Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Master in Advanced Mathematics and Mathematical Engineering

Master's thesis

Algorithmic geometry with infinite time computation

Adrian Tobar Nicolau

Supervised by Clemens Huemer and Moritz Mueller

June, 2020

Thanks to the people fighting COVID-19.

Abstract

In this project we do an algorithmic study of problems from computational geometry with countably infinite input, especially countable sets in $\mathbb{R}^n$. To do so, we use the infinite time Blum-Shub-Smale (ITBSS) machine, which is capable to extend computations to infinite time. We present this framework, explained with several algorithms, some results on the ITBSS Machine, and a storage system capable of encoding, editing and extracting sequences of real numbers.

We study different geometric problems, giving algorithmic solutions to several of them. The accumulation points problem in $\mathbb{R}^2$ is presented and solved for countable sets with finitely many accumulation points. Also, the convex hull problem is studied. We show how to compute the closure of the convex hull of countable bounded sets in $\mathbb{R}^n$. The non-crossing perfect matching problem with infinite input is addressed as well.

Keywords

Algorithms, Infinite Computation, Infinite input, Infinite Time, Computational Geometry, Blum-Shub-Smale, Accumulation point, Convex Hull, Matching

Contents

1 Introduction **4**

2 Ordinals **8**

3 Computational model **14**
- 3.1 BSS machine 14
- 3.2 ITBSS machine 16
- 3.3 Some results on the ITBSS machine 18
- 3.4 Observations and basic algorithms 19
- 3.5 How the machine stores data 20
- 3.6 Inputs and technical details 26
- 3.7 Scroll through a list 27

4 Accumulation points **29**

5 Convex Hull **37**
- 5.1 Convex hull computation 38

6 Matchings **42**
- 6.1 Stable Marriage 44

7 Conclusions **48**

8 Bibliography **49**

A Density of rationals **51**

B Auxiliar algorithms **53**

List of Algorithms

1 Absolut value of a real number . . . 16
2 Computation of $\frac{\pi^2}{6}$ . . . 17
3 Value of π using the Leibniz formula . . . 19
4 Code of a real . . . 22
5 Real of a code . . . 22
6 Extract one code . . . 24
7 Extract one digit . . . 24
8 Alter a code . . . 24
9 Modify a digit . . . 26
10 Supremum of a countable bounded set . . . 27
11 Inverse of the Cantor pairing function used in Proposition 3.22 . . . 28
12 Mass center of a set in $\mathbb{R}^2$ . . . 31
13 Removing an accumulation point . . . 32
14 Deciding if a segment L has an accumulation point of X . . . 33
15 Finding the accumulation points of X on a segment L . . . 34
16 Compute the finite number of accumulation points of a bounded set . . . 35
17 Compute the finite number of accumulation points of a set . . . 35
18 Decide if $x \in CH(X) \subseteq \mathbb{R}^n$ . . . 38
19 Decide if $x \in \overline{CH(X)} \subset \mathbb{R}^n$ with $int\, CH(X) \neq \emptyset$ . . . 38
20 Compute closure of convex hull as a countable intersection of rational halfspaces . . . 40
21 Well order . . . 46
22 Infinite Stable Marriage Algorithm . . . 47
23 Exponentiation . . . 53
24 Floor function . . . 53
25 Number of integer-digits of a number . . . 54
26 Binary expansion of a natural number as a natural number . . . 54
27 Square root of a non-negative real number . . . 54

1. Introduction

The objective of this project is doing computational geometry with infinite input size, in particular countable infinite subsets of $\mathbb{R}^n$. In computational geometry diverse problems on point sets are studied, being convex hull calculation a well-known example. The convex hull problem studies how to find the smallest convex set containing a particular finite set, see Figure 1. This problem has been widely studied since the 1970s [21] and several optimal algorithms have been discovered, see for instance [4, 5]. In this project, we present the convex hull problem and other geometrical problems for point sets which are infinite countable.

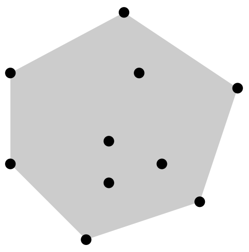

Figure 1: Convex hull, in gray, of a finite set of points in $\mathbb{R}^2$.

To do computational geometry with infinite sets a framework is needed. The classical computational models, such as Turing machines [22], Random acces machines [6] and Blum-Shub-Smale machines [15] are not suitable to be such framework, since their computations are limited to only finitely many steps. The model used in this project is the Infinite-Time-Blum-Shub-Smale (ITBSS) machine, an extension of the BSS machine with the capacity to extend the computation to infinitely many steps.

In this project we study three main geometric problems: accumulation points, convex hull and matchings.

Accumulation points, as limit points of a set in $\mathbb{R}^n$, are a proper problem of infinite input computation. The accumulation points problem that we study is how to find the accumulation points of a set in $\mathbb{R}^n$. The main difficulty that emerges from this problem is that the accumulation points of a set do not have to be elements in the set, otherwise an easy computation could check for each point in the set if it is an accumulation point and give the solution. From this difficulty it is necessary to study the relative position of the elements in the set to determine where the accumulation points are. We prove that in the ITBSS model, the accumulation point problem can be solved for point sets with a finite number of accumulation points.

The convex hull problem for infinite sets of points presents several difficulties. In general, it is not possible to report all the points on the boundary of the convex hull. Consider, for example, a closed ball in $\mathbb{R}^2$. To overpass this limitation we use a countable family of halfplanes, with common intersection the solution set, as a method to report the closure of the convex hull. Since this method is limited to closed sets, we give a partial solution to the convex hull problem showing that the ITBSS machine can solve the convex hull problem for the closure of convex hulls of countable sets.

As the third studied geometric problem, we present diverse matching problems. A non-crossing perfect matching between two disjoint point sets is a bijection between the two sets, such that segments which connect paired points do not intersect, see Figure 2. We show several counterexamples to the existence of a non-crossing perfect matching, and of related non-crossing matchings.

We end with a positive result in a non-geometric problem: stable matchings. As originally defined in

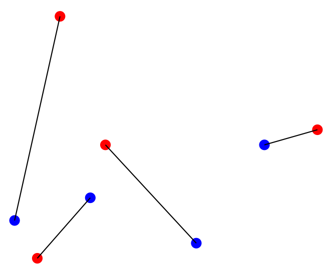

Figure 2: Non-crossing perfect matching between two finite sets with colors red and blue respectively.

[9], the stable matching problems consists in finding a correspondence between two sets such that certain preferences are satisfied. We present the stable matching for infinite sets as done in [3] and prove that the stable matching problem can be solved by a ITBSS machine as long as certain conditions on the preferences are satisfied.

Computational geometry with infinite input has certain differences with finite computational geometry. Starting with the convex hull, the solution for finite instances of this problem is always a convex polyhedron, not like for the infinite instances where, for example, the set of rational points, in $\mathbb{R}^2$, inside the open ball with center $(0, 0)$ and radius 1 has convex hull a disk. The accumulation points problems is in radical difference with the finite computation since this problem does not have a finite counterpart. The non-crossing perfect matching problem for the point sets of infinite size does not always have a solution, not like in the finite case, where under the assumption of general position, i.e., there are no three points aligned, the existence of a solution is guaranteed.

More differences appear when studying other problems, or when trying to adapt elementary tools from finite computation to the infinite case. For example, finite sets of points can be sorted and a least element can be identified. This cannot be done for most infinite sets. Consider, for example, the set of rational points in the interval $(0, 1)$ where no least element exists. In general, in infinite computational geometry some properties, that are assumed in the finite case, do not hold. A minimum distance among the points of a set does not have to exist. An input can be unbounded. An input set can have accumulation points, and the set of accumulation points can be uncountable. For these reasons during this project several restrictions are imposed to the input sets to guarantee that the algorithms work properly.

For these and many other reasons, the finite computational model and the infinite one are substantially different, and are interesting as independent fields of study.

The problems commented so far are only a very small subset of all the geometric problems that can be studied for infinite inputs. Problems such as the existence of minimal spanning tree [12], or the stucture of Voronoi diagrams [23] have also been studied for infinite sets. With all commented before, we see that computational geometry with infinite input size can be a fertile ground for many studies on several different problems not only extending the finite ones but also unique questions that emerge from the infinity of the input.

The organization of this projects can be divided in two main parts:(i) the model of computation,(ii) the study of problems. Section 2 gives an introduction to ordinal numbers which are necessary for a full understanding of the model. The main operations and properties of ordinals are presented, showing the well known relation between ordinals and well-ordered sets. Section 3 presents the Blum-Shub-Smale machine model and its generallization to the Infinite Time Blum-Shub-Smale machine [15]. This presentation shows several examples of computations and some of their properties. It ends with a discussion of the capacity of the model to work with countable sets.

Sections 4, 5, and 6 are studies on different problems from the field of computation and geometry. Starting with accumulation points in Section 4, different algorithms are presented, with the final Algorithm 17 capable of finding the accumulation points of a countable set under reasonable conditions. Section 5 covers the problem of convex hull computation of a countable set X, i.e. finding the smallest convex set that contains X. A partial solution to the countable case is presented under reasonable conditions of boundedness. In Section 6 diverse matching problems are presented, giving a partial solution to the stable matching problem, using results in [3]. To end, Section 7 is a conclusion that sums up the results obtained and possible future work. Additionally two appendices with some auxiliar results and algorithms are presented.

List of symbols

$\mathbb{N}, \mathbb{Q}, \mathbb{R}$: natural, rational and real numbers.

$\mathbb{N}_{>0}$: non-zero naturals

Ord: class of ordinals

ω: first limit ordinal

$\alpha, \beta, \gamma ...$: ordinals

$X, Y, Z, ...$: sets

$a, b, c, ..., x, y, z$: elements of a set

$< d >$: code of a real number

$\pi(x, y)$: Cantor pairing function

$\sup X$: supremum of X

$\inf X$: infimum of X

$\overline{X}$: closure of X

$int(X)$: interior of X

$CH(X)$: convex hull of X

$d(x, y)$: Euclidean distance between x and y

$d(x, Y)$: infimum of the distances from x to points of Y

$d(X, Y)$: infimum of the distances from points of X to points of Y

$B(x, r)$: open ball with center x and radius r.

$\overline{B}(x, r)$: closed ball with center x and radius r.

$H(\vec{c}, d)$: hyperplane with normal vector $\vec{c}$ and containing point d

$H(\vec{c}, d, \leq)$: lower halfspace

$H(\vec{c}, d, \geq)$: upper halfspace

$H(\vec{c}, d, <)$: strictly lower halfspace

$H(\vec{c}, d, >)$: strictly upper halfspace

BSS: Blum-Shub-Smale

ITBSS: Infinite Time Blum-Shub-Smale

2. Ordinals

This text uses ordinals in the definition and proof of some results. To ensure clarity in the notation and arguments a brief introduction to the ordinals is made. This section is based on [13] and [10].

Definition 2.1. A binary relation $<$ on a set P is a partial order of P if:

(i) $p \nless p$ for any $p \in P$.

(ii) If $p < q$ and $q < r$, then $p < r$.

$(P, <)$ is called a partially ordered set. A partial order $<$ of P is a linear order if moreover

(iii) $p < q$ or $p = q$ or $q < p$ for all $p, q \in P$.

A linear order $<$ of a set P is a well-order if every non-empty subset of P has a least element, i.e. an element a such that for all $p \in P$ $a \leq p$ ($a \leq b$ if and only if $a = b$ or $a < b$).

Definition 2.2. An element a is an upper bound of X if $x \leq a$ for all $x \in X$. The supremum of X is the least upper bound. The supremum of X is denoted $\sup X$.

Definition 2.3. A set T is transitive if every element of T is a subset of T, that is, if $x \in T$ and $y \in x$, then $y \in T$. An ordinal is a transitive set X such that the binary relation $\{(x, y) \in X \times X \mid x \in y\}$ is a well-order on X. The class of all ordinals is denoted by Ord.

Remark 2.4. We use the small Greek letters $\alpha, \beta, \gamma, ...$ to range over ordinals. We use $\alpha < \beta$ when $\alpha \in \beta$ and $\alpha \leq \beta$ when $\alpha \in \beta$ or $\alpha = \beta$.

We denote as $0 = \emptyset$, $1 = \{0\}$, $2 = \{0, 1\}$ and so on.

Lemma 2.5.

(i) $0 = \emptyset$ *is an ordinal.*

(ii) If α is an ordinal and $\beta \in \alpha$, then β is an ordinal.

(iii) If $\alpha \neq \beta$ are ordinals and $\alpha \subseteq \beta$ then $\alpha \in \beta$.

(iv) If α, β are ordinals, then either $\alpha \subseteq \beta$ or $\beta \subseteq \alpha$.

(v) If X is a non-empty set of ordinals, then $\bigcup X = \bigcup_{z \in X} z$ is an ordinal and $\bigcup X = \sup X$.

Proof.

(i) Obvious.

(ii) β is a subset of a transitive and well-ordered set, that implies that it is also transitive and well-ordered.

(iii) If $\alpha \subseteq \beta$, let γ be the least element of the set $\beta - \alpha$. Let us prove $\alpha = \gamma$.

$\gamma \subseteq \alpha$: Let $x \in \gamma$. Since $\gamma \in \beta$, from the transitivity of β we have $x \in \beta$. As $x < \gamma$, the choice of γ implies $x \in \alpha$.

$\alpha \subseteq \gamma$: Let $x \in \alpha$. If $x = \gamma$ or $x > \gamma$, then $\gamma \in \alpha$ since α is transitive which is a contradiction with the definition of γ.

(iv) Let $\gamma = \alpha \cap \beta$. Note that γ is an ordinal. Assume that γ is different from α and β. Since γ is a subset of α and β, (iii) implies that it is an element of α and β. But if γ is an element of α and β it is also an element of the intersection, then $\gamma \in \gamma$, a contradiction with $\in$ being a strict order.

We deduce that $\alpha \cap \beta = \alpha$ or $\alpha \cap \beta = \beta$ which implies $\alpha \subseteq \beta$ or $\beta \subseteq \alpha$, in both cases we conclude (iv).

(v) Since each element of $\bigcup X$ is contained in an ordinal $\alpha \in X$, it is an ordinal. Then $\bigcup X$ is a set of ordinals. Let α, β, γ be ordinals in $\bigcup X$, clearly $\alpha \notin \alpha$ and from transitivity of ordinals if $\alpha \in \beta$ and $\beta \in \gamma$ then $\alpha \in \gamma$. Now (iii) and (iv) imply that $\alpha = \beta$ or $\alpha \in \beta$ or $\beta \in \alpha$. Since all these properties hold for any choice of α, β, γ, we conclude that $\in$ is a linear order. Furthermore, $\bigcup X$ is well-ordered by $\in$ because every non-empty set x has a $\in$-least element, namely, an element $y \in x$ such that $y \cap x = \emptyset$ (this is the so-called axiom of foundation in ZFC).

To see transitivity consider $\beta \in \bigcup X$. Since β is in $\bigcup X$ then there exists an ordinal $\alpha \in X$ such that $\beta \in \alpha$. From here, we know $\alpha \subseteq \bigcup X$ and since α is an ordinal $\beta \subseteq \alpha$. We conclude $\beta \subseteq \bigcup X$. All together $\bigcup X$ is an ordinal.

To see $\bigcup X = \sup X$ first we see that it is an upper bound. Suppose $\alpha \in X$. Then $\alpha \subseteq \bigcup X$.

To end we show that it is the least upper bound. Let γ be an upper bound and $\beta \in \bigcup X$. Since $\beta \in \alpha$ for some $\alpha \in X$, from the definition of γ we have $\alpha \leq \gamma$ which implies $\alpha \subseteq \gamma$. Finally we conclude $\beta \in \gamma$. Since this holds for β arbitrary we have $\bigcup X \subseteq \gamma$. □

Definition 2.6. An ordinal α is a successor if there exists β such that $\alpha = \beta \cup \{\beta\}$. If α is not a successor ordinal, then is called a limit ordinal.

Definition 2.7. We denote the least non-zero limit ordinal ω.

The ordinals less than ω are called natural numbers. Now we present the basic operations. To do so, we need some previous definitions and results.

Theorem 2.8. *Let G be a function from sets to sets and α_0 be an ordinal. Then there exists a unique function $F(\beta)$ from the ordinals to the ordinals such that for all ordinals α, β, λ such that $\alpha = \beta \cup \{\beta\}$ and $\lambda = \bigcup \lambda$:*

$$\begin{aligned} F(0) &= \alpha_0 \\ F(\alpha) &= G(F(\beta)) \\ F(\lambda) &= \sup_{x_i < \lambda} F(x_i). \end{aligned}$$

This theorem, intuitively, says that given a function and a starting value we can uniquely construct a function F over all Ord recursively, i.e., using at each step the previous values of F.

