Department of Computer Science
Faculty of Mathematics, Physics and
Informatics
Comenius University, Bratislava

The Tape–size and Extended Chomsky hierarchy

(Master Thesis)

Ľuboš Steskal

Advisor:
Prof. RNDr. Branislav Rovan, PhD.

Bratislava, 2006

I hereby declare that I wrote this thesis by myself, only with the help of the referenced literature, under the careful supervision of my thesis advisor.

.....

Acknowledgments

I thank my supervisor, Prof. Branislav Rovan, for insightful conversations during the development of the ideas in this thesis, and for helpful comments on the text.

I warmly thank my family for everlasting support, care and love. A special thank you goes to all my friends, they mean very much to me.

I would like to express my gratitude to those, who have ignited my motivation for this work by discussing interesting problems–lessons on metaphysics and numbers.

Abstract

By proposing a new computational model we studied properties of infinite computations. It showed up, that they are closely linked to the structure of the Σ_3 level of the arithmetical hierarchy. We proved that the Δ_3 set consists of an infinite hierarchy starting with the set Σ_2. We showed that the structure of the top five levels of this hierarchy is in some sense similar to the structure of the Chomsky hierarchy, while the bottom levels remind us of the bounded oracle query hierarchy.

Keywords: Super Turing computation, arithmetical hierarchy, infinite computation, Chomsky hierarchy.

Contents

Chapter 1

Introduction

The theory of recursive functions and effective computability is a scientific field with many interesting results; some of them might have philosophical impact. It is well known that there are problems, which cannot be solved by algorithms. In fact, a whole hierarchy of unsolvable problems opens before us, if we go a little bit deeper in our studies.

There are many results that give us better understanding of both formal languages of logic and of computer science [HR92]. Although, there are several ways to study this subject, oracle machines surely provide a good computational model to consider.

In recent years several new computational models differing from the standard framework were proposed. These models shared the attribute of being stronger then the Turing Machine. Some of them were purely theoretical concepts, some tried to model some phenomenon that could be observed in reality [EN02]. Such models are sometimes called *Hyper-machines*, or machines with *Super Turing computational powers*. The aim of this thesis is to study the properties of possibly infinitely long computations. Inspired by the models of the *Accelerated Turing Machine* proposed by Bertrand Russell, Ralph Blake and Hermann Weyl and the *Infinite Time Turing Machine* proposed by Joel Hamkins and Andy Lewis [HL00] we define a new model of our own. The reason why we do this is that we want to separate the infinite and non infinite parts of the computation. This shall allow us to study the asset of infinite computation and identify factors affecting the power of models

utilizing such computations.

Our model consists of two parts. A machine capable of an infinitely long computation and a language (which can be interpreted as another machine) processing its output. By placing constrains to the first or second machine, we obtain classes of languages which form hierarchies. One such hierarchy is similar to the one obtained by limiting the number of queries of an oracle machine [Bie95] [BGH89]. The other hierarchy looks more like the *Chomsky hierarchy*, but it is in the Σ_3 class of the Arithmetical hierarchy.

1.1 Super Turing Computation – an Overview

We shall now present various Super Turing models in a more detailed view. We shall put emphasis on the motivation for such models as well as their computational power and resources used to achieve this power. One study regarding the resources used is presented in [Ord02].

One of the first models capable of hyperturing computation was the Oracle Turing Machine proposed by Alan Turing. This model consists of a standard Turing Machine equipped with an Oracle – a device capable of answering a specified group of questions[1]. The oracle actually models some form of a subroutine or a subprogram realized by a 'third person'. In case the answers of the oracle are not computable by algorithmic means (e.g., it is a non–recursive set), the Oracle Turing Machines obviously gain Super Turing computational power. The actual computational limits depend on the non–recursiveness of the oracle. This model is often used in logic and computer science as well.

Another concept, very similar[2] to the Oracle Machines is the Turing Machine with Initial Inscriptions. Such machine has one additional read–only tape with one possibly infinite word written on it (this word is fixed for all inputs). This way, we may model the situation when an algorithm has access to an external database. Sometimes oracles are modeled in this way by making the infinite word consist of lexicographically ordered elements of

[1] Oracles providing yes/no answers are often represented as sets consisting of all questions with a 'yes' answer.

[2] and in some aspects identical

the oracle (separated by a special delimiter symbol). Thus again by the use of some non–recursive initial inscription, the respective machine would gain Super Turing computational power. However, this model does not share all of its properties with the Oracle Machines, since the answer of an oracle query is realized in just one computational step whilst the examination of the read–only tape may take much more time.

The Turing Machines with Advice represent another concept capable of Super Turing computation. They actually represent a special class of oracle machines[3]. The idea is that the answer of the Advice depends only on the size of the word on input. The answer however may be of any size (in the general case). In the theory, the length of the Advice is often bounded by some function having the length of input as its argument. Again by having the Advice complicated (and long) enough, this model gains computational power beyond the Turing Machines.

All of these models have one common feature. They consist of a standard Turing machine and an external source of information that they can harness at any time as 'desired' by the respective machine. We shall now present another group models differing in their inability to control their external source of information.

Interactive Turing machines proposed in [WvL01] differ from standard Turing machines in that they have in addition a special input port for receiving information from the outside world. However, they cannot directly control what information is to be provided. Still, if the information supplied by the external source is of non–recursive character the model might gain Super Turing power.

Another group of models of the same type are Turing Machines with some form of noise. This means, that from time to time the head writes something else as desired, or the contents of some squares change due to some external agent and not by the means of the machine[4]. If the machine can identify these changes and the information gained by the rewriting is non–recursive,

[3]Here the oracle is rather called Advice (function).

[4]This can be seen as an influence of noise or an intervention of an out side observer.

the model may again gain powers greater than those of the Turing Machines.

The next class of Super Turing models are models with infinite specification allowed. By this we mean non–uniform Boolean Circuits and Turing Machines with infinitely many states and an infinite transition function. If there are absolutely no bounds placed on their infinite specification, these models obviously represent the strongest possible form of computation, since they can actually represent any set by directly listing all of its elements.

There is another interesting model lying somewhere between the domains of infinite specification and external information source – Neural Networks with real valued weights. These machines consist of a finite net (or oriented graph) of nodes. Each edge has its weight and in each node, there is a processor. In one computational step, each processor counts the linear combination of the outputs[5] of the processors attached to it and the weights of their respective connections. Then this result is used as an argument in some sigmoid function[6] and the result is its output. It has been shown in [SS92] that such machines (without any bounds) have computational power equal to the non–uniform circuits. However, if we bound the numbers used as weights to be rational or natural numbers, their computational power is reduced to the domain of standard algorithms [SS95]. It is expectable that by bounding the weights to be Σ_i enumerable reals, we would gain a hierarchy of different computational power[7].

The models in the last group draw their computational power from a little bit different source. They have finite specification and lack any additional (possibly infinite) external information. The quality that they utilize to achieve supreme computational power is infinitely long computational time.

First such model is the Accelerating Turing Machine. This model is inspired by the Zeno's aporiae with the tortoise. After the first computational

[5] from the previous computational step

[6] The concrete function depends on the specific model

[7] The author believes that this hierarchy would match with the Arithmetical Hierarchy.

step takes place, the time until the second step is realized is one half of a second. Each of the following steps takes only half of the time as the previous did. Thus, after one second the machine has preformed infinitely many computational steps. In this model the tape of the machine contains one special square whose content may be changed only once during the whole computation. The acceptance of the machine depends on the content of this square after infinitely many steps.

Another model working in a similar way was proposed in [EN02]. In that model the infinite computation is realized by a pair of computers – one orbiting a black hole and another falling into it. The model utilizes the fact, that in the Malament-Hogarth Spacetimes the time flows differently for both of the computers. While from the orbiting computer's point of view it would take infinite time until the falling computer 'hits' the black hole, the falling computer will reach it in finite time (from its point of view). Thus the orbiting computer may perform a possibly infinite computation and if the computation halts, send the result to the falling machine. So by the time the falling machine reaches the black hole, it will know the answer to the halting problem of the orbiting machine. In [WvL02], it has been shown that this model is equal to a special class of Turing Machines with Advice.

The last model mentioned in this overview is the Infinite Time Turing Machine [HL00]. This model works as a standard Turing machine, only that it is able to preform a number of computational steps equal to some ordinal. For each computational step whose number is a limit ordinal, the machine enters a special transfinite state, the content of the tape is the limit of the contents so far and the head is placed on the first square of the tape. This is the most general model for infinite computation.

We can see that there are several, maybe at first sight heterogeneous groups of Super Turing models: those with controlled/uncontrolled external information, infinite specification and infinite computation time. If we take a closer look, we might see that these groups are linked closely together. It can easily be seen, that models with external information can by simulated by oracle machines with an appropriate oracle. For example, to simulate

machines with an external observer intervening in the computation, the oracle might be the set of all actions of the observer and after simulating each step of the respective machine, we might ask the oracle what changes will be done by the observer.

