UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – 2. stopnja

Anja Petković

RAČUNANJE FIKSNIH TOČK ODSEKOVNO LINEARNIH FUNKCIJ

Magistrsko delo

Mentor: prof. dr. Alex Simpson

Ljubljana, 2017

UNIVERSITY OF LJUBLJANA
FACULTY OF MATHEMATICS AND PHYSICS

Mathematics – 2nd degree

Anja Petković

COMPUTING FIXED POINTS OF MONOTONE PIECEWISE LINEAR FUNCTIONS

Master thesis

Adviser: prof. dr. Alex Simpson

Ljubljana, 2017

Univerza v Ljubljani
Fakulteta za matematiko in fiziko

Izjava o avtorstvu, istovetnosti tiskane in elektronske verzije magistrskega dela in objavi osebnih podatkov študenta

Spodaj podpisana študentka Anja Petković avtorica magistrskega dela (v nadaljevanju: pisnega zaključnega dela študija) z naslovom:

Računanje fiksnih točk odsekovno linearnih funkcij

IZJAVLJAM

1. *Obkrožite eno od variant a) ali b)*

   a) da sem pisno zaključno delo študija izdelala samostojno;

   b) da je pisno zaključno delo študija rezultat lastnega dela več kandidatov in izpolnjuje pogoje, ki jih Statut UL določa za skupna zaključna dela študija ter je v zahtevanem deležu rezultat mojega samostojnega dela;

   pod mentorstvom prof. dr. Alexa Simpsona

2. da je tiskana oblika pisnega zaključnega dela študija istovetna elektronski obliki pisnega zaključnega dela študija;

3. da sem pridobila vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v pisnem zaključnem delu študija in jih v pisnem zaključnem delu študija jasno označila;

4. da sem pri pripravi pisnega zaključnega dela študija ravnala v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobila soglasje etične komisije;

5. da soglašam, da se elektronska oblika pisnega zaključnega dela študija uporabi za preverjanje podobnosti vsebine z drugimi deli s programsko opremo za preverjanje podobnosti vsebine, ki je povezana s študijskim informacijskim sistemom fakultete;

6. da na UL neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve avtorskega dela v elektronski obliki, pravico reproduciranja ter pravico dajanja pisnega zaključnega dela študija na voljo javnosti na svetovnem spletu preko Repozitorija UL;

7. da dovoljujem objavo svojih osebnih podatkov, ki so navedeni v pisnem zaključnem delu študija in tej izjavi, skupaj z objavo pisnega zaključnega dela študija.

Kraj: Podpis študentke:

Datum:

University of Ljubljana
Faculty of Mathematics and Physics

Declaration of authorship, identity of electronic and printed versions of the master thesis and publication of personal data

I, the undersigned student Anja Petković the author of master thesis (below the written final work of studies), entitled:

Computing fixed points of monotone piecewise linear functions

DECLARE

1. *The following (choose a) or b)):*

   a) The written final work of studies is a result of my independent work.

   b) The written final work of studies is a result of own work of more candidates and fulfils the conditions determined by the Statute of UL for joint final works of studies and is a result of my independent work in the required share.

   under supervision of prof. dr. Alexa Simpsona

2. The printed form of the written final work of studies is identical to the electronic form of the written final work of studies.

3. I have acquired all the necessary permissions for the use of data and copyrighted works in the written final work of studies and have clearly marked them in the written final work of studies.

4. I have acted in accordance with ethical principles during the preparation of the written final work of studies and have, where necessary, obtained agreement of the ethics commission.

5. I give my consent to use of the electronic form of the written final work of studies for the detection of content similarity with other works, using similarity detection software that is connected with the study information system of the university member.

6. I transfer to the UL – free of charge, non-exclusively, geographically and time-wise unlimited – the right of saving the work in the electronic form, the right of reproduction, as well as the right of making the written final work of studies available to the public on the world wide web via the Repository of UL.

7. I give my consent to publication of my personal data that are included in the written final work of studies and in this declaration, together with the publication of the written final work of studies.

Place: Student's signature:

Date:

Acknowledgements

First, I would like to thank my advisor prof. dr. Alex Simpson for all the work he has done for this project. For providing me with a very interesting topic, helping me find all the necessary references, taking time for our long weekly meetings, used to discuss all the issues and ideas needed to proceed, giving me detailed comments on my work, kindly helping me with the time frame to finish the project and mostly providing the atmosphere in which I enjoyed working.

A big thanks goes to my family. To my parents for (financially and emotionally) supporting me during all these years of studying, to my sisters for their moral support and patience, and to my partner, who tolerated all the time shortage during exams and projects.

I would also like to thank prof. dr. Sergio Cabello for some pointers during the project. Thanks to ms. Nika Vardjan Naglič and prof. dr. Andrej Bauer for all the help with the bureaucracy. And of course to all the professors at Faculty of Mathematics and Physics, who inspired me during the past few years.

Lastly, I would like to thank my schoolmates for the best university experience and all the ideas we have shared.

Contents

Program dela

Program dela v slovenščini

Cilj projekta je raziskati iterativni algoritem za iskanje negibnih točk monotonih odsekovno linearnih funkcij iz [1]. Raziskalo se bo tako iz teoretičnega kot iz eksperimentalnega vidika.

V teoretičnem delu se bo študentka naučila matematičnega ozadja vključno s teorijo negibnih točk za urejene množice [2] in eliminacijo kvantifikatorjev za predikatno teorijo linearne aritmetike [3]. Glavni cilj teoretičnega dela je razumevanje dokaza pravilnosti algoritma iz [1].

V eksperimentalnem delu bo študentka implementirala program za algoritem iz [1] in raziskala njegovo praktično učinkovitost na testnih primerih.

Če bo čas dopuščal, ko bosta zgoraj navedena glavna cilja dosežena, se lahko projekt razširi z razvijanjem in raziskovanjem različic algoritma z namenom optimizacije časovne in/ali prostorske učinkovitosti. Takšne različice morajo biti opremljene z dokazom pravilnosti.

Program dela v angleščini

The goal of the project is to investigate the iterative algorithm for finding fixed-points of monotone piecewise linear functions from [1]. This will be done from both theoretical and experimental angles.

On the theoretical side, the student will learn the background mathematics, which includes fixed-point theory for ordered sets [2], and quantifier elimination for the first-order theory of linear arithmetic [3]. The main theoretical goal is to understand the correctness proof for the algorithm in [1].

On the experimental side, the student will implement a program for the algorithm from [1], and explore its practical efficiency on test examples.

If time permits, once the two main goals above have been achieved, the project may be further extended by developing and investigating variations on the algorithm, aiming for time and/or space-efficiency improvements. Such variations should be proven correct.

Literatura

[1] M. Mio and A. Simpson. *Łukasiewicz μ-calculus.* Fundamenta Informaticae 150:317—346, 2017.

[2] A. Arnold and D. Niwinsky. *Rudiments of μ-calculus*, Studies in Logic and the Foundations of Mathematics 146, Elsevier, Amsterdam, 2001.

[3] J. Ferrante and C. Rackoff. *A Decision Procedure for the First Order Theory of Real Addition with Order.* SIAM Journal of Computing, 4(1):69—76, 1975.

Podpis mentorja:

Računanje fiksnih točk odsekovno linearnih funkcij

Povzetek

Računamo fiksne točke monotonih odsekovno linearnih funkcij. Podan je opis odsekovno linearnih funkcij preko pogojnostnih linearnih izrazov. Dokazano je, da je vsako takšno funckijo mogoče podati s formulo iz predikatne teorije linearne aritmetike. Nekatere monotone odsekovno linearne funkcije je mogoče podati tudi preko μ-termov. Opisan je algoritem za računanje najmanjše in največje negibne točke monotone odsekovno linearne funkcije. Za izboljšanje njegove učinkovitosti podajamo tudi njegovo neposredno posodobitev in optimizacijo preko eliminacije kvantifikatorjev. Vse različice algoritma so dokazano pravilne in implementirane v programskem jeziku python 3. Med seboj so tudi primerjane po eksperimentalni učinkovitosti.

Computing fixed points of monotone piecewise linear functions

Abstract

We take a look at monotone piecewise linear functions and an algorithm, that computes the least and the greatest fixed points of such functions. A description of piecewise linear functions via the conditioned linear expressions is given and it is proven, that such functions can be expressed in the first-order theory of linear arithmetic. Another decription of some monotone piecewise linear functions is given via μ-terms. Several modifications of the fixed point algorithm (a direct modification, a quantifier elimination optimisation) are considered and proven to be correct. All the versions of the algorithm are implemented in python 3 and their performance is compared.

Math. Subj. Class. (2010): 06D30, 03G20 , 03G10, 06B23, 94D05, 03B52, 03C10
Ključne besede: odsekovno linearne funkcije, μ-račun, negibna točka, algoritem
Keywords: piecewise linear functions, μ-calculus, fixed point, algorithm

1 Introduction

In this project we are looking at monotonic piecewise linear functions and various algorithms, that compute the least and the greatest fixed points of such functions. Fixed points have been studied in various settings, from the Banach fixed point theorem [9], that deals with convergence, to the Brouwer fixed point theorem [12], that focuses on the topological point of view and studies functions on discs. Here, we consider fixed points in the light of the order-theoretic Knaster-Tarski theorem [1].

The problem of computing fixed points of the monotone piecewise linear functions originated in the study of algorithms for "model checking" [2] used in the specification and verification of systems cobining nondeterminism and probabilistic behaviour. Since the functions that arise from that are discontinuous, we cannot make a restriction to deal only with continuous functions, therefore we adopt a more general notion of piecewise linearity that does not presuppose continuity.

The main reference for this project is an article by M. Mio and A. Simpson [13], where one version of the desired algorithm for computing fixed points of monotonic piecewise linear functions is described. However, we have made some modifications in order to improve efficiency of the algorithm and those modifications, as well as the modifications in the proof of correctness and termination of the algorithm, are original work. We also suggest an additional optimisation via quantifier elimination, which is an original contribution as well.

The main thesis body is divided into 6 sections. Section 2 is devoted to introducing basic definitions and notation on lattices and fixed points, deriving some necessary properties. The main central result of this part is the Knaster-Tarski Theorem, concerning the existence of least and greatest fixed points. The material for that section is mostly adapted from [1].

In the next section we define piecewise linear functions and state some of their properties. Here we have relied on [13]. However, one contribution is that we make explicit the notion of a *local algorithm* for a piecewise linear function, which is only implicit in [13]. It is designed for the purpose of capturing the properties of the algorithm for computing fixed points.

The entire Section 4 is about the iterative algorithm for computing fixed points of monotonic piecewise linear functions, its modifications and improvements. This section is the central theoretical contribution of the dissertation. The reference is again [13]. Section 5 provides us with an alternative (and more consise) way to describe some monotonic piecewise linear functions as introduced in [13] via μ-calculus.

The next two Sections 6 and 7 give details on the implementation of the algorithms in python 3 and the obtained results. The entire source code available at the link given at the begining of Section 6 is original work of the author, designed solely for this project. Since there is too much code to be included as an appendix, we merely provide a link to an open online repository.

2 Complete lattices, fixed points and the Knaster-Tarski theorem

In this section we will get to know the basics of lattice theory and μ-calculus needed for computing the fixed points. Many of the theorems and their proofs are covered in [1] in detail, we will only state the relevant part of order theory and fixed point calculus.

2.1 Complete lattices

Let us begin by the following definition of a lattice. We will use notation $(E, \leq)$ for an ordered set, where $\leq$ is a partial order (reflexive, antisymmetric and transitive). We will leave out the notation for the ordering and just call E an ordered set, where the relation will be known from the context. Recall that an element $e \in E$ is an *upper bound* of X, where $X \subseteq E$, if for all $x \in X$, it holds that $x \leq e$. Similarly e is a *lower bound* if $e \leq x$ for all $x \in X$. An element $e \in E$ is a *least upper bound* of the set $X \subseteq E$, if it is the least element in the set of all upper bounds, i. e. the conditions

- $\forall x \in X$, $x \leq e$,
- if $\forall x \in X$, $x \leq f$ for some $f \in E$, then $e \leq f$

hold. Similarly for *greatest lower bound*. The least upper bound is unique and we will denote it by $\bigvee X$ (and $\bigwedge X$ for the greatest lower bound).