We state now the operations

Definition 2.9. (Addition). For all ordinal numbers α

(i) $\alpha + 0 = \alpha$,

(ii) $\alpha + (\beta + 1) = (\alpha + \beta) + 1$, for all β,

(iii) $\alpha + \beta = \sup\{\alpha + \xi \mid \xi < \beta\}$, for all limit ordinals $\beta > 0$.

Definition 2.10. (Multiplication). For all ordinal numbers α

(i) $\alpha \cdot 0 = 0$,

(ii) $\alpha \cdot (\beta + 1) = (\alpha \cdot \beta) + \alpha$, for all β,

(iii) $\alpha \cdot \beta = \sup\{\alpha \cdot \xi \mid \xi < \beta\}$, for all limit ordinals $\beta > 0$.

Definition 2.11. (Exponentiation). For all ordinal numbers α

(i) $\alpha^0 = 1$,

(ii) $\alpha^{\beta+1} = \alpha^\beta \cdot \alpha$, for all β,

(iii) $\alpha^\beta = \sup\{\alpha^\xi \mid \xi < \beta\}$, for all limit ordinals $\beta > 0$.

Now, these operations may look hard to compute or understand. To give a more intuitive way of understanding the ordinals we present the following results.

Definition 2.12. Let $(P, <), (Q, <')$ be two linearly ordered sets. A function $f : P \to Q$ is increasing if for all $x, y \in P$, $x < y$ implies $f(x) <' f(y)$. If f is bijective, P and Q are isomorphic.

Definition 2.13. Let $(W, <)$ a well-ordered set and $u \in W$, then $W(u) = \{x \in W \mid x < u\}$ is the initial segment of W given by u.

Lemma 2.14. *If $(W, <)$ is a well-ordered set and $f : W \to W$ is an increasing function, then $f(x) \geq x$ for all $x \in W$.*

Proof. Suppose otherwise, then the set $X = \{x \in W \mid f(x) < x\}$ is non-empty, let z be the least element of X. We have $w = f(z) < z$ and now since $w < z$ we also have $f(w) < f(z) = w$ a contradiction with z being the least element of X. □

Corollary 2.15. *The only automorphism, i.e. isomorphism from a set into itself, of a well-ordered set is the identity.*

Proof. Let f be an automorphism. From Lemma 2.14 $f(x) \geq x$ and $f^{-1}(x) \geq x$. □

Corollary 2.16. *If two well-ordered sets are isomorphic, the isomorphism is unique.*

Proof. Let f, g be isomorphisms. It follows from Corollary 2.15 that the $f^{-1}(g(x)) = g^{-1}(f(x)) = x$ which implies $f = g$. □

Lemma 2.17. *No well-ordered set is isomorphic to an initial segment of itself.*

Proof. Let $(W, <)$ be isomorphic to $\{x \in W \mid x < u\}$ for some $u \in W$. Then the function that defines the isomorphism satisfies $f(u) < u$, getting a contradiction with Lemma 2.14. □

Lemma 2.18. *Let W be a well-ordered set, $x \in W$, $f : W(x) \to \alpha_x$ an isomorphism between an initial segment and an ordinal, let $y \in x$ an ordinal, and $g : W(y) \to g(W(y))$ the restriction of f to $W(y)$. Then g is an isomorphism between an ordinal in α_x and $W(y)$.*

Proof. The function g is an isomorphism to an initial segment of α_x. To prove it, suppose otherwise. Then, there exists $z < y$ and $\beta < g(z) = f(z)$ such that β is not in the image of g. Let $u \in W(x)$ such that $f(u) = \beta$, then $u \notin W(y)$, as a consequence $u > z$ and $f(u) < f(z)$, a contradiction with f being an isomorphism.

We deduce that the image of g is transitive and, from Lemma 2.5 (iii), we conclude that $W(y)$ is isomorphic to $\alpha_y \in \alpha_x$ an ordinal. □

Theorem 2.19. *Every well-ordered set is isomorphic to a unique ordinal number.*

Proof. Given a well-ordered set W, we define $F(x) = \alpha$ if α is isomorphic to the initial segment $W(x)$. To see that for all $x \in W$ such α_x exists suppose otherwise. Let x be the least element without α_x. For each $y < x \in W$ let f_y be an isomorphism of $W(y)$ with α_y.

For $y_0 < y_1 < x$ let us prove $f_{y_0} \subseteq f_{y_1}$. From Lemma 2.18 we have seen that the restriction g_{y_0} of f_{y_1} to $W(y_0)$ is an isomorphism to an ordinal β. From Corollaries 2.15 and 2.16 we conclude that $\beta = \alpha_{y_0}$ and $g_{y_0} = f_{y_0}$.

We know that $\bigcup_{y<x} f_y$ is an isomorphism with domain $D = \bigcup_{y<x} W(y)$ and image $\gamma = \bigcup_{y<x} \alpha_y = \sup_{y<x} \alpha_y$ an ordinal by Lemma 2.5 (v) .

Now if $D = W(x)$ we reach a contradiction with the definition of x. This is the case if there exists no greatest element under x.

Otherwise $W(x)$ contains a maximal element z and then, we have an isomorphism $f_x = f_y \cup \{(z, \gamma)\}$ from $W(x)$ to $\gamma + 1$, again a contradiction with the definition of x. □

The original definitions of basic operations give almost no intuition in how they behave. To have a better understanding on how these operation work we present the following known results.

Definition 2.20. Let $\alpha, \beta \in Ord$, then $\alpha \oplus \beta$ is the unique ordinal isomorphic to the total order $<$ on $\alpha \times \{0\} \cup \beta \times \{1\}$ defined as

$$(x, b) < (x', b') \text{ if and only if } b = 0 \text{ and } b' = 1, \text{ or } b = b' \text{ and } x \in x'.$$

Proposition 2.21. *The operation $\oplus$ is well defined.*

Proof. We show first that $<$ is a well-order and then that such an ordinal exists.

From the definition of $<$ clearly for any two elements a, b of $\alpha \times \{0\} \cup \beta \times \{1\}$ we have $a < b$ or $a > b$ or $a = b$ and $a \not< a$. The transitivity, i.e., $a < b$ and $b < c$ implies $a < c$, is a straightforward case analysis. Then we only have to prove that each set has a least element. Let X be a non-empty subset. If $(x, 0) \in X$ for some $x \in \alpha$, let y be the least element $y \in \alpha$ such that $(y, 0) \in X$, then $(y, 0)$ is the least element in X. Otherwise let $y \in \beta$ least element such that $(y, 1) \in X$, then $(y, 1)$ is the least element in X.

Now from Theorem 2.19 there exists a unique ordinal isomorphic to the well-ordered set $(\alpha \times \{0\} \cup \beta \times \{1\}, <)$. □

Proposition 2.22. *For any $\alpha, \beta \in Ord$ the equality $\alpha + \beta = \alpha \oplus \beta$ holds.*

Proof. The proof is done by induction over β. For $\beta = 0$ we have $\alpha + 0 = \alpha \oplus 0$ equal to α. Similarly, $\alpha + 1 = \alpha \oplus 1$ also holds since the function $f(\gamma) = (\gamma, 0)$ for $\gamma < \alpha$ and $f(\alpha) = (1, 1)$ is an isomorphism. Let $\beta = \gamma + 1$, we know by induction hypothesis that $\beta \oplus \gamma = \beta + \gamma$. Now, $(\alpha + \beta) + 1 = (\alpha \oplus \beta) \oplus 1$ also holds by induction hypothesis.

When $\beta = \sup_{\gamma<\beta} \gamma$ is a limit ordinal we know that for each γ there is an isomorphism $f_\gamma : \alpha+\gamma \to \alpha\oplus\gamma$. From Lemma 2.18 we know that if $\gamma < \gamma' < \beta$ then $f_\gamma \subseteq f'_\gamma$. Then $f_\beta = \bigcup_{\gamma\in\beta} f_\gamma$ is a function with domain

$$\bigcup_{\gamma\in\beta} \alpha + \gamma = \alpha + \beta.$$

Its range is

$$\begin{aligned}\bigcup_{\gamma\in\beta} \alpha \times \{0\} \cup (\gamma \times \{1\}) &= (\alpha \times \{0\}) \cup \bigcup_{\gamma\in\beta} (\gamma \times \{1\}) \\ &= (\alpha \times \{0\}) \cup \left((\bigcup_{\gamma\in\beta} \gamma) \times \{1\} \right) \\ &= (\alpha \times \{0\}) \cup (\beta \times \{1\}) \\ &= \alpha \oplus \beta.\end{aligned}$$

And $\bigcup_{\gamma\in\beta} \alpha \oplus \gamma$ is exactly $\alpha \oplus \beta$. From construction, f is an isomorphism from $\alpha + \beta$ to $\alpha \oplus \beta$. $\square$

Definition 2.23. Let $\alpha, \beta \in Ord_{>0}$, then $\alpha \otimes \beta$ is the unique ordinal isomorphic to the well-order $<$ on $\alpha \times \beta$ defined as

$$(\gamma, \eta) < (\gamma', \eta') \in \alpha \times \beta \text{ if and only if } \eta < \eta', \text{ or } \eta = \eta' \text{ and } \gamma < \gamma'.$$

Proposition 2.24. *The operation $\otimes$ is well defined.*

Proof. From the definition of $<$ clearly for any two elements a, b of $\alpha \times \beta$ we have $a < b$ or $a > b$ or $a = b$ and $a \not< a$. The transitivity, i.e., $a < b$ and $b < c$ implies $a < c$, is a straightforward case analysis. We only have to prove that each non-empty set has a least element. Let X be a non-empty subset of $\alpha \times \beta$ and let $\xi = \min\{\eta \in \beta | (\gamma, \eta) \in X\}$ and $\theta = \min\{\eta \in \alpha \mid (\gamma, \eta) \in X \wedge \gamma = \xi\}$, both elements exists because $\{\eta \in \beta | (\gamma, \eta) \in X\}$ and $\{\eta \in \alpha \mid (\gamma, \eta) \in X \wedge \gamma = \xi\}$ are non-empty subsets of β and α respectively. Observe that $(\theta, \xi) \in X$ and for all $(\gamma, \eta) \in X$ we have $\xi \leq \eta$ and $\theta \leq \gamma$. We conclude that (θ, ξ) is the least element of X.

Now from Theorem 2.19 there exists a unique ordinal isomorphic to the well-ordered set $(\alpha \times \beta, <)$. $\square$

Lemma 2.25. *The equality $\alpha \otimes (\beta + 1) = (\alpha \otimes \beta) \oplus \alpha$ holds for any $\alpha, \beta \in Ord$.*

Proof. We have that $\alpha \otimes (\beta + 1)$ is isomorphic to $\alpha \times (\beta + 1)$ with total order $<$ of Definition 2.23 and $(\alpha \otimes \beta) \oplus \alpha$ is isomorphic to $((\alpha \times \beta) \times \{0\}) \cup (\alpha \times \{1\})$ where $\alpha \times \beta$ has total order $<'$ of Definition 2.23 and, the whole set, total order $<''$ of Definition 2.20.

Consider the function

$$F : \alpha \times (\beta + 1) \to ((\alpha \times \beta) \times \{0\}) \cup (\alpha \times \{1\})$$

$$F(x, y) = \begin{cases} ((x, y), 0) & \text{if } x \in \alpha \text{ and } y \in \beta, \\ (x, 1) & \text{if } x \in \alpha \text{ and } y = \beta. \end{cases}$$

F is order preserving and bijective. It implies that both ordered sets are isomorphic and we conclude that $\alpha \otimes (\beta + 1) = (\alpha \otimes \beta) \oplus \alpha$. □

Proposition 2.26. *For any $\alpha, \beta \in Ord$ the equalities $\alpha \otimes \beta = \alpha \cdot \beta$ holds.*

Proof. As shown in Proposition 2.24 relation $<$ from Definition 2.23 is a well-order.

The proof is done by induction over β. For $\beta = 1$ we have $\alpha \otimes 1 = \alpha \cdot 1 = \alpha$. For the general case

$$\begin{aligned} \alpha \cdot (\beta + 1) &= (\alpha \cdot \beta) + \alpha \\ &= (\alpha \otimes \beta) + \alpha \\ &= (\alpha \otimes \beta) \oplus \alpha \\ &= \alpha \otimes (\beta + 1). \end{aligned}$$

Where the second equality holds by induction hypothesis, the third from Proposition 2.22 and the last from Lemma 2.25.

When β is a limit ordinal the argument is analogous to the one given for $\oplus$. □

Corollary 2.27. *The operations $+, \cdot$ and $\oplus, \otimes$ are, respectively, equal.*

To end we present the notion of limit which will be useful in the definition of our model of computation.

Definition 2.28. Let $\alpha \in Ord$ a limit ordinal and $r : \alpha \to \mathbb{R}$ a function. We say that z is the limit at α, and write

$$\lim_{\xi \to \alpha} r(\xi) = z$$

if and only if for any real $\epsilon > 0$ exists an ordinal $\gamma \in \alpha$ such that for all ordinal $\eta \geq \gamma$ the inequality $|r(\eta) - z| < \epsilon$ holds. The limit of vectors is defined component-wise.

Proposition 2.29. *Let $g : \alpha \to \mathbb{R}$, if $f : \mathbb{R} \to \mathbb{R}$ is continuous at z, and $\lim_{\xi \to \alpha} g(\xi) = z$, then*

$$f(\lim_{\xi \to \alpha} g(\xi)) = f(z) = \lim_{\xi \to \alpha} f(g(\xi)).$$

Proof. The first equality is trivial. Let us prove $f(z) = \lim_{\xi \to \alpha} f(g(\xi))$. Let $\epsilon > 0$, from continuity of f there exists $\delta > 0$ such that $|f(y) - f(z)| < \epsilon$ if $|y - z| < \delta$. Let $y = g(\xi)$. Then, if $|g(\xi) - z| < \delta$ we have $|f(g(\xi)) - f(z)| < \epsilon$. Since $\lim_{\xi \to \alpha} g(\xi) = z$ there exists $\gamma \in Ord$ such that $|g(\xi) - z| < \delta$ if $\xi \geq \gamma$. We conclude that for any $\epsilon > 0$ there exists γ such that $|f(g(\xi)) - f(z)| < \epsilon$ if $\xi \geq \gamma$. □

3. Computational model

The main goal of this text is to explore algorithms for geometric problems with input of infinite size. To do so, an adequate model of computation must be chosen. The typical models of computation cannot be considered since their finiteness in time and capacity make them not powerful enough. This discards the classical Turing machines [22] and the Blum-Shub-Smale machines [1]. Among the diverse options, the ones that hold similarities with the classical ones are their infinite extensions, namely, the Infinite Time Turing Machine which have been presented and studied by Joel David Hamkins and Andy Lewis in [11] and the Infinite Time Blum-Shub-Smale (ITBSS) machine presented and studied by Peter Koepke and Benjamin Seyfferth in [15]. Both models present a natural extension of the finite computation to a transfinite computation able to carry on much more complicated tasks. Nonetheless, both models present some limitation in their capacity to manage infinite size inputs. The ITBSS machine model was chosen for this project, given the relative simplicity to present the programs as a combination of IF and EDIT commands.

In the following sections the capacity and limitations of the ITBSS machine model are shown, all with focus in the construction of algorithms to solve the infinite extension of well known problems of finite algorithmic geometry and some problems unique for the infinite case, in particular, the calculation of accumulation points.

3.1 BSS machine

To explain the final model that will be used we present first the Blum-Shub-Smale (BSS) machine model, originally presented in [1].

We start with an intuitive explanation of the BSS machine which will be later formalized.

A BSS machine needs: an indexed finite list of commands of the form $IF(f, \ell, k)$, $EDIT(f)$ or $HALT$, a finite lists of real numbers called input $(i_0, ..., i_{n-1})$ and a finite set of registers $R = (R_0, R_1, ..., R_{n-1})$.

The machine works as follows. Set the value of the registers to $R = (i_0, ..., i_{n-1})$. Execute the command with index 0. If the instruction is $EDIT(f)$ set the values of the registers $R = f(R)$ and execute the next command in the list. If the command is $IF(f, \ell, k)$ check the condition $f(R) \leq 0$, if it is satisfied, the command with index ℓ is executed next, the instruction k is executed otherwise. This process is continued until a $HALT$ instruction is executed, after that the machine stops. This process is called "computation".