The infinite specification can be simulated in a natural way by machines with initial inscription, by simply having the code of the infinite machine written on the read only tape.

The Accelerating Turing Machines actually correspond to the halting problem of standard Turing Machines. The interesting thing is, that in this model, there are no problems with diagonalization, since these machines cannot make any computation after the first second and thus, cannot accept if another machine rejected by an infinite computation.

Thus, most (if not all) of these machines can be more or less simulated by the oracle machines.

There is also another point of view at these groups of models, namely the constructive one. We might argue, that to supply one model with some non–recursive[8] information, the information must have been created in advance. From this point of view the models with infinite time might be considered most interesting, since they have neither external nor internal non–recursive information. Thus in this thesis, our aim is to explore the infinite computation and identify the factors having influence on its power. We shall only study infinite computation with just one transfinite step.

Looking at all these models, another interesting question might arise. Most of these models simulate some (at least in our imagination) physical devices. The concepts of movement, speed and time are obviously based on Newtonian view of physics. It might be interesting to mix these models with the relativity or quantum theory.

The mathematical apparatus is based on the standard set theory. It might also turn out to be of some interest, to study these models from the alternative set theory point of view.

[8]external in an oracle, internal in a specification

Chapter 2

Notation and Preliminaries

In this part, we introduce a new computational model– the *Display Turing Machine*[1]. These machines enable us to study functions that require possibly infinite computational time. The model looks like a standard one–tape Turing Machine equipped with an additional tape. We shall call this additional tape the Display Tape[2]. During its computation, the TMD can use both of the tapes as standard tapes. After the computation is done, the content of the display tape shall be considered to be the result of the computation. If the computation does not halt after finite time, then the limit of the display tape shall be the result. The display tape represents the information, we are able to directly receive from the infinite computation (e.g. by measurement). We shall also say that the TMD transforms the input word to a display word (the limit contents of the display tape).

We can also use this model to accept languages. This can be done by specifying a set of words that are allowed to occur on the display. Thus a TMD will accept an input word w if and only if it transforms it into a display word from this set. We shall denote this set as *Control*, since it controlls whether a word is accepted or not. We shall call this whole accepting mechanism a *Display Turing Machine with Control* and use the acronym $TMDC$.

[1] We shall use the acronym TMD (Turing Machine with Display) to avoid confusion with the Deterministic Turing Machine.

[2] We might refer to these tapes as to the *firs tape* and *second tape*. By second tape we mean the display tape, first tape represents the "standard" tape while .

2.1 Definitions

We shall now provide formal definitions of the Display Turing Machine and Display Turing Machine with Control. We use the notation and terminology similar to the one used in [HU79]. We also expect the reader to have basic knowledge of formal languages and automata.

Definition 2.1.1. *Let Λ be a finite alphabet*[3]*. We denote by Λ^* the set of all finite words consisting solely of letters in Λ. We denote by Λ^ω the set of all infinite words consisting solely of letters in Λ. We denote by $\Lambda^\infty = \Lambda^* \cup \Lambda^\omega$.*

Informally, a Display Turing Machine works in the following way. At the beginning of the computation, there is only the input word w written on the first tape. The head of this tape is on the first letter of the input. The display tape is empty and the head of the display tape is at the beginning of the display tape. The machine is in a special state called the initial state. Once the computation starts, each head reads the content of the respective square of the respective tape. Then according to the letters red by the heads and the state of the machine, each head writes some symbol onto its current position on the tape and moves to one of the adjacent squares, or remains where it is.

These changes constitute a computational step. After the first computational step is accomplished, the second computational step is realized, etc. Thus the whole computation consists of a sequence of consecutive computational steps. With a word on input, all the information necessary to perform each computational step is stored in the transition function δ. Given the symbols read by both heads and given the state of the machine, δ tells the heads what to do as well as what should the new state of the machine be.

The result of the computation is the content of the Display tape after infinitely (ω–many) computational steps. The special symbol † used in the output of the machine is an indicator saying that the content of the square where it is was changed infinitely many times and thus there is no *limit content* of that square[4].

[3] We use the symbol Λ instead of Σ because Σ_i is used to denote the i-th enumerable level of the Arithmetical Hierarchy.

[4] We can view this situation so that the square of the tape was scratched and damaged

Definition 2.1.2. *A* Display Turing Machine *is a* 5*-tuple* $(\Lambda, \Gamma, \delta, K, q_0)$ *where*

$\Lambda \subseteq \Gamma$ *are the input and auxiliary alphabets,* B *and* \$ *in* $\Gamma - \Lambda$ *are special symbols for a blank space and the beginning of the display tape,* $\dagger \notin \Gamma$ *is a special symbol that is not allowed to be in* Γ*,*

K *is a finite set of all states of the machine,* $q_0 \in K$ *is the initial state and*

$\delta\colon K \times \Gamma^2 \to K \times (\{\Gamma - \{B\}\} \times \{-1, 0, 1\})^2$ *is the transition function such, that* $\delta(q, a_0, a_1) = (p, a_0', d_0, \$, d_1)$ *if and only if* $a_1 = \$$*, and* d_1 *never equals* -1 *if* a_1 *is* \$*.*

The configuration of the Display Turing Machine is a fivetuple containing the current state of the computation, i.e. the state of the machine, the content of the tapes and the positions of heads. An example is $(p, BwB, 5, \$vB, 2)$. Here p denotes the current state of the machine, w is the word written on the first tape, 5 means, that the head on the first tape is on the fifth square, v is the word written on the display tape and 2 stands for the position of the head reading the display tape.

Definition 2.1.3. *A* Configuration of TMD T *is an element of the set* $K \times B(\Gamma - \{B\})^*B \times \mathbb{N} \times \$(\Gamma - \{B, \$\})^*B \times \mathbb{N}$

We can now formally define a computational step. It is actually the transition from one configuration to another by only one application of the δ–function.

Definition 2.1.4. A computational step of a TMD T *is a relation* $\vdash_T$ *on configurations of TMD defined by* $(p, Bu_0u_1 \dots u_nB, k, \$v_0v_1 \dots v_mB, l) \vdash_T (q, Bu_0u_1 \dots u_{k-1}au_{k+1} \dots u_nB, k+d_0, \$v_0v_1 \dots v_{l-1}bv_{l+1} \dots v_mB, l+d_1) \iff \delta(p, u_k, v_l) = (q, a, d_0, b, d_1)$*. We omit the index* T *and shall write only* $\vdash$ *if it is obvious what machine we are referring to.*

The resulting content of the TMD is defined for each square of the display tape and these are then concatenated to form the resulting word. The result of each square is seen as a limit of the contents of that square during

by infinite rewriting.

computation. If the content of the square does not change from some moment on (if it changes only a finite number of times), then its display content of it is the last letter written. On the other hand, it the content changes unbounded number of times, then its display content is a special symbol † which means, that the square was scratched and there is no proper result.

Definition 2.1.5. *Let T be an TMD and $w \in \Lambda^*$. Let $D^{T(w)} = \{d^{T(w)}(n)\}_{n=0}^{\infty}$ denote the sequence of contents of the display tape of the machine T working on the input w. Let $D_i^{T(w)} = \{d_i^{T(w)}(n)\}_{n=0}^{\infty}$ denote the sequence of letters written on the i-th square of the display tape during the computation of T on the input w (where the beginning of the tape \$ is considered to be the minus first square). Let $\vec{d}_i^{T(w)}$ be the limit of $D_i^{T(w)}$ if it converges, † if it does not. If there exists an l such that $l = \min\{i | D_i$ converges to $B\}$, then we say that*

$$T(w) = \vec{d}_0^{T(w)} \vec{d}_1^{T(w)} \dots \vec{d}_{l-1}^{T(w)}$$

is the result of the TMD T *with w on input.*

If there is no such l then the result of the TMD T *with w on input is the infinite word*

$$T(w) = \vec{d}_0^{T(w)} \vec{d}_1^{T(w)} \vec{d}_2^{T(w)} \dots$$

We shall now define the Display Turing Machine with Control. It is a computation model consisting of two main parts. The TMD and a set called the Control language. If the output of the Display Turing Machine is in the Control language, then the word that was on input is accepted by the $TMDC$. In this thesis, we shall show that using more complex Control language allows the $TMDC$ to accept more languages. Thus the infinite computation can not do all the job by itself. In some sense, the Control represents some after-processing of the infinite computation.

Definition 2.1.6. *The* Display Turing Machine with Control *is a pair $A = (T, S)$ such, that T is an TMD and S is a set. We shall call S the* Control set *of A and sometimes refer to it as $C(A)$.*

Definition 2.1.7. *Let $A = (T, S)$ be a Display Turing Machine with Control. We define $L(A) = \{w \in \Lambda^* | T(w) \in S\}$. $L(A)$ is the language accepted by A.*

We shall now define a (partial) function alternative to the Display Turing Machine with Control.

