië, alwaar in een onherbergzaam en van de buitenwereld afgesloten gebied sinds jaar en dag de stam der Yakhushi is gevestigd. Ik wil hier (wellicht ten overvloede) stellen dat, hoewel de Yakhushi in sommige opzichten beschouwd kunnen worden als familie, ik hun politieke inzichten niet deel, noch dat ik instem met de aldaar gebruikelijke riten.

Door een genetische afwijking worden in het geslacht der Yakhushi sinds het begin der eeuwigheid zowel witte als zwarte individuen geboren, en alle vormen van selectie hebben in deze situatie geen verandering kunnen brengen. Er zijn perioden in de geschiedenis geweest dat de witte Yakhushi allen heengezonden werden naar de Noordpool om daar te verkommeren, er zijn perioden geweest dat men de zwarte exemplaren onderdompelde in de geitemelk in de hoop dat door de inname van veel witte vloeistof de kleur van de huid zou veranderen, en er zijn perioden geweest dat de *beide* rassen een voortplantingsverbod kregen opgelegd. (Dit betekende destijds bijna het einde der stam. Gelukkig heeft men dit op tijd ingezien, en werd de wetgeving veranderd.) Alle vormen van opgelegde selectie door combinatie van individuen heeft niet geleid tot een verbetering van de situatie. Voornoemde afwijking heeft namelijk tot gevolg dat zowel uit een huwelijk tussen twee witte Yakhushi zwarte nakomelingen kunnen komen, als dat uit een huwelijk tussen twee zwarte Yakhushi witte nakomelingen kunnen komen.

Sinds het diepgravende werk "De Scheiding der Kleuren" van (de inmiddels hooggeleerde) Tsjing Bo Ru, die voor zijn promotieonderzoek gedurende vier jaren van zijn promotor, de Heilige Keizer Kwa Ni Ping de volledig vrije hand kreeg om huwbare Yakhushi aan elkaar te verbinden is men er echter wel van overtuigd dat het maximaliseren van het in de omgeving huisvesten van één der twee soorten, het geboren worden van de andere soort bemoeilijkt. Zoals het een goed wetenschapper betaamt heeft Tsjing zich in zijn streven naar objectief verkregen resultaten bij het experiment onder

40.000 huwbare Yakhushi uitsluitend geconcentreerd op de effecten van combinaties van verschillende kleuren op het nageslacht, en zich niets aangetrokken van mogelijkheden van emotionele verbondenheid tussen twee verschillende exemplaren. De periode volgend op zijn promotie heeft zich dan ook gekenmerkt door een grote bloei in de horeca sector in het land der Yakhushi.

Zoals eveneens door Tsjing is vastgesteld, is het vooral de kleur der mannen die zwaar doorweegt in de kleur van het nageslacht, en is de kleur der vrouwen nauwelijks van enige betekenis. Het gerucht heeft een tijd gegaan dat deze stelling pas later (na de grote bloei der horeca) aan het proefschrift werd toegevoegd, en er heeft zich intussen een afscheidingsbeweging "Nee tegen stelling 347" gevormd die de wetenschappelijke onderbouwdheid van deze stelling in twijfel wenst te trekken, en voortaan ook de kleur der vrouwen in de selectieprocedure wil laten meetellen. Tot nu toe heeft deze groepering echter nog geen meerderheid verkregen, en gezien de razzias die voortdurend onder andersdenkenden in dit land worden gehouden ziet het ernaar uit dat dit ook voorlopig niet het geval zal zijn. Wij kunnen dus in het volgende de rol die door vrouwen gespeeld wordt in de bepaling van de kleur van het nageslacht rustig verwaarlozen.

Zoals gezegd worden de Yakhushi geregeerd door een Keizer die persoonlijk alle huwelijkszaken voor zijn volk regelt. (Het onderzoek van Tsjing is het laatst bekende geval waarin dit gedelegeerd is geweest.) De Keizer (zelf al sinds "De Grote Slag" een exemplaar van het witte ras) streeft ernaar door middel van het tot stand brengen van de juiste huwelijken, zoveel mogelijk (liefst alle) witte Yakhushi mannen aan zich te verbinden, en zoveel mogelijk (liefst alle) zwarte Yakhushi mannen uit het hof te weren. Eeuwenoude wetgeving betreffende de uithuwelijking en de toekenning van lintjes, verzameld in het boek "De Dans om de Macht" stellen de Keizer in staat dit doel te verwezenlijken. Conform de eeuwenoude wetten gaat de ceremonie van uithuwelijking als volgt in zijn werk:

Met grote regelmaat nodigt de keizer een aantal huwbare Yakhushi mannen uit in de Kamer der Keuzen, welke zij niet ongehuwd èn levend meer zullen verlaten. Gedurende een aantal dagen toont de keizer deze mannen nu elke dag een aantal vrouwen, en zal tevens aan het eind van de dag bekend maken welke vrouwen draagster zullen zijn van de Keizerlijke Grootzegelring. De keizer bepaalt tevens onder de vrouwen een relatie, die we bij gebrek aan betere naamgeving maar "zuster" zullen noemen. Opgemerkt dient de worden dat deze "zuster" relatie niet noodzakelijk symmetrisch is. De Keizer is geheel vrij in het bepalen van deze zuster relatie, en ook in het al dan niet toekennen van een zegel. Een eenmaal bepaalde relatie kan echter niet meer worden ontbonden, en een eenmaal toegekend zegel kan niet meer worden ontnomen.

De Yakhushi man maakt nu bij elke vertoonde vrouw kenbaar, haar al of niet tot zijn echtgenote te willen nemen. Als het antwoord ja is, dan worden hij en zijn echtgenote *onmiddelijk* door de Keizer in de echt verbonden, en zal hij *nimmer* meer van haar kunnen scheiden. In het verleden heeft men wel getracht gesloten huwelijken te ontbinden, doch de toorn des Keizers achtervolgde de (inmiddels niet meer) geliefden tot in de verste uithoeken van het land, en er is dan ook nog nooit een *succesvolle* poging tot ontbinding van een huwelijk gedaan. Om de Yakhushi man niet elke vrijheid te ontnemen, zorgt de oude wetgeving ervoor dat de Yakhushi man niet een ja antwoord hoeft te geven, maar ook een "ja, mits..." mag laten horen. Hij mag een ja voor een bepaalde vrouw laten afhangen van de status van vijf *andere* vrouwen. Bepalend is hierbij of deze vijf andere vrouwen al dan niet zegeldraagster zijn of worden aan het eind van de dag waarop ze door de Keizer worden getoond. De Yakhushi man kan bv. zeggen "Ik zal vrouw0 wel willen hebben als vrouw1 wel-, vrouw2 niet-, vrouw3 en vrouw5 wel- en vrouw4 niet zegeldraagster zijn/worden." Hierbij kan het zijn dat alle vijf vrouwen in het verleden zijn getoond, doch het is ook toegestaan de keuze te laten afhangen van *in de toekomst aan vrouwen toe te kennen zegels.* Daar onder de Yakhushi het geruchtencircuit op volle toeren draait, is de keizer er nog nimmer in geslaagd geheim te houden welke vrouwen hij op welke dag van plan is te gaan vertonen binnen een maand na de huidige dag. Het is wellicht daarom dat de bepaling is ontstaan dat deze "toekomstige" vrouwen dan wel binnen vijf dagen dienen te worden getoond. Eeuwen van witte heerschappij hebben op dit punt een ongelijkheid tussen wit en zwart doen ontstaan. Mag de zwarte man bij een enkele vrouw slechts een vaste verzameling van vijf vrouwen kiezen. De witte man mag bij een vrouw een willekeurige hoeveelheid van verzamelingen van vijf vrouwen kiezen. Hij is hierbij echter wel gehouden om indien één van die verzamelingen een geactualiseerd wordt door de Keizer, zijn ja woord gestand te doen. De witten hebben dus in dit opzicht meer keuze, maar zij leven ook gevaarlijker.

Natuurlijk is het ook bij deze stam niet zo dat de echtgenotes uitsluitend op grond van hun uiterlijke kwaliteiten, dan wel op andere emotionele gronden worden verkozen. De achtergrond voor elk huwelijk dat bij de Yakhushi wordt gesloten is het streven naar Politiek Gewin. De oude wetgeving schrijft immers voor, dat de man die gehuwd is met een vrouw die een zuster heeft die zegeldraagster is voor het leven aan het hof verbonden is terwijl de man wier echtgenote niet in deze omstandigheid verkeert, voor eeuwig naar een onherbergzaam oord zal worden verbannen. Zoals we al hebben opgemerkt streeft de keizer ernaar door zegeltoekenning de witten aan zich te verbinden, en de zwarten te verwijderen. Gezien de bovengeschetste ingewikkelde wetgeving betreffende de keuzevrijheid der Yakhushi mannen, zal de lezer begrijpen dat dit voor de Keizer geen eenvoudige opgave is. Temeer daar onder het zwarte deel van de bevolking het plan leeft een voet tussen de deur te krijgen in het Hof om aldaar een paleisrevolutie te creëren, de Keizer af te zetten, en de broeders uit het onherbergzame gebied onder begeleiding van veel feestgedruis weer naar de hoofdstad te halen. Erger is nog, dat er onder de witte Yakhushi mannen onbetrouwbare figuren rondlopen die erop uit schijnen te zin naar het onherbergzame oord te ontsnappen om aldaar een democratie te stichten. Tot op de dag van vandaag zijn de Keizers echter afkomstig uit één familie, en is nog geen witte erin geslaagd te ontsnappen. De reden hiervoor is dat de Keizers een ijzersterke algoritme hebben om te bepalen welke vrouwen wel, en welke vrouwen niet het Keizerlijke zegel zullen dragen.

Allereerst worden aan het begin van elke dag niet meer dan tien witte, en tien zwarte Yakhushi in de Kamer der Keuzen worden toegelaten. Niet alle tien zullen zij onmiddelijk tijdens deze zitting van een vrouw kunnen worden voorzien, dus is er een voorrangsregeling opgesteld. Alle Yakhushi krijgen een cijfer tussen de 0 en de 9, zo dat voor elk cijfer één vertegenwoordiger van het witte ras, en één vertegenwoordiger van het zwarte ras aanwezig is. Hierbij hebben de nablijvers van de vorige zitting de laagste nummers, en is de *volgorde* van de vorige zitting onverstoord gebleven. Als een witte en een zwarte met hetzelfde nummer dezelfde vrouw wensen, dan krijgt de witte voorrang.

De Keizer nu zal vijf dagen na elke zitting aan geen enkele vrouw een zegel toekennen, hiermee voorkomend dat een gegeven ja antwoord door een der mannen op grond van de gewijzigde situatie in het aantal zegeldraagsters kan veranderen in een nee antwoord. De duur van een zitting is sterk afhankelijk van het verloop der gebeurtenissen. Zolang geen der witten heeft gezegd een vrouw te willen, is er geen reden aan enige vrouw het zegel toe te kennen, dus gebeurt er niets. Als een witte heeft gezegd een vrouw te willen, en een zwarte *met een lager cijfer* wil dezelfde vrouw, dan wordt deze vrouw toegekend aan de zwarte (met het laagste cijfer), en alle vijf vrouwen die bepalend zijn voor het ja antwoord van deze zwarte wordt het zegel onthouden. (Uiteraard voor zover zij al niet een zegel hebben.) De zwarte wordt verbannen en de zitting wordt gesloten. Doet een dergelijk conflict zich niet voor, dan zal de Keizer ertoe overgaan het ja antwoord van de witte te garanderen door een verzameling van vijf vrouwen waarop dit ja antwoord gebaseerd is te bewaren voor de eeuwigheid door alle vrouwen in deze verzameling die het zegel nog niet dragen dit ook nimmer te geven. Tevens zal een nieuwe zegeldraagster worden benoemd, die zuster is van de onderhavige vrouw. Hiermee bindt de Keizer de witte aan het hof. De situatie van vorige dagen kan echter door het benoemen van een nieuwe zegeldraagster zijn gewijzigd. Daarom worden voor de afgelopen vijf dagen alle vrouwen herschouwd, en het bovenbeschreven spel opnieuw opgevoerd (eventueel een *nieuwe* herschouwing implicerend). Aangezien er niet meer dan tien witte candidaten in de zaal zijn, kunnen er niet meer dan tien herschouwingen plaatsvinden, daarom wacht de keizer nadat de laatste keer een witte dan wel een zwarte is gebonden aan een ja antwoord een periode van vijf en vijftig dagen, alvorens de zitting gesloten te verklaren en een nieuwe zitting te openen. Gebeurt er niets, dan wordt de zitting gesloten. Zegt een der witten ja, dan wordt het gehele spel nogmaals opgevoerd.

Omdat er een beperkt aantal witten en zwarten in de zaal is, zal de zitting uiteindelijk worden gesloten, en zullen er nieuwe candidaten in de zaal verschijnen, die alleen nieuwe vrouwen, en eventueel vrouwen uit herschouwingen te zien zullen krijgen. De bovengeschetste algoritme heeft ervoor gezorgd dat geen der witte Yakhushi mannen een vrouw kan kiezen zonder zich daardoor aan het hof te verbinden, en dat geen der zwarte Yakhushi mannen een vrouw kan kiezen zonder zich daardoor een verbanning op de hals te halen. Een bewijs voor de correctheid ervan zal ik u besparen, omdat de correctheid niet alleen afhankelijk is van de algoritme als boven geschetst, maar ook van een nauwkeurige en slimme bepaling van het *aantal* zusters dat aan een bepaalde vrouw wordt toegekend. Er moeten immers bij alle uitsluitingen altijd nog genoeg zusters voor een bepaalde vrouw over zijn om haar man door toekenning van het zegel aan deze zuster aan het hof te verbinden. Bovendien zou de situatie zich heel goed kunnen voordoen dat de Keizer is opgescheept met een verzameling nee knikkers, d.w.z. een twintigtal candidaten dat elke voorbijkomende vrouw afwijst, en daarmee het voortbestaan van de stam als geheel ernstig in gevaar brengt. Het gerucht gaat dat een dergelijke situatie zich in het verleden wel eens heeft voorgedaan. De Keizer, aan het belang van de stam denkend (en zijn geduld verliezend) heeft bij die gelegenheid besloten *alle* twintig candidaten te doen koppen, daarmee zowel de invariant van de algoritme bewarend als toekomstige halsstarrigheden ontmoedigend. Tegenwoordig is de standaardsituatie dat elke voor de Keizer geroepen Yakhushi regelmatig zijn instemming met de geboden waar betuigt.

De bovengeschetste algoritme is een weergave van de constructie die in dit proefschrift als stelling vijf verschijnt. Deze weergave is sterk vereenvoudigd door het kiezen van constanten waar in de oorspronkelijke vorm functies staan, en het is niet geheel duidelijk of de correctheid van de algoritme onder deze transformatie bewaard is gebleven. De lezer die, ondanks deze uitleg, de werking van de bovengeschetste algoritme niet onmiddelijk doorgrondt, behoeft niet te wanhopen. Hij bevindt zich in goed gezelschap. Met de ontwikkeling van deze algoritme is een begin gemaakt in het voorjaar van 1984 en het is niet tot het verschijnen van dit proefschrift dat een begrijpelijke en vermoedelijk correcte versie ervan (in Hoofdstuk 2) werd opgeschreven.

ACKNOWLEDGEMENTS

As a dissertation is generally viewed as the beginning of the period of scientific maturity of the author of the thesis, there should be a place in any thesis where the persons who have contributed to his education are commemorated.

First there were my parents who have, sometimes at great personal sacrifice, created a surrounding in which I could fight my dragons without worrying about troubles creeping in from the outside. Then I would like to thank all of my teachers some of whom have just contributed to the expansion of my knowledge (many times at their own despair) and others who have handed me tools to gain insight in the structure of problems. Special thanks is due to (in chronological order) Mrs. J. van Winsen, who believed that my disinterest was not caused by sheer stupidity, and probably gave my life a different course, to H.W. Boot who taught me the value of "Systematisch Denken", (I always remember these words when confronted with a new and dazzling puzzle, though I'm seldom able to put them to practice), to J. A. Bergstra who in the spring of 1985 took over 2 hours in his busy schedule to have a "where do we go from here" conversation with me and showed me by example that organisation is at least as important as scientific qualities in a research environment, and most of all to P. van Emde Boas (not fitting in the chronological order, since it seems that he was always there) who first started to light up my way into science in the spring of 1977, and has never stopped since. (Even during his eight month sabbatical in San José, he kept in touch with me on a daily basis.) I owe practically everything I know about Computer Science and most of what I know of Mathematics to his wise lessons. He never failed to correct me when I was strolling along dark paths of ignorance, and with enormous patience he corrected the errors in many wild ideas I put on his doorstep. Without him this document could certainly not have been written.

Then I would like to thank all of my colleagues both in the department of Mathematics and the department of Computer Science, who have greatly inspired me to continue with scientific work. Special thanks is due to J. Bruijning, with whom I had in the spring of 1984 a conversation of three days trying to establish the theorem which appears as theorem 5 in this thesis, (we failed then, but this conversation greatly helped to gain the insight which eventually lead to the construction), to K. Koymans who helped me with useful comments on most of the constructions in chapters III and IV, and to W. Bouma and H. van der Meer who joined me in a quest to which experience I probably owe much of the perseverance which was needed to develop the algorithms in this thesis.

I also like to thank the readers of this thesis who took the time to correct errors in an earlier version. Special thanks is due to K. Ambos-Spies, who sent me a 14 (!) page comment on one of the first versions. I believe it greatly helped to improve the quality, though I do *not* believe the work is now flawless.

I am much indebted to M.G. de Bruin, who graciously allowed me to use the hardware to print the manuscript. In this respect thanks is also due to D. Zwarst for fitting me in the very busy schedule of the printing-office at the CWI and thanks on forehand for undoubtedly doing a magnificent job in printing, and T. Baanders for producing the final version of the cover from the rough sketch I produced with an electronic painting program. Finally I like to thank my wife Marian for putting up with me all of those sleepless nights.

Almost all statements which (i) have been extensively studied by mathematicians and (ii) are known to be arithmetically expressible can be seen from a relatively superficial examination, to have quite low level in the Σ^{ZF}_{n}. *As has been occasionally remarked, the human mind seems limited in its ability to understand and visualize beyond four or five alternations of quantifier. Indeed, it can be argued that the inventions, subtheories, and central lemmas of various parts of mathematics are devices for assisting the mind in dealing with one or two additional alternations of quantifier.*

Rogers, Theory of Recursive Functions
and Effective Computability,
Mc Graw-Hill '67 p 322.

I INTRODUCTION.

1. The Natural Numbers and Computational Complexity.

One of the immediate implications of dualism is the 'urge to count'. Dividing the perceived world into categories automatically implies the creation of sets of different cardinality of objects with apparently the same properties. In some Oriental Philosophies there is only room for the Universe or Karma with which man tries to unify, and since human perception and ways to describe the world are essentially of a dualistic nature, there are claims that such philosophies (e.g. Zen) are after the destruction of language [50].

The western world is filled with objects which all have a well defined place in space. At a very early stage in life children are introduced to the natural numbers, which according to some are supplied by God (we will return to this at the end of this section), and learn to use and abuse them. Not only human beings seem to experience this urge to count. It seems that the natural numbers also play an important role in the life of animals. The interested reader should await the publication of [130] in which an extensive survey is given on habits and anomalies of counting all over the world as an introduction to a study of the behaviour and possibilities of counting and counter machines. In this thesis we will present some aspects of counting, but we will use counting methods as a vehicle rather than as an object of study, as is the case in [130].

Virtually all theorems in this thesis are based on ways of counting elements in sets of which the sizes vary with time. Correctness proofs of diagonalisation methods, the common core of all constructions in this thesis, are all based on comparison of the sizes of several sets of strings which are growing during intervals of eternity. It seems that the intricate and beautiful arguments used in this thesis, almost always boil down to the following two simple observations:

Introduction

1) Infinite is greater than finite.

2)Exponential is eventually greater than polynomial.

These two arguments will appear again and again in the following sections, and their importance will become clearer and clearer. When kept firmly in mind during proofreading even the most intricate method will become understandable.

In this first chapter we will introduce the reader to the part of computational complexity necessary to understand the results of the following chapters. We will presume no prior knowledge of any kind except the ability to count. In the next subsection we will introduce a model of computation which we will embed in the natural numbers. Using this model of computation, we will introduce the complexity classes P, NP and Co-NP, and the concept of NP-completeness. Finally we will define the Polynomial Time Hierarchy which is the central object of study in this thesis

Already in the 17th century mechanisms were built to do the more complicated counting actions faster and more reliable than could by done by hand. At the beginning of this (the 20th) century electronic devices were introduced to do the job. Machines could do faster, and more complicated counting than was ever possible before. As problems have a habit to grow with the capacity to solve them, problems emerged which were so complicated that even the fastest machines took considerable time to solve them. Inventing faster and faster machines did not (and will not in the future) really help. However fast a machine may get, a step in a computation must take some time. Very fast machines will need very little time to do a single step, but a large number of steps will still sum up to considerable time.

Instead of standing by the machine waiting till it has finished, one can by studying the algorithm or program, do forecasts on the number of steps (and therefore the amount of time) the machine will take to finish. Usually the size of the problem is a parameter in this type of metacounting. This branch of mathematics, later called analysis of algorithms, can also be generalised away from the study of specific algorithms. More than sometimes it is possible to do a classification of difficulty of a problem depending only on its size. This general branch of mathematics, called computational complexity theory, is the source and main inspiration for the present work.

Many different machines have many different sorts of steps, and one step for one machine may represent many steps for another. Mathematicians, who were occupied with counting long before anyone else, are by nature far more familiar with blackboard and chalk than with real life machines. Instead of standing by the machine and counting the clicks, they were more at ease with an abstract entity. It was Alan M. Turing [128] who in 1932 designed an abstract model for machinery, claiming that on this model -if constructed- any computation on a real life machine can be simulated. In fact any number which can be computed algorithmically in finite time, can be computed by a machine corresponding to the model. This claim which has become known as the Church-Turing Thesis has not (yet) met with a counterexample. Theorems in this thesis depend on (variations of) the Turing machine model, and we will therefore introduce this model to the reader in order to provide him with a clear and precise basis for the results.

The variety of existing models for computation makes it necessary to make precise on which variant the present results are built. The difference between various descriptions of Turing Machines and related models and results on simulation of one model by the other is however all ancient history, and in fact most of it is (or should be) presently part of any introductory course in Computer Science. Therefore in this survey we deal with results requiring long and complicated proofs in a few sentences. The suspicious reader is encouraged to read the standard reference [53] in which all details can be found.

The Turing machine model

Initially the Turing Machine was not at all intended to model real machines. The model of a computing machine described by Turing was inspired by the picture of a man doing calculations with a pencil and a piece of paper. Inspired by the observation that human memory capacity is finite, Turing modelled the man as a finite control, which can be in any of a finite number of states. The piece of paper was modelled by a (two way infinite) tape divided into cells as a "childs arithmetic book"([128] p135). Communication between man and paper (reading and writing) is modelled by a tape head which scans the tape cell by cell moving left, right or staying stationary. The man has a fixed plan or algorithm which is pictured by a *state transition diagram* or *Turing Machine program* consisting of fixed number of instructions stating:
"If in state q reading symbol s goto state q' write symbol s' and move left/right/don't."

Initially the Turing Machine is assumed to be in a special Initial State. The machine halts if and when the combination of the state it is in and the symbol currently read is not compatible with any instruction in the program.

Since Turing studied computable numbers, i.e. numbers which can be produced algorithmically in finite time, the demand of a two way infinite tape is not an unreasonable (though an unnecessary) demand. In finite time the head can only visit finitely many cells. The man modelled by the machine on the other hand can go out and buy more paper if such an action is necessary. In the model there are only finitely many different symbols which can be put on the tape (or read from it). This set of symbols is usually called (tape) alphabet. Many different ways to encode sets in binary numbers show that this is also a reasonable constraint. Besides, the man modelled by the machine could never have mastered an infinite number of symbols to be used in his computation. In both cases infinity of resources is not a necessary condition, but to be a general model resources *are* required to be *unlimited.*

Initially only a finite portion of the tape is filled with symbols from the alphabet. A special symbol called blank - which is denoted by □- is assumed to be part of any alphabet and is assumed to be in any cell which is not yet visited by the tape head. The portion of the tape initially written is called *input*. The machine can be pictured as follows:

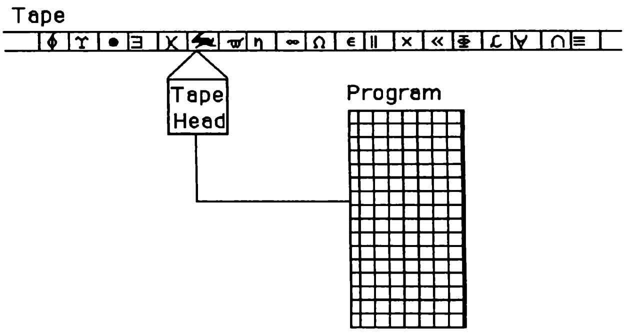


Fig 1. Turing Machine Model

Basically there are two different ways of using a Turing Machine. Originally it was thought of as a model for the computation of functions. In this case the input is viewed as an integer number argument to the function. If and when the machine halts, an integer number being value of the function value for this argument will have been written on the tape. In this thesis we will only consider the alternative use of the model: the language acceptor model. Here the input is viewed as a string of alphabet symbols and the Turing machine program determines membership of this string to a set represented by its program. There are different ways of communicating a yes or no answer to the outside world. As in the case of the computation of functions the Turing Machine can before halting write an some (sequence of) alphabet symbol(s) on the tape, where a special (sequence of) symbol(s) means "yes", and all other (sequences of) symbol(s) mean "no". As there are only two types of answers required, we can also determine acceptance or rejection by halting. If the machine halts, then the answer is "yes", otherwise the answer is no. This means the introduction of unbounded computations, but this is generally not viewed as an obstruction. The two sketched models are equivalent to a model with "accepting states". if the machine can halt at all, there will be certain state/symbol combinations for which there is no successor state. By introduction of new state q_f and inserting the instruction "goto state q_f" in the program for all state/symbol combinations which have no successor, we enforce that if the machine halts, then it will be in state q_f. Of course there will be no instruction in the program for the combination of q_f and *any* tape symbol. All three models are equivalent in the sense of computational power, but we have to choose. In the sequel we will assume this last model of acceptation. In case of a yes answer we say that the Turing Machine *accepts* its input, otherwise it *rejects* its input. The set of all strings that are accepted by a Turing Machine is usually called its *language*.

In general we will let the tape alphabet consist of 0,1 and blank, and sometimes the special separator symbol #. Since any finite alphabet can be encoded as strings of symbols of this alphabet, this is no serious objection. Numerous variations of the model sketched here exist, which can all be shown to be equivalent to this model from a computational point of view in a very strong sense. Therefore we will abstain from describing them.

A Turing Machine program is conceived as a set of five tuples (q,s,q',s',D), where q is the present state of the machine, s is the symbol presently scanned by the tape head, q' is the next state of the machine, s' is the symbol written on the tape, and D is a direction which is one of Left, Right and Ø. Since both the alphabet and the set of states are finite, we can order them and encode tape symbols and states by integer numbers. Hence -as there are only three different directions for the tape head- a Turing machine program can be written as a string like:
$\#i_1\#i_2\#i_3\#i_4\#i_5\#i_6\#\ldots\#i_k$, where each i_j is an integer from a finite set written in binary. Refining the encoding by writing 00 for 0, 01 for 1 and 11 for #, we can write an entire Turing machine program as a single string consisting of 0's and 1's. Fixing a (meaningless) 1 to the left hand side of this string gives an encoding where every possible Turing Machine program can be represented by a single integer written in binary. To determine the special status of the initial and

accepting state among the encoded states many conventions are possible. For instance we can agree to encode the initial state by the smallest integer in the sequence, and the accepting state by the largest integer in the sequence. On the other hand any integer number can be viewed as a Turing Machine program, though many of them will not fit the above encoding, and hence cannot be interpreted. However it is a mechanical effort to show that a Turing Machine program can be written to *check* if an integer encodes a series of five-tuples in this way. By *defining* that all integers which do not encode a Turing Machine program in this way represent an acceptor of the empty set, we get around this problem.

A Universal Machine.

Having convinced ourselves that the correct encoding of a Turing Machine program can be checked mechanically, it is not a big step to introduce a 'programmable' Turing Machine. Instead of just receiving input, this machine finds pairs "program#input" on its tape in the initial configuration. First it checks whether or not the (binary encoded) "program" part is the correct encoding of a Turing Machine program. If not, it erases the tape and gives the answer no (simulating an acceptor for the empty set). If so it simulates a computation using the program. In order to simplify this simulation, in some models the Universal Machine is equipped with one or more extra tapes for administrative purposes. In each step, when having decided upon the next state the machine must search for an instruction corresponding to the combination of this state and the symbol presently under the (simulated) tape head. As the number of states and the number of tape symbols are by definition also limited for this universal model, the simulated states cannot be encoded in the states of this machine, and therefore have to be stored on tape. Though the use of "worktapes" is conceptually easier, the same simulation can also be performed by a single tape universal machine by allowing for more than one track on the tape. On a two track tape the left hand side of a two way infinite tape can be simulated by using the upper track, whilst the lower track is used for the simulation of the right hand side. The use of four tracks allows for marking the place of the tape head in the simulated instruction whilst the real tape head is doing administration or searching for a next instruction in the program. The administrative information itself (remembering present state and symbol) may grow rather large, but never larger than the size of the (encoding of the) program. Therefore before the simulation, the input can be shifted to the right far enough to create space for the administrative data. The multitrack tape in its turn can be simulated on a single track model if we allow for extension of the alphabet; it is also possible by increasing the size of the program to represent a larger alphabet by blocks of symbols from a binary alphabet.