Definition 2.1. A *lattice* is an ordered set $(E, \leq)$ such that for any two elements $x, y \in E$, the set $\{x, y\}$ has a least upper bound $x \vee y$ and a greatest lower bound $x \wedge y$. A *complete lattice* is an ordered set $(E, \leq)$ such that every subset $X \subseteq E$ has a least upper bound $\bigvee X$ and a greatest lower bound $\bigwedge X$.

In particular a complete lattice has a least element $\bigvee \emptyset$ and a greatest element $\bigwedge \emptyset$.

Example 2.2. *The cannonical example of the complete lattice is the powerset with the subset relation. Let S be any set and $\mathcal{P}(S)$ be the set of all subsets. Then $(\mathcal{P}(S), \subseteq)$ is an ordering and for any $X \subseteq \mathcal{P}(S)$, we have $\bigvee X = \bigcup X$ and $\bigwedge X = \bigcap X$.*

Another way to construct complete lattices is via product. If E and F are ordered sets, we consider the Cartesian product $E \times F$ with the product ordering

$$(e, f) \leq (e', f') \iff (e \leq e' \text{ and } f \leq f').$$

This is trivially generalised for a product of n ordered sets. Recall the projections π_1 and π_2 on E and F respectively. If E and F are complete lattices, their product is also a complete lattice, since for $X \subseteq E \times F$, we have $\bigwedge X = (\bigwedge \pi_1(X), \bigwedge \pi_2(X))$ and $\bigvee X = (\bigvee \pi_1(X), \bigvee \pi_2(X))$.

We note that for an ordered set to be a complete lattice it is sufficient that $\bigvee X$ exists for every subset $X \subseteq E$. It then follows that $\bigwedge X$ exists for every $X \subseteq E$. This is a consequence of the following proposition:

Proposition 2.3. *For $X \subseteq E$ if the set of lower bounds of X has a least upper bound, then this is the greatest lower bound of X.*

Symmetrically if the set of upper bounds of X has a greatest lower bound, then it is the least upper bound of X.

Proof. Let B denote the set of all lower bounds of X. Then $\bigvee X$ is a lower bound of X for the following reason: Let $x \in X$. For all $y \in B$ it holds that $y \leq x$. Therefore x is an upper bound of B and $\bigvee B \leq x$. Because $\bigvee B$ is an element of B, it makes it the greatest element of B. The proof is similar for upper bounds. □

We observe that the notion of the lower bound and the upper bound are in a way very symmetric. If for an ordered set $(E, \leq)$ we define the relation $\leq'$ by

$$x \leq' y \iff y \leq x,$$

we also get an ordered set, and furthermore the least upper bound for a subset $X \subseteq E$ for ordering $\leq$ is the greatest lower bound with ordering $\leq'$ and vice versa. So any property that holds for least upper bounds has its dual formulation for greatest lower bounds and we do not need to prove the propery again. We call this *principle of symmetry* and we will use it to shorten the proofs.

Let us take a look at a useful relation between subsets and least upper bounds.

Proposition 2.4. *Let E be a complete lattice and $X \subseteq X' \subseteq E$. Then*

$$\begin{aligned} \bigwedge X' &\leq \bigwedge X \\ \bigvee X &\leq \bigvee X'. \end{aligned}$$

The statement follows straight from definitions and the proof is not included in [1]. However, let us take a closer look to fully understand the definitions.

Proof. Let $B(X)$ denote the set of lower bounds of X and $B(X')$ the set of lower bounds of X'. By definition of the lower bound for every $b \in B(X')$ and for every $x' \in X'$ it holds that $b \leq x'$. Because $X \subseteq X'$ for every $x \in X$ it automatically holds that $b \leq x$ and b is also a lower bound for X. This implies that $B(X') \subseteq B(X)$. When we find the greatest lower bound of X' we take the maximum element of $B(X')$, but since $B(X)$ has $B(X')$ as a subset, there are more candidates to take a maximum. Therefore $\bigwedge X' \leq \bigwedge X$. The second statement holds by the principle of symmetry. □

The proposition clearly holds for any complete lattice, but in the previous example with the powerset and subset ordering, the statement is obvious. If $X \subseteq X'$, then an element $x \in \bigwedge X'$ is in all subsets $S \in X'$ and therefore also in all subsets $S \in X$. This implies that $x \in \bigcap X$, so $x \in \bigwedge X$. By definition of the relation we get $\bigwedge X' \leq \bigwedge X$. In the second part we take unions instead of intersections and we get a similar result.

2.2 Monotonic functions and fixed points

We are mainly interested in functions between lattices, namely monotonic functions. For ordered sets E and F we will use notation E^F for the set of monotonic functions from F to E. We would like to generalize the notion of monotonic functions on real numbers to be used in ordered sets.

Definition 2.5. Let $(E, \leq_E)$ and $(F, \leq_F)$ be two ordered sets. A function $f\colon E \to F$ is said to be *monotonic*, if

$$\forall x, y \in E, x \leq_E y \implies f(x) \leq_F f(y).$$

This definition coincides with monotonicity we already know from functions on $\mathbb{R}$. Although we are used to specifying whether a function is monotone increasing or decreasing, here we only deal with monotone increasing functions (as in definition 2.5) and we will use the term monotonicity in this sense.

Example 2.6. *Let us revisit the powerset example $(\mathcal{P}(S), \subseteq)$. If $g\colon S \to S$ is any mapping, we can construct a monotonic function $\tilde{g}\colon \mathcal{P}(S) \to \mathcal{P}(S)$ with $\tilde{g}(X) := \{g(x) | x \in X\}$. To prove monotonicity, we take two sets $X, Y \in \mathcal{P}(S)$ with $X \leq Y$, which means $X \subseteq Y$. When we use $\tilde{g}$ on X and Y, we get*

$$\tilde{g}(X) = \{g(x) | x \in X\} \subseteq \{g(x) | x \in Y\} = \tilde{g}(Y).$$

By definition of $\leq$ we get $\tilde{g}(X) \leq \tilde{g}(Y)$.

Here by $f(x)$ we mean the direct image of X under f. (In example 2.6 this was written $\tilde{f}(x)$ in order to distinguish f and $\tilde{f}$. Henceworth, we revert to the standard notation $f(x)$.)

For the proof of Knaster-Tarski theorem later we will also need the following property of monotonic functions.

Proposition 2.7. *Let E and F be complete lattices and let $f\colon E \to F$ be* monotonic. *For any subset $X \subseteq E$,*

$$\bigvee_F f(X) \leq_F f(\bigvee_E X)$$

and

$$f(\bigwedge_E X) \leq_F \bigwedge f(X).$$

Proof. It is enough to show that $f(\bigvee_E X)$ is an upper bound of $f(X)$. Then the least upper bound $\bigvee_F f(X)$ will be smaller or equal by definition. For any $y \in f(X)$ there is $x \in X$ such that $y = f(x)$. Since $x \leq_E \bigvee_E X$ and f is monotonic $y = f(x) \leq_F f(\bigvee_E X)$ and we have an upper bound. The second statement holds by the principle of symmetry. □

We now focus on the functions from an ordered set E onto itself and define a *fixed point*.

Definition 2.8. Let E be any set and $f\colon E \to E$ a function. A *fixed point* of f is an element $x \in E$, such that $f(x) = x$. The set of all fixed points of f is denoted by $\mathrm{Fix}(f)$.

In the case that we have a function $f\colon \mathbb{R} \to \mathbb{R}$, a fixed point would be at an intersection between the graph of the function and the diagonal $y = x$.

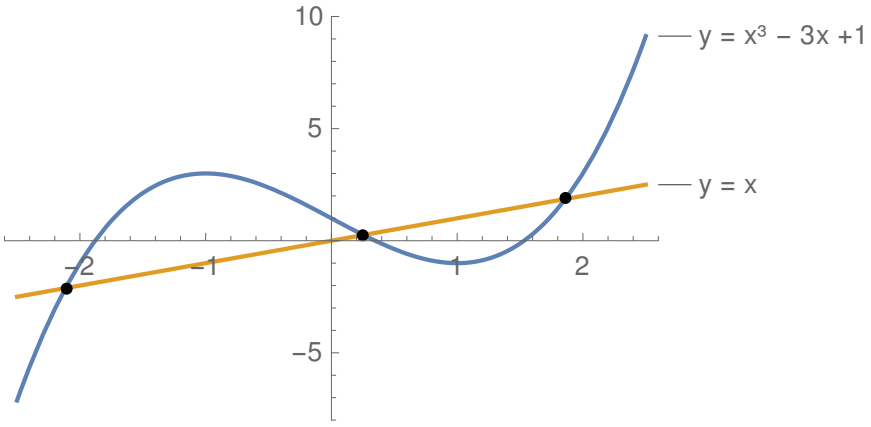


Figure 1: Fixed points of a function $f(x) = x^3 - 3x + 1$.

Example 2.9. *Let us look at function $f(x) = x^3 - 3x + 1$ and its graph on Figure 1. We can observe that f has three fixed points, so $|\mathrm{Fix}(f)| = 3$. Note that fixed points are only x-coordinates, but for clarity intersections are marked.*

Finding fixed points is a standard problem in many different contexts. For example the Banach fixed point theorem (see [9]) is used for finding fixed points in metric spaces, but if we are trying to find a fixed point numerically, we translate this problem to finding a zero of a function $f - \mathrm{id}$. However in this project we are dealing with special functions and therefore have an alternative algorithm, that finds a fixed point as an exact rational number (but more on that later).

When E is an ordered set, $\mathrm{Fix}(f)$ is an ordered subset of E and it can be empty.

2.3 Knaster-Tarski theorem

The most interesting fixed point for us are the least and the greatest fixed points of a monotonic function. But so far we have not even proven the existence of a fixed point for such functions. This is taken care of by the Knaster-Tarski theorem, which ensures the existence of fixed points of monotonic function as well as the existence of the least and greatest one. Note, that E must be a complete lattice rather than just an ordered set. The statement of the theorem and its proof are taken from [1].

Theorem 2.10 (Knaster-Tarski)**.** *Let $(E, \leq)$ be a complete lattice and $f\colon E \to E$ be a monotonic mapping. Then $\bigwedge \mathrm{Fix}(f)$ and $\bigvee \mathrm{Fix}(f)$ belong to $\mathrm{Fix}(f)$.*

Proof. We will show that

$$\bigwedge \mathrm{Fix}(f) = \bigwedge \{x \in E | f(x) \leq x\} \in \mathrm{Fix}(f)$$

and

$$\bigvee \mathrm{Fix}(f) = \bigvee \{x \in E | x \leq f(x)\} \in \mathrm{Fix}(f).$$

The proof for the second statement is dual to the first by the principle of symmetry, so we will only show the first part. Let $X = \{x \in E | f(x) \leq x\}$. We now show that $\bigwedge \mathrm{Fix}(f) = \bigwedge X \in \mathrm{Fix}(f)$. It holds that $\bigwedge f(X) \leq \bigwedge X$, because every lower bound for $f(X)$ is automatically a lower bound for X by definition of X.

By monotonicity of f, $f(X) \subseteq X$ and by Proposition 2.4, $\bigwedge X \leq \bigwedge f(X)$, so $\bigwedge X = \bigwedge f(X)$. By Proposition 2.7, $f(\bigwedge X) \leq \bigwedge f(X)$ and thus $\bigwedge X \in X$. Again by monotonicity of f it follows that $f(\bigwedge X) \in X$ and because $\bigwedge X$ is a lower bound for X, $\bigwedge X \leq f(\bigwedge X) \leq \bigwedge f(X) = \bigwedge(X)$, so $\bigwedge X \in \mathrm{Fix}(f)$.

Obviously (by definition of X) $\mathrm{Fix}(f) \subseteq X$ and so it follows that $\bigwedge X \leq \bigwedge \mathrm{Fix}(f)$. Since $\bigwedge X \in \mathrm{Fix}(f)$, we get $\bigwedge X = \bigwedge \mathrm{Fix}(f)$. □

The Knaster-Tarski theorem gives us a method, by which we can prove that a certain element is a least (or a greatest) fixed point of a monotonic function.