Definition 2.1.8. *Let T be a TMD with Γ_1 as its working alphabet, Γ_2 be a finite alphabet and f a (partial) function $f\colon (\Gamma_1 \cup \{\dagger\})^\infty \to \Gamma_2^\infty$. We shall call the pair (T, f) a* Display Turing Machine function.

We shall now define a few useful functions and notations.

Notation 2.1.1. *Let $f(n)\colon \mathbb{N} \to \mathbb{N}$ be a nondecreasing function. We denote by $\mathcal{T}_{f(n)}$ the class of all TMD for which the size of their result on input words of length at most n does not exceed $f(n)$.*

Definition 2.1.9. *Let T be a TMD. $\#_\dagger T(w)$ shall denote the number of $\dagger$ symbols in $T(w)$.*

Definition 2.1.10. *$PAR(u)$ shall denote a predicate that is true if the number of non–$\dagger$ symbols in u is even*[5]*.*

We shall now define bounded classes of Display Turing Machines with Control.

Notation 2.1.2.

- *Let $(\mathcal{T}_{f(n)}, \mathcal{L})$ denote a class of Display Turing Machines with Control such that $A = (T, S)$ is in $(\mathcal{T}_{f(n)}, \mathcal{L})$ iff A is in $\mathcal{T}_{f(n)}$ and S is in $\mathcal{L}$ and there is no w in S containing the symbol $\dagger$.*
- *Let $(\mathcal{T}^\dagger_{f(n)}, \mathcal{L})$ denote a class of Display Turing Machines with Control such that $A = (T, S)$ is in $(\mathcal{T}^\dagger_{f(n)}, \mathcal{L})$ iff A is in $\mathcal{T}_{f(n)}$ and S is in $\mathcal{L}$.*
- *Let $\mathcal{C}$ be a class of machines. We shall denote by $\mathcal{L}(\mathcal{C})$ the class of all languages accepted by machines in $\mathcal{C}$ (i.e., L is in $\mathcal{L}(\mathcal{C})$ iff there exists A in $\mathcal{C}$ such that $L(A) = L$).*

We shall sometime refer to the control language S of a $TMDC$ $A = (T, S)$ by $C(A)$.

[5] We shall use this predicate for $T(w)$, the result of a TMD T on an input w, and write $PAR\ T(w)$ instead of $PAR(T(w))$.

Notation 2.1.3. *We shall denote by $\mathcal{TM}$ ($\mathcal{TM}'$) the set of all total (partial) functions computable by standard Turing Machines.*

We shall now define a machine equivalent to the Infinite Time Turing machine with one transfinite step as presented by Hamkins and Lewis [HL00].

Notation 2.1.4. *Let $\mathcal{L}_{RE^*}$ be the class of languages containing possibly infinite words, for which there exists a Deterministic Turing Machine M accepting it. An infinite word w is accepted by a Deterministic Turing Machine M if the machine M working on input w enters its accepting state after finite number of steps.*

Definition 2.1.11. *The* Infinite Time Turing Machine with one transfinite step *is a TMDC A with $C(A) \in \mathcal{L}_{RE^*}$.*

In what follows, we might omit some indices if they are clearly implied by the context. If $P(x)$ is a predicate, we shall denote by P the set of all x satisfying $P(x)$.

Chapter 3

Building hierarchies in Σ_3

3.1 General Properties of Display Turing Machine with Control

We shall now show some basic properties of the Display Turing Machines and Display Turing Machine with Control using very simple control sets and small display tape. In fact in this section we shall only use TMD in $\mathcal{T}_1$, i.e., TMD with display tape size limited to one square. In this way, one can get some intuition of what these machines can do and what are the major factors affecting their computational power.

First of all, we shall show what is the power of 'nice' machines, machines with control set not using the † symbol.

Lemma 3.1.1. $\Sigma_2 \subseteq \mathcal{L}(\mathcal{T}_1, \mathcal{R})$.

Proof. Let L be in Σ_2 and let M be an oracle Turing machine with an oracle for the halting problem of Turing machines accepting L. We shall construct a corresponding $TMDC$ $A = (T, C)$ with display tape consisting of a single square and a regular control language, accepting L. T shall simulate a multi tape machine, which can change the number of its tapes during its computation. This can be achieved by serially writing the contents of all simulated tapes onto the working tape and separating them by the # symbol.

T shall simulate M in an obvious way, as long as there is no oracle query. If M writes something to the oracle tape, then T creates a new tape (for

each query a new tape) and writes the same onto that tape[1]. If M enters the query state, the following happens:

1. T writes its current configuration at the end of a special tape. We shall call it *Query Configurations Stack Tape*, ($QCST$).

2. T writes a unique identifier (e.g., the serial number of this oracle call) at the beginning of the respective oracle tape and also at the end of the $QCST$.

3. T starts a parallel simulation of the machine whose code is written on the appropriate oracle tape.

4. T continues in parallel the computation of M assuming the oracle answer was no (i.e., as if the machine encoded on the oracle tape would not halt).

If the simulated oracle machine M enters a halting state, T shall write the symbol 1 onto the display tape square if M accepts and 0 if it does not. This way, there are numerous tapes used. There is one tape of the original machine M, one $QCST$ and k oracle tapes where k is the total number of oracle calls in the computation of M so far. We can see, that this way, there can be many computations simulated in parallel. Of course, the parallelism is only simulated by T.

Should the simulation of a machine Z written on the oracle tape z halt, then T must roll back its entire computation and free the tape z which is no longer needed. There is no problem with the rollback, for all the necessary information is stored on the $QCST$. All oracle simulations, which started after the simulation of Z, are also to be disposed of. This can also be done without any problem because their identifiers are also written on the $QCST$.

Finally, if the wrong simulation of M already stopped and T has printed an appropriate symbol on the display tape square, it needs to be cleared. If this happens, T shall change the output square to 0.

We define $C = \{1\}$

[1]Note that a query to the Halting Problem oracle consists of an encoding of some Turing machine and its input.

Machine A as described above is in $(\mathcal{T}_1, \mathcal{R})$, since T is a TMD with a single square on a display tape and C is obviously a regular set.

$L \subseteq L(T, C)$:

Let w be in $L(M)$. Then there exists a finite accepting computation of M. This means, that there is a finite number of oracle queries. We shall show, that the simulation of M by T eventually follows the same computation as M does. From the construction of T it is obvious, that there is no problem with those parts of the computations which do not contain an oracle call. So all we have to show is that after some finite time T shall provide the correct answer to each oracle query in the computation of M.

We shall prove this by contradiction. Let k denote the serial number of the first query in the computation of M for which T never generates the correct answer. There are only two reasons why this could happen. The first is, that the query was never asked. This is not possible, since for each of the queries before the k-th one there was a correct answer generated in finite time. Thus the simulation had to reach the point, where the k-th query was asked.

The second reason is that the correct answer to the query was not generated in finite time. If the machine that is the subject of this query ever halts, then the machine T obtains this information after some finite number of steps (since the query simulation is performed in parallel). So to make our assumption true, the query computation must never halt. But in this case, T has the correct answer to the query immediately, since *no* is the default answer to any query. This again is a contradiction.

This implies, that T shall follow the computation of M for some finite time and write the appropriate symbol onto the display square. From that moment on it shall not change the content of it. For, all queries that are still being simulated never halt and thus no rollback shall be done. Thus the sequence of characters written on the single display square converges to 1.

$L(T, C) \subseteq L$

We show, that $w \notin L$ implies $w \notin L(T, C)$.

From the first part of the proof we already know, that for all $p \in \mathbb{N}$ there exists an $q_p \in \mathbb{N}$ such that the first p steps of M are correctly simulated by T in q_p steps. Thus if the computation of M halts after k steps and does not

accept, then after q_k steps T shall print 0 onto the output square and shall not change the content of it anymore.

Let us consider the case, when M does not stop. Then it is obvious from the construction of T, that after simulating exactly l steps of M (needing exactly q_l steps of T) there is 0 or B written on the display square. Let $\{d_{q_k}\}_{k=1}^{\infty}; d_{q_k} \in \{0, B\}$ be the sequence of contents of the output square after q_k steps. This is a subsequence of $D = \{d_i\}_{i=1}^{\infty}$ where d_i is the content of the display square after i steps of T. From that it follows that if D converges, then it must converge to either B or 0. Thus if $w \notin L$ then $w \notin L(T, C)$. $\square$

Lemma 3.1.2. $\mathcal{L}(\mathcal{T}_1, \mathcal{R}) \subseteq \Sigma_2$

Proof. Let $A = (T, C)$ be in $(\mathcal{T}_1, \mathcal{R})$[2]. We shall construct an oracle machine M (with a Σ_1 oracle) accepting w if and only if w is in $L(A)$. All we need to do is to determine the result $T(w)$ of the display machine T in finite time assuming, that the output belongs to C. We shall utilize the fact, that if $T(w) \in C$ then $T(w) \neq \dagger$ (in other words the number of changes of the display is finite). So M shall simulate A and ask the oracle, whether the content of the display square shall be changed during the forthcoming computation (how to ask such questions is shown in the next paragraph). If it does not change, we shall just check if it is in C. If it does, it must happen after some finite time so we can simulate the computation to the point of the change and then ask the oracle again. This implies that if $w \in L(A)$ then $w \in L(M)$. On the other hand if $T(w) = \dagger$ then M never halts. If $T(w) \neq \dagger$ but $T(w)$ is not in $C(A)$ then $w \notin L(M)$ also. Thus if $w \notin L(A)$ then $w \notin L(M)$. Thus $w \in L(A) \iff w \in L(M)$.