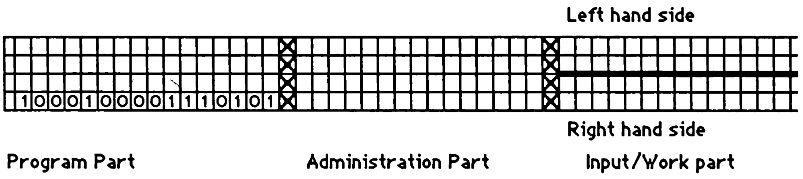


Fig 2. A simulation tape

Up to this point in the discussion we have kept one important aspect under the table. Searching for a next instruction when simulating a Turing Machine program, the Universal Turing Machine may get into trouble. We have already agreed on the definition that a Turing Machine halts if there is no next instruction applicable, so there is no problem simulating this behaviour. If the Universal Turing Machine finds no next instruction to simulate, it halts also. However in our definition of a Turing Machine program there is no obstruction for having *more than one* possible next instruction for given state and tape symbol. Since we did not bother to define the meaning of such a situation for the original model, we can hardly expect to agree on how to simulate the computation in this case.

Therefore we define the behaviour of a Turing Machine with a program in which there are state/symbol combinations with more than one possible successor state. When arriving in such a state the machine may *choose any of the possible instructions*. One can easily see that this may lead to a conflicting situation. One possible sequence of choices may lead to rejection of the input whilst another sequence of choices may lead to acceptation. Here we define that the Turing Machine will accept if there is *any* possible sequence of choices leading to a acceptation of the input; otherwise the machine rejects (note that this definition also incorporates the situation where the Turing Machine never halts on an input). We will name the class of programs where in any situation there is *at most* one possible instruction (the class of) *deterministic* programs. As a consequence the class of programs which do not have this restriction is called (the class of) *non-deterministic* programs. It is convenient to view (as we did) a deterministic program just as a special case of a nondeterministic program and therefore the class of deterministic programs as *included* in the class of nondeterministic programs

Based on this interpretation we can come to an agreement on the simulation of the *computation* by the Universal Turing Machine.

The encoding of a Turing Machine program as described above can also be interpreted as a log on the actions of a Turing Machine during its computation on a certain input. (i.e. Turing machine program and computation may have *the same* encoding.) The only difference is that a certain five tuple may appear more than once in the encoding. Therefore an integer number can be interpreted as the encoding of the history of a computation as easily (and with the same encoding) as it can be interpreted as a Turing Machine program. By the Church-Turing Thesis a Turing Machine can count, and as we have left half of the infinite tape unused in the model, the Universal Turing Machine can subsequently generate on this part of the tape all natural numbers. This leads to a model for the simulation of the behaviour of any Turing Machine program. Starting with 0 on the left hand side of the tape the Universal Turing Machine checks by simulation if the number generated is the correct encoding of an accepting computation using the input program, and the encoded input to this program. If not, it generates the next natural number. Since any computation can be encoded in a natural number, the Universal Turing Machine *must* find an encoding of an accepting computation if such an encoding exists. If not, it will run forever. Though this is perhaps not the most efficient simulation possible, it is correct. Perhaps the only bug in the model is that some program-input combinations may take forever to establish that the input will not be accepted, but then we also don't have any guarantee that the Turing Machine will halt on a given input in the original case. As this thesis is on complexity theory rather than on recursion theory, we are only interested in *finite* computations. We will only consider languages $L \subseteq \{0,1\}^*$ which are recognised by some Turing Machine with the additional constraint that for *any* string x in $\{0,1\}^*$ the Turing machine halts on input x after finitely many steps. We say that for any string x in $\{0,1\}^*$ the question "$x \in L$?" is *decidable*. Such languages are called *Recursive Sets*. Moreover we will consider special Recursive Sets for which the number of steps is bounded by a special type of function of the length of the input. These *Subrecursive Sets* are considered in the next section.

Polynomially Bounded Computations

Instead of defining a set by "acceptation by a given Turing Machine program", we can also define a set of strings by "acceptation by a given Turing Machine program within a bounded number of steps". Obviously if this number of steps is fixed, then the set of strings accepted by the Turing Machine is finite or otherwise trivial. (In a fixed number of steps only a fixed part of the input tape can be scanned, in which only a fixed number of different strings can be written. For the rest it is an "accept all" or "reject all" without inspection situation.) The case where the bound on the number of steps is some kind of function of the input is therefore more interesting one. For various reasons we let the bound be a function of the *size of the input* (i.e. the number of nonblank tape cells at the start of the computation.) Moreover in this thesis we will consider only the case where the bound is a *polynomial* in the size of the input. There are various reasons for considering polynomial time bounded Turing Machines. First all known different models of computation (2-tapes, k-tapes, 2-dimensional tapes and even entirely different models like RAMS, RASPS, etc) can -with the possible exception of models for unbounded parallelism which form a "second machine class"[129]- simulate each other with a polynomially bounded overhead in the number of steps. Therefore as "a polynomial of a polynomial is still a polynomial" the (two) classes of languages defined by polynomially bounded computations are invariant under the used model for computation. Second any finite tape alphabet can be encoded by {0,1,□} at the price of multiplying the input length by a constant factor. As polynomial time bounds remain polynomial if the argument is multiplied by a constant factor, we can restrict ourselves to the binary tape alphabet.

Finally the class of polynomially bounded nondeterministic programs forms a characterisation for many interesting problems in practice, and the characterisation of this class as a class of deterministic programs is a long standing and important open problem in computational complexity theory which is also of considerable importance for algorithms used in practice. Richard Karp recently devoted a large part of his Turing Award Lecture [66] to this subject, and we gladly direct the readers attention to this outstanding and most comprehensible paper.

A *Clocked Polynomial Time Bounded Turing Machine* first counts the number of nonblank input cells, then computes a (usually *the*) polynomial of this number, before starting the computation. After each step the number computed is diminished by 1. If this number reaches 0 before the machine has halted and accepted its input, it will halt and reject its input. Usually the resulting number is written on an extra work/clock tape but we have already seen that this can be simulated on a single tape. In fact the clocked machine can be simulated very efficiently on a single tape. One can easily see in a single tape-multi track model that simulation can be done in a logarithmic time bounded overhead if the encoding of the clock is kept near the tape head. (If a sufficient part of the clock is kept near the tape head this can even be reduced to loglog time overhead.) Fürer in [41] designed a redundant representation of the clock which makes it even possible to maintain a clock in real time (i.e. no overhead at all.)

Simulation of clocked machines

The usual form of a polynomial is $c_nX^n+c_{n-1}X^{n-1}+\ldots+c_1X+c_0$. Therefore we can fix an encoding for a polynomial by simply mentioning the coefficients separated by #. Of course the encoding $c_n\#c_{n-1}\#\ldots\#c_0$ can be converted to a single integer by using the same encoding as in the case of programs and computations, and then a pair of "program#polynomial" can again be encoded by a single integer. Hence any integer can be interpreted by a Universal Turing Machine as a pair of "program#polynomial". Since a polynomial is a computable function, and the Universal Machine can count the size of the input, it can by using (a second track of) the left hand side of its tape for counting the number of simulated steps, also simulate a Clocked Turing Machine. In the case of clocked machines the Universal Machine can determine that the simulated machine rejects its input. It only needs to generate a finite number of computation encodings. Encodings which are longer (i.e. contain more steps) than the polynomial in the size of the input need not be generated. The integers (inputs for the Universal Machine) which do not represent the correct encoding of a pair "program#polynomial" are treated in the same manner as before. These incorrect programs are *defined* to represent the acceptor of the empty set. The bound on the number of steps for this acceptor of the empty set is defined to be one.

Program Generators.

We have seen in the previous sections that any natural number can be interpreted as a pair program#polynomial, and conversely to any program a natural number can be assigned. For the purpose of our diagonalisation algorithms in the next chapter however, we wish to have a relation of a slightly different kind between natural numbers and programs. We wish to have separate enumerations for deterministic and non-deterministic programs. (Although the former class is contained in the latter.) In this section we will show how such a relation might be obtained.

We have already observed that checking whether or not an integer number represents the correct encoding of a pair "program#polynomial" is a doable job for a Universal Turing machine. Therefore we can use this fact as a source for Turing Machine programs. We define an *Enumerator* for the class of nondeterministic clocked polynomially time bounded Turing Machine programs to be a Universal Turing machine which on input i starts with writing consecutive natural numbers on the left hand side of its tape starting with zero. Each time the natural number represents the correct encoding (and here we mean correct in the strong sense of an actual listing of five tuples followed by # followed by the encoding of a polynomial in some fixed well-defined encoding) of a "program#polynomial" pair, the number i is diminished by 1. When i has reached zero, the machine halts, and the tape is cleaned up after which the output represents the i'th pair in this enumeration. This pair, which is usually denoted by φ_i for the program part and Φ_i for the polynomial part can be used for simulation of computation of the machine on certain inputs by another Universal Turing Machine. Of course the program of the Universal machine can also be reorganised to first generate the required "program#polynomial" pair and then simulate the computation. Note that a single program when combined with different polynomial time bounds can in principle encode many different languages. Since Turing Machines cannot "take longer to reject an input" this class of languages is ordered by "inclusion modulo finite difference". If polynomial p_1 is greater than polynomial p_2 then the combination of program M with polynomial p_1 will eventually recognise all strings (and maybe more) which are recognised by the combination M#p_2. However constants in p_2 may cause p_2 to be the greater one on an initial segment of the natural numbers.

The number i which is input to the enumerator is usually called the *index* of the machine, this set of indices introduce an ordering on the clocked nondeterministic polynomial time bounded Turing

Machines which we will call *priority*. In many constructs we make use of this ordering. As the enumerator discards invalid codings, an infinite number of acceptors for the empty set is ruled out in this model. Still the *padding lemma* in combination with the existence of a correct program which rejects everything, ensures that an infinite number of programs for an acceptor of the empty set is generated by the enumerator:

A program that rejects any input either by state or by exceeding its computation time is clearly an acceptor for the empty set. Such a program can be extended with any length of correctly encoded instructions without changing its behaviour. We will make use of this observation in many proofs to follow.

As we have already observed, the property of being nondeterministic is a mechanically checkable property of programs. Hence the program generator may be modified in such a way that it discards generated non-deterministic programs. (Note the difference. Here we mean by non-deterministic programs which are *not* deterministic.) The modified generator then becomes an enumerator for the clocked *deterministic* polynomial time bounded Turing Machines. We now have two enumerators: one for the general class of Turing Machine programs together with corresponding bounding polynomials, and one for the class of Deterministic Turing Machine programs together with corresponding bounding polynomials. As we have seen that natural numbers may encode any program#polynomial pair, any *legal* pair will actually appear for some input i in the corresponding enumeration.

To prevent confusion we will agree on the following notation:

1) When the enumerator for deterministic programs is used, we will use the notation ψ_e for the e'th generated program, and Ψ_e for the corresponding time bound.

2) When the general enumerator is used, we will use the notation φ_e for the e'th generated program, and Φ_e for the corresponding time bound.

In the sequel we will often use a notation like $x \in \varphi_e$. The meaning of such a notation is explained by the fact that we can use the notation φ_e not only to identify the Turing Machine, but also to identify the language formed by all strings x for which there exists an accepting computation of φ_e on input x in length not exceeding the time bound $\Phi_e(|x|)$.

We make one final exclusion. By the choice of "polynomials" above constant functions are also polynomials. Many proofs to follow however would be complicated by having to deal with constants appearing infinitely often as the polynomial part of a "program#polynomial" pair. It is quite easy to see (as we have already observed) that any "program#polynomial" pair where the polynomial is a constant function represents a finite set. A finite set can also be recognised by a program which enumerates all of its members on the tape and consequently compares the input to all of these members. It is therefore no loss of generality to assume that the two enumerators defined above only produce "program#polynomial" pairs where the polynomial part is *at least linear*. (i.e. $\geq X$)

The choice of using two enumerators for different types of machines clears away some trouble we might have encountered in diagonalisations. In some cases the same program has to be handled twice by a diagonalisation algorithm. Once because it is a deterministic program, and once because being a deterministic program it is also a nondeterministic program. The use of two separated enumerators for the two classes solves this problem because the two enumerators will know the same program under different names. (i.e. will generate this program#polynomial pair for different inputs and hence the index corresponding to this pair when generated by the general enumerator will be different (usually smaller) then the index of this pair when it is generated by the deterministic enumerator)

Oracle Machines.

A popular way of classification of problems according to their computational complexity is by translation of problems of which the complexity is (more or less) known. If problem P1 can be translated into problem P2 with reasonable efficiency, then we know that P2 is at least as difficult as P1. Any program that solves P2 can be transformed into a program for solving P1 with a negligible overhead in time. Therefore knowing that P1 is difficult (i.e. costly) to solve implies that P2 is difficult to solve, and on the other hand showing that P2 is easy proves that P1 is easy as well. This method of classification is known as *reduction* and stems from Recursion Theory. This important tool which was first brought into prominence by the work of logician Emil Post, was used to prove thousands of NP-Completeness results after Richard Karp had shown the existence of a resource bounded tool in [65]. The entire theory of NP-Completeness in [43] is built on this method.

A standard way of modelling reductions is by *Oracle Turing Machines*. The original one tape machine is extended with a second tape (the so called oracle tape), which is a write-only tape, and a second head. Three special states called QUERY, YES and NO are introduced. During computation the machine may write a string on the oracle tape. It may also enter state QUERY. (This process is of course controlled by the program) The state QUERY has two successor states: If the string currently written on the Oracle Tape is a member of a fixed language called *Oracle Set,* then the successor state of QUERY is YES. If it is not a member of this set, then the successor state is NO.

The states YES and NO may have any possible successor. This has the important consequence that an acceptor for a language may be converted to an acceptor for its complement in the following way. Acceptor M copies its input to the oracle tape, and then enters the state QUERY. The state YES has only itself as successor (i.e. the machine doesn't halt), or equivalently a rejecting state, and the state NO has q_f as successor (i.e. the machine halts and accepts).

There also exists a limited version of the Oracle Machine model: All state/symbol combinations which have no successor in the original case (i.e. where the machine halts) now have successor state QUERY. The state QUERY has two successors. The state NO has a rejecting successor state, and the state YES has q_f as successor state. Hence the model may only query the oracle once during computation, and the result of the query is the same as the result of the computation. In Recursion Theory the general oracle model is used to model so called "Turing-Reduction" whilst this limited form of reduction is known as "Many-One-Reduction". When limiting *oracle sets* to languages recognizable by non-deterministic polynomial time bounded Turing Machines, and oracle machines to deterministic polynomial time bounded machines, it is presently an open problem whether the two sketched models of Oracle Machines are equivalent.

A Universal Machine for Oracle Machines.

The simulation of an Oracle Machine can also be done by a Universal Machine. We extend the model of the Universal Machine with the same Oracle Tape. The program of a single tape Oracle Machine consists of the (encoding of) a row of 7-tuples. (q,s,q',s',D,s",D') where s" encodes the symbol to be written on the oracle tape, and D' encodes the direction of the head on the oracle tape. The Universal Machine performs a step by step simulation of theses instructions. When the machine simulated enters the state QUERY, the Universal Machine also enters the state QUERY, and the YES state for the Universal Machine has as its successor a state where a subprogram starts to simulate the behaviour of a YES answer to the original Query. The situation is similar for the behaviour of the machine in case of a NO answer.

A simple way of giving special status to the Initial State, and the states QUERY, YES and NO is by stipulating that they are the first four states in any encoding of an Oracle Machine Program. It is of course understood that the oracle set is the same for both the original and the Universal Machine. The rest of the theory remains unchanged.

In many a paper on computational complexity theory handling relativised results, on can find a sentence like: "We assume an enumeration .... of the clocked non-deterministic polynomial time bounded Oracle Machines".(e.g. [10] [11] [12] [109]) The above 17 pages give one *possible* interpretation of such a sentence.

As promised in the second paragraph of this section we will now get back to the origin of the natural numbers. There appears to be some confusion to the exact words of this claim. Cajori[30] and Fine[38] both ascribe these magic words to Kronecker. Cajori (p362) claims that Kronecker said: "Die ganze zahl schuf der liebe Gott, alles Uebrige ist Menschenwerk.", whereas Fine (p183) claims to "once heard him say: 'God created numbers and geometry, but man the functions.'" Both may be right if Kronecker gave vent to this conviction more than once. Kronecker himself however[73] (p337) ascribed the idea of divinity of numbers to Jacobi. With the text of the above sections I hope to have convinced the reader that Kronecker was wrong. God just needed to create *one* natural number (and being God, He of course knew which one), after which the natural numbers created (create and keep on creating) *themselves*.

2. The classes P, NP and Co-NP.

We have already mentioned that many interesting problems in practice can be solved by non-deterministic polynomial time bounded Turing machines. From now on we will use the notation NP for the class of all languages for which there is a non-deterministic polynomial time bounded acceptor, and the notation P for the class of all languages for which there is a deterministic polynomial time bounded acceptor. By our definition $P \subsetneq NP$. An impressive list of problems can be encoded as a language recognition problem and falls in the class NP. In 1979 the first major compilation of problems which have been shown to be "complete" in NP in the preceding ten years, was published in [43]. D. Johnson continued the history writing on problems in NP in [60]. In this thesis we will limit ourselves to giving just a few examples, and establishing the importance of a question which has become known as the P vs. NP problem.

For the Universal Turing Machine simulating an NP-acceptor is equivalent to successively checking all possible computation paths which have a length bounded by a polynomial in the length of the input. It can easily be seen that the number of different computation paths is limited by an exponential function in the size of the input. Hence -as the Universal Turing Machine is a deterministic device- any polynomial time bounded nondeterministic language acceptation algorithm can easily be transformed to a deterministic exponential time bounded algorithm.

An exponential time bounded algorithm is however not an attractive algorithm to run on a real life computer. When the size of the problem becomes interesting, the number of steps needed to solve the problem becomes so large that even the fastest computer will take forever to finish. In many textbooks on Computational Complexity Theory (see for example the introduction of [43] "Computers and Intractability") the following picture is used to illustrate the intuition on the difference between polynomial and exponential time bounded algorithms. For various sizes of the input the number of steps needed by a time bounded automaton is pictured for various polynomial and exponential time-bounding functions. We copy this figure below:

Time Complexity function	Size n					
	10	20	30	40	50	60
n	.00001 second	.00002 second	.00003 second	.00004 second	.00005 second	.00006 second
n^2	.0001 second	.0004 second	.0009 second	.0016 second	.0025 second	.0036 second
n^3	.001 second	.008 second	.027 second	.064 second	.125 second	.216 second
n^5	.1 second	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes	13.0 minutes
2^n	.001 second	1.0 second	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	.059 second	58 minutes	6.5 years	3855 centuries	2×10^8 centuries	1.3×10^{13} centuries

Fig 3: Polynomial vs Exponential time. Computation times on a hypothetical machine executing 10^6 instructions per second for various input sizes, and time bounds on the computation.

This illustration should discourage the use of algorithms which may take exponential time for real life computations.

Most known problems in NP have obvious recognition algorithms which are all more or less a variation of the "exhaustive search method": Formulate all possible solutions to a problem and throw away solutions which do not meet boundary conditions. Frequently these problems have exponentially many possible solutions. *Checking the validity* of a possible solution is (as an accepting computation must by nature be polynomial time bounded) a polynomial time bounded process. Hence the existence of a deterministic polynomial time bounded algorithm cannot be ruled out a priori.

Since nondeterministic machines are not presumed to exist in real life, we have to do with deterministic machines and therefore the question of the existence of a deterministic polynomial time bounded algorithm is crucial to the tractability of a problem. For many problems in NP, the exhaustive search method has been refined by trying to predict whether computations are heading to success or not, in this way pruning huge chunks from the exponentially large tree of possible computations. "Branch and Bound methods" are a typical example of such refinements. Up till now however none of these algorithms can be guaranteed to run in polynomial time in all possible cases. Hence modifications of exhaustive search are generally not accepted as practical solutions but more as emergency exits.

Knapsack.

Perhaps the best known problem having the above sketched behaviour is the so called "knapsack problem". In its original form a number of objects are presented, all of which have a certain weight and a certain value. Any of the objects can be put into a knapsack which is to be carried by human strength. The human in the story has a known carrying capacity which is exceeded by the total sum of the weights of the objects, and hence some of the objects have to be left home. The question is: "Which of the objects should be put into the knapsack, and which should be left home in order to maximise the total value of the objects in the of the knapsack without exceeding the capacity of the carrier."

First we observe that the exhaustive search method gives a solution to the problem. If there are n objects then the knapsack might hold any of 2^n different combinations of the objects when we disregard the capacity. If we investigate all 2^n different possible solutions, discarding any solution which exceeds the capacity and select the maximum of the remaining solutions, we have solved the problem.

It is not quite obvious that this problem is in NP, since we have characterised problems in NP as having a solutions verifiable in polynomial time. Given a combination of objects however, it is unclear whether it is easily decidable if this is *the* optimal combination, without comparing all possible other combinations which do not exceed the capacity. Therefore the knapsack problem is usually formulated differently:

KNAPSACK:
INSTANCE : n objects $x_1 \ldots x_n$ all of which have a known (integer) weight $w_1 \ldots w_n$.
and a(n integer) bound b.
QUESTION: Is it possible to find a subset S of {1,...,n} such that $\Sigma_{i \in S} w_i = b$.

Note that we have also identified weight and value.
A given solution (i.e. a given combination) is easily checked in polynomial time by summing up the corresponding weights. Given an algorithm for the decision problem we can solve the optimisation problem sketched above, for if we know the capacity b, we can successively formulate the decision problem for b, b-1, b-2, b-3, etc until the answer to the decision problem is yes. Then we will have found the solution. If the decision problem can be solved in time t, then the optimisation problem can be solved in time b×t. Strange as it may seem however b×t is a time overhead which may *not* be polynomially bounded. If the encoding of the capacity b is a substantial part of the input then the input may have a length of order log b (which is the length of b in binary notation), and hence b steps may be an exponential time overhead. Fortunately there exists an algorithm to speed up things. Instead of trying b-1, b-2 etc we try b/2, and if the answer is no try b/4, and if the answer is yes try (3×b)/4 etc. This *bisection* algorithm is known to run in log b steps, so the total time becomes log b × t, which is certainly a polynomial overhead. So the optimisation problem and the decision problem are equivalent in the sense that either both belong to P or none of the two belongs to P.

Another point to be made here is that the decision problem corresponds to *actually* generating a solution. In the case of KNAPSACK we may not be interested in *if* it is possible to select objects for which the weights sum up to the given bound, but rather *which* objects to take. However a P-time decision algorithm for KNAPSACK would easily give a solution in hands. To see this suppose the objects are $x_1,\ldots,x_n$. For i=1,...,n perform the following actions:

1) *throw away* x_i ;

2)run the decision algorithm with the same bound b;

3)If the result is YES then do nothing, else add x_i again to the set of objects; x_i is one of the objects to take.

After n runs, we are left with a set of objects of which the weights sum up to b. (Unless such a set does not exist, but then we will find this out in the first run.)

Completeness.

The 'guess and verify' nature of problems in NP is perhaps more fundamental then might appear at first sight. It is a basic property of the acceptor of these languages, and of course finding a polynomial time deterministic simulator for the non-deterministic acceptor means (indirectly) finding a polynomial time deterministic algorithm for all problems in NP. Because of the strong *structural* similarity of problems in NP however, this 'handle' to identify P and NP is not limited to finding such a simulator. There exists an extensive list of problems in NP each of which has the property that finding a deterministic polynomial time bounded algorithm for one of them indirectly means finding a deterministic polynomial time bounded algorithm for *any problem in* NP. These "maximal" problems in NP are called *NP-complete* problems. Finding (and proving) a problem to be NP-complete is based on the Oracle Turing Machine Model. If, given problem P as an oracle set, every problem in NP is recognizable by a deterministic polynomial time bounded oracle machine then of course P is at least as difficult as any problem in NP, or *NP-hard.* If moreover P is in NP, then P is NP-complete by definition. Note at this point that the nature of oracle machines implies that NP-hardness is a transitive property. If problem P1 is NP-hard, and problem P1 can be recognised by a polynomial time bounded deterministic oracle machine with oracle P2, then P2 is NP-hard, since any polynomial time bounded oracle machine with oracle P1 can be converted to a polynomial time bounded oracle machine with oracle P2 recognising the same language.

In 1971 Steve Cook [34] was the first to show that the problem of determining the existence of an assignment to the boolean variables of a propositional formula satisfying this formula (i.e. making the value of the formula true) is NP-complete, and it is at that point in history that we place the "date of birth" of the notorious P vs NP problem. If it can be shown that there is one problem in NP for which there cannot exist a deterministic polynomial time bounded algorithm, then of course P≠ NP. On the other hand: finding a deterministic polynomial time bounded algorithm for an NP-complete problem implies P=NP. As Cooks proof is rather complicated, we choose to show the completeness of another problem first and come back to Cooks problem (known as SATISFIABILITY) at a later point in this section.

Bounded Tiling.

A problem which by nature is *very* close to finding a deterministic simulator for the nondeterministic machine, and should therefore be the basis of NP-completeness theory is BOUNDED TILING. It first appeared in the literature in [79]. Lewis & Papadimitriou[80] and later Savelsbergh and van Emde Boas [102] showed how the entire theory of NP-completeness could be set up from BOUNDED TILING instead of from SATISFIABILITY. This strategy has two advantages. First the BOUNDED TILING problem is easy understandable (contrary to satisfiability it requires no familiarity with logic), and second the reductions of BOUNDED TILING to other NP-Complete problems seem easier. The problem is presented thus:

BOUNDED TILING
INSTANCE:A finite set of tiles; an N×N square with a colouring on the border.
QUESTION: Does there exist a tiling of the entire square, extending the colouring along the border.

Finding a deterministic polynomial time bounded algorithm which solves any instance of BOUNDED TILING, (i.e. says "yes" to all instances for which such a tiling exists and "no" to all instances for which such tiling does not exist) leads to the construction of a polynomial time bounded deterministic simulator for the non-deterministic polynomial time bounded Turing Machine. Given a (nondeterministic) Turing Machine program, a polynomial time bound, and an input, we construct in polynomially bounded time, an instance of BOUNDED TILING which is guaranteed to have a tiling if and only if the given Turing Machine program accepts its input within the time bound.

For technical reasons we first do a little "preprocessing" on the input Turing Machine program to guarantee that the computation to be simulated by the tiling of the square has the following two properties:

1)The tape head never moves left of the cells originally occupied by the input.

2)There exists no state q such that the machine can move both left and right while going into this state.

Assume that the resulting program has states $q_1 \dots q_n$, and time bound Φ.

Next we create an instance of BOUNDED TILING:

We introduce the following colours:

1) To encode the n different states: $c_1, \dots, c_n$

2) To encode the 3×n different state/symbol combinations

$c_{01}, \dots, c_{0n}$

$c_{11}, \dots, c_{1n}$

$c_{21}, \dots, c_{2n}$

3)To encode the tape symbols 0,1, and □

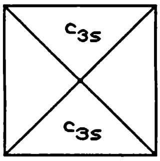
c_{30}, c_{31}, c_{32}

4)To encode two "clean up" states combined with symbols

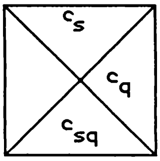
$c_F, c_{0F}, c_{1F}, c_{2F}$

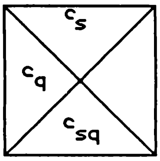
$c_{F'}, c_{0F'}, c_{1F'}, c_{2F'}$

The first tile we introduce is used to transport symbols which are not presently under thc tape head along horizontal strips. For s ∈ {0,1,2} we have the tile:

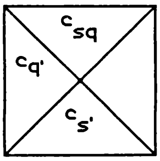


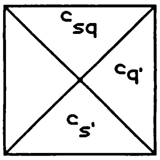
To simulate the arrival of the reading head in a tape cell in state q reading symbol s, we have for q∈ {0,...,n} and for s∈ {0,1,2}:

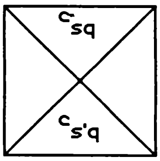




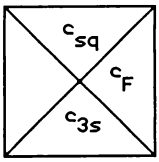
To simulate the instructions (q,s,q',s',L), (q,s,q',s',R), and (q,s,q',s',Ø) we have for all of these instructions which appear in the program respectively:



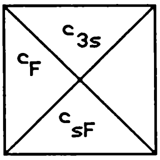




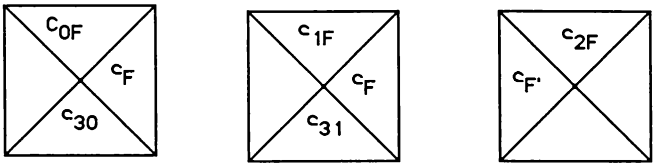
For any symbol/state combination s/q, where q= q_f (i.e. at which point the machine accepts the input), we have the tile:



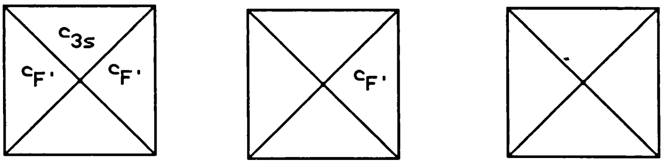
which forces the simulation into the first of the two "final states" and simulates a move right. For this first final state we have a tile to accept the head:



and one to transport the head in state F in the next move if the symbol under the head is 0 or 1, and to reverse the direction and get into the second final state if the right end of the tape is reached:



In the second final state F', the entire tape is erased, upto the leftmost column, and the rest of the tape is filled with blank tiles:



With this set of tiles we formulate the problem. If $x=x_1,\ldots,x_k$ is an encoding of the input, we construct a square of size $(\Phi(|x|) + 2) \times 3\Phi(|x|)$. Only the top border of the square is coloured. Cells 3 to $|x| +1$ have colour c_{30} if the corresponding bit of x is 0, and colour c_{31} if it is 1. Cell 2 has colour c_{01} if $x_1=0$, and c_{11} if $x_1=1$ (Initial state should be encoded here). All other border cells are white. If there exists an accepting computation of $\Phi(|x|)$ steps or less, then a tiling can be found in which after $\Phi(|x|)$ all colours can be erased. As the Turing Machine can visit no more than $\Phi(|x|)$ cells in $\Phi(|x|)$ steps this takes up no more than $\Phi(|x|)+1$ extra rows. The square can then be filled with blank tiles.