Corollary 2.11. *Let E be a complete lattice and $f\colon E \to E$ a monotonic function. Then $e \in E$ is the least fixed point of f if and only if it satisfies:*

1. *for each $x \in E$ it holds $f(x) \leq x \implies e \leq x$,*
2. *$f(e) \leq e$.*

Similarly, $e \in E$ is the greatest fixed point of f if and only if it satisfies:

1. *for each $x \in E$ it holds $x \leq f(x) \implies x \leq e$,*
2. *$e \leq f(e)$.*

Proof. If $e \in E$ is the least fixed point, then (1) follows from Knaster-Tarski theorem 2.10 and (2) is obvious. Conversely, if $e \in E$ satisfies (1) and (2), then e is a lower bound of the set $X = \{x \in E | f(x) \leq x\}$ and $e \in X$. Therefore $e = \bigwedge X$ and by Knaster-Tarski theorem 2.10 e is the least fixed point of f.

The part for the greatest fixed point follows from the principle of symmetry. □

2.4 Properties of fixed points and nested fixed points

Before we go further, we intoduce a notation for least and greatest fixed points. We borrow the idea from first-order logic, where we use quantifiers $\exists$ and $\forall$ to bind variables [1]. There we use notation $\exists x.expr[x]$, to mean the variable x is bound by the existential quantifier $\exists$ in some expression $expr$, that may involve variable x. To avoid name collisions, we can rename the variables in the following way:

$$(\exists x.expr[x]) \to (\exists y.expr[y]).$$

In the case of fixed points of function f we use the following notation:

- for least fixed point, we write $\mu x.f(x)$,
- for greatest fixed point, we write $\nu x.f(x)$.

[1] The same idea is used in λ-calculus abstraction, when we use the letter λ for arguments and for a function f with $f(x) = expr$, where $expr$ is some expression which may involve x, we write $f = \lambda x.expr$. This approach may be more familiar to someone with background from computer science. The process of renaming the variables is called an α-conversion. For further information see [3].

Again the variable x is bound by an extremal fixed point and we may use renaming of the variables in the following sense $\mu x.f(x) = \mu y.f(y)$, $\nu x.f(x) = \nu y.f(y)$. The two notions are dual by the principle of symmetry.

We can also have functions in multiple arguments. Suppose E is a complete lattice, F is an ordered set and $f\colon E \times F \to E$ is a monotonic function in its two arguments. For any $y \in F$ we define $f_y\colon E \to E$ by $f_y(x) = f(x, y)$. We denote $\mu x.f(x, y)$ to be the function from F to E defined by $\mu x.f(x, y) = \mu x.f_y(x)$ (similarly $\nu x.f(x, y)$). The μ and ν notation seems to be the standard way to represent the greatest and the least fixed points. The following is also adapted from [1].

Proposition 2.12. *Let E be a complete lattice and F an ordered set. If $f\colon E\times F \to E$ is monotonic in its two arguments, then $\mu x.f(x, y)$ and $\nu x.f(x, y)$ are monotonic functions from F to E.*

Proof. We shall prove this using the idea from the Knaster-Tarski theorem 2.10. Suppose $y \leq y'$. We need to show $\mu x.f(x, y) \leq \mu x.f(x, y')$. But by the proof of the Knaster-Tarski theorem, we know that $\mu x.f(x, y) = \bigwedge Y$, where $Y = \{x \mid f(x, y) \leq x\}$ and $\mu x.f(x, y') = \bigwedge Y'$ where $Y' = \{x \mid f(x, y') \leq x\}$.

Now suppose $x \in Y'$. Then $f(x, y) \leq x$. But by monotonicity of f, from $y \leq y'$ it follows that $f(x, y) \leq f(x, y')$. Therefore we have $f(x, y) \leq f(x, y') \leq x$ and $Y' \subseteq Y$. By Proposition 2.4, we deduce that $\bigwedge Y \leq \bigwedge Y'$ and so $\mu x.f(x, y) \leq \mu x.f(x, y')$. □

The following lemma is also called a golden lemma of μ-calculus, since it is used many times in proofs. It enables us to reduce nested fixed points and get a simpler expression.

Lemma 2.13 (Golden lemma of $\boldsymbol{\mu}$-calculus)**.** *Let E be a complete lattice and $f\colon E\times E \to E$ a monotonic function in both arguments. Then*

$$\mu x.\mu y.f(x, y) = \mu x.f(x, x) = \mu y.\mu x.f(x, y)$$

and

$$\nu x.\nu y.f(x, y) = \nu x.f(x, x) = \nu y.\nu x.f(x, y).$$

Proof. We give proof only for μ-case, the case for ν is similar by the principle of symmetry.

Let $f'(x) = \mu y.f(x, y)$ and by the definition of the least fixed point we have $\mu y.f(x, y) = f(x, f'(x))$. Let $a = \mu x.f'(x) = \mu x.\mu y.f(x, y)$ and $b = \mu x.f(x, x)$. Now we observe that, because a is a fixed point, $a = f'(a) = f(a, f'(a)) = f(a, a)$, so $b = \mu x.f(x, x) \leq a$. However $b = h(b, b)$, so $b \geq \mu y.f(b, y)$ and $b \geq \mu x.\mu y.f(x, y) = a$. Thus we have $a = b$ and we have proven the necessary equality. □

This property will be used many times to argue some simplifications of the algorithms later and to test the correctness of implementations.

3 Piecewise linear functions

Since we are dealing with piecewise linear functions, we take a closer look at their definition. The term itself suggests that we slice the domain of a function into pieces and on every piece we have a linear expression. The function need not be

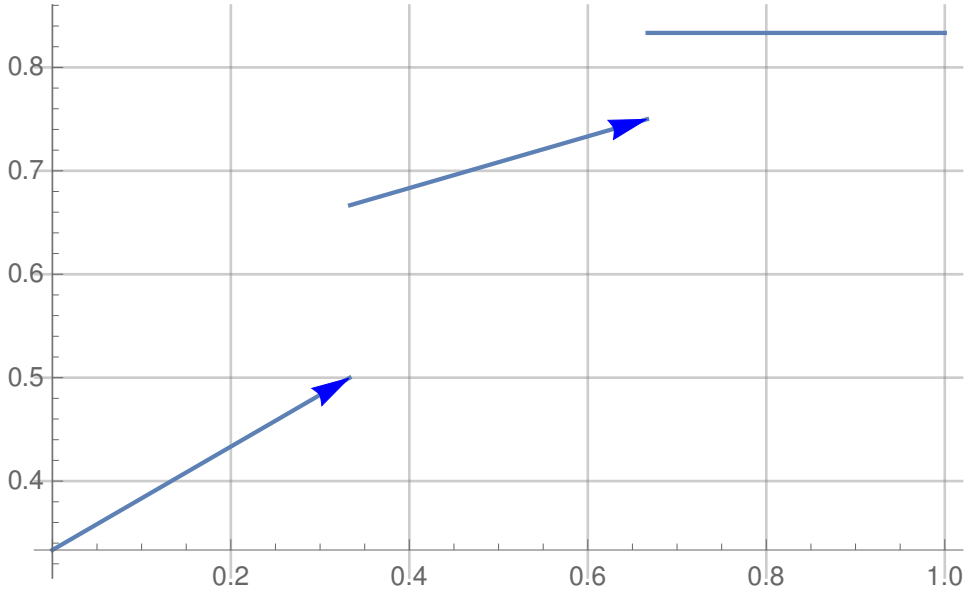


Figure 2: A discontinuous monotone piecewise linear function on $[0, 1]$. An open interval is indicated by an arrow for us to see, that the function takes a unique value at every point in the domain.

continuous as we can see in the one-dimensional example in Figure 2. The domain we are focusing on is $[0, 1]^n$. To elaborate the notion in detail we need the following definitions taken from [13].

3.1 Conditioned linear expressions

Definition 3.1. A *linear expression* in variables $x_1, x_2, \ldots, x_n$ is an expression

$$q_0 + q_1 x_1 + q_2 x_2 + \ldots + q_n x_n,$$

where $q_0, \ldots, q_n$ are real numbers. We say that a linear expression is *rational* if all $q_0, \ldots, q_n$ are rational numbers.

Since we will mostly be dealing with rational linear expressions, we will omit the word “rational" and only use the term linear expression. If however we will need real numbers this will be explicitly written.

We write $e(x_1, \ldots, x_n)$ to mean a linear expression in variables $x_1, \ldots, x_n$ and if we want to take real numbers $r_1, \ldots, r_n$ to replace the variables, we write $e(\vec{r})$. Linear expressions are closed under substitution, meaning that given linear expressions $e(x_1, \ldots, x_n), e_1(y_1, \ldots, y_m), \ldots, e_n(y_1, \ldots, y_m)$ we write $e(e_1, \ldots, e_n)$ for the substituted linear expression in variables $y_1, \ldots, y_m$, which is obtained by multiplying and adding coefficients.

Definition 3.2. A *conditioned linear expression* is a pair $C \vdash e$, where e is a linear expression and C is a finite set of strict and non-strict inequalities between linear expressions, i.e. each element in C has one of the forms

$$e_1 \leq e_2, \qquad e_1 < e_2. \tag{3.1}$$

We write $C(\vec{r})$ to mean the conjunction of inequalities, in which variables in C are replaced by real numbers $r_1, \ldots, r_n$. We use a conditioned linear expression $C \vdash e$ to express one piece of a piecewise linear function. The domain of the piece is the set of vectors $\vec{r}$ that satisfy $C(\vec{r})$ and the expression e sprecifies the linear function over that domain.

Proposition 3.3. *The domain $\{\vec{r} \mid C(\vec{r})\}$ is always convex, which means if $C(\vec{r})$ and $C(\vec{s})$, then for all $\lambda \in [0,1]$ it holds that $C(\lambda\vec{r} + (1-\lambda)\vec{s})$.*

The proof of that is not completed in [13]. To prove it, we need the following lemma.

Lemma 3.4. *If $A, B \subseteq \mathbb{R}^n$ are convex sets, then $A \cap B$ is a convex set.*

This is a standard easy fact about the convex sets (see [5]). Nevertheless, the proof is very short and thus included below.

Proof. Let $x, y \in A \cap B$, $\lambda \in [0,1]$. Since A is a convex set $\lambda x + (1-\lambda)y \in A$. Similarly $\lambda x + (1-\lambda)y \in B$. So we have $\lambda x + (1-\lambda)y \in A \cap B$ and $A \cap B$ is also a convex set. □

Now we can prove the proposition 3.3.

Proof. First, let us see the case, when there is only one inequality in C, say

$$q_0 + q_1 x_1 + \ldots q_n x_n \leq t_0 + t_1 x_1 + \ldots t_n x_n.$$

Suppose $C(\vec{r})$, $C(\vec{s})$ and $\lambda \in [0,1]$. Then if we substitute $\lambda\vec{r} + (1-\lambda)\vec{s}$ into first linear expression, we get

$$\begin{aligned} q_0 + q_1(\lambda r_1 + (1-\lambda)s_1) + \ldots + q_n(\lambda r_n + (1-\lambda)s_n) = \\ \lambda(q_0 + q_1 r_1 + \ldots q_n r_n) + (1-\lambda)(q_0 + q_1 s_1 + \ldots q_n s_n) \leq \\ \lambda(t_0 + t_1 r_1 + \ldots t_n r_n) + (1-\lambda)(t_0 + t_1 s_1 + \ldots t_n s_n) = \\ t_0 + t_1(\lambda r_1 + (1-\lambda)s_1) + \ldots + t_n(\lambda r_n + (1-\lambda)s_n). \end{aligned}$$

Thus we have a convex domain. The proof is the same for strict inequality.

The general case follows from Lemma 3.4, because when C has n constraints it specifies a finite intersection of the convex sets associated with the individual constraints. □

We will exploit this convexity in the algorithm later on. Note that, as stated in [13], the domain of a piece does not need to be open or closed and may be empty. Since it is given by a set of linear inequalities, we have a clear idea of the form of the domain. Every linear inequality gives us a half-space (a closed half-space from strict inequality and an open half-space from non-strict inequality). We therefore have a finite intersection of half-spaces, some closed and some open. If the inequalities were merely non-strict, we would get a closed convex polytope (see [7]). Our domains are more general since we allow a mix of both types of inequalities, but we can be sure, that the closure of the domain is in fact a closed convex polytope.

Definition 3.5. A function $f\colon [0,1]^n \to [0,1]$ is *piecewise linear* if there exists a finite set $\mathcal{F}$ of conditioned linear expressions in variables $x_1, \ldots, x_n$, such that the following conditions hold:

1. For all $\vec{r} \in [0,1]^n$, there exists a conditioned linear expression $(C \vdash e) \in \mathcal{F}$ such that $C(\vec{r})$ holds and

2. for all $\vec{r} \in [0,1]^n$ and every conditioned linear expression $(C \vdash e) \in \mathcal{F}$, if $C(\vec{r})$ holds, then $f(\vec{r}) = e(\vec{r})$.