To complete the proof we need to show that we can ask queries as described above. The question we are asking is *Given a configuration of the display machine T, shall the content of the display change after some finite number of computational steps?* Formally, let (p, w_0, n_0, w_1, n_1) be a configuration of T. We are asking about the truth of the predicate $P \equiv \exists k \in \mathbb{N} : R(k)$, where $R(k) \equiv ((p, w_0, n_0, w_1, n_1) \vdash_T^k (q, w_0', n_0', w_1', n_1')) \wedge w_1 \neq w_1'$. $R(k)$ is obviously a recursive predicate thus $P \in \Sigma_1$. $\square$

[2]Since the size of the display is limited to 1 only very specific languages from $\mathcal{R}$ are usable.

Theorem 3.1.1. $\mathcal{L}(\mathcal{T}_1, \mathcal{R}) = \Sigma_2$.

Proof. The proof follows from Lemma 3.1.1 and Lemma 3.1.2. □

We see, that the infinite computation of the Display Turing Machine can be used to answer (arbitrary many, but finite) number of queries to the Halting problem of standard Turing machines. It can be seen that if all the words in the control set are †–free it suffices to consider control sets consisting of words of length one. Thus having display tape of size $k, k > 1$, does not increase the power of $TMDC$ in this case. We shall now consider the question whether $TMDC$ can compute more in case the words in the control language can contain the symbol †.

Lemma 3.1.3. $\Pi_2 \subseteq \mathcal{L}(\mathcal{T}_1^{\dagger}, \mathcal{R})$

Proof. Let $L \in \Pi_2$ then $L^C \in \Sigma_2$. Thus there exists an oracle Turing machine M with a Σ_1 oracle accepting L^C. We shall construct a Display Turing Machine with Control $A = (T, \{\dagger\})$ such that $T(w) = \dagger \iff w \notin L(M)$. Let T' be the display machine used in the proof of Lemma 3.1.1. We already know, that $T'(w) = \dagger$ if the computation of M requires infinitely many oracle queries. We shall construct T by a slight modification of T'. The only reason, why T' does not satisfy our needs is that if the computation of M needs only a finite number of queries and then rejects the input, T' does not produce a † on its display. So T shall differ from T' in such case. T shall simulate M just as T' does, but should the simulation come to the point, where M would reject, T shall enter a loop in which it continually rewrites the content of the display square. Of course, the parallel simulation of oracle queries does not stop. This implies, that $w \in L(M) \Rightarrow T(w) = 1$ as in the proof of Lemma 3.1.1 and $w \notin L(M) \Rightarrow T(w) = \dagger \Rightarrow w \in L(A)$. □

We see, that the † symbol is 'stronger' then any non–† symbol. What power does the use of this symbol provide?

Lemma 3.1.4. *Let $A = (T, C)$ be in $(\mathcal{T}_1^{\dagger}, \mathcal{R})$ and let $\dagger \in C$, then $L(A)$ is in Π_2.*

Proof. Let $A = (T, C)$ be in $(\mathcal{T}_1^{\dagger}, \mathcal{R})$ such that $\dagger \in C(A)$. We shall construct an oracle machine M with a Σ_1 oracle such that $L(M) = L(A)^C$. M shall

simulate the computation of T on w and try to determine the display $T(w)$ of T. It shall ask the oracle, whether given a configuration of the display machine T, the content of the display changes after finite number of computational steps. If it does not, M will know the resulting content of the display $\bar{d}_0$ of T and shall accept if $\bar{d}_0 \notin C$ and vice versa. Since $C \in \mathcal{R}$ (and all its elements are of size 1), this will not be a problem.

If the content is to be rewritten, then M can simulate the computation of T to that point and then ask again. If $T(w) = \dagger$ then M never halts and thus does not accept w.

Thus $w \in L(M) \iff T(w) \notin C$. $\square$

Theorem 3.1.2. $\{L | \exists A = (T, C) \in (\mathcal{T}_1^\dagger, \mathcal{R}) : A \textit{ accepts } L \textit{ and } \dagger \in C\} = \Pi_2$

Proof. The theorem is an obvious corollary of Lemma 3.1.3 and Lemma 3.1.4. $\square$

Corollary 3.1.1. $\mathcal{L}(\mathcal{T}_1^\dagger, \mathcal{R}) = \Pi_2 \cup \Sigma_2$

Proof. The proof follows directly from Theorem 3.1.1 and Theorem 3.1.2. $\square$

After these results, we might get the feeling, that in one square, we can compute the answer to one oracle call to a Σ_2 oracle. So if there was a larger display tape, we might be able to compute more. Since we know (see [Bie95]), that for an oracle machine $k+1$ queries to Σ_2 are better then k queries, it would be tempting to prove such conjecture for our model. This shall be the aim of the next section.

3.2 The Tape–size Hierarchy

In this section, we shall examine the relation between oracle machines with a Σ_2 oracle and Display Turing Machines with Control. The main result of this section is the proof, that display tape of fixed size $k+1$ is stronger, then the one of size k for all k. We shall call the hierarchy established in this way the *Tape–size Hierarchy*. Since all machines studied in this section have a constant bound of their display tape, their control sets can be reduced to finite sets. Thus the control sets of all machines in this section are clearly regular.

We shall first formulate a simple but useful "concatenation" property of the Display Turing Machines with Control. For any two machines, we can create a third one by "gluing" these two machines together. Conversely, if we have one display machine using a display tape of size k, we can "separate" it into two machines with display tapes of sizes a and $k-a$ that shall compute their respective parts of the output of the original machine.

Lemma 3.2.1 (Concatenation Lemma).

i. Let (T_1, S_1) and (T_2, S_2) be two TMDC. Then there exists a machine (T, S) such that $L(T, S) = L(T_1, S_1) \cdot L(T_2, S_2)$.

ii. Let T be a TMD. Then there exists two machines T_1 and T_2 such that for all w is $T(w) = T_1(w) \cdot T_2(w)$ and if $|T(w)| > 0$ then $|T_1(w)| = 1$.

Proof.

i. T shall simulate the computation of T_1 and T_2 in parallel and write the output of T_2 right behind the output of T_1. There are many ways how to do this, we shall omit unnecessary technical details. $S = S_1 \cdot S_2$. It is obvious, that (T, S) satisfies our demands.

ii. T_1 shall simulate in an obvious way T by using only its first tape[3]. After each computational step of T, T_1 shall copy the content of the first square of T's display tape onto its own display tape. Thus the resulting content of the display tape of T_1 is the content of the first square of the display tape of T. T_2 shall work as T_1 with the difference, that it shall copy onto its display tape the content of the whole display tape of T except its first square. Thus for each w is $T(w) = T_1(w) \cdot T_2(w)$ and if $|T(w)| > 0$ then $|T_1(w)| = 1$.

□

We might refer to T by writing $T_1 \cdot T_2$.

We shall continue by establishing several technical Lemmas, which we shall use in the proofs that will follow.

[3]This can be done in many ways, for example by serially writing the contents of both tapes of T and separating them with some special symbol.

Lemma 3.2.2. *Let T be a TMD with display tape fixed to the size k. Let $LE_\dagger(T(w), n)$ denote a predicate which is true if and only if $\#_\dagger T(w) \leq n$. Then for all $n \leq k$ is the set $S_n^T = \{w | LE_\dagger(T(w), n)\}$ in Σ_2.*

Proof. Let T and n be given. Clearly a lower bound for the number of non–† symbols in $T(w)$, implies an upper bound for $\#_\dagger T(w)$, namely k minus this lower bound. Thus the set of all words w for which $T(w)$ contains $k - n$ or more non–† symbols is the same set as S_n^T. Formally

$$\{w | k - \#_\dagger T(w) \geq k - n\} = \{w | \#_\dagger T(w) \leq n\}$$

Let T_i be a TMD that writes only one symbol onto its display tape, namely the i-th symbol of $T(w)$. It can be easily seen that there is such a machine. Let S be the set of all output symbols of T except for †. Then $A_i = (T_i, S)$ is a $TMDC$ accepting those w for which the i-th letter of $T(w)$ is a non–† symbol. Since S_n^T is the set of all words whose output contain at least $k - n$ non–† symbols, the following equation holds:

$$S_n^T = \bigcup_{\substack{M \subseteq \{1,\ldots,k\} \\ |M| = k-n}} \bigcap_{i \in M} L(A_i)$$

From Theorem 3.1.1 we know, that $L(A_i) \in \Sigma_2$ for all i. Since Σ_2 is closed under union and intersection, S_n^T is in Σ_2. □

Now, we can show how to simulate a Display Turing Machine with Control set restricted by a fixed tape size by an oracle machine using only logarithmically[4] many queries.

