Computing Exact Solutions of Initial Value Problems

Niels van der Weide
Supervisor: Herman Geuvers

Key words and phrases. Exact Real Arithmetic, Computable Analysis, Domain Theory, Initial Value Problems

ABSTRACT. Finding the exact solution of initial value problems is the main topic of this thesis. Because differential equations in practice can be hard to solve, we require algorithms to solve them on a computer. Since the solution of the equation needs to be stored on a computer, we first discuss the representation of real functions. We show that approximating a function by polynomials gives a suitable representation, and we prove that this representation is admissible. Finally, we discuss an algorithm described by Edalat, Lieutier and Krznarić which finds the exact solution of initial value problems, and we solve a simple equation with it. This algorithm works only for a certain class of differential equations, and hence extensions are required for more difficult problems.

Acknowledgements

I would like to thank those who helped me develop this thesis. First of all, my supervisor Herman Geuvers has showed me some interesting articles and explained a lot to me. Also, he improved my overal skills as computer scientist and mathematician. Furthermore, several people has proofread this work and found a lot of mistakes in my writings. Mostly Lotus and Bas have done this, and I am very thankful for their help.

Contents

Acknowledgements iii

Contents vi

Chapter 1. Introduction 1

Chapter 2. Preliminaries 3
2.1. Notation 3
2.2. Topology 3
2.3. Computable Analysis 7
2.4. Exact Real Arithmetic 11
2.5. Differential Equations 16
2.6. Domain Theory 18

Chapter 3. Polynomial representation of real functions 22
3.1. Effective Uniform Continuity and Pointwise Computability 22
3.2. Bernstein Polynomials 25
3.3. Conclusion 27

Chapter 4. Algorithms in Exact Real Arithmetic 29
4.1. Numerical Integration 29
4.2. Numerical Solutions of Ordinary Differential Equations 30
4.3. Solving Initial Value Problems using Domain Theory 32

Chapter 5. Conclusion 37

Appendix A. Heat equation 38

Bibliography 39

Index 40

CHAPTER 1

Introduction

Mathematical models of reality are common in science, and differential equations occur frequently in such models. Often it is challenging to solve these equations, and because of this one wants to solve them using computer support. Let us consider the Lotka-Volterra system as found in biology as example. Here we are looking for two functions u and v describing a population of preys and predators in thousands. Furthermore, these function should satisfy the following system of equations at any moment t in a certain period of time:

$$\begin{aligned} u'(t) &= 2 \cdot u(t) - u(t) \cdot v(t) \\ v'(t) &= -9 \cdot v(t) + 3 \cdot u(t) \cdot v(t) \\ u(0) &= 0.2, v(0) = 0.1. \end{aligned} \tag{1.1}$$

It is hard to find an explicit expression for these two functions, hence we need different methods to determine the values at a given moment. Several numerical methods have been developed for that goal. However, these are error-prone. A plot of the solutions can be found in Figure 1. These solutions are acquired using Euler's method with step size 0.001. One notices the populations will have higher extrema over time, but one can mathematically prove the solutions are periodic over time [**Olv12**]. Hence, the result is inprecise even though the method is mathematically correct.

These algorithms are error-prone, because every real number is represented by a finite approximation which causes inaccuracies in the result. However, this can be improved in two ways. On the one hand, someone can attempt to improve the algorithms. Smart tricks can decrease the loss of information which will improve the accuracy of the results. On the other hand, one can attempt to improve the representation of real numbers. When every digit of numbers can be computed, the loss of information can be prevented. Developments in computer science have made this solution feasible [**Gos72, Pot98**].

Furthermore, this issue motivates foundational questions. One can wonder '*Which real numbers can be represented on a computer?*' and '*Which real functions can be computed on a computer?*'. Both questions have extensively been studied and have been answered [**Grz55, PER83, PER84, Wei00**]. One expects that every normal function would be computable, but this is not the case. For example, we cannot decide in general whether a real number is equal to zero or not, because if this is the case, then we need to process an infinite amount of information in finite time. If an arbitrary real number is given as a sequence of digits or in another way, then we would only know a finite prefix of the string at any given moment. Therefore we cannot be sure the number is equal to zero or not. If one represents numbers as infinite sequences of digits, multiplication by three is problematic for the same reason. If a given number is equal to $\frac{1}{3}$ with a certain precision, we cannot be sure whether the result is lesser than, greater than or equal to one and thus we are not sure about its first digit. However, when one considers a different representation, multiplication by three will be fine. Because of this, it depends on the representation whether certain functions are computable. It gets more complicated when one looks at problems like root-finding, integration or finding the solutions of differential equations. Therefore, one needs a theory about representations, and this has been developed in [**Wei00**].

Various representations of functions have their own advantages in practice as well. Solutions of differential equations can be found using a certain representation of a function which can be seen in appendix A. Therefore, it is interesting to study these representations in a more general theory. Furthermore, several algorithms exist to approximate the solutions of ordinary differential equations, and one may wonder whether these can give approximations with a certain specified precision.

The main contribution of this thesis can be found in Chapter 3 and Chapter 4. In the former chapter we consider a representation of real functions based on approximation with polynomials from [**Wei00**] and we prove it is admissible which basically means it works. In the latter chapter we discuss several algorithms

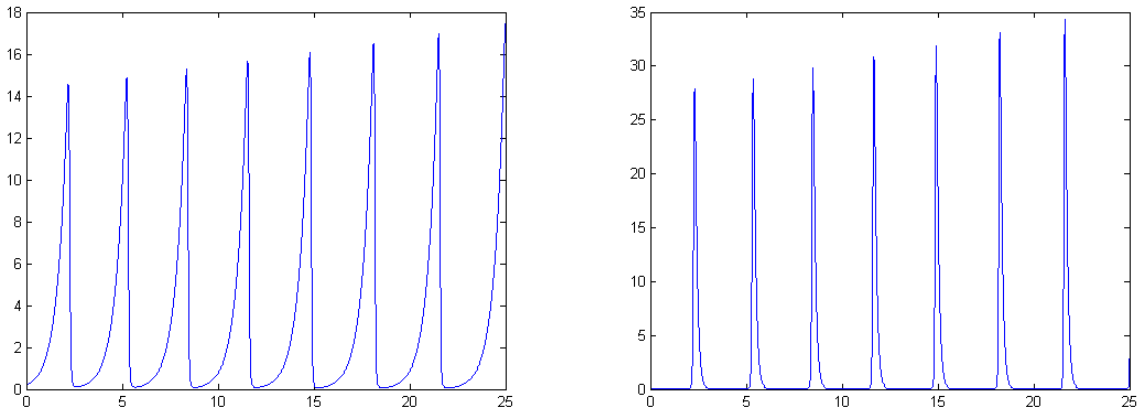

FIGURE 1. The solution of u on the left and v on the right

for solving ordinary differential equations which are descibed in [**EKL03, Pat05**]. Also, we argue that the classical Runge-Kutta methods are not suitable for finding an exact solution of a differential equation.

Below we describe the content of this thesis.

In Chapter 2 we discuss the notation and preliminaries required for this thesis. We start reviewing several notions from topology which are used for computable analysis. Computable analysis forms the theoretical basis of exact real arithmetic in which results can be approximated with any desired precision. Then we discuss several theorems on the existence and uniqueness of solutions for differential equations. Last we review some notions from domain theory which can be used to develop better algorithms for approximating the solutions of differential equations.

In Chapter 3 we discuss a different representation of real functions which representents them by an approximation with polynomials. We prove this representation works by proving it is equivalent to another representation. This is basically an effective version of Weierstrass' theorem.

In Chapter 4 we discuss several algorithms which can be implemented with exact real arithmetic. We start with an implementation of the classical Runge-Kutta methods and discuss it is not suitable. Then we discuss another algorithm based on Domain Theory which is better for an implementation in exact real arithmetic.

CHAPTER 2

Preliminaries

In this chapter we discuss the basic definitions and notations required for this thesis. Our most important tools are computable analysis, exact real arithmetic and domain theory. Because the former requires topology, we begin by discussing that. Furthermore, we discuss some theorems about differential equations.

2.1. Notation

Most of our notation is the usual notation in the literature. The set of infinite words with elements from Σ is denoted as Σ^ω. We use $v \triangleleft w$ to say that v is a subword of w. Furthermore, the word $v.w$ is the result of concatenating v and w. The pre-image of a set U with respect to the function f is denoted as $f^{-1}(U)$ and the image of U is written as $f(U)$. To denote the domain of a function f, we use $\mathrm{dom}(f)$. We write $\mathrm{limsup}_{x\to y} f(x)$ and $\mathrm{liminf}_{x\to y} f(x)$ for respectively the limes superior and limes inferior.

2.2. Topology

Topology can be viewed as a general theory for approximation, and this interpretation makes it important for (computable) analysis. In this section we summarize several definitions from topology which are taken from [**Kel55, Wil04**]. Furthermore, we discuss several recurring examples and their properties. We start with the notion of *topological spaces* which are the basic mathematical structures for approximations:

Definition 2.2.1 (Topological space)**.** We call a family of subsets $\tau \subseteq \mathscr{P}(X)$ of a set X a *topology* on X iff it satisfies the following conditions:

- It contains the empty set and X,
- It is closed under arbitrary unions,
- It is closed under finite intersections.

If τ is a topology on X, we call (X,τ) a *topological space*.

The elements in X are called *points*, and the sets in τ are called the *open* sets. The complement of an open set is called *closed* and a subset which is both open and closed, is called *clopen*. An open set can be viewed as an approximation of a point. We can make an approximation vaguer with unions, and more precise with intersections. We need the following notions:

Definition 2.2.2 (Interior, Exterior)**.** Given is a topological space (X,τ) and a subset $U \subseteq X$. We define the *interior* $\mathrm{int}(U)$ as the largest open set contained in U and the *closure* $\overline{U}$ as the smallest closed set containg U.

Often we want to describe a topology by stating which sets should be in it. Therefore, we introduce *bases* and *subbases*:

Definition 2.2.3 (Base)**.** Let (X,τ) be a topological space and $\mathbb{B}$ a collection of open subsets of X. The collection $\mathbb{B}$ is called a *base* for τ iff for every point $x \in X$ and open set $U \in \tau$ with $x \in U$ the collection $\mathbb{B}$ contains a set $V \subseteq U$ contained in U.

Definition 2.2.4 (Subbase)**.** Let (X,τ) be a topological space and $\mathbb{S}$ a collection of open subsets of X. Then $\mathbb{S}$ is called a *subbase* for τ iff the family of finite intersections with sets in $\mathbb{S}$ is a base for τ.

If we have a set, then every family of subsets is a subbase of a topology which contains every union of finite intersections of that collection. The empty set and X are contained, because $\bigcap_{i\in\emptyset} A_i = X$ and $\bigcup_{i\in\emptyset} A_i = \emptyset$ by definition. Next we discuss the 'size' of a topological space which could be defined by the number of points. However, the following definition is more informative about the number of approximations:

Definition 2.2.5 (Second countable)**.** A topological space (X, τ) is called *second countable* iff it has a countable base.

If a space has a countable subbase, then the set of finite intersections is a countable base, and therefore the space is second countable. Often we are not just interested in individual topological spaces, but in multiple topological spaces with functions defined between them. Not every function works convenient, because it might ignore the structure of spaces. Therefore we define *continuity* of functions:

Definition 2.2.6 (Continuous functions)**.** Given two topological spaces (X, τ) and (Y, σ) and a function $f : X \to Y$ from X to Y, we call f *continuous* iff the pre-image of every open set in Y is open in X. We denote $C(X \to Y)$ for the set of continuous functions from X to Y.

If we want to determine whether a function is continuous or not, we should look at every open set in the co-domain. However, it is sufficient to restrict to a subbase:

Proposition 2.2.7. *Let (X, τ) and (Y, σ) be topological spaces, let $\mathbb{S}$ be a subbase for σ and let $f : X \to Y$ be a function. Then f is continuous iff for every $U \in \mathbb{S}$ the pre-image $f^{-1}(U)$ is open.*

The proof follows from the fact that pre-images act well with intersections. Another interesting property of topological spaces is the T_0*-property*:

Definition 2.2.8 (T_0-space)**.** A topological space (X, τ) is called a T_0*-space* iff for every pair $x, y \in X$ of distinct points, there is an open set $U \in \tau$ which contains only one of them.

This property is about the separation of points. This property can be restricted to a subbase as well:

Proposition 2.2.9. *Let (X, τ) be a topological space, and let $\mathbb{S}$ be a subbase for τ. Then is (X, τ) is a T_0-space iff for every pair $x, y \in X$ of distinct points, there is an set in the subbase $U \in \mathbb{S}$ which contains just one of them.*

PROOF. Let (X, τ) be an arbitrary topological space with a subbase $\mathbb{S}$, and assume that for every pair $x, y \in X$ of distinct points, there is a set in the subbase $U \in \mathbb{S}$ which contains just one of them. Then (X, τ) is a T_0-space, because every set in the subbase is open.

To prove the converse we assume (X, τ) is a T_0-space, and we take two distinct points $x, y \in X$. We know from Definition 2.2.8 that there is a set $U \in \tau$ which contains either x or y, and not the other point. Without loss of generality, we assume $x \in U$ and $y \notin U$. With Definition 2.2.4 we can find a finite set I, a set J, and for each $i \in I$, $j \in J$ a set $S_{i,j} \in \mathbb{S}$ in the subbase such that we can construct U:

$$U = \bigcup_{j \in J} \bigcap_{i \in I} S_{i,j} = U.$$

Since $x \in U$ is contained, we know there is an index $j \in J$ such that for every $i \in I$ the point $x \in S_{i,j}$ is contained in $S_{i,j}$. Furthermore, because the other point y is not in U, we know $y \notin \bigcap_{i \in I} S_{i,j}$. Hence, there is a $i \in I$ with $y \notin S_{i,j}$. And now we have found the desired set, because $S_{i,j}$ contains x and not y. □

This proposition tells us that points in a T_0-space can be distinguished by the subbase. Sometimes we are interested in a certain subset of a topological space. To reason about the properties of a subset, the *subspace topology* is introduced:

Definition 2.2.10 (Subspace topology)**.** Given an topological space (X, τ) and a subset $Y \subseteq X$ of X, then we define the *subspace topology* τ_Y on Y as follows:

$$\tau_Y := \{U \cap Y : U \in \tau\}.$$

A useful property of topological spaces is *compactness*. This basically says the space is 'small'. In words, if we cover such a space with open sets, then we can choose a finite amount of them which cover the space as well. This gives the following definition:

Definition 2.2.11 (Compact topological space)**.** A topological space (X, τ) is called a *compact* iff for every open cover $\{U_i \in \tau : i \in I\}$ such that $\bigcup_{i \in I} U_i = X$, there is a finite subcover $F \subseteq I$ which covers X.

A subset of a topological space is called compact iff it is a compact space with respect to the subspace topology. Continuous functions preserve compactness which means the image of a compact set is compact:

Proposition 2.2.12. *Let two topological spaces (X,τ) and (Y,σ) be given such that X is compact. If $f : X \to Y$ is any continuous function, then the image $f(X)$ is a compact subspace of Y.*

If an arbitrary open cover of the image $f(X)$ is given, then the pre-images form an open cover of X. Hence, by compactness a finite subcover can be given. Furthermore, the notion of *connectedness* is useful, because it is a generalisation of intervals on the real line. Connected sets cannot be split in multiple non-empty disjoint clopen pieces:

Definition 2.2.13 (Connected topological space)**.** A topological space (X,τ) is called a *connected* iff every clopen subset $K \subseteq X$ is either empty or the whole set.

A subspace of a topological space is called connected if it is connected with respect to the subspace topology. These spaces have certain properties. Firstly, continuous functions preserve connectedness.

Proposition 2.2.14. *Let $f : X \to Y$ be a continuous function from a connected topological space (X,τ_X) to a topological space (Y,τ_Y), then the image $f(X)$ is connected.*

The proof is similar to the proof of Proposition 2.2.12. Secondly, the closure of a connected subset is connected too.

Proposition 2.2.15. *If $C \subseteq X$ is a connected subset of a topological space (X,τ), then $\overline{C}$ is connected as well.*

For Proposition 2.2.15 the set X is contained in the intersection. With the subspace topology one can prove X cannot be connected if $\bigcap_{K\in Q} K$ is not connected. A topological space can be studied by splitting it in several pieces, for example in *connected components*:

Definition 2.2.16 (Connected component)**.** A subset $Y \subseteq X$ of a topological space (X,τ) is called a *connected component* iff for every connected subset $C \subseteq X$ with $Y \subseteq C$ the equality $Y = C$ holds.

Some topological spaces do not satisfy a certain property globally, but just locally which means every point has a neighborhood with that property. Sometimes this is useful, and we define:

Definition 2.2.17 (Locally connected)**.** A topological space (X,τ) is called *locally connected* iff for every $x \in X$ and $U \in \tau$ with $x \in U$ there is a connected set $W \subseteq X$ such that $x \in \mathrm{int}(W)$ and $W \subseteq U$.

So it says every neighborhood of a point contains a connected set around that point. The following proposition from [**Wil04**] is needed:

Proposition 2.2.18. *If a topological space is compact and locally connected, then it has a finite number of connected components.*

Because the connected sets form a open cover of the space, a finite number can be chosen which cover the space. Now we discuss at some recurring examples which are taken from [**Wei00, Wil04**]. The first two are crucial for computable analysis, and the last two examples are used in real analysis:

Example 2.2.19 (Finite words)**.** Let Σ be an arbitrary finite set, and let Σ^* be the set of finite sequences of elements from Σ. We define a topology $\tau_* := \mathscr{P}(\Sigma^*)$ as the powerset of Σ^* and a subbase $\mathbb{S} := \{\{x\} : x \in \Sigma^*\}$ containing only the singleton sets. Notice that τ_* is indeed a topology on Σ^*, since finite intersections and arbitrary unions of subsets of a set X are a subset of X. Furthermore, $\mathbb{S}$ is indeed a subbase for τ_*, because it is a base for τ_*. Since Σ^* is countable, the subbase is countable too. With Proposition 2.2.9 we can see (Σ^*,τ_*) is a T_0-space. We call τ_* the *discrete topology* on Σ^*.

Example 2.2.20 (Infinite words)**.** Let Σ^ω be the set of all infinite sequences of Σ. This time we define the topology with a subbase $\mathbb{S} := \{w.\Sigma^\omega : w \in \Sigma^*\}$ containing of sets of infinite words starting with a certain prefix where $w.\Sigma^\omega$ is the set of all infinite words starting with w. The generated topology is called the *Cantor topology*, and is denoted by τ_C. Since Σ^* is countable, we see the subbase $\mathbb{S}$ is countable as well, and hence the space (Σ^ω,τ_C) is second countable. Furthermore, if x and y are two distinct infinite sequences, there is a finite prefix u of x, such that u is not a prefix of y. That means $x \in u.\Sigma^\omega$, and $y \notin u.\Sigma^\omega$. Therefore, we conclude (Σ^ω,τ_C) is a T_0-space.

The subbases of Examples 2.2.19 and 2.2.20 are informative. If we work with finite words, we have complete information, and thus we can give a perfect approximation of every word. However, if we work with infinite words and read them sequentially, we just know a finite prefix of the word. Therefore, our approximations contain complete information about a finite prefix followed by an undetermined part.

Example 2.2.21 (Real numbers)**.** Let $\mathbb{R}$ be the set of all real numbers, and define the following subbase $\mathbb{S} := \{(a,b) : a,b \in \mathbb{Q}\}$ which are the open intervals with rational endpoints. The generated topology is known as the *Euclidian topology* which we denote by τ_E. This topological space is second countable, because subbase is countable. It is known from analysis $(\mathbb{R}, \tau_E)$ is a (locally) connected T_0-space.