The computation is formalised in the following definitions.

Definition 3.1. A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is rational if it is a fraction of polynomials on each coordinate.

Definition 3.2. A command is an instruction of the form $EDIT(f)$, $IF(g, \ell, k)$ or $HALT$ where $f : \mathbb{R}^n \to \mathbb{R}^n$ is a rational function, $g : \mathbb{R}^n \to \mathbb{R}$ is a rational function and ℓ,k are natural numbers.

A BSS program is a finite list of commands, each one with a distinct natural number associated with it, called index.

Definition 3.3. We define:

- $R(t)$:= the value of the registers after step t.
- $I(t)$:= the command executing after step t.

Now we are able to define computation.

Definition 3.4. Let P be a BSS program. The BSS machine computation by P on some input $(i_0, ..., i_{n-1}) \in \mathbb{R}^n$ is defined as the sequence $(C_t)_{t<L} = (R(t), I(t))_{t<L}$ with $(R(t), I(t)) \in (\mathbb{R}^n \times \mathbb{N})$ for all $t < L$, where:

- $L \in \mathbb{N}$ or $L = \infty$.
- $R(0) = (i_0, .., i_{n-1})$ and $I(0) = 0$.
- $EDIT(f)$ commands always have a function f with image in $\mathbb{R}^n$.
- (EDIT) If $t < L$ and $I(t) = i$ let $EDIT(f)$ be the command of P with index i. Then, $R(t+1) = f(R(t))$ and $I(t+1) = i+1$.
- $IF(f, \ell, k)$ commands always have a function f with image in $\mathbb{R}$.
- (IF) If $t < L$ and $I(t) = i$, let $IF(f, \ell, k)$ be the command of P with index i. Then $R(t+1) = R(t)$ and if $f(R(t)) \leq 0$, then $I(t+1) = \ell$; otherwise $I(t+1) = k$.
- (HALT) If $t < L$ and $I(t) = i$, let $HALT$ be the command of P with index i. Then there exists no t' such that $t < t' < L$.

Definition 3.5. If an execution of P with some input does not create a computation,i.e, some division by zero occurs or the input is not valid or the registers do not converge, we say that the computation diverges.

Remark 3.6. We assume that the denominator of the rational functions of *EDIT* commands do not vanish in any evaluation during the computation. Therefore $f(R(t))$ is well-defined in the EDIT command.

In general, the computation of a program has the objective of extracting some particular information (factorization of a natural number, ordered list, convex hull,...). This information is called output.

Definition 3.7. The output of a computation is the value of the register R_0.

Example 3.8. Consider the simple program P that computes the absolute value with indexed commands

0) $IF(R_0, 1, 2)$

1) $EDIT(f(R) = -R)$

2) $HALT$

The computation of P with input -5, first sets the registers $R = (-5)$, goes to the command 0) and checks $-5 \leq 0$. Since it is satisfied it runs the command 1). The register is updated to $R = (5)$. Finally the HALT command 2) is executed and the process stops. As a sequence $(R(t), I(t))_{t<L}$:

0: $((-5), 0)$

1: $((-5), 1)$

2: $((5), 2)$

Program P with input x can be expressed with pseudocode, see Algorithm 1, where Return indicates which register is the output.

Algorithm 1: Absolut value of a real number

```
    Input: $x \in \mathbb{R}$
    Output: $|x|$.
1 Set $R = (x)$;
2 if $R_0 \leq 0$ then
3     $R_0 = -R_0$;
4 Return $R_0$;
```

3.2 ITBSS machine

We present the Infinite time Blum-Shub-Smale (ITBSS) machine, which will be the one that carries all computations during the rest of the text. The main reference is [15] where it was originally presented.

The ITBSS machine needs the same as a BSS machine, i.e., an indexed finite list of commands of the form $IF(f, \ell, k)$, $EDIT(f)$ or $HALT$, a finite list of real numbers called input $(i_0, ..., i_{n-1})$ and a finite set of registers $R = (R_0, R_1, ..., R_{n-1})$.

The ITBSS machine is an extension of the BSS machine, where the number of steps that can be performed can be transfinite. The ITBSS machine works, for each finite step in the same way as a BSS machine. The main differences appears when an infinite number of steps are performed, in this case the value of the register is updated to a limit value. This limit value is determined by the infinitely many values that are taken before. This limit process is formally presented in the following definition.

Definition 3.9. Let P be a BSS program. The infinite time BSS machine (ITBBS) computation by P on some input $(i_0, ..., i_{n-1}) \in \mathbb{R}^n$ is defined as the transfinite sequence $(C_t)_{t\in\theta} = (R(t), I(t))_{t\in\theta}$ with $(R(t), I(t)) \in (\mathbb{R}^n \times \mathbb{N})$ for all $t \in \theta$, where:

- $\theta \in \text{Ord}$ or $\theta = \text{Ord}$.
- $R(0) = (i_0, ..., i_{n-1})$ and $I(0) = 0$.
- $EDIT(f)$ commands always have a function f with image in $\mathbb{R}^n$.
- (EDIT) If $t < \theta$ and $I(t) = i$, let $EDIT(f)$ be the command of P with index i. Then, $R(t+1) = f(R(t))$ and $I(t+1) = i+1$.
- $IF(f, \ell, k)$ commands always have a function f with image in $\mathbb{R}$.
- (IF) If $t < \theta$ and $I(t) = i$, let $IF(f, \ell, k)$ be the command of P with index i. Then $R(t+1) = R(t)$ and if $f(R(t)) \leq 0$ then $I(t+1) = \ell$; otherwise $I(t+1) = k$.
- (HALT) If $t < \theta$ and $I(t) = i$, let $HALT$ be the command of P with index i. Then there exists no t' such that $t < t' < \theta$.
- If $t < \theta$ is a limit ordinal and $y = \lim_{s\to t} R(s)$, then $R(t) = y$ and $I(t) = \liminf_{s\to t} I(s)$.
- If $\theta < \text{Ord}$, then $\theta = \gamma + 1$ is a successor ordinal and $I(\gamma)$ calls a $HALT$ command.

Remark 3.10. Observe how we are accessing the limit case assuming the existence of a limit in the values of $R(s)$ as s tends to a limit value. This limit clearly does not have to exist, in that case the computation fails.

Example 3.11. Let P be the program with indexed commands

0) $EDIT(R_0 + R_1^2, \frac{R_1}{R_1+1})$

1) $IF(R_1, 2, 0)$

2) $HALT$

and registers $R = (0, 1)$. The execution will set $R = (0, 1)$. The first command will be carried on updating $R = (1, \frac{1}{2})$. The IF instruction will send us to command 0) for any finite step. Notice how the values of the registers will be updated to $(\sum_{n=1}^{t} \frac{1}{n^2}, \frac{1}{n+1})$. The machine will update the values of the registers to the limit values, obtaining $(\sum_{n=1}^{\infty} \frac{1}{n^2}, 0)$. From Definition 3.9, the next step that will be carried on will be the $\liminf_{s\to\omega} I(s) = 0$. In this command the registers will remain unchanged. Finally the condition of the *IF* command 1) will be satisfied and the computation will stop after executing the *HALT* command.

As a sequence, the computation is given by

$$(R(t), I(t)) = \left(\left(\sum_{n=1}^{t} \frac{1}{n^2}, \frac{1}{t+1} \right), t \mod 2 \right) \text{ for } t \in \omega.$$

$$(R(\omega + i), I(\omega + i)) = \left((\frac{\pi^2}{6}, 0), i \right) \text{ for } i \in \{0, 1, 2\}.$$

The program can be presented with pseudocode as Algorithm 2.

Algorithm 2: Computation of $\frac{\pi^2}{6}$

```
   Input: No input.
   Output: $\frac{\pi^2}{6}$.
1  Set $R = (0, 1)$;
2  $R_0 = R_0 + R_1^2$;
3  $R_1 = \frac{R_1}{R_1+1}$;
4  while $R_1 > 0$ do
5      $R_0 = R_0 + R_1^2$;
6      $R_1 = \frac{R_1}{R_1+1}$;
7  Return $R_0$;
```

Remark 3.12. Example 3.11 shows how a while loop is made. We execute a list of ordered EDIT commands and use an IF instruction to repeat them if the condition is not satisfied.

3.3 Some results on the ITBSS machine

Here we present some of the main results presented in [15].

Definition 3.13. Let $\alpha < \theta$ be a limit ordinal and $(C_t)_{t<\theta}$ an ITBSS computation by some program P. We say that a command in the computation is carried cofinally often below α if for any $\beta < \alpha$ there exists γ such that $\beta \leq \gamma < \alpha$ and the command has been executed at time γ.

Lemma 3.14. *[15] Let $(C_t)_{t<\theta}$ be an ITBM computation by some program P and let $\alpha < \theta$ be a limit ordinal. Then the first command in the computation that alters the register contents after time α has not been carried out cofinally often below α.*

Proof. Suppose otherwise. Let $c = EDIT(f)$ a command which is called cofinally often below α. Suppose that c has been called at some time $\beta > \alpha$ and for all $\alpha < \gamma < \beta$ the register content has not been changed. Since f is locally continuous (it is a rational function) we have:

$$\begin{aligned} R(\beta+1) &= f(R(\beta)) \\ &= f(R(\alpha)) \\ &= f(\lim_{t\to\alpha} R(t)) \\ &= \lim_{t\to\alpha, I(t)=i} \phi(R(t)) \\ &= \lim_{t\to\alpha, I(t)=i} R(t+1) \\ &= R(\alpha). \end{aligned}$$

A contradiction. □

Theorem 3.15. *[15] Let P be a program with k EDIT commands. Then, for any computation $(C_t)_{t<\theta}$ according to P, the register stabilize before ω^{k+1}, i.e., the values of the registers do not change any more.*

Proof. The proof is done by induction on k. If $k = 0$, all commands are IF or HALT instructions. That implies that the values of the registers R do not change. As the program is executed, the IF commands can do two things: halt the program or execute a new IF command. If the machine halts, it will be after a finite number of steps. If it does not halt after $b+1$ steps, where b is the number of *IF* commands, at least one *IF* instruction has been visited twice which implies that the program has fallen in a loop. Assume true for k. Let P be a program with $k+1$ *EDIT* commands. Suppose that the register does not stabilize before ω^{k+2}. Let c be the first EDIT command that changes the register after ω^{k+2}, As we have seen in Lemma 3.14, c does not change the registers cofinally often below ω^{k+2}. Let α be the supremum of the times below ω^{k+2} that c was executed non trivially (the values of the registers changed). $(C_t)_{\alpha<t<\theta}$ is the computation of P changing c for a trivial $IF(0, i, i)$, where i is the index of c plus one, with input $R(\alpha)$. By induction, the registers of this computation stabilized in ω^{k+1} steps. Since $\alpha + \omega^{k+1} < \omega^{k+2}$, the original computation stabilized before ω^{k+2}. □

Since the argument used before is independent of the input, we have the following result.

Theorem 3.16. *[15] Every ITBM computation halts before ω^ω-many steps or the execution diverges.*

3.4 Observations and basic algorithms

In this section we show several examples of simple algorithms that will be used in the following chapters.

Remark 3.17. When the value of the registers is set as $R = (R_0, R_1, ...)$ the dots indicate that a finite number of registers are added to execute secondary computations.

Computing a limit: From the definition of the computation in Definition 3.9 it is enough to update a register R at each step t to values that tend to a limit value to obtain the desired result. This process can be seen in the following example.

Example 3.18. How to compute π using the Leibniz formula. Since $\sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = \frac{\pi}{4}$ we can explicitly compute the sum. See Algorithm 3 for the pseudocode.

Algorithm 3: Value of π using the Leibniz formula

```
   Input: No input.
   Output: $\pi$.
1  Set $R = (1, 1, -1)$;
2  while $R_1 > 0$ do
3      $R_0 = R_0 + \frac{R_2}{R_1+2}$ (store the partial sum);
4      $R_2 = -\frac{R_2}{R_1}$ ( store $(-1)^k$);
5      $R_1 = \frac{R_1}{1+R_1}$ (store $\frac{1}{k}$);
6      $R_2 = R_2 R_1$ (store $\frac{(-1)^k}{k}$);
7  $R_0 = 4R_0$;
8  Return $R_0$;
```

The program is given by the list of commands

0) $IF(R_1, 3, 1)$

1) $EDIT\left(R = \left(R_0 + \frac{R_2}{R_1+2}, \frac{R_1}{1+R_1}, \frac{R_2}{R_1}\right)\right)$

2) $EDIT\,(R = (R_0, R_1, R_2R_1))$

3) $IF(R_1, 4, 0)$

4) $EDIT\,(R_0 = 4R_0)$

5) $HALT$

At each step the different registers store the necessary information to carry the computation, in this case

$$R(t) = \left(\sum_{k=0}^{t} \frac{(-1)^k}{2k+1}, \frac{1}{t+1}, \frac{(-1)^{t+1}}{t+1}\right).$$

Once the computation halts the value of the first register will be the complete computation of the Leibniz formula.

Remark 3.19. As we slowly increase the complexity of the algorithms we see that the presentation of a program as a list of commands increases. From now on, only the pseudocode representation will be used to keep the programs understandable.

Example 3.18 shows the limitation of the model that needs convergence in each register. In general, this limitation can be circumvented forcing a convergence to a constant or 0 using a denominator that tends to infinity.

3.5 How the machine stores data

This section presents how the data can be stored inside the machine. Our objective is storing a countable list of reals inside one register. This codification must allow us to recover each value. We will use the Cantor pairing function to store families $(r_i)_{i\in\mathbb{N}}$ of real numbers inside one register of the machine. This will be done using subsets of digits of the decimals of a real number to store infinitely many codes, each one coding a real number.

We present the Cantor pairing function. The results are taken from [2].

Definition 3.20. The Cantor pairing function $\pi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is defined as

$$\pi(x, y) = \frac{1}{2}(x + y)(x + y + 1) + y.$$

Proposition 3.21. *The Cantor pairing function is a bijection.*

Proof. To see injectivity we first prove that $x + y < x' + y'$ implies $\pi(x, y) < \pi(x', y')$. Given $k = m + n$ with k fixed, the maximum value of the function $\pi(m, n)$ is $\pi(0, k) = \frac{1}{2}(k)(k + 1) + k$ and the minimum value $\pi(k, 0) = \frac{1}{2}(k)(k + 1)$. If $k' \geq k + 1$, then $\pi(k', 0) \geq \frac{1}{2}(k + 1)(k + 1) + k + 1 > \pi(0, k)$.

Now assume that $\pi(m, n) = \pi(m', n')$ then $n + m = n' + m'$ by above, and hence

$$\pi(m, n) - \pi(m', n') = n - n' = 0,$$

which implies $n = n'$ and $m = m'$.

To see surjectivity let $z \in \mathbb{N}$ and $t_k = \frac{1}{2}(k)(k + 1)$ be the largest value not bigger than z with $k \in \mathbb{N}$. Let $m = z - t_k$ and $n = k - m$. Then $\pi(n, m) = z$. $\square$

Proposition 3.22. *The inverse of the Cantor pairing function is given by*

$$\pi^{-1}(z) = \left(z - \frac{w^2 + w}{2}, -z + \frac{w^2 + 3w}{2}\right),$$

where $w = \left\lfloor \frac{\sqrt{8z+1}-1}{2} \right\rfloor$.

The proof is omitted since it can be done by case analysis.

By definition our machine model takes finite tuples of real numbers as inputs, but we are actually interested in countably infinite sets of real numbers as inputs. To simulate such inputs we code a family $(r_i)_{i\in\mathbb{N}}$ of real numbers by a single real r in such a way that an ITBSS machine can compute the map $(r, i) \mapsto r_i$. There are many ways to achieve this and the particular choice is a matter of no consequence for the rest of this thesis.

For completeness we now detail such a coding. It is based on a coding of reals r by "digit sequences" $d \in \{0, ..., 9\}^{\mathbb{N}}$. Formally, such a sequence is a function $d : \mathbb{N} \mapsto \{0, ..., 9\}$, also written as a family $(d_i)_{i\in\mathbb{N}}$, or, more informally, spelled out as $d_0 d_1 d_2 ...$.