Lemma 3.2.3. $\mathcal{L}(\mathcal{T}_k^\dagger, \mathcal{R}) \subseteq \{L \in \Sigma_3 | \exists$ *oracle machine M with a Σ_2 oracle such that $L(M) = L$ and M needs at most $\lceil \lg(k+1) \rceil + 1$ oracle calls*$\}$.

Proof. Let $L \in \mathcal{L}(\mathcal{T}_k^\dagger, \mathcal{R})$ and let $A = (T, C)$ be the $TMDC$ machine accepting it. We shall construct an oracle machine M with the desired properties.

Given w on input, M will work in the following way. At first, M shall compute $\#_\dagger T(w)$ the exact number of † written on the display of $T(w)$. We shall show, that M will not need more then $\lceil \lg(k+1) \rceil$ oracle queries

[4] We use the term $\lg x$ instead of $\log_2 x$.

to achieve this. Then M shall use only one more query to compute the acceptance of A.

Since $LE_{\dagger}(T(w), n)$ can be computed using only one query (due to Lemma 3.2.2). M can determine the value of $\#_{\dagger}T(w)$ by performing a binary search on the set $\{0, 1, \ldots, k\}$. This shall obviously take no more than $\lceil \lg(k+1) \rceil$ oracle queries.

Now, M should find the display $T(w)$ of T and whether it is in C. We shall show that this can by achieved by using only one query. M shall ask the query, whether there exists a number t, that after t steps of T, all convergent display squares have come to the point that they will not be altered in the forthcoming computation. Furthermore, M will determine whether the word obtained by replacing the non convergent squares by † symbols is in C. We can write this in a formal way.

Let $S = \{i_1, i_2, \ldots, i_{k-\#_{\dagger}T(w)+1}\}$ denote a set of all indices of squares of the display tape, that contain a non–† symbol in $T(w)$. Let

$$P(v_1, v_2, \cdots, v_m, i_1, i_2, \cdots, i_m)$$

denote a word of length k such, that its i_l-th symbol is v_l and all other symbols (whose indices are not in the list) are †. Let t denote the time, by which all the convergent squares of the display contain the correct result that will not change any more.

Thus given S and t, the question we want to ask is whether[5]

$$P\left(d_{i_1}(t), d_{i_2}(t), \cdots, d_{i_{k-\#_{\dagger}T(w)+1}}(t), i_1, i_2, \cdots, i_{k-\#_{\dagger}T(w)+1}\right) \in C \qquad (3.1)$$

To verify that a given t has all the properties laid on it, the following predicate must be true

$$(\forall m \in \mathbb{N}) : (i \in S \Rightarrow d_i(t) = d_i(t+m)) \qquad (3.2)$$

[5] $d_i(t)$ denotes the content of the i-th square of the display tape after t computational steps.

Thus the question M should ask is

$$(\exists S)(\exists t \in \mathbb{N}) : ((3.2) \wedge (3.1)) .$$

Since we know the size and domain of S, we can compact the two quantifiers to one by encoding the variables $i_1, i_2, \cdots, t$ into one integer t'. Then the question M should ask is whether there is such an integer t', that encodes an correct list of values $i_1, i_2, \cdots, t$. We shall use coding into composite numbers, where exponents of primes represent the encoded integers. Let $p_i; i \in \mathbb{N}$ denote the i-th prime ($p_0 = 2$).

To verify if t' is the code of an list of appropriate length, the following predicate must be true

$$\left(t' = p_0^t \cdot \prod_{i=1}^{k-\#_\dagger T(w)+1} p_i^{\alpha_i} \right) \wedge \left(\bigwedge_{i=1}^{k-\#_\dagger T(w)+1} \alpha_i \neq 0 \right) . \tag{3.3}$$

Here, the zeroth exponent represents t and all the successive exponents represent S.

Thus the question M needs to ask could be written as

$$(\exists t' \in \mathbb{N})((3.3) \wedge (3.2) \wedge (3.1))$$

By expanding each of the terms 3.1, 3.2, 3.3 and placing the universal quantifier to the front we obtain the predicate

$$\begin{aligned}
&(\exists t' \in \mathbb{N})(\forall m \in \mathbb{N}) : \\
&\left(\left(t' = p_0^t \cdot \prod_{i=1}^{k-\#_\dagger T(w)+1} p_i^{\alpha_i} \right) \wedge \left(\bigwedge_{i=1}^{k-\#_\dagger T(w)+1} \alpha_i \neq 0 \right) \wedge \left(\bigwedge_{i=1}^{k-\#_\dagger T(w)+1} d_{\alpha_i}(t) = \right. \right. \\
&\left. \left. = d_{\alpha_i}(t+m) \right) \wedge P\left(d_{\alpha_1}(t), d_{\alpha_2}(t), \cdots, \alpha_1, \alpha_2, \cdots, \alpha_{k-\#_\dagger T(w)+1} \right) \in C(A) \right) .
\end{aligned}$$

Since the body of this predicate is recursive, the whole predicate refers to a set in Σ_2 and thus, we need only one oracle query to obtain the answer to our final question. $\square$

In the previous proof, we used the coding of lists of numbers into exponents of primes. Of course, we could have used any other numbering function.

We shall now show how to simulate an oracle machine that uses at most k oracle calls by using an exponentially long tape. This is somehow to be expected given the previous Lemma.

Lemma 3.2.4. *Let M be an oracle machine that is allowed to make at most k oracle calls (to a Σ_2 oracle). Then $L(M) \in \mathcal{L}(\mathcal{T}^{\dagger}_{2^{k+1}-1}, \mathcal{R})$*

Proof. We shall find a $TMDC$ $A = (T, C)$ simulating M. The proof is based on the following idea. Let O be the oracle used and w its input. We have seen (Corollary 3.1.1), that an TMD is able to compute the answer to an Σ_2 query by using only one square of the display tape. Since we know, that the number of queries is not greater then k, T can enumerate all possible computations of M on w with an arbitrary oracle. These computations shall form a binary tree and at each branching point, there shall be an oracle query. We shall call this tree a computation tree. Since the number of queries to O is bounded by k, the tree has at most $2^k - 1$ branching points. All that we need to do in order to ensure that we can correctly simulate M with O is to compute the answer to each query in the computation tree of M on w. We shall also compute the result of M for each possible path of the computation. Since there are 2^k leaves, there are 2^k possible computational paths (after the last query, there can still be some computation). To write all this information (the result of each query and the result of each computational path), T needs a display tape of length $2^{k+1} - 1$.

Having this information, A can easily determine the true computation of M on w with O as its oracle[6].

We can effectively number each node in the computation tree. Let T_i compute the output of the query with number i and print 1 if the answer is positive and † if it is negative. The existence of such T_i is guaranteed by Corollary 3.1.1. Since each of the 2^k possible computational paths has a fixed unique combination of k query answers, there is no problem to simulate the

[6]The results of the oracle queries (starting from the first query) form the true computational path of M. The required information is retrievable, since there is the result of each possible computational path written on the display tape.

result of each such computation by a standard Turing machine. Let T'_i be a display Turing machine which simulates the i-th computational path and prints the symbol 1 on the i-th display square if the computation accepts.

By the use of the Concatenation Lemma (Lemma 3.2.1), we can construct a TMD such that the content of the i-th output square is the output of T_i if $i < 2^k$ or the output of T'_{i-2^k+1} otherwise[7].

Let C be the set of words of length $2^{k+1}-1$ that are codes of computational trees as described above, for which the correct computational path accepts. Since this set is finite, it is surely in $\mathcal{R}$.

Thus (T, C) is a $TMDC$ simulating M. $\square$

The results presented so far give us an idea of how strong the Display Turing Machine with Control with constant size of the display tape are. By using Lemma 3.2.4 we can see that with a display tape of size $2^{k+1} - 1$ we could compute more, then we could with a $2^k - 1$ squares long tape. But is it true that if we increased the maximum size of the display tape just by one square, we would obtain greater computational power? We shall show, that there is an affirmative answer to this question.

In fact, our computational model with fixed size display tape is similar to the model of a Turing machine making parallel queries presented by Richard Biegel in [Bie95] [BGH89]. So we shall try to use similar proof techniques to achieve our goal.

We shall start by examining the properties of the predicate $PAR\ T(w)$ which is true if the number of non–† symbols in $T(w)$ is even.