Example 2.2.22 (Compact-open topology)**.** Let two topological spaces (X, τ_X) and (Y, τ_Y) be given. We define for every compact set $K \subseteq X$ and open set $U \in \tau_Y$:

$$K_U := \{f : X \to Y : f(K) \subseteq U\}.$$

Furthermore, we define the *compact-open topology* τ_{Co} on $C(X \to Y)$ as the generated topology by the following subbase:

$$\sigma := \{K_U : K \subseteq X \text{ compact}, U \in \tau_Y\}.$$

Properties of this space depend on X and Y. Firstly, the compact-open topology has the T_0-property if Y is a T_0-space, because if two functions $f, g \in C(X \to Y)$ are not equal, there is a point $x \in X$ in the domain such that the images $f(x) \neq g(x)$ are not equal. Because Y has the T_0-property, we can find an open set $U \in \tau_Y$ which contains either $f(x)$ or $g(x)$. Therefore, the set $\{x\}_U$ contains either f or g. Now we want to prove the function space $C(\mathbb{R} \to \mathbb{R})$ is second countable. This requires some work:

Lemma 2.2.23. *The function space $C(\mathbb{R} \to \mathbb{R})$ is second countable when both spaces are equiped with the Euclidian topology.*

PROOF. Our goal is to find a countable subbase for the compact-open topology. Our candidate is the following family of sets:

$$\Upsilon := \{[a,b]_{(c,d)} : a,b,c,d \in \mathbb{Q}\}.$$

Notice that sets in Υ are open. Let $f \in C(\mathbb{R} \to \mathbb{R})$ be an arbitrary continuous function and let $V \in \tau_{Co}$ be any open set containing f. Then there is an open set $U \subseteq \mathbb{R}$ and a compact set $K \subseteq \mathbb{R}$ such that $K_U \subseteq V$ and $f \in K_U$, because σ is a subbase of the compact-open topology. Firstly, we want to split the compact set K into a finite number of connected components. Since K is compact and $\mathbb{R}$ is locally connected, we can conclude this is possible by Proposition 2.2.18. Therefore, there is a finite set I and a family of subsets $\{K_i \subseteq K : i \in I\}$ such that:

$$K = \bigcup_{i \in I} K_i.$$

Now we analyse the function on every connected component K_i. Our goal is to find an rational interval contained in U and disjoints rational intervals containing one of the connected components. Let $i \in I$ be arbitrary. We know that $f(K_i) \subseteq U$ and that $f(K_i)$ is compact and connected by Propositions 2.2.12 and 2.2.14. Therefore, we can write $f(K_i) = [x,y]$ for certain $x, y \in \mathbb{R}$. Since U is open, there are radii $\varepsilon_x, \varepsilon_y \in (0, \infty)$ such that the intervals $(x - \varepsilon_x, x + \varepsilon_x)$ and $(y - \varepsilon_y, y + \varepsilon_y)$ are contained in U. Because the rational numbers are a dense subset of the reals, we can find two rational numbers $c_i \in (x - \varepsilon_x, x)$ and $d_i \in (y, y + \varepsilon_y)$. This gives $f(K_i) \subseteq (c_i, d_i) \subseteq U$.

We define $d := \min\{|x - y| : x \in K_i, y \in K_j, i \neq j\}$ to be the minimum distance between the connected components. Because there is a finite number of connected components, we know $d > 0$. Since f is continuous, we can find a $\delta < d$ such that $f((x - \delta, y + \delta)) \subseteq (c_i, d_i)$. Again we can find two rational numbers $a_i \in (x - \delta, x)$ and $b_i \in (y, y + \delta)$ such that $f([a_i, b_i]) \subseteq (c_i, d_i)$. Now we define the following set:

$$W := \bigcap_{i \in I} [a_i, b_i]_{(c_i, d_i)}.$$

By construction we see $f \in W$, and $W \subseteq K_U \subseteq V$. Therefore we can conclude Υ is a countable subbase for the compact-open topology. □

In conclusion, the function space $C(\mathbb{R} \to \mathbb{R})$ is both T_0 and second countable. Notice the proof easily generalizes to subsets of the real numbers.

2.3. Computable Analysis

There are several approaches to computable analysis, for example Grzegorczyk's approach [**Grz55**], L^p-computability [**PER84**], Type-2 theory of effectivity [**Wei00**] and Gray code computability [**Tsu02**]. In this thesis we work with Type-2 theory of effectivity, because it allows comparison between several representations. In this section we summarize most important definitions from [**Wei00**].

Type-2 computability starts with computability on 'concrete' objects in Σ^* and Σ^ω. With *naming systems* similar notions on 'abstract' objects are defined. The topological spaces (Σ^*, τ_*) and (Σ^ω, τ_C) in Examples 2.2.19 and 2.2.20 are very important, because they are required to define the quality of representations. We start with the definition of a *Turing machine*:

Definition 2.3.1 (Turing machine)**.** Given are an alphabet Σ, a set Ξ of work letters, a set Q of states and two natural numbers $k, n \in \mathbb{N}$ where k is the amount of input tapes and n the amount of work tapes. We assume that $\Sigma \cap \Xi = \emptyset$ and that there is a blank tape symbol $\mathrm{B} \in \Xi$. Furthermore, let there be a starting state $q_0 \in Q$ and a final state $q_f \in Q$. First, we define the work alphabet Γ as the union of the alphabet and the work letters. This set contains the characters which can be written on the work tape while only the elements of Σ can be written on the input and output tapes. Then we define the set of actions, Act, as follows:

$$\begin{aligned} Act :=& \{(L,i) : i \in \mathbb{N}, k < i \leq k+n\} \cup \{(R,i) : i \in \mathbb{N}, i \leq n+1\} \\ &\cup \{(W,i,a) : i \in \mathbb{N}, a \in \Sigma, i \leq k \text{ or } i = n+1\} \\ &\cup \{(W,i,a) : i \in \mathbb{N}, a \in \Gamma, k < i \leq n+1\} \end{aligned}$$

where L indicates a step to the left, R a step to the right, and W the write action. Let N be the set of natural numbers between 1 and $k+n+1$ and let a partial function $\delta :\subseteq Q \times \Gamma \times N \to Q \times Act$ be given. Then we define the *Turing machine* with states Q, alphabet Σ, work alphabet Γ, transition function δ, initial state q_0, final state q_f, k input tapes and n worktapes as the tuple $(Q, \Sigma, \Gamma, \delta, (q_0, q_f), (k, n))$. We denote $M(y)$ to be the result on the output tape when the machine M has input $y \in \Sigma^*$.

This definition is rather complicated, but the idea is quite simple. The input and output tape of a Turing machine accept words from Σ^*, but on the work tape words from Γ^* can be written with the W action. The other actions R and L move the head on the tapes, and one cannot move the head on an input tape or the output tape to the left. Only heads on the work tapes can be moven to the left. The transition function can be visualized as a flowchart. We have an initial state q_0 which tells us where to start on the flow chart, and q_f is the end. Every state is a certain step in the flowchart, and the required actions depend on the state and on the input on a certain tape. The output tape is numbered $n+1$, the input tapes are numbered from 1 to k and the work tapes are numbers from $k+1$ to n.

A Turing machine only allows finite input and output, and thus they cannot have input from Σ^ω. This is very inconvenient for computable analysis, because real numbers are infinite objects. Therefore the definition is extended to *Type-2 machines* which can work with infinite strings:

Definition 2.3.2 (Type-2 machine)**.** Given two natural numbers $k, n \in \mathbb{N}$ and types $Y_i \in \{\Sigma^*, \Sigma^\omega\}$ for $1 \leq i \leq k$, then we define the *Type-2 machine* as the Turing machine with k input tapes and N work tapes with a type specification $(Y_i)_{i=1}^k$.

A Type-2 machine is a Turing machine in which the input and output can be infinite. Every input tape has a type which says whether it accepts finite or infinite words. Also, the output tape has a type which says whether the output is finite or infinite. Now we can define *computable functions* and *computable elements* of Σ^* and Σ^ω:

Definition 2.3.3 (Computable function)**.** Let $Y_i \in \{\Sigma^*, \Sigma^\omega\}$ for every $i \leq n$ and let $Y = \Pi_{i \in \{j \in \mathbb{N} : j \geq 1, j \leq n\}} Y_i$ be a type specification. A string function $f : Y \to Y_0$ is called *computable* iff it is computed by a Type-2 machine. This means that for every $y \in Y$ the values are $f(y) = M(y)$.

Definition 2.3.4 (Computable element)**.** Every finite word $w \in \Sigma^*$ is called computable. An infinite word $w \in \Sigma^\omega$ is called computable iff the constant function $f(x) = w$ is computable. Computable words are called *computable elements*.

An important property of Type-2 machines is the finiteness property which states that every finite part of the output is determined by a finite part of the input. This gives a realistic way of computing with real

numbers, because one can compute the result with any desired accuracy using only finite information. The finiteness property is stated as follows:

Proposition 2.3.5 (Finiteness property)**.** *Given a machine M and a input y, then every finite portion of the output of $M(y)$ is already determined by a finite part of y.*

The proof uses that the output of a machine cannot be changed during the computation, because the output tape cannot move to the right. Combining the topological structure with the definitions from computability, we can find certain properties of computable functions. Computable functions are necessarily continuous:

Theorem 2.3.6. *Every computable string function is continuous with respect to its topology.*

Furthermore, the domain satisfies certain properties. For example:

Theorem 2.3.7. *Let $Y_i \in \{\Sigma^*, \Sigma^\omega\}$ for every $i \leq n$ and let $Y = \prod_{i\in\{j\in\mathbb{N}:\, j\geq 1, j\leq n\}} Y_i$. Every computable string function $f : Y \to \Sigma^*$ with finite output has an open domain, and the domain of computable string functions $f : Y \to \Sigma^\omega$ with infinite output is a countable intersection of open sets.*

Before we can discuss naming systems, we need *tupling functions*. Those functions are required to wrap finite words into different strings such that their image do not overlap. This means that if two images v and w are given such that $v \triangleleft w$, then they must be equal. First we define those functions in a concrete way, and then we discuss the properties:

Definition 2.3.8 (Tupling function)**.** Given an alphabet Σ with $\mathbf{0}, \mathbf{1} \in \Sigma$, we define the *tupling function* $\iota : \Sigma^* \to \Sigma^*$ with $c_i \in \Sigma$:

$$\iota((c_i)_{i=0}^n) = \mathbf{110}.(c_i.\mathbf{0})_{i=0}^n.\mathbf{011}.$$

The beginning of a word is indicated by the string **110** and the end by **011**. The characters are separated by **0**. The following proposition is useful:

Proposition 2.3.9. *The function ι is computable, and if $\iota(w)\iota \triangleleft (v)$, then $\iota(w) = \iota(v)$.*

To extend computability from 'concrete' objects to 'abstract' objects, *naming systems* are required. There are two kinds of naming systems, namely *notations* and *representations* which are defined as follows:

Definition 2.3.10 (Notation)**.** We call a function $\gamma : \Sigma^* \to M$ a *notation* for a set M iff it is a surjection.

Definition 2.3.11 (Representation)**.** A function $\gamma : \Sigma^\omega \to M$ is a *representation* for a set M iff it is a surjection.

Definition 2.3.12 (Naming systems)**.** A *naming system* for a set M is either a notation or representation.

If a word w is mapped by a naming system γ to an element x, then we call w a *γ-name* of x. We define *computability induced by naming systems* as follows:

Definition 2.3.13 (Computablity by naming systems)**.** Let M and N be sets and let $\gamma_M : Y_M \to M$ and $\gamma_N : Y_N \to N$ be naming systems with $Y_M, Y_N \in \{\Sigma^\omega, \Sigma^*\}$. An element $m \in M$ is *computable* iff there is a computable element $y \in Y_M$ with $\gamma(y) = m$. Furthermore, a function $f : M \to N$ is called *computable* respectively *continuous* iff there is a computable respectively continuous function $g : Y_M \to Y_N$ such that for every $y \in Y_M$ the equality $f(\gamma_M(y)) = \gamma_N(g(y))$ holds if $f(\gamma_M(y))$ exists.

An important property of naming systems is *admissibility* which means approximation properties are preserved by the naming system. However, this property is not defined on arbitrary sets, but just on a certain class of spaces. Furthermore, we use a specific representation for this definition. Hence, this definition requires some steps, and we start with the notion of *effective topological spaces*:

Definition 2.3.14 (Effective topological space)**.** Let M be a set, let $\sigma \subseteq \mathcal{P}(M)$ be a countable family of subsets of M and let $\gamma : \Sigma^* \to \sigma$ be a notation for σ. If the topology τ_S generated by σ makes (M, τ_S) a T_0-space, then we call (M, σ, γ) an *effective topological space*.

Notice that we can turn a second countable T_0-space into an effective topological space by defining a notation for the subbase. A subbase of the topology contains atomic approximations of the points. These can be written as finite strings which means we can compute with in finite time. Every effective topological space induces a representation of its points called the *standard representation*:

Definition 2.3.15 (Standard representation)**.** Let an effective topological space $\mathbb{S} = (M, \sigma, \nu)$ be given. First, we define:

$$\mathbb{T} := \{p \in \Sigma^\omega : \text{every } w \in \Sigma^* \text{ with } \iota(w) \triangleleft p \text{ is an element of } \mathrm{dom}(\nu)\ \}.$$

Now we define the *standard representation* $\delta_S :\subseteq \mathbb{T} \to M$ such that for every $p \in \mathbb{T}$ in the domain the value $\delta_S(p) = x$ with $x \in M$ iff the following equality holds

$$\{A \in \sigma : x \in A\} = \{\nu(w) : w \in \Sigma^*, \iota(w) \triangleleft p\}.$$

This definition gives a well-defined function, because the space has the T_0-property. If an infinite word represents two elements $x, y \in M$, then the families of approximations $\{A \in \sigma : x \in A\}$ and $\{A \in \sigma : y \in A\}$ are equal and therefore x and y are equal by the T_0-property. Furthermore, the properties of the tupling function ι guarantees subwords can be recognized. This representation enumerates every property of an element. Now we discuss how to compare different naming systems:

Definition 2.3.16 (Equivalence)**.** Let $Y, Y' \in \{\Sigma^*, \Sigma^\omega\}$ be alphabets and let $\gamma : Y \to M$ and $\gamma' : Y' \to M$ be naming systems for a set M. A function $f : Y \to Y'$ *translates* γ to γ' iff for every $x \in \mathrm{dom}(\gamma)$ the elements $\gamma(x)$ and $\gamma'(f(x))$ are equal. Furthermore, we say:

- $\gamma \leq \gamma'$ iff there is a computable function which translates γ to γ',
- $\gamma \leq_t \gamma'$ iff a continuous function translates γ to γ',
- $\gamma \equiv \gamma'$ iff $\gamma \leq \gamma'$ and $\gamma' \leq \gamma$,
- $\gamma \equiv_t \gamma'$ iff $\gamma \leq_t \gamma'$ and $\gamma' \leq_t \gamma$.

Notice that $\equiv$ and $\equiv_t$ are equivalence relations, and that $\leq$ and $\leq_t$ are pre-orders. As conclusion we define the notion of *admissible naming systems*:

Definition 2.3.17 (Admissible naming system)**.** Let an second countable T_0-space (M, τ) and a naming system $\gamma : Y \to M$ with $Y \in \{\Sigma^*, \Sigma^\omega :\}$ be given. We call γ *admissible* iff there is an effective topological space $\mathbb{S} = (M, \sigma, \nu)$ such that γ is equivalent to the standard representation on S and the topologies τ and τ_S are equal.

Basically this definition says a the information of a 'good' representation can be used to every property of an element. Admissable representations are of interest, because they preserve continuity of functions. An abstract function is continuous if and only if its algorithm is continuous. This property is described in the following theorem which we do not prove:

Theorem 2.3.18. *Let I be a finite set such that for every $i \in I$ there is a topological space (M_i, τ_i) and an admissible representation $\delta_i : \Sigma^\omega \to M_i$. Furthermore, let a topological space (M, τ), an admissible representation $\delta : \Sigma^\omega \to M$ and a function $f : \Pi_{i \in I} M_i \to M$ be given. Then f is continuous with respect to the topologies iff f is continuous with respect to the representations.*

Now we discuss several examples of naming systems. Some are admissible, and some are not. Remember that admissibility is a property for representations. We begin with a notation for countable number systems, because those are required for the representations of real numbers and real functions. We assume $\Sigma = \{\mathbf{0}, \mathbf{1}, \mathbf{-}, \mathbf{/}\}$:

Example 2.3.19 (Natural numbers)**.** We define the notation $\nu_{\mathbb{N}} : \{\mathbf{0}, \mathbf{1}\}^* \to \mathbb{N}$ by:

$$\nu_{\mathbb{N}}((w_i)_{i=0}^n) := \sum_{i=0}^{n} 2^i \cdot w_i.$$

Example 2.3.20 (Integers)**.** First we define the following set:

$$\mathbb{I} := \{\mathbf{0}\} \cup \mathbf{1}.\{\mathbf{0}, \mathbf{1}\}^* \cup \mathbf{-}.\mathbf{1}.\{\mathbf{0}, \mathbf{1}\}^*.$$

On this set we define the notation $\nu_{\mathbb{Z}} : \mathbb{I} \to \mathbb{Z}$ for integers by:

$$\nu_{\mathbb{Z}}((w_i)_{i=0}^n) := \begin{cases} 0 & \text{if } w(0) = \mathbf{0} \\ \nu_{\mathbb{N}}((w_i)_{i=1}^n) & \text{if } w(0) = \mathbf{1} \\ -\nu_{\mathbb{N}}((w_i)_{i=1}^n) & \text{if } w(0) = \mathbf{-} \end{cases}.$$

Example 2.3.21 (Rational numbers)**.** The domain of the notation is the following set:

$$\mathbb{B} := \{u./.v : u \in \mathrm{dom}(\nu_{\mathbb{Z}}), v \in \mathrm{dom}(\nu_{\mathbb{N}}), \nu_{\mathbb{N}}(v) \neq 0\}.$$

Now we define a naming system $\nu_{\mathbb{Q}} : \mathbb{B} \to \mathbb{Q}$ for rational numbers as follows:

$$\nu_{\mathbb{Q}}(u./.v) := \frac{\nu_{\mathbb{Z}}(u)}{\nu_{\mathbb{N}}(v)}.$$

For the notation of rational intervals we can either introduce a new character in the alphabet or use the tupling functions. We use the tupling functions, because that generalizes more easily to $\mathbb{R}^n$:

Example 2.3.22 (Open Rational Intervals)**.** First we define the set of every word containing two fractions:

$$RI := \{x \in \Sigma^* : x = \iota(v).\iota(w) \text{ for some } v, w \in \mathrm{dom}(\nu_{\mathbb{Q}})\}.$$

The notation $I^1 : RI \to \mathrm{Cb}^{(1)}$ is defined as follows:

$$I^1(\iota(v).\iota(w)) := \mathrm{B}(v, w)$$

where $\mathrm{B}(v, w)$ is the ball around v with radius w.

A rational interval is viewed as two rational numbers. The first one is a point on the real line, and the second number is the diameter. This generalizes very well to multiple dimensions, because those balls form a base for the Euclidian topology if a suitable metric is chosen. Now we can easily find a representation for the real numbers combining Examples 2.2.21 and 2.3.22 with the previous definitions:

Example 2.3.23 (Real numbers)**.** In Example 2.2.21 we discussed the Euclidian topology for the real numbers which is a second-countable T_0-space. To construct an effective topological space, we use Example 2.3.22. We have a set $\mathbb{R}$, a subbase σ and a notation I for the subbase, thus we have an effective topological space $\mathbb{S} := (\mathbb{R}, \sigma, I^1)$. This induces a standard representation $\delta_{\mathbb{S}}$ and we define $\rho := \delta_{\mathbb{S}}$.

It can be hard to work with this representation, because every property of the number is in the subset. However, this representation is equivalent to the naming system in which real numbers are viewed as a shrinking sequence of nested intervals. The latter is often more convenient in proofs. The computable real numbers have the following property:

Proposition 2.3.24. *If $a, b \in \mathbb{R}$ are computable real numbers, then their sum $a + b$ and product $a \cdot b$ are computable.*

Another representation for the real numbers uses continued fractions. This representation is not admissible to the Euclidian topology. However, it is useful in practice, because several algorithms and numbers can be defined using continued fractions. Furthermore, continued fractions have a high convergence speed which makes them efficient.