On the other hand any tiling of the square encodes an accepting computation within $3\Phi(|x|)$ steps. Note: the size of the square is no longer of the form N×N, but this can easily be patched by adding more columns which can only be filled by entirely blank tiles.

Having established a "master problem" for the NP-completeness theory, it is no big effort to show that other problems are NP-complete. First we treat the old master problem SATISFIABILITY, which is presented thus:

SATISFIABILITY

INSTANCE: A collection $C=\{c_1,\ldots,c_m\}$ of clauses on a finite set U of variables.

QUESTION: Is there a truth assignment for U that satisfies all the clauses in C.

Here:

Variables are elements drawn from the set $\{x_1,\ldots x_n\}$. The variables $x_1,\ldots,x_n$ can have value **true** or **false**. A *literal* is either variable x_i or the negation of a variable $\overline{x}_i$.For all i: If the *value* of x_i is **true**, then the value of $\overline{x}_i$ is **false**, and vice versa. A *truth value assignment* to a set of variables is a function which gives each variable the value **true** or **false.** A *clause* is either a literal, or a sequence "clause connective clause", where "connective" stands for $\wedge$ or $\vee$. By definition if c_1 and c_2 are clauses then $c_1\wedge c_2$ is **true** if and only if both c_1 and c_2 are **true** and $c_1\vee c_2$ is **true** if and only if at least one of c_1, c_2 is **true**. A clause is *satisfied by a truth value assignment* if this assignment makes the clause **true**.

The presented proof can also be found in Lewis and Papadimitriou. We introduce the variables x_{ijk},$1\leq i,j\leq N$, and $1\leq k\leq |T|$, representing the possibility of locating the tile T_k on the square (i,j). We need four types of clauses:

(1) To guarantee that at least one tile will be placed upon each square of the plane:

$$\bigwedge_{i,j}(\bigvee_k x_{ijk})$$

(2) To guarantee that exactly one tile will be placed upon each square of the plane:

$$\bigwedge_{i,j} \bigwedge_{k\neq k'}(\bar{x}_{ijk} \vee \bar{x}_{ijk'})$$

(3) To guarantee that the adjacency conditions between tiles are satisfied

$$\bigwedge_{i,j} \bigwedge_{k,k'}(\bar{x}_{ijk} \vee \bar{x}_{ij+1k'})$$

For all pairs $(T_k,T_{k'})$ which cannot occur as a horizontally adjacent pair.

$$\bigwedge_{i,j} \bigwedge_{k,k'}(\bar{x}_{ijk} \vee \bar{x}_{i+1jk'})$$

For all pairs $(T_k,T_{k'})$ which cannot occur as a vertically adjacent pair

(4) To guarantee that the boundary conditions are satisfied

$\bar{x}_{1jk}$ for all tiles T_k which cannot be placed upon the square (1,j);

$\bar{x}_{Njk}$ for all tiles T_k which cannot be placed upon the square (N,j);

$\bar{x}_{i1k}$ for all tiles T_k which cannot be placed upon the square (i,1);

$\bar{x}_{iNk}$ for all tiles T_k which cannot be placed upon the square (i,N);

For any instance of BOUNDED TILING, we now have created a Boolean Formula for which a truth value assignment can be found satisfying the formula *if and only if* a tiling of the square satisfying the boundary conditions exists. Hence a recogniser for SATISFIABILITY can be transformed into a recogniser for BOUNDED TILING, which means that SATISFIABILITY is also NP-hard.(It is obvious that SATISFIABILITY is in NP, and hence SATISFIABILITY is NP-Complete.)

Using this knowledge we can prove that the problem KNAPSACK with which we started this subsection is also NP-complete. Given any boolean formula we construct an instance of KNAPSACK which has a solution if and only if the input formula is satisfiable:

Let $C_1,\ldots,C_m$ be the clauses in the Boolean Formula

Let k be the maximal number of literals in any clause.

Let $d > 4k^2$.

Let $x_1,\ldots,x_n,\bar{x}_1,\ldots,\bar{x}_n$ be the literals in the Boolean Formula.

Let $\varepsilon_{i,j} = 1$ if $x_i \in C_j$ else $\varepsilon_{i,j} = 0$

Let $\delta_{i,j}=1$ if $\bar{x}_i \in C_j$ else $\delta_{i,j}= 0$

Construct objects $o_1,\ldots,o_n,\bar{o}_1,\ldots\bar{o}_n$

With resp weights $w_1,\ldots,w_n,\bar{w}_1,\ldots,\bar{w}_n$

Where: $w_i = \sum_{j\leq m} \varepsilon_{i,j} \times d^j + d^{m+i}$

$\bar{w}_i = \sum_{j\leq m} \delta_{i,j} \times d^j + d^{m+i}$

and "fill" objects $\{f_{pj} \mid 1 \leq j \leq m, 0 \leq p \leq k\}$ with weights

$w(f_{pj}) = (k+p)\times d^j$

Finally let $b = (2k+d^m) \times (d^m-1) / (d-1)$

Now since $d > 4k^2$, the coefficients $1,\ldots,2k$ cannot sum up to d so this choice forces that for $1 \leq i \leq m$ one of w_i or $\bar{w}_i$ be part of the sum of weights, which means that one of $\{o_i,\bar{o}_i)$ has to be chosen, and since the coefficient of d^{m+i} is 1 in b, they cannot both be chosen. This choice represents a truth value assignment to the variables $x_1,\ldots,x_n$. If for some $j\leq m$ both ε_{ij}, and δ_{ij} are 0 for all i in the chosen set of objects, then since any of the fill objects has weight greater than $k\times d^j$ *and* less than $2k\times d^j$, the coefficient of cannot sum up to 2k in the sum of weights, which is required to meet b. Hence the truth value assignment *has to satisfy all the clauses C_j*. If on the other hand at least one of $\{\varepsilon_{ij}, \delta_{ij}\}$ is equal to 1, then the fill objects can be used to make up the difference.

The NP-hardness proof for KNAPSACK given here remains unreferenced for lack of enthusiasm to search for the proof in the literature. Usually KNAPSACK is proven to be NP-complete via two or more other problems using the transitivity of this property. Moreover, for all of these "natural" NP-complete problems, Hartmanis and Mahaney [44] have constructed a general method with which the existence of an invertible reduction can be demonstrated (without actually generating the reduction). Because NP-completeness is not the main subject of this thesis we wish to limit the number of examples of NP-complete problems and therefore we have chosen the direct method of constructing a reduction here.

Co-NP and Co-NP-completeness.

Another interesting class of languages is formed by the complements of NP languages. This class is called Co-NP. Of course if P= NP, then Co-NP= NP, since P is closed under complementation. But let us assume for the sake of argument that P≠ NP. What then is the complement of e.g. a language like SATISFIABILITY? A Boolean formula F is not a satisfiable formula if and only if all possible truth value assignments to its variables make F false. That is to say the complement ¬F is true under all possible truth value assignments. Which means that ¬F is a tautology. So TAUTOLOGIES which is (almost) the complement of SATISFIABILITY is a language in Co-NP. Moreover TAUTOLOGIES is Co-NP-complete.

Since SATISFIABILITY is NP-complete, there exists for any problem P in NP a deterministic polynomial time bounded oracle machine M with SATISFIABILITY as its oracle set. If the oracle is replaced by TAUTOLOGIES, this machine can be transformed into a deterministic polynomial time bounded recogniser for the complement of P. For given an instance x of P, the machine M transforms this instance into a satisfiable formula F iff x∈P, and into a non-satisfiable formula if x∉P. Converting a boolean formula to its negation is a polynomial time bounded operation, hence M can be extended to produce ¬F which is a tautology iff x∉P. Note that this conversion works for both the limited and the general type of oracle machines.

3. A Polynomial Time Hierarchy.

If we consider only the general type of oracle machines, then we could limit ourselves to considering only SATISFIABILITY as an oracle to recognise both all NP and all Co-NP languages(and more). With the general type machine 'no' answers can easily be converted to 'yes' answers and vice versa. Therefore the class of languages NP $\cup$ Co-NP is included in the class of languages characterised by P(SAT). We define formally:

For a set A: $P(A) = \{ \psi_i(A) \mid i \in \omega \}$; $NP(A) = \{ \varphi_i(A) \mid i \in \omega \}$;

For a class of sets C: $P(C) = \bigcup_{A\in C} P(A)$; $NP(C) = \bigcup_{A\in C} NP(A)$

Since SATISFIABILITY is NP-complete one can easily see that since for any language L in NP there is an index i such that $L = \psi_i(SAT)$ we have that $P(L) = \{ \psi_j(\psi_i(SAT)) \mid j \in \omega \}$ and hence that $P(NP) = \bigcup_{A\in NP}(\{ \psi_i(A) \mid i \in \omega \}) = \bigcup_{A\in NP}(\{ \psi_j(\psi_i(SAT)) \mid j \in \omega \ \&\ A= \psi_i(SAT) \}) =$ P(SAT) since the composition of two deterministic oracle machines is a deterministic oracle machine, and the composition of two polynomials is a polynomial.

Instead of considering only deterministic polynomial time bounded oracle machines, we can also consider non-deterministic polynomial time bounded oracle machines, and obtain the P vs NP problem on a next level. That is "is P(NP) equal to NP(NP) or not?", and of course NP(NP) can itself be used as a class of oracles, and yet another level can be obtained considering the classes P(NP(NP)) and NP(NP(NP)).

Thus we obtain a hierarchy of deterministic and non-deterministic complexity classes, at each level of which the P vs NP problem plays a crucial role. It is of course clear that if at one level we find that P(NP...(NP)...) = NP(NP...(NP)...) then the hierarchy does not exist above that level. Especially if. P =NP then the hierarchy does not exist at all.

We can also consider bringing the class Co-NP into this game by observing Co-NP(NP) and Co-NP(NP(...(NP)...). Then we obtain the NP vs Co-NP question repeated at infinitely many levels. At each level proving an *in*equality means solving the P vs NP problem.

Characterisation with quantifiers.

The hierarchy introduced above can also be characterised in an equivalent way which seems entirely different at first sight. The class NP may be thought of as a class of deterministic polynomial time bounded relations to which one polynomially bounded existential quantifier is applied. Co-NP on the other hand may be thought of as a class of deterministic polynomial time bounded relations to which one universal quantifier is applied. In fact virtually all NP problems when formulated with words sound like "does there exist a ... such that...", and as a consequence all Co-NP problems are formulated as "Is it the case that for all..."

By using quantifiers for characterisation, we obtain a hierarchy of expressions like: $Q_1x_1 \ldots Q_nx_nR(x_1,\ldots,x_n,y)$ where the quantifiers $\exists$ and $\forall$ alternate. It turns out that for every language L in NP(NP...(NP)...) there is a deterministic polynomial time bounded computable relation R such that if n is equal to the number of times NP appears in the expression minus one then for any y: $y \in L$ iff $\exists x_1 \ldots Q_nx_nR(x_1,\ldots,x_n,y)$ (So for every language L in NP there is deterministic polynomial time computable relation R such that $y \in L$ iff $\exists xR(x,y)$) Likewise for every language L' in Co-NP(NP...(NP)...) there exists a deterministic polynomial time bounded computable relation R' such that for any y: $y \in L'$ iff $\forall x_1 \ldots Q_nx_nR'(x_1,\ldots,x_n,y)$ Celia Wrathall first proved this relation in [132].

The classes of languages in the hierarchy are *named* Σ_k^p, Π_k^p , and Δ_k^p. At first sight this may seem somewhat peculiar. Σ and Π seem to have more to do with sums and products then with alternation of quantifiers or Oracle Turing Machines. The reason for this terminology is that the P-Time Hierarchy was introduced as a resource bounded analog of Kleene's [69] *Arithmetical Hierarchy*, and that the names of this Arithmetical Hierarchy were inherited.

Instead of from Polynomial Time Bounded computable relations, Kleene's Hierarchy is set up from

Primitive Recursive Predicates on natural numbers: If x is a natural number and P(x) is a primitive recursive predicate then $\exists x.P(x)$ is equivalent to P(0) **or** P(1) **or** P(2) **or** .... Using + for **or**, this can be denoted by P(0) + P(1) + P(2)... or Σ P(x). Likewise $\forall x P(x)$ is equivalent to P(0) **and** P(1) **and** ... , which can be denoted by ΠP(x), using $\times$ for **and**. Now the class Σ_1 is defined as the sum of Primitive Recursive Predicates, Π_1 is defined as the product of primitive recursive predicates, Σ_k is defined as the sum of Π_{k-1} predicates and Π_k is defined as the product of Σ_{k-1} predicates. We will adopt the names for this hierarchy and define the *Polynomial Time Hierarchy:*

$$\Sigma_0^P = \Pi_0^P = \Delta_0^P = P,$$

$$\Delta_{k+1}^P = P(\Sigma_k^P),$$

$$\Sigma_{k+1}^P = NP(\Sigma_k^P), \text{ and}$$

$$\Pi_{k+1}^P = Co\text{-}\Sigma_{k+1}^P$$

Thus having set up the theory in the absolute case, we can conclude the introductory part of this thesis and start to relativise. Because the ordering of the polynomial time hierarchy is based on the Oracle Machine Model we can easily add an extra oracle A to the definition and thus obtain for any oracle A the *Polynomial Time Hierarchy Relativised to Oracle A* as:

$$\Sigma_0^P(A) = \Pi_0^P(A) = \Delta_0^P(A) = P(A),$$

$$\Delta_{k+1}^P(A) = P(\Sigma_k^P(A)),$$

$$\Sigma_{k+1}^P(A) = NP(\Sigma_k^P(A)), \text{ and}$$

$$\Pi_{k+1}^P(A) = Co\text{-}\Sigma_{k+1}^P(A)$$

It is this Relativised Polynomial Time Hierarchy which is the central object of study in this thesis.

II THE FIRST LEVEL OF A RELATIVISED HIERARCHY

In the present and following chapters we shall present the construction of oracle sets relative to which complexity classes have certain properties. Traditionally the name for an oracle set is A. We will however present many different oracle sets with many different properties. To discern between the different oracle sets many authors (including myself) have turned to the use of the other capitals of the Roman alphabet B,C, etc. In any particular construction however there is always only *one* oracle set under construction therefore the use of a single capital A to identify all oracle sets to follow in turn cannot lead to confusion. We choose to use the letter A for all oracles constructed in this and the following chapters with the exception of quantification. When a quantification over all possible oracles is presented we use the capital X to identify the (unknown) oracle set.

Oracle sets are constructed by infinitely many stages by means of diagonalisation. A single *requirement* (like e.g. $L(A) \notin P(A)$) to be met by the construction is replaced by an infinite set of requirements R_e (like e.g. $R_e : \psi_e(A) \neq L(A)$) which are all met or *satisfied* at some stage $s \geq e$ during the construction, for all $e \in \omega$. A construction always starts with a basic set which is of fairly simple structure. This set will invariably be called A_0. Then at following stages elements are added to A or deleted from A to achieve a next refinement, and to achieve the desired property for more and more machines. We formalise this by constructing at stage s+1, a new set A_{s+1} which consists of all the elements of A_s plus or minus some new elements. It is then understood that the oracle A is the limit of the sets A_s as s approaches infinity. For the satisfaction of requirements (and to prevent that requirements already satisfied at earlier stages become unsatisfied) it is often equally important *not* to add certain strings to the oracle set A, and making a commitment not to do so in the future, as it is to extend A at some stage. For this purpose either an explicit *restraint set* is maintained throughout the construction, or a "bound on the length of strings", where it is understood that at no stage a string which is in the restraint set, or has a length less than or equal to the bound is added to A. In the case of an injury priority argument this agreement may be violated under certain conditions. However we feel that this is too complicated to treat generally. Therefore we defer the treatment of priority constructions until the point where we really need them

We will start this section by showing the existence of an oracle set A such that NP(A) $\neq$ P(A), as is historically correct. This construction was first presented by Baker, Gill, and Solovay[10] and was the inspiration for many results to follow. The construction is done by diagonalisation with the help of an enumerator for the clocked deterministic polynomial time bounded oracle machines.

Consider the language $L(X) = \{0^s \mid \exists x \in X. |x| = s\}$. Clearly for any oracle X the language $L(X) \in NP(X)$, since the corresponding oracle machine need only guess a string which has the same length as the input and write it on the oracle tape. The intention of the construction of the oracle set A is to ensure that $L(A) \notin P(A)$. Therefore the construction has to ensure that the language L(A) is not recognised by any clocked deterministic polynomial time bounded oracle machine. Or $\forall e$: $\psi_e(A) \neq L(A)$. Here we first use the argument that the exponential function 2^x eventually outgrows any polynomial. On input x the machine ψ_e can only take $\Psi_e(|x|)$ steps, therefore it cannot write more than $\Psi_e(|x|)$ *different* strings on its oracle tape during a computation on input x. On the other hand since x is written in binary there are $2^{|x|}$ different strings of length $|x|$. By our definition of L(X) for any such string y, if $y \in A$ then $x \in L(A)$. So after simulation of $\psi_e(A_s)$ on input x at stage s+1 of the diagonalisation if we have that $2^{|x|} > \Psi_e(|x|)$ then a string y of length $|x|$ can be chosen such that $x \in L(A_s \cup \{y\})$ *and* y is not queried in (and hence does not influence) the simulated computation. By the same argument it cannot write a string *longer than* $\Psi_e(|x|)$ symbols on the oracle tape, since each symbol takes at least one step to write. Hence ensuring that changes to the oracle made at stage s+1 are limited to strings of which the length is exponential in strings involved at stage s, ensures that actions taken at stage s need not be influenced by actions taken at stage s+1. Now the diagonalisation is simply described by the following:

Requirements: $\forall e : \psi_e(A) \neq L(A)$

Construction

```
    stage 0: $A_0 = \emptyset; m_0=0;$
    stage s+1: First compute $m_{s+1}$ which is the smallest integer such that $m_{s+1} > \Psi_s(m_s)$ and
            $2^{m_{s+1}} > \Psi_{s+1}(m_{s+1})$. Then find a string y of length $m_{s+1}$ such that y is
            not queried by $\psi_{s+1}$ on input $0^{m_{s+1}}$;
            Now if $\psi_{s+1}(A_s)$ accepts $0^{m_{s+1}}$ then we do nothing, but
            else we let $A_{s+1} = A_s \cup \{y\}$.
    end of stage s+1
end of construction
```

Note: As we have assumed that Ψ_{s+1} is at least linear we have that $m_{s+2} > \Psi_{s+1}(m_{s+1}) \geq m_{s+1}$ so oracle changes at later stages cannot disturb computations of earlier stages.

The construction above yields our first theorem which is (according to the paper [10] in which it first appeared) due to Richard Ladner.

Theorem 1: There exists a recursive oracle set A such that $P(A) \neq NP(A)$.

Proof: Construct A as above.

Suppose L(A) is recognised by some deterministic polynomial time bounded oracle machine $\psi_e(A)$.

Let $\lambda = m_e$. Then there are two cases:

case 1: $0^\lambda \in \psi_e(A_{e-1})$ then there is no stage s+1 at which a string of length less than or equal to $\Psi_e(\lambda)$ is in $A_{s+1}-A_{e-1}$, and since no string of length λ is in A_{e-1} at stage e, we have that $0^\lambda \in \psi_e(A) - L(A)$, or

case 2: $0^\lambda \notin \psi_e(A_{e-1})$ but then $0^\lambda \in L(A_e)$ and hence $0^\lambda \in L(A)$. Moreover since $2^\lambda > \Psi_e(\lambda)$

have that $0^\lambda \notin \psi_e(A_e)$, and since $(\forall e' > e)\ [m_{e'} > \Psi_e(\lambda)]$ we have that $0^\lambda \notin \psi_e(A)$.
In either case $\psi_e(A) \neq L(A)$ contradicting our assumption.

End of proof

Thus we find that separating relativised P from relativised NP requires no big effort. With comparative ease relativised NP can be separated from relativised Co-NP as is shown in the same paper by Baker, Gill and Solovay. In this case we want to diagonalise over the class of nondeterministic machines, so we use the general enumerator to obtain all nondeterministic polynomial time bounded Turing Machines in priority order.

We use the same language $L(X) = \{0^s \mid \exists x \in X. |x| = s\}$ as above, but now instead of constructing A such that L(A) is not entirely recognised by any ψ_e machine, we construct A such that *the complement* of L(A) is not recognised by any of the machines φ_e considered when e ranges over all natural numbers, thus achieving that there is at least one language in (Co- NP) - NP.
We present the construction first.

Requirements: $\forall e : \varphi_e(A) \neq L(A)^c$

Construction:

```
    **stage 0:** $A_0 = \emptyset$; $m_0=0$;
    **stage s+1:** First compute $m_{s+1}$ which is the smallest integer such that $m_{s+1} > \Phi_s(m_s)$ and
            $2^{m_{s+1}} > \Phi_{s+1}(m_{s+1})$.
        If $0^{m_{s+1}} \in \varphi_{s+1}(A_s)$ then choose any accepting computation of $\varphi_{s+1}(A_s)$ on input
            $0^{m_{s+1}}$ and let y be a string of length $m_{s+1}$ which is not queried in this computation.
            Let $A_{s+1}=A_s \cup \{y\}$, otherwise $A_{s+1}= A_s$
    **end of stage s+1**
**end of construction.**
```

If $0^{m_{s+1}} \in \varphi_{s+1}(A_s)$ then $0^{m_{s+1}} \in \varphi_{s+1}(A_{s+1})$ since the string y does not influence *the chosen accepting computation.* On the other hand φ_{s+1} can query only strings of length less than or equal to $\Phi_{s+1}(m_{s+1})$ on input $0^{m_{s+1}}$ and if $0^{m_{s+1}} \notin \varphi_{s+1}(A_s)$ then all strings in A-A_s are of length greater than $\Phi_{s+1}(m_{s+1})$.
From this : $0^{m_{s+1}} \in \varphi_{s+1}(A) \Leftrightarrow 0^{m_{s+1}} \in \varphi_{s+1}(A_s) \Leftrightarrow 0^{m_{s+1}} \in L(A) \Leftrightarrow 0^{m_{s+1}} \notin L(A)^c$. Hence

Theorem 2: There exists a recursive oracle A such that NP(A)≠ Co-NP(A).

The nature of the separation of relativised NP(A), P(A) and Co-NP(A) is left somewhat unclear by these two theorems. An oracle A is constructed such that NP(A) has a language which cannot be recognised by any P(A) machine; but is it just one point, or just finitely many points which escape the computational power of a single P(A) machine? In other words is any subset of this language deterministic polynomial time computable? Or is the difference stronger?

It is certainly so that any *finite* subset of the language is in P(A) as any finite language is deterministically recognizable in constant time by simple enumeration. However it is possible to construct the oracle A in such a way that any *infinite* subset of the language in NP(A)-P(A) is not in P(A). This new demand introduces some constraints on the diagonal method.

To achieve that no subset of L(A) is in P(A) we must ensure that any deterministic Turing Machine operating in polynomial time and accepting some infinite language accepts at least one string in the complement of L(A). Therefore the diagonalisation method cannot, at stage s, fix the oracle in such a way that the requirement is satisfied for machine with index s; it has to wait until machine s accepts some string. Then by fixing this computation and changing the oracle such that this string is *not* in L(A), the requirement can be satisfied.

The diagonalisation method now at stage s considers all requirements with index less than or equal to s, and tries to satisfy one of them. Therefore requirement with index s is satisfied at some stage t which is greater than or equal to s. Because of this behaviour the diagonalisation method used is called *slow diagonalisation* or *wait and see argument.* Formally the construction is described by:

Requirements: $\forall e : |\psi_e(A)| = \infty \Rightarrow \psi_e(A) \cap L(A)^c \neq \emptyset$

Construction

```
    stage 0: $A_0 = \emptyset; m_0 = 0; Req = \emptyset;$
    stage s+1: First compute $m_{s+1}$ which is the smallest integer such that
        $m_{s+1} > \max\{\Psi_i(m_s) \mid i < s\}$ and $2^{m_{s+1}} > \sum_{i<s} \Psi_i(m_{s+1})$ ;
        If $\forall i \in Req : 0^{m_{s+1}} \notin \psi_i(A_s)$ then find a string y such that $|y| = m_{s+1}$ and y is
            not queried by any of the $\psi_i(A_s)$ on input $0^{m_{s+1}}$ for all $i < s$;
            $A_{s+1} = A_s \cup \{y\}$
        else Req := Req - $\{ i \mid 0^{m_{s+1}} \in \psi_i(A_s)\}$;
            $A_{s+1} = A_s$
        endif;
        Req := Req $\cup \{s\}$
    end of stage s+1
end of construction
```

Lemma 1: $\forall i \in \omega : |\psi_i(A)| = \infty \Rightarrow \psi_i(A) \cap L(A)^c \neq \emptyset$

Proof: Consider an arbitrary stage s+1 >i ;either $0^{m_{s+1}} \in \psi_i(A_s)$ but then since $m_{s+2} > \Psi_i(m_{s+1})$ we have that $0^{m_{s+1}} \in \psi_i(A)$ and since $m_{s+2} > m_{s+1}$ we have that $0^{m_{s+1}} \notin L(A)$, and so $\psi_i(A) \cap L(A)^c \neq \emptyset$, or $0^{m_{s+1}} \notin \psi_i(A_s)$ whence $0^{m_{s+1}} \notin \psi_i(A)$. In either case $\psi_i(A)$ can have only finitely many elements of the form $0^{m_{s+1}}$. Since L(A) is a subset of $\{0^{m_s} \mid s \in \omega\}$, this completes the proof.

End of Proof

Lemma 1 states that no infinite subset of L(A) can be recognised by a deterministic polynomial time bounded Turing Machine. For completeness of the argument we must also have that L(A) itself is infinite (or else L(A) itself would be in P(A), and the whole subject would become trivial). To prove this we use a consequence of the padding lemma.

Lemma 2: L(A) is infinite.

Proof: Suppose L(A) is finite. Then at all but finitely many stages the **else** case in the construction is chosen. This means that at all but finitely many stages at least one index is removed from **Req.** This means that **Req** remains bounded in size throughout the construction. However there are infinitely many indices of the empty set, each of which is added to **Req** at is own stage, and is never removed from **Req.**

End of proof

The strong separation between relativised P and relativised NP above is of a more structural kind than the incidental separation described by Baker, Gill and Solovay. A general name for this kind of separation of language classes stems from Recursion Theory:

Definition 1: Let C be a class of languages $C \subsetneq 2^{\{0,1\}^*}$. A language L is *C-immune* if |L| is infinite and there is no $L' \in C$ such that |L'| is infinite and $L' \subsetneq L$.

It was first shown by Bennet and Gill [17] that an oracle A exists such that NP(A) has a P(A) immune set. Later Schöning and Book [109] showed that a *recursive* oracle A exists such that NP(A) has a P(A) immune set. The construction above of course yields a recursive oracle set. Whence:

Theorem 3: There exists a recursive oracle A such that NP(A) has a P(A)-immune set.

A similar strong separation can be achieved between relativised NP and relativised Co-NP. We can construct an oracle A such that Co-NP(A) has an NP(A)-immune set. The complement of this set is by definition an element of NP(A), hence this set can be viewed as an NP(A) set with an NP(A)-immune complement. Such sets are also known from Recursion Theory, and are called "simple" sets:

Definition 2: Let C be a class of languages $C \subsetneq 2^{\{0,1\}^*}$. A language L is C-simple. If $L \in C$ and L^c is C-immune.

A C-immune set is often called a set "immune to C"; a C-simple set is often called a set "simple in C", or just "simple" since it is by definition clear to what class of languages this set belongs. The first construction of an oracle A such that NP(A) has a simple set is due to Homer and Maass [52]. Balcázar [12] later constructed a recursive oracle set by slow diagonalisation. The language used in this construction must have a non-empty intersection with any infinite NP(A) language (This is equivalent to the definition), therefore a language like the one we used above will not suffice. The language used above is a subset of $\{0\}^*$, and can therefore have no intersection with for instance $\{1\}^*$, which is clearly an NP(A) language. Balcázar proposed the following language for all oracles X:

$L(X) = \{0,1\}^* - \{0\}^* \cup \{0^m \mid \exists x \in X : |x|=m\}$.