Definition 3.23. We say that $d \in \{0, \ldots, 9\}^{\mathbb{N}}$ is a digit code if and only if there is a positive natural number n such that $d_n = 2$ and $d_m \in \{0, 1\}$ for all $m < n$. The real r coded by d is

$$r = (-1)^{d_0} \sum_{j=1}^{k} 10^{k-j+1} d_{n+j} + \sum_{i=n+k+1}^{\infty} 10^{n+k-i} d_i$$

where $k := \sum_{i=1}^{n-1} 2^{n-i-1} d_i$.

In other words, a code d is a bit followed by a non-empty bit-string s followed by 2, followed by a sequence of digits.The first bit determines the sign, the sequence of digits determines the real with "." at place k, where $k \in \mathbb{N}$ is the natural number with binary expansion s (allowing leading 0s).

It is clear that that every digit code d codes exactly one real. Furthermore, it is clear that every real has a digit code, in fact, for every real r there are infinitely many digit codes $d \in \{0, \ldots, 9\}^{\mathbb{N}}$ such that d codes r. For example, the digit sequences

01210000...
0029999...
0000000299999...
01209999...

all code the real 1. Conversely, every $d \in \{0, ..., 9\}^{\mathbb{N}}$ has a "real code"

$$< d >:= \sum_{i=0}^{\infty} 10^{-(i+1)} d_i \in (0, 1).$$

Proposition 3.24. *There is an ITBSS computation of a program P with input $r \in \mathbb{R}$ that returns as output $< d >$ for some digit code d of r. Further there is a ITBSS computation of a program P with input $< d >$ for some digit code d of some real r with output r.*

Remark 3.25. In Algorithm 4, when the binary expansion of R_3 is done, the output is a natural number which coincides digit to digit with the binary representation R_3 as a natural number in base 2.

Proof. The proof of Proposition 3.24 is done showing an algorithm that can do each process. How to compute $< d >$ given r, see Algorithm 4.

Following the steps of the algorithm we see:

- Steps 1,2,3,4: It is checked if R_0 is negative or not. R_0 is set to its absolute value and R_4 is fixed to 1 in the case R_0 was negative.
- Steps 5,6,7: We have $R_0 = r_1 r_2 ... r_k . r_{k+1} ...$ a non-negative real . The register R_1 is updated to $(R_0 \cdot 10^{-R_3} + 2)/10$ where $R_3 = k$. After Step 7 register R_3 equals $0.2 r_1 r_2 ...$

Algorithm 4: Code of a real

```
Input: $x \in \mathbb{R}$.
Output: $< d >$ that codes $x$.
Set $R = (x, 0, 0, 0, 0, ...)$;
if $R_0 < 0$ then
    $R_4 = 1$;
    $R_0 = -R_0$
$R_2 = \lfloor R_0 \rfloor$ (Algorithm 24);
$R_3 =$ the number of integer-digits of $R_2$ (Algorithm 25);
$R_1 = (R_0 \cdot 10^{-R_3} + 2)/10$;
$R_3 =$ binary expansion of $R_3$ (Algorithm 26);
$R_1 = R_1 + R_3$;
$R_3 =$ the number of integer-digits of $R_3$ (Algorithm 25);
$R_1 = R_1 \cdot 10^{-R_3-1}$ (Algorithm 23);
$R_1 = R_1 + R_4$;
Return $R_1$;
```

Algorithm 5: Real of a code

```
Input: $< d > \in (0, 1)$.
Output: Real number $r$ coded by $< d >$.
Set $R = (< d >, 0, ...)$;
while $\lfloor R_0 \rfloor < 2$ (Algorithm 24) do
    $R_1 = 2 \cdot R_1$;
    if $\lfloor R_0 \rfloor == 1$ then
        $R_1 = R_1 + 1$;
        $R_0 = R_0 - 1$;
    $R_0 = 10 \cdot R_0$;
$R_0 = R_0 - 2$;
$R_0 = 10^{R_1} R_0$ (Algorithm 23);
Return $R_0$;
```

- Step 8: We store in R_3 the binary expansion of k.
- Step 9: We add to R_1 the binary expansion $b_1 b_2 b_3 ... b_n$ of k. After this step $R_1 = b_1 b_2 ... b_n.2 r_1 r_2 ...$.
- Steps 10,11,12: We update R_1 to $s.0 b_1 b_2 ... b_n 2 r_1 r_2 ...$ where $s = 1$ if x was positive and 0 otherwise.

Now R_1 is a valid code.

To compute r given $< d >$ see Algorithm 5. Following the steps of the algorithm we see:

- Steps 2,3,4,5,6,7: The first decimal digits $d_0 d_1 d_2 ...$ of $< d >$ are the binary expansion of the number of integer digits of the real encoded. At the i-th loop the digit d_i of $< d >$ is checked, if it is 0, R_1 is multiplied by two and we advance to the next digit. If $d_i = 1$ then we multiply R_1 by 2 and then, we add 1 to it. As this process advances the final value of R_1 will be the number of integer values of the real encoded. In the particular case of $d_0 d_1 ... = 110$, the first loop will set $R_1 = 1$. At the second loop it will be set $R_1 = 1 \cdot 2 + 1 = 3$. At the last loop $R_1 = 3 \cdot 2 = 6$. In general with this loops we are doing the nested products

$$R_1 = 2(2(2(d_0) + d_1) + d_2)...$$

  Which clearly give us the desired solution. The loop will stop once the digit 2 is found.

- Step 8,9: We know that the digits that appear after the first 2 are the digits of the real r. With step 8 we remove from R_0 the digit 2. With Step 9 we adjust the magnitude of R_0 to obtain the correct value of r. $\square$

Definition 3.26. A real r codes $(r_i)_{i \in \mathbb{N}}$ if $r =< d >$ for some $d \in \{0, \dots, 9\}^{\mathbb{N}}$ such that for all $i \in \mathbb{N}$, d^i is a digit code of r_i where $d^i : \mathbb{N} \mapsto \{0, \dots, 9\}^{\mathbb{N}}$ is given by

$$d^i_j = d_{\pi(i,j)},$$

for all $j \in \mathbb{N}$.

Clearly, every r codes at most one family $(r_i)_{i \in \mathbb{N}}$.

Proposition 3.27. *There is an ITBSS computation of a program P with input a real r that codes $(r_i)_{i \in \mathbb{N}}$ and an index $i \in \mathbb{N}$ computes r_i. Further, there is an ITBSS computation of a program P with input a real r, that decides whether there exists a family $(r_i)_{i \in \mathbb{N}}$ such that r codes $(r_i)_{i \in \mathbb{N}}$.*

Proof. See Algorithm 6

At the first loop $R_3 = (\pi(i, 0))$. Let $< d >= 0.d_0 d_1 d_2 ...$ At Step 4 the value R_5 is updated to $10^{R_3} R_0 - \lfloor 10^{R_3} R_0 \rfloor$ which is exactly $0.d_{R_3} d_{R_3+1} ...$. When updating R_4 we are adding d_{R_3} to it at the decimal position $\frac{1}{R_2} - 1$, in the first loop the position is 0. After the first loop, the register R_4 equals $0.d_{R_3} 000...$. As this process advances we are placing the next decimals on R_4. This process makes $R_4 = 0.d_{\pi(i,0)} d_{\pi(i,1)} d_{\pi(i,2)} ...$. In this process, the register R_2 is used as an auxiliar number to keep track of the decimal position of R_2 that will be updated next.

For an algorithm to check if r codes a family it is enough to compute each $d^i(0), d^i(1), ...$ for each $i \in \mathbb{N}$ and check that the entries are 1 or 0 until a 2 is found. $\square$

```
Algorithm 6: Extract one code
    Input: A code $< d > \in (0,1)$ that codes $(r_i)_{i\in\mathbb{N}}$ and $i \in \mathbb{N}$.
    Output: $d^i$ digit code of $r_i$.
$R = (d, i, 1, 0, 0, 0, ...)$;
while $R_2 > 0$ do
    $R_3 = \pi(R_1, \frac{1}{R_2} - 1)$ (position of the digit);
    $R_5 = 10^{R_3} R_0 - \lfloor 10^{R_3} R_0 \rfloor$ (remove $d_0 ... d_{R_3 - 1}$; Algorithms 23, 24);
    $R_4 = R_4 + 10^{-\frac{1}{R_2}} \lfloor 10 \cdot R_5 \rfloor$(update $d^i$; Algorithms 23, 24);
    $R_2 = \frac{R_2}{R_2+1}$ (next digit);
Return $R_4$;
```

```
Algorithm 7: Extract one digit
    Input: A positive real number $r$ and a natural number $n$.
    Output: Natural number of the decimal digit at position $n$ (starting at 0).
Set $R = (0, r, n, ...)$;
$R_0 = \lfloor 10^{R_2+1} \cdot R_1 \rfloor - 10 \cdot \lfloor 10^{R_2} R_1 \rfloor$ (Algorithms 23, 24);
Return $R_0$;
```

```
Algorithm 8: Alter a code
    Input: A real number $r$ that codes a family $(r_i)_{i\in\mathbb{N}}$, a real number $s$ and a natural number $j$.
    Output: Real number $r'$ such that $r'$ codes $(s_i)_{i\in\mathbb{N}}$ a family defined as $s_i = r_i$ when $i \neq j$ and
            $s_j = s$.
Set $R = (r, s, j, 1, 0, 0, ...)$;
$R_5 =$ code of $R_1$ (Algorithm 4);
while $R_3 > 0$ do
    $R_4 =$ decimal digit at position $\pi(j, \frac{1}{R_3} - 1)$ of $R_0$ ( Algorithm 7);
    $R_0 = R_0 - 10^{-\pi(j, \frac{1}{R_3} - 1) - 1} \cdot R_4$ (Algorithm 23);
    $R_0 = R_0 + 10^{-\pi(j, \frac{1}{R_3} - 1) - 1} \lfloor 10 \cdot R_5 \rfloor$ (Algorithms 23, 24);
    $R_5 = 10 \cdot R_5 - \lfloor 10 \cdot R_5 \rfloor$ (Algorithm 24);
    $R_3 = \frac{R_3}{R_3+1}$;
Return $R_0$;
```

From the results above we have an effective way to store and recover a family $(r_i)_{i\in\mathbb{N}}$. Many other approaches are possible, an alternative codification can be found in [7].

So far, we have seen that the machine can store countable sets of real numbers inside a register. The next natural step is creating a method to update the value of a real r that codes a sequence $(r_i)_{i\in\mathbb{N}}$ to change the value of one r_i without modifying the rest. To do so we present some technical algorithms.

Proposition 3.28. *Let r be a real number that codes a family $(r_i)_{i\in\mathbb{N}}$, s be a real number and j be a natural number. Then, there exists an ITBSS program with input s, r, j that ouputs a real number r' such that r' codes $(s_i)_{i\in\mathbb{N}}$ a family defined as $s_i = r_i$ when $i \neq j$ and $s_j = s$.*

Proof. See Algorithm 8. At the nth iteration of the loop, let $k_n = \pi(j, n) + 1$:

- Step 4: We store in R_4 the value of the decimal digit at position $\pi(j, n)$ of R_0.
- Step 5: We subtract from R_0 the value $10^{-k_n} R_4$ making the value of the decimal digit at position $\pi(j, n)$ equal to 0.
- Step 6: We add to R_0 the value $10^{-k_n}\lfloor 10R_5 \rfloor$ making the $\pi(j, n)$ decimal digit of R_0 equal to $\lfloor 10R_5 \rfloor$ which is exactly the nth decimal digit of the code of R_1.
- Step 7: we update R_5 to be $0.d_{n+1}d_{n+2}\ldots$ where $0.d_0d_1d_2\ldots$ is the code of R_1.
- Step 8: We update the auxiliar register R_3 to the next value.

When going through the while loop for the n-th time we are placing the digit d_n in the decimal position $\pi(j, n)$ of R_0. We have that the sequence of digits of R_0 at positions $\pi(j, n)$ for all $n \in \mathbb{N}$ code s.

Since we have only changed the values of the digits at positions $\pi(j, n)$ for all $n \in \mathbb{N}$, the real r codes the family $(s_i)_{i\in\mathbb{N}}$. □

Note that, since we can store countable sets of real numbers inside one real, we can also iterate this process. We can store, countably many families of real numbers inside a real.

3.6 Inputs and technical details

This section is devoted to the technical decisions taken for a formal approach to the algorithmic problems that will follow. It shows an organized procedure to control the information used in the algorithms.

```
Algorithm 9: Modify a digit
    Input: A non-negative real number $r$, natural numbers $n$ and $p$.
    Output: A real number $r'$ with same digits as $r$ except the one at decimal position $p$ which is
            changed to $n$ .
1 Set $R = (r, 0, n, p)$;
2 $R_1$ = digit at position $n$ of $R_0$ (Algorithm 7);
3 $R_0 = R_0 + 10^{-p-1} \cdot (n - R_1)$ (Algorithm 23);
4 Return $R_0$;
```

From Algorithms 6 and 8 we have a method to extract and edit values of a family $(r_i)_{i\in\mathbb{N}}$ coded by a real r. We also have Algorithm 9 that allow us to change the values of each decimal digit of a non-negative real number.

Definition 3.29. The following statements are conventions imposed to the real numbers that store sequences of real numbers.

a) For each real r we call r_i the real number coded by the digit code formed by the ordered digits of r on positions $\pi(i, j)$ with $j \in \mathbb{N}$.

b) If for some real code r and $i \in \mathbb{N}$ the digit with decimal position $\pi(i, 0)$ is 3 then it is understood that the real r_i is not defined in the sequence.

c) (Remove holes) If an algorithm has a sequence as input, that sequence does not have undefined values, we call such undefined values holes. If that were the case, the hole is removed creating a new real r that codes the same sequence without undefined values. This process is done with Algorithms 6 and 8.

d) (Empty register/list) An empty register is a register with value a real number with all digits zero except the decimals digits at positions $\pi(i, 0)$ for i natural, which have digit value 3.

e) Whenever a real is added to a list coded by a real number, it is added using Algorithm 8 on the first i such that the digit at position $\pi(i, 0)$ is 3.

f) Whenever a real number r_i is removed from a sequence, the values of the digits at positions $\pi(i, j)$ with $j > 0 \in \mathbb{N}$ are set to 0, and the one at position $\pi(i, 0)$ is set to 3. After that, the real number that codes it is subtituted by the one constructed as commented in c), i.e. a new code without holes.

Definition 3.30. The following statements are conventions imposed to the inputs and algorithms presented in the following chapters.

1) A countable set $X \subseteq \mathbb{R}^n$ is stored in a real number r as a sequence $(r_i)_{i\in\mathbb{N}}$ where each r_i encodes the n coordinates of a value of X. The sequence does not have repetitions of elements.

2) A list of rectangles in $\mathbb{R}^2$ is stored in a real number r as a sequence $(r_i)_{i\in\mathbb{N}}$ where each r_i codes the defining points of a rectangle as defined in 1).

3) Whenever an algorithm has a set X encoded in a real r inside a register, we will write the register as X. If a register has a set X, the notation $X(i)$ correspond to the element encoded in r_i. For example, if X is a set of points in $\mathbb{R}^2$, then $X(0) = (x, y)$ is the point encoded by r_0. If we need to refer to a particular component of the point, we use $X(0)(0)$. In this case $X(0)(0) = x$ and $X(0)(1) = y$.

4) A segment L is stored in a real r having r_0 and r_1 encoding the coordinates of the two extremal points, respectively. The r_i with $i > 1$ are not defined in the sequence as in Definition 3.29 b).

5) A well-order on the naturals is encoded as a real r if the digit at decimal position $\pi(n, m)$ equals 0 if and only if $n < m$.

6) A family of well-ordered sets $(<_i)_{i\in\mathbb{N}}$ is encoded in a real r if each r_i encodes $<_i$.

Definition 3.31. The following statements are conventions imposed to the operations of the algorithms.

i) The boolean logic of Yes/No is implemented with the use of 0 and 1, respectively.

ii) For two sets $X, Y \subseteq \mathbb{R}^n$, when we write in pseudocode the sentence $R_k = X(i) + Y(j)$ for some $k, i, j \in \mathbb{N}$, then we are expressing that the register R_k will have the real number r that encodes the coordinates of the point in $\mathbb{R}^n$ that is obtained from the sum of $X(i)$ and $Y(j)$ as elements in $\mathbb{R}^n$.

iii) Given two points $x = (x_1, ..., x_i) \in \mathbb{R}^i$ and $y = (y_1, ..., y_j) \in \mathbb{R}^j$. The pseudocode $R_k = (x, y)$ means that the register R_k will have the real r that encodes the point $z = (x_1, ..., x_i, y_1, ..., y_j) \in \mathbb{R}^{i+j}$.