Example 2.3.25 (Continued fractions)**.** Let $x \in \mathbb{R}$ with $x \geq 0$ be a non-negative real number. First, we define its *continued fraction expansion* as follows:

$$\mathrm{Fr}(x) := (a_i)_{i=0}^{\infty}.$$

Where $x_0 := x$, $a_n := \lfloor x_n \rfloor$, $x_{n+1} := \frac{1}{x_n - a_n}$ if $x_n \neq a_n$ and $x_{n+1} := 0$ if $x_n = a_n$. Furthermore, we define the following set as the domain:

$$\mathbb{I} := \{(\iota(w_i))_{i=0}^{\infty} : w_i \in \mathrm{dom}(\nu_{\mathbb{N}})\} \cup \{-.(\iota(w_i))_{i=0}^{\infty} : w_i \in \mathrm{dom}(\nu_{\mathbb{N}})\}.$$

Whenever the equality $\mathrm{Fr}(x) = (\nu_{\mathbb{N}}(w_i))_{i=0}^{\infty}$ holds, we define:

$$\rho_{\mathrm{Cf}}((\iota(w_i))_{i=0}^{\infty}) = x,\ \rho_{\mathrm{Cf}}(-.(\iota(w_i))_{i=0}^{\infty}) = -x.$$

Now we discuss some representations for the function space $C(\mathbb{R} \to \mathbb{R})$, and one uses the compact-open topology of Example 2.2.22. First, we need a notation for pairs of rational intervals:

Example 2.3.26 (Pairs of intervals)**.** First we define the set:

$$PI := \{x \in \Sigma^* : x = \iota(v).\iota(w) \text{ for some } v, w \in \mathrm{dom}(I^1)\}.$$

Now we define the following notation:

$$\nu_{\mathrm{Co}}(\iota(v).\iota(w)) := \overline{I^1(v)}_{I^1(w)}.$$

To construct a representation for real functions, we combine Examples 2.2.22 and 2.3.26:

Example 2.3.27 (Compact-open representation)**.** In Example 2.2.22 we discussed a topology on the function space $C(\mathbb{R} \to \mathbb{R})$. Now we define a representation for $C(A \to \mathbb{R})$ for $A \subseteq \mathbb{R}$. Remember the topology has a countable subbase $\mathbb{S}$ which consists of pairs of rational intervals. Therefore, the space $\mathbb{T} := (C(A \to \mathbb{R}), \mathbb{S}, \nu_{\mathrm{Co}})$ is an effective topological space, and we define $\delta^A_{\mathrm{co}} := \delta_{\mathbb{T}}$.

The compact-open representation enumerates the possible values in every compact interval of the domain. Another possible way is to describe a function as an approximation of simple functions, like rational polygons:

Definition 2.3.28 (Rational polygon)**.** Let $a, b \in \mathbb{R}$ be real numbers with $a < b$, and $f \in C([a,b] \to \mathbb{R})$ be a real function. Assume, there is a natural number $k \in \mathbb{N}$ and there are two rational vectors $p, q \in \mathbb{Q}^k$ such that $p(0) \leq a$ and $p(k) \geq b$ and for every natural number $i \in \mathbb{N}$ with $0 < i < k$ the inequality $p(i) < p(i+1)$ holds. Then f is called a rational polygon iff the following equality holds for every natural number $i \in \mathbb{N}$ with $0 < i < k$ and real number $x \in [a,b]$ with $x \in [p(i), p(i+1)]$:

$$f(x) = q(i) + \frac{q(i+1) - q(i)}{p(i+1) - p(i)} \cdot (x - p(i)).$$

The space of rational polygons over an interval $[a,b]$ is denoted $\mathrm{Pg}([a,b] \to \mathbb{R})$.

A rational polygon is the result of linear interpolation. One choses certain interpolation nodes with corresponding weights, and then draws straight lines between them. It is easy to define a notation ν^A_{Pg} of rational polygons on a closed set A by using the tupling functions. Before we specify the representation, we need to define effective convergence based on [**PER83**]:

Definition 2.3.29 (Effective convergence)**.** Let $f_n : A \to \mathbb{R}$ be a sequence of real functions defined on a subset $A \subseteq \mathbb{R}$ and let $g : A \to \mathbb{R}$ be a function. Then we say f converges to g or $\lim_{n\to\infty} f_n = g$ iff there is a recursive function $\varepsilon : \mathbb{N} \to \mathbb{N}$ such that for every $n, m \in \mathbb{N}$ with $n \geq \varepsilon(m)$ the inequality $|f_n(x) - g(x)| < 10^{-m}$ holds.

Notice that effective convergence implies uniform convergence, because ε does not depend on x. Now we define the following representation:

Example 2.3.30 (Cauchy representation)**.** Let $a, b \in \mathbb{R}$ be computable real numbers. First, we define the following set:

$$\mathbb{I} := \{(\iota(w_i))^\infty_{i=0} : w_i \in \mathrm{dom}(\nu^A_{\mathrm{Pg}}) \text{ for } i \in \mathbb{N}\}.$$

Then we define $\delta^A_{\mathrm{C}} : \mathbb{I} \to C([a,b] \to \mathbb{R})$ whenever the sequence $\nu^A_{\mathrm{Pg}}(w_i)$ converges effectively to a function:

$$\delta^A_{\mathrm{C}}((\iota(w_i))^\infty_{i=0}) = \lim_{n\to\infty} \nu^A_{\mathrm{Pg}}(w_i)$$

The Cauchy representation is admissible with respect to the compact-open topology on any interval $[a,b]$ when $a, b \in \mathbb{R}$ are computable. One can similary define a notation ν^A_{P} on polynomials with rational coefficients which can be used for an alternative representation:

Example 2.3.31 (Polynomial representation)**.** Given are two computable real numbers $a, b \in \mathbb{R}$. Our representation has the following domain:

$$\mathbb{I} := \{(\iota(w_i))^\infty_{i=0} : w_i \in \mathrm{dom}(\nu^A_{\mathrm{P}}) \text{ for } i \in \mathbb{N}\}.$$

We define the naming system $\delta^A_{\mathrm{Cp}} : \mathbb{I} \to C([a,b] \to \mathbb{R})$ as follows:

$$\delta^A_{\mathrm{Cp}}((\iota(w_i))^\infty_{i=0}) = \lim_{n\to\infty} \nu^A_{\mathrm{P}}(w_i)$$

whenever the sequence $\nu^A_{\mathrm{P}}(w_i)$ converges effectively to some real function.

2.4. Exact Real Arithmetic

Computable analysis can be applied to develop exact representations of real numbers in programming languages. These datastructures are infinite objects which can compute the result with any desired accuracy. Intermediate results can be computed with any precision, and therefore errors can be prevented. At first exact real arithmetic was done using continued fractions [**Gos72**], but continued fractions do not give an admissible representation. Therefore, we look at an alternative based on nested intervals which is related to the representation in Example 2.3.23. It uses *linear fractional transformations* to represent real numbers, and

was developed by Potts and Edalat [**PE96, PEE97, Pot98, ER98, EK99**]. We start with the definition of *linear fractional transformations* and their equivalence:

Definition 2.4.1 (Vectors, matrices, tensors)**.** We define the following sets:

$$\mathbb{V} := \{\begin{pmatrix} a \\ b \end{pmatrix} \in \mathbb{Z}^2 : a \neq 0 \text{ or } b \neq 0\},$$

$$\mathbb{M} := \{\begin{pmatrix} a & c \\ b & d \end{pmatrix} \in \mathbb{Z}^{2\times 2} : \begin{vmatrix} a & c \\ b & d \end{vmatrix} \neq 0\},$$

$$\mathbb{T} := \{\begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} \in \mathbb{Z}^{2\times 4} : x, y \in \mathbb{R} \text{ exist with } \begin{vmatrix} a\cdot x+e & c\cdot x+g \\ b\cdot x+f & d\cdot x+h \end{vmatrix} \neq 0 \neq \begin{vmatrix} a\cdot y+c & e\cdot y+g \\ b\cdot y+d & f\cdot y+h \end{vmatrix}\},$$

$$\mathbb{V}^+ := \{\begin{pmatrix} a \\ b \end{pmatrix} \in \mathbb{N}^2 : a \neq 0 \text{ or } b \neq 0\},$$

$$\mathbb{M}^+ := \{\begin{pmatrix} a & c \\ b & d \end{pmatrix} \in \mathbb{N}^{2\times 2} : \begin{vmatrix} a & c \\ b & d \end{vmatrix} \neq 0\},$$

$$\mathbb{T}^+ := \{\begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} \in \mathbb{N}^{2\times 4} : x, y \in \mathbb{R} \text{ exist with } \begin{vmatrix} a\cdot x+e & c\cdot x+g \\ b\cdot x+f & d\cdot x+h \end{vmatrix} \neq 0 \neq \begin{vmatrix} a\cdot y+c & e\cdot y+g \\ b\cdot y+d & f\cdot y+h \end{vmatrix}\}.$$

Elements in $\mathbb{V}$, $\mathbb{M}$ and $\mathbb{T}$ are called *vectors*, *matrices* and *tensors* respectively. Notice that we can see a matrix as a pair of vectors and a tensor as a pair of matrices.

Definition 2.4.2 (Linear Fractional Transformations)**.** Let $\begin{pmatrix} a \\ b \end{pmatrix} \in \mathbb{V}$ be a vector, $\begin{pmatrix} c & e \\ d & f \end{pmatrix} \in \mathbb{M}$ be a matrix and $\begin{pmatrix} g & i & k & m \\ h & j & l & n \end{pmatrix} \in \mathbb{T}$ be a tensor. Then for $x \in [0, \infty]$ we define the following *linear fractional transformations*:

$$t\begin{pmatrix} a \\ b \end{pmatrix} = \frac{a}{b},$$

$$t\begin{pmatrix} c & e \\ d & f \end{pmatrix}(x) = \frac{c\cdot x+e}{d\cdot x+f},$$

$$t\begin{pmatrix} g & i & k & m \\ h & j & l & n \end{pmatrix}(x,y) = \frac{g\cdot x\cdot y+i\cdot x+k\cdot y+m}{h\cdot x\cdot y+j\cdot x+l\cdot y+n}.$$

We abbreviate linear fractional transformation with lft.

Definition 2.4.3 (Equivalence)**.** Given two vectors $V, W \in \mathbb{V}$, matrices $M, N \in \mathbb{M}$ and tensors $S, T \in \mathbb{T}$, we call them *equivalent* denoted by in one the following cases:

- $V \sim W$ iff there is a $q \in \mathbb{Q}$ such that $q \cdot V = W$,
- $M \sim N$ iff there is a $q \in \mathbb{Q}$ such that $q \cdot M = N$,
- $S \sim T$ iff there is a $q \in \mathbb{Q}$ such that $q \cdot S = T$.

Notice that the multiplication is the usual scalar multiplication.

Notice that if the determinant of a matrix is zero, then the corresponding linear fractional transformation would be constant. Furthermore, $\sim$ is indeed an equivalence relation on the linear fractional transformations. Our goal is to represent the real numbers with products of lfts. First, we see we can easily compute the range of a linear fractional transformation:

Proposition 2.4.4. *The following equalities hold for matrices* $\begin{pmatrix} a & c \\ b & d \end{pmatrix} \in \mathbb{M}$ *and tensors* $\begin{pmatrix} e & g & i & k \\ f & h & j & m \end{pmatrix}$*:*

$$\operatorname{ran}(t\begin{pmatrix} a & c \\ b & d \end{pmatrix}) = [\min(\{\frac{a}{b}, \frac{c}{d}\}), \max(\{\frac{a}{b}, \frac{c}{d}\})],$$

$$\mathrm{ran}(t\begin{pmatrix} e & g & i & k \\ f & h & j & m \end{pmatrix}) = [\min(\{\frac{e}{f}, \frac{g}{h}, \frac{i}{j}, \frac{k}{m}\}), \max(\{\frac{e}{f}, \frac{g}{h}, \frac{i}{j}, \frac{k}{m}\})].$$

This proposition tells us we can view a lft as a rational interval by considering its range, and therefore we can view a real number as an infinite sequence of matrices. Now we want to turn this sequence into a product of matrices, and therefore we have the following two propositions:

Proposition 2.4.5. *Let $A, B \in \mathbb{M}$ be matrices, then* $\mathrm{ran}(t_A \circ t_B) \subseteq \mathrm{ran}(t_A)$.

Proposition 2.4.6. *Given two matrices $A, B \in \mathbb{M}$, then the range* $\mathrm{ran}(t_A)$ *of A is a subset of the range* $\mathrm{ran}(t_B)$ *of B iff there is a matrix $K \in \mathbb{M}$ such that $A = B \cdot K$.*

The range of a matrix can be seen as an approximation of a real number, and Proposition 2.4.5 says that the approximation becomes more precise if you multiply matrices. Furthermore, if we have an sequence of matrices converging to a real number, we can turn it into a product of matrices with Proposition 2.4.6. This gives us a representation of real numbers. Therefore, we can define the following data type of real numbers:

Definition 2.4.7 (Normal products)**.** We define the following data types of real numbers for $V \in \mathbb{V}$, $V^+ \in \mathbb{V}^+$, $M \in \mathbb{M}$ and $M^+ \in \mathbb{M}^+$:

$$snp := V \mid M(unp)$$
$$unp := V^+ \mid M^+(unp)$$

Now we consider computing with this data type. Since we are working with an infinite data type, we need to consider absorption and emission. First, we need to define several operations on matrices and tensors:

Definition 2.4.8 (Mediant)**.** Given a matrix $\begin{pmatrix} a & c \\ b & d \end{pmatrix} \in \mathbb{M}$, then we define the *mediant*:

$$\overline{\begin{pmatrix} a & c \\ b & d \end{pmatrix}} := \begin{pmatrix} a+c \\ b+d \end{pmatrix}.$$

Definition 2.4.9 (Transpose)**.** The *transpose* of a tensor $\begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} \in \mathbb{T}$ is defined as follows:

$$\begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix}^T := \begin{pmatrix} a & e & c & g \\ b & f & d & h \end{pmatrix}.$$

Definition 2.4.10 (Left and right product)**.** Given a vector $V \in \mathbb{V}$, a matrix $M \in \mathbb{M}$ and a tensor $T = (T_1, T_2) \in \mathbb{T}$ where $T_1, T_2 \in \mathbb{M}$, then we define the *left product* $\cdot_L$ and *right product* $\cdot_R$ as follows:

$$T \cdot_R V := (T_1 \cdot V, T_2 \cdot V),$$
$$T \cdot_R M := (T_1 \cdot M, T_2 \cdot M),$$
$$T \cdot_L V := T^T \cdot_R V,$$
$$T \cdot_L M := (T^T \cdot_R M)^T.$$

Transposition switches the arguments of a tensor T, thus $t_T(x, y) = t_T^T(y, x)$. The following properties are important for absorption:

Proposition 2.4.11. *The following identities hold for vectors $V \in \mathbb{V}$, matrices $M, N \in \mathbb{M}$, tensors $T \in \mathbb{T}$ and $x \in \mathbb{R}$:*

- $t_M(t_V) = t_{M \cdot V}$,
- $t_M(t_N(x)) = t_{M \cdot N}(x)$,
- $t_T(t_V, y) = \begin{cases} \overline{T \cdot_L V} & \text{if } |T \cdot_L V| = 0 \\ t_{T \cdot_L V}(y) & \text{if } |T \cdot_L V| \neq 0 \end{cases}$,
- $t_T(x, V) = \begin{cases} \overline{T \cdot_R V} & \text{if } |T \cdot_R V| = 0 \\ t_{T \cdot_R V}(y) & \text{if } |T \cdot_R V| \neq 0 \end{cases}$,
- $t_T(M(x), y) = (t_{T \cdot_L M})(x, y)$,
- $t_T(x, M(y)) = (t_{T \cdot_R M})(x, y)$.

The normal matrix product is used to absorb from the arguments of a normal product, and the tensor products are used to absorb information from the arguments of a tensor. If the product of a tensor with a vector is singular, then the resulting matrix gives a constant map. Now we consider the emission of information. There are different ways to emit information and we consider the simplest. First we define the *pseudoinverse* of a linear fractional transformation:

Definition 2.4.12 (Pseudoinverse)**.** Given a matrix $\begin{pmatrix} a & c \\ b & d \end{pmatrix} \in \mathbb{M}$, then we define the pseudoinverse:

$$\begin{pmatrix} a & c \\ b & d \end{pmatrix}^{-1} := \begin{pmatrix} d & -c \\ -b & a \end{pmatrix}.$$

If the matrix is invertible, the pseudoinverse is equivalent to the inverse, and the equivalence classes of lfts induced by $\sim$ form a group with the pseudoinverse as inverse. The *tail* and the *head* of a tensor are defined as follows:

Definition 2.4.13 (Head and tail of tensors)**.** Given a tensor $T \in \mathbb{T}$ and assume that $\mathrm{ran}(T) = [\frac{a}{b}, \frac{c}{d}]$, then we define the *head* T^{head} and the *tail* T^{tail}:

$$T^{\mathrm{head}} := \begin{pmatrix} a & c \\ b & d \end{pmatrix},\ T^{\mathrm{tail}} := (T^{\mathrm{head}})^{-1} \cdot T.$$

The head gives the roughest estimate of the result by just considering the range of a tensor. The following proposition tells how the emission works:

Theorem 2.4.14. *The following emission equation holds for every tensor $T \in \mathbb{T}$:*

$$t_T(x, y) \sim t_{T^{\mathrm{head}}}(t_{T^{\mathrm{tail}}}(x, y)).$$

Using the left and right product we can absorb information from one of the arguments of a tensor. However, since a tensor has two arguments, a choice needs to be made. Therefore, we need an *absorption strategy*. With Proposition 2.4.11 we see that absorption of the right argument improves the information of T, and absorption of the left argument improves the information of T^T. This is related to the left and right product. From [**PEE97**] we know that absorption of a tensor $T = (T_0, T_1)$ is better if $\mathrm{ran}(T_0) \cap \mathrm{ran}(T_1) \neq \emptyset$. Therefore, we need to know how to compute whether this property is satisfied or not. This is described in the following proposition:

Proposition 2.4.15. *Let a tensor $T = \begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} \in \mathbb{T}$ be given. Then the intersection of* $\mathrm{ran}(\begin{pmatrix} a & c \\ b & d \end{pmatrix})$ *and* $\mathrm{ran}(\begin{pmatrix} e & g \\ f & h \end{pmatrix})$ *is empty iff the following condition is satisfied*

$$\mathrm{ran}(\begin{pmatrix} d & -c \\ b & -a \end{pmatrix} \cdot \begin{pmatrix} a & g \\ f & h \end{pmatrix}) \subseteq (0, \infty).$$

By computing the range of the product with Proposition 2.4.4 one can see this proposition hold by working out the inequalities. Hence, we can compute whether the information of a tensor's matrices overlap. Now we describe an algorithm for the *absorption strategy*:

Definition 2.4.16 (Tensor Strategy)**.** We define the *tensor strategy* function $\mathrm{strategy} : \mathbb{T} \to \{\mathrm{L}, \mathrm{R}\}$ as follows:

$$\mathrm{strategy}((T_0, T_1)) := \begin{cases} L & \text{if } \mathrm{ran}(T_0) \cap \mathrm{ran}(T_1) = \emptyset \\ R & \text{otherwise} \end{cases}.$$

The symbols L and R say that information from respectively the left and right argument should be absorbed.

Now we consider a data type for real functions. First, we define the data type of *expression trees* which are trees with tensors and matrices as nodes and vectors as leafs:

Definition 2.4.17 (Expression tree)**.** We define *expression trees* as follows:

$$set := V \mid M(uet) \mid T(uet, uet)$$
$$uet := V^+ \mid M^+(uet) \mid T^+(uet, uet)$$

where $V \in \mathbb{V}$, $V^+ \in \mathbb{V}^+$, $M \in \mathbb{M}$, $M^+ \in \mathbb{M}^+$, $T \in \mathbb{T}$ and $T^+ \in \mathbb{T}^+$.