Lemma 3.2.5. *Let* $(T, S) \in (\mathcal{T}_k^\dagger, \mathcal{R})$. *Then* $L = \{w | PAR\ T(w)\} \in \mathcal{L}(\mathcal{T}_k^\dagger, \mathcal{R})$.

Proof. Let $A = (T, S')$ where $S' = \{v | \#\text{non–† symbols in } v \text{ is even}\}$. Obviously $L(A) = L$. $\square$

Lemma 3.2.6. *If* $(\forall m \in \mathbb{N})(\exists n \in \mathbb{N})(\exists T \in \mathcal{T}_n) : (PAR\ T \notin \mathcal{L}(\mathcal{T}_m^\dagger, \mathcal{R}))$ *then* $(\forall i \in \mathbb{N}) : \mathcal{L}(\mathcal{T}_i^\dagger, \mathcal{R}) \subsetneq \mathcal{L}(\mathcal{T}_{i+1}^\dagger, \mathcal{R})$[8].

[7] Thus the content of the 2^k-th square shall be the result of T'_1

[8] By $PAR\ T$ we mean the language of all words w, for which $PAR\ T(w)$ is true.

Proof. Let n' be the maximum, for which all the machines in $\mathcal{T}'_n$ have their PAR language acceptable by a machine from $\mathcal{L}(\mathcal{T}^\dagger_m, \mathcal{R})$. By the assumption, such a n' surely exists for each $m \in \mathbb{N}$. Formally let

$$n' = \max\{n | (\forall T \in \mathcal{T}_n)(PAR\ T \in \mathcal{L}(\mathcal{T}^\dagger_m, \mathcal{R}))\}.$$

Thus there exists a $T' \in \mathcal{T}_{n'+1}$ such, that

$$PAR\ T' \notin \mathcal{L}(\mathcal{T}^\dagger_m, \mathcal{R}).$$

Using the Concatenation Lemma (Lemma 3.2.1) we know, that there are machines T_n and T_1 such that $T' = T_n \cdot T_1$ where T_n has its display tape of size n and T_1 has its display tape of size 1. Since

$$PAR\ T_n \in \mathcal{L}(\mathcal{T}^\dagger_m, \mathcal{R}) \wedge PAR\ T_1 \in \mathcal{L}(\mathcal{T}^\dagger_1, \mathcal{R})$$

we can find a machine in $(\mathcal{T}^\dagger_{m+1}, \mathcal{R})$ accepting $PAR\ T'$. If (T_0, S) is the machine accepting $PAR\ T_n$ then we shall construct the machine A for accepting $PAR\ T'$ as follows:

$$A = (T_0 \cdot T1, S'); S' = \{w' | \, (w' = wv) \wedge ((w \in S \wedge v = \dagger) \vee (w \notin S \wedge v \in \Gamma_{T_1}))\}.$$

Thus, $PAR\ T' \notin \mathcal{L}(\mathcal{T}^\dagger_m, \mathcal{R})$ and $PAR\ T' \in \mathcal{L}(\mathcal{T}^\dagger_{m+1}, \mathcal{R})$. If there exists a T' for each m, then $\mathcal{L}(\mathcal{T}^\dagger_a, \mathcal{R}) \subsetneq \mathcal{L}(\mathcal{T}^\dagger_{a+1}, \mathcal{R})$ for all a. □

We shall now show, that the assumption of the previous Lemma is true and thus is its consequence true as well.

Lemma 3.2.7. *Let $n \in \mathbb{N}$ and let $bin_i(n)$ denote the i-th bit in the binary encoding of n. Then $bin_i(n) = bin_0\binom{n}{2^i}$.*

We shall not provide proof of this Lemma, since it is a well known combinatorial property.

Lemma 3.2.8. *If $(\forall T \in \mathcal{T})(PAR\ T \in \mathcal{L}(\mathcal{T}^\dagger_m, \mathcal{R}))$ then $(\forall T \in \mathcal{T}_k)(\exists (T', f) \in \mathcal{L}(\mathcal{T}_{m \cdot \lceil \lg k \rceil}, \mathcal{TM})$ computing $\#_\dagger T)$.*

Proof. T' shall use $PAR\ T$ to compute $\#_\dagger T(w)$ bit by bit. Obviously if $PAR\ T(w)$ is true, then the last bit of $\#_\dagger T(w)$ is 1. (Since the parity of † symbols is $1 - PAR\ T(w)$). Let n denote the number of non-† symbols in $T(w)$. Obviously

$$n = k - \#_\dagger T\,(w)\,.$$

Let T_i denote the display machine computing the content of the i-th output square of T. Let S be a subset of $\{1, 2, \cdots, k\}$. Let T_S be a display machine returning 1 if $(\forall i \in S)(T_i(w) \neq \dagger$ and returning † otherwise. (Such a display machine obviously exists).

We shall prove, that if $\mathcal{S}_{2^j}$ is the set of all 2^j element subsets of $\{1, \cdots, k\}$ and $\mathbb{T}_{2^j}$ is the machine

$$\mathbb{T}_{2^j} = \bigodot_{S \in \mathcal{S}_{2^j}} T_S$$

(where $\bigodot$ denotes the concatenation operator) then

$$PAR\ \mathbb{T}_{2^j}(w) = bin_j\,(n)\,.$$

Let $S \in \mathcal{S}_{2^j}$. If there is an i in S such that $T_i(w)$ returns the symbol † then $T_S(w)$ returns † as well. On the other hand, if there is no such i in S, then $T_S(w)$ returns the symbol 1. Let n denote the number of non-† symbols in $T(w)$. We shall now compute the number of sets in $\mathcal{S}$ for which $T_S(w) \neq \dagger$. Since the size of each S is 2^j, there are $\binom{n}{2^j}$ such sets. By Lemma 3.2.7 we can see, that

$$PAR\ \mathbb{T}_{2^j}(w) = bin_0\binom{n}{2^j} = bin_j(n).$$

This way, (T', f) can compute n in the following way.

Let $PAR\ \mathbb{T}_l = (T_{PAR\ \mathbb{T}_l}, C)$. Let us presume, that $T_{PAR\ \mathbb{T}_l}$ always produces an output of size m (we can achieve this by adding necessary padding and appropriately changing the set C). Let J_k be the set of all powers of 2 smaller or equal to k. Then

$$T' = \bigodot_{l \in J_k} (T_{PAR\ \mathbb{T}_l})\,.$$

Since each PAR needs only m squares, T' needs $m \cdot \lceil \lg k \rceil$ squares. All that

needs to be done now is the calculation of n and then $\#_\dagger T(w)$. This is the task for f. Since A is a finite set and the size of output of each PAR is normalized to m, f can transform each block of length m to a 0 or 1 depending on its correspondence to A and thus compute n. Since k is fixed, $\#_\dagger T(w)$ can be computed as

$$\#_\dagger T(w) = k - n.$$

Since T and k are fixed, there can always exist a (T', f) as described above. □

Lemma 3.2.9. $(\forall m \in \mathbb{N})(\exists n \in \mathbb{N})(\exists T \in \mathcal{T}_n) : (PAR\ T \notin \mathcal{L}(\mathcal{T}_m^\dagger, \mathcal{R}))$

Proof. By contradiction.
Let $(\exists m \in \mathbb{N})(\forall n \in \mathbb{N})(\forall T \in \mathcal{T}_n) : (PAR\ T \in \mathcal{L}(\mathcal{T}_m^\dagger, \mathcal{R}))$ be true. Then by Lemma 3.2.8 for each $T \in \mathcal{T}_k$ there is a $M = (T', f) \in (\mathcal{T}_{m\cdot\lceil \lg k\rceil}, \mathcal{TM})$ computing $\#_\dagger T$. We shall construct an oracle machine M' simulating $A = (T, C)$ with only $\lceil \lg(m\cdot\lceil \lg k\rceil + 1)\rceil + 1$ oracle queries. There is a big similarity between the computation of M' and the machine used in Lemma 3.2.3. The main difference shall be, that M' shall use T' to compute $\#_\dagger T$.