The construction starts with $A_0 = \{0\}^*$. Hence $L(A_0) = \{0,1\}^*$, which has a nonempty intersection with any language. Now at each stage a string is "added to the complement of the language" if it satisfies no requirement.

Formally the construction is described by:

Requirements:

$\forall e[\ |\varphi_e(A)| = \infty \Rightarrow \varphi_e(A) \cap L(A) \neq \emptyset]$

Construction:

```
    Stage 0: $A_0 = \{0\}^*$; $m_0=0$; $Req_0 = \emptyset$;

    Stage s+1: First compute $m_{s+1}$ which is the smallest integer such that
            $m_{s+1} > \max\{\Phi_i(m_s) \mid i < s\}$ and $2^{m_{s+1}} > \sum_{i<s} \Phi_i(m_{s+1})$ ;
            Let $B_s = A_s - \{0^{m_{s+1}}\}$;
            If $\exists i \in Req_s$ s.t. $0^{m_{s+1}} \in \varphi_i(B_s)$ then
                    for all i such that $0^{m_{s+1}} \in \varphi_i(B_s)$ choose some accepting computation $\gamma_i$,
                    and let y be a string such that $|y| = m_{s+1}$, and y is not queried in any of the
                    chosen $\gamma_i$ ;
                    $A_{s+1}=B_s \cup \{y\}$;
                    $Req_{s+1} = (Req_s - \{i \mid 0^{m_{s+1}} \in \varphi_i(A_{s+1})\}) \cup \{s\}$
            else    $A_{s+1} = B_s$;
                    $Req_{s+1} = Req_s \cup \{s\}$
            endif
    end of stage s+1
end of construction
```

We can easily see that

Lemma 3: $\forall i \in \omega : |\varphi_i(A)| = \infty \Rightarrow \varphi_i(A) \cap L(A) \neq \emptyset$

Proof: suppose $\varphi_i(A) \subsetneq L(A)^c$ and consider an arbitrary stage s+1 > i. First $0^{m_{s+1}} \in \varphi_i(A_{s+1})$ would mean that $0^{m_{s+1}} \in \varphi_i(A)$ since $m_{s+2} > \Phi_i(m_{s+1})$. If the **then** case is chosen we have that $0^{m_{s+1}} \in L(A)$, and then by assumption that $0^{m_{s+1}} \notin \varphi_i(A)$ whence $0^{m_{s+1}} \notin \varphi_i(A_{s+1})$. Hence we may infer from the assumption that i is not removed from Req_{s+1} at *any stage s >i.* This means that the **else** case can only be chosen at stage s+1 if $0^{m_{s+1}} \notin \varphi_i(B_s)$, and then since $A_{s+1} = B_s$ we see that $0^{m_{s+1}} \notin \varphi_i(A)$. So in *both cases* $0^{m_{s+1}} \notin \varphi_i(A)$ and since by assumption $\varphi_i(A)$ consists *only* of strings of the form $0^{m_{s+1}}$, the assumption made leads to $|\varphi_i(A)| \leq i$.

End of proof

The same argument as used before leads to:

Lemma 4: $L(A)^c$ is infinite

Proof: Suppose $L(A)^c$ is finite then the else case in the construction is chosen at only finitely many stages, but at each stage s+1 where the **then** case is chosen an index is in $\mathbf{Req}_s$-$\mathbf{Req}_{s+1}$. Since at each stage only one index is in $\mathbf{Req}_{s+1}$ - $\mathbf{Req}_s$, the cardinality of this set must remain bounded by some constant throughout the construction. However there are infinitely many indices of the empty set...

End of proof

Lemma 3 and 4 together yield

Theorem 4: There exists a recursive oracle A such that NP(A) has a simple set.

The construction presented by Balcázar is one of the few oracle constructions known from the literature in which strings are *removed* from the oracle set at a given stage. Usually such an approach leads to disaster, since the computational classification of such a set is an unachievable goal. The reason why the construction yet leads to a rather tame recursive set is that Balcázar has a moving bound which guarantees that for any length n, there is a stage s in the construction such that membership to the oracle of strings with length less than or equal to n is fixed for all stages greater than or equal to s. Hence membership of an arbitrary string to the oracle can be decided upon in finite time. We will later see that this "trick" can be used to obtain new separation results from existing ones with trivial effort.

There exists a construction of an oracle A such that NP(A) has a simple set which involves only *addition* of strings to the oracle at any given stage. This method however involves a rather complicated diagonalisation method. This method is part of the construction of an oracle A such that NP(A) has a set which is both simple and P(A)-immune, with which construction we will conclude this chapter. The idea of the construction is the same as in [52]. At this point we present the construction merely as an introduction to a complicated construction in an easy context.

First we present the language; for all oracles X:

$L(X) = \{ x \mid \exists y \text{ s.t. } |y|=|x| \ \& \ x\bullet y \in X\}$

In the following the concatenation of x and y is frequently denoted by $x\bullet y$.

Now Starting with the empty set, the method proceeds examining all strings x in Γ^* until it finds some φ_i such that $x \in \varphi_i(A_s)$ with A_s the oracle constructed thus far. (To do this in some predetermined order we need some ordering of Γ^* . Such an ordering is easily obtained e.g. by fixing a meaningless 1 to the left hand side of a string in Γ^* and interpreting the result as a natural number written in binary. In this way we transfer the ordering of the natural numbers to Γ^*. The inverse of this function applied to the natural numbers provides an enumeration of Γ^*.) Then it simply adds a string $x\bullet y$, with $|x|=|y|$, to A_s which is not queried in some accepting computation of $\varphi_i(A_s)$ on input x thus forcing that at stage s+1 this string $x \in \varphi_i(A_{s+1}) \cap L(A_{s+1})$ (and hence $x \in L(A)$). If we can prevent the existence of a stage $t > s+1$ such that $x \notin \varphi_i(A_t) \cap L(A_{t+1})$ then the requirement R_i is satisfied at stage s+1. However there may exist a string $y < x$ and an index j such that for all $u < s$: $y \notin \varphi_j(A_u)$ but changes made to the oracle at stage s+1 to add x to $L(A_{s+1})$ imply that for all $t \geq s+1$: $y \in \varphi_j(A_t) - L(A_t)$. An NP(A) machine may in this fashion grow infinite without being recognised by the diagonalisation. To prevent this from happening we install a *backward search* in the method which reexamines all strings less than or equal to x *after* the addition of x to the language. (i.e. relative to the expanded oracle set). To prevent requirements which are satisfied at earlier stages to become unsatisfied (*injured* is the right term) as a consequence of this backward search, we maintain for each of these requirements a *restraint set* which consists of the strings queried in some chosen accepting computation certifying the satisfaction of this requirement, which were not yet in the oracle at the time this requirement was satisfied. Now it is *only* allowed to add a string to the oracle at stage t to satisfy a requirement R_e, and possibly injure other requirements, if this string is not a member of the restraint sets maintained for requirements $R_{e'}$ with $e' < e$. By this method we ensure that a requirement R_e is injured at most a *finite* number of times. Hence if we have the guarantee that R_e has the possibility of being satisfied *infinitely* many times then R_e will eventually be satisfied at some stage s and never be injured at later stages. Say R_e is *permanently* satisfied at stage s.

The (also required) infinity of the complement of L(A) is not an automatic consequence of the construction as it was in the immunity construction. We have to enforce this property by the installation of requirements also. To do this we introduce an infinite number of requirements (hereby *stratifying* the requirement "$L(A)^c$ is infinite") N_e: $\exists x\ [|x| > e\ \&\ x \notin L(A)]$. These requirements are satisfied by restraining *all* strings of a given (even) length with priority corresponding to index e, and since they can also be injured at most finitely many times, we have the guarantee that infinitely many strings will constitute $L(A)^c$. To prevent clashes, we agree that positive requirement P_e has priority 2e and negative requirement N_e has priority 2e+1. At this point of the discussion however we observe that since all strings in Γ^* are reexamined, the special treatment of the first requirement to be satisfied is unnecessary. Instead we can just at each stage choose some point in Γ^* and look back to 0 from that point to see if some requirement can be satisfied.

The diagonalisation method presented here is characterised by the fact that requirements are injured only a finite number of times. Therefore this method is called a *finite injury method*, which name stems from Recursion Theory. Since it cannot be decided at any finite stage whether a requirement

R_e will be injured at some future stage, this method is also of the *infinite extension type.* (i.e. we need an infinite extension of the oracle to see whether or not a requirement is permanently satisfied. A requirement is injured iff $\exists x \exists s$ such that $x \in A_s \cap$ Restraint set (R_e). Because of the classification of sets in the arithmetical hierarchy, the method presented here is also classified as a 0' method. The diagonalisation algorithm needs at least a 0' oracle to decide upon the satisfaction of requirements. We use at stage s+1 of the diagonalisation below a set $\mathbf{POS}_s$ to store the "positive" requirements which are satisfied by extension of the language with some string. We use a set $\mathbf{NEG}_s$ to store the indices of the "negative" requirements, i.e. requirements of which the satisfaction is used to make $L(A)^c$ infinite. We use a set $\mathbf{RES}_s$ to store the "restraint sets" belonging to requirements, i.e. sets of strings which may not be added to the oracle if the satisfaction of the corresponding requirement is to be preserved. $\mathbf{RES}_s$ consists of pairs (j,V_j) where V_j is a set of strings belonging to priority j. Sets P_1, P_2, N_1,N_2,R_1,R_2, and R_3 are used at each stage to store (proposed) extensions and reductions of the sets $\mathbf{POS}_{s+1}$, $\mathbf{NEG}_{s+1}$ and $\mathbf{RES}_{s+1}$ w.r.t. $\mathbf{POS}_s$, $\mathbf{NEG}_s$ and $\mathbf{RES}_s$

We present the construction.

Requirements:

$\forall e\, [P_e\!: |\varphi_e(A)| = \infty \Rightarrow \varphi_e(A) \cap L(A) \neq \emptyset\,]$

$\forall e\, [N_e\!: \exists x\, [|x| > e\ \&\ x \notin L(A)]]$

Construction

```
Stage 0: $A_0 = \emptyset$ ; $POS_0 = \emptyset$ ; $NEG_0 = \emptyset$; $RES_0=\emptyset$; $x_0=0$;
Stage s+1:    $N_1 := N_2 := \emptyset$;
              $P_1 := P_2 := \emptyset$;
              $R_1 := R_2 := R_3 := \emptyset$;
              $x_{s+1} = x_s+1$;
```

The first level

```
        **Backward Search**
        If ( $\exists y < x_{s+1}$ )( $\exists e < \min NEG_s$) [ $e \in POS_s$ & $y \in \varphi_e(A_s)$ & $2^{|y|} > \Phi_e(|y|)$ ]
        and an accepting computation $\gamma_e$ of $\varphi_e(A_s)$ on input y exists s.t. $\exists z$ with:
                1) $|y| = |z|$
                2) $y \bullet z$ is not queried in $\gamma_e$
                3) $\forall j \le s$ [ $(j, V(j)) \in RES_s$ & $y \bullet z \in V(j)$ ] $\Rightarrow j > 2e$
        Then Let e be minimal in $POS_s$ with these properties, and $\gamma_e$ the
                corresponding computation;
                Let $V_e$ be the set of strings queried in $\gamma_e$;
                $R_1 := \{ (j,V(j)) \in RES_s \mid y \bullet z \in V(j) \}$;
                $P_1 := \{ j \mid (2j, V(2j)) \in R_1 \}$;
                $N_1 := \{ j \mid (2j+1, V(2j+1)) \in R_1 \}$;
                $R_2 := \{(2e, V_e)\}$;
                $P_2 := \{e\}$;
                $A_{s+1} = A_s \cup \{ y \bullet z \}$
        Else {No positive requirement can be satisfied, or the smallest negative
                requirement has higher priority }
                Let $i = \min NEG_s$;
                $N_2 := \{i\}$;
                $R_3 := (2i+1, \{ x_{s+1} \bullet z \mid |z| = |x_{s+1}| \})$;
                $A_{s+1} = A_s$
        Endif
        $POS_{s+1} := ((POS_s \cup P_1) - P_2) \cup \{s\}$ ;
        $NEG_{s+1} := ((NEG_s \cup N_1) - N_2) \cup \{s\}$;
        $RES_{s+1} := ((RES_s - R_1) \cup R_2) \cup R_3$
    End of stage s+1
End of construction
```

The first level

The important difference between this construction and constructions presented earlier is that the satisfaction of requirements is no longer *permanent*. Hence proving that all requirements are satisfied at a given stage no longer suffices for the proof of the required property for L(A).

First we show a so called *injury lemma*, which gives a coarse estimate of the number of stages at which a given requirement is added to the set of requirements to be satisfied.

Lemma 5: Let I(e) denote the number of stages s for which $e \notin \mathbf{POS}_s$ and $e \in \mathbf{POS}_{s+1}$.

Then $I(0) = 1$, and $I(e) \leq 1 + \sum_{f<e} I(f)$.

Proof: First by construction $0 \notin \mathbf{POS}_0$ and $0 \in \mathbf{POS}_1$. Moreover if there is any stage s for which $0 \notin \mathbf{POS}_s$, then for all stages $t \geq s$ $0 \notin \mathbf{POS}_t$, the presence of the string $y \bullet z$ in V(0) makes it impossible to satisfy condition 3) of the backward search.

Next take an arbitrary index $e > 0$ and assume that at stage s +1: $e \notin \mathbf{POS}_s$ and $e \in \mathbf{POS}_{s+1}$ then either $s = e$, or $e \in P_1$ at stage s+1. The latter means that there is a $j < e$ such that $j \in \mathbf{POS}_s$ and $j \notin \mathbf{POS}_{s+1}$. For any particular j, this situation can occur at most I(j) times by definition of I. Hence: $I(e) \leq 1 + \sum_{f<e} I(f)$.

End of proof

By the same argument we show:

Lemma 6: Let N(e) denote the number of stages s such that $e \notin \mathbf{NEG}_s$ and $e \in \mathbf{NEG}_{s+1}$.

Then $N(e) \leq 1 + I(e)$.

Proof: Any index e is in $\mathbf{NEG}_{e+1}$-$\mathbf{NEG}_e$ at stage e+1, and if it is ever in $\mathbf{NEG}_s$-$\mathbf{NEG}_{s-1}$ at stage s>e+1, then an index $i \leq e$ is in $\mathbf{POS}_{s-1}$-$\mathbf{POS}_s$ at the same stage.

End of proof

Now we show that the construction has the required properties. First for the negative requirements.

Lemma 7: $\forall\, e \in \omega\, \exists\, s\, \forall\, t>s\; e \notin \mathbf{NEG}_t$.

Proof: To arrive at a contradiction first suppose that $\exists\, s\, \forall\, t>s\; e \in \mathbf{NEG}_t$. (By lemma 6 this is equivalent to $\forall s\, \exists\, t>s\; e \in \mathbf{NEG}_t$) Without loss of generality we assume that e is the least index with this property. So take a stage u so large that $(\forall\, t \geq u)\; e = \mathbf{min}\, \mathbf{NEG}_t$. From this we have by construction that $(\forall\ t \geq u)\, (\exists i < e)\, [\, i \in \mathbf{POS}_t\, \&\, i \notin \mathbf{POS}_{t+1}]$, which contradicts lemma 5.

End of proof

Since each satisfied negative requirement represents a string permanently in the complement of L(A) this gives.

Corollary 1: $L(A)^c$ is infinite.

The positive requirements are also satisfied.

Lemma 8: $\forall e \in \omega\; |\varphi_e(A)| = \infty \Rightarrow \varphi_e(A) \cap L(A) \neq \emptyset$

Proof: Assume for a contradiction that $|\varphi_e(A)| = \infty\; \&\; \varphi_e(A) \cap L(A) = \emptyset$ By lemma 5 we may assume that $(\forall i \leq e)\, (\exists\, s(i))\, (\forall t > s(i))\, [\, i \in \mathbf{POS}_t \Leftrightarrow i \in \mathbf{POS}_{t+1}]$.

Since $\forall t>s(e)$: $e \notin \mathbf{POS}_t$ would mean that $\varphi_e(A) \cap L(A) \neq \emptyset$, we have that $\forall t>s(e)$: $e \in \mathbf{POS}_t$ by assumption. By lemma 7 there is a stage u such that $(\forall t>u)\, [\mathbf{min}\, \mathbf{NEG}_t > e]$.

Then take $v \geq \mathbf{max}\, \{\, u, \max\, \{\, s(i) \mid i \leq e\, \}\, \}$.

Take $z \in \varphi_e(A)$ so large that :

1) $z > x_{v+1}$

2) $(\forall i < e)\;\; [(i, V(i)\,) \in \mathbf{RES}_{v+1}\;\; \Rightarrow\;\; z > \mathbf{max}\; V(i)]$.

3) $2^{|z|} > \Phi_e(|z|)$

Note that at stage v all positive and negative requirements of priority greater than e are either satisfied, or never will be satisfied. Hence restraint sets belonging to requirements with higher priority will not move after stage v. From this we infer that z satisfies the conditions of a backward search at the first stage t+1 where t+1 > v+1, and $z \in \varphi_e(A_t)$. Now since by assumption we have that e is the least index remaining in $\mathbf{POS}_t$ with this property, we have that $e \notin \mathbf{POS}_{t+1}$ a contradiction

End of proof

We are now ready to present the last theorem of this chapter, which requires probably the most involved construction of this thesis. The result appeared earlier in [123] in a much more difficult (and probably incorrect) form. I am very grateful to K. Ambos-Spies for his comments which lead to the present form of the construction.

For a language we use the same language as above.
For all oracles X: $L(X) = \{ x \mid \exists y \text{ s.t. } |y|=|x| \;\&\; x \bullet y \in X\}$

This language has the nice property that its local density can entirely be controlled by the oracle. Inserting many strings locally into the oracle set means that the language locally becomes very thick, which is needed to acquire the simplicity property, and inserting few strings locally into the oracle results in a locally very thin language, which is needed to acquire the immunity property. As above, the diagonalisation examines all strings of $\{0,1\}^*$ in some predetermined order to find out if a nonempty intersection with one of the machines $\varphi_e(A)$ can be enforced. As above, a backward search is installed to prevent machines to grow infinite without being caught by the method. However we observe that a backward search does not have to involve *all* strings smaller that the string with which the present stage is initiated to guarantee the simplicity property.

On input x machine $\varphi_e(A_s)$ can only query strings of length less than or equal to $\Phi_e(|x|)$. Hence if at stage s a string y is added to the oracle, we only have to reexamine all strings greater than or equal to the *least* x for which $\Phi_e(|x|) \geq |y|$, to make sure the algorithm does not miss φ_e. If indeed a new string is found and the oracle is changed again, then the backward search has to be repeated, but then only for indices of priority higher than e, so the repetition is performed not more than e times. This means that given an index e and a string y, we can compute the length of such a cascade of backward searches as a function of e and y.

If at stage s+1 a string y can be found such that for some string x presently considered: $x \notin L(A_s)$ and $x \in \varphi_e(A_s \cup \{y\}) \cap L(A_s \cup \{y\})$, then we would like to say $A_{s+1} = A_s \cup \{y\}$ and make sure that $x \in \varphi_e(A_t) \cap L(A_t)$ for all $t \geq s+1$. The observation that the cascade of backward searches reaches back to a border which can be computed on forehand makes this possible. Simply prevent the diagonalisation from adding a string to the oracle at stage s+2, which would imply a backward search reaching back beyond $\Phi_e(|x|)$.

The thus created gap may hide strings from the method with which other requirements of higher priority may be satisfied, but this can be overcome by installing also a forward search stretching out over this gap. As in the case of a backward search finding a new string to be put in the oracle may make repetition of the forward search necessary with new computed borders, but again only for indices with higher priority (< e). So in this case also a border can be computed which given the first index e, and the first string y will certainly not be crossed.

The diagonalisation can now at stage s *given the last border* try to find an unsatisfied requirement e and a string x, such that if x is added to $L(A_{s+1})$ at the next stage, a cascade of backward searches would not reach back beyond this border, and the requirement with the least possible index is satisfied in the region between this and the next computed border. Note that this computed border depends not only on the border of stage s, but also on the index e of the first requirement to be satisfied at stage s+1. Perhaps the idea is best illustrated with a picture:

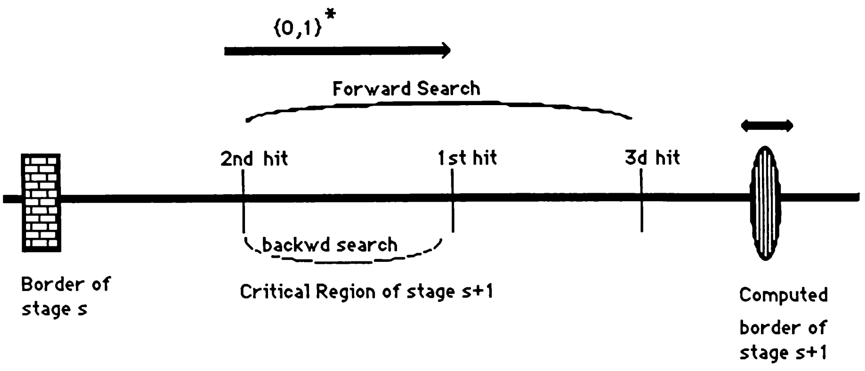


Fig 4: Harmonica method in progress

At this point the reader may wonder why all this complicated forward and backward searching is necessary, if the simplicity property can be established installing only one backward search as is indicated above. The answer is of course that the method using one (unlimited) backward search at each stage *seems* to disallow the establishment of the immunity property.

The immunity property enforces that for certain indices with high priority corresponding to deterministic machines some strings may not be put in the language if they are recognised by this machine. However a diagonalisation method using a single backward search may find at stage s that a string x is not recognised by any of the deterministic machines under consideration, and consequently change the oracle such that this string will be eventually in L(A). After a successful backward search the oracle may be changed such that x *is* recognised by some deterministic machine. At that point changing the oracle such that x is no longer in L(A) may lead to complications which we cannot oversee.

An at first sight rather complicated construction of backward and forward searches however makes it possible to ensure for an ever growing number of (and hence eventually all) indices of deterministic machines that if a given string is put in the language then it will never be recognised at a later stage by any of these machines. To see this we need the observation (and we will derive this below) that by definition of the critical region the *difference* in length of the longest and the shortest string in this region is polynomially bounded.

Before we present the construction we will explain this in some more detail. Suppose y is the first string added to the oracle in an attempt to satisfy a requirement with index e. Let ξ_e be a polynomial such that $\xi_e(x) \geq \Phi_{e'}(x)$ for all x and all $e' \leq e$. If y is the first string added to the oracle at stage s+1 to satisfy requirement with index e, then the shortest string examined at this stage need have a length no shorter than **min** $\{ x \mid \xi_e(\xi_e(\ldots(\xi_e(x))\ldots)) \geq |y| \}$, where ξ_e is iterated e times. Given a border b, we wish to preserve the invariant that no string of length less than or equal to b is examined as a candidate for the oracle set (hence preserving computations of satisfied requirements at stage s+1). By the argument above we know that this will exclude strings y from being a candidate for examination, having a length such that **min** $\{ x \mid \xi_e(\xi_e(\ldots(\xi_e(x))\ldots)) \geq |y| \} < b$. So these strings will be candidate for a forward search before stage s+1 is closed. A new border is computed from index e, candidate string x, and the polynomials bounding the computations for machines (deterministic and non-deterministic) with index less than or equal to e. Anyway given a candidate string x, the proposed border b is polynomial in the length of x. So the forward search stretches out to strings of length bounded by a polynomial in the length of x.

The forward search is repeated at most e times at stage s+1. Hence the maximal length of strings *actually* considered in a forward search is less than or equal to a polynomial in the length of x, where x is the first string considered. Given the fact that |x| *itself* is less than or equal to a polynomial in the length of the shortest string considered in a backward search, we arrive at the conclusion that the lengths of both the longest and shortest string considered at stage s+1 have polynomial relations which can be computed from e, the first candidate index of a nondeterministic machine for satisfaction at this stage, and x, the candidate string for this satisfaction. We will later see that the relation between the longest and the shortest string can be formulated only depending on the length of these strings and e. In **Fact 5** below we will get back to this issue, when we have developed a more precise notation.

At this point we assume to have shown this polynomial relation and wish to draw the readers attention to an important consequence of this fact. Suppose for given pair (e,x) we compute at stage s+1 that m is the minimal length of strings, and n is the maximum length of strings considered at stage s+1. By the argumentation above there exists a polynomial ξ_e such that $\xi_e(m) \geq n$. Moreover ξ_e *depends only on e*. If an attempt is made to satisfy index e at stage s+1, then the backward and forward searches are executed only to search to satisfy a requirement with index e' < e, and the searches stop when this fails for the first time. Consequently stage s+1 will consist of no more than e substages (to prevent confusion we will speak of a new *substage* of stage s+1 each time a search as described above is performed), and $|A_{s+1}-A_s| \leq e$ since only one string has to be added to satisfy a positive requirement. By the arguments presented above we have to choose m so large that $m > b_s$ (b_s is the border of stage s).

Suppose we choose m *so large* that $2^m > e\times(\sum_{i<e}\Psi_i(n)+\mathbf{max}\{\Phi_i(n) \mid i \leq e\})$. Then on e different strings of length less than or equal to n all deterministic machines with index < e *together* cannot query all available strings of length m' where m' ≥ m, moreover there's no nondeterministic machine with index ≤ e that can query all available strings of length m on input of e different strings of length m. (allowed of course *one specific* computation on each of these strings.) This is in fact the crucial observation which gives the algorithm "room to diagonalise". We will also get back to this point in **Fact 5** below.

The final observation we want to make before starting the formal part of the construction is that forward and backward searches may be merged. Instead of first performing a backward search and then performing a forward search we may compute at some substage of stage s+1 a *search window* in which we search for a requirement of higher priority to be satisfied. This probably does not alter the construction essentially. However we think that it simplifies the intuition, and it certainly helps to clean up proofs.

To specify the polynomials mentioned above we define the following *border computing polynomials*:

1) Let k(e) be the least natural number > 1 such that

$$\forall n \in \omega.\mathbf{max}\{\Psi_i(n), \Phi_i(n), 2(n) \mid i \leq e\} \leq n^{k(e)} + k(e)$$

2) For m,n $\in \omega$ and fixed e define:

inductively: $\xi_e(1,n) = n^{k(e)} + k(e)$;

$$\xi_e(m,n) = (\xi_e(m-1,n))^{k(e)}+k(e);$$

3) For n,e $\in \omega$ let fmin(e,n) $= \mathbf{min}\ \{ m \mid \xi_e(e,m) \geq n \}$

4) For m,n $\in \omega$ and fixed e define

inductively: $\zeta_e(1,n) = \xi_e(e, n^{k(e)}+k(e)+1)$

$$\zeta_e(m,n) = \xi_e(e, \zeta_e(m-1,n))$$

5) For n,e $\in \omega$ let fmax(e,n) $= \zeta_e(e,n)$

6) Finally for n,e $\in \omega$ let

$$\text{fsat}(e,n) = \mathbf{min}\{m \mid \xi_e(e,n) < m \ \& \ 2^{\text{fmin}(e,m)} > \text{ex}\ (\textstyle\sum_{i \leq e}\Phi_i(\text{ fmax } (e,m))\} + \mathbf{max}\{\Phi_i\ (\text{fmax } (e,m)) \mid i \leq e\}\ )$$

$$\}$$

An informal interpretation for these functions is as follows:

Suppose at stage s+1 we wish to satisfy a requirement with index e by adding a string x to the language at this stage. We know that:

1) fmin(e, $|x|$) gives minimal length of strings consequently examined in searches

2) fmax(e, $|x|$) gives the maximal length of strings consequently examined in searches

3) if $|x| \geq$ fsat(e,b) for some b, then fmin(e,$|x|$) > b and not all strings of length fmin(e,$|x|$) are queried during a set of $\leq e \times (e+1)$ computations simulated at substages of stage s+1

We will now present the construction:

A set of indices $\mathbf{POS_s}$ is maintained to represent the positive requirements corresponding to NP(A) languages (we call these requirements positive because they require expansion of the oracle to be satisfied.) Likewise a set $\mathbf{NEG_s}$ is maintained to represent the negative requirements. At each stage s+1, a new index s is added to both sets. Initially A_0 is empty, and the border of stage 0 is set to 0 (b_0). Whenever a positive requirement e is found which has the possibility of being satisfied, then an attempt is made to expand A_s accordingly. Stage s+1 then consists of $\leq$ e substages at which forward and backward search is performed.