A real function is an expression tree depending on an unknown x. For a large class of expression trees we need some building blocks, namely the basic operations of arithmetic. These are defined with tensors:

Definition 2.4.18 (Arithmetic)**.** We define the following tensors:

$$T_A := \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, T_M := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, T_D := \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, T_S := \begin{pmatrix} 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Notice that the corresponding linear fractional transformation gives the correct function. With these tensors one can define real analytic functions by expressing its power series with the expression tree. Furthermore, polynomials can be defined as a finite expression tree depending on a variable. Therefore, this representation represents a large class of useful functions in analysis.

Now we discuss some examples of real numbers and functions expressed with lfts using the tensors in Definition 2.4.18. We start with exponentiation with natural numbers and then we give a normal product describing $\sqrt{2}$. Finally, we give an expression tree of the exponential function e^x. Our expression tree for e^x differs from the one in [**Pot98**] where continued fraction are used.

Example 2.4.19. Exponentiation with a natural number is repeated multiplying, and an efficient algorithm for this is repeated squaring. This can be implemented as follows:

$$x^0 := \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

$$x^{n+1} := \begin{cases} T_M(T_D(x, \begin{pmatrix} 2 \\ 1 \end{pmatrix})^{\frac{n+1}{2}}, T_D(x, \begin{pmatrix} 2 \\ 1 \end{pmatrix})^{\frac{n+1}{2}}) & \text{if } n+1 \text{ is even} \\ T_M(x^n, x) & \text{if } n+1 \text{ is odd} \end{cases}.$$

Notice that the expression tree is defined recursively, and that the tensor $T_D(x, \begin{pmatrix} 2 \\ 1 \end{pmatrix})$ is equal to $\frac{x}{2}$. Hence, this algorithm computes x^n.

Example 2.4.20. One can prove $\sqrt{2}$ has the following continued fraction expansion:

$$\sqrt{2} = [1; \overline{2}] = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \ldots}}.$$

This continued fraction can be used to describe $\sqrt{2}$ as an infinite normal product. By truncating the expansion into finite pieces, we get the following parts:

$$1 + x,\ 1 + \cfrac{1}{2 + x},\ 1 + \cfrac{1}{2 + \cfrac{1}{2 + x}},\ \ldots.$$

The function $1 + x$ is linear fractional transformation with the matrix $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$, and the other parts are formed by composition with another linear fractional transformation. To acquire the new approximation, $\cfrac{1}{2 + x}$ is subsituted for x, and this is described as a lft with the matrix $\begin{pmatrix} 0 & 1 \\ 1 & 2 \end{pmatrix}$. Therefore, this continued fraction is an infinite composition of lfts, and that gives the following normal product:

$$M_0 := \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, M_i := \begin{pmatrix} 0 & 1 \\ 1 & 2 \end{pmatrix} \text{ for } i > 0, \text{ and } \sqrt{2} = \prod_{i=0}^{\infty} M_i.$$

Notice that this method can be used for every continued fraction. This works for real functions expressed as a continued fraction as well, but we use the power series in the following example.

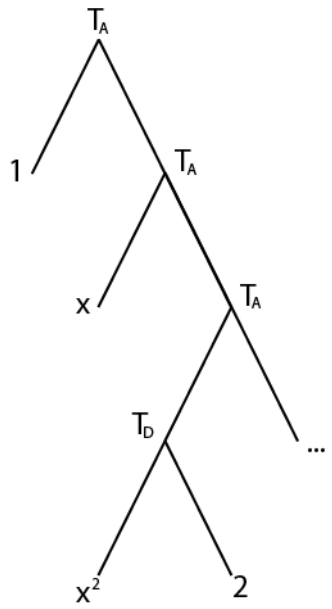

FIGURE 1. The expression tree of e^x

Example 2.4.21. The power series of the exponential function are as follows:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}.$$

This gives an expression tree for e^x, and a truncation of it can be found in Figure 1. It can be continued similarly to get the complete expression tree.

2.5. Differential Equations

A function whose values are not known but whose change is known, can be modeled with *differential equations*. These occur a lot in practice, for example biology and physics. In this section we discuss the basic definitions in the theory of ordinary differential equations and important existence theorems following [**CL55**]. An extension to *boundary value problems* can be given using [**Pat05**], but we refrain to do so. Constructive proofs are prefered, because those give algorithms to find the solution, but these cannot always be given.

Differential equations come in different types depending on the conditions the solution ought to satisfy, and the type gives important information about the existence and uniqueness of the solution. The theorems in this section are just for a specific type.

Definition 2.5.1 (Differential Equation)**.** Given a subset $D \subseteq \mathbb{R}^{n+1}$ and a function $f : D \to \mathbb{R}^n$, we call $u'(t) = f(t, u(t))$ a *differential equation*. A differential equation is called *scalar* iff $n = 1$, and when $n \geq 2$, it is called a *system* of differential equations. Given a certain interval $[t_0, T]$ and a differentiable function $u : [t_0, T] \to \mathbb{R}^n$ defined on that interval, we call u a *solution* of the equation iff $u'(t) = f(t, u(t))$ for every $t \in [t_0, T]$.

Definition 2.5.2 (Initial Value Problem)**.** Given a differential equation $u'(t) = f(t, u(t))$ on a set $D \subseteq \mathbb{R}^{n+1}$ and a point $(t_0, y_0) \in D$, then $u'(t) = f(t, u(t))$, $u(t_0) = y_0$ is called an *initial value problem*. A function $u : [t_0, T] \to \mathbb{R}^n$ is called a solution of the initial value problem on the interval $[t_0, T]$ iff it is a solution of the differential equation and if $u(t_0) = y_0$.

Definition 2.5.3 (Integral Form)**.** The integral form of an initial value problem $u'(t) = f(t, u(t))$, $u(t_0) = y_0$ is $u(t) = y_0 + \int_{t_0}^{t} f(s, u(s))\,\mathrm{d}s$.

The variable t indicates the time and the desired function only depends on the time. Only when certain condition are satisfied, the integral form of an initial value problem has the same solution space as the differential equation. We want to find a solution of the equation, but not every differential equation has a solution. Furthermore, sometimes the solution can only be guaranteed to exist locally. The following theorem, known as the *Cauchy-Peano Existence Theorem*, gives a sufficient condition:

Theorem 2.5.4 (Cauchy-Peano Existence Theorem)**.** *Let a point $(t_0, y_0) \in D$, a rectangle $D = [t_0 - a, t_0 + a] \times [y_0 - b, y_0 + b]$ and a continuous function $f : D \to \mathbb{R}$ be given. Then the initial value problem $u'(t) = f(t, u(t))$, $u(t_0) = y_0$ has a solution.*

Note that this theorem does not say the solution is unique. The proof of the theorem is quite interesting, and hence a sketch is given. Basically, it approximates a solution of the differential equation. First, we notice f is bounded by some $M \in \mathbb{R}$ and is uniformly continuous on D. This can be used to construct a time interval I for the solution such that it is guaranteed to stay in D. Because the derivative is bounded by M as well, we know that the solution is in $[y_0 - b, y_0 + b]$ for $t \in [t_0, t_0 + \frac{b}{M}]$. Therefore, we have to restrict our time interval to $[t_0, t_0 + \min(a, \frac{b}{M})]$.

When a desired accuracy ε is given, we want to construct a function ϕ whose derivatve has at most ε distance from $f(t, \phi(t))$ at any time t. Since f is uniformly continuous, we find $\delta > 0$ such that a difference at most δ between points implies a difference of at most ε between their function values. If our interpolation points differ less than the minimum of δ and $\frac{\delta}{M}$, then linear interpolation will give an ε-precise approximation. Ascoli's lemma states that every uniformly bounded sequence of functions has a uniform convergent subsequence under some conditions, and that gives us a sequence of approximations which converges uniformly to a solution. Because the sequence can have different limit points, uniqueness cannot be guaranteed.

Unfortunately Ascoli's lemma is not constructive and not every sequence of approximations converges uniformly to a solution, and hence this proof does not give an algorithm. Furthermore, uniqueness is a desirable property, and hence we look at differential equations satisfying stronger properties:

Definition 2.5.5 (Lipschitz condition)**.** A function $f : D \to \mathbb{R}$ on a subset $D \subseteq \mathbb{R}^2$ satisfies a *Lipschitz condition* with constant $L \in \mathbb{R}$ iff for every $t, x, x' \in \mathbb{R}$ the inequality $|f(t, x) - f(t, x')| \leq L \cdot |x - x'|$ holds.

When a continuous function satisfies a Lipschitz condition, we can guarantee uniqueness of the solution:

Theorem 2.5.6. *Let a point $(t_0, y_0) \in D$, a rectangle $D = [t_0 - a, t_0 + a] \times [y_0 - b, y_0 + b]$ and a continuous function $f : D \to \mathbb{R}$ which satisfies a Lipschitz condition, be given. Then the initial value problem $u'(t) = f(t, u(t))$, $u(t_0) = y_0$ has a unique solution.*

Notice that this theorem implies that any solution of the integral form is a solution of the differential equation if the initial value problem satisfies a Lipschitz condition. This theorem has an interesting proof as well. Because one can estimate the difference between two approximations of the solutions, we can edit the proof of the Cauchy-Peano theorem slightly to get this result. First of all, Ascoli's lemma is not required to find a convergent sequence, but this can be done constructively. Second of all, the distance between two solutions can be estimated, and equality follows by the Lipschitz condition. There is an alternative method to approximate solutions of differential equations which uses the *Picard operator*:

Definition 2.5.7 (Picard Operator)**.** Given an function $f : D \to \mathbb{R}$ defined on a subset $D \subseteq \mathbb{R}^2$ and a point $(t_0, y_0) \in D$, we define the *Picard operator* P_{f,y_0} on f in y_0 as follows:

$$P_{f,y_0}(g)(t) = y_0 + \int_{t_0}^{t} f(s, g(s)) \, \mathrm{d}s.$$

This operator is related to the initial value problem $u'(t) = f(t, u(t)), u(t_0) = y_0$. By rewriting it to integral form we see that every fixed point of the Picard operator is a solution of the differential equation. If a Lipschitz condition is satisfied, then we can find a solution with an iteration:

Theorem 2.5.8 (Picard-Lindelöf theorem)**.** *Given an initial value problem $u'(t) = f(t, u(t))$, $u(t_0) = y_0$ where f is continuous and satisfies a Lipschitz condition, we define $u_0(x) = y_0$ and $u_{k+1}(x) = P_{f,y_0}(u_k)(x)$. Then the sequence $(u_n)_{n \in \mathbb{N}}$ converges uniformly to the unique solution of the initial value problem.*

This is proved with the famous fixed point theorem by Banach as in [**GG83**]. Again the integral equation is considered. The idea is that when you restrict the domain, the Picard operator is around (t_0, y_0) a strict contraction, and hence it has a fixed point. However, a restriction in the time is required to guarantee it maps the space in itself which is required for a contraction. The time restriction can be found by working out the integrals.

2.6. Domain Theory

Domain theory originated from ideas of Dana Scott to develop models for the λ-calculus, and it has been used for the foundations of analysis as well [**ES99, EL04**]. The basic structures in domain theory are *partially ordered sets* which are sets with a specific relation. This relation is an order in information, and says which elements contain more information than other elements. With domain theory an alternative model of computation with real numbers can be described, and the computability notions are equivalent to Type-2 theory of effectivity [**Wei00**]. Therefore, it can be used to describe good algorithms for problems in analysis.

In this section we review several definitions from [**ES99, EL04, AJ94**] and we start with the notion of a partially ordered set:

Definition 2.6.1 (Partially Ordered Set)**.** Let P be a set and let $\sqsubseteq$ be a relation on P, then we call $(P, \sqsubseteq)$ a *partially ordered set* iff the following conditions are satisfied:

- $\sqsubseteq$ is reflexive,
- $\sqsubseteq$ is antisymmetric,
- $\sqsubseteq$ is transitive.

If an element $y \in P$ and a subset $A \subseteq P$ are given, we write $A \sqsubseteq y$ iff $x \sqsubseteq y$ for every $x \in A$. A partially ordered set is called *pointed* iff there is an element $\bot \in P$ such that $\bot \sqsubseteq y$ for every $y \in P$.

Often we abbreviate partially ordered set to poset. We require the following notation:

Definition 2.6.2 (Upper set and lower set)**.** Let $(P, \sqsubseteq)$ be a poset and $A \subseteq P$ be a subset, then we define the *upper set* and *lower set* of A as follows:

$$\uparrow A := \{y \in P : \text{there is a } x \in A \text{ such that } x \sqsubseteq y\},$$
$$\downarrow A := \{x \in P : \text{there is a } y \in A \text{ such that } x \sqsubseteq y\}.$$

When $x \in P$ is an element, we write $\uparrow x := \uparrow \{x\}$. Similary for $\downarrow x$.

Definition 2.6.3 (Supremum)**.** Let a poset $(P, \sqsubseteq)$ and a subset $A \subseteq P$ be given. The *supremum* or *least upper bound* of A is any element $y \in P$ such that $A \sqsubseteq y$ and if $z \sqsubseteq A$ for any $z \in P$, then $y \sqsubseteq z$. This is denoted by $\bigsqcup A = y$.

In partially ordered sets suprema are unique, and hence $\bigsqcup$ is a partial function. However, there are posets in which some subsets do not have a least upper bound. Because suprema are of interest, it would be convenient if certain susets always have a supremum. Posets in which every *directed set* has a supremum, are called a *dcpo*:

Definition 2.6.4 (Directed sets)**.** A subset A of a partially ordered set $(P, \sqsubseteq)$ is called *directed* iff for every $x, y \in A$ there is a $z \in A$ such that $x \sqsubseteq z$ and $y \sqsubseteq z$.

Definition 2.6.5 (Directed Complete Partial Order)**.** A poset $(P, \sqsubseteq)$ is called a *Direct Complete Partial Order* (dcpo for short) iff every directed set has a supremum.

Notice that every dcpo is pointed, because the least element $\bot$ is the supremum of the empty set which is directed. This definition is not very natural, because directed sets can be complicated. Simpler subsets are so-called *chains* which are increasing sequences of elements:

Definition 2.6.6 (Chain)**.** Given a partially ordered set $(P, \sqsubseteq)$ and a sequence $(p_i)_{i \in \mathbb{N}}$ in P, then we call $(p_i)_{i \in \mathbb{N}}$ a *chain* iff for every $i \in \mathbb{N}$ the condition $p_i \sqsubseteq p_{i+1}$ is satisfied.

Chains model evaluations of programs and computations. Every element of the sequence is the information after a step of the program or computation. Since no information is lost, the sequence is increasing. An equivalent definition of dcpos uses chains:

Proposition 2.6.7. *A poset is a dcpo iff it is pointed and every chain has a least upper bound.*

Now we are going to look at the notion of approximation which is captured in the *way-below relation*. Basically, we say x is way-below y iff every computation whose result contains at least the information of y, requires the information of x. This definition needs the notion of *directed sets*:

Definition 2.6.8 (Way-below relation)**.** Given a dcpo $(P, \sqsubseteq)$ and two elements $x, y \in P$, then we say that x is *way-below* y iff for every directed subset $A \subseteq P$ with $y \sqsubseteq \bigsqcup A$ there exist an element $a \in A$ such that $x \sqsubseteq a$. We denote this by $x \ll y$. If $M \subseteq P$, we say $M \ll y$ iff every $x \in M$ is way-below y.

Notice that if x is way-below y, then $x \sqsubseteq y$. Now we introduce some extra notation, namely the *upper way-below set* and the *lower way-below set*:

Definition 2.6.9 (Upper and lower way-below set)**.** Given a poset $(P, \sqsubseteq)$ and a subset $A \subseteq P$, we define the *upper way-below set* and *lower way-below set* of A as follows:

$$\Uparrow A := \{y \in P : \text{there is a } x \in A \text{ such that } x \ll y\},$$

$$\Downarrow A := \{x \in P : \text{there is a } y \in A \text{ such that } x \ll y\}.$$

For $x \in P$ we write $\Uparrow x := \Uparrow \{x\}$. Similary for $\Downarrow x$.

It is often convenient to describe a dcpo with a smaller set of elements which can be compared to the notions of a base and subbase in topology. However, not every subset can be used, but only those which can approximate every element of the dcpo. This gives rise to the notion of a *basis*:

Definition 2.6.10 (Basis)**.** A subset $B \subseteq P$ of a dcpo $(P, \sqsubseteq)$ is called a *basis* iff for every $x \in P$ the subset $B \cap \Downarrow x$ contains a directed subset with x as supremum.

Definition 2.6.11 (Continuous Domain)**.** A dcpo is called a *continuous domain* iff there is a basis.

Notice that the sets $B_x := B \cap \Downarrow x$ are directed. Let $A \subseteq B_x$ be directed with supremum x. Every two elements from B_x have upper bounds in A, because they are way-below x. Then with directedness of A one can show B_x is directed. A useful property is the *interpolation property*:

Proposition 2.6.12 (Interpolation property)**.** *Let $(P, \sqsubseteq)$ be a continuous domain with a basis B, $y \in P$ an element and let $M \subseteq P$ be a finite subset of P such that $M \ll y$. Then there exists a $x \in B$ such that $M \ll x \ll y$.*

The interpolation property can be proven by considering the set $A := \{p \in B : \text{there is a } p' \in P \text{ with } p \ll p' \ll y\}$. One can prove A is non-empty, directed and $\bigsqcup A = y$. Hence, for every $m \in M$ we can find $a_m \in A$ with $m \sqsubseteq a_m$. Using directedness we can find an upper bound a for those a_m in A, and this element can be used to find the desired $x \in B$.

Now we consider continuous functions between dcpos and there are two ways to do this. On one hand, we can define a topology on arbitrary dcpos. On the other hand, one can define continuity purely in domain-theoretic terms. We take the first approach:

Definition 2.6.13 (Scott topology)**.** The *Scott topology* on a dcpo $(P, \sqsubseteq)$ is defined as follows:

$$\tau_S := \{O \subseteq P : O = \uparrow O, \text{if } A \subseteq P \text{ such that } \bigsqcup A \in O, \text{ then } A \cap O \neq \emptyset\}.$$

Scott open sets are those sets which are *upward closed* and *inaccessible by suprema*. When an intermediate result of a computation is in an open set O, then we know the result is in that set, because O is upward closed. The converse can be proven by considering the closed set $\downarrow (f(\bigsqcup A))$. Since open sets are inaccessible by suprema, an intermediate result in the open set is required to end in O. Examples of open sets are $\Uparrow x$, and examples of closed sets are $\downarrow x$. Continuity can be characterized in algebraic terms:

Proposition 2.6.14. *Given two dcpos $(P, \sqsubseteq_P)$, and $(Q, \sqsubseteq_Q)$ and a function $f : P \to Q$, then f is continuous iff f preserves suprema of directed subsets.*

Notice that the second formulation implies that f preserves the order which can be used to prove the inverse image of an upward closed set is upward closed. A direct corollary from this lemma is that continuous functions between dcpos are monotone. Next we discuss *function spaces* which are a way to construct new dcpos.

Definition 2.6.15 (Function Space)**.** Let two dcpos $(P \sqsubseteq_P)$ and $(Q, \sqsubseteq_Q)$ be given. Denote the set of continuous functions from P to Q as $[P \to Q]$. We define the following order on $[P \to Q]$:

$$f \sqsubseteq g \text{ iff for every } x \in P \; f(x) \sqsubseteq_Q g(x).$$

We define the *function space* from P to Q as $([P \to Q], \sqsubseteq)$.