Example 3.32. How to compute the supremum of a countable bounded set $X \subseteq \mathbb{R}$. See Algorithm 10.

Algorithm 10: Supremum of a countable bounded set

```
Input: A countable bounded subset $X$ of $\mathbb{R}$.
Output: supremum of $X$.
1 Set $R = (X, 1, 0...)$;
2 $R_2 = X(0)$ (Definition 3.30);
3 while $R_1 > 0$ do
4     if $R_2 < X(\frac{1}{R_1})$ then
5         $R_2 = X(\frac{1}{R_1})$;
6     $R_1 = \frac{R_1}{R_1+1}$
7 Return $R_2$;
```

The Algorithm 10 stores in R_2 the biggest value of the first $\frac{1}{R_1}$ elements of X. As we do this process two things can happen: the value of R_2 is fixed after a finite number of steps or ω steps are done and R_2 takes a limit value.

3.7 Scroll through a list

In future algorithms it is necessary to go through the elements of $\mathbb{N}^n$ and $\mathbb{Q}^n$. Here we present a method to do it using an exhaustive function from $\mathbb{N}$ to $\mathbb{N}^k$ for any $k \geq 2$.

Proposition 3.33. *There exists an ITBSS program that computes $\pi^{-1}(n)$, the inverse of the Cantor pairing function, for any $n \in \mathbb{N}$.*

Proof. Algorithm 11 implements the inverse function used in Proposition 3.22. □

Algorithm 11: Inverse of the Cantor pairing function used in Proposition 3.22

```
    Input: A natural number $n$.
    Output: $\pi^{-1}(z)$.
1   Set $R = (n, 0, 0, 0)$;
2   $R_3 = \left\lfloor \frac{\sqrt{8z+1}-1}{2} \right\rfloor$ (Algorithms 24 and 27);
3   $R_1 = R_0 - \frac{R_3^2+R_3}{2}$;
4   $R_2 = -R_0 + \frac{R_3^2+3R_3}{2}$;
5   $R_0 = (R_1, R_2)$ (Definition 3.31);
6   Return $R_0$;
```

Since $\pi^{-1}(z)$ is a bijection between $\mathbb{N}$ and $\mathbb{N}^2$ we can use it recursively to make a bijection from $\mathbb{N}$ to $\mathbb{N}^k$ for any k natural greater than 1. We present that bijection in the following definition.

Definition 3.34. We define the functions $B_k(z) : \mathbb{N} \to \mathbb{N}^k$ for naturals $k \geq 2$ as.

$$B_k(z) = \begin{cases} \pi^{-1}(z) & \text{if } k = 2, \\ B_k(z) = (x_1, \dots, x_{k-2}, \pi^{-1}(x_{k-1})) & \text{if } k \geq 3, \end{cases}$$

where $(x_1, \dots, x_{k-1}) = B_{k-1}(z)$.

Proposition 3.35. *The functions B_k are bijective for all naturals $k \geq 2$.*

Proof. The proof is inductive. The case $k = 2$ is satisfied for being the preimage of $\pi(z)$, a bijective function. Assume B_k bijective for $k < n$. Clearly, for all $(m_1, \dots, m_{n-1}) \in \mathbb{N}^{n-1}$ there exists a unique $(t_1, \dots, t_n) \in \mathbb{N}^n$ such that $t_i = m_i$ for $i \in \{1, \dots, n-2\}$ and $\pi^{-1}(m_{n-1}) = (t_{n-1}, t_n)$. Since B_n is the composition of B_{n-1} and the unique correspondence stated before, it is bijective. □

These functions can be easily computed using $k - 1$ times Algorithm 11. Additionally, we can make an exhaustive partial function from $\mathbb{N}$ to $\mathbb{Q}^k$.

Definition 3.36. We define the partial functions $Q_k : \mathbb{N} \to \mathbb{Q}^k$ for all naturals $k \geq 2$ as

$$Q_k(z) = \left(\frac{x_1}{x_2}, \frac{x_3}{x_4}, \dots, \frac{x_{2k-1}}{x_{2k}} \right) \text{ when } x_{2i} \neq 0 \text{ for all } i \in \{1, \dots, k\},$$

where $(x_1, \dots, x_{2k}) = B_{2k}(z)$.

Proposition 3.37. *The partial functions Q_k are exhaustive.*

Proof. Since B_{2k} is bijective, for any $\left(\frac{x_1}{x_2}, \frac{x_3}{x_4}, \dots, \frac{x_{2k-1}}{x_{2k}} \right) \in \mathbb{Q}^k$ there exits some $z \in \mathbb{N}$ such that $B_{2k}(z) = (x_1, \dots, x_{2k})$. □

These partial functions can be easily implemented and used to visit all points in $\mathbb{Q}^n$.

4. Accumulation points

Now that the framework has been established, we apply it to geometric problems with infinite countable input. We start with accumulation points. The traditional models of computation can not work with accumulation points since for their appearance we need an infinite set of points.

Definition 4.1. A point x is an accumulation point of the set $X \subseteq \mathbb{R}^n$ if

$$\forall \epsilon > 0 \ \exists y \in X - \{x\} \text{ such that } d(x, y) < \epsilon,$$

where $d(x, y)$ is the Euclidean distance between x and y.

Proposition 4.2. *A point x is an accumulation point of a set $X \subset \mathbb{R}^n$ if and only if*

$$\inf\{d(x, y) \mid y \in X - \{x\}\} = 0.$$

Proof.

$\Rightarrow$) Assume that $\inf\{d(x, y) \mid y \in X - \{x\}\} = k > 0$. Using the definition of accumulation point, with $0 < \epsilon < k$, we know that exists $y \in X$ such that $d(x, y) < k$, a contradiction.

$\Leftarrow$) Assume that x is not an accumulation point. That implies that there exists $\epsilon > 0$ such that no point y in X satisfies $d(x, y) < \epsilon$. Then it is clear that $\inf\{d(x, y) \mid y \in X - \{x\}\} \geq \epsilon$. □

Next, we introduce some technical results.

Theorem 4.3. *(Bolzano-Weierstrass Theorem in $\mathbb{R}^n$) Let $\{a_i\}_{i \in \mathbb{N}}$ be a bounded sequence of points in the set $X = [a_1, b_1] \times \cdots \times [a_n, b_n]$ with $a_1, \ldots, a_n, b_1, \ldots, b_n \in \mathbb{R}$. Then, there exists a subsequence that converges to a point in X.*

The proof is not complicated but technical and long. See [8].

Proposition 4.4. *If an infinite set $X \subset \mathbb{R}^n$ is bounded, it has at least one accumulation point.*

Proof. Let $\{a_i\}_{i \in \mathbb{N}}$ be a sequence of points of X without repetitions. From Theorem 4.3 the sequence has a subsequence $\{b_i\}_{i \in \mathbb{N}}$ that converges. From the definition of limit, $\lim_{n \to \infty} b_n$ is an accumulation point of X. □

From now on, we present several definitions and concepts in $\mathbb{R}^2$. Most of the concepts can be easily adapted to $\mathbb{R}^n$ but, since the final results will concern $\mathbb{R}^2$, we only define them for the planar case.

Definition 4.5. The mass center of a set $\{(x_i, y_i)\}_{i \in \mathbb{N}}$ of $\mathbb{R}^2$ is defined as

$$\lim_{n \to \infty} \left(\sum_{i=0}^{n-1} \frac{x_i}{n}, \sum_{i=0}^{n-1} \frac{y_i}{n} \right).$$

Lemma 4.6. *The mass center of a bounded set of $\mathbb{R}^2$ always exists.*

Proof. Since the set is bounded, there exists $c = \sup\{x_i\} < \infty$ and $d = \inf\{x_i\} > -\infty$. From these bounds we obtain

$$d = \lim_{n\to\infty}\sum_{i=0}^{n-1}\frac{d}{n} \le \lim_{n\to\infty}\sum_{i=0}^{n-1}\frac{x_i}{n} \le \lim_{n\to\infty}\sum_{i=0}^{n-1}\frac{c}{n} = c.$$

We can assume that all coordinates are positive, otherwise we can make a translation of the points to the first quadrant. Given that all the entries are positive, the series converge. The same argument holds for the second coordinate of the points. □

Proposition 4.7. *The mass center of a bounded and countable set of points with exactly one accumulation point is the accumulation point.*

Proof. Let (x, y) be the accumulation point and (a, b) the mass center of the given set $\{(x_i, y_i)\}_{i\in\mathbb{N}}$. The following argument applies to each coordinate of the points. Suppose that $a < x$. Then

$$\lim_{n\to\infty}\sum_{i=0}^{n-1}\frac{x_i}{n} = a.$$

Let $c = \frac{a+x}{2}$. Since the limit exists, we can express it as the sum

$$\lim_{n\to\infty}\sum_{i=0}^{n-1}\frac{x_i}{n} = \lim_{n\to\infty}\left(\sum_{x_j\le c}\frac{x_j}{n} + \sum_{x_i>c,i<n}\frac{x_i}{n}\right),$$

where the first sum is the sum of points with x-coordinate smaller than or equal to c. This set must be finite because otherwise the ball $B(x_j, c)$ would have infinitely many points, and Proposition 4.4 would imply the existence of another accumulation point. The second sum is the sum of the rest of the points, an infinite set. Since both series converge,

$$\begin{aligned}\lim_{n\to\infty}\sum_{i=0}^{n-1}\frac{x_i}{n} =& \lim_{n\to\infty}\left(\sum_{x_j\le c}\frac{x_j}{n}\right) + \lim_{n\to\infty}\left(\sum_{x_i>c,i<n}\frac{x_i}{n}\right)\\ =&0 + \lim_{n\to\infty}\left(\sum_{x_i>c,i<n}\frac{x_i}{n}\right)\\ \ge& \lim_{n\to\infty}\left(\sum_{x_i>c,i<n}\frac{c}{n}\right)\\ =&c,\end{aligned}$$

which is a contradiction. An analogue argument works for $a > x$. □

Algorithm 12 shows how to compute the mass center of a countable set X in $\mathbb{R}^2$.

Definition 4.8. We define "removing" an accumulation point x from a set X as finding a subset Y of X such that

- x is not an accumulation point of Y.

Algorithm 12: Mass center of a set in $\mathbb{R}^2$

```
    Input: A countable set $X \subset \mathbb{R}^2$.
    Output: Mass center of $X$.
1   Set $R = (X, X(0), 1, ...)$;
2   while $R_2 > 0$ do
3       $R_1 = \frac{R_1}{R_2+1} + \frac{X(\frac{1}{R_2})R_2}{R_2+1}$ (sum from 0 to n-1; It is a sum of points in $\mathbb{R}^2$, see Definition 3.31);
4       $R_2 = \frac{R_2}{R_2+1}$;
5   Return $R_1$;
```

- If $z \neq x$ is an accumulation point of X, it is also an accumulation point of Y.

We say that Y removes x from X.

Definition 4.9. The open ball of center x and radius r, $B(x, r)$ is defined as the set

$$B(x, r) = \{z \mid d(x, z) < r\}.$$

Given a set X with a finite number of accumulation points, Algorithm 13 shows how to remove an accumulation point x. See Figure 3.

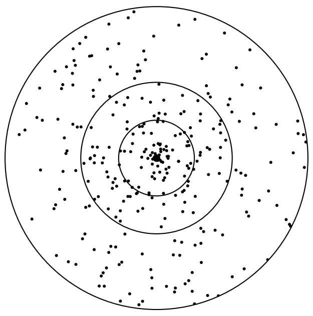

Figure 3: The balls $B(x, \frac{1}{2^n})$, $B(x, \frac{1}{2^{n+1}})$ separate the neighbours of the accumulation point x into regions. There exists a natural number n_0 such that for all $n > n_0$, the region $B(c, \frac{1}{2^{n-1}}) \setminus B(x, \frac{1}{2^n})$ contains only a finite number of point of X.

Proposition 4.10. *Algorithm 13 outputs a valid solution.*

Proof. To see that this algorithm works it is enough to see that the output R_5 contains all accumulation points of X except x.

First observe that the while loop of Step 2 checks for each element of X if it lies between the two balls. If the number of points that satisfy the condition is infinite, the value of the register R_2 will converge to 0. If $R_2 = 0$, the register R_4 is updated to a new value. This process will update R_4 until the balls with radius $\frac{1}{2^{1/R_2}}, \frac{1}{2^{1/R_2+1}}$ have finitely many elements in between for all natural $n \geq \frac{1}{R_3}$. This will eventually happen because otherwise, we would have infinitely disjoint bounded regions with infinitely many points inside, which would imply the existence of infinitely many accumulation points. In this process R_1 is used

Algorithm 13: Removing an accumulation point

```
    Input: A countable set $X \subset \mathbb{R}^2$ with finitely many accumulation points, and an accumulation
           point $x$ of $X$.
    Output: A set $Y$ that removes $x$ from $X$.
1  Set $R = (X, 1, 1, 1, 1, E, \ldots)$ (E is an empty list as defined in subsection 3.6);
2  while $R_3 > 0$ do
3      while $R_1 > 0$ do
4          if $X(\frac{1}{R_1}) \in B(x, \frac{1}{2^{1/R_3}}) \setminus B(x, \frac{1}{2^{1/R_3+1}})$ then
5              $R_2 = \frac{R_2}{R_2+1}$
6          $R_1 = \frac{R_1}{R_1+1}$
7      if $R_2 \leq 0$ then
8          $R_4 = \frac{R_3}{R_3+1}$
9      $R_3 = \frac{R_3}{R_3+1}$;
10     $R_2 = 1$;
11     $R_1 = 1$;
12 while $R_1 > 0$ do
13     if $X(\frac{1}{R_1} - 1) \notin B(x, \frac{1}{2^{1/R_4}})$ then
14         $E(\frac{1}{R_2} - 1) = X(\frac{1}{R_1} - 1)$;
15         $R_1 = \frac{R_1}{R_1+1}$;
16         $R_2 = \frac{R_2}{R_2+1}$;
17     else
18         $R_1 = \frac{R_1}{R_1+1}$;
19 Return $R_5$;
```

to go through all points of X to check if they lie between the balls. And R_3 is used to go through all values of $\mathbb{N}$ to study all the possible radius of the balls.

When R_4 stabilizes to a value n, the ball $B = B(x, 1/2^n)$ contains x and all points z in X such that $d(x,z) < 1/2^n$. From here it is clear that for all $z \in X \setminus B$, $d(x,z) \geq \frac{1}{2^n}$, which implies that x is not an accumulation point of $X \setminus B$. Steps 12 to 16 store in R_5 the set $X \setminus B$. The register R_1 is used to go through all the elements of X and R_2 to go through all the elements of E.

To end, all the accumulation points of X different from x are in $X \setminus B$. Suppose that $z \neq x$ is an accumulation point in X but it is not in $X \setminus B$. Then for all $\epsilon > 0$ exists $b \in B$ such that $d(z,b) < \epsilon$, a contradiction with the value of n. □

Algorithm 14 shows a simple criterion, based in Proposition 4.2 to decide if a segment contains an accumulation point of a set X.

Algorithm 14: Deciding if a segment L has an accumulation point of X

```
   Input: A set $X$ and a segment $L$ with at most finite number of points in common.
   Output: Yes/No answer.
1  Set $R = (X, L, 1, 1, 1)$;
2  while $R_3 > 0$ do
3      $R_4 = d(L, X(\frac{1}{R_3} - 1))$ (use the point-segment distance formula of $\mathbb{R}^2$);
4      if  $0 < R_4 < R_5$ then
5          $R_5 = R_4$
6      $R_3 = \frac{R_3}{R_3+1}$;
7  if $R_5 \leq 0$ then
8      Return Yes;
9  else
10     Return No;
```

We present now Algorithm 15. In this case we state the algorithm in words without the detailed operations on registers, though it is possible to explicitly give the ITBSS program.

Remark 4.11. In Algorithm 15 we need that the segment and X do not have more than finitely many points in common, otherwise an accumulation point of X could be the limit of points in L and we would not detect it. From now on, we assume that the sets X studied have at most finitely many points aligned.