We say, that such $\mathcal{F}$ *represents* f.

Note, that two conditional linear expressions $(C_1 \vdash e_1) \in \mathcal{F}$ and $(C_2 \vdash e_2) \in \mathcal{F}$ need not have disjoint domains, however e_1 and e_2 must agree on any overlap.

Example 3.6. *We again look at the one-dimensional example in Figure 2. The function is given by the following conditioned linear expression:*

$$\begin{aligned} 0 \leq x < \frac{1}{3} &\vdash \frac{1}{2}x + \frac{1}{3} \\ \frac{1}{3} \leq x < \frac{2}{3} &\vdash \frac{1}{4}x + \frac{7}{12} \\ \frac{2}{3} \leq x \leq 1 &\vdash \frac{5}{6}. \end{aligned}$$

It is discontinuous and has its domain sliced into three pieces.

3.2 First-order theory of linear arithmetic

Giving the function via conditioned linear expressions may not be optimal if the domain has many different pieces. We will provide an alternative way to represent some monotonic piecewise linear functions via μ-terms (see Section 5). In order to prove, that those terms are in fact piecewise linear functions, we will use the first-order theory of *linear arithmetic* as it is done in [13]. It has linear expressions as terms and strict and non-strict inequalities between linear expressions (see equation (9.1)) as atomic formulas. Equality can be expressed as a conjunction of two non-strict inequalities, i.e. $e_1 = e_2$ is expressed by $(e_1 \leq e_2) \wedge (e_2 \leq e_1)$. The negation of an atomic formula can also be expressed as an atomic formula: $\neg(e_1 \leq e_2)$ is equal to $(e_2 < e_1)$. The truth of a first-order formula is given by its interpretation in real numbers, but we can also restrict terms to rational linear expressions and get the theory of rational linear arithemetic. Both theories enjoy the property of *quantifier elimination*, i.e. every first-order formula has an equivalent version without quantifiers (see [14]). We will see some ideas from the proof of quantifier elimination in Section 4.3, where we will use them to optimise constraint sets.

Every formula in frst-order linear arithmetic can be put in *disjunctive normal form*. This means that it has an equaivalent formulation as a disjunction of conjunctions of atomic formulas. This is obtained by using logical equivalences such as distributivity laws and De Morgan's laws.

Example 3.7. *Suppose we have a formula* $(x_1 < x_2) \vee (x_2 < x_3) \implies (x_1 < x_2)$. *We first rewrite the implication using negation and disjunction and we get* $\neg((x_1 < x_2) \vee (x_2 < x_3)) \vee (x_1 < x_2)$. *Now pushing the negation inside, we get* $((x_2 \leq x_1) \wedge (x_3 \leq x_2)) \vee (x_1 < x_2)$, *which is in disjunctive normal form.*

Proposition 3.8. *A function* $f\colon [0,1]^n \to [0,1]$ *is piecewise linear if and only if its graph* $\{(\vec{x}, y) \in [0,1]^{n+1} | f(\vec{x}) = y\}$ *is definable by a fomula* $F(x_1, \ldots, x_n, y)$ *in the first-order theory of linear arithmetic.*

The proposition holds for rational theory of linear arithmetic as well as the version with reals. However the proof for both versions is basically the same, so we will not focus on that.

Proof. We apply quantifier elimination to obtain the result. Suppose we have f represented by k conditional linear expressions. For each conditional linear expression $C \vdash e$ we construct an implication $(\bigwedge C) \implies (y = e)$, so we have a conjunction of k such implications. We now only add range constraints for each variable $\bigwedge_{1\leq j\leq k}(0 \leq x_j) \wedge (x_j \leq 1)$ and range constraints for y to the conjunction and we have a formula for the graph of the function f.

Conversely, suppose $F(x_1, \ldots, x_n, y)$ defines a graph of f. By quantifier elimination, we can assume that F is quantifier free and in disjunctive normal form. Then F is a disjunction of conjunctions and each conjunction K can be rewritten in the form

$$(\bigwedge C) \wedge (\bigwedge_{1\leq i\leq h} y > a_i) \wedge (\bigwedge_{1\leq i\leq k} y \geq b_i) \wedge (\bigwedge_{1\leq i\leq l} y \leq c_i) \wedge (\bigwedge_{1\leq i\leq m} y < d_i), \tag{3.2}$$

such that the only variables in finite set of atomic formulas C and linear expressions a_i, b_i, c_i and d_i are $x_1, \ldots, x_n$. Since F represents the graph of the function f, for all reals $r_1, \ldots r_n$ there is at most one such s, that $K(\vec{r}, s)$ holds and if so, then $(\vec{r}, s) \in [0, 1]^{n+1}$. When we have such an s, it holds that

$$\max\{a_i(\vec{r}) | 1 \leq i \leq h\} < \max\{b_i(\vec{r}) | 1 \leq i \leq k\} = s,$$

and

$$s = \min\{c_i(\vec{r}) | 1 \leq i \leq l\} < \min\{d_i(\vec{r}) | 1 \leq i \leq m\}.$$

Therefore we obtain the conditioned linear expressions in the following way. For every conjunction K in F, rewritten in the form of equation (3.2), and for every $j \in \{1, 2, \ldots, k\}$ we include the conditioned linear expression

$$C, \{b_j > a_i\}_{1\leq i\leq h}, \{b_j \geq b_i\}_{1\leq i\leq k}, \{b_j \leq c_i\}_{1\leq j\leq l}, \{b_j < d_i\}_{1\leq i\leq m} \vdash b_j.$$

□

This dissertation concerns least and greatest fixed points of piecewise linear functions, we need to ask oursleves, what the result looks like, i.e. if $f\colon [0, 1]^{n+1} \to [0, 1]$ is a piecewise linear function, that is monotonic in the last variable x_{n+1}, the following proposition 3.9 will show, that $\mu x_{n+1}.f$ is also a piecewise linear function.

Proposition 3.9. *Let $f\colon [0, 1]^{n+1} \to [0, 1]$ be a piecewise linear function, that is monotonic in the last variable x_{n+1}, , i. e. if $t, s \in [0, 1]$ and $t \leq s$, then for all $x_1, \ldots, x_n \in [0, 1]$ it holds that $f(x_1, \ldots, x_n, t) \leq f(x_1, \ldots, x_n, s)$. Then $\mu x_{n+1}.f(x_1, \ldots, x_n, x_{n+1})$ and $\nu x_{n+1}.f(x_1, \ldots, x_n, x_{n+1})$ are piecewise linear functions from $[0, 1]^n$ to $[0, 1]$.*

We require monotonicity to be sure, that the least fixed point exists by the Knaster-Tarski theorem 2.10.

Proof. The proof for ν case is similar to the μ case by the principle of symmetry, so we focus only on the least fixed point. By definition of the least fixed point,

$\mu x_{n+1}.f(\vec{x})$ is a function from $[0,1]^n$ to $[0,1]$. By proposition 3.8 we need to show, that graph of $\mu x_{n+1}.f(x_1,\ldots,x_n,x_{n+1})$ is definable by a formula in first-order linear arithmetic. Let $F(x_1,\ldots,x_n,x_{n+1},y)$ be the formula for f. Then the appropriate formula for $\mu x_{n+1}.f(x_1,\ldots,x_n,x_{n+1})$ is

$$F(x_1,\ldots,x_n,y,y) \wedge \forall z.(F(x_1,\ldots,x_n,z,z) \implies y \leq z).$$

□

3.3 Local algorithm

If one is just interested in computing a piecewise linear function f, then all one needs to do is have an algorithm that, given a vector $\vec{r}$ as input, outputs the value $f(\vec{r})$. However, for certain manipulations of functions (such as computing fixed-point finding functions in Section 4), one needs more information. One possibility would be to explicitly carry around the full representation of the function as a finite set of conditioned linear expressions. However, this can be extremely large. Instead we will work with a less explicit form of representation, namely a local algorithm, that given a point in the domain, outputs an appropriate conditioned linear expression. This approach is new and thus a part of an original contribution.

Definition 3.10. A *local algorithm* for a piecewise linear function $f\colon [0,1]^n \to [0,1]$ is an algorithm, that for $r_1,\ldots r_n \in [0,1]$ outputs a conditioned linear expression $C \vdash e$ such that

1. $C(r_1,\ldots,r_n)$ holds and
2. if for any $s_1,\ldots s_n \in [0,1]$ $C(s_1,\ldots,s_n)$ holds, then $f(\vec{s}) = e(\vec{s})$.

Furthermore, only finitely many distinct $C \vdash e$ can be returned.

It is worth remarking, that an explicit representation is easily converted to a local algorithm, and a local algorithm trivially provides a way of mapping input vectors to output values. However, there are cases, where we have a far more efficient representation via local algorithms (see Section 5 on μ-terms).

In Section 4 we will describe how to compute the fixed-point-finding function $\mu x_{n+1}.f(\ldots)$, which is piecewise linear by Proposition 3.9, as a local algorithm. In order to do this, it is only required that the function f itself be given as a local algorithm. In this sense, the operation that maps an $n+1$-argument function f to the n-argument function $\mu x_{n+1}.f(\ldots)$ can be seen as a transducer of local algorithms. No further information about how the local algorithms carry out their computation is required. Thus local algorithms can be viewed quite abstractly as providing a sort of "black box", whose internals are hidden, whose only requirement is to map input vectors to appropriate conditioned linear expressions.

4 Iterative algorithm for finding fixed points

We now take a look at an iterative algorithm for finding least and greatest fixed points of monotone piecewise linear functions. Let $f\colon [0,1]^{n+1} \to [0,1]$ be a piecewise

linear function, that is monotone in the last variable. The function is not necessarily monotone in other variables, but if we want to find nested fixed points, i.e. fixed points od the piecewise linear function $\mu x_{n+1}.f$ (which is piecewise linear function according to the Proposition 3.9), we need monotonicity in other variables as well. Keeping in mind that the procedure for calculating greatest fixed point is dual to the procedure for the least fixed point by the principle of symmetry, we will focus only on the least fixed points.

One way of calculating the least fixed point would be to convert the function $\mu x_{n+1} f(x_1, \ldots, x_n, x_{n+1})$ to a formula in the first-order theory of linear arighmetic as in Propositions 3.8 and 3.9 and then apply quantifier elimination [6]. However constructing an efficient algorithm for quantifier elimination is known to be a hard problem, and in this case the entire procedure is very slow, since we have to iterate through all pieces of the domain. We present an alternative to that, where rather than computing the enitre set of conditioned linear expressions, the algorithm works locally and provides a single conditioned linear expression that applies to the given input vector $\vec{r}$. Since we are dealing with piecewise linear functions locally, we represent them by local algorithms as defined in 3.10.

We are of course dealing with rational piecewise linear functions and we are inputing rational numbers, since the real world computers can only precisely calculate rationals and not arbitrary reals. Because we have a working implementation of the algorithm, this restriction in in fact necessary. However it is convenient to consider the running of the algorithm in the general case, where $r_1, \ldots, r_n$ are arbitrary real numbers in $[0, 1]$. This can be understood as an algorithm in the *Real RAM* model of computation (see [10]). When we input rational numbers, all real numbers generated in the algorithm are rational themselves and all linear expressions generated are rational.

4.1 Motivating the algorithm on example

The main aim of this projet is to explore an iterative algorithm for computing fixed points of monotone piecewise linear functions. The algorithm was proposed in [13], where proved correct. We directly present a *modified version* of the algorithm, designed for improved efficiency and is an original result. The original version of the algorithm will then be discussed by describing how it differs from the modified version. In this section we will see the idea of the altgorithm on a one-dimensional example inspired by [13] and comment on its properties.

Fixed point are computed by iterating through their approximations. We start with 0 for the least fixed point and 1 for the greatest fixed point. To illustrate the main idea we take another look at the piecewise linear function from example 3.6. The function $f\colon [0, 1] \to [0, 1]$ is given by the following conditioned linear expression:

$$\begin{aligned} 0 \leq x < \frac{1}{3} &\vdash \frac{1}{2}x + \frac{1}{3} \\ \frac{1}{3} \leq x < \frac{2}{3} &\vdash \frac{1}{4}x + \frac{7}{12} \\ \frac{2}{3} \leq x \leq 1 &\vdash \frac{5}{6}. \end{aligned}$$

We can see the graph of the function f and of the diagonal on Figure 3. Clearly f is monotonic and it has a unique fixed point at $\frac{5}{6}$.