Functions are ordered pointwise. Notice that the function space is a poset and even a dcpo. Our goal is to prove the function space is a continuous domain. Therefore we define *step functions*:

Definition 2.6.16 (Single-step function)**.** Let two dcpos $(P, \sqsubseteq_P)$ and $(Q, \sqsubseteq_Q)$ and two elements $p \in P$ and $q \in Q$ be given. Then we define the *single-step function* $p \searrow q : P \to Q$ as follows:

$$(p \searrow q)(x) := \begin{cases} q & p \ll x \\ \bot_Q & \text{otherwie} \end{cases}.$$

Definition 2.6.17 (Consistency)**.** Let two dcpos $(P, \sqsubseteq_P)$, $(Q, \sqsubseteq_Q)$ and a finite set $S = \{a_i \searrow b_i : i \in I\}$ of single-step functions be given. Then we say S is *consistent* iff for every subset $J \subseteq I$ such that an $a \in P$ with $\{a_j : j \in J\} \ll a$ exists, then there is a $b \in Q$ such that $\{b_j : j \in J\} \sqsubseteq b$.

Definition 2.6.18 (Step function)**.** A *step function* is the supremum of a finite set of consistent single-step functions.

Step functions are continuous, and they form a basis of $[P \to Q]$. Therefore, we can conclude that the function space between continuous dcpos is a continuous dcpo as well. Now we consider two examples. We start with the *interval domain* of the real numbers, and we characterize several notions. This is our basic domain-theoretic model of the real numbers. In the second example we consider the real numbers with the trivial order. Both are interpreted as data types for computations.

Example 2.6.19. We define the *interval domain* as the set $\mathrm{I}\mathbb{R} := \{[a, b] : a, b \in \mathbb{R}\} \cup \{\mathbb{R}\}$ of all connected compact subsets of $\mathbb{R}$ and $\mathbb{R}$ ordered by reverse inclusion. This is a poset, and with properties of the real numbers we can prove it is a dcpo. Notice that $\mathbb{R}$ is the least element of this dcpo. The elements $\{x\}$ for any $x \in \mathbb{R}$ are maximal. Furthermore, the supremum of two elements is the intersection of intervals.

The way-below relation can be characterized using the interior of a set. Given two compact connected subsets $I, J \subseteq \mathbb{R}$, then $I \ll J$ iff $J \subseteq \mathrm{int}(I)$. It is easy to see that $J \subseteq \mathrm{int}(I)$ imples $I \ll J$. For the reverse let $j \in J$, and consider $\{[j - \frac{1}{n}, j + \frac{1}{n}] : n \in \mathbb{N}\}$. Since the supremum of it is contained in J and $I \ll J$, there must be a $n \in \mathbb{N}$ such that $[j - \frac{1}{n}, j + \frac{1}{n}] \subseteq I$. Hence, $j \in \mathrm{int}(I)$.

With this characterization we see that $\twoheaduparrow [a, b] = \{[c, d] : [c, d] \subseteq (a, b)\}$ and $\twoheaddownarrow [a, b] = \{[c, d] : [a, b] \subseteq (c, d)\}$. We use this to find a basis for the domain and a base for the Scott topology. A possible basis is the set $\{[p, q] : p, q \in \mathbb{Q}\}$ of rational intervals. Let an arbitrary real interval $x := [a, b]$ be given. First we notice that $x \subseteq \bigsqcup(B \cap \twoheaddownarrow x)$. Because a is a real number, there is an increasing sequence $(p_i)_{i \in \mathbb{N}}$ of rational numbers which converges to a. Similary, we can find a decreasing sequence $(q_i)_{i \in \mathbb{N}}$ in $\mathbb{Q}$ which converges to q. This gives $\bigcap_{i \in \mathbb{N}} [p_i, q_i] = x$, and thus $x = \bigsqcup(B \cap \twoheaddownarrow x)$ and thus B is a basis for $\mathrm{I}\mathbb{R}$.

The family $\{\twoheaduparrow [p, q] : p, q \in \mathbb{Q}\}$ is a base of the Scott topology on $\mathrm{I}\mathbb{R}$. First notice that every $\twoheaduparrow [p, q]$ is open which follows from the interpolation property. Second we prove it satisfies the property in Definition 2.2.3. Let $O \in \tau_S$ be any open set containing a point $I \in \mathrm{I}\mathbb{R}$. Because the rational intervals form a basis of $\mathrm{I}\mathbb{R}$, there is a sequence $[p_i, q_i]$ such that $\bigcap_{i \in \mathbb{N}} [p_i, q_i] = a$. We can find a rational interval $J := [p_j, q_j]$ in that sequence which is an element of O, because O is an open set. Now we see that $\twoheaduparrow J$ is contained in O, because $O = \uparrow O$. Hence, the family of compact rational intervals is a base for the Scott topology on $\mathrm{I}\mathbb{R}$.

Another useful property is that every real function can be extended to a Scott continuous real map in $\mathrm{I}\mathbb{R}$. If a function $f : \mathbb{R} \to \mathbb{R}$ is given, then we define the extension $\tilde{f} : \mathrm{I}\mathbb{R} \to \mathrm{I}\mathbb{R}$ with $\tilde{f}(A) = f(A)$.

Notice that this domain can be restricted to a given compact interval by intersecting every interval such that similar properties hold. This is a realistic data type of real numbers. Since real numbers are infinite objects, weonly know a finite amount of information about it at any time. Therefore, only approximations are available, and those can be viewed as a rational interval.

Example 2.6.20. We can restrict $\mathrm{I}\mathbb{R}$ to the singleton sets, and this is essentially the same as $\mathbb{R}$ with the trivial order. Upper way-below sets $\uparrow \{x\} = \{\{x\}\}$ are inaccessible by suprema, and hence those sets are open. Since these are singleton sets, the Scott topology is discrete.

The second data type of the real numbers assumes perfect information is available, and it is therefore not the basic type of real number computation. However, it is possible that complete information about the input is available.

CHAPTER 3

Polynomial representation of real functions

In Examples 2.3.30 and 2.3.31 we discussed two different representations for functions. The idea of these representations was to approximate functions with simple functions which is similar to the approach in Appendix A. The goal of this chapter is to prove that the Cauchy representation with polynomials is admissible and therefore equivalent to Cauchy representation with polygons.

We firstly consider *effective uniformly continuous* functions. These are functions whose error can be computed uniformly over the domain. This property is useful for computing, because when we want to compute the result with a certain error, we can calculate which precision is required for the input independently from the input. Secondly, we discuss *pointwise computable functions* which are functions that map computable real numbers to computable real numbers. Both classes of functions satisfy useful properties. Thirdly, we discuss Bernstein polynomials which can be used to approximate arbitrary continuous functions with polynomials. At last we use these techniques to give the desired proof.

3.1. Effective Uniform Continuity and Pointwise Computability

Computability has been defined by Pour-El and Richard as *effective uniformly continuous* and *sequentially computable* functions. These notions can be used for our goal, but we use the equivalent notion of *pointwise computability* instead of sequential computability. The following definition is taken from [**PER83**]:

Definition 3.1.1 (Effective uniformly continuous functions)**.** Let $f : [a,b] \to \mathbb{R}$ be a real function. Then f is called *effective uniformly continuous* iff there is a recursive function $\delta : \mathbb{N} \to \mathbb{N}$ such that for any $n \in \mathbb{N}$ and $x, y \in [a,b]$ with $|x-y| < \frac{1}{\delta(n)}$:

$$|f(x)-f(y)| < 10^{-n}.$$

An important fact is that effective uniformly continuous functions which are defined on a compact interval, form a vector space. That means they are closed under addition and scalar multiplication:

Proposition 3.1.2. *Let $f : [a,b] \to \mathbb{R}$ and $g : [a,b] \to \mathbb{R}$ be effective uniformly continuous functions on an interval $[a,b]$. Then the scalar product $c \cdot f$ for every $c \in \mathbb{R}$ and the sum $f+g$ are effective uniformly continuous.*

PROOF. Let two effective uniformly continuous functions $f : [a,b] \to \mathbb{R}$ and $g : [a,b] \to \mathbb{R}$ and a real number $c \in \mathbb{R}$ be arbitrary. Because f is effective uniformly continuous, there is a function δ_f such that for every $x, y \in [a,b]$ with $|y-x| < \frac{1}{\delta_f(m)}$ the following inequality holds $|f(x)-f(y)| < 10^{-m}$. Also, we can find a function δ_g with the same property for g. We know there is a $k \in \mathbb{N}$ such that $c < 10^k$. Define $\delta_{c\cdot f}(m) := \delta_f(m+k)$. Then for $x, y \in [a,b]$ with $|y-x| < \frac{1}{\delta_{c\cdot f}(m)} = \frac{1}{\delta_f(m+k)}$:

$$|c \cdot f(y) - c \cdot f(x)| = |c| \cdot |f(y)-f(x)| < 10^k \cdot 10^{-m-k} = 10^{-m}.$$

Therefore, the function $c \cdot f$ is effective uniformly continuous.

Now we prove the sum of f and g is effective uniformly continuous. Therefore, we define $\delta_{f+g}(m) := \min\{\delta_f(m-1), \delta_g(m-1)\}$. Then for every $x, y \in [a,b]$ with $|y-x| < \frac{1}{\delta_{f+g}(m)}$ we know $|f(y)-f(x)| < 10^{-m-1}$ and $|g(y)-g(x)| < 10^{-m-1}$. Therefore, the following inequality holds:

$$|f(y)+g(y)-f(x)-g(x)| \le |f(y)-f(x)| + |g(y)-g(x)| < 2 \cdot 10^{-m-1} < 10^{-m}.$$

Hence, the sum $f+g$ is effective uniformly continuous. □

Now we prove that certain class of functions are effective uniformly continuous using the well-known *Mean Value Theorem* from analysis [**Rud64**]:

Theorem 3.1.3 (Mean Value Theorem)**.** *Let $f : [a,b] \to \mathbb{R}$ be a continuous real function which is differentiable on (a,b). Then there is an $x \in (a,b)$ such that:*

$$\frac{f(b)-f(a)}{b-a} = f'(x).$$

The following corollary is obvious:

Corollary 3.1.4. *For every continuous real function $f : [a,b] \to \mathbb{R}$ which is differentiable on (a,b), the following inequality holds:*

$$|f(b)-f(a)| \leq (b-a) \cdot \sup\{|f'(x)| : x \in [a,b]\}.$$

Therefore, it is sufficient to find a computable bound for the derivative to prove effective uniform continuity, because the bound can be used with the Mean Value Theorem to prove the continuity. We start with functions x^n:

Proposition 3.1.5. *The function $f_n : [a,b] \to \mathbb{R}$ with $f_n(x) = x^n$ is effective uniformly continuous for every $n \in \mathbb{N}$.*

PROOF. Let $n \in \mathbb{N}$ be given. We need to compute a bound of $n \cdot x^{n-1}$ on the interval $[a,b]$. Assume, $n = 0$. That means the function is constant, and therefore we can choose $\delta(n) := 1$.

Assume, $n \neq 0$. Let $c := \max\{|a|,|b|\}$ be the endpoint with the greatest absolute value. Define $\delta(m) := \lfloor \frac{n \cdot c^{n-1}}{10^{-m}} \rfloor$. Let $m \in \mathbb{N}$ and $x, y \in [a,b]$ be given such that $|y-x| < \frac{1}{\delta(m)}$. We can assume without loss of generality that $x < y$. If we restrict f to the interval $[x,y]$, we see with the Mean Value Theorem:

$$|f(y)-f(x)| \leq (y-x) \cdot n \cdot c^{n-1} \cdot 10^{-m} < \frac{10^{-m}}{n \cdot c^{n-1}} \cdot n \cdot c^{n-1} < 10^{-m}. \qquad \square$$

Now we conclude that polynomials on a closed interval are effective uniformly continuous, because they are a linear combinations of non-negative exponents of the identity function. The result can easily be extended to trigonometric polynomials:

Proposition 3.1.6. *The function $f : [a,b] \to \mathbb{R}$ with $f(x) = \sin(n \cdot x)$ is effectively uniformly continuous for every $n \in \mathbb{N}$.*

PROOF. Let $n \in \mathbb{N}$ be given. If $n = 0$, then the function is constant, and there is nothing left to prove. Assume, $n \neq 0$. We define $\delta(m) := n \cdot 10^m$. Then the following inequality holds for every $m \in \mathbb{N}$ and $x, y \in [a,b]$ with $x < y$ and $y - x < \frac{1}{\delta(m)}$:

$$\sin(n \cdot y) - \sin(n \cdot x) \leq \sup\{n \cdot \cos(z) : z \in [x,y]\} \cdot (y-x) < \frac{n}{n \cdot 10^m} = 10^{-m}. \qquad \square$$

Similary we can prove that $f(x) = \cos(n \cdot x)$ is effective uniformly continuous for every $n \in \mathbb{N}$, and therefore trigometric polynomials are effective uniformly continuous too. Furthermore, rational polygons satisfy this property, but the proof requires more work:

Proposition 3.1.7. *Let $f : [a,b] \to \mathbb{R}$ be a rational polygon. Then f is effective uniformly continuous.*

PROOF. Given a rational polygon $f : [a,b] \to \mathbb{R}$ which is the result of linear interpolation of two rational vectors $p, q \in \mathbb{Q}^k$ with $k \in \mathbb{N}$. Define s to be the maximal slope:

$$s := \max\{|\frac{q(i+1)-q(i)}{p(i+1)-p(i)}| : i \in \mathbb{N}, 0 < i < k\}.$$

If $s = 0$, then the polygon is constant, so we assume $s \neq 0$. Firstly we are going to prove that for every $x, y \in [a,b]$ with $x < y$ the inequality $|f(y)-f(x)| \leq s \cdot |y-x|$ holds. We know there are two natural numbers $i, j \in \mathbb{N}$ with $i \leq j$ such that $x \in [p_i, p_{i+1}]$ and $y \in [p_j, p_{j+1}]$. Using $f(x) = q_i + \frac{q_{i+1}-q_i}{p_{i+1}-p_i} \cdot (x-p_i) \leq q_i + s \cdot (x-p_i)$ we can compute:

$$q_i - f(x) = -\frac{q_{i+1}-q_i}{p_{i+1}-p_i} \cdot (x-p_i) = \frac{q_{i+1}-q_i}{p_{i+1}-p_i} \cdot (p_i - x) \leq s \cdot (p_i - x).$$

By using telescoping series on the previous inequality, we obtain:

$$q_j = f(x) - f(x) + q_i + q_j - q_i = f(x) + (q_i - f(x) + \sum_{n=i}^{j-1}(q_{n+1} - q_n))$$

$$\leq f(x) + s \cdot (p_i - x + \sum_{n=i}^{j-1}(p_{n+1} - p_n)) = f(x) + s \cdot (p_j - x).$$

Now we obtain the following:

$$f(y) = q_j + \frac{q_{j+1} - q_j}{p_{j+1} - p_j} \cdot (y - p_j) \leq f(x) + s \cdot (p_j - x) + s \cdot (y - p_j) = f(x) + s \cdot (y - x).$$

Similary, we can prove $f(y) \geq f(x) - s \cdot (y - x)$, thus we can conclude $f(y) - f(x) \geq -s \cdot (y - x)$ and $f(y) - f(x) \leq s \cdot (y - x)$. Therefore, we know $|f(y) - f(x)| \leq s \cdot (y - x)$.

Now we are going to construct δ. Because there is a $k \in \mathbb{N}$ such that $s < 10^k$, we define $\delta(m) = 10^k \cdot 10^m$ for $m \in \mathbb{N}$. Then for every $m \in \mathbb{N}$ and $x, y \in [a, b]$ with $|y - x| < \frac{1}{\delta(m)}$:

$$|f(y) - f(x)| \leq s \cdot |y - x| < 10^k \cdot \frac{1}{10^k \cdot 10^m} = 10^{-m}. \qquad \square$$

We know from classical analysis that uniformly continuous functions are bounded on any compact interval. In the following proposition we discuss this result for effective uniformly continuous functions which are computable in one point:

Proposition 3.1.8. *Every effective uniformly continuous function $f : [a, b] \to \mathbb{R}$ on an interval $[a, b]$ such that $f(x)$ is computable for some $x \in [a, b]$, has a computable rational upper bound.*

PROOF. Let an arbitrary function $f : [a, b] \to \mathbb{R}$ be given such that f is effective uniformly continuous and $f(x)$ is computable for some $x \in [a, b]$. Firstly, we give a overestimate of $|f(x)|$. Since we know there is a rational interval (p, q) of length 1 containing $|f(x)|$, the number q is greater than $|f(x)|$. Using effective uniform continuity, we construct an upper bound. We have a function $\delta : \mathbb{N} \to \mathbb{N}$ satisfying the property of Definition 3.1.1, and we can compute our stepsize $\tau := \delta(0)$. We partition our interval $[a, b]$ in n smaller intervals $[a_i, b_i]$ of length $\frac{\tau}{2}$, and assume that $x \in [a_i, b_i]$ for some $i \in \mathbb{N}$. We define our upper bound $M := q + n + 2$.

Now we take any $y \in [a, b]$ and our goal is to prove $|f(y)| \leq M$. There is a $j \in \mathbb{N}$ such that $y \in [a_j, b_j]$, and now we compare j with i. If $j = i$, then we know $||f(x)| - |f(y)|| < 2$, thus $|f(y)| < |f(x)| + 2 < q + n + 2$. Now we assume $i < j$. We know that for any k the inequality $||f(x_{k+1})| - |f(x_k)|| \leq |f(x_{k+1}) - f(x_k)| < 1$ holds, and with this we see that for any $k, l \in \mathbb{N}$ with $l \geq 1$:

$$|f(x_{k+l})| - |f(x_k)| = \sum_{m=1}^{l} |f(x_{k+m})| - |f(x_{k+m-1})| < \sum_{m=1}^{l} 1 = l,$$

$$|f(y)| < |f(x_j)| + 1 < |f(x_i)| - i + j + 1 < |f(x)| - i + j + 2 < q + n + 2 = M.$$

By working in the reverse direction, we can prove it holds for the remaining case $j < i$ as well. $\square$

By considering $-f$ we can find a computable rational lower bound of the function f with this proposition, and hence those functions are bounded. At last, we prove the vector space of effective uniformly continuous functions is closed under effective limits:

Theorem 3.1.9. *Let $a, b \in \mathbb{R}$ be real numbers and let for every $n \in \mathbb{N}$ a function $f_n : [a, b] \to \mathbb{R}$ be given. If f_n converges effectively to some function $f : [a, b] \to \mathbb{R}$, then f is effective uniformly continuous.*

PROOF. Let $a, b \in \mathbb{R}$ be real numbers and let $f_n : [a, b] \to \mathbb{R}$ be a sequence of effective uniformly continuous functions. We assume it converges effectively to some function $f : [a, b] \to \mathbb{R}$. Let $m \in \mathbb{N}$ be given. Because the sequence convergens effectively to f, there is a recursive function $\varepsilon : \mathbb{N} \to \mathbb{N}$ such that for every

natural number $n \geq \varepsilon(m+1)$ the inequality $|f_n(x) - f(x)| < 10^{-m-1}$ holds for every $x \in [a,b]$. Furthermore, we can find $\delta_{f_{m+1}}$ as in Definition 3.1.1, because f_{m+1} is effective uniformly continuous. We define $\delta_f(m) = \delta_{f_{m+1}}(m+1)$, so the following inequality holds for $x, y \in [a,b]$ with $|y-x| < \frac{1}{\delta_f(m)}$:

$$|f(y) - f(x)| \leq |f(y) - f_m(y)| + |f_m(y) - f_m(x)| + |f_m(x) - f(x)| < 3 \cdot 10^{-m-1} < 10^{-m}.$$

Therefore, f is effective uniformly continuous. □

Now we discuss *pointwise computable* functions. The following definition is based on [**PER83**]:

Definition 3.1.10 (Pointwise computable functions)**.** A function $f : [a,b] \to \mathbb{R}$ is called *pointwise computable* iff for every computable real number $c \in \mathbb{R}$ the image $f(c)$ is computable too.