M' shall at first compute the output of T'. To do this M' shall first enumerate the number of † symbols in the output of $T'(w)$. Since

$$|T'(w)| = m \cdot \lceil \lg k\rceil$$

this can be done by

$$\lceil \lg(m \cdot \lceil \lg k\rceil + 1)\rceil$$

queries. Now M' shall compute the output of T', calculate $\#_\dagger T$ and then simulate the output of T. This computation can be realized by on oracle machine M'' using a Σ_1 oracle. The result of M'' can be obtained by one query of M'. The computation of M'' is equivalent to this predicate[9] (we use similar notation as in Lemma 3.2.3), where $m \cdot \lceil \lg k\rceil - \#_\dagger T'(w) + 1$ is denoted $\#$ and $f\left(P\left(d_{\alpha_1}^{T'}(t), d_{\alpha_2}^{T'}(t), \cdots, \alpha_1, \alpha_2, \cdots, \alpha_\#\right)\right)$ is denoted $f_{1:\#}^{T'}$:

[9] Commentary is provided below.

$$(\exists t' \in \mathbb{N})(\forall l \in \mathbb{N}) : \Bigg(\Bigg(t' = p_0^t \cdot \prod_{i=1}^{\#+f_{1:\#}^{T'}} p_i^{\alpha_i} \Bigg) \wedge \Bigg(\bigwedge_{i=1}^{\#+f_{1:\#}^{T'}} \alpha_i \neq 0 \Bigg) \wedge$$
$$\Bigg(\bigwedge_{i=1}^{\#} d_{\alpha_i}^{T'}(t) = d_{\alpha_i}^{T'}(t+l) \Bigg) \wedge \Bigg(\bigwedge_{i=\#+1}^{\#+k-f_{1:\#}^{T'}} d_{\alpha_i}^{T}(t) = d_{\alpha_i}^{T}(t+l) \Bigg) \wedge$$
$$P\left(d_{\alpha_{\#+1}}^{T}(t), d_{\alpha_{\#+2}}^{T}(t), \cdots, \alpha_{\#+1}, \alpha_{\#+2}, \cdots, \alpha_{\#+k-f_{1:\#}^{T'}} \right) \in C \Bigg).$$

This predicate differs from the one in Lemma 3.2.3 in that we also need to "compute" the value of $\#_\dagger T$ and then use it. Thus the number t' is not only the coding of t and the appropriate indices of non–† symbols in $T(w)$, but also the indices of non–† symbols $T'(w)$.

The predicate (3.2) is enriched by conditions checking, if the indices of non–† symbols in $T'(w)$ are correct.

The actual value of $\#_\dagger T$ is computed by

$$f\left(P\left(d_{\alpha_1}^{T'}(t), d_{\alpha_2}^{T'}(t), \cdots, \alpha_1, \alpha_2, \cdots, \alpha_{\#} \right) \right)$$

and denoted $f_{1:\#}^{T'}$.

Since there is only one set of indices of all non–† symbols in $T'(w)$, there is also only one result of $f(\cdots)$ and thus the predicate is true only if w is in C.

Let M_0 be a Turing machine with a Σ_2 oracle such that M_0 needs 2^i oracle calls to accept $L(M_0)$ and that there is no Σ_2 oracle machine M_1 accepting $L(M_0)$ with less then 2^i oracle calls. Due to results in [BGH89] there is such an M_0 for each i. Then by Lemma 3.2.4

$$L(M_0) \in \mathcal{L}(\mathcal{T}^{\dagger}_{2^{2^l+1}-1}, \mathcal{R}).$$

But if this is true, then by the previous construction $L(M_0)$ can be computed by making only

$$\lceil \lg(m \cdot \lceil \lg 2^{2^i+1} - 1 \rceil + 1) \rceil \leq \lceil \lg(m \cdot 2^i + 1 + 1) \rceil \leq \lceil \lg m + i + 1 \rceil \leq m + i + 1$$

queries. Since m is a constant, there exists an i big enough to make the following inequality hold

$$m + i + 1 < 2^i.$$

But this is a contradiction, since there cannot exist a machine M_1 accepting $L(M_0)$ by making only $m + i + 1$ queries. □

Theorem 3.2.1. $k < l \Rightarrow \mathcal{L}(\mathcal{T}_k^\dagger, \mathcal{L}) \subsetneq \mathcal{L}(\mathcal{T}_l^\dagger, \mathcal{L})$

Proof. The proof is a straightforward consequence of Lemma 3.2.6 and Lemma 3.2.9. □

We can see, that there is an infinite hierarchy of machines with constant display tape sizes. In the next section, we shall let the tape size more lose, but shall stress the conditions laid on the Control language. This way we shall form a new and different hierarchy.

3.3 The Extended Chomsky Hierarchy

In this section we shall examine the impact on $TMDC$ computational power by placing different constrains on the Control language. We shall not only work with Control languages, but also (and most by) with machines accepting these languages. We shall refer to these machines as Control Machines.

It is obvious, that if do not put a bound on neither of the machines (the display machine or the control machine) we can get unlimited computational power (we could accept any language just by using that language as the Control language and using a display machine, that only rewrites the input onto the output). In the following study, we shall also put a 'small' bound on the display machine. We shall demand it to always produce an output of finite size.

Notation 3.3.1. *By $\mathcal{T}_{<\omega}^\dagger$ we denote the set of all TMD having for each v on input an output on the display tape of finite size.*

We shall show (for some Control sets), that the stronger the Control is, the more languages we can accept. In our proofs, we shall use the diagonalization argument.

Lemma 3.3.1 (diagonalization). *Let $\mathcal{C}$ be a class of machines and let there exist a code for each machine from this class. Then no machine in $\mathcal{C}$ can accept the language consisting of codes of machines rejecting their own codes. We shall denote this language by $D_\mathcal{C}$ and call it the diagonal language for $\mathcal{C}$.*

Proof. By contradiction.
Let A be such a machine. Does A accept its own code? If this was the case, then the code of A could not be in $L(A)$ since this is the language of all codes that are rejected by machines they represent. Thus A cannot accept it own code.

But if A does not accept its own code, then the code of A must be in $L(A)$, since this is the language of all codes that are rejected by machines they represent. Thus A must accept its own code. Since this is a contradiction, there cannot be such A. □

We shall now start to examine the relations between particular Control classes by showing, that regular Controls are weaker then context–free Controls. This result shall be achieved by providing a machine using a context–free Control accepting $D_{\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega},\mathcal{R})}$.

Lemma 3.3.2. $D_{\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega},\mathcal{R})} \in \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{CF})$

Proof. Let A be a finite automaton with its code $< A >$ and let T be a TMD with code $< T >$. Thus $M = (T, L(A)) \in (\mathcal{T}^{\dagger}_{<\omega}, \mathcal{R})$ and $< T >< A >$ is a proper code of M. The machine $M_D = (T_D, L(A_D))$ (where A_D is a push–down automaton) accepting the diagonal language shall work in the following way. Given $< T >< A >$ on input, T_D shall output the output of T concatenated with all words of size $T(< T >< A >)$ (we shall denote this number by P) and for each such word there shall be the information, if A accepts it. For better understanding look at the following figure:

$T(< T >< A >)$	w_1^R	$A(w_1)$	w_2^R	$A(w_2)$	$\cdots$	w_P^R	$A(w_P)$

Then A_D shall read the output $T(< T >< A >)$ into its buffer and continue to move along the display tape. If A_D should read $(< T >< A >)^R$ (A_D can find this out using non-determinism) then it shall compare it letter by letter with the content of its buffer. After verifying, that the block read was really $(< T >< A >)^R$ A_D reads the next letter. It is the symbol 1 if $< T >< A >\in L(A)$ and 0 otherwise. If it was the symbol 0, A_D enters an accepting state, reads the remaining part of the display tape and accepts. If

the symbol read by A_D was 1, then the automaton enters a non accepting state, reads the remaining content of the display tape and rejects.

Thus we created a machine $T_D \in (\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{CF})$ accepting D.

Some technical notes: The codes of A and T can be understood as codes of the respective Turing machine or the display machine. The computation of T_D proceeds in the following way. If T should come to the point, that it needs more display tape (i.e., it would be rewriting a B symbol on its display tape), T_D can rewrite the contents of its display tape following after the output of T by all possible words of the required length. Since the output of T is always finite, there are no problems with this and no unwanted † symbols will be created.

Once all the words of the appropriate size are written, T may simulate A on their reverses and print the output of each such simulation after the respective word.

After all this has been done, T_D may continue in the simulation of T. Since the output of T is finite the output of T_D shall be also finite, thus $T_D \in \mathcal{T}_{<\omega}$.

Some more technical notes: Since T_D cannot create a † symbol in finite time, it would seem impossible to both generate all possible outputs of T and then simulate A on them. Although this can be done by using two squares of T_D's display tape to encode one square of the display tape of T. For example, we might code a square containing letter X by a pair of squares XX. This way, we can also code the † symbol by a pair of letters not used so far. Naturally, T_D shall simulate A by treating two letters as one.

To avoid ambiguity, T_D shall use delimiters on its display tape to separate the output of T, all its possible outputs and results of each machine. We can use another so far unused pair of letters to represent the delimiter. This also means, that A_D must work with pairs of letters and interpret them correctly, but this is obviously no problem.

Thus $M_D = (T_D, L(A_D))$ is a machine in $\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{CF})$ accepting D. □

Now, we can proof our firs result.