Requirements:

$(\forall e)\ [P_e : |\varphi_e(A)| = \infty \Rightarrow \varphi_e(A) \cap L(A) \neq \emptyset]$

$(\forall e)\ [N_e : |\psi_e(A)| = \infty \Rightarrow \psi_e(A) \cap L(A)^c \neq \emptyset]$

Construction:

```
Stage 0: $A_0 = \emptyset$; $b_0 = 0$; $POS_0 = \emptyset$; $NEG_0 = \emptyset$;
Stage s+1:
    If $\exists e \in POS_s$ and $\exists x \in \{0,1\}^*$ such that $x \in \varphi_e(A_s)$ and $fsat(e,b_s) \leq |x| \leq s$
    Then
        Step 1)
            Let $e_0$ be minimal in $POS_s$ with this property and $x_0$ the corresponding x.
            Let $X_0 = \emptyset$;  {Strings to be added to the oracle at the end of stage s+1}
        $Y_0 = \emptyset$; {Set of strings restrained at stage s+1}
        i=0;     {Index of the substage}
        $bs_0 = b_s$;{Computed border; candidate for $b_{s+1}$}

    Step 2)
        $\varepsilon = e_i$; $\gamma_i$ = some accepting computation of $\varphi_\varepsilon (A_s \cup X_i)$ on input $x_i$;
        $Y^1 = [\bigcup_{j \leq \varepsilon}${strings queried in the computation of $\psi_j(A_s \cup X_i; x_i)$ on
                   input $x_i$ which are not in $A_s \cup X_i$ } ] $\cup Y_i$
        If $(\forall j \in NEG_s)$: $[x_i \notin \psi_j(A_s \cup X_i)$ or $j > \varepsilon]$
        Then   $Y^2$ = { strings queried in $\gamma_i$ which are not in $A_s \cup X_i$ };
               $Y_{i+1} = Y_i \cup Y^1 \cup Y^2$;
               Let $y_i$ be a string such that: (See Fact 5 below)
               1) $y_i \notin Y_{i+1}$
               2) $x_i \in \varphi_\varepsilon (A_s \cup X_i \cup \{y_i\}) \cap L(A_s \cup X_i \cup \{y_i\})$
               and set:
               $X_{i+1} = X_i \cup \{y_i\}$;
               $bs_{i+1} = \max \{ \max \{|z| \mid z \in Y_{i+1}\}, bs_i\}$ ;
        Else $X_{i+1} = X_i$ ; $Y_{i+1} = Y_i \cup Y^1$;
               $bs_{i+1} = \max \{ \max \{|z| \mid z \in Y_{i+1}\}, bs_i, 2 \times |x_i| \}$;
               $\varepsilon = \min \{ j \in NEG_s \mid x_i \in \psi_j (A_s \cup X_i) \}$
        Endif

    Step 3 : {Search}
        If $X_{i+1} \neq X_i$ Then $min_{i+1} = \min\{ k \mid (\exists j < \varepsilon) [\Phi_j(k) \geq \min (X_{i+1} - X_i)] \}$
        Else $min_{i+1} = x_i$
        Endif;
        $max_{i+1} := \max \{fsat(j, bs_{i+1}) \mid j < \varepsilon \}$;
```

```
            If $\exists(e < \varepsilon) \in POS_s$ and $\exists y$ s.t:
                $min_{i+1} \le |y| \le max_{i+1}$ & $y \in \varphi_e(A_s \cup X_{i+1})$
            Then let $e_{i+1}$ be the *minimal* index with this property, and and set $x_{i+1}$
                to one of the values of such an y corresponding to $e_{i+1}$.
                Perform steps 2 and 3 with i=i+1
            Endif;
        Step 4:
            $b_{s+1} = bs_{i+1}$;
            $A_{s+1} = A_s \cup X_{i+1}$;
            $POS_{s+1} = (POS_s - \{ j \in POS_s \mid \exists x [ x \in \varphi_j(A_{s+1}) \cap L(A_{s+1})$
                        $\& \Phi_j(|x|) \le b_{s+1}]\} ) \cup \{s\}$;
            $NEG_{s+1} = (NEG_s \cup \{ j \in NEG_s \mid \exists x [ x \in \psi_j(A_{s+1}) \cap L(A_{s+1})^c$
                        $\& \Psi_j(|x|) \le b_{s+1}]\} ) \cup \{s\}$;
    Else { No positive requirement can be satisfied at this stage }
        $A_{s+1} = A_s$; $POS_{s+1} = POS_s \cup \{s\}$; $NEG_{s+1} = NEG_s \cup \{s\}$; $b_{s+1}=b_s$
End of stage s+1
End of Construction
```

It remains to be shown that A constructed as above indeed has the property that L(A) is both simple and P(A)-immune. This requires some effort. First we state a few more or less trivial facts about the construction:

Fact 1: For any index e and stage s+1: if e is in $POS_s - POS_{s+1}$ ($NEG_s - NEG_{s+1}$) then there is no stage t+1>s+1 such that $e \in POS_t - POS_{t+1}$ ($NEG_t - NEG_{t+1}$)

Proof: An index can only be removed if it is added at a previous stage, and this happens for index e not before stage e+1. Hence any s+1 for which $e \in POS_s - POS_{s+1}$ must be *greater than* e+1. The only index added to POS_{t+1} at stage t+1 is index t for which t > s> e.

End of proof

Fact 2: For any index e and stage s+1 : if $e \in$ $\mathbf{POS}_s$ - $\mathbf{POS}_{s+1}$ ($e \in$ $\mathbf{NEG}_s$ - $\mathbf{NEG}_{s+1}$) then $\varphi_e(A) \cap L(A) \neq \emptyset$. ($\psi_e(A) \cap L(A)^c \neq \emptyset$)

Proof: If $e \in$ $\mathbf{POS}_s$ - $\mathbf{POS}_{s+1}$ ($e \in$ $\mathbf{NEG}_s$ - $\mathbf{NEG}_{s+1}$) then (cf **step 4**) $\exists x \in \varphi_e(A_{s+1}) \cap L(A_{s+1})$: [$\exists x \in \varphi_e(A_{s+1}) \cap L(A_{s+1})^c$] moreover b_{s+1} is set to a value of at least $\Phi_e(|x|)$ ($\Psi_e(|x|)$), which ensures that this fact is maintained for all following stages.

End of proof

Fact 3: The search windows are overlapping i.e. $max_{i+1} \geq min_i$ and $min_{i+1} \leq max_i$

Proof: $x_{i+1} \leq max_{i+1}$ so $min_{i+2} \leq max_{i+1}$; $x_{i+1} \geq min_{i+1}$ so $max_{i+2} \geq min_{i+1}$ **End of proof**

Fact 4: For any pair e,m if $2^{fmin(e,m)} > e \times(\sum_{i<e} \Psi_i(fmax(e,m)) + \mathbf{max}\{ \Phi_i(fmax(e,m)) \mid i \leq e\})$ then $2^{fmin(e',n)} > e' \times(\sum_{i<e'} \Psi_i(fmax(e',n)) + \mathbf{max}\{ \Phi_i(fmax(e',n)) \mid i \leq e'\})$ for any $e' \leq e$ and any $n \geq m$

Proof: By definition of fmin we have for fixed e that $\xi_e(e,fmin(e,m)) \geq m$ for any m. By definition of fmax for fixed e and arbitrary n: $\zeta_e(e,n) = fmax(e,n)$.Hence we see that the *polynomial* $p_e(x) = \zeta_e(e, \xi_e(e,x))$ has the property that $p_e(fmin(e,x)) \geq fmax(e,x)$ for any x. Moreover $p_{e'}(x) \leq p_e(x)$ where $e' \leq e$ by definition of k(e). So once by simple "exponential vs polynomial growth" once we have that:

$$2^{fmin(e,m)} > e \times(\sum_{i<e} \Psi_i(fmax(e,m)) + \mathbf{max}\{ \Phi_i(fmax(e,m)) \mid i \leq e\})$$

we have that

$$2^{fmin(e',n)} > e' \times(\sum_{i<e'} \Psi_i(fmax(e',n)) + \mathbf{max}\{ \Phi_i(fmax(e',n)) \mid i \leq e'\})$$

for all $e' \leq e$, and $n \geq m$.

End of proof

The final observation to be made is our "there's room to diagonalise" observation **Fact 5**. It follows almost immediately from **Fact 4**.

Fact 5: For any stage s+1: if stage s+1 has an i-th substage, then y_i can be found.

Proof: Since the desired length of y_i is $2|x_i|$ it suffices to show that there can be no substage s at which $| Y_{i+1} \cap \Gamma^{2|x_i|} | \geq 2^{2|x_i|}$. We first derive that there is an upperbound on the length of strings x_i considered at any substage i.

Recall that fsat(e,n) is the minimal m such that:

1) $\xi_e(e,n) < m$

2) $2^{\text{fmin}(e,m)} > e\times ((\sum_{i\leq e}\Phi_i(\text{fmax}(e,m))) + \mathbf{max}\{\Phi_i(\text{fmax}(e,m)) \mid i\leq e\})$

Since *all* $bs_i \geq b_s$ and *all* $e_i \leq e_0$ we have by **Fact 4** that in **Step 3** whenever $\xi_{e'}(bs_i) < |z|$, then consequently $|z| \geq \text{fsat}(e,bs_i)$. Hence at each substage i we find that $\max_{i+1} \leq \xi_{e'}(bs_i)$. From the definition of fmax we can now easily infer that there can be no substage i at which $\max_{i+1} > \text{fmax}(e_0, |x_0|)$. Hence any x_{i+1} will be $\leq \text{fmax}(e_0, |x_0|)$ (Since by definition $n^{k(e)}+k(e)$ also maximises $2|x_i|$).

We can now derive an upperbound for $|Y_{i+1}|$. For notation let $\varepsilon = e_0$; $\iota = e_i$. Now at any substage i we have that:

1) $|Y^1| \leq \sum_{i<\iota}\Psi_i(x_i) \leq \sum_{i<\varepsilon}\Psi_i(\text{fmax}(\varepsilon,|x_0|))$

2) $|Y^2| \leq \Phi_\iota(|x_i|) \leq \mathbf{max}\{\Phi_j(\text{fmax}(\varepsilon, |x_0|)) \mid j\leq\varepsilon\}$

So since there are no more than e substages we have that

$|Y_{i+1}| \leq \varepsilon\times ((\sum_{i<e}\Psi_i(\text{fmax}(\varepsilon, |x_0|))) + \mathbf{max}\{\Phi_j(\text{fmax}(\varepsilon, |x_0|)) \mid j\leq\varepsilon\})$

On the other there can be no substage i at which $|x_i| \leq \text{fmin}(e,|x_0|)$ as can be easily seen from the computation of $\min_{i+1}$ in **Step 3** and the definition of fmin. So for all substages i we have that $2^{|x_i|} \geq 2^{\text{fmin}(\varepsilon,|x_0|)} > |Y_{i+1}|$ which was to be proven.

End of proof

Now we show first that no infinite P(A) language can be entirely within L(A).

Lemma 9: $\forall e \in \omega$: $|\psi_e(A)| = \infty \Rightarrow \psi_e(A) \cap L(A)^c \neq \emptyset$.

Proof: Assume for a contradiction that $|\psi_e(A)| = \infty$ and $\psi_e(A) \subsetneq L(A)$. First observe that if a string is added to L(A) to satisfy a positive requirement $i \in \mathbf{POS}_s$ at some stage s+1, then the corresponding computation is preserved by expansion of Y^2 at the corresponding substage, and its index is consequently in $\mathbf{POS}_s$-$\mathbf{POS}_{s+1}$ in **step 4**, and so $(\forall t>s)$ $[i \notin \mathbf{POS}_t]$ If we have that $\psi_e(A)$ is infinite within L(A) then there must be infinitely many stages s +1 such that:

$(\exists i \in \mathbf{POS}_s)$ $[\exists x \in (L(A_{s+1}) - L(A_s)) \cap \varphi_i(A) \cap \psi_e(A)]$, since strings are only added to L(A) to satisfy positive requirements. And since each positive requirement can be removed only once (Fact 1), all these indices are different, whence infinitely many indices in this row are greater than e. Consider the first x in this sequence for which i > e. We will argue that x cannot be in $L(A) \cap \psi_e(A)$.

The string x is added at the j-th substage of stage s+1 in **Step 2** of the diagonalisation when a set X_j is constructed such that $x \in \varphi_i(A_s \cup X_j)$. Either $x \in \psi_e(A_s \cup X_j)$ or $x \notin \psi_e(A_s \cup X_j)$. In the former case x is not in $L(A_{s+1}-A_s)$, and b_{s+1} is at least equal to $2\times|x|$, whence $x \notin L(A)$, and in the latter case all strings which are queried in the computation of $\psi_e(A_s \cup X_j)$ are in Y^1 whence $x \notin \psi_e(A)$. So an infinite sequence as presumed cannot exist.

End of proof

We can also show that positive requirements are satisfied if the corresponding machines recognise an infinite language. In lemmas 10 and 11 we will first show that if a given positive requirement has the possibility of being satisfied infinitely many times, then it will eventually be satisfied.

Lemma 10: If the diagonalisation algorithm enters **Step 2** at the i-th substage of stage s+1 with $e_i = e$, then for at least one requirement $e' \in \mathbf{POS}_s \cup \mathbf{NEG}_s$ with $e' \leq e$: e' is removed from $\mathbf{POS}_s$ or $\mathbf{NEG}_s$ in **Step 4** of stage s+1.

Proof: if the diagonalisation enters step 2 at the i-th substage of stage s+1 with i=m then from this point in the diagonalisation until step 4 is reached (and stage s+1 is closed) with i=n either:

1) $X_{n+1} = X_m$ (i.e. no extra string will be added to the oracle)

or 2) $X_{n+1} \supset X_m$ (i.e. at least one string will be added to the oracle)

In case 1 there's an $e' < e$ such that $e' \in \mathbf{NEG}_s$-$\mathbf{NEG}_{s+1}$ since x_m must have been recognised by $\psi_{e'}(A_s \cup X_m)$ and certainly no string queried in the corresponding computation will be in any A_t for $t \geq s+1$ moreover since $b_{s+1} \geq 2 \times |x_m|$ we have that $x \notin L(A_{s+1})$; and in case 2 at least one of the positive requirements is satisfied, since the accepting computation is always protected by Y^2 whenever X_i is altered.

End of proof

Lemma 11: For a given positive requirement P_e there can only be finitely many stages s+1 at which the diagonalisation algorithm enters **step 2** in the i-th substage with $e_i = e$.

Proof: Assume the contrary. Then there must exist an infinite sequence of stages $(s_j+1)_{j \in \omega}$ at which the diagonalisation enters **Step 2** in the i-th substage with e_i=e. (Here i depends on s_j) According to lemma 10 in **step 4** of each of the stages s_j+1 at least one of the requirements $\{P_{e'} \mid e' \leq e\} \cup \{N_e \mid e' \leq e\}$ is removed from either **POS** or **NEG**. Since there are only finitely many indices e' < e, at least of these indices must be removed an infinite number of times, contradicting fact 1.

End of proof

In lemmas 12, 13 and 14 we present two possible "types of behaviour" of NP(A) languages. There are languages which "stay close to the diagonalisation" i.e. there are strings in these languages only around the border or (worse) before the border line for all but finitely many stages. Or there are the (more friendly) type of languages which have strings far beyond the border. The first type is discussed in lemma 12, the second type in lemma 13 and 14. It turns out that both types of languages are "caught" by the diagonalisation.

Lemma 12: For a given positive requirement P_e there can only be finitely many stages s+1 such that

1) $e \in \mathbf{POS}_s$

2) $b_{s+1} > b_s$

3) For the some substage, say the i-th substage of stage s+1 $\exists x \in (\varphi_e(A_s \cup X_i) \cap \varphi_e(A_{s+1}))$ with $\min_{i+1} \leq |x|$ and $|x| \leq \text{fsat}(e, b_{s+1})$

Proof: Assume an infinite sequence. At all stages where $b_{s+1} > b_s$ the diagonalisation algorithm must have entered **Step 2** at least once, and consequently have removed an index from either **POS** or **NEG** in **Step 4**. Only finitely many indices can be less than e. We may therefore assume the existence of a stage s+1 at which no requirement with higher priority than P_e can be satisfied. Without loss of generality we assume that $x \in \varphi_e(A_s \cup X_j)$ for all $j \geq i$. If not then there is a maximal j such that $x \in \varphi_e(A_s \cup X_j) - \varphi_e(A_s \cup X_{j-1})$ and for this substage $|x| \geq \min_{j+1}$, and we take i=j. Since by assumption any index e' removed from **POS** or **NEG** in **Step 4** is greater than e, we have that for the last substage of stage s+1, say the n-th substage that $\max_{n+1} \geq$ **max** $\{fsat(j,b_{s+1}) \mid j \leq e_0\} \geq |x|$, since $e_0 \geq e$ by assumption. (Recall that $b_{s+1}=bs_{n+1}$, and hence that $\max_{n+1} \geq fsat(e,b_{s+1})$) By **Fact 3** there is a k with $i \leq k \leq n$ such that $\min_{k+1} \leq |x| \leq \max_{k+1}$ and at this substage P_e is the least requirement which fits the conditions of **Step 3** and will consequently be in $\mathbf{POS}_s$-$\mathbf{POS}_{s+1}$

End of proof.

Lemma 13: For a given positive requirement P_e there can only be finitely many stages s+1 such that

1) $e \in \mathbf{POS}_s$

2) $\exists x \in \varphi_e(A_s) \cap \varphi_e(A_{s+1})$ such that $fsat(e,b_s) < |x| \leq fsat(e,b_{s+1})$

Proof: Assume an infinite sequence. As in lemma 12 there are only finitely many stages in this sequence at which requirements with higher priority can be satisfied. Hence without loss of generality we may assume that no requirement of higher priority can be satisfied at stage s+1.

There are two cases:

case 1: $|x| \leq s$ Then e is the least index which satisfies the conditions of **Step 1**. So according to lemma 11 and the assumption of an infinite sequence this case can occur only finitely many times.

case 2: $|x| > s$. Since $fsat(e,b_{s+1}) > fsat(e,b_s)$ and hence $b_{s+1} \neq b_s$, we have that at least one requirement is satisfied at stage s, and hence the diagonalisation algorithm must have entered **Step 2** at least once. Let (e',y) be the pair satisfying the conditions of **Step 1** when **Step 2** is first entered. Since $|y| \leq s$, we have that $|y| < |x|$. Since by assumption we also have that $e'>e$ we have that $\max_{i+1} > \mathrm{fsat}(e,bs_{i+1})$ for all substages i. First we argue that $x \in \varphi_e(A_s \cup X_i)$ for all substages i of stage s+1. Suppose the contrary then since $x \in \varphi_e(A_{s+1})$, there is a substage j (maximal) such that $x \in \varphi_e(A_s \cup X_{j+1}) - \varphi_e(A_s \cup X_j)$, consequently $x > \min_{j+1}$ and infinite occurrence of this situation is defied by lemma 12. Now if $|y| < |x|$ then $|x| > \min_1$and since $|x| \leq \mathrm{fsat}(e,b_{s+1})$ we get exactly the same result, namely the existence of a substage k with $0 \leq k \leq n$ such that $\min_{k+1} \leq |x|$ and infinite occurrence of this situation is defied by lemma 12. We must conclude that **case 2** can also occur only finitely many times, and that an infinite sequence as presumed cannot exist.

End of proof.

Lemma 14: For given positive requirement P_e there are only finitely many stages s+1 such that:

1) $e \in$ **POS**$_s$

2) $(\exists x \in \varphi_e(A_s))\ [\ x \geq \mathrm{fsat}(e,b_s)\ \&\ (\forall t \geq s)\ [\ x \in \varphi_e(A_t)\ ]\ ]$

Proof: Assume for a contradiction that e is the least index such that there are infinitely many stages s+1 such that $(\exists x \in \varphi_e(A_s))\ [\ x \geq \mathrm{fsat}(e,b_s)\ \&\ (\forall t \geq s)\ [x \in \varphi_e(A_t)]\ ]$ (*) and $e \in$ **POS**$_s$. By this assumption there is a stage s_0 such that for all $u > s_0$ and all $e' < e$ either $e' \notin$ **POS**$_u$ or $(\forall x \in \varphi_e(A_s))\ [\ (\forall t \geq s)\ [x \in \varphi_e(A_t) \Rightarrow x < \mathrm{fsat}(e,b_s)]\ ]$ (**)

Choose an arbitrary stage $s+1 > s_0$ and let x_{s+1} be the least x satisfying (*). Assume $|x_{s+1}|=t$. Either:

case 1 $t \leq s$, but since in this case the e is the least index satisfying the conditions of **Step 1**, **Step 2** is entered at such stages with $e_0=e$. Lemma 11 then defies the existence of an infinite sequence.

case 2 $t>s$. In this case consider stage t+1. Since $t>s$ we have that $t > s_0$ and hence that (**) holds for t and all $e' < e$ at stage t+1. Either:

case 2a: $|x_{s+1}| \geq \mathrm{fsat}(e,b_t)$, but then again we have that step 2 is entered with $e_0=e$, and we cannot have such a sequence of stages t+1 by Lemma 11.

case 2b: $|x_{s+1}| < \text{fsat}(e,b_t)$ Hence we have that there exists an infinite sequence of stages s+1 and t+1 such that $(\exists x \forall u \geq s\ [x \in \varphi_e(A_u)\ \&\ |x| > \text{fsat}(e,b_s)\ \&\ |x| \leq \text{fsat}(e,b_t)\ ]$. We infer that for any such pair s+1, t+1 there must exist a stage v+1 such that $s \leq v \leq t-1$ and $x \in \varphi_e(A_v)\ \&$ $x \in \varphi_e(A_{v+1})\ \&\ |x| > \text{fsat}(e,b_v)\ \&\ |x| \leq \text{fsat}(e,b_{v+1})$. An infinite sequence of these stages v+1 however contradicts lemma 13.

End of proof.

Now it is time to show that the types of languages discussed in lemmas 12, 13 and 14 are indeed the only types of infinite NP(A) languages. This is done in lemmas 15 and 16. Lemma 15 first states 2 facts which may seem trivial. In lemma 16 then, the actual work is done. Due to the right preparation this work is limited to a few lines.

Lemma 15: For any NP(A) language $\varphi_e(A)$:

1) $\forall x\ [x \in \varphi_e(A) \Leftrightarrow x \in \varphi_e(A_s)$ for all but finitely many s].

2)$\forall x\ [x \in \varphi_e(A) \Rightarrow [\ \Phi_e(|x|) > \mathbf{min}\{|y| \mid y \in A_{s+1}\text{-}A_s \mid s = \mathbf{max}\ \{t \mid x \notin \varphi_e(A_t)\ \}\}$

Proof: 1) We show that b_s moves to infinity with s, and hence that eventually $b_s > \Phi_e(|x|)$. As no string of length less than or equal to b_s is in $A_s\ \text{-}A_{s-1}$ for any s, this means that $x \notin \varphi_e(A)$ if $x \notin \varphi_e(A_s)$ when $b_s > \Phi_e(|x|)$.

Suppose that for all but finitely many s: $b_{s+1} = b_s = C$ for some constant C. Consider the language $L = \{x \mid x > C\}$. Evidently L is an NP(A) language represented by some positive requirement P_e and P_e is never satisfied. L certainly has elements x with $|x| > \text{fsat}(e,b_s)$ and hence **Step 2** of the diagonalisation algorithm will be entered with $e_0 \leq e$ when stage $t+1 = |x|+1$ is reached, since at least one of the requirements in $\mathbf{POS}_t$ (namely e) satisfies the conditions of **step 1**. But then $b_{t+1} \geq |x| > b_s$.

2) If $x \in \varphi_e(A_s)$ for all s then then were done, else since $x \in \varphi_e(A)$ there is a maximal t such that $x \notin \varphi_e(A_t)$ by 1) Assume that for this t: $\Phi_e(|x|) > \mathbf{min}\{|y| \mid y \in A_{t+1}\text{-}A_t\}$ then the computation of $\varphi_e(A_t)$ on input x is the same as the computation of $\varphi_e(A_{t+1})$ on input x whence $x \notin \varphi_e(A_{t+1})$, a contradiction.

End of proof

Lemma 16: For all e: if $|\varphi_e(A)| = \infty$ then $L(A) \cap \varphi_e(A) \neq \emptyset$

Proof: By lemma 15 for all $x \in \varphi_e(A)$ there is a maximal stage s(x) such that $x \notin \varphi_e(A_{s(x)})$.

There are two cases:

case1: For all but finitely many x,y s.t. $x \in \varphi_e(A)$ & $y \in \varphi_e(A)$ we have $b_{s(x)} = b_{s(y)}$. Then let $t = \mathbf{max}\{ s(x) \mid x \in \varphi_e(A) \}$. Then for all stages $s+1 > t+1$ we have for all $x \in \varphi_e(A)$ that $x \in \varphi_e(A_s)$. Then since $\varphi_e(A)$ is infinite we can choose for each of these stages s+1 an x(s+1) such that $\varphi_e(A_t)$ for all $t \geq s$ and $x \geq \mathrm{fsat}(e,b_s)$. By lemma 14 there can only be finitely many of these stages s+1 at which $e \in \mathbf{POS}_s$.

case 2: There is an infinite sequence $S = (x_i)_{i \in \omega}$ s.t. $x_i \in \varphi_e(A_s)$ and $s(x_{i+1}) > s(x_i)$. Let $X = \{ s(x) \mid x \in \varphi_e(A_s) \ \& \ x \geq \mathrm{fsat}(e,b_{s(x)}) \}$ and $Y = S - X$; Since $|S| = \infty$, we have that either $|X| = \infty$ or $|Y| = \infty$. But if $|X| = \infty$ then X is an infinite sequence of stages satisfying the conditions of lemma 14, and hence there can only be finitely many of these stages s(x)+1 at which $e \in \mathbf{POS}_s$. If on the other hand $|Y| = \infty$ then we have for all of the stages $s(x) \in Y$, that $x \geq \mathbf{min} \{|y| \mid y \in A_{s(x)+1} - A_{s(x)} \}$ and hence there can only be finitely many of these stages s(x)+1 at which $e \in \mathbf{POS}_s$ by lemma 12.

End of proof

It remains to show that both L(A) and $L(A)^c$ are infinite. This is however an almost immediate consequence of the work already done.

Lemma 17: L(A) is infinite

Proof: this is an immediate consequence of the proof (part 1) of lemma 15.**End of proof**

Lemma 18: $L(A)^c$ is infinite

Proof: Suppose $L(A)^c$ is finite. Then $L(A)^c \in \mathsf{P}(A)$ and as $\mathsf{P}(A)$ is closed under complementation, it follows that $L(A) \in \mathsf{P}(A)$. But then by lemma 17 we have that L(A) is infinite, and clearly $L(A) \subseteq L(A)$, contradicting lemma 9.

End of proof

We now have established all the required properties to yield the last theorem of this chapter and state:

Theorem 5: There exists a recursive oracle A such that NP(A) has a set which is both simple and P(A)-immune.

This theorem concludes our survey of separation results between the language classes of the first level in the relativised P-Time Hierarchy. In the following chapter we will study separation results between classes in the first level, and classes of the second level of this hierarchy.

III STRONG SEPARATIONS BETWEEN THE FIRST TWO LEVELS.

In this chapter we will step up one level in the relativised hierarchy and consider separation results between classes NP and Δ_2^P on one hand and the classes Σ_2^P and Π_2^P on the other hand. The following chapter will then deal with separations on the second level, i.e. separations *between* Σ_2^P and Π_2^P. First we will construct an oracle A such that $\Sigma_2^P(A)$ has an NP(A)-immune set.

Homer and Gasarch [51] first suggested that the construction of such an oracle might be possible in the section concerning further research of their paper on exponential time bounded classes, and the relation of these classes with relativised NP. It turns out that there exists an easy to understand straightforward slow diagonalisation for this construction.

First we present the language. For all oracles X consider the language:
$L(X)= \{ 0^s \mid \exists u_{|u|=s} \ \forall v_{|v|=s} \ . \ u \bullet v \in X\}$
Clearly $L(X) \in \Sigma_2^P(X)$ for any oracle X. We will construct an oracle A such that L(A) is NP(A)-immune. As in the previous chapters we first give an informal description of the strategy.

At stage 0 we start with the empty oracle $A_0 = \emptyset$. At each stage s+1, the set A_{s+1} is defined as an extension of A_s. We maintain a border b_s to preserve accepting computations of machines on inputs considered at earlier levels. We construct A in such a way that if $|x| \leq b_{s+1}$ then $x \notin A_{s+1}-A_s$. As before we let $A = \lim_{s\to\infty} A_s$.

During the construction we maintain a set **Req** of indices of requirements. At each stage s+1 the index s is added to **Req**. Requirement R_e is *active* at stage s+1 if e is in $\mathbf{Req}_s$ at stage s+1, and there's a set V such that all strings in V have length $2\times b_s$ and moreover $0^{b_s} \in \varphi_e(A_s \cup V)$. We choose the active requirement with the least index e_0 at stage s and let A_{s+1} be $A_s \cup V(e_0)$. Since

$V(e_0)$ can be chosen to have only polynomially many elements, and *none of the strings of length $2\times b_s$ is added to A before stage s+1,* we can force φ_e to recognise a string in the complement of L(A), and thus satisfying the corresponding requirement R_e.

If no requirement is active we add an exponential number of strings to A at stage s+1 trying to make L(A) infinite.