A real number is computable iff there is a computable sequence of rational intervals converging to it which is equivalent to the definition in Example 2.3.23. We want to show polynomials are pointwise computable functions. Therefore, we firstly notice the identity and the function with constant value 1 are computable. Secondly we prove that the pointwise computable functions form a $\mathbb{R}$-algebra, which means they are closed under (scalar) multiplication and addition. With Proposition 2.3.24 we see:

Corollary 3.1.11. *If $f : [a,b] \to \mathbb{R}$ and $g : [a,b] \to \mathbb{R}$ are pointwise computable functions and $c \in \mathbb{R}$ is a computable real number, then the functions $f+g$, $f \cdot g$ and $c \cdot f$ are pointwise computable functions.*

In addition the space of pointwise computable functions is closed under effective limits:

Proposition 3.1.12. *Let $f_n : [a,b] \to \mathbb{R}$ be a sequence of pointwise computable functions. If f_n converges effectively to some function $f : [a,b] \to \mathbb{R}$, then f is pointwise computable as well.*

PROOF. Assume, we have a sequence $f_n : [a,b] \to \mathbb{R}$ of pointwise computable functions which converges effectively to some function $f : [a,b] \to \mathbb{R}$. Let $x \in [a,b]$ be a computable real number. We have a function $\varepsilon : \mathbb{N} \to \mathbb{N}$ such that for every $n, m \in \mathbb{N}$ with $m \geq \varepsilon(n)$ the equality $|f_m(y) - f(y)| < 10^{-n}$ holds for every $y \in [a,b]$. Now we want to construct a sequence of rational intervals around $f(x)$.

Let $n \in \mathbb{N}$ be given. Then we know $|f_{\varepsilon(2\cdot n)}(x) - f(x)| < 10^{-2\cdot n}$. Because x is a computable real number, we can find an rational open interval (p_n, q_n) containing the image $f_{\varepsilon 2\cdot n}(x)$ such that $|q_n - p_n| < 10^{-2\cdot n}$. We see:

$$|p_n - f(x)| \leq |p_n - f_{\varepsilon(2\cdot n)}(x)| + |f_{\varepsilon(2\cdot n)}(x) - f(x)| < 2 \cdot 10^{-2\cdot n} < 10^{-n}.$$

Hence, the rational interval $(p_n, p_n + \frac{1}{n})$ contains $f(x)$, and its length is $\frac{1}{n}$. Therefore, we can construct a sequence of shrinking nested rational intervals containing $f(x)$, sp $f(x)$ is a computable number. We conclude f is pointwise computable. □

The previous proposition uses the fact that uniform convergence implies pointwise convergence. However, pointwise limits are not considered, because then the error function ε should depend on a computable real number.

3.2. Bernstein Polynomials

In this section we use Bernstein polynomials to prove a computable version of Weierstrass' approximation theorem. Weierstrass' approximation theorem states that every continuous function can be uniformly approximation by polynomials. Normally Bernstein polynomials are used for a constructive proof for this theorem. Our goal is to prove every computable function can be effectively approximated with polynomials, and hence a different proof is needed. Most lemmas and definitions are from [**Phi03**]. Furthermore, without loss of generality we can approximate functions on $[0,1]$.

Definition 3.2.1 (Space of polynomials)**.** For every interval $[a,b]$ we define the *space of polynomials* on $[a,b]$ as follows:

$$\mathrm{P}([a,b] \to \mathbb{R}) := \mathrm{L}(\{\lambda x.x^n : n \in \mathbb{N}\}).$$

The definition says polynomials are finite linear combinations of functions x^n for $n \in \mathbb{N}$n. Every function f has corresponding *Berstein polynomials*, and we define the following operator:

Definition 3.2.2 (Bernstein operator)**.** We define the *Bernstein operator* B_n for every $n \in \mathbb{N}$ and $f : [0,1] \to \mathbb{R}$ as follows:

$$(B_n(f))(x) := \sum_{r=0}^{n} f(\frac{r}{n}) \binom{n}{r} \cdot x^r \cdot (1-x)^{n-r}.$$

This is not always the most convenient way to describe this operator. Another approach uses the *forward step functional* which is similar to derivatives:

Definition 3.2.3 (Forward Step Functional)**.** For any $h \in \mathbb{R}$ with $h > 0$ we define the *forward step functional* by:

$$(\Delta_h(f))(x) := f(x+h) - f(x).$$

One can easily see the Bernstein operator and the forward step functional are linear. Besides, repeated application of the forward step functional gives the following expansion:

Lemma 3.2.4. *For every $r \in \mathbb{N}$ and $h \in \mathbb{R}$ with $h > o$ the following equality holds:*

$$\Delta_h^r(f) = \sum_{i=0}^{r} (-1)^{r-i} \cdot \binom{r}{i} \cdot f(x + i \cdot h).$$

With the help of this lemma we can prove that the Bernstein operator can be expressed with the forward step functional:

Lemma 3.2.5. *For every $n \in \mathbb{N}$ with $n > 0$ the following equality holds:*

$$(B_n(f))(x) = \sum_{r=0}^{n} \binom{n}{r} \cdot f(0) \cdot \Delta_{\frac{1}{n}}^r(\lambda x.x^r).$$

Expanding $(1-x)^r$ in Definition 3.2.2 gives a double series. By changing the order of summation, one can obtain the expansion of Lemma 3.2.5 which completes the proof.

Now we compute the result of the Bernstein on several simple functions:

Lemma 3.2.6. *For the functions $f(x) = c$, $I(x) = x$ and $g(x) = x^2$, the result of the Bernstein operator with any $n \in \mathbb{N}$ is as follows:*

$$(B_n(f))(x) = c,\ (B_n(I))(x) = x,\ (B_n(g))(x) = x^2 + \frac{1}{n} \cdot x \cdot (1-x).$$

Now we give a weak computable version of the Weierstrass theorem which only consider effective uniformly continous functions which are pointwise computable. It is enough to prove a stronger version for every computable function. Without loss of generality we assume the domain is the closed interval $[0,1]$. The proof is based on the proof in [**Phi03**].

Theorem 3.2.7 (Weak computable Weierstrass Theorem)**.** *For every effective uniformly continuous function $f : [0,1] \to \mathbb{R}$ which is pointwise computable, there exists a sequence of Bernstein polynomials which converges effectively to f.*

PROOF. Let an arbitrary effective uniformly continuous function $f : [0,1] \to \mathbb{R}$ and assume f is pointwise computable. Using Proposition 3.1.8 we can find a $M \in \mathbb{Q}$ such that $|f(x)| \leq M$ for every $x \in [a,b]$. Furthermore, we can find a function $\delta : \mathbb{N} \to \mathbb{N}$ satisfying the properties in Definition 3.1.1. Let $n \in \mathbb{N}$ be given. We compute $d := \delta(2 \cdot n)$.

Let $m := \frac{n \cdot M}{d^2} + 1$. We claim the Bernstein polynomial $B_m(f)$ has distance at most $\frac{1}{n}$ to f. Therefore, we choose $x \in [0,1]$ arbitrary. We consider two subsets of interpolation nodes:

$$S := \{r \leq m : |\frac{r}{m} - x| < d\}, T := \{r \leq m : |\frac{r}{m} - x| \geq d\}.$$

Now we can give our first estimate of the distance between $(B_m(f))(x)$ and $f(x)$. We see $f(x) = 1 \cdot f(x) = \sum_{r=0}^{m} f(x) \cdot \binom{m}{r} \cdot x^r \cdot (1-x)^{m-r}$, because of Lemma 3.2.6. With $S \cap T = \emptyset$ we compute:

$$\begin{aligned}
|(B_m(f))(x) - f(x)| &= |\sum_{r=0}^{m} f(\frac{r}{m}) \cdot \binom{m}{r} \cdot x^r \cdot (1-x)^{m-r} - \sum_{r=0}^{m} f(x) \cdot \binom{m}{r} \cdot x^r \cdot (1-x)^{m-r}| \\
&= |\sum_{r=0}^{m} (f(\frac{r}{m}) - f(x)) \cdot \binom{m}{r} \cdot x^r \cdot (1-x)^{m-r}| \\
&\leq \sum_{r=0}^{m} |f(\frac{r}{m}) - f(x)| \cdot \binom{m}{r} \cdot x^r \cdot (1-x)^{m-r} \\
&= \sum_{r \in S} |f(\frac{r}{m}) - f(x)| \cdot \binom{m}{r} \cdot x^r \cdot (1-x)^{m-r} + \sum_{r \in T} |f(\frac{r}{m}) - f(x)| \cdot \binom{m}{r} \cdot x^r \cdot (1-x)^{m-r}.
\end{aligned}$$

We consider both series individually. Firstly, we consider the sum running over S. Using the fact that for every $r \in S$ the inequality $|\frac{r}{m} - x| < d$ holds, we know $|f(\frac{r}{m}) - f(x)| < \frac{1}{2 \cdot n}$ and hence:

$$\sum_{r \in S} |f(\frac{r}{m}) - f(x)| \cdot \binom{m}{r} \cdot x^r \cdot (1-x)^{m-r} \leq \sum_{r \in S} \frac{1}{2 \cdot n} \cdot \binom{m}{r} \cdot x^r \cdot (1-x)^{m-r} = \frac{1}{2 \cdot n}.$$

The series running over T is more complicated. If $r \in T$, then we only know that $|f(x) - f(\frac{r}{m})| \leq 2 \cdot M$. Because $|\frac{r}{m} - x| \geq d$ and both sides are positive, we get $|\frac{r}{m} - x|^2 \geq d^2$ by squaring both sides. Hence, we see that $\frac{(\frac{r}{m} - x)^2}{d^2} \geq 1$. Now we compute:

$$\begin{aligned}
\sum_{r \in T} |f(\frac{r}{m}) - f(x)| \cdot \binom{m}{r} \cdot x^r \cdot (1-x)^{m-r} &\leq 2 \cdot M \cdot \sum_{r \in T} \binom{m}{r} \cdot x^r \cdot (1-x)^{m-r} \\
&\leq \frac{2 \cdot M}{d^2} \cdot \sum_{r \in T} (\frac{r}{m} - x)^2 \cdot \binom{m}{r} \cdot x^r \cdot (1-x)^{m-r} \\
&\leq \frac{2 \cdot M}{d^2} \cdot \sum_{r \in T} (x^2 - \frac{2 \cdot r \cdot x}{m} + \frac{r^2}{m^2}) \cdot \binom{m}{r} \cdot x^r \cdot (1-x)^{m-r} \\
&\leq \frac{2 \cdot M}{d^2} \cdot \sum_{r=0}^{m} (x^2 - \frac{2 \cdot r \cdot x}{m} + \frac{r^2}{m^2}) \cdot \binom{m}{r} \cdot x^r \cdot (1-x)^{m-r}.
\end{aligned}$$

By splitting the series and with Lemma 3.2.6 we see it is equal to:

$$(B_m(\lambda x.x^2))(x) - 2 \cdot x \cdot (B_m(\lambda x.x))(x) + x^2 \cdot (B_m(\lambda x.1))(x) = \frac{1}{m} \cdot x \cdot (1-x).$$

Furthermore, because $x \in [0,1]$, the function $\lambda y.y \cdot (1-y)$ attains the maximum $\frac{1}{4}$ at $y = \frac{1}{2}$, and thus $x \cdot (1-x) \leq \frac{1}{4}$. Hence, we can conclude $\sum_{r \in T} |f(\frac{r}{m}) - f(x)| \binom{m}{r} \cdot x^r \cdot (1-x)^{m-r} \leq \frac{M}{2 \cdot m \cdot d^2}$. By combining both estimates and using that $m > \frac{n \cdot M}{d^2}$, we obtain:

$$|(B_m(f))(x) - f(x)| < \frac{1}{2 \cdot n} + \frac{M}{2 \cdot m \cdot d^2} = \frac{1}{n}.$$

Therefore, we can construct a sequence of Bernstein polynomials converging effectively to f. □

3.3. Conclusion

Using the technique developed in the previous sections, we prove the following theorem in this section:

Theorem 3.3.1. *The representation δ^A_{Cp} is equivalent to δ^A_{C}.*

For the proof we need the following:

Lemma 3.3.2. *If a function $f : [a,b] \to \mathbb{R}$ defined on an interval $[a,b]$ with computable endpoints is effective uniform continuous and pointwise computable, then there exists a sequence of rational polygons which converges effectively to f.*

PROOF. Let an effective uniformly continuous function $f : [a,b] \to \mathbb{R}$ be given with a and b computable real numbers. Furthermore, we assume f is pointwise computable, and we have $\delta : \mathbb{N} \to \mathbb{N}$ as in Definition 3.1.1. Our goal is to construct a sequence of rational polygons which converges effectively to f. Let $n \in \mathbb{N}$ be given. Firstly we compute $\delta(n+1)$. Because a is a computable real number, we can find a rational interval (r,r') containing a with $r'-r < \frac{1}{2\cdot\delta(n+1)}$. Also, we can find a rational interval (s,s') containing b with $s'-s < \frac{1}{2\cdot\delta(n+1)}$. Now we construct a rational polygon on the interval $[r,s']$. We define the amount of interpolation nodes as $c := \frac{s'-r}{2\cdot\delta(n+1)}+1$, and rational vectors $p,q \in \mathbb{Q}^c$ with interpolation nodes as $p(i) := r+\frac{i}{c}\cdot(s'-r)$. When $0 < i < c$ the node $p(i)$ is in the domain of f and computable, thus we can find a rational number $t \in \mathbb{Q}$ with $|t - f(p(i))| < 10^{-m-1}$ and we define the weights $q(i) := t$ for those nodes. Furthermore, we define $q(0) = q(1)$ and $q(c) = q(c-1)$.

Let g be the rational polygon with interpolation nodes p and weights q. Now we prove the distance between f and g is at most 10^{-m}. Let $x \in [a,b]$ be arbitrary. Then there is an $i \in \mathbb{N}$ with $i < c$ such that $x \in [p(i), p(i+1)]$. Assume $i \neq 0$ and $i \neq c-1$. Firstly, we estimate the distance between $g(p(i))$ and $g(x)$. Because $x \in [p(i), p(i+1)]$, we see:

$$|g(p(i)) - g(x)| \leq |g(p(i)) - g(p(i+1))| = |f(p(i)) - f(p(i+1))| < 10^{-m-1}.$$

Therefore, we can estimate the distance between $f(x)$ and $g(x)$:

$$|f(x) - g(x)| \leq |f(x) - f(p(i))| + |f(p(i)) - g(p(i))| + |g(p(i)) - g(x)| < 3 \cdot 10^{-m-1} < 10^{-m}.$$

If $i = 0$ or $i = c-1$, then the following inequality holds:

$$|f(x) - g(x)| \leq |f(x) - f(p(i))| + |f(p(i)) - g(x)| < 2 \cdot 10^{-m-1} < 10^{-m}.$$

Hence, there exists a sequence of rational polygons which converges effectively to f. □

By combining Lemma 3.3.2, Proposition 3.1.12 and Theorems 3.1.9 and 3.2.7, we obtain the following corollaries:

Corollary 3.3.3. *A function is computable with respect to δ_{C}^{A} iff it is effective uniform continuous and pointwise computable.*

Corollary 3.3.4. *A function is computable with respect to δ_{Cp}^{A} iff it is effective uniform continuous and pointwise computable.*

Theorem 3.3.1 follows directly from these two corollaries. When a rational polygon is given, one can use Theorem 3.2.7 to compute a name for the polygon. Hence, $\delta_{\mathrm{C}}^{A} \leq \delta_{\mathrm{Cp}}^{A}$. With Lemma 3.3.2 one can compute a name for every polynomial, and thus $\delta_{\mathrm{Cp}}^{A} \leq \delta_{\mathrm{C}}^{A}$.

CHAPTER 4

Algorithms in Exact Real Arithmetic

In the introduction we saw an example of a mathematically correct algorithm which gives inprecise results, and in the previous chapters we saw techniques for exact computation. In this chapter we discuss robust algorithms for solving ordinary differential equations such that the solution can be computed on every point with any prescribed accuracy.

First we discuss the classical Runge-Kutta methods. Because these methods are closely related to numerical integration, we start with algorithms for integrating. An exact implementation of these is described in [**EK99**], but we argue the classical Runge-Kutta methods are not suitable for an exact implementation. Therefore, we look at different algorithms. These are based on domain theory and are from [**EKL03, Pat05**].

4.1. Numerical Integration

In this section we discuss the *general quadrature formula* for integration. The main goal is finding the error of this method. We follow [**Hun13, EK99**] in our approach. To approximate the integral of a function on a certain interval, one should first partition the interval. This can be done by cutting it in n pieces of equal length or in some other way. Given such a partition, the integral can be computed on every smal interval and the total integral can be found by summing up those values. To find the integral on a small interval, *quadrature formulas* are used:

Definition 4.1.1 (General Quadrature Formula)**.** The *general quadrature formula* for approximating the integral $\int_0^h f(x)\,\mathrm{d}x$ is

$$h \cdot \sum_{i=1}^{s} b_i \cdot f(c_i \cdot h)$$

where h is the step-size, b_i the *weights* and c_i the *nodes* of the method.

Without loss of generality we can take the intergral on $[0,h]$, otherwise we would use the subsitution rule. General quadrature formulas have a simple algorithm which can be found in Listing 4.1 where R is a data type for real numbers.

LISTING 4.1. Quadrature formula

```
data Quad = Quad [R] [R] R

integrate :: Quad → (R → R) → R
integrate (Quad [] [] _) _      = 0
integrate (Quad b c h) f        = h·series b c h f

series :: [R] → [R] → R → (R → R) → R
series  [] [] _ _               = 0
series (b:bs) (c:cs) h f        = b·f(c·h) + series bs cs h f
```

For the error of a certain quadrature formula we consider the *order* of a method. A method has a certain order n when polynomials of degree lesser than n are integrated exactly. It us useful when a function is approximated with its Taylor series, because then we can easily compute the error when it is integrated.

Definition 4.1.2 (Order)**.** A quadrature formula is said to have *order* p iff it integrates polynomials of degree at most $p-1$ exactly.

If s distinct integration nodes are given, we can determine the unique weights such that the method has order s. Because the quadrature formula is linear, it integrates polynomials of degree at most $p-1$ exactly whenever it integrates x^i correctly for $i \leq p-1$. This can be used to determine a system of s linear equations which has a unique solution. Now we describe the error of the general quadrature formula:

Theorem 4.1.3. *Let a quadrature formula of order p and a function $f : [0,h] \to \mathbb{R}$ which is p times differentiable, be given. Then the error of the method can be estimated by:*

$$\frac{1}{p!} \cdot h^{p+1} \cdot \sup\{|f^p(x)| : x \in [0,h]\} \cdot |\frac{1}{p+1} - \sum_{i=1}^{s} b_i \cdot c_i^p|.$$

This theorem can be proved by considering the Taylor expansion of f around 0 which gives the bound $\frac{1}{p!} \cdot h^{p+1} \cdot \sup\{|f^p(x)| : x \in [0,h]\}$ for the remainder term. The term $\frac{1}{p+1} - \sum_{i=1}^{s} b_i \cdot c_i^p$ is the error of integrating a polynomial of degree p. Notice that the derivative of a function is required to find this error, and a bound of the derivative can be hard to find in practice. Furthermore, the function should satisfy some smoothness assumptions which are not satisfied by continuous functions in general. Now we discuss two concrete quadrature formulas.

Example 4.1.4. Let $b_1 = b_2 = \frac{1}{2}$, $c_0 = 0$, and $c_1 = 1$. Then we get the following quadrature formula which is known as the *trapezoidal rule*:

$$\frac{h}{2} \cdot (f(0) + f(1)).$$

This method has order 2. The error of integrating a 2 times differentiable function is $h^3 \cdot B \cdot \frac{1}{12}$ where B is bound for the second derivative.

Example 4.1.5. *Simpson's rule* is a quadrature formula with $b_1 = b_3 = \frac{1}{6}$, $b_2 = \frac{4}{6}$, $c_1 = 0$, $c_2 = \frac{1}{2}$ and $c_3 = 1$. Thus, the form is as follows:

$$\frac{h}{6} \cdot (f(0) + 4 \cdot f(\frac{1}{2}) + f(1)).$$

It can be proven that this method has order 4, and hence the error for integrating a 4 time differentiable function with a fourth derivative bounded by B is $\frac{1}{2880} \cdot h^5 \cdot B$.