Proposition 4.12. *Algorithm 15 is correct.*

Proof. Since we check if L has accumulation points of X, we can assure that the sequence $\{s_i\}$, defined in the algorithm, is computed, and that at least one accumulation point in L exists. By construction the values of the sequence $\{s_i\}$ get closer to L as i tends to infinity. Now, since

$$y_a = \lim_{n\to\infty} \left(\inf_{i \geq n} y(s_i) \right)$$

there exists an infinite subsequence $\{a_i\}$ in $\{s_i\}$ such that $\lim_{i\to\infty} y(a_i) = y_a$. By construction, the x-coordinates tend to the x-coordinate of L. We conclude that $\lim_{i\to\infty} a_i \in L$ and it is an accumulation point of X. Since this process always finds one accumulation point, removing the point found and repeating the process will eventually complete the search. □

Algorithm 15: Finding the accumulation points of X on a segment L

```
Input: A set $X \subset \mathbb{R}^2$ with finitely many accumulation points , and a vertical segment $L$.
Output: Accumulation points of $X$ in $L$.
1 Use Algorithm 14 to check if $X$ has accumulation points on $L$. If the answer is negative return the
   empty set. Otherwise continue.
2 Check if the extremal points of the segment are accumulation points. If an accumulation point is
   found, remove it using Algorithm 13.
3 Store $\{r_i\}$, a sequence containing all points with y-coordinate between the extremal values of the
   segment not in $L$. To do this, check for each element of $X$ if it satisfies this condition on the
   y-coordinate. If it does, add it to the sequence; do not otherwise. From this set take a subset
   $\{s_i\}$ as follows: Pick the initial value $s_1 = r_1$. Then search the first $r_i$ in the sequence such that
   $d(L, r_i) < \frac{1}{2}d(L, s_1)$, this point is $s_2$. Repeat until step $\omega$. Compute

   $$y_a = \lim_{n\to\infty}\left(\inf_{i\geq n} y(s_i)\right)$$

   where $y(s)$ is the y-coordinate of point $s$ (use the method of Algorithm 10). Check if the point
   on the segment with y-coordinate $y_a$ is an accumulation point. If it is, remove it to obtain a
   subset $Y$. Check if $L$ has an accumulation point of $Y$; if it does, repeat Step 3, processing with
   $Y$. Otherwise return all the accumulation points found.
```

Remark 4.13. Some remarks on the proof and the Algorithm 15.

- In this case the algorithm was a exposed in text to facilitate the comprehension.
- In the condition $d(L, s_i) < \frac{1}{2}d(L, s_{i-1})$ the $\frac{1}{2}$ is necessary to assure that the points tend to L. Any $0 < \epsilon < 1$ could be used instead.
- The sequence $\{s_i\}$ does not have to converge to a value. For example the sequence could be an alternation of $\{(\frac{1}{n}, 1)\}_{n\in\mathbb{N}_{>0}}$ and $\{(\frac{1}{n}, 2)\}_{n\in\mathbb{N}_{>0}}$.
- Using the inferior limit does not guarantee us that the accumulation points will be found in ascending order, the order of appearance depends on how the sequence $\{s_i\}$ is created. It could always choose elements from a subsequence that tends to an accumulation point which is not the lowest, nor the highest.

Algorithm 16: Compute the finite number of accumulation points of a bounded set

```
  Input: A bounded set $X \subset \mathbb{R}^2$ with finitely many accumulation points.
  Output: Accumulation points of $X$.
1 Check for each $n \in \mathbb{N}$ if $X$ is contained in the square $[-n, n]^2$ until a natural $m$ is found such that
   $X$ is contained in $[-m, m]^2$ (Check for each point in $X$ if it lies inside the square).
2 Check, using Algorithm 15, if the vertical sides of $[-m, m]^2$ contain accumulation points and use
   the process of Algorithm 13 to remove them.
3 Let $R$ be the list containing only the square $[-m, m]^2$.
4 For a generic rectangle $T = [a, b] \times [c, d]$ in $R$:
     • Remove $T$ from $R$.
     • Check if there are infinitely many points of $X$ inside $T$ (Method of algorithm 13). If there are,
         – Check if the vertical segments from $(a, c)$ to $(a, d)$ and $(b, c)$ to $(b, d)$ have accumulation
           points. If they have, remove them.
         – For the rectangles $T_1 = [a, (a + b)/2] \times [c, d]$ and $T_2 = [(a + b)/2, b] \times [c, d]$, check if there
           are infinitely many points inside each one. If some has, add it to $R$.
     Take a square in $R$ and repeat the process until $R$ is empty.
  Return the accumulation points found.
```

Now, we are finally able to compute the accumulation points of a given bounded set X.

Proposition 4.14. *Algorithm 16 is correct.*

Proof. Simply observe that if an accumulation point lies in the interior of a region, there must be infinitely many points inside it. At each step the vertical segments will be closer to the x-coordinate of the accumulation point until the values converge to a line that contains the accumulation point. □

With a simple trick, Algorithm 17 can be adapted to work for unbounded sets.

Algorithm 17: Compute the finite number of accumulation points of a set

```
  Input: A set $X \subset \mathbb{R}^2$ with finitely many accumulation points.
  Output: Accumulation points of $X$.
1 Apply Algorithm 16 to the points of $X$ inside the square $[-1, 1]^2$.
2 Remove the points of $X$ inside the square $[-1, 1]^2$ (Create a new sequence with the elements of $X$
   outside the square).
3 Apply the complex transformation $f(z) = \frac{1}{z}$ to the points of $X$ outside the square and apply
   Algorithm 16. Once the accumulation points are found, to all the solution points different from
   $(0, 0)$, apply again the complex transformation $\frac{1}{z}$ (change each element for each complex
   transformation).
4 Return the accumulation points found.
```

Remark 4.15. In Algorithm 17, the point (0,0) is removed because it will never be an accumulation point after the transformation $\frac{1}{z}$ is done. Observe that the tranformation $\frac{1}{z}$ sends big module numbers to small module numbers, and if the point $(0,0)$ appears it is because the supremum of the modules tends to infinity.

Remark 4.16. Sadly, Algorithm 16 cannot be extended to a countable amount of accumulation points because the process to remove accumulation points needs the existence of a minimum distance ϵ among them. Nonetheless, if a set X has a countable amount of accumulation points and if there exists a non-zero minimum distance among them, then Algorithm 16 can be adapted to work, by simply dividing the space into squares of the form $[a, a+1] \times [b, b+1]$ with $a, b \in \mathbb{Z}$, and applying Algorithm 16 to each square.

5. Convex Hull

Calculation of convex hull is a very studied field in finite and infinite geometry. Here, we present some results and algorithms for the infinite case using well known facts about them. A good reference is [17].

Definition 5.1. A set X is convex if

$$\forall x, y \in X \subseteq \mathbb{R}^n, \quad \forall t \in [0,1] \quad tx + (1-t)y \in X.$$

Definition 5.2. The convex hull of X, $CH(X)$ is the intersection of all convex sets containing X.

Definition 5.3. We define an hyperplane $H(\vec{c}, d)$, with $\vec{c} \in \mathbb{R}^n$ and $d \in \mathbb{R}$ as

$$H(\vec{c}, d) = \{x \in \mathbb{R}^n \mid x \cdot \vec{c} = d\}.$$

If $\vec{c} \in \mathbb{Q}^n$ and $d \in \mathbb{Q}$ we say it is a rational hyperplane.

Definition 5.4. Given non-empty sets X_1, X_2 of $\mathbb{R}^n$. The hyperplane $H(\vec{c}, d)$ separates X_1 and X_2 if

$$\sup_{x \in X_1} (x \cdot \vec{c}) \leq d \leq \inf_{x \in X_2} (x \cdot \vec{c}).$$

If the inequality is strict, we say that $H(\vec{c}, d)$ strongly separates X_1 and X_2.

Theorem 5.5. *(Strong Separating Hyperplane Theorem) Let K and C be disjoint and non-empty convex subsets of $\mathbb{R}^n$. Suppose K is compact and C is closed. Then there exists an hyperplane $H(\vec{p}, d)$ that strongly separates K and C.*

Proof. This fact is well known. A proof can be found in [16]. $\square$

Here we present a result that motivates the use of closed halfspaces.

Proposition 5.6. *Let C be the family of convex sets that contain X and H be the family of closed halfspaces that contain X, then the following equality holds:*

$$\overline{\bigcap_{c \in C} c} = \overline{CH(X)} = \bigcap_{h \in H} h, \tag{1}$$

where the overline denotes the closure of the set.

Proof. The first equality holds immediately by the definition of convex hull.

$\overline{\bigcap_{c \in C} c} \subset \bigcap_{h \in H} h$: Clearly $H \subset C$ and that implies

$$\bigcap_{c \in C} c \subset \bigcap_{h \in H} h,$$

now, that implies that

$$\overline{\bigcap_{c \in C} c} \subset \overline{\bigcap_{h \in H} h},$$

but, since H is a family of closed sets the intersection is a closed set. We conclude

$$\overline{\bigcap_{c \in C} c} \subset \overline{\bigcap_{h \in H} h} = \bigcap_{h \in H} h.$$

$\bigcap_{h\in H} h \subset \overline{\bigcap_{c\in C} c}$: It is enough to prove that if $x \in CH(X)$, then for all $y \notin CH(X)$ such that $d(y, CH(X)) > 0$, there exists a halfspace $h \in H$ such that $y \notin H$. The results follows from the hyperplane separation Theorem, see for instance [19]. □

5.1 Convex hull computation

Definition 5.7. Let $x_0, x_1, ..., x_n$ be points in $\mathbb{R}^n$. We say that x_0 is in the convex combination of $x_1, ..., x_n$ if there exist non-negative real numbers $a_1, ..., a_n$ such that $a_1x_1 + ... + a_nx_n = x_0$ and $a_1 + ...a_n = 1$.

Theorem 5.8. *(Carathéodory's Fundamental Theorem) Each point in the convex hull of a set S in $\mathbb{R}^n$ is in the convex combination of $n+1$ or fewer points of S.*

For a proof of Theorem 5.8 see [25].

Now we present Algorithm 18 that computes if a point lies inside the convex hull of a set.

Algorithm 18: Decide if $x \in CH(X) \subseteq \mathbb{R}^n$

```
Input: $x$ a point in $\mathbb{R}^n$ and a countable set $X \subset \mathbb{R}^n$.
Output: Yes/No answer.
1 For each simplex of $n+1$ points $x_1, x_2, ..., x_{n+1} \in X$, check if the point $x$ lies inside. If such
    simplex is found, return Yes and return No otherwise (use the function of Definition 3.34 to go
    through all (n+1)-tuples of $X^{n+1}$).
```

Remark 5.9. In Algorithm 18, to check if a point x lies inside a simplex it is enough to do elementary operations which can be implemented in the ITBSS machine, but details are omitted since this process is exactly the same as in the finite model.

Proposition 5.10. *The algorithm 18 is correct.*

Proof. This is an easy corollary of Theorem 5.8. □

Theorem 5.11. *Let S be a convex set with $int(S) \neq \emptyset$. Then $\overline{int(S)} = \overline{S}$.*

Proof. See [16], Corollary 3.4.13. □

We present now Algorithm 19 that computes if a point lies in the closure of the convex hull of a set.

Algorithm 19: Decide if $x \in \overline{CH(X)} \subset \mathbb{R}^n$ with $int CH(X) \neq \emptyset$

```
Input: $x$ a point in $\mathbb{R}^n$ and a countable set $X \subset \mathbb{R}^n$.
Output: Yes/No answer.
1 For each $m \in \mathbb{N}_{>0}$ and for each rational $q \in \mathbb{Q}^n$ check if $q \in B(x, \frac{1}{m}) \cap CH(X)$ (Use the distance
    for the ball, Algorithm 18 for the convex hull and the functions in Definition 3.36 to go through
    all $\mathbb{Q}^n$). If for all $n \in \mathbb{N}_{>0}$ a rational is found such that $q \in B(x, \frac{1}{n}) \cap CH(X)$, then $x \in \overline{CH(X)}$
    and return Yes, return No otherwise.
```

Proposition 5.12. *The algorithm 19 is correct.*

Proof. If $x \notin \overline{CH(X)}$, then there is $\epsilon > 0$ such that $d(x, CH(X)) > \epsilon$ and for all $n > \frac{2}{\epsilon}$ we have $B(x, \frac{1}{n}) \cap CH(X) = \emptyset$.

If $x \in \overline{CH(X)}$, by Theorem 5.11, there exists a sequence $\{a_n\}_{n \in \mathbb{N}}$ of points in $int(CH(X))$ with $\lim_{n \to \infty} a_n = x$. Then, for each $n \in \mathbb{N}_{>0}$ we have $B(x, \frac{1}{n}) \cap int(CH(X)) \neq \emptyset$. Let z_n be in this intersection. Since the finite intersection of open sets is open, z_n is an interior point. From that, we know that there exists ϵ_n such that $B(z_n, \epsilon_n) \subseteq B(x, \frac{1}{n}) \cap int(CH(X))$. From the density of rationals, see Theorem A.4, there is a rational $r_n \in B(z_n, \epsilon_n)$. Since this argument holds for any $n \in \mathbb{N}_{>0}$, at each step at least one rational must be found in $B(x, \frac{1}{n}) \cap CH(X)$. □

Theorem 5.13. *If X, Y are topological spaces and Y is compact, then for any continuous function $f : X \times Y \mapsto \mathbb{R}$, the function $g(x) = \inf_{y \in Y} f(x, y)$ is well defined and continuous. The same holds for $g(x) = \inf_{y \in Y} f(x, y)$.*

The proof of this theorem uses topological arguments that are of no interest for the comprehension of the future arguments. For a proof see [24].

We will need the following version of Theorem 5.5.

Theorem 5.14. *(Strong Separating Rational Hyperplane Theorem) Let X_1, X_2 be two disjoint, non-empty, convex and compact subsets of $\mathbb{R}^n$. Then, there exists $(\vec{p}, t) \in \mathbb{Q}^n \times \mathbb{Q}$ such that $H(\vec{p}, t)$ strongly separates X_1 and X_2.*

Proof. From Theorem 5.5 we know that there exists $(\vec{p}, d) \in \mathbb{R}^n \times \mathbb{R}$ such that $H(\vec{p}, d)$ strongly separates X_1 and X_2. Consider the functions

$$\begin{aligned} f(z) &= \sup_{x \in X_1} x \cdot z, \\ g(z) &= \inf_{x \in X_2} x \cdot z, \\ h(z) &= g(z) - f(z). \end{aligned}$$

From Theorem 5.13 these functions are continuous. This implies that for all $\epsilon > 0$ there exists $\delta > 0$ such that $|h(z) - h(p)| < \epsilon$ for all $z \in B(p, \delta)$. If we take $\epsilon = h(p)$, we have for all $z \in B(p, \delta)$

$$\begin{aligned} |h(z) - h(p)| &< h(p) \\ -h(p) < h(z) - h(p) &< h(p) \\ 0 < h(z) &< 2h(p). \end{aligned}$$

Since rationals are dense, there exists $q \in B(p, \delta) \cap \mathbb{Q}^n$ such that $0 < h(q) = g(q) - f(q)$, which implies $f(q) < g(q)$. Again, from density of rationals, we know that there exists $t \in (f(q), g(q)) \cap \mathbb{Q}$. We conclude that the hyperplane $H(\vec{q}, t)$ strongly separates X_1 and X_2. □

Finally, we present an algorithm that, for X a subset of $\mathbb{R}^n$ with $int(CH(X)) \neq \emptyset$, computes a countable set of closed rational halfspaces L such that $\cap_{\ell \in L} \ell = \overline{CH(X)}$.

```
Algorithm 20: Compute closure of convex hull as a countable intersection of rational halfspaces
    Input:  $X \subset \mathbb{R}^n$ a countable set with $int(CH(X)) \neq \emptyset$.
    Output: A family of closed rational halfspaces whose intersection is $\overline{CH(X)}$.
  1 Let $L$ be an empty set. For each pair $(\vec{c}, d) \in \mathbb{Q}^n \times \mathbb{Q}$, with $c \neq 0$, check whether
        a) $\mathbb{Q}^n \cap H(\vec{c}, d, >) \cap \overline{CH(X)} = \emptyset$?
        b) $\mathbb{Q}^n \cap H(\vec{c}, d, <) \cap \overline{CH(X)} = \emptyset$?
           (This process is done by checking for each rational if it lies inside each set. Use the functions in
           Definition 3.36 to do it).
    We distinguish cases according to the result of a) and b).
        • If Yes/No: Add $H(\vec{c}, d, \leq)$ to $L$,
        • If No/Yes: Add $H(\vec{c}, d, \geq)$ to $L$,
        • If No/No: Do not add a halfspace to L,
    Return L
```

Remark 5.15. In Algorithm 20, the case Yes,Yes cannot happen because that would imply that $\overline{CH(X)}$ is contained in a hyperplane of dimension $n - 1$, contradiction with having non-empty interior.