The algorithm calculates the fixed point by iteratively correcting an approximation d. We start with $d = 0$. Note, that simple iteration of f from 0 (like in Banach fixed point theorem [9]) $0 \leq f(0) \leq f(f(0)) \leq \ldots$ generally does not work, since the sequence may never reach a fixed point. In the Banach fixed point theorem this sequence may not reach a fixed point in finite time, but it is guaranteed to converge to one. In our setting, the limit of the sequence need not be a fixed point because of discontinuity of f. Instead we iterate through pieces.

The initial approximation is $d = 0$ and we retrieve the linear piece for $x = 0$ given by the conditioned linear expression $0 \leq x < \frac{1}{3} \vdash \frac{1}{2}x + \frac{1}{3}$. The unique solution of $x = \frac{1}{2}x + \frac{1}{3}$ is $x = \frac{2}{3}$, which is outside the domain for this piece. So we replace d by a new approximation, given by $\frac{1}{2}x + \frac{1}{3}$ calculated at the upper bound $x = \frac{1}{3}$ of the domain. Therefore the next approximation for the fixed point is $x = \frac{1}{2}$.

We again retrieve the linear piece for $x = \frac{1}{2}$, which is given by $\frac{1}{3} \leq x < \frac{2}{3} \vdash \frac{1}{4}x + \frac{7}{12}$. The solution to the equation $x = \frac{1}{4}x + \frac{7}{12}$ is $x = \frac{7}{9}$, which is again outside the domain of the linear piece. So we find the next approximation by calculating $\frac{1}{4}x + \frac{7}{12}$ at $x = \frac{2}{3}$ and we have $d = \frac{3}{4}$.

Finally we retrieve the last piece for $x = \frac{3}{4}$, which is given by $\frac{2}{3} \leq x \leq 1 \vdash \frac{5}{6}$. We get a candidate $x = \frac{5}{6}$, which in fact is in the domain of the current linear piece and therefore the least fixed point of f. Although on this example, we examine all pieces of the domain, that is not necessarily the case.

The general algorithm for finding fixed points of monotone piecewise linear functions with n arguments, where $n > 1$ is substantially more complicated, because we are calculating fixed points of linear expressions rather than just numbers.

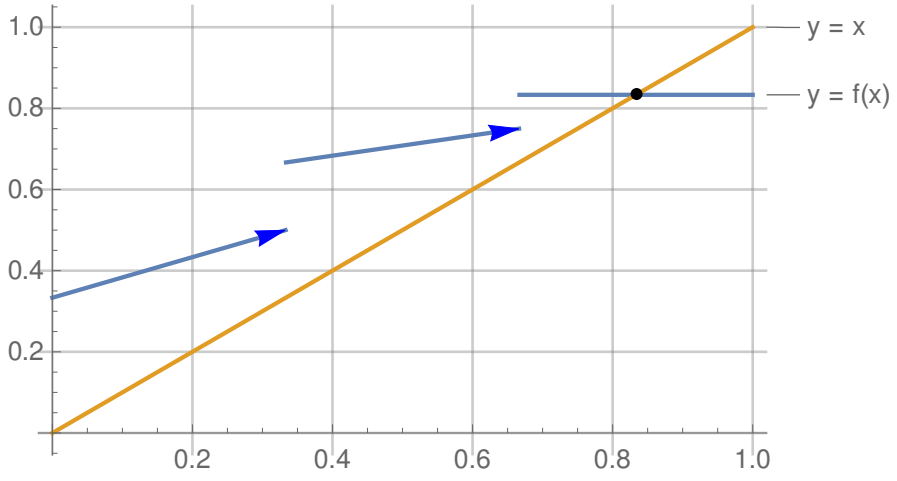


Figure 3: Graph of the function f and the diagonal.

4.2 The iterative algorithm

In this section we present the iterative algorithm directly in a modified version differing from the algorithm in [13] in certain critical aspects. The original algorithm

has some substantial drawbacks regarding space complexity, which to some degree resolved here and on which we comment in Section 4.2.3.

4.2.1 Modified algorithm

Input: A vector of real numbers $(r_1, \ldots, r_n) \in [0,1]^n$, a local algorithm representing a piecewise linear function $t\colon [0,1]^{n+1} \to [0,1]$, that is monotone in x_{n+1}.

Output: A conditioned linear expression $C \vdash e$ in variables $x_1, \ldots x_n$ with the following properties:

(P1) $C(r_1, \ldots, r_n)$ holds,

(P2) for all $s_1, \ldots, s_n \in \mathbb{R}$, if $C(\vec{s})$ holds, then $s_1, \ldots, s_n \in [0,1]$ and $e(\vec{s}) = \mu x_{n+1}.t(\vec{s}, x_{n+1})$.

Initializing values: $D = \emptyset$ (current set of inequalities between linear expressions); $d = 0$ (current linear expression);

loop

The loop invariants are:

(I1) $D(\vec{r})$ holds and

(I2) for all $\vec{s} \in [0,1]^n$, if $D(\vec{s})$, then $d(\vec{s}) \leq (\mu x_{n+1}.t)(\vec{s})$.

We think of D as constraints propagated from previous iterations, and of d as the current approximation of the fixed point subject to the constraints.

Using the local algorithm for t compute $t(x_1, \ldots, x_n, x_{n+1})$ at $(\vec{r}, d(\vec{r}))$ as $C \vdash e$, where e has the form $q_0 + q_1 x_1 + \ldots q_n x_n + q_{n+1} x_{n+1}$.

Arrange inequalities in C in the following way.

$$\begin{aligned} &C' \cup \{x_{n+1} > a_i\}_{1 \leq i \leq l'} \cup \{x_{n+1} \geq a_i\}_{l' \leq i \leq l} \\ &\cup \{x_{n+1} \leq b_i\}_{1 \leq i \leq m'} \cup \{x_{n+1} < b_i\}_{m' \leq i \leq m}, \end{aligned} \tag{4.1}$$

so the only variables in inequalities C' and linear expressions a_i and b_i are the variables $x_1, \ldots, x_n$.

if $q_{n+1} \neq 1$ **then**

define the linear expression

$$f := \frac{1}{1 - q_{n+1}}(q_0 + q_1 x_1 + \ldots q_n x_n) \tag{4.2}$$

if $C(\vec{r}, f(\vec{r}))$ **then**

$$\begin{aligned} E :=& D \cup C' \cup \{d \leq f\} \cup \{d > a_i\}_{1 \leq i \leq l'} \cup \{d \geq a_i\}_{l' \leq i \leq l} \\ &\cup \{f \leq b_i\}_{1 \leq i \leq m'} \cup \{f < b_i\}_{m' \leq i \leq m}, \end{aligned} \tag{4.3}$$

return $E \vdash f$

else

Define $N(x_1, \ldots, x_n)$ to be the negation of the inequality

$$e_1(x_1, \ldots, x_n, f(x_1, \ldots, x_n)) \triangleleft e_2(x_1, \ldots, x_n, f(x_1, \ldots, x_n)),$$

where $\triangleleft$ is used for either $<$ or $\leq$ and $e_1(\vec{x}, f(\vec{x})) \triangleleft e_2(\vec{x}, f(\vec{x}))$, is the chosen inequality in C for which $e_1(\vec{r}, f(\vec{r})) \triangleleft e_2(\vec{r}, f(\vec{r}))$ is false and go to **find next approximation** below.

end if

else(case that $q_{n+1} = 1$)

if $q_0 + q_1 r_1 + \ldots q_n r_n = 0$ **then**

return $D \cup C(x_1, \ldots, x_n, d(x_1, \ldots, x_n)) \cup \{q_0 + q_1 x_1 + \ldots q_n x_n = 0\} \vdash d$

else

Define $N(x_1, \ldots, x_n)$ to be the one of the inequalities

$$q_0 + q_1 r_1 + \ldots q_n r_n < 0, \qquad 0 < q_0 + q_1 r_1 + \ldots q_n r_n,$$

that is true for $\vec{r}$ and go to **find next approximation** below.

end if

end if

end loop

Find next approximation:

Consider inequalities in C arranged as in (4.1) and choose a j with $1 \leq j \leq m$ such that $b_j(\vec{r}) \leq b_i(\vec{r})$ for all $1 \leq i \leq m$ (we shall refer to b_j as a *supremum term*). If there is a candidate for a supremum term, which arises from strict inequality, choose b_j to be one such.

if b_j arises from a strict inequality of the form $x_{n+1} < b_j$ **then**

go back to the loop with

$$\begin{aligned} D :=& D \cup C' \cup \{N\} \cup \{d(x_1, \ldots, x_n) < b_j(x_1, \ldots, x_n)\} \cup \\ & \{b_j(x_1, \ldots, x_n) \leq b_i(x_1, \ldots, x_n)\}_{1 \leq i \leq m} \cup \\ & \{d(x_1, \ldots, x_n) > a_i(x_1, \ldots, x_n)\}_{1 \leq i \leq l'} \cup \\ & \{d(x_1, \ldots, x_n) \geq a_i(x_1, \ldots, x_n)\}_{l' \leq i \leq l} \\ d :=& e(x_1, \ldots, x_n, b_j(x_1, \ldots, x_n)) \end{aligned} \tag{4.4}$$

else

all candidates for supremum term have non-strict inequalities, therefore go back to the loop with

$$\begin{aligned} D :=& D \cup C' \cup \{N\} \cup \{d(x_1, \ldots, x_n) \leq b_j(x_1, \ldots, x_n)\} \cup \\ & \{b_j(x_1, \ldots, x_n) \leq b_i(x_1, \ldots, x_n)\}_{1 \leq i \leq m'} \cup \\ & \{b_j(x_1, \ldots, x_n) < b_i(x_1, \ldots, x_n)\}_{m' \leq i \leq m} \cup \\ & \{d(x_1, \ldots, x_n) > a_i(x_1, \ldots, x_n)\}_{1 \leq i \leq l'} \cup \\ & \{d(x_1, \ldots, x_n) \geq a_i(x_1, \ldots, x_n)\}_{l' \leq i \leq l} \\ d :=& e(x_1, \ldots, x_n, b_j(x_1, \ldots, x_n)) \end{aligned} \tag{4.5}$$

end if

If we have a local algorithm for a monotone piecewise linear function

$$t\colon [0,1]^{n+1} \to [0,1],$$

the iterative algorithm used on the local algorithm for t provides a local algorithm for the function $(\mu x_{n+1}.t)\colon [0,1]^n \to [0,1]$.

Let us illustrate running of the algorithm on an example.

Example 4.1. *Suppose we have a function* $t\colon [0,1]^2 \to [0,1]$ *given by a local algorithm* [2] *and we wold like to compute* $\mu x_2.t$ *at a point* $x_1 = 1$. *We start by setting* $D = \emptyset$ *and* $d = 0$. *By calling the local algorithm we obtain a conditioned linear expression* $C \vdash e$ *with* $C = \{0 \le x_1, x_1 \le 1, 0 \le x_2, x_2 \le \frac{1}{4}\}$ *and* $e = \frac{3}{8}x_1 + \frac{1}{8}$. *The coefficient* q_2 *of variable* x_2 *is* 0, *which is not equal to* 1, *so we proceed by defining the expression* $f := \frac{3}{8}x_1 + \frac{1}{8}$. *The condition* $C(1, f(1)) = C\left(1, \frac{1}{2}\right)$ *does not hold, since* $\frac{1}{2} \not\le \frac{1}{4}$, *and thus we define* $N(x_1) := \frac{3}{8}x_1 + \frac{1}{8} > \frac{1}{4}$ *as the negation of* $f(x_1) \le \frac{1}{4}$. *When we simplify, we get* $N(x_1) := x_1 > \frac{1}{3}$. *We proceed to the subroutine find next approximation and we find the supremum term* $b_j(x_1) := \frac{1}{4}$. *It is the only candidate and comes from a non-strict inequality, so we proceed to the second case and modify* D *as the union of the following sets of inequalities:*

- $\emptyset$ *from previous* D,
- $\{0 \le x_1 \le\}$ *from* C',
- $\{x_1 > \frac{1}{3}\}$ *from* N,
- $\{0 \le \frac{1}{4}\}$ *from* $d \le b$,
- $0 \le 0$ *from* $a_i \le d$.