Both methods are based on polynomial approximation. The trapezoidal rule approximates the integrand with a piecewise linear function and Simpson's rule uses a piecewise quadratic function.

4.2. Numerical Solutions of Ordinary Differential Equations

Now we discuss *Runge-Kutta methods* using [**Hun13, EK99**] which form a class of algorithms which can be used to solve initial value problems. To find the value of the solution at a certain moment, it takes multiple steps starting at the initial value. By rewriting a problem in the integral form, the relation with numerical integration can be seen. This is used to develop a general class of methods.

Definition 4.2.1 (General Runge-Kutta Method)**.** Given a matrix $a \in \mathbb{R}^{s \times s}$, two vectors $b, c \in \mathbb{R}^s$ such that $c_i = \sum_{j=1}^{s} a_{i,j}$, and a number $h \in \mathbb{R}$, define $t_n = t_0 + n \cdot h$ for any $n \in \mathbb{N}$. Then the *general Runge-Kutta method* with coefficients a, b, c and h is described as follows:

$$v_{n+1,i} = u_n + h \cdot \sum_{j=1}^{s} a_{i,j} \cdot f(t_n + c_j \cdot h, v_{n+1,j}),$$

$$u_0 = y_0,$$

$$u_{n+1} = u_n + h \cdot \sum_{i=1}^{s} b_i \cdot f(t_n + c_i \cdot h, v_{n+1,i})$$

where $u'(t) = f(t, u(t))$, $u(t_0) = y_0$ is an initial value problem. We will abbreviate these methods as $u_{n+1} = u_n + R(t_n, u_n, h)$ where R is the *increment function*. When for $j \geq i$ the value $a_{i,j}$ is equal to 0, then the method is called *explicit*. Otherwise, we call it *implicit*.

These methods are quite complicated. The variable n says how many steps are taken, and the values t_n form a grid of points on which we compute the solution. To determine the next value we calculate the integral. However, the integration nodes might be unknown values of the functions, and thus those need to be approximated first. This is done with the variables $v_{n,i}$. The approximations of the solution are defined on the points t_i are the values u_i. If the method is implicit, the approximations are defined implicetly, and root-finding is required to find them. The result of an explicit method can still be determined by computing the series. Therefore, an algorithm for a general Runge-Kutta method is more complicated than an algorithm for a quadrature formula, but for explicit methods we can write a similar algorithm.

The error of a numerical method for solving differential equation is more complicated than the error of integration as well. After every step a small error is made, and those all contribute to the error at the nth step. These errors can be determined using Taylor's theorem, and therefore the derivative of the solution occurs in the final estimate. With an upper bound for these errors, we can find an upper bound for the total error:

Theorem 4.2.2. *Given a Runge-Kutta method $u_{n+1} = u_n + R(t_n, u_n, h)$ for an initial value problem be given and constants $h_* \in \mathbb{R}$ and $L \in \mathbb{R}$ such that for every $x, y \in \mathbb{R}$, every time $t \in [t_0, T]$ and $h \in (0, h_*]$ the condition $|R(t,x,h) - R(t,y,h)| \leq L \cdot |x-y|$ is satisfied. Then the error at t_n can be estimated as follows:*

$$|u_n - u(t_n)| \leq e^{L\cdot t} \cdot |u_0 - u(t_0)| + \frac{1}{L} \cdot (e^{L\cdot t_n - 1}) \cdot \sup\{d_j : 1 \leq j \leq n\}$$

where $d_j = |\frac{u(t_n) - u(t_{n-1}) - h\cdot R(t_{n-1}, u(t_{n-1}), h))}{h}|$.

The values d_j are the local errors. If there is no error until step j, then $h \cdot d_j$ indicates the error at step j. If the initial value is not given exactly, the term $e^{L\cdot t} \cdot |u_0 - u(t_0)|$ is the contribution to the error. Notice that L depends on the Lipschitz constant of the initial value problem. Now we discuss some examples of Runge-Kutta methods.

Example 4.2.3. *Euler's method* has coefficients $a = 0$, $b = 1$ and $c = 0$. This gives the following scheme:

$$v_{n+1} = u_n,$$
$$u_{n+1} = u_n + h \cdot f(t_n, v_{n+1}).$$

This method is explicit and the approximations are defined explicitly.

Example 4.2.4. An example of an implicit method is the *implicit trapezoid rule* in which the coefficients are $a = \begin{pmatrix} 0 & 0 \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}$, $b = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}$, $c = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. This can be summarized as:

$$v_{n+1,1} = u_n,\ v_{n+1,2} = u_n + h \cdot (\frac{1}{2} \cdot f(t_n, u_n) + \frac{1}{2} \cdot f(t_n, v_{n+1,2})),$$
$$u_{n+1} = u_n + h \cdot (\frac{1}{2} \cdot f(t_n, u_n) + \frac{1}{2} \cdot f(t_n, v_{n+1,2})).$$

Notice that $u_{n+1} = v_{n+1,2}$ and that those approximations are defined implicitly.

Some Runge-Kutta methods give bad results for a certain class of differential equations known as *stiff* equations. An in-depth study about stiff problems is out of our scope, but the following example is interesting:

Example 4.2.5. Consider the following differential equation:

$$u'(t) = -3737 \cdot u(t),\ u(0) = 1$$

The solution is $u(t) = e^{-3737\cdot t}$ and a Lipschitz constant for this problem is 3737. When one attempts to solve this differential equation with Euler's method, the plots in Figure 1 are obtained. Notice that the error of the method is very high when the step sizes are relatively large. Besides, the error-estimate gives a high bound for the error, namely $\frac{1}{3737} \cdot e^{3737\cdot 0.01 - 1} \approx 10^{12}$.

The Runge-Kutta methods have several disadvantages for an exact implementation. In Example 4.2.5 we saw that problems with a high Lipschitz constant give a bad error estimate. Also, the derivative of the solution is required for the error of the method, since the local errors are estimated with the Taylor expansion of the solution.

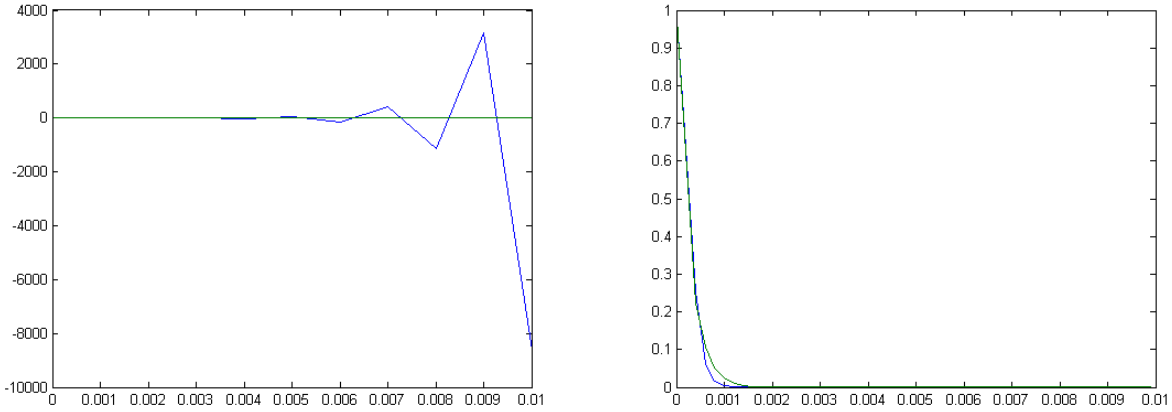

FIGURE 1. On the left: step size 0.001. On the right: step size 0.0002. The actual solution is green.

4.3. Solving Initial Value Problems using Domain Theory

In the previous section we saw that classical Runge-Kutta methods are not suitable for finding exact solutions of some ordinary differential equations. In this section we discuss an algorithm from [**EKL03**] which solves scalar initial value problems. The datatypes and algorithms have a domain theoretic nature, and the algorithm is based on the Picard operator from Definition 2.5.7. Therefore we need to assume that the differential equation satisfies a Lipschitz condition and that the solution is defined on a compact real interval. A fixed point of the operator is a solution of the differential equation, and can be found by repeated application. However, an algorithm is required to compute the Picard operator.

The idea is to split it into two subroutines. Additionally, the data is represented as a pair of functions (u,v) such that u is an approximation of the solution and v is an approximation of its derivative. We define the following operators for a scalar differential equation $u'(t) = f(t,u(t))$, $u(t_0) = y_0$:

$$U(u,v) := (\lambda t.y_0 + \int_{t_0}^{t} v(s)\,\mathrm{d}s, v), A_f(u,v) := (u, \lambda t.f(t,u(t)).$$

The operator U updates the function with the information of v, and A_f applies the differential equation to u for a new approximation of the derivative. Notice that the first coordinate of $U \circ A_f$ is the result of applying the Picard operator to u, because:

$$\begin{aligned} U(A_f(u,v)) &= U((u, \lambda t.f(t,u(t)))) \\ &= (\lambda t.y_0 + \int_{t_0}^{t} f(s,u(s))\,\mathrm{d}s, \lambda t.f(t,u(t))). \end{aligned}$$

When we have algorithms for the operators U and A_f, we have an algorithm for solving initial problems. However, when u, v and f are arbitrary functions, these algorithms can become complicated, and therefore we represent them as an approximation of simple functions. Step functions from Definition 2.6.18 are useful for this goal, although primitives of step functions are linear. Therefore, we introduce *linear functions* on the interval domain $\mathrm{I}\mathbb{R}$:

Definition 4.3.1 (Single-Linear Function)**.** Given a real interval $a \in \mathrm{I}\mathbb{R}$ and two linear functions $f^- : a \to \mathbb{R}$ and $f^+ : a \to \mathbb{R}$ be given such that $f^-(x) \leq f^+(x)$ for every $x \in a$, we define the *single-linear function* $a \searrow [f^-(x), f^+(x)] : \mathrm{I}\mathbb{R} \to \mathrm{I}\mathbb{R}$ as follows:

$$(a \searrow [f^-(x), f^+(x)])(x) := \begin{cases} [f^-, f^+] & \text{if } a \ll x \\ \bot & \text{otherwise} \end{cases}.$$

Definition 4.3.2 (Consistency)**.** Let a finite set $S = \{a_i \searrow b_i : i \in I\}$ of single-linear functions be given. Then we call S *consistent* iff for every subset $J \subseteq I$ such that an $a \in \mathrm{I}\mathbb{R}$ with $\{a_j : j \in J\} \ll a$ exists, there is a $b \in \mathrm{I}\mathbb{R}$ such that $\{b_j : j \in J\} \sqsubseteq b$.

Definition 4.3.3 (Linear Function)**.** The supremum of a finite set of consistent linear single-step functions is called a *linear function*.

The definition of linear functions is similar to Definition 2.6.18. Notice that these functions form a base for $[\mathrm{I}\mathbb{R} \to \mathrm{I}\mathbb{R}]$ as well, because step functions are linear. The linear and step functions can be used as datatypes. Also, we assume that the maps f and v are suprema of step function, and u is a supremum of linear functions. The advantage of these datatypes is that the corresponding algorithms are easier. Now we can describe an algorithm for updating the approximation of the derivative of a pair. Therefore, we need a simple formula describing $\lambda t.f(t,u(t))$ which is given in the following theorem:

Theorem 4.3.4. *Let two functions $u = \bigsqcup\{d_j \searrow g_j : j \in J\}$ and $f = \bigsqcup\{(a_i \times b_i) \searrow c_i : i \in I\}$ be given as the finite supremum of respectively linear functions and step functions. If for $i \in I$ and $j \in J$ we write $g_j = [g_j^-, g_j^+]$, $b_i = [b_i^-, b_i^+]$ and $d_{ij} = \overline{(g_j^-)^{-1}((b_i^-,\infty)) \cap (g_j^+)^{-1}((-\infty, b_i^+))}$, then:*

$$\lambda t.f(t,u(t)) = \bigsqcup\{(\bigsqcup\{a_i, d_{ji}\}) \searrow c_i : g^{-1}(\uparrow b_i) \neq \emptyset, \mathrm{int}(a_i \cap d_{ij}) \neq \emptyset\}.$$

On the intervals d_j we know that $u(t) \in g_j(t)$ and on the rectangles $a_i \times b_i$ we know $f(a,b) \in c_i$. The intervals d_{ij} are the sets on which the result $g_j(t)$ and b_i match, because the result of an element in d_{ij} is atleast b_i^- and at most b_i^+. Therefore, we have $t \in \mathrm{int}(d_{ij})$ iff $b_i \ll g_j(t)$. We know that elements d_{ij} are intervals, because the functions are linear. If we know that $t \in a_i$ as well, then we have $f(t, g_j(t)) = c_i$ on $\bigsqcup\{a_i, d_{ji}\}$. Because u is the supremum of all the g_j and because v is continuous, the result is the supremum of all these functions. Furthermore, it is a step function, because it is the finite supremum of single-step functions. This gives the following algorithm for updating the derivative:

LISTING 4.2. Derivative Updating

```
u := ⨆{d_j ↘ g_j : 1 ≤ j ≤ m}
f := ⨆{a_i × b_i ↘ c_i : 1 ≤ i ≤ n}
C := ∅

for 1 ≤ i ≤ n
  for i ≤ j ≤ m
    if match b_i g_j
      obtain d_ij
      if a_i ∩ d_ij ≠ ∅
        C := C ∪ (⨆{a_i, d_ij} ↘ c_i)
```

The match function determines whether the interval d_{ij} is empty. Notice that this algorithm can be implemented using rational arithmetic by using rational linear and step functions, and hence no round-off mistakes are made.

Now we need an algorithm for updating functions, and therefore we need to compute the Picard operator on step functions. Therefore, we need the notion of the *indefinite integral* for interval-valued functions:

Definition 4.3.5 (Indefinite Integral for Single-Step Functions)**.** We define the *indefinite integral* of a single step-function in $[\mathbb{R} \to \mathrm{I}\mathbb{R}]$ as follows:

$$\int a \searrow b := \delta(a,b),$$

$$\delta(a,b) := \{g \in [\mathbb{R} \to \mathrm{I}\mathbb{R}] : \text{for all } x,y \in \mathrm{int}(a) \text{ holds } b \cdot (x-y) \sqsubseteq g(x) - g(y)\}.$$

Definition 4.3.6 (Indefinite Integral)**.** We define the *indefinite integral* of any function in $[\mathbb{R} \to \mathrm{I}\mathbb{R}]$ given as least upper bound of single step-function as follows:

$$\int \bigsqcup\{a_i \searrow b_i : i \in I\} := \bigcap_{i \in I} \delta(a_i, b_i).$$

Again this definition can be extended to functions in $[[0,1] \to \mathrm{I}\mathbb{R}]$. This definition is based on normal integration. Because f is supposed to be a primitive of the step function, we know $f(y)-f(x) = b\cdot(x-y)$ if $b \in \mathbb{R}$. However, f is interval-valued, and therefore the primitive should give atleast $b\cdot(x-y)$ as information. The following functions are convenient for our goal:

Definition 4.3.7 (Some functions)**.** For interval-valued functions $f, g \in [[0,1] \to \mathrm{I}\mathbb{R}]$ such that $f = [f^-, f^+]$ and $g = [g^-, g^+]$, we define the following functions for real numbers $x, y \in \mathbb{R}$:

$$s(f,g) := \inf\{h : \mathrm{dom}(g) \to \mathbb{R} : h \in \int g \text{ and } h \geq f^-\},$$

$$t(f,g) := \sup\{h : \mathrm{dom}(g) \to \mathbb{R} : h \in \int g \text{ and } h \leq f^+\},$$

$$K^{+-}(g)(x,y) := \begin{cases} \int_y^x g^-(u)\,\mathrm{d}u & \text{if } x \geq y \\ -\int_x^y g^+(u)\,\mathrm{d}u & \text{if } x < y \end{cases},$$

$$K^{-+}(g)(x,y) := \begin{cases} \int_y^x g^+(u)\,\mathrm{d}u & \text{if } x \geq y \\ -\int_x^y g^-(u)\,\mathrm{d}u & \text{if } x < y \end{cases},$$

$$S(f,g)(x,y) := f^-(y) + K^{+-}([g^-,g^+])(x,y),$$
$$T(f,g)(x,y) := f^+(y) + K^{-+}([g^-,g^+])(x,y).$$

The functions s and t give a lower bound repsectively upper bound for a primitive of g which is consistent with the information of f. The function h computes with an interval a value in $\mathbb{R}$. The new approximation of the function is defined as $[s(f,g)(x), t(f,g)(x)]$. Hence, we need a more convenient formula for these expressions. When a point y is given, the functions K^{+-} and K^{-+} give respectively a lower estimate and an upper estimate of the integral of g from y to x. The function S is the least primitive of g which is in y atleast $f^-(y)$ and T has a similar property. Notice that the following identities hold:

$$s(f,g) = \lambda x.\sup\{S(f,g)(x,y) : y \in \mathrm{dom}(g)\} \tag{4.1}$$

$$t(f,g) = \lambda x.\inf\{T(f,g)(x,y) : y \in \mathrm{dom}(g)\} \tag{4.2}$$

The function $s(f,g)$ is an upper bound for the set $\{S(f,g)(x,y) : y \in \mathrm{dom}(g)\}$, hence the left side of Equation (4.1) is atleast the right side of Equation (4.1). The reversed inequality holds as well, because $S(f,g)$ is primitive of g and for every $y \in \mathbb{R}$ at least $f^-(y)$. A similar argument holds for Equation (4.2). Now we give an more convenient expression for $s(f,g)$ and $t(f,g)$:

Theorem 4.3.8. *Let two interval-valued functions $f, g \in [[0,1] \to \mathrm{I}\mathbb{R}]$ be given as a finite supremum of respectively linear and step functions. If we write $f = [f^-, f^+]$ and $g = [g^-, g^+]$, then $[s(f,g), t(f,g)]$ is a linear function. Furthermore, the following identities hold on any connected component of* $\mathrm{dom}(g)$ *where J is the induced partition of f and g:*

$$s(f,g)(x) = \max(\{f^-(x)\} \cup \{\mathrm{limsup}_{y \to y_k} S(f,g)(x,y) : y_k \in J \cap \mathrm{dom}(f)\}),$$
$$t(f,g)(x) = \min(\{f^+(x)\} \cup \{\mathrm{liminf}_{y \to y_k} T(f,g)(x,y) : y_k \in J \cap \mathrm{dom}(f)\}).$$

Notice that the identities imply the theorem, because they imply that $s(f,g)$ and $t(f,g)$ are the finite maximum of functions. Every connected component of a real set is an open interval. The functions f and g are both on a certain partition respectively a linear and step function, and this gives the induced partition. The proof of this theorem uses the fact that f and g are simple functions, because then S and T have a simple explicit expression. Equality follows by rewriting it. With this theorem we get the following algorithm for updating the function:

LISTING 4.3. Function Updating

```
f := ⨆{a_j ↘ [f_j^-, f_j^+] : 1 ≤ j ≤ n}
g := ⨆{b_k ↘ [e_k^-, e_k^+] : 1 ≤ k ≤ n}
y_k is induced partition of f and g, g has the value e_k^- on (y_{k-1}, y_k)

d(y_0) := lim_{y ↓ y_0} f^-(y)
```

$e(y_0) := \lim_{y \downarrow y_0} f^+(y)$
for $1 \le k \le n$
  for $x \in [y_{k-1}, y_k)$
    $d(x) := \max\{f^-(x), d(y_{k-1}) + (x - y_{k-1}) \cdot e_k^-\}$
    $e(x) := \min\{f^+(x), e(y_{k-1}) + (x - y_{k-1}) \cdot e_k^+\}$
  $d(y_k) := \max\{\text{limsup}_{y \uparrow y_k} f^-(y), d(y_{k-1}) + (y_k - y_{k-1}) \cdot e_k^-\}$
  $e(y_k) := \min\{\text{liminf}_{y \uparrow y_k} f^+(y), e(y_{k-1}) + (y_k - y_{k-1}) \cdot e_k^-\}$
$s(y_n) := d(y_n)$
$t(y_n) := e(y_n)$
for $k = n \ldots 1$
  for $x \in [y_{k-1}, y_k)$
    $s(f,g)(x) := \max\{d(x), s(y_k) + (x - y_k) \cdot e_k^+\}$
    $t(f,g)(x) := \min\{e(x), t(y_k) + (x - y_k) \cdot e_k^-\}$

The algorithm has two stages. First, it goes from left to right, and then from right to left. The following values are computed in the first respectively second stage:

$$\max(\{f^-(x)\} \cup \{\text{limsup}_{y \uparrow y_k} S(f,g)(x,y) : y_k \in J \cap \text{dom}(f)\}),$$

$$\max(\{f^-(x)\} \cup \{\text{limsup}_{y \downarrow y_k} S(f,g)(x,y) : y_k \in J \cap \text{dom}(f)\}).$$

The maximum of the two values is taken in the second stage. By Theorem 4.3.8 this gives $s(f,g)$. Notice that the algorithm does not use the initial value, but this can be fixed by adding the initial value to $s(f,g)$ and $t(f,g)$. With the previous algorithms we get a method to find the exact solution of the scalar differential equation $u'(t) = f(t, u(t))$, $u(0) = 0$:

LISTING 4.4. Algorithm

$\varepsilon \in \mathbb{Q}$ such that $\varepsilon > 0$
$a_0 \in \mathbb{Q}$ such that $a_0 > 0$
$M \in \mathbb{Q}$

$f := \bigsqcup\{a_i \times b_i \searrow c_i : 1 \le i \le n\}$

for $k \in \mathbb{N}$, $k \neq 0$
  $u_{k0} := \bigsqcup\{[\frac{-a_0}{2^i}, \frac{a_0}{2^i}] \searrow [\frac{-a_0 \cdot M}{2^i}, \frac{a_0 \cdot M}{2^i}] : 0 \le i \le k\}$
  $(u_{k0}, v_{k0}) = A_f(u_{k0}, \bot)$
  for $1 \le n \le k$
    $(u_{kn}, v_{kn}) = (U \circ A_f)^n(u_{j0}, v_{j0})$
    if $|u_{jn}^+ - u_{jn}^-| \le \varepsilon$
      return u_{kn}

The steps with U and A_v can be computed with Listings 4.2 and 4.3. The variable ε is the desired precision. The value M is an upper bound of the scalar field v which can be determined by computing a maximum of v. The rational number a_0 is chosen such that the function v satisfies a Lipschitz condition on the rectangle $[-a_0, a_0] \times [-M \cdot a_0, M \cdot a_0]$, and the first approximation uses this property. Notice the similarity with the proof of Cauchy-Peano existence theorem.