Proposition 5.16. *If X is a convex set with $int(X) \neq \emptyset$, then for all $x \in X$ and for all $r > 0$ $B(x, r) \cap int(X)$ is non-empty.*

Proof. From Theorem 5.11 all the points of X are limit points of $int(X)$ so, for all r there is some point of $int(X)$ inside the ball $B(x, r)$. □

Proposition 5.17. *The algorithm 20 outputs a valid solution.*

Proof. Let H be the intersection of the halfspaces of L.

$H \subseteq \overline{CH(X)}$: Suppose that there exists $x \in H \setminus \overline{CH(X)}$. Then by Theorem 5.14, there exists a rational hyperplane $H(\vec{c}, d)$ that strongly separates x and $\overline{CH(X)}$. Suppose that $\overline{CH(X)} \subseteq H(\vec{c}, d, <)$. Since $\overline{CH(X)}$ has non-empty interior it has at least one rational in $\mathbb{Q}^n \cap (H(\vec{c}, d, <))$. We conclude that when this halfspace is studied the answer is Yes/No and $H(\vec{c}, d, \leq)$ is included in L. From here we reach a contradiction with the existence of $x \in H \setminus \overline{CH(X)}$. The case $\overline{CH(X)} \subseteq H(\vec{c}, d, >)$ is analogous.

$\overline{CH(X)} \subseteq H$: Let $x \notin H$. Then, there exists a rational halfspace $H(\vec{c}, d, \leq)$ (the case $\geq$ is analogous) in L such that $x \notin H(\vec{c}, d, \leq)$. If such a halfspace is added to L during the computation, the tests a) and b) were answered Yes/No. This implies that

$$\mathbb{Q}^n \cap H(\vec{c}, d, >) \cap \overline{CH(X)} = \emptyset. \tag{2}$$

But now notice the following: $x \in H(\vec{c}, d, >)$, an open set. If $x \in \overline{CH(X)}$ then there exists $r > 0$ such that $B(x, r) \subseteq H(\vec{c}, d, >)$ and by Proposition 5.16 this ball contains a point y of $\overline{CH(X)}$. Now again, there exists $0 < s \leq r$ such that $B(y, s) \subseteq B(x, r) \subseteq H(\vec{c}, d, >) \cap \overline{CH(X)}$. From density of rationals, at least one rational point exists inside that ball. Contradiction with equality (2). $\square$

6. Matchings

In this section we study matchings between infinite sets. It is shown how, in the infinite case, the existence of non-crossing perfect matching is not always guaranteed. A set of points $X \subseteq \mathbb{R}^2$ is in general position if it does not have three points in a common line. This represents a major difference with the finite case where there always is a non-crossing perfect matching if the set $X = R \cup B$ is in general position. Additonally, a non-geometric problem is studied, the so-called stable matching problem [9]. Under certain conditions it is shown that a stable matching always exists. This result is from [3]. We start by defining matchings.

Definition 6.1. Given two disjoint sets R, B of $\mathbb{R}^2$, a function $f : R \to B$ is an (R, B)-matching if it is injective.

Definition 6.2. Given two disjoint sets R, B of $\mathbb{R}^2$, we define a perfect matching as a bijection $F : R \to B$.

Remark 6.3. A perfect matching always exists if R and B have the same cardinality.

Definition 6.4. We say that a (R, B)-matching f of $R, B \subset \mathbb{R}^n$ is non-crossing if for each pair of segments $(r, f(r)), (r', f(r'))$, $r \neq r' \in R$, their intersection is empty. We say that two segments cross, if their intersection is non-empty.

First we show the main difference between the infinite and the finite case. The perfect non-crossing matching does not always exist for infinite sets, as illustrated in the following examples. This may happen even when the set $X = R \cup B$ is in general position. These examples also indicate that even further restrictions on the point set, such as existence of minimum distance, or boundedness of the set do not guarantee the existence of a non-crossing perfect matching.

Example 6.5. Consider the set X of points $(x, y) \in \mathbb{R}^2$ such that $x \in \mathbb{N}_{\geq 6}$ and $y = x^2$, see Figure 4. Consider the set $R = \{(x, y) \in X \mid \ x \text{ is a perfect square}\}$ and $B = X \setminus R$. Let F be a bijection between B and R. Consider the segment L that join $(6, 36)$ with $F(6, 36)$. Since the points lie inside a convex curve, any segment connecting a point below L and a point above L must cross L; but there are more points of B than points of R below L. We conclude that at least one segment intersects L in any perfect matching.

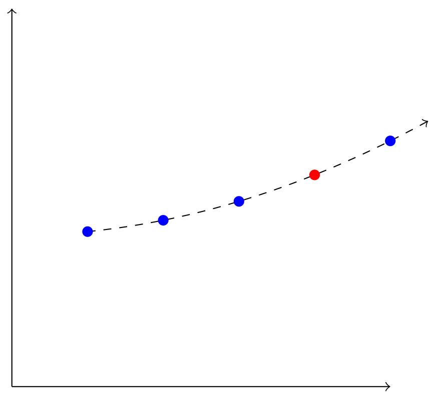

Figure 4: Countable set of points in general position without a non-crossing perfect matching. Example 6.5

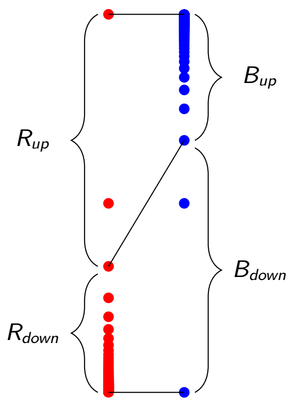

Figure 5: Separated sets without a non-crossing perfect matching. Example 6.7.

Definition 6.6. Two sets $R, B \subseteq \mathbb{R}^2$ are separable if there exists a halfspace that strongly separates them (see Definition 5.4).

We present an example of two separable sets without non-crossing perfect matching.

Example 6.7. Consider $R = \{(0, \frac{1}{n}) \mid n \in \mathbb{N}_{>0}\} \cup \{(0, 0)\}$ and $B = \{(1, 1 - \frac{1}{n}) \mid n \in \mathbb{N}_{>0}\} \cup \{(0, 0)\}$. See Figure 5. If a non-crossing perfect matching exists, then $(0, 1)$ is connected to $(1, 1)$ and $(0, 0)$ with $(1, 0)$. Let $(0, a)$ be a point in R and $(1, b)$ its pair in the matching. The segment that joins them separates R and B in

- $B_{up} = \{(1, y) \in B \mid y > b\}$, an infinite set.
- $B_{down} = \{(1, y) \in B \mid y < b\}$, a finite set.
- $R_{up} = \{(0, y) \in R \mid y > a\}$, a finite set.
- $R_{down} = \{(0, y) \in R \mid y < a\}$, an infinite set.

Now, that makes it impossible not to cross the segment that joins $(0, a)$ and $(1, b)$ in a perfect matching.

Example 6.8. Let $R = \{(0, n) \mid n \in \mathbb{N}_{>0}\}$ and $B = \{(x, 0) \mid x \in \mathbb{Q}_{>0}\}$. Suppose that a non-crossing perfect matching F exists. Consider the segments r and s which connect $(0, 1), (0, 2)$ and their images of F, respectively. Clearly, between $F(0, 1)$ and $F(0, 2)$, infinitely many of rationals exist, in particular $p = \frac{1}{2}(F(0, 1) + F(0, 2))$. The preimage of p must be of the form $(0, n)$ with $n > 2$, which implies that the segment that connects p and $(0, n)$ must intersect s. See Figure 6.

Remark 6.9. With the sets R, B defined in Example 6.8, the function $f(0, n) = (n, 0)$ is a non-crossing (R, B)-matching. But no non-crossing (B, R)-matching exists. With the sets R, B defined in Example 6.5 no non-crossing (B, R)-matching exists and no non-crossing (R, B)-matching exists.

From examples 6.5, 6.7 and 6.8 we see that boundedness, separability or existence of a minimum distance does not imply the existence of a non-crossing perfect matching. In the following subsection we will see that having well-orderings allows stable matchings.

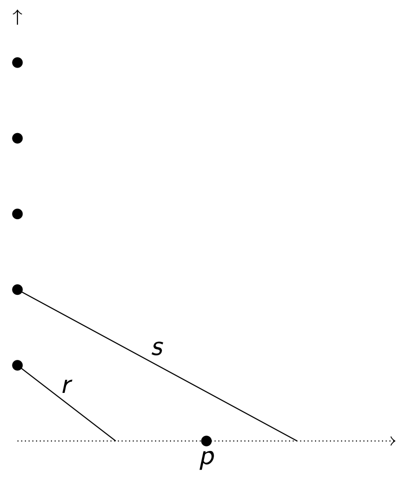

Figure 6: Example 6.8.

6.1 Stable Marriage

The stable marriage problem consists in two finite lists $B = \{b_1, b_2, ..., b_n\}$, $G = \{g_1, g_2, ..., g_m\}$ where each $b \in B$ has a list of preference for the elements of G. Respectively each $g \in G$ has a list of prefence for the elements of B. Formally this preference is stated as a binary relation $<$ where $a < a'$ implies that a is prefered over a'. This has been historically stated as a list of boys B and a list of girls G where the preference indicates which boy/girl prefers to marry. Given an injection (or marriage) $M : B \to G$, b and g are matched if $M(b) = g$. A marriage is unstable if there exists a pair $(b, g) \in B \times G$ such that they are not matched but b prefers g over their matched partner and g also prefers b over its matched partner. Such a pair is called blocking pair. A marriage is stable if it is not unstable. This problem was presented and solved by Gale and Shapley in [9].

First we formalize the definitions.

Definition 6.10. Let B, G be sets such that for each $b \in B$ there exists a well-order $<$ on G, for each $g \in G$ there exists a well-order $<'$ on B and let $M : G \to B$ be a matching. Let $g, g' \in G$ with respective orders $<_g, <_{g'}$ on B and $b, b' \in B$ with respective orders $<_b, <_{b'}$ on G.

- We say that g prefers b over b' if $b < b'$.
- We say that g and b are matched if $M(g) = b$.
- We say that b and g are a blocking pair if they are not matched and prefer each other over their matched partner of M.

We state the infinite stable marriage problem as follows, see [3].

Definition 6.11. A stable marriage problem instance is a tuple $\langle B, G, \{<_i\}_{b_i \in B}, \{<^j\}_{g_j \in G}\rangle$ such that

- $B = (b_i)_{i\in\mathbb{N}}$ and $G = (g_j)_{j\in\mathbb{N}}$ are sequences.
- For each $i \in \mathbb{N}$ the ordered set $(G, <_i)$ is isomorphic to ω.

- For each $j \in \mathbb{N}$ the ordered set $(B, <^j)$ is isomorphic to ω.

Remark 6.12. Each $<_i, <^j$ encodes the preference of each element. If $b_x <^j b_y$ then g_j prefers b_x over b_y. Since $(G, <_i), (B, <^j)$ are isomorphic to ω they are well-orders, which implies that we always have a least element for any non-empty subset.

Definition 6.13. Let $\langle B, G, \{<_i\}_{b_i \in B}, \{<^j\}_{g_j \in G}\rangle$ be a stable marriage problem instance. $M : B \to G$ (or $M : G \to B$) is a stable matching if there is no blocking pair $(b, g) \in B \times G$.

Now we state the result presented in [3].

Theorem 6.14. *([3]) Let $\langle B, G, \{<_i\}_{b_i \in B}, \{<^j\}_{g_j \in G}\rangle$ be an instance of the stable marriage problem. Then there is a stable (R, B)-matching or a stable (B, R)-matching.*

Proof. The proof is constructive. Start matching b_0 with the first girl of his preference $<_0$ (i.e., least element of $(G, <_0)$). This girl exists because $<_i$ is a well-order. Note that we could as well start matching g_0 with the first boy of her preference.

At the end of step s assume we have a matching $M_s = \{(b_0, g_{\sigma(0)}), ...(b_{s-1}, g_{\sigma(s-1)}\}$. We consider now the element b with least index in B that is not matched. We check for the least element $g \in (G, <_s)$ such that g is not matched or prefers b over the current matched partner. If g is not matched then we match b and g and start step $s+1$ with b' the element with least index in B that is not matched. If g was matched to some b'', we update M_s removing (b'', g) and adding (b, g). Now we do again the procedure (round) stated before with b'' (we are still in the same step). Notice that after a finite number of rounds we will match some b with some unmatched g. But in each such change some of the g changes its partner to a preferred one; this can happen only a finite number of times for each g and thus only for a finite number of rounds in total.

It can be seen from induction that, at each step one new girl is matched and that she will remain matched at the end of each following step (this is done in the original proof in [9]). In particular, at each step the girls that change partner will do it only to partners that are higher in their preference. Since there are finitely many partners with higher preference than the first one, for each girl g there exists a finite step t_g after which g does not change partner. Then we will have

$$M = \{(b_i, g_{\tau(i)}) \mid (b_i, g_{\tau(i)}) \in M_s \text{ for all steps after some finite step } t_{g_{\tau(i)}}\}.$$

Let us prove that M is a stable matching.

First we prove injectivity. It is enough to see that if g (or b) is inside some pair in M no other pair can have g (or b). Let $(b, g) \in M$ be the pair that contains b with lowest t_g. The element b will never be considered after step t_g in the procedure since b is already paired and will not be unpaired. That implies that no (b, g') can be added. Then (b, g) is the only pair that contains b in M. Let (b, g) the pair that contains g with lowest t_g. After step t_g no b' will be considered such that $b' < b$, which implies that no pair (b', g) will be added. Since in both cases the pair has the lowest t_g no pair containing b or g, respectively, can be in M.

If some $b' \in B$ is not matched, let b be the least of all unmatched elements of B. Then there are infinitely many steps where b was left unmatched. But if b has been left unmatched infinitely many times it must have checked all girls in their preference and checked if they were unmatched or matched with a lower preference b''. That implies that all girls are matched. Then M is a (B, G)-matching or a (G, B)-matching.

Now we prove the stability of M. Suppose that $(b, g) \in B \times G$ is a blocking pair, i.e., contradicts the stability of M. Then, from the argument used before, if b is not matched, it must have checked at some step if g was unmatched or if she preferred her current match. If the match (b, g) was in M_s after that step, then some other b' must have checked g and removed this matching from M indicating that $b' < b$ with $<$ the preference of g. Alternatively if b is matched with some g' two things can happen. If $g' < g$ with $<$ the preference of b, then (b, g) is not a blocking pair. And if $g < g'$, then g must have been checked before g'. Since it is not matched with b it must be matched with a b' such that $b' < b$, with $<$ being the preference of g. In conclusion, M is a stable matching. □

Remark 6.15. Note that not all the elements of B must be matched to obtain a solution. The solution is not always a perfect stable matching.

From this result we can extract an algorithm to compute stable matchings. The following Algorithm 21 will be used as a sub-routine.

Algorithm 21: Well order

```
    Input: A well-order $<$ and two natural numbers $n, m$
    Output: Yes/No to the question $n < m$
1   Set $R = (<, n, m, 0, ...)$
2   $R_3$ = decimal digit of $<$ at position $\pi(n, m)$ (Algorithm 7);
3   if $R_3 \leq 0$ then
4       Return Yes;
5   else
6       Return No;
```

Corollary 6.16. *Let $I = \langle B, G, \{<_i\}_{b_i \in B}, \{<^j\}_{g_j \in G}\rangle$ be an instance of the stable marriage problem. Then there is an ITBSS computation that finds a stable matching of I.*

Proof. The proofs follows from Algorithm 22. This algorithm will reproduce the procedure of the proof of Theorem 6.14 identifying B and G with $\mathbb{N}$. The output will be a positive real number with the decimal digit at position $\pi(n, m)$ 1 if n and m are matched, and 0 otherwise.

□

Remark 6.17. In Algorithm 22, the value b should be used carefully to avoid a divergence in the register. To do so use the trick of storing $\frac{1}{b+1}$, instead. This has been eliminated from the code to make it more understandable.

Remark 6.18. A stronger result is presented in [3] where a stable matching exists when the ordered sets $(G, <_i), (B, <^j)$ are isomorphic to some ordinal, not necessarily ω. Hence, using a similar algorithm as the one given in the proof of Theorem 6.14 it could be possible to extend Corollary 6.16 under some reasonable conditions. This limitation should guarantee that the limitations shown in Theorem 3.16 are not broken.