Formally we describe the construction by:

Requirements:

$(\forall e)\ [\ R_e\colon |\varphi_e(A)| = \infty \ \Rightarrow\ \varphi_e(A) \cap L(A)^c \neq \emptyset]$

Construction

```
    Stage 0: $A_0 = \emptyset$; Req$_0 = \emptyset$;$b_0=0$;
    Stage s+1:
        If $\exists\ e\in$ Req$_s$ such that $\Phi_e(b_s) < 2^{b_s}$, and $\exists\ V \subset \Gamma^*$ such that:
        1) For all v in V: $|v| = 2\times b_s$
        2) $0^{b_s}$ is recognised by $\varphi_e(A_s \cup V)$.
        Then let $e_0$ be the minimal e with this property, and $V_0$ a corresponding set of
            minimal cardinality.
            $A_{s+1} = A_s \cup V_0$ ;
            Req$_{s+1}$ = (Req$_s$ - $\{e_0\}$) $\cup$ $\{s\}$
        Else
            $A_{s+1}= A_s \cup \{\ 0^{b_s}\bullet w \mid |w|=b_s\}$;
            Req$_{s+1}$ = Req$_s$ $\cup$ $\{s\}$
        endif;
        $b_{s+1}=\max\{2\times b_s ,\max\{\Phi_i(b_s) \mid i \leq s\ \}\} +1$;
    End of stage s+1
End of construction.
```

We will now show that A constructed as above has the desired properties. First we show that $L(A)^c$ has a nonempty intersection with each $\varphi_e(A)$ if the corresponding requirement is satisfied at some stage. This lemma is used just to streamline proof.

Lemma 18: If is $e \in \mathbf{Req}_s - \mathbf{Req}_{s+1}$ at stage s+1 then $\varphi_e(A) \cap L(A)^c \neq \emptyset$

proof: If $e \in \mathbf{Req}_s - \mathbf{Req}_{s+1}$ at stage s+1 then $0^{b_s} \in \varphi_e(A_{s+1})$, and since the minimal string in $A_{t+1} - A_t$ has length $> \Phi_e(b_s)$ consequently $0^{b_s} \in \varphi_e(A)$. Moreover by construction we have that for all $t \leq s$ $A_t \cap \Gamma^{b_s} = \emptyset$, so $0^{b_s} \notin L(A_s)$. But then since $V_0 \leq \Phi_e(b_s) < 2^{b_s}$ at stage s+1, we have that $0^{b_s} \notin L(A_{s+1})$ and hence since $b_{s+1} > 2 \times b_s$ for all $t > s$: $0^{b_s} \notin L(A_t)$ and so $0^{b_s} \notin L(A)$.

End of proof

Next we show that the requirements belonging to infinite languages are all satisfied.

Lemma 19: For all e if $\varphi_e(A)$ is infinite then $\varphi_e(A) \cap L(A)^c \neq \emptyset$.

Proof: Suppose to the contrary that $\varphi_e(A)$ is infinite within L(A). By Lemma 18 we must assume that Requirement R_e is never satisfied. Since at each stage the least active requirement is satisfied if there are any requirements active , we must assume that there are only finitely many stages at which R_e is active. Let s+1 >e be such that:

1) $\Phi_e(b_s) < 2^{b_s}$

2) R_e is not active at stage s+1.

3) there is a $t \geq s+1$ such that $0^{b_s} \in \varphi_e(A_t) \cap L(A_t)$

Since R_e is inactive at stage s+1, there is no set of strings $V \subset \Gamma^{2 \times b_s}$, such that $0^{b_s} \in \varphi_e(A_s \cup V)$. However since b_{s+1} is at least $\Phi_e(b_s)+1$, for all t>s the only strings in $A_t - A_s$ which can be queried by φ_e on input 0^{b_s} have length $2 \times b_s$(note $0^{b_s} \in L(A)$). Hence there can be no accepting computation of $\varphi_e(A_t)$ on input 0^{b_s} and we have a contradiction.

End of proof

As before the padding lemma provides for the infinity of L(A)

Lemma 20: L(A) is infinite.

proof: Suppose L(A) is finite. Then there can only be finitely many stages s+1 for which we have that $L(A_{s+1})-L(A_s) \neq \emptyset$. This means that at all but finitely many stages the "then" case in the construction is chosen, a requirement is satisfied and its index removed from **Req**. This means that **Req** remains finite. However there are infinitely many indices of the empty set, each of which is added to **Req** at its own stage, and never removed from s, which means that **Req** grows infinite, and we have a contradiction.

End of proof

This gives us:

Theorem 6: There exists a recursive oracle A such that Σ_2^p (A) has an NP(A) immune set.

Applying Balcázar's trick to the oracle construction of theorem 6, we easily obtain a strong separation between relativised NP, and relativised Π_2^p. Consider the language for all oracles X:

$L(X) = \Gamma^* - \{0\}^* \cup \{0^s \mid \exists u_{|u|=s} \forall v_{|v|=s} \ . \ u \bullet v \in X\}$

Then L(X) is clearly a $\Sigma_2^p(X)$ language and hence $L(X)^c$ is a $\Pi_2^p(X)$ language. Now we adapt the construction by inverting the decisions. Instead of trying to establish a nonempty intersection with $L(A)^c$, we try to establish a nonempty intersection with L(A). A minor complication is formed by the fact that A is extended simultaneously to fix an accepting computation *and* add a new element to L(A), but the restriction $\Phi_e(b_s) < 2^{b_s}$ can now be used to ensure that there is at least one string of length b_s such that no extension of this string can be queried in a chosen accepting computation. With these extensions we can extend A_{s+1}.

Requirements:

$(\forall e)\,[\,R_e\colon |\varphi_e(A)| = \infty \;\Rightarrow\; \varphi_e(A) \cap L(A) \neq \emptyset]$

Construction

```
Stage 0: $A_0 = \{ x \mid |x| \text{ is even} \}$; Req$_0 = \emptyset$; $b_0=0$;
Stage s+1:
        $B_s = A_s - \{x \mid |x|=2b_s\}$
        If $\exists\, e \in$ Req$_s$ such that $\Phi_e(b_s) < 2^{b_s}$, and $\exists\, V \subset \Gamma^*$ such that:
                1) For all v in V: $|v| = 2\times b_s$
                2) $0^{b_s}$ is recognised by $\varphi_e(B_s \cup V)$.
        Then let $e_0$ be the minimal e with this property, $V_0$ a corresponding set of minimal
                cardinality. $\gamma_0$ some accepting computation of $\varphi_e(B_s \cup V)$ on input $0^{b_s}$ and
                x some string of length $b_s$ such that none of $\{x \bullet w \mid |w|=|x|\}$ is queried in $\gamma_0$;
                $A_{s+1} = B_s \cup V_0 \cup \{ x \bullet w \mid |w|=|x| \}$;
                Req$_{s+1}$ = (Req$_s$ - $\{e_0\}$) $\cup \{s\}$
        Else
                $A_{s+1}=B_s$;
                Req$_{s+1}$ = Req$_s \cup \{s\}$
        endif;
        $b_{s+1}=\max\{2\times b_s, \max\{\Phi_i(b_s) \mid i \leq s\}\} +1$;
   End of stage s+1
End of construction.
```

An analogous proof yields:

Theorem 7: There exists a recursive oracle A such that $\Pi_2^P(A)$ has an NP(A) -immune set.

The results above can be strengthened by a so called bi-immune separation. We can construct a single language $L(A) \in \Sigma_2^P$ such that both L(A) and $L(A)^c$ (Which is a Π_2^P language) are immune to NP(A). First bi-immunity is formally defined.

Definition 3: Let C be a class of languages, $C \subseteq 2^{\Gamma^*}$. A language L is *C-bi-immune* (or bi-immune to C) if both L and L^c are C-immune.

In the constructions above we could afford to be rather careless with strings. The complement of L(A) was allowed to grow rather uncontrolled. Therefore L(A) itself could be rather thin (or thick in the second case). When constructing a bi-immune set, we cannot afford such luxury. In particular, the language L(A) cannot be a tally set. (a set which is a subset of $\{0\}^*$) Else the language $\{1\}^*$ would be entirely in its complement and so $L(A)^c$ would not be NP(A)-immune.

We recall that we have been in a similar situation on level 1 when providing simultaneously for simplicity and P(A) immunity. The present problem is just a shifted version of the problem we had there. Therefore we propose the following language. For all oracles X:

$L(X)=\{x \mid \exists u_{|u|=|x|} \ \forall v_{|v|=|x|} \ . \ x{\bullet}u{\bullet}v \in X \}$

Which is just a shifted version of the language used to establish theorem 5. Again it's clear that L(X) is in $\Sigma_2^P(X)$. First we give an informal description of the strategy.

Again we start the construction with an empty oracle A_0, and a moving border b_s initially set to 0. As in the previous construction no string with length less than b_s is added to A at stage s+1. We can however in the present case not limit the addition of strings to A at stage s+1 to strings of length $2{\times}b_s$.

In the previous construction we could, if the answer to the question: "Is there an extension $V \subset \Gamma^{2\times b_s}$, of the present oracle A_s such that the input x now under consideration is recognised by any of the machines now under consideration with oracle $A_s \cup V$?", was negative, freeze this answer by creating a gap in the construction.

In the present case an infinite union of these gaps may 'hide' some infinite NP(A) set, thus making $L(A)^c$ not NP(A) immune. Therefore, if the question above is answered in the negative, we can do nothing else than pose the same question for the string x+1.(Here x+1 stands for the successor of the string x in some chosen but fixed ordering of Γ^*). This however creates another problem since now if at some stage s the answer of machine φ_e to input x was "no", changes may be made to the oracle influencing this answer. It may well be that there is a $V \subset \Gamma^{4 \times b_s}$, such that $x \in \varphi_e(A_s \cup V)$, and expansions of the oracle at stage t>s may cause A to locally look like $A_s \cup V$. Since now $x \in L(A)^c$, infinite repetition of this situation for a single requirement may cause the loss of NP(A) immunity for $L(A)^c$. Therefore we change the question tested at stage s+1 to: "Is there an index $e \leq s$ for which the nonempty intersection of $\varphi_e(A)$ with either L(A) or $L(A)^c$ is not yet fixed, and an extension V of the oracle such that for all $v \in V$ and the present subject string x: $b_s < |v| \leq \Phi_e(|x|)$ and $x \in \varphi_e(A_s \cup V)$?" If not then consider x+1, if so let e be the least index of such a requirement. If the negative requirement is not yet satisfied, we let A_{s+1} be $A_s \cup V$, fixing the intersection with the complement of L(A). If the negative requirement is satisfied, and the positive requirement is not yet satisfied, we extend A_s with a set W such that $x \in L(A_s \cup W) \cap \varphi_e(A_s \cup W)$. To make this possible we have to impose the condition $\Phi_e(|x|) < 2^{|x|}$ on the length of x. For any index e this condition will eventually become true.

After having satisfied a requirement at stage s, we wish to preserve this satisfaction at later stages, and therefore we wish to move b_s beyond **max**$\{ \Phi_e(|x|), 3|x|\}$ to make sure that at later stages neither the accepting computation of φ_e on input x is altered, nor that x is inadvertently added to the language. (Actually 3|x| is only needed in the case of the satisfaction of a negative requirement.) This gap construction may again lead to the loss of NP(A)-immunity of $L(A)^c$. The difference between the gaps constructed here, and the gaps proposed above is that this gap is proposed to preserve the *satisfaction* of a *named* requirement e. Therefore we can again make use of the priority construction *forward search* explained in the proof of theorem 5. Before making the extension V or W and closing the gap, we investigate all strings in the interval between x+1 and $0^{\lambda}+1$, where $\lambda=\mathbf{max}\{\ \Phi_e(|x|), 3|x|)\ \}$ to see if there is some unsatisfied requirement e' < e and a string in this interval such that an extension V' or W' of B_s may be found with which this smaller requirement may be satisfied. If so, we overrule the extension V, and repeat the forward search with e' substituted for e. Only for the last requirement found in this chain of forward searches we make the corresponding extension to the oracle and create a gap.

Several 'finiteness observations' guarantee the correct performance of the diagonalisation.
For given index e let N_e denote the negative requirement $\varphi_e(A) \cap L(A)^c \neq \emptyset$, and P_e denote the positive requirement $\varphi_e(A) \cap L(A) \neq \emptyset$ which have to be satisfied by the diagonalisation in the case that $|\varphi_e(A)| = \infty$.

1) For any e there are only finitely many requirements (positive and negative) with higher priority than $N_e(P_e)$, hence each stage can consist of only finitely many forward searches.

2) For any e there are only finitely many requirements with higher priority than $N_e(P_e)$, hence the assumption that $N_e(P_e)$ is 'the first to be found' at infinitely many stages will result in a contradiction since a forward search can only succeed at finitely many of these stages. (Whence $N_e(P_e)$ is satisfied after finitely many stages.)

3) For any e there are only finitely many requirements with higher priority than $N_e(P_e)$, hence $N_e(P_e)$ can only 'hide' in finitely many gaps without being found in a forward search.

The version of the forward search we use in the following construction is actually less complicated than the one used in the proof of theorem 5, because in the present construction we don't have to keep track of earlier changes to the oracle. In this case however there is a (degenerate form of a) finite injury argument in the construction since the satisfaction of a requirement can be undone. As the least requirement satisfied at a given stage is permanently satisfied this finite injury argument is also of the finite extension type, and hence the method is not a so called 0' diagonalisation method as was the case in the second version of the construction of an oracle for a simple set in relativised NP. Since we can only guarantee the satisfaction of the requirement found in the last forward search, we don't have to keep track of all earlier proposed extensions of the oracle. For any e and x if V is the proposed extension of the oracle, an extension V' exists such that $\varphi_e(A_s \cup V \cup V')$ accepts x only if an extension W exists such that $\varphi_e(A_s \cup W)$ accepts x. Hence we don't have to 'remember' all extensions proposed at earlier substages. We will impose the condition that $\max\{\Phi_i(|x|) \mid i \leq e\} < 2^{|x|}$ for the first e to be found at stage s+1. Then on one hand none of the sets V can consist of all $2^{2|x|}$ possible extensions of one string x, and hence $x \notin L(A_s \cup V)$, on the other hand there is at least one string y such that none of the strings of length $3|x|$ that are extensions of y is queried by the considered NP machine in the accepting computation. This string can be used to achieve $x \in L(A_{s+1})$.

We now come to a formal description of the construction.

Requirements:

$(\forall e)\ [P_e\colon |\varphi_e(A)| = \infty \Rightarrow \varphi_e(A) \cap L(A) \neq \emptyset]$

$(\forall e)\ [N_e\colon |\varphi_e(A)| = \infty \Rightarrow \varphi_e(A) \cap L(A)^c \neq \emptyset]$

Construction

```
Stage 0: $A_0 = \emptyset$; $b_0=0$; $NEG_0=\emptyset$; $POS_0=\emptyset$;
Stage s+1:
    Step 1) i =0;
    Step 2)
        IF $\exists e \in NEG_s \cup POS_s$ s.t. $\exists x$ such that:
            a) $|x| = b_s$
            b) max $\{ \Phi_i(|x|) \mid i \le e\} < 2^{|x|}$
            c) $\exists V \subset \Gamma^*$ s.t.
            c1) $\forall v \in V\, [b_s < |v| \le \Phi_e(|x|)]$
            c2) $x \in \varphi_e(A_s \cup V)$
    Then i:= i+1;
            Let $e_i$ be the least e with this property, $x_i$ the corresponding x,
            $V_i$ corresponding V of minimal cardinality , and (for $\varepsilon = e_i$)
            let $\gamma_i$ be some accepting computation of $\varphi_\varepsilon(A_s \cup V_i)$ on input $x_i$.
            If $e_i \in POS_s$ Then
            $u_i$ = min $\{v \mid |v| = |x_i| \,\&\, \forall w\, [\,|w| = |x_i| \Rightarrow x_i v w$ is *not* queried in $\gamma_i]\}$
                    Let $Y_i = \{ y \mid y = x_i u_i w,\ w \in \Gamma^*,\ |w| = |u_i|\}$
                    $W_i = V_i \cup Y_i$
            Else
                    $W_i = V_i$
            Endif
            *Forward search*
            If $\exists e < e_i$ & $\exists x$ with $x > x_i$ and $|x| <$ max$\{3|x_i|,$max$\{\Phi_j(|x_i|) \mid j \le e_i\}\}$
            such that condition c holds for this pair e,x and
            $e \in NEG_s \cup POS_s$
            Then perform the then case of step 2
            Endif
    Endif
    Step 3)
    If i>0 Then
            $b_{s+1}$ = max $\{3|x_i|,$ max $\{\Phi_j(|x_i|) \mid j \le e_i\}\}$;
            $A_{s+1} = A_s \cup W_i$;
```

```
                If $x \in L(A_{s+1})$
                    Then  $POS_{s+1} = (POS_s - \{e_i\}) \cup \{s\}$;
                          $NEG_{s+1} = NEG_s \cup \{s\}$
                    Else  $NEG_{s+1} = (NEG_s - \{e_i\}) \cup \{s\}$;
                          $POS_{s+1} = POS_s \cup \{s\}$
                Endif
            Else  $A_{s+1} = A_s$;
                  $b_{s+1} = b_s + 1$;
                  $POS_{s+1} = POS_s \cup \{s\}$;
                  $NEG_{s+1} = NEG_s \cup \{s\}$
            Endif
    End of stage s+1
End of construction
```

We will now show that A constructed as above indeed has the desired properties. First we will show that the intersection of L(A) (and of $L(A)^c$) with an infinite NP(A) set is nonempty.

Lemma 21: For all e, if $|\varphi_e(A)|=\infty$ then $\varphi_e(A)\cap L(A) \neq \emptyset$ and $\varphi_e(A)\cap L(A)^c \neq \emptyset$

proof: We show the case of the negative requirement, the other part of the proof is an almost complete analog. Assume for a contradiction that $|\varphi_e(A)|=\infty$ and $\varphi_e(A) \cap L(A)^c=\emptyset$. Without loss of generality we assume there is a stage s in the diagonalisation such that for all indices i<e either $\varphi_i(A) \cap \Gamma^{\geq b_s} = \emptyset$ or $i \notin NEG_t \cup POS_t$ for all $t \geq s$. (Since we have that at each stage s+1: $b_{s+1} \geq b_s+1$ this assumption takes care of all finite sets, and infinite sets of higher priority.) Choose an arbitrary t for which this holds and choose $y \in \varphi_e(A)$ so large that:

1) $\max\{\Phi_i(|y|) \mid i \leq e\} \leq 2^{|y|}$

2) $|y| \geq b_t$.

Then either:

case 1: there is a stage u+1 > t+1 such that $|y|=b_u$ but then the pair (e,y) satisfies the conditions of **step 2** and since no index of higher priority can be satisfied at this stage, requirement R_e will be satisfied and $\varphi_e(A) \cap L(A)^c \neq \emptyset$ (forward search is excluded by priority), in contradiction to our assumption.

case 2: There is a stage u+1 > t+1 such that $b_u < |y| < b_{u+1}$, but then $b_{u+1} \neq b_u+1$, and we conclude that a forward search has taken place at stage u+1. Since $|y| < b_{u+1}$, there must have been a substage of stage u+1 at which strings of length |y| were considered, but at this substage, e was the least index of an active requirement. (Note that since $y \in \varphi_e(A)$ there must be a set of strings V in which all strings have length $\leq \Phi_i(|y|)$ such that A_u can be extended such that $y \in \varphi_e(A_u \cup V)$, and since such sets are apparently considered at stage u+1 requirement R_e is active) and requirement R_e will consequently be satisfied. Again after R_e is satisfied, no forward search can take place

End of proof

The second property of bi-immunity is a corollary of the following lemma.

Lemma 22: Let $L \subset \Gamma^*$ and for all e: $L \cap \varphi_e \neq \emptyset$, then $|L| = \infty$.

Proof : If $\forall x \in L\ [x \leq C]$ then $L \cap \{ x \mid x > C \} = \emptyset$ **End of proof**

Corollary 2: L(A) and $L(A)^c$ are infinite.

Lemma 21 and corollary 2 together yield

Theorem 8: There exists a recursive oracle A such that $\Sigma_2^P(A)$ has an NP(A)-bi-immune set.

Again we do a step up in the hierarchy to study the relationship between relativised Δ_2^P , relativised Σ_2^P and relativised Π_2^P. First we will construct an oracle A such that $\Sigma_2^P(A)$ has a $\Delta_2^P(A)$ immune set. This construction is the shifted version of the Schöning construction of a P(A) immune set in

NP(A). To perform the diagonalisation we need a somewhat special characterisation of relativised Δ_2^p.

Consider the following language.

K(X) = {e#x#w| φ_e recognises input x in less than |w| steps, relative to oracle X}

Obviously for all oracles X, K(X) is NP(X) complete and hence: $\Delta_2^p(X) = P(K(X))$
The technique diagonalising over machines belonging to the class Δ_2^p must be essentially different from the techniques used in the previous section. Before we could decide on the basis of a polynomial amount of strings. The presence of a string x in the oracle K(X) depends on the existence of an accepting computation of a non-deterministic Turing Machine. Since there may be exponentially many accepting computations possible, this presence may depend on an exponential number of strings in the oracle X. Yet there is room to diagonalise.

The intuition here is that if the oracle K(X) gives the answer *no* to a string x, then this answer cannot be prevented from turning into a yes answer without restraining exponentially many strings from the oracle X, but if the answer is yes, then the corresponding computation of the NP machine encoded in x is not longer than |x| steps and can therefore depend on at most |x| queries to the oracle X. Hence if a yes answer of oracle K(X) to a string x is at all *possible*, it can be preserved by adding to X (and restraining from X at later stages) a *polynomial* amount of strings.

The diagonalisation strategy now tries for requirements and strings satisfying a certain length condition to *maximise* the yes answers to queries asked of the oracle K(X). The amount of strings restrained from or added to the oracle X is now polynomial, and hence after freezing the computation on input x of all eligible P(K(X)) machines, a decision of whether or not to add x to L(X) at the present stage can still be met. With this strategy we obtain for a certain input x an oracle

X_{s+1} such that for all machines with index $\leq$ e the answers to queries made on input x are *stable* under *all* extensions of X_{s+1}.

This technique of obtaining stable answers of a K(X) oracle appears in the proof of the following "There's room to diagonalise" Lemma, and was developed as a consequence of a private (electronic) communication with P. van Emde Boas in the spring of 1985. The technique is used by the diagonalisation method to follow. We introduce at this point the notation $\psi_e(A;x)$ to denote the outcome of the computation (i.e. accept or reject) of machine ψ_e on input x relative to oracle A.

Lemma 23: Let A and B be disjoint sets of strings, e an index and x a string then there exist sets V and W such that $(A \cup V) \cap (B \cup W) = \emptyset$ and:

I) for all $X \subset \Gamma^*$: $[X \supset (A \cup V)\ \&\ X \cap (B \cup W) = \emptyset] \Rightarrow \psi_e(K(X);x) = \psi_e(K(A_s \cup V);x)$

II) $|V \cup W| \leq (\Psi_e(|x|))^2$

III) **max** $\{ |y| \mid y \in V \cup W \} \leq \Psi_e(|x|)$

proof: I) Start V_0 and W_0 initially empty. The construction of V and W consists of k stages where we will see that $k \leq \Psi_e(|x|)$. At stage $1 \leq s \leq k$ run ψ_e on input x with oracle $K(A_s \cup V_{s-1})$ for exactly s steps. If ψ_e queries string y in step s then if a set V_y can be found such that $(V_y \cap (W_{s-1} \cup B)) = \emptyset$ and $y \in K(A \cup V_{s-1} \cup V_y)$ then by definition of K(X) there is some index i, and strings x and w , such that y=i#x#w, and φ_i recognises x using the oracle set $A \cup V_{s-1} \cup V_y$ within |w| steps. Choose for an arbitrary triple an accepting computation and let V_i be the set of strings queried in this computation (Note that $|V_i| \leq |y|$). Set $V_s = (V_i \cap (A \cup V_{s-1} \cup V_y)) \cup V_{s-1}$; $W_s = (V_i \cap (A \cup V_{s-1} \cup V_y)^c) \cup W_{s-1}$. (Note that $|V_s - V_{s-1}| + |W_s - W_{s-1}| \leq |y|$) If such a set cannot be found then trivially for all oracles X s.t. $X \supset A_s \cup V_{s-1}$ and $X \cap (W_{s-1} \cup B) = \emptyset$, we have that $y \notin K(X)$ so we let $V_s = V_{s-1}$ and $W_s = W_{s-1}$.

II) The cardinality of $V \cup W$ will be maximal if at each stage $\leq k$ a set V_y can be found.

As we have already seen in this case the sum of the expansions of the sets V_s and W_s is less than or equal to $|y|$ strings. On input x, machine ψ_e can make no more than $\Psi_e(|x|)$ queries to the simulated oracle K(X), each of which has length less than or equal to $\Psi_e(|x|)$. So at each stage $|V_s-V_{s-1}| + |W_s-W_{s-1}| \leq |y| \leq \Psi_i(|x|)$ and there are no more than $\Psi_e(|x|)$ stages. This means that $| [\cup_{s\leq k} V_y] \cup [\cup_{s\leq k} W_y] | \leq (\Psi_e(|x|))^2$.

III) This is immediate from the fact that the set V_i of part I can contain no strings longer than $|y|$, which in turn is a string queried by ψ_e on input x, and hence cannot be longer than $\Psi_e(|x|)$

End of proof

Lemma 23 has an easy corollary that allows us to diagonalise.

Corollary 3: Let A and B be disjoint sets, e an index, and x a string. Then there exist disjoint finite sets V and W s. t: $(A\cup V) \cap (B\cup W) = \emptyset$ and,

I) $\forall X \subset \Gamma^* \, \forall (i \leq e) \, [X \supset (A\cup V) \; \& \; X \cap (B\cup W) = \emptyset] \Rightarrow \psi_i(K(X);x) = \psi_i(K(A\cup V);x)$

II) $|V\cup W| \leq \Sigma_{i\leq e}(\Psi_i(|x|))^2$

III) $\mathbf{max} \{ |y| \mid y \in V \cup W \} \leq \mathbf{max} \{\Psi_i(|x|) \mid i \leq e \}$

proof: Successively apply Lemma 23 for all indices $i\leq e$ and find sets V_i and W_i for the disjoint sets $A \cup (\cup_{j<i} V_j)$, and $B \cup (\cup_{j<i} W_j)$. Finally $V=\cup_{i\leq e} V_i$ and $W=\cup_{i\leq e} W_i$

End of proof

Now we are ready to present the diagonalisation. For a language we use L(X). For all oracles X:

$L(X)= \{ 0^s \mid \exists u_{|u|=s} \text{ s.t. } \forall v_{|v|=s} \; . \; u\bullet v \in X\}$

The construction of the oracle A such that L(A) is Δ_2^p-immune is now more or less analogous to the construction in the proof of theorem 6.

Requirements

$(\forall e)\,[\,|\,\psi_e(K(A))\,| = \infty \Rightarrow \psi_e(K(A)) \cap L(A)^c \neq \emptyset\,]$

Construction

```
    Stage 0: $A_0=\emptyset$; $B_0=\emptyset$; $b_0=0$; $Req_0=\emptyset$;
    Stage s+1:
        Find the largest index e in $Req_s$ such that $\sum_{i\le e}(\Psi_i(b_s))^2 < 2^{b_s}$;
        $b_{s+1} = \max\{\max\{\Psi_i(b_s) \mid i \le e\}, 2\times b_s\} + 1$;
        By corollary 3 construct sets V and W such that :
            1) $|V\cup W| \le \sum_{i\le e}(\Psi_i(|x|))^2$
            2) $(A_s\cup V) \cap (B_s\cup W) = \emptyset$
            3) $\forall X \subset \Gamma^*[X\supset(A_s\cup V) \;\&\; X\cap(B_s\cup W) = \emptyset] \Rightarrow$
                    $\forall i\le e.\ \psi_i(K(X);0^{b_s}) = \psi_i(K(A_s\cup V);0^{b_s})$
            4) $\max\{|y| \mid y \in V\cup W\} < b_{s+1}$
        If $\exists i\le e$ such that $0^{b_s}$ is recognised by $\psi_i(K(A_s \cup V))$
            then Let $i_0$ be minimal with this property.
                    $A_{s+1} = A_s \cup V$;
                    $Req_{s+1} = (Req_s - \{i_0\}) \cup \{s\}$
            else Let v be a string such that $|v| = b_s$ and $\{v\bullet w \mid |v| = |w|\} \cap W = \emptyset$;
                    $A_{s+1} = A_s \cup V \cup \{v\bullet w \mid |v| = |w|\}$;
                    $Req_{s+1} = Req_s \cup \{s\}$
        endif;
        $B_{s+1} = \{y \mid y \in \Gamma^* \;\&\; |y| \le b_{s+1}-1\} - A_{s+1}$;
    End of stage s+1
End of Construction.
```

The correctness proof for this construction is -given the Stable Query Method- really a mechanical effort. Eventually the condition $\sum_{i\le e}(\Psi_i(b_s))^2 < 2^{b_s}$ for becoming active will become true for any index e at some stage. From that stage on the computation of the corresponding machine on the candidate input for the language is frozen, so if the string 0^{b_s} is not recognised *then and there*, it never will. Note that at stage s+2 the set B_{s+1} contains the union of all sets W determined at previous stages. So no string can be added to A at this stage which can alter computations considered at these earlier stages. Moreover at stage s+1, b_{s+1} is set to a value of greater than the maximum of the maximum of strings in $V \cup W$ and two times b_s this means that b_{s+1} is *greater than* the length of any string in A_{s+1} (by induction). Whence at stage s+2 the oracle A_{s+1} will contain *no* strings of length b_{s+1}, and since B_{s+1} contains only strings of length less than or equal to b_{s+1} - 1, there must remain at least one string x of length b_{s+1} at stage s+2 after computation of the set W at this stage, such that $\{ x \bullet w \mid |w| = |x| \} \cap W = \emptyset$. To see that requirements for which machines recognise 0^{b_s} for an infinite number of s, are satisfied, just use a proof of a slow diagonalisation.