When put together and simplified, the new D *equals to* $\{\frac{1}{3} < x_1 \le 1\}$. *We also modify* d *to be* $d(x_1) := e(x_1, b_j(x_1)) = \frac{3}{8}x_1 + \frac{1}{8}$. *We now repeat the loop and again call the local algorithm for* $x_1 = 1$ *and* $x_2 = d(1) = \frac{1}{2}$ *to obtain the conditioned linear expression* $C \vdash e$ *with* $C = \{0 \le x_1 \le 1, \frac{1}{4} \le x_2 \le 1\}$ *and* $e = \frac{3}{8}x_1 + \frac{1}{2}x_2$. *We observe that* $q_2 = \frac{1}{2} \ne 1$ *and we define* $f(x_1) := \frac{3}{4}x_1$. *We now check the condition* $C(1, f(1)) = C(1, \frac{3}{4})$, *which holds, and in order to return a result, we define* E *as a union of the following sets of inequalities:*

- $\{\frac{1}{3} < x_1 \le 1\}$ *from* D,
- $\{0 \le x_1 \le 1\}$ *from* C',
- $\{\frac{3}{8}x_1 + \frac{1}{8} \le \frac{3}{4}x_1\}$ *from* $d \le f$,

[2] This example is taken from [13, section 3.4] and the function t is given by a μ-term

$$\left(\frac{5}{8} \oplus \frac{3}{8}x_1\right) \odot \left(\frac{1}{2} \sqcup \left(\frac{3}{8} \oplus \frac{1}{2}x_2\right)\right).$$

For the definition of a μ-term see Section 5 below. In this example we only use the function via a local algorithm, but the details on how the pieces of this piecewise linear function are computed are described in Section 5 as well.

- $\{\frac{1}{4} \leq \frac{3}{8}x_1 + \frac{1}{8}\}$ *from* $d \geq a_i$,
- $\{\frac{3}{4}x_1 \leq 1\}$ *from* $f \leq b_i$.

When simplified, E *equals to* $\{\frac{1}{3} < x_1 \leq 1\}$ *and we return* $E \vdash f$ *as*

$$\{\frac{1}{3} < x_1 \leq 1\} \vdash \frac{3}{4}x_1,$$

which is the final result.

4.2.2 Correctness and termination of modified algorithm

Theorem 4.2. *Let* $t\colon [0,1]^{n+1} \to [0,1]$ *be a piecewise linear function, that is monotone in the last variable and represented by a local algorithm. Then for every vector* $(r_1, \ldots, r_n) \in [0,1]^n$ *of real numbers the above algorithm terminates with a conditioned linear expression* $C_{\vec{r}} \vdash e_{\vec{r}}$ *satisfying properties (P1) and (P2). Moreover the set of all possible resulting conditioned linear expressions*

$$\{C_{\vec{r}} \vdash e_{\vec{r}} | \vec{r} \in [0,1]^n\} \tag{4.6}$$

is finite and thus represents the piecewise linear function $\mu x_{n+1}.t\colon [0,1]^n \to [0,1]$.

The proof of the above theorem follows the correctness proof of the initial version of the algorithm in [13]. However since there are some nontrivial modifications to the algorithm, we will redo the proof entirely and show it in detail.

We introduce some useful terminology for the stated properties. For a monotone piecewise linear function t, we call the cardinality of the set (4.6) of possible results the *basis size*, and we call the maximum number of inequalities in any conditioned linear expression $C \vdash e$ the *condition size*.

Proof. First, we need to show that the loop invariants (I1) and (I2) guarantee, that any result satisfies (P1) and (P2). By definiton of the local algorithm, the computation at $(\vec{r}, d(\vec{r}))$ of $C \vdash e$ satisfies $C(\vec{r}, d(\vec{r}))$ and for all $s_1, \ldots, s_n, s_{n+1} \in \mathbb{R}^{n+1}$ if $C(s_1, \ldots, s_n, s_{n+1})$, then $(s_1, \ldots, s_n, s_{n+1}) \in [0,1]^{n+1}$ and $e(s_1, \ldots, s_n, s_{n+1}) = t(s_1, \ldots, s_n, s_{n+1})$.

In the case that $q_{n+1} \neq 1$, the linear expression f defined in (4.2), maps any $s_1, \ldots, s_n \in \mathbb{R}$ to the unique solution $f(\vec{s})$ to the equation $x_{n+1} = e(s_1, \ldots, s_n, x_{n+1})$ in $\mathbb{R}$. Suppose, that $D(\vec{s})$ holds. Then, by loop invariant (I2), $d(\vec{s}) \leq (\mu x_{n+1}.t)(\vec{s})$.

Let us consider the case, when $C(\vec{r}, f(\vec{r}))$ holds and let E be defined by (4.3). We now show, that any $\vec{s} \in [0,1]^n$, that satisfies constraints in E also satisfies the constraints in the set

$$O = D \cup C(x_1, \ldots, x_n, d(x_1, \ldots, x_n)) \cup C(x_1, \ldots, x_n, f(x_1, \ldots, x_n)).$$

Suppose, that $\vec{s}$ satisfies the constraints in E. We already know that $\vec{s}$ satisfies constraints in $D \cup C'$, where C' is defined by (4.1) (because those constaints appear in E as well as in O). Now we take a look at the rest of constraints in C. We have rearranged constraints as in equation (4.1) and now we consider each constraint individually. If the inequality is of the form $x_{n+1} \leq b_i$ then (by construction of E) $\vec{s}$

satisfies $f(\vec{s}) \leq b_i(\vec{s})$. Because $d(\vec{s}) \leq f(\vec{s})$ (a constraint explicitly added to E), we get $d(\vec{s}) \leq b_i(\vec{s})$. A similar argument applies to the other types of inequalities and we have that $\vec{s}$ satisfies both $C(\vec{s}, d(\vec{s}))$ and $C(\vec{s}, f(\vec{s}))$, so $\vec{s}$ satisfies the inequalities in O.

Now suppose $E(\vec{s})$ holds. Because this implies $C(\vec{s}, f(\vec{s}))$, we have $t(\vec{s}, f(\vec{s})) = e(\vec{s}, f(\vec{s})) = f(\vec{s})$ as given by the local algorithm, i.e. $f(\vec{s})$ is a fixed point of the function $x_{n+1} \mapsto t(\vec{s}, x_{n+1})$. Since μ denotes the least fixed point of a function, it holds that $(\mu x_{n+1}.t)(\vec{s}) \leq f(\vec{s})$. Now because both $C(\vec{s}, d(\vec{s}))$ and $C(\vec{s}, f(\vec{s}))$ hold and $d(\vec{s}) \leq (\mu x_{n+1}.t)(\vec{s}) \leq f(\vec{s})$, we have, by convexity of constraints as proven in proposition 3.3, that $t(\vec{s}, s_{n+1}) = e(\vec{s}, s_{n+1})$ for all $s_{n+1} \in [d(\vec{s}), f(\vec{s})]$. So $f(\vec{s})$ is the unique fixed point of $x_{n+1} \mapsto t(\vec{s}, x_{n+1})$ on $[d(\vec{s}), f(\vec{s})]$. Since $d(\vec{s}) \leq (\mu x_{n+1}.t)(\vec{s})$, we have $f(\vec{s}) = (\mu x_{n+1}.t)(\vec{s})$. So the conditioned linear expression $E \vdash f$ satisfies (P2). It satisfies (P1), because we return the result if $C(\vec{r}, f(\vec{r}))$ and we already know by definiton of the local algorithm, that $C(\vec{r}, d(\vec{r}))$ holds and by (I1) that $D(\vec{r})$ holds. We only have to make sure, that $d(\vec{r}) \leq f(\vec{r})$, which follows from the fact that $C(\vec{r}, f(\vec{r}))$ and so $d(\vec{r}) \leq (\mu x_{n+1}.t)(\vec{r}) = f(\vec{r})$.

In the case that $q_{n+1} = 1$ then, for any $s_1, \ldots, s_n \in \mathbb{R}$ the equation $x_{n+1} = e(s_1, \ldots, s_n, x_{n+1})$ has a solution if and only if $q_0 + q_1 x_1 + \ldots q_n x_n = 0$. Then any x_{n+1} is a solution. Suppose that $q_0 + q_1 s_1 + \ldots q_n s_n = 0$ and $C(\vec{s}, d(\vec{s}))$ hold. Then we have $t(\vec{s}, d(\vec{s})) = e(\vec{s}, d(\vec{s}))$ and $d(\vec{s})$ is a fixed point of $x_{n+1} \mapsto t(\vec{s}, x_{n+1})$. Suppose $D(\vec{s})$ also holds. Then by loop invariant (I2) we have $d(\vec{s}) = (\mu x_{n+1}.t)(\vec{s})$ thus justifying that the returned conditioned linear expression satisfies property (P2). It satisfies property (P1) if $q_0 + q_1 x_1 + \ldots q_n x_n = 0$, which is the condition under which the result is returned.

Next we have to show, that the loop invariants (I1) and (I2) are preserved. They are trivially true for initial values $D = \emptyset$ and $d = 0$. Now we must show that they are preserved, when they are modified in the subroutine **find next approximation**. Let D' be the modified D. We first consider inequalities rearranged as in (4.1). Because $C(\vec{r}, d(\vec{r}))$, we must have $m \geq 1$, otherwise $C(\vec{r}, s)$ would hold for all real $s \geq d(\vec{r})$, contradicting that $C(\vec{r}, s)$ implies $s \in [0, 1]$ (similarly $l \geq 1$). So there exists j with $1 \leq j \leq m$ such that $b_j(\vec{r}) \leq b_i(\vec{r})$ for all $1 \leq i \leq m$.

We now show, that the constraints in D' are true for $\vec{r}$. We consider two cases. If we modify D as in (4.4), the chosen b_j arose from a strict inequality $x_{n+1} < b_j$. The constraints in $D \cup C' \cup N$ are trivially satisfied by $\vec{r}$. Since $C(\vec{r}, d(\vec{r}))$, we have $d(\vec{r}) < b_j(\vec{r})$, $d(\vec{r}) > a_i(\vec{r})$ for $1 \leq i \leq l'$ and $d(\vec{r}) \geq a_i(\vec{r})$ for $l' \leq i \leq l$. By definition of the supremum term, we have $b_j(\vec{r}) \leq b_i(\vec{r})$ for $1 \leq i \leq m$. In the second case, if we modify D by (4.5) and b_j comes from a non-strict inequality $x_{n+1} \leq b_j$, we can again argue that $D \cup C' \cup \{N\}$ are trivially satisfied. From $C(\vec{r}, d(\vec{r}))$, we have $d(\vec{r}) \leq b_j(\vec{r})$, $d(\vec{r}) > a_i(\vec{r})$ for $1 \leq i \leq l'$ and $d(\vec{r}) \geq a_i(\vec{r})$ for $l' \leq i \leq l$. By definition of the supremum term, we have $b_j(\vec{r}) \leq b_i(\vec{r})$ for all $1 \leq i \leq m$, thus also for $1 \leq i \leq m'$. For $m' \leq i \leq m$ we have strict inequalities, which are are in fact true for $\vec{r}$, otherwise suppose there would be such i with $m' \leq i \leq m$, that $b_j(\vec{r}) = b_i(\vec{r})$. Since b_j is a supremum term, b_i would also be a candidate for a supremum term. Because b_i comes from a strict inequality this contradicts our choice of b_j. Therefore all strict inequalities are true at $\vec{r}$ and (I1) is preserved.

To show (I2) we again consider two cases. But because both are very similar,

we will look at them separately only when needed. Suppose we modify D as in (4.4) or (4.5) and $\vec{s}$ satisfies the constraints in D'. We define $r' = (\mu x_{n+1} t)(\vec{s})$. By the invariant (I2) for D and d we have $d(\vec{s}) \leq r'$. We want to show that $e(\vec{s}, b_j(\vec{s})) \leq r'$. By the choice of N we know that $N(\vec{s})$ implies that $C(\vec{s}, r')$ does not hold. Now we show that $D'(\vec{s})$ and $s \in \{d(\vec{s})\} \cup (d(\vec{s}), b_j(\vec{s})) \cap [0,1]$ implies $C(\vec{s}, s)$. If we show $C(\vec{s}, d(\vec{s}))$ holds, then by the choice of j, $C(\vec{s}, s)$ also holds for $s \in \{d(\vec{s})\} \cup (d(\vec{s}), b_j(\vec{s})) \cap [0,1]$. To be sure of this implication let us take a look at one type of inequalities in C, say $x_{n+1} \geq a_i$. Then we have $s > d(\vec{s}) \geq a_i(\vec{s})$, so $s \geq a_i(\vec{s})$. A similar argument proves the implication for the other types of inequalities in C.