Example 4.3.9. In this example we approximate the solution of the differential equation $u'(t) = u(t)$, $u(0) = 1$ by hand with this algorithm, and work out the first step. We take $\varepsilon = \frac{1}{2}$. On the rectangle $[-1,1] \times [-1,3]$ the function satisfies a Lipschitz condition, and therefore we can take $M = 4$ and $a_0 = 1$. We denote $\mathrm{I} := [-1,1]$. Now we describe v as supremum of step functions. We choose for $i \in \mathbb{N}$ with $i > 1$:

$$f_i = \bigsqcup\{\mathrm{I} \times [\frac{j}{3 \cdot i}, \frac{j+1}{3 \cdot i}] \searrow [\frac{j}{3 \cdot i}, \frac{j+1}{3 \cdot i}] : j \in \mathbb{Z} \text{ such that } -i \le j < 3 \cdot i\},$$

$$f_1 = \bigsqcup\{\mathrm{I} \times [-4,6] \searrow [-4,6], \mathrm{I} \times [-3,3] \searrow [-3,3]\}.$$

The approximation f_1 is different, because then the calculations are simpler. When i goes to infinity, the f_i converges to the function of the scalar field $u'(t) = u(t)$. Now we start Listing 4.4 with $k = 1$. Then our

start function is $u_{k0} = \bigsqcup\{[-1,1] \searrow [-3,5], [\frac{-1}{2}, \frac{1}{2}] \searrow [-1,3]\}$, because the initial value is added. Now we compute v_{k0} with the following approximation of the scalar field:

$$f_1 = \bigsqcup\{\mathrm{I} \times [-4,6] \searrow [-4,6], \mathrm{I} \times [-3,3] \searrow [-3,3]\}.$$

Now we find the matching pairs b_i and g_j. The resulting single-step functions can be found in Table 1 and the function v_{10} is the supremum of these single-step functions.

Pair	Domain d_{ij}	Single-step function
$[-4,6] \ll [-1,3]$	$[\frac{-1}{2}, \frac{1}{2}]$	$[\frac{-1}{2}, \frac{1}{2}] \searrow [-4,6]$
$[-3,3] \ll [-1,3]$	$[\frac{-1}{2}, \frac{1}{2}]$	$[\frac{-1}{2}, \frac{1}{2}] \searrow [-3,3]$
$[-4,6] \ll [-3,5]$	I	$\mathrm{I} \searrow [-4,6]$

TABLE 1. Intermediate values of the derivative updating

Now we can write v_{10} as supremum of step functions, and this approximation is less accurate than the approximation of u_{10}. The next step is updating the function. First, we compute the induced partition of u_{10} and v_{10}:

$$\{[-1, \frac{-1}{2}], [\frac{-1}{2}, \frac{1}{2}], [\frac{1}{2}, 1]\}.$$

Value	d	$s(u_{10}, v_{10})$	e	$t(u_{10}, v_{10})$
-1	-3	$\frac{-5}{2}$	5	$\frac{9}{2}$
$(-1, \frac{-1}{2})$	-3	$-1 + (-x + \frac{1}{2}) \cdot 3$	5	$3 + (x + \frac{1}{2}) \cdot -3$
$\frac{-1}{2}$	-1	-1	3	3
$(\frac{-1}{2}, \frac{1}{2})$	-1	-1	3	3
$\frac{1}{2}$	-1	-1	3	3
$(\frac{1}{2}, 1)$	$-1 + (x - \frac{1}{2}) \cdot -4$	$-1 + (x - \frac{1}{2}) \cdot -4$	$3 + (x - \frac{1}{2}) \cdot 3$	$3 + (x - \frac{1}{2}) \cdot 3$
1	-3	-3	$\frac{9}{2}$	$\frac{9}{2}$

TABLE 2. Intermediate values of the function updating

The values $1 + s(u_{10}, v_{10})$ and $1 + t(u_{10}, v_{10})$ give the new approximation of the solution. The algorithm does not terminate after this, because the difference is not on every point at most $\frac{1}{2}$. Notice that the error increases around $x = 1$ and $x = -1$. A reason for this is the inaccuracy in the scalar field.

CHAPTER 5

Conclusion

In this thesis we have discussed several things. We started with exact real arithmetic and computable analysis which we applied to determine properties of data types for real numbers and algorithms for solving differential equations. Our goal in this thesis was proving the representation in Example 2.3.31 was admissible, and finding algorithms to compute the exact solutions of a certain class of initial value problems.

Theorem 3.3.1 says the representation with polynomials is admissable. As a consequence representing function as an approximation of rational polynomials is adequate in practice. Furthermore, the algorithm in Listing 4.4 finds the solution of a differential equation up to any desired accuracy. An exact implementation of this algorithm is feasible. Round-off errors can be prevented, because it only uses arithmetic with rational numbers. It can be used to solve arbitrary initial value problems which satisfy a Lipschitz condition.

The algorithm in Listing 4.4 only works for scalar problems and that is an obvious disadvantage, because most problems in practice are not scalar. Besides, no explicit formula for the error-estimate is given, and hence one does not know how many steps are required to approximate the solution with a certain precision. It is thus difficult to determine the complexity.

There are alternative methods for finding the exact solution of differential equations. The algorithm in Listing 4.4 approximates the solution on a specific interval and not in general if the solution should be defined on an arbitrary interval. A possible extension using similar techniques is described in [**EP05**] which works for time-independent vector fields. Another possibility is to extend the algorithm to time-independent vector fields. This can be done by implementing Euler's method in domain theory [**EP06**]. Furthermore, this can be used for an algorithm which solves linear boundary value problems [**Pat05**]. By rewriting the problem to a number of initial values, the fundamental solution and a particular solution can be found. Solving a system of linear equations gives the final solution.

Further research topics include a representation based on trigonometric polynomials. By approximating functions with sums of sines and cosines a different representation is made. We suspect this representation is admissable, and we think this can be proved using the Discrete Fourier Transform. A possible application is constructing algorithms which can solve a certain class of equations. Another research topic is a different class of algorithms for solving differential equations, namely the implicit methods. These methods use root-finding to find the solution, and a certain class of differential equations can be solved efficiently with them. We suspect [**EP06**] is useful for this.

APPENDIX A

Heat equation

We discuss the solution of the one dimensional heat equation on the line segment $[0, \pi]$ using *seperation of variables* as described by [**Piv05**]. The heat equation is the following partial differential equation:

$$\frac{\partial f(t,x)}{\partial t} = \frac{\partial^2 f(t,x)}{\partial x^2}. \tag{A.1}$$

Furthermore, we assume f satisfies Dirichlet boundary conditions and has an initial distribution ϕ:

$$f(0,x) = \phi(x), f(t,0) = f(t,\pi) = 0.$$

We assume $\phi(x) = \sum_{i=1}^{\infty} d_i \cdot \sin(i \cdot x)$ has a Fourier series expansion. To solve Equation (A.1), we assume the solution has the form $f(x,t) = X(x) \cdot T(t)$. Now we compute the partial derivatives:

$$\frac{\partial f(t,x)}{\partial t} = X(x) \cdot T'(t), \ \frac{\partial^2 f(t,x)}{\partial x^2} = X''(x) \cdot T(t)$$

Therefore, $X(x) \cdot T'(t) = X''(x) \cdot T(t)$ which is equivalent to $\frac{X''(x)}{X(x)} = \frac{T'(t)}{T(t)}$. Since this equality holds for every $(x,t) \in \mathbb{R}^2$, the value is a constant $\lambda \in \mathbb{R}$. Now we find two ordinary differential equations:

$$T'(t) = \lambda \cdot T(t), \tag{A.2}$$

$$X''(x) = \lambda \cdot X(x), X(0) = X(\pi) = 0. \tag{A.3}$$

Both equations are linear and can easily be solved. The solution of Equation (A.2) is $T(t) = e^{\lambda \cdot t}$. To solve Equation (A.3) we need to find non-trivial solutions. This requires case distinction.

First we assume $\lambda > 0$, and in this case the solution is given by $X(x) = c_0 \cdot \sinh(\sqrt{\lambda} \cdot x) + c_1 \cdot \cosh(\sqrt{\lambda} \cdot x)$. The initial value $X(0) = 0$ gives $c_1 = 0$, and the other initial value gives $c_0 = 0$. In this case only the trivial solution satisfies the equation. If $\lambda = 0$, then the solution is $X(x) = a \cdot x + b$. The initial values give $a = 0$ and $b = 0$, and therefore only the zero solution satisfies the equation in this case. Finally, we consider $\lambda < 0$. The solution is $X(x) = c_0 \cdot \sin(\sqrt{|\lambda|} \cdot x) + c_1 \cdot \cos(\sqrt{|\lambda|} \cdot x)$. The initial value $X(0) = 0$ gives $c_1 = 0$. The other initial value says $c_0 \cdot \sin(\sqrt{|\lambda|} \cdot \pi)$ gives a non-trivial solution if $\sqrt{|\lambda|}$ is an integer. Therefore, the solution is $X(x) = c_0 \cdot \sin(n \cdot x)$ with $n \in \mathbb{N}$ and $\lambda = -n^2$.

Combining these two solutions and summing over all possible values gives the following form:

$$f(t,x) = \sum_{n=1}^{\infty} c_i \cdot e^{-n^2 \cdot t} \cdot \sin(n \cdot x).$$

If we put $t = 0$, then we see $f(0,x) = \sum_{n=1}^{\infty} c_i \cdot \sin(n \cdot x)$, and thus the coefficients c_i of the solution are the Fourier coefficients d_i of ϕ.

Bibliography

[AJ94] Samson Abramsky and Achim Jung, *Domain theory*, Handbook of Logic in Computer Science **3** (1994), 1–168.

[CL55] Earl A. Coddington and Norman Levinson, *Theory of Ordinary Differential Equations*, McGraw-Hill, 1955.

[EK99] Abbas Edalat and Marko Krznarić, *Numerical Integration with Exact Real Arithmetic*, Automata, languages and programming (1999), 702–702.

[EKL03] Abbas Edalat, Marko Krznarić, and André Lieutier, *Domain-theoretic Solution of Differential Equations (Scalar Fields)*, Proceedings of MFPS XIX **83** (2003).

[EL04] Abbas Edalat and André Lieutier, *Domain theory and differential calculus (functions of one variable)*, Mathematical Structures in Computer Science **14** (2004), no. 6, 771–802.

[EP05] Abbas Edalat and Dirk Pattinson, *Domain theoretic solutions of initial value problems for unbounded vector fields*, Proc. MFPS XXI, Electr. Notes in Theoret. Comp. Sci, 2005.

[EP06] _____, *A domain theoretic account of eulerâĂŹs method for solving initial value problems*, Proc. PARA 2004, volume 3732 of Lecture Notes in Comp. Sci, 2006, pp. 112–121.

[ER98] Abbas Edalat and Fabien Rico, *Two Algorithms for Root Finding in Exact Real Arithmetic*, Third Real Number and Computer Conference (1998), 27–44.

[ES99] Abbas Edalat and Philipp Sünderhauf, *A domain-theoretic approach to computability on the real line*, Theoretical Computer Science **210** (1999), no. 1, 73–98.

[GG83] Theodore W. Gamelin and Robert E. Greene, *Introduction to Topology*, Dover Books on Mathematics Series, Dover Publications, Incorporated, 1983.

[Gos72] Ralph W. Gosper, *Continued Fraction Arithmetic*, HAKMEM Item 101B, MIT Artificial Intelligence Memo **239** (1972).

[Grz55] Andrzek Grzegorczyk, *Computable functionals*, Fundamenta Mathematicae **42** (1955), 168–202.

[Hun13] Willem Hundsdorfer, *Numerical Methods*, 2012/2013, Available from: `http://homepages.cwi.nl/~willem/Coll_NM13/Notes12.pdf`.

[Kel55] John L. Kelley, *General Topology*, D. van Nostrand, 1955.

[Olv12] Peter J. Olver, *Applied Mathematics Lecture Notes*, 2012, Available from: `http://www.math.umn.edu/~olver/appl.html`.

[Pat05] Dirk Pattinson, *Domain-theoretic Formulation of Linear Boundary Value Problems*, New Computational Paradigms, Springer, 2005, pp. 385–395.

[PE96] Peter John Potts and Abbas Edalat, *Exact Real Arithmetic based on Linear Fractional Transformations*.

[PEE97] Peter John Potts, Abbas Edalat, and Martín Hötzel Escardó, *Semantics of Exact Real Arithmetic*, Logic in Computer Science, 1997. LICS '97. Proceedings., 12th Annual IEEE Symposium on, jun-2 jul 1997, pp. 248 –257.

[PER83] Marian Boykan Pour-El and Ian Richards, *Computability and Noncomputability in Classical Analysis*, Trans. Am. Math. Soc **275** (1983), 539–560.

[PER84] Marian Bokykan Pour-El and Ian Richards, *L^p-computability in Recursive Analysis*, Proceedings of the American Mathematical Society **92** (1984), no. 1, 93–97.

[Phi03] George M. Phillips, *Interpolation and Approximation by Polynomials*, vol. 14, Springer, 2003.

[Piv05] Marcus Pivato, *Linear Partial Differential Equations and Fourier Theory*, Cambridge University Press, 2005.

[Pot98] Peter John Potts, *Exact Real Arithmetic using Möbius Transformations*, Ph.D. thesis, Imperial College London, 1998, 1998.

[Rud64] Walter Rudin, *Principles of Mathematical Analysis*, vol. 3, McGraw-Hill New York, 1964.

[Tsu02] Hideki Tsuiki, *Real Number Computation through Gray Code Embedding*, Theoretical Computer Science **284** (2002), no. 2, 467–485.

[Wei00] Klaus Weihrauch, *Computable Analysis, An Introduction*, Springer, 2000.

[Wil04] Stephen Willard, *General Topology*, Dover Publications, 2004.

Index

B_n, 24
I^1, 10
P_{f,y_0}, 17
T_0-space, 4
T_A, 14
T_D, 14
T_M, 14
T_S, 14
Δ_h, 24
δ^A_{Cp}, 11
δ^A_{C}, 11
δ^A_{co}, 10
$[P \to Q]$, 19
$\equiv$, 9
$\equiv_\iota$, 9
ι, 8
$\mathrm{I}\mathbb{R}$, 20
$\cdot_L$, 13
$\leq$, 9
$\leq_\iota$, 9
$\twoheaddownarrow$, 19
$\downarrow$, 18
$\mathbb{M}^+$, 11
$\mathbb{M}$, 11
$\mathbb{T}^+$, 11
$\mathbb{T}$, 11
$\mathbb{V}^+$, 11
$\mathbb{V}$, 11
ν^A_{Pg}, 11
$\nu_{\mathbb{Z}}$, 9
$\nu_{\mathbb{N}}$, 9
$\nu_{\mathbb{Q}}$, 9
ν_{Co}, 10
$\mathrm{Pg}([a,b] \to \mathbb{R})$, 10
$\mathrm{P}([a,b] \to \mathbb{R})$, 24
ρ, 10
ρ_{Cf}, 10
$\cdot_R$, 13
$\sim$, 12
$\bigsqcup$, 18
$\twoheaduparrow$, 19
$\uparrow$, 18
$\ll$, 19
$s(f,g)$, 33
$t(f,g)$, 33

admissible, 9

base, 3
basis, 19
Bernstein operator, 24
Bernstein polynomial, 24

Cantor topology, 5
Cauchy-Peano existence theorem, 16
chain, 18
compact
  topological space, 4
compact-open topology, 6
computable element, 7, 8
computable function, 7, 8
connected
  component, 5
  topological space, 5
consistent
  linear single-step functions, 32
  single-step functions, 20
continuous, 4
continuous domain, 19

dcpo, *see also* directed complete partial order
differential equation, 16
  initial value problem, 16
directed complete partial order, 18
directed set, 18
discrete topology, 5

effective convergence, 11
effective topological space, 8
effective uniform continuity, 21
Euclidian topology, 6
Euler's method, 30
expression tree, 14
exterior, 3

finiteness property, 8
forward step operator, 24

head, 14
heat equation, 37

implicit trapezoid rule, 30
indefinite integral, 33
  single-step function, 32
integral form, 16
interior, 3
interpolation property, 19
interval domain, 20

lft, *see also* linear fractional transformation
linear fractional transformation, 12
Lipschitz condition, 17
locally connected, 5

Mean Value Theorem, 22
mediant, 13

naming system, 8
normal product, 13
notation, 8

partially ordered set, 18
  pointed partially ordered set, 18
Picard operator, 17
Picard-Lindelöf theorem, 17
pointwise computable functions, 24
poset, *see also* partially ordered set
pseudoinverse, 13

quadrature formula, 28

rational polygon, 10
representation, 8
Runge-Kutta method, 29

Scott topology, 19
second countable, 4
Simpson's rule, 29
standard representation, 8
step function, 20
  single-step function, 19
subbase, 3
subspace topology, 4

tail, 14
tensor strategy, 14
topological space, 3
transpose, 13
trapezoidal rule, 29
tupling function, 8
Turing machine, 7
type-2 machine, 7

way-below relation, 19
Weak computable Weierstrass Theorem, 25