The other type of indices are actually more interesting. For the first time in this thesis we see that the oracle set is changed at an infinite number of stages *without the intention to satisfy a requirement*. There are requirements for which stabilisation of the computation on input 0^{b_s} *always yields the answer no*. Hence such a requirement is not *actually* satisfied at *any stage s*. Yet the stabilisation of the computation serves to achieve that the intersection of the language corresponding to this requirement and $\{0^{b_s} \mid s \in \omega \}$ remains finite throughout the construction. Therefore this requirement is *satisfied at infinity*. The method maintains an ever growing set of requirements for which it takes measures at each following stage to satisfy them at infinity, sometimes one ore more of the requirements "drop out" because they can be satisfied at a finite stage. However it is undecidable if a requirement is of the second type, so all requirements have to be considered until they byte.

The properties described here are characteristic for a so called *forcing method* and the above diagonalisation is the first example in this thesis of such a method. One could argue that these characteristics have also appeared in the immunity construction, and especially in the combined simplicity and immunity construction for the sake of keeping languages recognised by deterministic automata finite, but there the measures taken by the diagonalisation method to achieve this were limited to changing the restraint set. In the above case we see for the first time that the oracle set itself is changed at an infinite number of stages to satisfy a single requirement.

It just remains to show that L(A) is infinite. However we again observe (since the diagonalisation is concerned with the enumerator for the deterministic machines) that there will be infinitely many indices of the empty set (which machines just won't use the oracle K(A)), and find that L(A) grows infinite by the same arguments as used in lemma 20.

These observations together yield:

Theorem 9: There exists a recursive oracle A such that $\Sigma_2^P(A)$ has a $\Delta_2^P(A)$-immune set.

Finally applying Balcázar's trick to this diagonalisation and using the stable query method, we can also obtain a $\Delta_2^P(A)$-immune set in $\Pi_2^P(A)$.

Again we use the language for all oracles X:

$L(X) = \Gamma^* - \{0\}^* \cup \{ 0^s \mid \exists u_{|u|=s} \forall v_{|v|=s}\ u \bullet v \in X\}$

And analogous to the transformation of the construction of theorem 6 to the construction of theorem 7 we have:

Requirements:

$(\forall e)\ [\ |\ \psi_e(K(A))\ | = \infty \Rightarrow \psi_e(K(A)) \cap L(A) \neq \emptyset\ ]$

Construction

```
    Stage 0: $A_0=\{x \mid |x| \text{ is even}\}$ ; $B_0=\emptyset$; $b_0=0$;$Req_0=\emptyset$;
    Stage s+1:
        Find the largest index e in $Req_s$ such that $\sum_{i\leq e}(\Psi_i(b_s))^2 < 2^{b_s}$;
        Let $b_{s+1} = \max\{\max\{\Psi_i(b_s) \mid i \leq e\}, 2 \times b_s\} +1$;
        $C_{s+1} = (A_s - \{x \mid |x| = 2b_s\}) \cap \{y \mid |y| < b_{s+1}\}$;
        Find sets V and W such that $\forall X \subset \Gamma^*[X \supset (C_{s+1} \cup V) \ \& \ X \cap (B_s \cup W) = \emptyset] \Rightarrow$
        $\forall i \leq e. \ \psi_i(K(X);0^{b_s}) = \psi_i(K(C_{s+1} \cup V);0^{b_s})$
        If $\exists i \leq e$ such that $0^{b_s}$ is recognised by $\psi_i(K(A_s \cup V))$
        then Let i be minimal with this property, and let $v_i$ be a string of length $b_s$
                such that $\{v_i w \mid |w|=b_s\} \cap W = \emptyset$;
                $A_{s+1} = C_{s+1} \cup V \cup \{v_i w \mid |w|=b_s\} \cup \{y \mid |y| \text{ is even } \& \ y \geq b_{s+1}\}$;
                $Req_{s+1} = (Req_s - \{i\}) \cup \{s\}$
        else    $A_{s+1} = C_{s+1} \cup V \cup \{y \mid |y| \text{ is even } \& \ y \geq b_{s+1}\}$;
                $Req_{s+1} = Req_s \cup \{s\}$
        endif;
        $B_{s+1} = \{y \mid y \in \Gamma^* \ \& \ |y| \leq b_{s+1}-1\} - A_{s+1}$;
    End of stage s+1
End of Construction.
```

The correctness proof of this construction is actually the same as above, although C_{s+1} partly plays the role of A_{s+1}.(But note that $A_{s+1} \supseteq C_{s+1}$, $B_{s+1} \supseteq W$ by corollary 3 (III), and that $B_{s+1} \cap A_{s+1} = \emptyset$ by brute force) Instead of deriving that C_{s+1} is empty at length $2b_s$ at stage s+1 we force C_{s+1} to be empty at this length. However by the same reasoning as above we can show that there are no strings of this length that are important for computations preserved at earlier stages. Again B_{s+1} has only strings of length $\leq b_{s+1}$ -1 as above. We have:

Theorem 10: There exists a recursive oracle A such that $\Pi_2^P(A)$ has a $\Delta_2^P(A)$-immune set.

The next logical step would of course be to shift the strategy used to obtain theorem 11, and construct an oracle A such that $\Sigma_2^P(A)$ has a $\Delta_2^P(A)$-bi-immune set. This is however not straightforward. In the construction of theorem 8 a polynomially bounded extension to the oracle was made each time the possibility of satisfying a requirement occurred. Hence the maximal extension of the oracle could be kept polynomially bounded at each stage, though all strings of a given length were considered, simply by taking a bounded number of requirements into consideration. The stable query method however requires the stabilisation of computations on input strings regardless whether a requirement is satisfied or not. Stabilising computations for all strings of a given length would mean a possibly exponential extension of the oracle at a given stage, and hence the loss of control to add a string to the subject language or not. We infer that if it is at all possible to construct an oracle A such that $\Sigma_2^P(A)$ has a $\Delta_2^P(A)$-bi-immune set, a new observation is required to which we presently don't have access.

IV THE SECOND LEVEL

The present chapter is entirely devoted to a single theorem. We will show the existence of an oracle A such that $\Sigma_2^P(A)$ has a simple set. As before in the case of the construction of an oracle A such that $\Sigma_2^P(A)$ has a $\Delta_2^P(A)$-bi-immune set, the stable query method is not powerful enough to accomplish this result. Though we can apply Balcázar's trick and may consider only one string at each stage of the diagonalisation, stabilising computations by using a polynomially bounded set of strings fails. In the Δ_2^P case we had to deal with a polynomially bounded number of queries, and could force the existence of only one computation on a given input by a given machine. As $\Sigma_2^P(X)$ is equivalent to NP(K(X)), for all oracles X, we have to live with the fact that there are exponentially many different possible computations on a given input, regardless of the local structure of K(X). Any single computation in this set can be stabilised to be accepting or rejecting by using the stable query method developed in chapter III to stabilise computations. Unfortunately this method does not especially favour accepting computations. (Stabilisation of only one accepting computation would mean acceptance throughout the construction, and this would require a polynomial amount of strings) As we cannot choose between fixing acceptation or fixing rejection, fixation of the outcome of a computation may make it necessary to freeze exponentially many rejecting computations querying exponentially many different strings in K(X), which may imply a restraint set of exponential size on X_S as stage s, and thus make $x \in L(X)$ or $x \notin L(X)$ independent of the rest of the algorithm.

It seems that we have met here with Heller's thesis[49] that "Two polynomial quantifiers are as powerful as one exponential quantifier.", and hence diagonalisation over machines with recognising capacity of Σ_2^P level is not possible with standard techniques.

An entirely different argument has to be used to make diagonalisation possible. Instead of just counting strings in Γ^*, we will count in a counting lemma to follow *subsets* of strings of Γ^*. The idea of this subset counting is due to Baker and Selman who have in '76 successfully used a special form of the counting lemma for the separation of relativised Σ_2^r and Π_2^r. We will look at the number of *instantiation sets* which make Σ_2^r formulas and Π_2^r true. Recall that relativised Σ_2^r and relativised Π_2^r can be characterised as the classes of languages corresponding to a polynomial time computable 3-ary relation to which two alternating polynomially bounded quantifiers are applied.

Let R(x,y,z) denote the relation, and p(|x|) denote the polynomial bound on the length of strings to which the quantifiers refer. Baker and Selman [11] illustrate the difference in the number of different sets of strings which can determine the value of R(x,y,z) with the following picture:

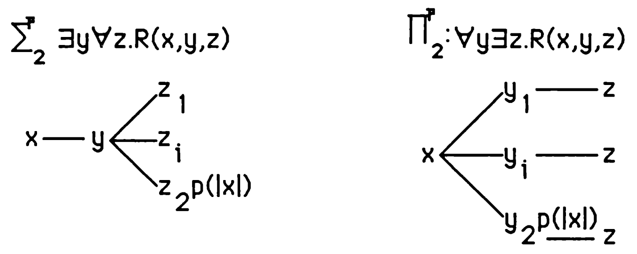


Fig 5: Sizes of the instantiation sets

This picture intends to illustrate the intuitive notion of the number of different subsets of an oracle set A which may determine the value of $R(x_0,y,z)$ for fixed x_0 when R is computed relative to A and quantified by ∃∀ and ∀∃ respectively. Experience learns that this picture requires some explaining:

Take x_0 fixed, and picture the relation $R(x_0,y,z)$ as a subset of $\Gamma^* \times \Gamma^*$. As we are dealing with bounded quantification we are only concerned with pairs of the relation for which both strings have length less than or equal to p(|x|). For notation let $N = 2^{p(|x|)+1} - 1$ be the number of different strings with this property. This leaves a subset of N^2 pairs. Suppose we have a mechanism (oracle construction) by which we can *set* to an arbitrary value the truth value of $R(x_0,y,z)$ at each particular point y,z of this square grid. Setting the value of $R(x_0,y_0,z_0)$ to false means that the expression $\forall z(x_0,y,z)$ gets the value false for all points y where $y=y_0$.

This means that there are N different *minimal* subsets S of size N of this grid for which setting $R(x_0,y,z)$ to true for all pairs $(y,z) \in S$ has the effect that $\exists y \forall z\ R(x_0,y,z)$ becomes true. (Here the word minimal is used to emphasise that any set S' such that $R(x_0,y,z)$ is set to true for all pairs $(y,z) \in S'$ which has the effect that $\exists y \forall z\ R(x_0,y,z)$ becomes true *must* contain one of these sets S as a subset.) On the other hand for all N different y, we can choose an arbitrary z such that setting $R(x_0,y,z)$ to true at point y,z means that $\forall y \exists z\ R(x_0,y,z)$ becomes true. This gives that there are N^N different *minimal* subsets of the grid for which setting $R(x_0,y,z)$ to true for all pairs $(y,z) \in S$ has the effect that $\forall y \exists z\ R(x_0,y,z)$ becomes true. This situation is perhaps best illustrated with a picture.

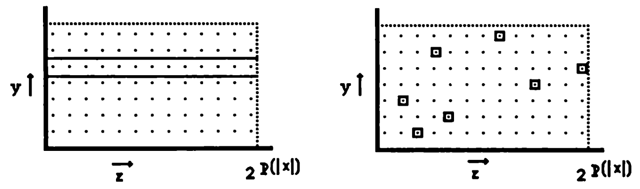


Fig 6: Sizes of the instantiation sets

The left hand grid in the above picture illustrates a set which makes $\exists y \forall z\ R(x_0,y,z)$ true, and the right hand side grid illustrates one of the sets for which $\forall y \exists z\ R(x_0,y,z)$ becomes true.

The gap between N and N^N is exponential, and therefore Baker and Selman observed that an oracle dependent Π_2^p-expression could be found having for each x at least $2^{|x|2^{|x|}}$ different sets S_x such that the expression is true at point x for all oracles A where one of the S_x is a subset of A, and since for any bounded Σ_2^p-expression with bounding polynomial p the number of different sets on which the expression can be made true is limited by $2^{p(|x|)}$, there must be a point at which the Π_2^p-expression can be made true without making the Σ_2^p-expression true or vice versa.

The extension needed for the construction of a simple set is that this reasoning goes through for a *growing* number of Σ_2^p-expressions. If the Π_2^p-expression expression can be made true at $2^{|x|2^{|x|}}$ different instantiation sets, and a number of Σ_2^p-expressions say $\alpha_1,\ldots,\alpha_i$ have *different values* at at most $2^{p_i(|x|)}$ different sets, then whenever $2^{|x|2^{|x|}} > \Sigma_{j\leq i}2^{p_j(|x|)}$ either there is a set such that the Π_2^p-expression is false and one of the Σ_2^p-expressions is true, or there is a set at which the Π_2^p-expression is true, and none of the Σ_2^p-expressions is true. As a Π_2^p-expression expression is the negation of a Σ_2^p-expression, this observation clears the path for diagonalisation.

At a later point in this chapter we will come to a formal proof of the difference in the number of instantiation sets explained above. For now let us assume that we have the 'room to diagonalise' explained in a-c below:

a) For all oracles X let $L(X) = \{0,1\}^*-\{0\}^* \cup \{0^s \mid \exists y_{|y|=s}\forall z_{|z|=s}\, yz\in X\}$. Clearly $L(X) \in \Sigma_2^p(X)$ for any oracle X.

b)We argue that the set of all relativised Σ_2^p-expressions can be represented by a recursively enumerable class of machines, i.e. we may assume an enumeration of all relativised Σ_2^p-expressions giving for each index $e \in \omega$ a polynomial time computable 3-ary relation $R_e(x,y,z)$, a deterministic oracle machine ψ_e which computes the relation $R_e(x,y,z)$, with time bound $\Psi_e(|x|)$ and a polynomial q_e such that $q_e(|x|)$ is a bound on the length of the strings y and z for which $R_e(x,y,z)$ is computed to determine the value of $\exists y\forall z R_e(x,y,z)$. Such an enumerator is easily constructed in the same fashion as in chapter 1. We denote this enumeration by $\{\sigma_i\}_{i\in\omega}$. We denote by σ_i the set of all $x \in \Gamma^*$ such that $\exists y_{|y|\leq q_i(x)}\forall z_{|z|\leq q_i(x)}R_i(x,y,z)$. Likewise for oracle set A we denote by $\sigma_i(A)$ the set of all $x \in \Gamma^*$ such that $\exists y_{|y| \le q_i(x)} \forall z_{|z| \le q_i(x)} R_i(A;x,y,z)$. (We will use $R_i(A;x,y,z)$ for $R(x,y,z)$ computed relative to A)

c) We will prove in lemma 24 that for any finite oracle set A, any index e and any string x with:

1) $A \cap \Gamma^{2|x|} = \emptyset$

2) $\sum_{i \le e} \psi_i(|x|) < 2^{|x|/2}$

3) $\max\{q_i(|x|) \mid i \le e\} < 2^{|x|/2}$

We can find a finite set $E \subsetneq \Gamma^{2|x|}$ such that for oracle $A \cup E$ either:

case a) $x \in L(A \cup E)$ *and* $\exists i \le e.\ x \in \sigma_i(A \cup E)$

or case b) $x \notin L(A \cup E)$ *and* $\forall i \le e.\ x \notin \sigma_i(A \cup E)$

We use this observation in a simple forcing method constructing an oracle A such that L(A) is simple. L(A) will be a simple set if for all indices e the following requirement is fulfilled:

$R_e : |\sigma_e(A)| = \infty \Rightarrow \sigma_e(A) \cap L(A) \neq \emptyset$

We start the construction with $A_0 = \bigcup_{s \in \omega}\{0^s \bullet v \mid |v|=s\}$, so that at stage 0 we have that $L(A_0) = \Gamma^*$. We maintain a set $\mathbf{Req}_s$ of active requirements. Requirement with index s is added to $\mathbf{Req}_s$ at stage s. The inductive step is as follows:

At stage s+1 the diagonalisation first computes a length m_{s+1} which is so large that changes made to the oracle at length m_{s+1} cannot affect the value of expressions represented by indices smaller than s+1 for strings of length $\le m_s$. Next it makes the oracle empty at length m_{s+1}. For this purpose we maintain a set B, which at stage s+1 represents A_s minus all strings of length m_{s+1}. Now according to lemma 24 for the largest index e satisfying $\sum_{i \le e} \psi_i(|x|) < 2^{|x|/2}$ we can either find an oracle $E \subsetneq \Gamma^{2m_{s+1}}$ and an index $i \le e$ such that R_e can be satisfied, in which case we extend the oracle with this E (By the choice of m_{s+2} requirement R_e will remain satisfied.), or we can find an oracle $E' \subsetneq \Gamma^{2m_{s+1}}$ s.t. for *all* indices $i \le e$ the expression corresponding to this index is not true at $0^{2 \times m_s}$ *and* $0^{2 \times m_s} \notin L(B_{s+1} \cup E')$. In this case A_{s+1} will be equal to B_{s+1} extended with E'.

Formally we describe the construction by:

```
Construction
        Stage 0: $A_0 = \bigcup_{u \in \{0\}^*} \{ uv \mid |u|=|v|\}$ ; $m_0=0$;$Req_0=\emptyset$;
        Stage s+1:
                $\lambda = \max \{ q_i(m_s), \Psi_i(m_s) \mid i \leq s\} +1$;
                $m_{s+1} = \lambda$; $B_{s+1} = A_s - \{0^\lambda v \mid |v|=\lambda \}$;
                If $\exists i \in Req_s$ such that there exists a set $E \subsetneq \Gamma^{2\lambda}$ such that
                        $0^\lambda \in \sigma_i(B_{s+1} \cup E)$ and $0^\lambda \in L(B_{s+1} \cup E)$
                Then
                        Let j be the minimal i with this property, and $E_j$ the corresponding set.
                        $A_{s+1} = B_{s+1} \cup E_j$;
                        $Req_{s+1} = Req_s - \{j\} \cup \{s\}$;
                Else
                        Let $e_s = \max\{e \in Req_s \mid \sum_{i \leq e} \Psi_i(\lambda) < 2^{\lambda/2}$ & $\max \{q_i(\lambda) \mid i \leq e\} < 2^{\lambda/2}\}$
                        and let $E' \subsetneq \Gamma^{2\lambda}$ be such that $0^\lambda \notin L(B_{s+1} \cup E')$ and for all $i \leq e_s$ in $Req_s$
                        {cf lemma 24 to see that E' can in fact be chosen}
                        $0^\lambda \notin \sigma_i(B_{s+1} \cup E)$;
                        $A_{s+1} = B_{s+1} \cup E'$;
                        $Req_{s+1} = Req_s \cup \{s\}$
                Endif;
        End of stage s+1
End of construction
```

Before we show the correctness of this construction we will first show that our counting argument is indeed valid and state the announced counting lemma.

To keep the proof of this lemma readable we will agree upon some more notation:

$\forall^{i,x}y$ meaning "for all y such that $|y| \le q_i(|x|)$" , and

$\exists^{i,x}y$ meaning "there exists an y with $|y| \le q_i(|x|)$ such that..."

We will now state our "there's room to diagonalise lemma":

Lemma 24 : For all oracles X let $L(X)= \{0^s \mid \exists y_{|y|=s} \forall z_{|z|=s}\ yz \in X\}$

Suppose:

(1) I is a finite set of indices and $x \in \{0\}^*$ such that:

(a) $\sum_{i \in I} \Psi_i(|x|) < 2^{|x|/2}$

(b) $\max\{q_i(|x|) \mid i \in I\} < 2^{|x|/2}$

(2) F is a fixed oracle such that $F \cap \Gamma^{2|x|} = \emptyset$

(3) $\forall E \subseteq \Gamma^{2|x|}$ [$x \in L(F \cup E) \Rightarrow \forall i \in I\ x \notin \sigma_i(F \cup E)$

Then $\exists E \subseteq \Gamma^{2|x|}$ such that [$x \notin L(F \cup E)$ & $x \notin \sigma_i(F \cup E)$]

Proof: The proof follows the same lines as the proof of lemma 2.4 in [11] We assume the contrary:

Then for *all* oracles $E \subseteq \Gamma^{2|x|}$ (3) holds and there's an $i(E) \in I$ such that

(4) $x \notin L(F \cup E) \Rightarrow x \notin \sigma_{i(E)}(F \cup E)$

For $u \in \Gamma^{|x|}$ define $\tilde{u}=\{ uv \mid |u|=|v| \}$, and let $D'= \cup\, \tilde{u}$.

Clearly $x \in L(D')$ and $D' \subseteq \Gamma^{2|x|}$. There are $2^{|x|}$ different $\tilde{u}$ and each $\tilde{u}$ has cardinality $2^{|x|}$.

Let S_j, $j=1,\ldots,(2^{|x|})^{2^{|x|}}$ be the collection of all "sample sets" which consist of exactly one point from each $\tilde{u}$. Let $D = F \cup D'$.

If $Q \subseteq D'$ and $|Q| < 2^{|x|}$ then D-Q must include some $\tilde{u}$. Thus $x \in L(D-Q)$. Hence by assumption (3) we have:

(5) $\forall i \in I\ x \notin \sigma_i(D-Q)$

For each Sample set S_j, $x \notin L(D-S_j)$ and hence by (4) we have that $\forall j\, \exists i\ x \notin \sigma_i(D-S_j)$

For $i \in I$ and $|y| \le \max\{q_i(|x|) \mid i \in I\}$ define $V(i,y) = \{ S_j \mid \forall^{i,x}z\ R_i(D-S_j\ ;x,y,z) \}$

Since by assumption each S_j is in at least one $V(i,y)$, we have that

$\#[\bigcup_{i\in I} \bigcup_{|y|\leq \mathbf{max}\{q_i(|x|) \mid i\in I\}} V(i,y)] \geq 2^{|x|2^{|x|}}$

By (1)(b) the number of distinct y's satisfying $|y|<\mathbf{max}\{q_i(|x|) \mid i\in I\}$ is less than $2^{2^{\frac{1}{2}|x|}}$

so there must be one y_0 for which $| \bigcup_{i\in I}V(i,y_0)| > 2^{|x|2^{|x|}-2^{\frac{1}{2}|x|}}$

Let $K_0 = \cup_{i\in I}V(i,y_0)$ and $Q_0= \emptyset$

We summarise the current situation as follows:

(i) The cardinality of $K_0 > 2^{|x|2^{|x|}-2^{\frac{1}{2}|x|}}$

(ii) The cardinality of $Q_0=0$

(iii) $Q_0 \subsetneq \cap\{S: S\in K_0\}$

(iv) $S\in K_0 \Rightarrow \exists i\in I\ \forall^{i,x}z\ R_i(D\text{-}S;\ x,y_0,z)$

For $\mu=1,..,2^{|x|}$, we will next define sets K_μ and Q_μ such that:

(i) The cardinality of $K_\mu > 2^{|x|(2^{|x|}-\mu/2-2^{\frac{1}{2}|x|})}$

(ii)The cardinality of $Q_\mu = \mu$

(iii) $Q_\mu \subsetneq \cap\{S:S\in K_\mu\}$ and

(iv) $S\in K_\mu \Rightarrow \exists i\in I\forall^{i,x}z\ R_i(D\text{-}S;\ x,y_0,z)$.

We have thus far shown that conditions (i)-(iv) are true for $\mu=0$.

Assume as induction hypothesis that (i)-(iv) are true for μ, $0\leq\mu<2^{|x|}$. Since the cardinality of Q_μ is μ, and $\mu<2^{|x|}$ we have that (5) holds for y_0 and Q_μ, or:

$\forall i\in I\ \exists^{i,x}z\ \neg R_i(D\text{-}Q_\mu\ ;\ x,y_0,z)$

For each i choose z(i) such that $|z(i)| \leq q_i(|x|)$ and $\neg R_i(D\text{-}Q_\mu\ ;\ x,y_0,z(i))$. By (iv) we know that for each S in K_μ $\exists i\in I$ such that $\forall^{i,x}z\ R_i(D\text{-}S;x,y_0,z)$. Hence for *each* S in K_μ there's a pair (i,w) with $i\in I$ and $w\in S\text{-}Q_\mu$ such that oracle machine ψ_i with oracle $D\text{-}Q_\mu$ queries w on input $<x,y_0,z(i)>$. Otherwise $R_i(D\text{-}S;x,y_0,z(i))$ would also be false. Since each ψ_i

queries at most $\Psi_i(|x|)$ many strings on input $<x,y_0,z(i)>$, there can be no more than $\sum_{i\in I} \Psi_i(|x|)$ *different* points w, so there's some point w such that w is queried by some ψ_i on input $<x,y_0,z(i)>$ and such that w belongs to at least $(|K_\mu|)/\sum_{i\in I} \Psi_i(|x|)$ *distinct* sets S.

Let $K_{\mu+1}$ be this set of S's, then we have that

$$\begin{aligned}|K_{\mu+1}| = (|K_\mu|)/\textstyle\sum_{i\in I} \Psi_i(|x|) &> 2^{|x|(2^{|x|}-\mu/2-2^{\frac{1}{2}|x|})} / 2^{\frac{1}{2}|x|} \\ &= 2^{|x|(2^{|x|}-\mu/2-1/2-2^{\frac{1}{2}|x|})} \\ &= 2^{|x|(2^{|x|}-(\mu+1)/2-2^{\frac{1}{2}|x|})}\end{aligned}$$

Thereby proving that (i) is true at $\mu+1$. (iv) is clearly true since $K_{\mu+1}$ is a subset of K_μ. Let $Q_{\mu+1} = Q_\mu \cup \{w\}$. It follows that (ii) and (iii) are also true.

The induction argument is now complete and we are ready to observe a contradiction, namely for $\mu=2^{|x|}$.

By (iii) since each S in K_μ is a sample set of size $2^{|x|}$ we conclude that K_μ consists of exactly one set S. On the other hand by (i) the cardinality of $K_\mu > 2^{|x|(2^{|x|}-(2^{|x|}/2)-2^{\frac{1}{2}|x|})} > 1$

End of proof

Corollary 4: [Baker & Selman] There exists a recursive oracle A such that $\Sigma_2^p(A) \neq \Pi_2^p(A)$.

proof: Consider the case $|I| = 1$ in lemma 24 and use direct diagonalisation. **End of proof**

We will now show that the construction described above indeed makes the language L(A) (where $A = \lim_{s\to\infty} A_s$) simple in $\Sigma_2^p(A)$. Therefore we have to show that for each index e requirement R_e is eventually satisfied, and that L(A) has an infinite complement. We start with:

Lemma 25: $\forall\, e\in\omega[\ |\{x \in \Gamma^* \mid x\in \sigma_e(A)\}| = \infty \ \Rightarrow \exists x\in \Gamma^*: x\in \sigma_e(A) \cap L(A)]$

Proof: Suppose to the contrary that there is an index e such that the $\Sigma_2^p(A)$ expression is true at infinitely many points, and all these points are in $L(A)^c$. Without loss of generality we may assume that e is the least index with this property, hence we may assume that all indices $i < e$ representing an infinite $\Sigma_2^P(A)$ set are removed from $\mathbf{Req}_s$ after finitely many stages s, or the corresponding expressions are only true at points of the form 0^{m_s} finitely many times. All points in $L(A)^c$ are of the form 0^{m_s}. Now take stage t where this holds and:

1)$t>e$

2)$\sum_{i\leq e}\Psi_i(m_t) < 2^{m_t/2}$ & $\mathbf{max}\{q_i(m_t) \mid i \leq e\} < 2^{m_t/2}$

3) $0^{m_t} \in \sigma_e(A)$.

By assumption $0^{m_t} \in L(A)^c$, so the **else** case in the construction must have been chosen. However by construction the extension E' of the oracle at stage t is now chosen such that $0^{m_t} \in \sigma_e(B_{t+1} \cup E')$ contradicting assumption (3) Since all strings added to A at later stages have length at least $\mathbf{max}\ \{q_e(m_t), \Psi_e(m_t)\} +1$.

End of proof

The enumerator of relativised Σ_2^P expressions must generate infinitely many indices representing an empty set, we can again show that the diagonalisation is "slow enough" to render $L(A)^c$ infinite.

Lemma 26: $L(A)^c$ is infinite.

Proof: Suppose to the contrary that $L(A)^c$ is finite, then for all but finitely many stages the "then" case in the construction must be chosen. Hence at all but finitely many stages a requirement is satisfied, and its index is consequently in $\mathbf{Req}_s$-$\mathbf{Req}_{s+1}$. Since at each stage s+1 only one new index is in $\mathbf{Req}_{s+1}$-$\mathbf{Req}_s$. This set must remain bounded by a constant throughout the construction. However there are infinitely many indices of the empty set, each of which is added to **Req** at its own stage, and is never removed from **Req**.

End of proof

These observations together yield the final theorem of this chapter and the final theorem of this thesis:

Theorem 11: There exists a recursive oracle A such that $\Sigma_2^p(A)$ has a simple set.