Algorithm 22: Infinite Stable Marriage Algorithm

```
Input: Two real numbers that code the families $(<_i)_{i\in\mathbb{N}}$ and $(<^j)_{i\in\mathbb{N}}$ (Definition 3.30 6))
Output: A $(B,G)$ or $(G,B)$-matching
Set $b = 0$ (we identify $B$ and $G$ with $\mathbb{N}$);
Set $GM = 0$ (the decimal digit $n$ is 1 if $g_n$ is matched);
Set $BM = 0$ (the decimal digit $n$ is 1 if $b_n$ is matched);
Set $M = 0$ (the decimal digit $\pi(n,m) = 1$ if $b_n$ and $g_m$ are matched);
Search the least element $g \in (G, <^b)$ that is not matched or prefers $b$ over their actual partner $b'$
  (i.e. $b <_g b'$; use Algorithm 21).;
if $g$ was not matched then
    Set the decimal digit $\pi(b,g)$ of $M$ equal to 1;
    Set the decimal digit $b$ of $BM$ equal to 1;
    Set the decimal digit $g$ of $BM$ equal to 1;
    Set $b$ to the first decimal position in $GM$ with digit 0. (Next unmatched $b$; see Remark 6.17).;
else
    Set the decimal digit $\pi(b,g)$ of $M$ equal to 1;
    Set the decimal digit $\pi(b',g)$ of $M$ equal to 0;
    Set the decimal digit $b$ of $BM$ equal to 1;
    Set the decimal digit $g$ of $BM$ equal to 1;
    Set $b = b'$ (see Remark 6.17);
if $GM \neq 0.1111...$ and $BM \neq 0.1111...$ then
    Go to Step 5;
else
    Return $M$;
```

7. Conclusions

This project had two main objectives: (i) present a framework to explore the capacity of study and solvability of problems with infinite input and (ii) present algorithms inside that framework that can solve some problems. This study has focused on classical geometrical problems and the new questions that emerge from the infiniteness of the input.

In Section 2 an introduction to ordinal numbers and their basic operations has been made. This allows the understanding of infinite ordered sets which are fundamental for an effective algorithmic approach.

In Section 3 we have presented the structure of the whole framework. A presentation of the ITBSS machines has been done presenting definitions of program and computation. Several approachable examples of computation were shown. Some known properties of the model have been presented. Finally, a discussion on the capacity of the machine was made, showing codification and decodification algorithms.

Section 4 studied a proper problem of infinite inputs, the accumulation points problem. We have analysed accumulation points, presenting the mass center of a countable set and how it relates to the accumulation points. This study concluded with the presentation and proof of new algorithms to find and identify accumulation points on the real plane. These results are limited to finite sets of accumulation points, an interesting questions could be how to approach more general sets of accumulation points, or how to decide if the number of accumulation points is finite.

Section 5 studied the classical problem of convex hull computation of a set of points. Several known properties were stated, rational hyperplanes were presented and with the combination of diverse new algorithms a final program to compute closure of convex hulls was presented.

Section 6, which focuses on matching problems, showed some differences between the finite and infinite instance of some matching problems. It ends with the presentation of a result in [3] proving the capacity to make a stable matching under reasonable conditions on the input.

The nature of this project leaves many open questions. The extension of many other geometric problems to the infinite model could represent an interesting topic in several future projects. The study of Voronoi diagrams in sets with accumulation points, linear programming, and many more. Another interesting question is how to further adapt the model to handle the inputs.

The main questions that emerge from the topics studied in this thesis are about generalizations of the search of accumulation points, a promising method is using subdivisions of the space and progressively refine them. The use of subdivision also presents an interesting question in how to handle the information created by algorithms. In the particular case of convex hull, the use of rational hyperplanes allowed us to store them inside only one register using the methods presented in Section 3. The matching problem presents many questions. When does a non-crossing perfect matching exist? And a (B, R)-matching? All questions represent an interesting continuation of this project.

Another interesting line of research would be to study the complexity classes of these algorithms. There are two main options. The number of steps, which is upper bounded by Theorem 3.16, could be a natural extension to the classical classes. On the other hand, an interesting approach would be the one derived from analytic set theory. In the Descriptive set theory there exists a hierarchy of sets, which classifies the sets depending in how "complicated" they are. This approach can be seen in [3] (Corollary 2.3). For more information [14] and [18] are good references.

All together we have presented a different framework to confront countable infinite input problems and several results concerning the different topics studied.

8. Bibliography

References

[1] Lenore Blum, M. Shub, and Steve Smale. "On a Theory of Computation and Complexity over the Real Numbers: NP-Completeness, Recursive Functions and Universal Machines." In: *Bull. Amer. Math. Soc.* 21 (July 1989), pp. 1–46. DOI: `10.1090/S0273-0979-1989-15750-9`.

[2] G. Cantor. "Ein Beitrag zur Mannigfaltigkeitslehre." In: *Journal für die reine und angewandte Mathematik* 84 (1877), pp. 242–258. URL: `http://eudml.org/doc/148353`.

[3] Douglas Cenzer and Jeffrey B. Remmel. "Proof-theoretic strength of the stable marriage theorem and other problems". In: *Reverse Mathematics 2001*. Ed. by Stephen G.Editor Simpson. Lecture Notes in Logic. Cambridge University Press, 2005, 67–103. DOI: `10.1017/9781316755846.005`.

[4] T. M. Chan. "Optimal output-sensitive convex hull algorithms in two and three dimensions". In: *Discrete & Computational Geometry* 16.4 (1996), pp. 361–368. DOI: `10.1007/bf02712873`. URL: `https://doi.org/10.1007%2Fbf02712873`.

[5] Bernard Chazelle. "An optimal convex hull algorithm in any fixed dimension". In: *Discrete & Computational Geometry* 10.4 (1993), pp. 377–409. DOI: `10.1007/bf02573985`. URL: `https://doi.org/10.1007%2Fbf02573985`.

[6] Stephen A. Cook and Robert A. Reckhow. "Time bounded random access machines". In: *Journal of Computer and System Sciences* 7.4 (1973), pp. 354–375. DOI: `10.1016/s0022-0000(73)80029-7`. URL: `https://doi.org/10.1016%2Fs0022-0000%2873%2980029-7`.

[7] Oscar Cunningham. *A better representation for real numbers*. `https://oscarcunningham.com/494/a-better-representation-for-real-numbers/`.

[8] Patrick Fitzpatrick. *Advanced calculus*. Belmont, CA: Thomson Brooks/Cole, 2006. ISBN: 9780534376031.

[9] D. Gale and L. S. Shapley. "College Admissions and the Stability of Marriage". In: *The American Mathematical Monthly* 69.1 (1962), pp. 9–15. ISSN: 00029890, 19300972. URL: `http://www.jstor.org/stable/2312726`.

[10] Derek Goldrei. *Classic set theory : a guided independent study*. Chapman & Hall, 1996. ISBN: 978-0412606106.

[11] Joel David Hamkins and Andy Lewis. "Infinite Time Turing machines". In: *Journal of Symbolic Logic* 65.2 (2000), 567–604. DOI: `10.2307/2586556`.

[12] A. O. Ivanov and A. A. Tuzhilin. *Minimal Spanning Trees on Infinite Sets*. 2014. arXiv: `1403.3831 [math.MG]`. URL: `https://arxiv.org/abs/0811.1525`.

[13] Thomas Jech. *Set theory*. Berlin New York: Springer, 2003. ISBN: 978-3-540-44761-0.

[14] Alexander Kechris. *Classical Descriptive Set Theory*. New York, NY: Springer New York, 1995. ISBN: 978-1-4612-4190-4.

[15] Peter Koepke and Benjamin Seyfferth. "Towards a Theory of Infinite Time Blum-Shub-Smale Machines". In: *How the World Computes*. Ed. by S. Barry Cooper, Anuj Dawar, and Benedikt Löwe. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 405–415. ISBN: 978-3-642-30870-3.

[16] I. E. Leonard. *Geometry of Convex Sets*. City: Wiley, 2015. ISBN: 978-1119022664.

[17] Valeriu Soltan. *Lectures on convex sets*. World Scientific, 2015. ISBN: 978-9814656689.

[18] S. M. Srivastava. *A course on Borel sets*. New York: Springer, 1998. ISBN: 978-0-387-22767-2.

[19] Josef Stoer. *Convexity and Optimization in Finite Dimensions I*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1970. ISBN: 978-3-642-46216-0.

[20] William Terrell. *A passage to modern analysis*. American Mathematical Society, 2019. ISBN: 978-1-4704-5135-6.

[21] Godfried T Toussaint. "A historical note on convex hull finding algorithms". In: *Pattern Recognition Letters* 3.1 (1985), pp. 21–28. DOI: 10.1016/0167-8655(85)90038-8. URL: https://doi.org/10.1016%2F0167-8655%2885%2990038-8.

[22] A. M. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". In: *Proceedings of the London Mathematical Society* s2-42.1 (Jan. 1937), pp. 230–265. ISSN: 0024-6115. DOI: 10.1112/plms/s2-42.1.230. eprint: https://academic.oup.com/plms/article-pdf/s2-42/1/230/4317544/s2-42-1-230.pdf. URL: https://doi.org/10.1112/plms/s2-42.1.230.

[23] Ina Voigt. *Voronoi cells of discrete point sets*. 2008. arXiv: 0811.1525 [math.CO].

[24] Willie Wong. *Continuity of the infinitum*. https://williewong.wordpress.com/2011/11/01/continuity-of-the-infimum/.

[25] Günter M. Ziegler. *Lectures on Polytopes : Updated Seventh Printing of the First Edition*. New York, NY: Springer New York, 1995. ISBN: 978-1-4613-8431-1.

A. Density of rationals

A good understanding of $\mathbb{R}^n$ and $\mathbb{Q}^n$ is assumed from the reader, nonetheless, since the density of rationals is used in several proofs, this section states some basic properties derived from the Arquimedean property. The main reference is [20].

Definition A.1 ([20])**.** (The Archimedean property) For any real $x > 0$ exists $n \in \mathbb{N}_{>0}$ such that

$$0 < \frac{1}{n} < x.$$

Proposition A.2 ([20])**.** *For any real a, the interval $(a, a+1]$ contain an integer.*

Proof. Assume $a > 0$ and let $S = \{n \in \mathbb{N} \mid n > a\}$, the Archimedean property implies that there is $t \in \mathbb{N}$ such that $\frac{1}{t} < \frac{1}{a}$ which implies $t > a$. Then S is non-empty. Now consider the minimum element m of S. If $m > a+1$, then $m-1 > a$ and $m-1 \in S$ which contradicts minimality of m. We conclude $a < m \leq a+1$. $\square$

The rationals $\mathbb{Q}$, seen as a subset of reals $\mathbb{R}$, are dense in the following sense.

Proposition A.3 ([20])**.** *(Density of rationals in $\mathbb{R}$) For any $a, b \in \mathbb{R}$ with $a < b$ there is a rational q such that $a < q < b$.*

Proof. The Archimedean property applied to $b - a > 0$ implies the existence of $n \in \mathbb{N}$ such that

$$0 < \frac{1}{n} < b - a$$

which implies $na + 1 < nb$. From Proposition A.2 there exists an integer m in $(na, na+1]$. All together

$$na < m \leq na + 1 < nb$$

which proves that $\frac{m}{n}$, rational, lies between a and b. $\square$

Theorem A.4. *(Density of rationals in $\mathbb{R}^n$) For any $a \in \mathbb{R}^n$ and $r \in \mathbb{R}_{>0}$ the open ball $B(a, r)$ contains a rational $q \in \mathbb{Q}^n$.*

Proof. The proof is inductive. Case $n = 1$: it corresponds to Proposition A.3. Assume the statement holds for $n-1$. Let $B = B(a, r)$ be a ball in $\mathbb{R}^n$. The points inside the ball are points in $\mathbb{R}^{n-1} \times \mathbb{R}$. The projection,

$$\begin{aligned} \pi_1 : \mathbb{R}^{n-1} \times \mathbb{R} &\to \mathbb{R}^{n-1} \\ (x, y) &\to x \end{aligned}$$

is an open function, i.e., sends open sets to open sets.

Let $B_1 \neq \emptyset$ be the image $\pi_1(B)$. Since B_1 is open, there exists $r_1 > 0$ such that $B(\pi_1(a), r_1) \subset B_1$. From induction hypothesis, there exists a rational point $q_1 \in \mathbb{Q}^{n-1} B(\pi_1(a), r_1)$. Let $P = \pi_1^{-1}(q_1) \cap B$. P is an open set because π_1 is continuous. We consider now the projection

$$\begin{aligned} \pi_2 : \mathbb{R}^{n-1} \times \mathbb{R} &\to \mathbb{R} \\ (x, y) &\to y \end{aligned}$$

Let $P_2 = \pi_2(P)$. From being an open function P_2 is an open set. Using the case $n = 1$ we know that there exists a rational element in P_2, call it q_2. Now let $P_2 = \pi_2^{-1}(q_2) \cap P$. Since the projection functions do not alter the values of the entries, all the elements in P_2 are in $\mathbb{Q}^n$. Now P_2 is non-empty because q_2 was chosen from the set $\pi_2(P)$. $\square$

B. Auxiliar algorithms

Many algorithms are used in different proofs during the text. For completeness, several are shown here.

Remark B.1. When the value of the registers is set as $R = (R_0, R_1, \ldots)$ the dots indicate that a finite number of registers are added to execute secondary computations.

Algorithm 23: Exponentiation

```
Input: $(x, z) \in \mathbb{R} \times \mathbb{Z} \setminus \{(0,0)\}$.
Output: $x^z$.
Set $R = (x, z, 1, 1)$;
if $R_1 < 0$ then
    $R_2 = -1$;
    $R_1 = -R_1$;
while $R_1 > 0$ do
    $R_3 = R_0 \cdot R_3$;
    $R_1 = R_1 - 1$;
if $R_2 < 0$ then
    Return $\frac{1}{R_3}$;
else
    Return $R_3$;
```

Algorithm 24: Floor function

```
Input: $x \in \mathbb{R}$.
Output: $\lfloor x \rfloor$.
Set $R = (x, 0, 1)$;
if $R_0 < 0$ then
    $R_2 = -1$;
    $R_0 = -R_0$
while $R_1 < R_0$ do
    $R_1 = R_1 + 1$
if $R_1 == R_0$ then
    Return $R_1 \cdot R_2$
else if $R_2 < 0$ then
    Return $-R_1$
else
    Return $R_1 - 1$
```

```
Algorithm 25: Number of integer-digits of a number
    Input: $x \in \mathbb{R}$.
    Output: $n \in \mathbb{N}$ number of integer-digits of $x$.
1   Set $R = (x, 1, 0)$;
2   if $R_0 < 0$ then
3       $R_0 = -R_0$
4   while $R_0 \geq R_1$ do
5       $R_1 = 10R_1$;
6       $R_2 = R_2 + 1$
7   Return $R_2$
```

```
Algorithm 26: Binary expansion of a natural number as a natural number
    Input: $x \in \mathbb{N}$.
    Output: $n \in \mathbb{N}$ number with the same digits as the binary expansion of $x$.
1   Set $R = (x, 0, 0, 0, ...)$;
2   while $R_0 > 0$ do
3       $R_3 = \lfloor \frac{R_0}{2} \rfloor$ (algorithm 24);
4       if $R_3 \neq \frac{R_0}{2}$ (odd?) then
5           $R_1 = \frac{R_1+1}{10}$;
6           $R_0 = \frac{R_0-1}{2}$;
7       else
8           $R_0 = \frac{R_0}{2}$;
9           $R_1 = \frac{R_1}{10}$;
10      $R_2 = R_2 + 1$
11  $R_2 = 10^{R_2} R_1$ (algorithm 23);
12  Return $R_2$;
```

Remark B.2. Algorithm 26 is the algorithm “Divide by two” that uses R_0 as a stack.

```
Algorithm 27: Square root of a non-negative real number
    Input: $r$ non-negative real number.
    Output: $\sqrt{r}$
1   Set $R = r, 0, 1$;
2   if $R_0 == 0$ then
3       Return $R_0$;
4   $R_1 = \frac{r}{2}$;
5   while $R_2 > 0$ do
6       $R_1 = \frac{1}{2}\left(R_1 + \frac{R_0}{R_1}\right)$;
7       $R_2 = \frac{R_2}{R_2+1}$;
8   Return $R_1$;
```

Remark B.3. Algorithm 27 is the Babylonian method to compute square roots.