Theorem 3.3.1. $\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{R}) \subsetneq \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{CF})$

Proof. The fact that $\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{R})$ is a subset of $\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{CF})$ is obvious. If it was true, that

$$\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{R}) = \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{CF})$$

then by Lemma 3.3.2 $D \in \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{R})$. But this cannot be true due to the digitalization argument. □

Lemma 3.3.3. $D_{\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{CF})} \in \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{ECS})$

Proof. Let the code of a machine $M = (T, L(A)) \in (\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{CF})$ be the string $< T >< A >$ where $< T >$ is the code of the display machine T and $< A >$ is the code for the push-down automaton accepting $L(A)$. Then the machine $M_D = (T_D, L(A_D)) \in (\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{ECS})$ for accepting the diagonal language operates the same way as in the previous proof, only the reverses are not necessary any more and T_D simulates the computation of a push–down automaton. □

Theorem 3.3.2. $\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{CF}) \subsetneq \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{ECS})$

Proof. The fact that $\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{CF})$ is a subset of $\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{ECS})$ is obvious. If it was true, that

$$\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{CF}) = \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{ECS})$$

then by Lemma 3.3.3 $D \in \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{CF})$. But this cannot be true due to the digitalization argument. □

Lemma 3.3.4. $D_{\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{ECS})} \in \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{REC})$

Proof. Since $\mathcal{L}_{ECS} \subset \mathcal{L}_{REC}$, for each linearly bonded automaton (LBA) there exists a Turing machine halting on all its inputs accepting the same language. Let the code of a machine $M = (T, L(A)) \in (\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{ECS})$ be the string $< T >< A >$ where $< T >$ is the code of the display machine T and $< A >$ is the code for always halting Turing machine accepting $L(A)$. Then the machine $M_D = (T_D, L(A_D)) \in (\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{REC})$ works in the following way.

The output of T_D shall be $< A >$ followed by a delimiter and the output of T. Since there exists a universal Turing machine U, A_D shall simulate this machine on $< A >$. Since $< A >$ always halts, the computation of U halts with the same result. A_D halts with accepting if A rejected and vice versa. □

Theorem 3.3.3. $\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{ECS}) \subsetneq \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{REC})$

Proof. The fact that $\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{ECS})$ is a subset of $\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{REC})$ is obvious. If it was true, that

$$\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{ECS}) = \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{REC})$$

then by Lemma 3.3.4 $D \in \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{ECS})$. But this cannot be true due to the digitalization argument. □

Lemma 3.3.5. $D_{\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{REC})} \in \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{RE})$

Proof. Let the code of a machine $M = (T, L(A)) \in (\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{REC})$ be the string $< T >< A >$ where $< T >$ is the code of the display machine T and $< A >$ is the code for always halting Turing machine accepting $L(A)$. Then the machine $M_D = (T_D, L(A_D)) \in (\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{RE})$ works in the following way.

The output of T_D shall be $< A >$ followed by a delimiter and the output of T. Since there exists a universal Turing machine U, A_D shall simulate this machine on $< A >$. Since $< A >$ always halts, the computation of U halts with the same result. A_D halts with accepting if A rejected and vice versa. □

Theorem 3.3.4. $\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{REC}) \subsetneq \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{RE})$

Proof. The fact that $\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{REC})$ is a subset of $\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{RE})$ is obvious. If it was true, that

$$\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{REC}) = \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{RE})$$

then by Lemma 3.3.5 $D \in \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{REC})$. But this cannot be true due to the digitalization argument. □

Thus, we have proved the existence of a Chomsky like hierarchy. We shall call it the Extended Chomsky hierarchy.

$$\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{R}) \subsetneq \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{CF}) \subsetneq \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{ECS}) \subsetneq \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{REC}) \subsetneq \mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{RE})$$

Chapter 4

Conclusion

To conclude our study we shall prove, that by increasing the power of the Control as shown in the previous section, the models remain in the domain of the Σ_3 level of the arithmetical hierarchy. In fact, they are all in Δ_3.

Theorem 4.0.5. $\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{RE}) \subset \Sigma_3$

Proof. Let $A = (T, C)$ be a machine in $(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{RE})$. As we have seen in the first section, an oracle machine M with a Σ_2 oracle needs only one query to determine, if the content of the i-th output square of T is the † symbol. If T is working over the alphabet Γ it can be easily seen, that M needs at most $|\Gamma| + 1$ queries to determine the exact output of the i-th query (by simply asking, for each symbol in the alphabet). Since $T \in \mathcal{T}_{<\omega}$, the output $T(w)$ has a finite size for each input. Thus, the machine M needs at most

$$|T(w)| \cdot |\Gamma| + 1$$

queries to determine the output of T.

Since there is a standard Turing machine accepting C, M needs only one more query to find out, if $T(w)$ is in C. This implies, that M is an always halting machine, thus $\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{RE}) \subset \Delta_3$. □

We can now summarize all our results. In Section 2, we showed that there is a Tape–size hierarchy. Since all the control languages in the tape size hierarchy are regular, one can easily see that all classes of this hierarchy are in

the lowest level of the Extended Chomsky hierarchy. So by combining all our results we can have a more precise insight to the Σ_3 level of the Arithmetical hierarchy. Its structure is shown in the following figure:

Σ_3
Δ_3
$\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{RE})$
$\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{REC})$
$\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{ECS})$
$\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{CF})$
$\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{R})$
$\vdots$
$\mathcal{L}(\mathcal{T}^{\dagger}_{3}, \mathcal{R})$
$\mathcal{L}(\mathcal{T}^{\dagger}_{2}, \mathcal{R})$
$\Pi_2 \cup \Sigma_2 = \mathcal{L}(\mathcal{T}^{\dagger}_{1}, \mathcal{R})$
$\Sigma_2 = \mathcal{L}(\mathcal{T}_{1}, \mathcal{R})$

Our work shows, that one could continue our studies by examining the properties of more machines coupled together, i.e., using one $TMDC$ as control in another $TMDC$. On the other hand, it would be tempting to show, that each degree of the arithmetical hierarchy contains its own Extended–Chomsky hierarchy.

Bibliography

[BGH89] Richard Biegel, Williaam I. Gasarch, and Louise Hay. Bounded query classes and the difference hierarchy. *Archive for Mathematical Logic*, 29(2), 1989.

[Bie95] Richard Biegel. Query-limited reducibilities. *Dissertation at Stanford University*, 1995.

[EN02] G. Etesi and I. Németi. Non-turing computations via malament-hogarth space-times. *Int. J. Theor. Phys*, 41, 2002. see also: http://arXiv.org/abs/gr-qc/0104023.

[HL00] J.D. Hamkins and A. Lewis. Infinite time turing machines. *Journal of Symbolic Logic*, 65(2), 2000.

[HR92] Jr. Hartley Rogers. *Theory of Recursive Functions and Effective Computability*. The MIT Press, 1992.

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

[Ord02] Toby Ord. Hypercomputation: computing more then the turing machine. *Honours Thesis, University of Melbourne*, 2002. see also: http://arxiv.org/pdf/math.LO/0209332.

[SS92] H. Siegelmann and E. Sontag. Neural networks with real weights: Analog computational complexity. Technical Report SYCON-92-05, Rutgers Center for Systems and Control, Rutgers University, 1992.

[SS95] Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, 1995.

[WvL01] Jiří Wiedermann and Jan van Leeuwen. Beyond the turing limit: Evolving living systems. 2001.

[WvL02] Jiří Wiedermann and Jan van Leeuwen. Relativistic computers and non-uniform complexity theory. 2002.

Abstrakt

V posledných rokoch sa na scéne teoretickej informatiky objavili viaceré výpočtové modely, ktorých výpočtová sila prekonáva silu Turingových strojov. Tieto modely nesú spoločný anglický názov *Hypermachines* alebo *Super Turing machines*.

V úvode práce stručne predstavíme niektoré takéto modely, pokúsime sa ich voľne porovnať a poukázať na rozdiely ako aj spoločné črty. V ďaľšej časti sa zameriame na nekonečne dlhé výpočty. Predstavíme vlastný model, ktorý nám umožní ich podrobné skúmanie. Náš model sa skladá z časti realizujúcej nekonečný výpočet a z časti spracuvajúcej výsledok tohto výpočtu.

V práci skúmame vplyv veľkosti informácie získanej nekonečným výpočtom na silu nášho modelu. Tiež sa zapodievame otázkou vplyvu zložitosti výstupov nekonečného výpočtu na výpočtovú silu modelu.

Ukážeme, že nami zadefinované stroje (resp. jazyky nimi určené) s vyššie spomenutými obmedzeniami (obmedzenia veľkosti, alebo zložitosti informácie získanej nekonečným výpočtom) tvoria hierarchiu, ktorá je vnorená do stupňa Σ_3 aritmetickej hierarchie. Ukážeme, že najspodnejšie poschodie tejto novovzniknutej hierarchie je totožné s triedou Σ_2 a že štruktúra horných piatich poschodí pripomína Chomského hierarchiu.

Kľúčové slová: Super Turing computation, aritmetická hierarchia, nekonečné výpočty, Chomského hierarchia.