EPILOGUE

We have considered in this thesis a large number of diagonalisation methods developed for and used in a polynomial setting. The interesting question to be put forward at this point is of course: "What makes 'em tick?" Or more precisely : "What are the special structural properties of polynomial time bounded complexity classes that make these methods work?" We will sum up what we feel are the most important arguments used either explicitly or implicitly in the constructions above. Two inherent arguments immediately draw attention:

1) Exponential is greater than polynomial

The exponential function 2^x eventually outgrows all polynomials p(x). The time bounds on all machines considered are polynomials, and a machine consumes a computation step by performing a query to the oracle. Since we are working with $\{0,1\}^*$ as an embedding for our languages, this means that for any machine φ_e there is a length n, where the machine on input of length n, cannot query all strings of length n. This situation would drastically change if we were to consider monosymbolic languages. It is remarkable that this argument seems to fail on the 3d level of the P-Time Hierarchy.

As the sum of polynomials is still a polynomial we have also seen that for fixed e the function 2^x eventually outgrows $\sum_{i \leq e} \Phi_i(x)$, and therefore this observation can also be used by forcing arguments where an unbounded number of requirements has to be "controlled" during an infinite number of steps.

Two observations can be made. First the time bounds are not actually used as time bounds, but rather as a bound on the *number of different queries possible.* In fact in most cases we just need that we can find *one* string not queried in the computation of a set of machines on certain inputs. Second the gap between polynomial and exponential is not used entirely. We just use that exponential is greater than polynomial. An infinite row $(g_i)_{i \in \omega}$ of functions can be conceived between 0 and 2^x, such that if the number of queries for machines in an enumeration is bounded by g_i a machine can be found in the next (i.e. bounded by g_{i+1} class) separating these two classes. Kintala and Fischer first observed this in [67]. An extensive survey of this observation can be found in Book[24]. Basically the same observation is used by Young[136] for the unrelativised complexity class NP. Instead of looking at NP entirely, the class is stratified into an infinite number of classes NP^k, where all machines in NP^k are uniformly time bounded by the polynomial n^k+k. The difference in the number of *steps* can now be used to derive separations between the different classes.

2) Infinite is greater than finite.
This is an argument used also in Recursion Theory and hence not specific for the polynomial setting. If a requirement has the potential of being satisfied infinitely many times, e.g. because the corresponding machine recognises an infinite language, and there are only finitely many requirements of higher priority, then the requirement will eventually be satisfied. (Or permanently satisfied in case of an injury argument.)

Specific for the polynomial setting is the nice side effect of this argument that all time bounds are finite. Hence in a computation the length of the strings written on the oracle tape are bounded by some constant known in advance. A machine φ_e in time $\Phi_e(n)$ cannot write strings longer than $\Phi_e(n)$ on its oracle tape, since the writing of each bit takes up one step. Hence considering only strings as candidates for the oracle of length longer than $\Phi_e(n)$, fixes the situation for machine φ_e on all inputs of length less than or equal to n.

The second question to be answered at this point is: "Why bother?", or: "why do we pursue results in a relativised polynomial setting, while knowing that such results can have no meaning for the P vs. NP question from which the research originated?" There are many answers possible to this question. We name three:

1) Methods are interesting.

The development of weapons to attack problems with is always a worth-while cause. Posts question was not answered by the study of Simple, Hypersimple and Hhsimple sets, but this study opened an entirely new field in Recursion Theory. To achieve stronger and stronger separation results in the relativised setting ever more complicated diagonalisation seem to be needed. Kozen[71], and later Regan[97] have indicated that:"If P *can* be separated from NP, then it can also be done by diagonalisation." Therefore the study of diagonalisation methods remains interesting.

2)Results are easily obtainable.

While there has been a flood of relativised results in the last few years, results in the unrelativised setting have been very meager. Some progress has been made [62][63], but the complicated paper of [96] is a strong witness that it is beyond human capacity to directly prove the power of nondeterminism.

3) $P \neq NP$ by contradiction

By showing ever stronger separations in the relativised setting we keep up the hope that someday a strong separation can be shown which is not possible for any oracle if one assumes that P=NP. It is worth mentioning at this point that up to now (and again to our knowledge), there is no result whatsoever of the type "there can be no oracle A such that $C_1(A)$ and $C_2(A)$ are separated by...". As it is unlikely that such a result would require diagonalisation (or could be achieved by diagonalisation) however, this is not the place to speculate on such a possibility. Recent results of Selman, Book and Xu Mei-Rui[112] show that bounding the number of queries can imply that results for relativised classes have consequences for the relation between unrelativised classes.

It should be mentioned here also that Hartmanis and Hemachandra [48] have recently derived some results on the *type of oracles* which separate relativised P from relativised NP under the assumption that P=NP, and opened up another direction of research. We are aware of the fact that "NP $\neq$ P by contradiction" is not the strongest thinkable motivation for research, but it is as strong a motivation as the motivation which can be given for proving the NP-Completeness of the 1000 and so maniest problem.

And finally "What about the Hierarchy?". We have set up and separated two levels of the hierarchy, but made no comments on the third and following levels. As we've already stated Polynomial vs Exponential is an argument which fails on the 3d level. Since the sizes of the instantiation sets become "two to the power polynomial" on both the Σ and the Π side. The alternation of quantifiers itself gives different size of instantiation sets, but the number of points on which these sets may vary stays limited to $2^{p(|x|)}$, therefore adding more quantifiers cannot "lift" the size of the instantiation sets above double exponential, and double exponential size is reached on both sides already at the third level.

By using a straightforward diagonalisation method and a connection between circuit theory and the P-Time Hierarchy, Fürst, Saxe and Sipser showed in [42] that "If the *parity function* of n boolean variables cannot be computed by bounded depth, $n^{\text{poly} \log n}$ size circuits, then there is an oracle A that separates PSPACE(A) from Σ_k^P for every k". Yao recently succeeded in showing an exponential lower bound for bounded depth circuits for the parity function and hence we may conclude that there must exist an oracle relative to which the P-Time Hierarchy is infinite. Unfortunately the argument of Fürst, Saxe and Sipser cannot be stretched to show the existence of an immune or simple set at each level of the Hierarchy. In view of the difficulty of constructive methods on low levels of the Hierarchy, and arguments given above it seems unlikely at this point that there also exists a constructive method for separating the Polynomial Time Hierarchy, let alone a constructive method for obtaining strong separations.

BIBLIOGRAPHY.

1. Ambos-Spies K., *P-mitotic sets*, Techn Report Nr. 167 U. Dortmund 1983.

2. Ambos-Spies K., *Three Theorems on Polynomial Degrees of NP-Sets*, Proc. 26th IEEE FOCS (1985) pp51-55.

3. Ambos-Spies K., *Complete Problems for Complexity Classes*, Univ Dortmund Forschungsbericht **200** (1985)

4. Ambos-Spies K., *On the Structure of the Polynomial Time degrees of Recursive Sets*, Habilitationsschrift (1985) Univ. Dortmund.

5. Ambos-Spies K., H. Fleischhack & H.Huwig, *P-Generic Sets*, Proc. 11th ICALP, Lect. Notes in Comp. Sci. **172** (1984) pp58-68.

6. Angluin D., *On Counting Problems and the Polynomial Time Hierarchy*, Theor. Comp. Science **12** (1980) 161-173.

7. Ausiello G, A. D'Atri & M. Protasi, *Lattice Theoretic Ordering Properties for NP-Complete Optimization Problems*, Univ. Roma (1978) TR78-18.

8. Ausiello G, A. D'Atri & M. Protasi, *Combinatorial Problems over Power Sets*, Univ. di Roma (1979) TR79-43

9. Axt P., *On a subrecursive hierarchy and primitive recursive degrees*, Trans. AMS **92** (1959).

10. Baker T, J. Gill & R. Solovay, *Relativizations of the P =NP question*, SIAM J. Comp. **4** (1975) 431 - 442

11. Baker T.P. & A.L. Selman, *A second step toward the polynomial hierarchy*, Theor. Comp. Science **8** (1979) 177-187.

12. Balcazar J.L., *Simplictiy, Relativisations and Nondeterminism*, SIAM J. of Comp. **14** (1985) 148-157.

13. Balcazar J.L., *Separating , Strongly Separating and Collapsing Relativised Complexity Classes*, Proc. MFCS(Invited Lecture), Lect. Notes in Comp. Sci. **176** (1984) pp1-16

14. Balcazar J.L. & R.V. Book, *Sets with Small Generalised Kolmogorov Complexity*, TR-Berkely-Calif. MSRI 00918-86.

15. Balcazar J.L. & R.V. Book, *On Generalised Kolmogorov Complexity*, Proc. 3d STACS, Lect. Notes in Comp. Sci. **210** (1986) pp334-340.

16. Balcazar J.L. , R.V. Book & U. Schöning, *On Bounded Query Machines*, Theor. Comp. Science **40** (1985) 237-244.

17. Benett, C.H & J. Gill, *Relative to a random oracle* $P(A) \neq NP(A) \neq co\text{-}NP(A)$ *with probability 1*, SIAM J. Comp. **10** (1981) 96-113

18. Berman P., *Relationship between density and deterministic complexity of NP complete languages*, proc. 5th ICALP, Lect. Notes in Comp. Sci. **62** (1979), pp63-71.

19. Berman P. & J. Hartmanis, *On Isomorphisms and Density of NP and Other Complete sets* , SIAM J. of Comp. **6** (1977) 305-327.

20. Book R.V., *Tally Languages and Complexity classes*, Inf. & Contr. **26** (1974) 186-193.

21. Book R.V., *On Languages accepted in Polynomial Time* , SIAM J. on Comp. **1** (1972) 27-39.

22. Book R.V., C. Wilson & Xu Mei Rui, *Relativizing Time and Space*, SIAM J. on Comp. **11** (1982) 571-581.

23. Book R.V. & C. Wrathal, *Bounded Query machines on NP() and NPQUERY*, Theor. Comp. Science **15** (1981) 41-50.

24. Book R.V., *Separating Relativised Complexity Classes*, to appear.

25. Book R.V., *Bounded Query Machines on NP And PSPACE,* Theor. Comp. Science **15** (1981) 27-39

26. Book R.V., A.L. Selman & C. Wrathall, *Inclusion complete tally languages and the Hartmanis-Berman Conjecture* ,Math. Systems Theory **11** (1977) 1-8

27. Book R.V. , T.J.Long & A.L. Selman , *Controlled relativizations of P and NP*, Theor. Comp. Science **145** (1983) 85-99

28. Book R.V., C. Wilson & Xu Mei-rui, *Relativizing time, space and time-space* , SIAM J. Comp. **11** 571-581 (1982)

29. Borodin A. Constable R.L. & Hopcroft J.E., *Dense and nondense families of complexity classes* , IEEE Conf. Record Tenth Annual Symp on Switching and Automata Theory pp7-19 (1969)

30. Cajori F, *A History of Mathematics*, Chelsea NY (1980).

31. Chew P. & M. Machtey, *A note on Structure and Looking Back applied to the relative complexity of computable functions* ,J.of Comp. and System Sci. **22** (1981) 53-59.

32. Cook S.A., *A Hierarchy of Nondererministic Time Complexity* , J. of Computer & System Sci. **7** (1973) 343-353

33. Cook S.A., *An observation on Time-Storage Tradeoff,* Proc. 5th ACM STOC (1973) pp29-34.

34. Cook S.A., *The complexity of Theorem Proving Procedures*, Proc. 3d ACM STOC Science (1971) pp151-158

35. Daley R.P. & W. Reynolds, *Towards Modular constructions of unsolvability*, Univ. of Pittsburg TR80-05

36. Dowd M., *Forcing the P hierarchy* , Laboratory for Computer Science Research, Rutgers Univ. LCSR-TR-35 (to appear)

37. Dowd M. , *Isomorphism of complete sets* , Laboratory for Computer Science Research, Rutgers Univ. LCSR-TR-34 (to appear)

38. Fine R, *Kronecker and his Arithmetical Theory of the Algebraic Equation*, Bulletin of the American Mathematical Society **1** (1892) p183.

39. Flajolet P. & J. Steyaert , *On sets having only hard subsets* , Proc. 2nd ICALP, Lect. Notes in Comp. Sci. **14** (1974) pp446-457.

40. Fortune S. , *A note on sparse complete sets*, SIAM J. of Comp. **8** (1979) 431-433.

41. Fürer M, *The tight deterministic time hierarchy*, Report on the 1st GTI workshop Univ. of Paderborn (1983) pp96-104.

42. Fürst M., J. Saxe & M. Sipser, *Parity Circuits and the Polynomial Time Hierarchy*, Proc. 22nd IEEE FOCS (1981) pp260-270.

43. Garey M.R. & D.S. Johnson, *Computers & Intractability*, W.H. Freedman & Co S. Fransisco (1979)

44. Hartmanis J. & Mahaney S., *An essay about research on sparse NP-Complete sets*, Comp. Sci. Dept. TR80-422 Cornell (1980)

45. Hartmanis J. V. Sewelson & N. Immerman, *Sparse Sets in NP-P: EXPTIME versus NEXPTIME*, Proc. 15th ACM STOC (1983) pp381-391.

46. Hartmanis J. , *Generalized Kolmogorov Complexityand the Structure of Feasible Computations*, Proc. 24th IEEE FOCS(1983) pp439-450.

47. Hartmanis J. , *On sparse sets in NP-P*, Information Processing Letters **16** (1983) 55-60.

48. Hartmanis J. & L. Hemachandra , *On Sparse Oracles Separating Feasible Complexity Classes*, Proc. 3d STACS, Lect. Notes in Comp. Sci. **210** (1986) pp321-333.

49. Heller H. , *On relativized Polynomial Hierarchies extending two levels*, Conference on Computational Complexity,S. Barbara California pp109-114 (1983)

50. Hofstadter D.R., *Gödel, Escher Bach: an eternal golden braid*, Vintage Books N.Y.

51. Homer S. & I. Gasarch , *Relativizations comparing NP and Exponential time*, Information & Control (1985)

52. Homer S. & W. Maass , *Oracle dependent properties of the lattice of NP sets* , Theor. Comp. Science **24** (1983) 279-289

53. Hopcroft J. And J. Ullman , *Introduction to Automata Theory Languages and Computations*, Addison-Wesley Reading Mass. 1979

54. Hopcroft J. & W. Paul , *A Graph Theoretical approach to time versus space and other results*, Unpublished Manuscript (Cornell)

55. Ibarra O. , *A note concerning nondeterministic tape complexities*, Journal of the ACM **19** (1972) 608-612

56. Immerman & Mahaney , *Oracles for which NP has polynomial Size circuits*, Conference on Computational Complexity S. Barbara California (1983) pp89-93

57. Jockush C. , *Notes on genericity for r.e. sets*, to appear

58. John T. , *A weak separation result between deterministic and Non deterministic Log Space*, Conference on Computational Complexity S. Barbara California. pp128-139 (1983)

59. Johnson D.S. , *Approximation Algorithms for Combinatorial problems*, Proc. 5th ACM STOC pp38-49 (1973)

60. Johnson D.S. , *The NP Completeness Column:An ongoing guide*. Journal of Algorithms 3 $\rightarrow$...

61. Kannan R. , *Alternation and the power of nondeterminism*, Proc. 15th ACM STOC (1983) pp344-346

62. Kannan R. , *Toward separating Nondeterministic Time from Deterministic Time.* Proc 22th IEEE FOCS (1983) pp344-346

63. Kannan R. , *A method for proving the power of Nondeterminism*, Conference on Computational Complexity S. Barbara California pp56-63 (1983)

64. Karp R.M. & R.J. Lipton , *Some connections between non-uniform and uniform complexity classes*, Proc. 12th ACM STOC (1980) pp302-309

65. Karp R.M. , *Reducibility among Combinatorial Problems*, Complexity of Computer Computations, R.E. Miller & J.W. Thatcher Eds. Plenum N.Y. pp85-103 (1972)

66. Karp R.M., *Combinatorics, Complexity, and Randomness*, Communications ACM **29** (1985) 97-111.

67. Kintala C.M.R & P. Fischer , *Refining Nondeterminism in relativized polynomial time bounded computations*, SIAM J. Comp. **9** (1980) 46-53.

68. Kintala C.M.R. , *Computations with a Restricted Number of Nondeterministic steps*, Ph. D. Dissertation Pennsylvania State University (1977)

69. Kleene S.C. , *Recursive Predicates and Quantifiers*, Tans. Am. Math. Soc. **53** (1943) 41-73.

70. Ko, K & D. Moore , *Completeness, approximation and density*, SIAM J. of Comp. **10** (1981) 787-796

71. Kozen D.C. , *Indexing of Subrecursive Classes*, Proc 10th ACM STOC (1978) pp287-295

72. Kozen D. & M. Machtey , *On relative diagonals*, IBM Res. Rep. RC8184 (1980)

73. Kronecker L., *Ueber den Zahlbegriff,* Die Reine und Angewandte Mathematik Bd**101**(1887) p43.

74. Kurtz S.A. , *Sparse sets in NP-P: Relativizations,* SIAM J: on Comp. **14** (1985) pp113-119.

75. Kurtz S.A. , *The Fine structure of NP Relativizations.,* Conference on Computational Complexity S. Barbara California pp42-50 (1983)

76. Ladner R.E. , *On the structure of Polynomial Time Reducibility,* Journal of the ACM **22** (1975) 155-171

77. Ladner R.E., N.A. Lynch & A.L. Selman , *A Comparison of Polynomial time Reducibilities,* Theor. Comp. Science **1** (1975) 103-123

78. Landweber L. R.Lipton & E. Robertson , *On the structure of sets in NP and other complexity classe*s. Comp. Science Dept. TR342 Univ. of Wisconsin-Madison.(1978)

79. Levin L.A., *Universal Sequential Search Problems,* Problems of Information Transmission **9** (1973) 265-266

80. Lewis H.R. & C.H. Papadimitriou, *Elements of the Theory of Computation,* Prentice Hall (1981).

81. Long T. , *Strong nondeterministic time,* Theor. Comp. Science **21** (1982) 1-25

82. Long T. , *Relativizing nondeterministic time* , unpublished manuscript.(1981)

83. Long T.J. , *On relativizing Complexity Classes.* Conference on Computational Complexity S. Barbara California (1983) pp104-108

84. Lynch N. , *On reducibility to complete or sparse sets,* Journal of the ACM **22** 341-345 (1975)

85. Lynch N. , *Relativization of the theory of computational complexity,* Tech. Rep TR-99, Project NAC; Ph. D. Th. M.I.T Cambridge Mass. (1972).

86. Maass, W. , *Recursively enumerable generic sets* , J. Symb. Logic **47** (1982) 809-823.

87. Machtey M. , *Classification of computable functions by primitive recursive classes,* Proc. 3d ACM STOC (1971) pp251-257.

88. Machtey M. , *The honest subrecursive classes are a lattice.* Inf. & Control. **24** (1974) 247-263

89. Machtey M. , *On the density of honest subrecursive classes.* Tech Rep. CSD TR 92, Comp. Sci. Dept Purdue Univ. Lafayette Ind. (1973)

90. Machtey M. & P. Young , *An Introduction to the general Theory of Algorithms.,* Theory of Computation Series, P.J. Fischer (ed), Elsevier North Holland (1978)

91. Mc Aloon K. , *Models of Arithmetic & Problems in Complexity Theory,* Conference on Computational Complexity S. Barbara California pp16-29 (1983).

92. Mahaney, S.R. , *On the number of p-isomorphism classes of NP-Complete sets* , Proc. 22th IEEE FOCS (1981) pp271-278.

93. Mahaney, Stephen R., *Sparse Complete Sets for NP: Solution of a Conjecture by Berman & Hartmanis* , Rep TR80-417 Cornell Univ. DCS.

94. K. Melhorn , *Polynomial and abstract subrecursive classes,* Ph.D. Thesis Cornell Univ. (1974).

95. Papadimitriou C.H. & K. Steiglitz , *Some complexity results for the TSP* , Proc. 8th ACM STOC (1976), pp1-9.

96. Paul W., N. Pippenger, E. Szemeredi & W. Trotter , *On determinism versus nondeterminism and related problems,* Proc. 24th IEEE FOCS (1983) pp429-438

97. Regan K., *Computability enumerability and the polynomial hierarchy*, Oxford Univ. M.Sc. qualif. diss. 1982.

98. Regan K.W., *On Diagonalization Methods & The structure of Language Classes*, Proc. FCT, Lect. Notes in Comp. Sci. **158** (1983) pp368-380.

99. Rogers Jr. H., *Theory of Recursive Functions and Effective Computability*, Mc Graw-Hill Book Company (1967)

100. Rosier L.E. & H.C. Yen, *Logspace Hierarchies, Polynomial Time, and the Complexity of Fairness Problems Concerning ω-Machines*, Proc. 3d STACS, Lect. Notes in Comp. Sci. **210** (1986) pp306-320

101. Russo D. & Zachos St., *Relationships between probabilistic Polynomial Complexity Classes and their Relativizations*, Conf. on Comp. Complexity S. Barbara California (1983) pp115-122.

102. Savelsbergh M. & P. van Emde Boas, *BOUNDED TILING, an alternative to SATISFIABILITY?*, Report CWI(1984) OS-R8405

103. Sahni S., *General Techniques for combinatorial Approximation*, TR76- Univ. of Minnesota DCS.

104. Santos E., *Computability by probabilistic Turing Machines*, transactions AMS **159** (1971) pp165-185

105. Schöning U., *Relativization and infinite subsets of NP sets* unpublished manuscript (1982)

106. Schöning U., *A uniform approach to obtaining diagonal sets in complexity classes*, Theor. Comp. Science **18** (1982) 95-103

107. Schöning U., *Untersuchungen zur strukture von NP*, Ph.D. Dissertation Stuttgart Univ. 1981

108. Schöning U. , *A high and a low hierarchy within NP*, Conference on Computational Complexity S. Barbara California (1983) pp56-63.

109. Schöning U. & R.V. Book , *Immunity, Relativization & Nondeterminism*, SIAM J. Comp. **13** (1984) 329-337.

110. Seiferas J.I. , *Techniques for separating complexity Classes*, J. Comp. Sys. Sci. **14** (1977) 73-99.

111. Seiferas J.I., M.J. Fischer & A.R. Meyer , *Refinements of the non-deterministic time and space hierarchies* , IEE Symp on Switching and Aut.th.**14** (1973) pp130-137.

112. Selman A.L , Xu mei-rui & R.V. Book, *Positive relativizations of complexity classes*, SIAM J. of Comp. **12** (1983) 565-579.

113. Selman A.L. , *Analogues of Semirecursive Sets &Effective Reducibilities to The Study of NP Complexity*, Inf. & Control **52** (1982) 36-51.

114. Selman A.L. , *P-Selective sets, Tally Languages and the behaviour of polynomial time reducibilities on NP*, Math. Systems Theory **13** (1979) 55-65.

115. Selman A.L. , *Polynomial Time Reducibilities: The modus operandi of complexity theory*, Conference on Computational Complexity S. Barbara California (1983) pp82-88

116. Selman A.L. , *Reductions on NP and P-selective sets*, Manuscript Iowa State Univ.

117. Selman A.L. , *Polynomial time Enumeration Reducibility*, SIAM J. of Comp **7** (1978)

118. Sewelson V. , *The structure of NP under relativization*, Conference on Computational Complexity S. Barbara California (1983) pp93-103

119. Soare R.I. , *Computational Complexity of RE sets*, Inf. & Control **52** (1982) 8-18

120. Soare R.I. *Recursively Enumerable Sets and Degrees: The Study of Computable Functions and Computably Generated Sets*, to appear in Springer Ω series.

121. Solovay R. , *On sets Cook-reducible to Sparse sets,* SIAM J. of Comp. **5** (1976) 646-652

122. Stockmeyer L.J. , *The Polynomial Time Hierarchy,* Theor. Comp. Science **3** (1976) 1-22

123. Torenvliet L. & P. van Emde Boas , *Combined Simplicity and Immunity in Relativized NP* , Proc. 2nd STACS Lect. Notes in Comp. Sci. **182** (1985) pp339-350.

124. Torenvliet L .& P. van Emde Boas , *Diagonalisation Methods and the Separation of Relativized Complexity Classes,* Univ. of Amsterdam TR-MI-84-31.

125. Torenvliet L .& P. van Emde Boas , *Diagonalisation Methods in a Polynomial Setting,* Univ. of Amsterdam TR-MI-86-08.

126. Torenvliet L. , *Towards a Strong P-Time Hierarchy,* Univ. of Amsterdam TR-FVI-85-03.

127. Torenvliet L., *Simplicity for relativised* Σ_2^p, Univ. of Amsterdam TR-FVI-85-04.

128. Turing A.M. , *On Computable Numbers, with an application to the Entscheidungsproblem,* Proc. London Math Soc.**2** (1936/37) pp230-265

129. Van Emde Boas P., *The Second Machine Class: Models of Parallelism,* in J.van Leeuwen & J.K. Lenstra (eds) CWI Syllabus **9** (1986) pp133-161.

130. Vitány P.M.B, *Counting, Number Representation and Counter Machine Algorithms,* to appear.

131. Wilson C.B. , *Relativization, Reducibilities, and the exponential time Hierarchy,* Techn. Rep. 140/80 DCS Univ. of Toronto (1980)

132. Wrathal C. , *Complete sets and the polynomial hierarchy,* Theor. Comp. Science **3** (1976) 23-33.

133. Xu mei-rui, J. Doner & R.V. Book , *Refining Nondeterminism in Relativized Complexity Classes.* Journal of the ACM **30** (1983)

134. Yao A.C., *Separating the Polynomial-Time Hierarchy by Oracles: Part I*, Proc. 26th IEEE FOCS (1985) pp1-10

135. Yesha , *On certain polynomial time truth table reducibilities of complete sets to sparse sets.* SIAM J. of Comp. **12** (1983) 411-425

136. Young P., *Some Structural Properties of Polynomial Reducibilities and Sets in NP*, Proc. 15th ACM STOC (1983) pp392-401.

137. Zachos S., *Collapsing probabilistic polynomial hierarchies*, Conference on Computational Complexity S. Barbara California pp75-81 (1983)

SUMMARY

In this thesis we show by construction of *oracle sets* that a strong separation of complexity classes in the lower levels of the *Polynomial Time Hierarchy* is possible.

In chapter I we develop a model for the interpretation of the constructions in subsequent chapters in an informal but precise way, and we develop precisely that part of computational complexity theory needed to understand the notions used in these chapters.

In chapter II we (partially) quote existing theory, and prove one new theorem. We show in this chapter the constructions which would give to oracle set A the following properties respectively: NP(A) $\neq$ P(A), NP(A) $\neq$ Co- NP(A), NP(A) has a P(A)-immune set, NP(A) has a simple set (with two different constructions), and NP(A) has a single set which is both simple and P(A) immune.

In chapter III we treat the separation results between the first and second level of the hierarchy. We limit ourselves in this chapter to strong separations. We show the constructions which would give to oracle set A the following properties respectively: Σ_2^p(A) has an NP(A) immune set, Π_2^p(A) has an NP(A) immune set, Σ_2^p(A) has an NP(A) bi-immune set, Σ_2^p(A) has an Δ_2^p (A) immune set, and Π_2^p(A) has an Δ_2^p (A) immune set.

In chapter IV we treat the construction of an oracle set A such that Σ_2^p(A) has a simple set, and thus obtain a strong separation between Σ_2^p(A) and Π_2^p(A) on the second level of the hierarchy.

SAMENVATTING

In dit proefschrift wordt door constructie van een aantal *orakelverzamelingen* aangetoond dat een sterke scheiding tussen de complexiteitsklassen in van de onderste lagen de *Relatieve Polynomiale Tijd Hierarchie* mogelijk is.

In hoofdstuk I wordt op informele doch precieze wijze uitgaande van de natuurlijke getallen een model gemaakt voor de interpretatie van de constructies in de volgende hoofdstukken, en wordt precies dat deel van de complexiteitstheorie geschetst dat nodig is voor het hanteren van de begrippen die in deze volgende hoofdstukken worden gebruikt.

In hoofdstuk II wordt (een gedeelte van) de bestaande theorie aangehaald, en één nieuwe stelling bewezen. We laten in dit hoofdstuk de constructies zien van een orakel A voor achtereenvolgens de eigenschappen: $NP(A) \neq P(A)$, $NP(A) \neq Co\text{-}NP(A)$, NP(A) heeft een P(A)-immune verzameling, NP(A) heeft een simpele verzamelingen (met twee verschillende constructies), en NP(A) heeft een verzameling die zowel simpel als P(A)-immuun is. Deze resultaten behandelen het eerste niveau van de hierarchie.

In hoofdstuk III behandelen we de resultaten van scheidingen tussen het eerste en het tweede niveau van de hierarchie. We beperken ons in dit hoofdstuk tot de sterke scheidingen. We laten in dit hoofdstuk de constructies zien van een orakel A voor de volgende eigenschappen: $\Sigma_2^P(A)$ heeft een NP(A) immune verzameling, $\Pi_2^P(A)$ heeft een NP(A) immune verzameling, $\Sigma_2^P(A)$ heeft een NP(A) bi-immune verzameling, $\Sigma_2^P(A)$ heeft een $\Delta_2^P(A)$ immune verzameling, en $\Pi_2^P(A)$ heeft een Δ_2^P (A) immune verzameling.

In hoofdstuk IV behandelen we de constructie van een orakel A zo dat $\Sigma_2^P(A)$ een simpele verzameling heeft, en bereiken aldus een sterke scheiding tussen $\Sigma_2^P(A)$ en $\Pi_2^P(A)$ op het tweede niveau van de hierarchie