Now we have to show, that $C(\vec{s}, d(\vec{s}))$ holds. Trivially all inequalities in C' are satisfied. Now let us take a look at each type of inequality from (4.1):

- If the inequality is of the form $x_{n+1} \geq a_i$, we have $d(\vec{s}) \geq a_i(\vec{s})$.
- If the inequality is of the form $x_{n+1} > a_i$, we have $d(\vec{s}) > a_i(\vec{s})$.
- If the inequality is of the form $x_{n+1} \leq b_i$, we have $d(\vec{s}) \leq b_j(\vec{s}) \leq b_i(\vec{s})$. In case D is modified as in (4.4), the condition is $d(\vec{s}) < b_j(\vec{s})$, but all we need is a non-strict inequality.
- If the inequality is of the form $x_{n+1} < b_i$, we have have to consider two different cases:
  - If D is modified as in (4.4), we have $d(\vec{s}) < b_j(\vec{s}) \leq b_i(\vec{s})$, so $d(\vec{s}) < b_i(\vec{s})$.
  - If D is modified as in (4.5), we have $d(\vec{s}) \leq b_j(\vec{s}) < b_i(\vec{s})$, so $d(\vec{s}) < b_i(\vec{s})$.

Because $C(\vec{s}, r')$ is false and the fact that $d(\vec{s}) \leq r'$, we have $s < r'$ for every $s \in [0,1]$ with $s = d(\vec{s})$ or $d(\vec{s}) < s < b_j(\vec{s})$. Since r' is the least fixed point for $x_{n+1} \mapsto t(\vec{s}, x_{n+1})$, it is the least such x, that $t(\vec{s}, x) \leq x$ and because t is a monotonic function, we have $s < t(\vec{s}, s) \leq t(\vec{s}, r') = r'$. By the properties of the local algorithm for t, this means

$$s < e(\vec{s}, s) \leq r'. \tag{4.7}$$

So we have, using the continuity of e,

$$e(\vec{s}, b_j(\vec{s})) = \sup\{e(\vec{s}, s) | s = d(\vec{s}) \text{ or } d(\vec{s}) < s < b_j(\vec{s})\} \leq r'.$$

So in fact this is a good approximation of the fixed point. That is $d(\vec{s}) < e(\vec{s}, b_j(\vec{s}))$ and not $C(\vec{s}, e(\vec{s}, b_j(\vec{s})))$. The first holds by equation (4.7) if we substitute $d(\vec{s})$ for s (which we can, since it is in the domain for s) and by the fact that $d(\vec{s}) \leq b_j(\vec{s})$ and e represents a monotonic function in the last argument. The second is because if $C(\vec{s}, e(\vec{s}, b_j(\vec{s})))$, then $e(\vec{s}, b_j(\vec{s})) \leq b_j(\vec{s})$, but we know $e(\vec{s}, b_j(\vec{s})) \geq b_j(\vec{s})$ (if we substitute $b_j(\vec{s})$ for s in equation (4.7) using the definition of $e(\vec{s}, b_j(\vec{s}))$ via supremum), so $e(\vec{s}, b_j(\vec{s})) = b_j(\vec{s}) = r'$ (because r' is the fixed point), which contradicts not $C(\vec{s}, r')$.

To show termination, we first recall that piecewise linear functions are given by finite sets of conditioned linear expressions, so local algorithm for t can produce only finitely many results, say

$$C_1 \vdash e_1 \quad \ldots \quad C_{k'} \vdash e_{k'}, \tag{4.8}$$

where k' is the basis size of the local algorithm of t. We analyse the execution of the iterative algorithm on input vector $\vec{r} \in [0,1]^n$. On iteration number i, we enter the loop with D_i and d_i (where $D_1 = \emptyset$ and $d_1 = 0$), after which the local algorithm for t gives us one of the conditioned linear expressions $C_{k_i} \vdash e_{k_i}$ from (4.8) above, such that $C_{k_i}(\vec{r}, d_i(\vec{r}))$ holds. Then depending on conditions involving only $C_{k_i} \vdash e_{k_i}$ and $\vec{r}$ we either return a result or continue with subroutine find next approximation to construct D_{i+1} and d_{i+1}. By the previously shown inequality $d_i(\vec{s}) < e(\vec{s}, b_j(\vec{s})) = d_{i+1}(\vec{s})$ for $\vec{s}$ that satisfies the constraints in D_{i+1}, at iteration $i+1$, we have $d_{i+1}(\vec{r}) > d_i(\vec{r})$ and $C_{k_i}(\vec{r}, d_{i+1}(\vec{r}))$ is false. Since each conditioning set is convex, it follows that no C_j can occur twice in the list $C_{k_1}, C_{k_2}, \ldots$, so the algorithm must exit the loop after at most k' iterations and the computation terminates.

We now show, that the algorithm for finding the least fixed point produces only finitely many conditioned linear expressions $C_{\vec{r}} \vdash e_{\vec{r}}$.

We analyse the control flow in the algorihtm for $\mu x_{n+1}.t$ on given input vector $\vec{r} = (r_1, \ldots, r_n)$. On iteration i we enter the loop with D_i and d_i and we compute t and to get $C_{k_i} \vdash e_{k_i}$ from (4.8). Suppose $|C_{k_i}| = u$ and $|D_i| = v$. If the loop exits with $E \vdash f$, we have $u + v + 1$ inequalities. If the loop exits in the case $q_{n+1} = 1$, we have $u + v + 2$ inequalities (we count $=$ as two inequalities $\leq$ and $\geq$). Otherwise we repeat the loop and D_{i+1} has at most $v + 1 + 1 + u = u + v + 2$ inequalities (from D, C', N, $d \leq b_j$ and one for every b_i and a_i). If l' is the maximum number of inequalities in any C_j from (4.8) (condition size of t), the algorithm for $\mu x_{n+1}.t$, which runs for at most k' iterations results $C_{\vec{r}}$ with at most $k'(l'+2)$ inequalities: we start v with 0 and in every iteration we increase it by at most $l'+2$ inequalities.

To bind the number of results $C_{\vec{r}} \vdash e_{\vec{r}}$, we count all possible control flows of the algorithm. At iteration i, the algorithm uses $C_{k_i} \vdash e_{k_i}$ from (4.8) and it might terminate in two possible ways or it may reenter the loop in iteration $i+1$ with D_{i+1}, which arises from either (4.4) or (4.5) in a way, that depends on the choice of N and b_j. If $n = 0$, we only have inequalities between numbers, and they can be discarded. If $n \geq 1$, there are at least 2 inequalities in C, giving range constraints for x_1, so there are at most l' choices for N ($l'-2$ in case $q_{n+1} \neq 1$ and 2 choices in the case that $q_{n+1} = 1$). There are at most $l' - 2n - 1$ choices for b_j, which can be estimated by $l' - 1$. Therefore the execution of the algorithm is defined by the sequence

$$k_1, u_1, k_2, u_2, \ldots, k_m, u_m, v$$

where $m \leq k'$ is the number of iterations, u_i represents the choice of N and b_j with $1 \leq u_i \leq l' + (l'-1)$ and v is 1 or 2, according to the case in which the result is returned. Since each k_i is a distinct number, we can bind the number of such sequences by

$$2 \sum_{m=1}^{k'} \frac{k'!}{(k'-m)!} (l'(l'-1))^{m-1} \leq (k'(l')^2)^{k'}.$$

Therefore the number of different results is indeed finite. □

4.2.3 Comparison to the original version of the algorithm from [13].

As we have previously mentioned, the above version of the algorithm is an improvement of the original one, which differs in two places, where we construct the

constraint sets:

(D1) In the original version of the algorithm in [13], the constraint set E contains the inequalities

$$D \cup C(x_1, \ldots, x_n, d(x_1, \ldots, x_n)) \cup C(x_1, \ldots, x_n, f(x_1, \ldots, x_n)).$$

This produces a constraint set of size $|D| + 2|C|$ instead of one of size $|D| + |C| + 1$ in our modified version of the algorithm.

(D2) In the subroutine **find next approximation**, the choice of the supremum term b_j does not involve the condition on strict inequalities. Any b, that satisfies $b(\vec{r}) \leq b_i(\vec{r})$ for all $1 \leq i \leq m$ can be chosen for b_j. Therefore we only have one case and the new set of constraints D contains the following inequalities:

$$D \cup C(x_1, \ldots, x_n, d(x_1, \ldots, x_n)) \cup \{N(x_1, \ldots, x_n)\} \cup \{b_j \leq b_i | 1 \leq i \leq m\}.$$

Recall that m is the number of constraints, that restrict x_{n+1} from above (the constraints that contain the linear expressions b_i). The original version of the algorithm thus produces a constraint set of size $|D| + |C| + m + 1$ instead of one of size $|D| + |C| + 2$ in our modified version of the algorithm.

Because some of these constraints are redundant, there is a substantial slowdown of the process, that is in fact unnecessary. In the proof of correctness of the algorithm, we have shown, that any vector $\vec{s} \in [0, 1]^n$, that satisfies the constraints in E, also satisfies the constraints in $C(x_1, \ldots, x_n, d(x_1, \ldots, x_n)) \cup C(x_1, \ldots, x_n, f(x_1, \ldots, x_n))$, but in E we only have half as many constraints (and an additional one, that guarantees $d(x_1, \ldots, x_n) \leq f(x_1, \ldots, x_n)$). In the case (D2), every constraint in C of the form $x_{n+1} \triangleleft b_i$, where $\triangleleft$ is $<$ or $\leq$, appears in some form twice, but in the modified version there is only one constraint that substitutes it, again substantially lowering the number of constraints.

In Section 6 the impact of the optimisations in the modified version of the algorithm will be experimentally investigated.

4.3 Quantifier elimination optimisation

As we have seen, the modified version of the fixed-point algorithm builds smaller constraint sets than the original algorithm, eliminating some redundancy in constraints from the former. Nevertheless, it is still possible in principle that the constraint sets constructed may contain further redundancy. In this section, we show that, in principle, it is possible to further augment the algorithm with a procedure to remove all redundancy from constraint sets.

Recall, that a finite set od linear inequalities represents a domain, whose closure is a closed convex polytope. It is known that for a full-dimensional convex polytope, the minimal description with halfspaces is unique and is given by the set of the facet-defining halfspaces (see [7]). Since our domain is not a closed convex polytope, we cannot guarantee uniqueness of the minimal set of inequalities. However the

idea, that we only really need to focus on facets, leads us to believe, we may be able to shrink the sets of constraints even further. This augmentation is an original idea and a non-trivial conrtibution. However, the principle of quantifier elimination is well-known, but has not been previously used in the context of improving the conditioned linear expressions.

The main goal of this optimisation is to eliminate any unnecessary inequalities from the constraint set C of a conditioned linear expression $C \vdash e$ and so obtaining the minimal possible set of of constraints (minimal in the sense that it contains no redundant constraints) that represents the same domain. We introduce the terminology for these properties.

Definition 4.3. Let $C(x_1, \ldots, x_n)$ be a set of linear constraints from a conditioned linear expression.

- We say a constraint $c \in C$ is *redundant* for the set of constraints C, if for all $\vec{s} \in \mathbb{R}^n$ it holds that
$$(C \setminus c)(\vec{s}) \implies c(\vec{s}).$$
- A set of linear constraints $E(x_1, \ldots, x_n)$ is said to be *equivalent* to C, if for all $\vec{s} \in \mathbb{R}^n$, $C(\vec{s})$ holds if and only if $E(\vec{s})$ holds.

To tackle the task of removing redundant constraints we use quantifier elimination. We will only explain how the quantifier elimination algorithm works in the special case of a formula given by a sequence of existential quantifiers over a conjunction of linear inequalities - which is the only case we need.

For a set of constraints $C(x_1, \ldots, x_n)$, we construct an equivalent reduced set of constraints $E(x_1, \ldots, x_n)$ by adding constraints one-by-one and checking their necessity. Suppose at some point we have $E = \{e_1, e_2, \ldots, e_N\}$ and we would like to add a constraint $c \in C$ to the set E. The constraint is redundant, if adding its negation to the set gives an unfeasible (meaning empty) domain, i.e. if the formula

$$\exists x_1 . \exists x_2 . \ldots . \exists x_n . \left(\bigwedge_{1 \leq i \leq N} e_i(x_1, \ldots, x_n) \right) \wedge (\neg c(x_1, \ldots, x_n))$$

is false. Since this is a closed formula in the theory of linear arithmetic, we can apply quantifier elimination to compute its logical value. However, if c is necessary and we add c to the existing minimal set E, some of the constraints $e \in E$ may become redundant, as they may follow from the conjunction of c and the other constraints in E. If we want the set E to still be minimal, we need to check for all inequalities $e \in E$, if they are redundant for the set $E \cup \{c\}$.

We use the following procedure.

Input: A set of constraints $C(x_1, \ldots, x_n)$.
Output: A reduced set of constraints $E(x_1, \ldots, x_n)$.

$E := \emptyset$
for all $c(x_1, \ldots, x_n) \in C(x_1, \ldots, x_n)$ **do**

Using quantifier elimination, test if the formula

$$\exists x_1.\exists x_2.\ldots.\exists x_n.\left(\bigwedge_{e\in E} e(x_1,\ldots,x_n)\right)\wedge(\neg c(x_1,\ldots,x_n))$$

is true.
if the formula is false **then**
continue the for loop without adding c to E.
else
$U=\emptyset$ (set of redundant constraints)
for $e\in E$ **do**
Using quantifier elimination, test if the formula

$$\exists x_1.\exists x_2.\ldots.\exists x_n.\left(\bigwedge_{d\in E\setminus(U\cup\{e\})} d(\vec{x})\right)\wedge(c(\vec{x}))\wedge(\neg e(\vec{x}))$$

is true.
if the formula is false **then**
$U:=U\cup\{e\}$
end if
end for
$E:=(E\setminus U)\cup\{c\}$
end if
end for

Theorem 4.4. *Given a set of constraints $C(x_1,\ldots,x_n)$, the above procedure yields a subset of constraints $E(x_1,\ldots,x_n)\subseteq C(x_1,\ldots,x_n)$, with the following properties:*

1. *The constraint sets C and E are equivalent, i.e. for every $\vec{s}\in\mathbb{R}^n$, $C(\vec{s})$ holds if and only if $E(\vec{s})$ holds.*
2. *Every constraint $c\in E$ is necessary in E, i.e. there exists $\vec{s}\in\mathbb{R}^n$ such that $(E\setminus c)(\vec{s})$ holds, but $c(\vec{s})$ does not. Equivalently there are no redundant constraints in E.*

Proof. Since all the for loops go through finite sets, the procedure in fact terminates. We also observe that at every point, where we add constraints to E, they originate from C, we have $E\subseteq C$. We now only need to prove the two properties.

To prove the equivalence of the two sets, we need to consider both implications. Since $E\subseteq C$, every $\vec{s}\in\mathbb{R}^n$ that satisfies $C(\vec{s})$, automatically satisfies $E(\vec{s})$, because all the constraints in E are included in C and thus satisfied. So the first implication is trivial.

To prove the other implication it is useful to state the following property of the procedure: Let

$$E_1,E_2,E_3,\ldots,E_{|C|}=E$$

be the sets E after every iteration of the outer for loop. If for $\vec{s}\in\mathbb{R}^n$, $E_{|C|}$ holds, then $E_{|C|-1}(\vec{s}),\ldots,E_1(\vec{s})$ all hold. Suppose for some i with $1\leq i\leq|C|$, $E_{i+1}(\vec{s})$ holds. We want to prove, that $E_i(\vec{s})$ holds. If $E_i=E_{i+1}$ (in case we never enter the

second for loop), we have nothing to prove. Otherwise, since $E_{i+1} = (E_i \setminus U_i) \cup \{c_i\}$, we only need to check that $U_i(\vec{s})$ holds. Let $U_{i,1}, \ldots, U_{i,m} = U_i$ be the sets U when the inner for loop iterates at the i-th iteration of the outer loop, where $m = |E_i|$. Let j be the smallest index such that $U_{i,j} = U_i$ (thus the last constraint e_j is added to U_i at stage j). Since the formula

$$\exists x_1 . \exists x_2 . \ldots \exists x_n . \left(\bigwedge_{d \in E_i \setminus (U_{i,j-1} \cup \{e_j\})} d(\vec{x}) \right) \wedge (c_i(\vec{x})) \wedge (\neg e_j(\vec{x}))$$

is false, we can substitute $\vec{s}$ for $\vec{x}$ and we get false formula

$$\left(\bigwedge_{d \in E_i \setminus (U_{i,j-1} \cup \{e_j\})} d(\vec{s}) \right) \wedge (c_i(\vec{s})) \wedge (\neg e_j(\vec{s})).$$

Because $E_i \setminus (U_{i,j-1} \cup \{e_j\}) = E_{i+1} \setminus \{c_i\}$, $E_{i+1}(\vec{s})$ holds and $c_i \in E_{i+1}$, the only inequality in the conjunction, that can be false is $\neg e_j(\vec{s})$. Therefore $e_j(\vec{s})$ is true. The overall argument proceeds by taking the index j down from m to 0, at each stage establishing that $\vec{s}$ satisfies $E_i \setminus U_{i,j}$. At the end $U_{i,0}$ is $\emptyset$, so $\vec{s}$ satisfies E_i.

Now suppose there exists a vector $\vec{s} \in \mathbb{R}^n$, such that $E(\vec{s})$ holds, but $C(\vec{s})$ does not. So there exists a constraint $c \in C \setminus E$, such that $c(\vec{s})$ is false. Let us look the point in the algorithm, say iteration i, where the outer for loop considers the constraint c. The algorithm first checks whether c is redundant for the set of constraints $E_{i-1} \cup \{c\}$. By previously shown, since $E(\vec{s})$ holds, so do $E_i(\vec{s})$ and $E_{i-1}(\vec{s})$ and because $c(\vec{s})$ is not true, c is not redundant for the set $E_{i-1} \cup \{c\}$. Therefore E_i contains the constraint c. Since $c \notin E$, there is a point, say at iteration j, at which $c \in U_j$ and so

$$\exists x_1 . \exists x_2 . \ldots \exists x_n . \left(\bigwedge_{d \in E_j \setminus (U \cup \{c\})} d(\vec{x}) \right) \wedge (c_j(\vec{x})) \wedge (\neg c(\vec{x}))$$

is false. But if we substitute $\vec{s}$ for $\vec{x}$, the formula

$$\left(\bigwedge_{d \in E_j \setminus (U \cup \{c\})} d(\vec{s}) \right) \wedge (c_j(\vec{s})) \wedge (\neg c(\vec{s}))$$

is true, because $c(\vec{s})$ is false, but $E_j(\vec{s})$ and $E_{j+1}(\vec{s})$ (and subsequently $c_j(\vec{s})$) hold. So we have reached a contradiction and we have thus proven, that C and E are equivalent.

For the second property, suppose that there exists a constraint $c \in E$, that is redundant. We consider the point in the algorithm, say at iteration i of the outer loop, at which the constraint became such. If this is when the constraint is first added to E_i, we check, that it is indeed necessary for the set E_{i-1} from previous iteration. So the only way for it to become redundant is when another constraint $c_i \in C$ is added to the set E_i. Before we add c_i to E_{i-1}, we check whether c is redundant for the set $(E_{i-1} \setminus U) \cup \{c_i\}$ (where U = $U_{i,k}$ for some $1 \leq k \leq |E_{i-1}|$)

and since $E_i \subseteq (E_{i-1} \setminus U) \cup \{d\}$, c cannot be redundant for E_i and we reach a contradiction. It is useful to observe, that this property is the invariant of the outer for loop: After every iteration, the current set of constraints E does not contain any redundant constraints. □

The result, that the procedure returns, depends on the order in which we iterate through the set of constraints. To illustrate this we look at the following example.

Example 4.5. *Let us consider a two-dimensional domain*

$$([0,1] \times [0,1]) \setminus \{(0,0),(1,0),(0,1),(1,1)\},$$

which is a unit square without its vertices (see figure 4). We can describe this area

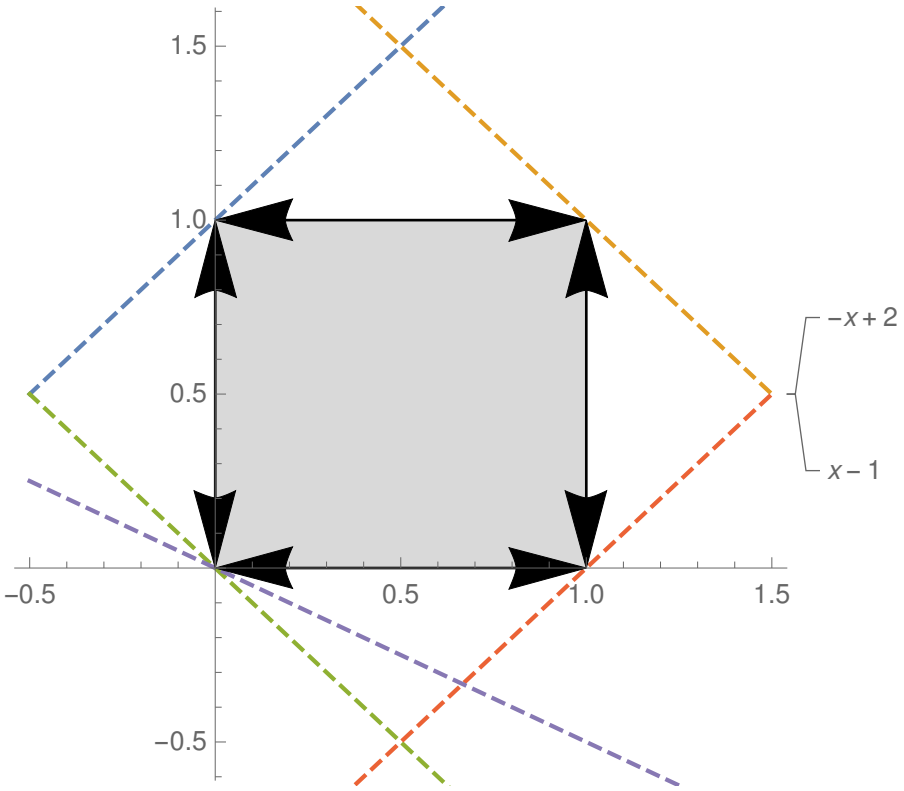


Figure 4: A unit square without its vertices.

in many ways, but let us choose the following set of inequalities:

$$C = \{x \leq 1, 0 \leq x, y \leq 1, 0 \leq y, y < x+1, y < -x+2, y > -x, y > x-1, y > -\frac{1}{2}x\}.$$

Clearly, this is not the minimal set of inequalities, since there are two different constraints, that exclude the vertex $(0,0)$: $y > -x$ and $y > -\frac{1}{2}x$. Whichever of the constraints the algorithm considers first, will end up in the resulting set of constraints E. Both options are equivalent and produce a set with the same number of constraints.

The above example, which shows the non-uniqueness of minimal sets, contrasts with the situation for closed convex polytopes, for which it is known that a unique minimal set exists [7].

Question 4.6. *Given a set of constraints $C(x_1, \ldots, x_n)$, the above procedure yields a minimal subset of constraints $E(x_1, \ldots, x_n) \subseteq C(x_1, \ldots, x_n)$, i.e. if there exists another subset $F \subseteq C$, that is equivalent to C, then $|E| \leq |F|$.*

In particular a positive answer to the question means, that all the sets of constraints, that do not contain any redundant constraints (equivalently all possible results of the procedure), have the same cardinality. If we could prove this, it would be a first step towards proving the above question.

The procedure described above involves simplifying formulas via quantifier elimination, which is a known problem (see [14]). The benefit of this is that the constraint sets used in the representation of a piecewise linear function are kept to minimum size. A potential drawback is the time required by the algorithm that reduces the constraint sets. The practical ramifications of this space/time trade-off will be explored in Section 7. Suppose we have a formula in first-order theory of linear arithmetic

$$\exists x_1 . \exists x_2 . \ldots \exists x_n . \left(\bigwedge_{1 \leq i \leq N} c_i(\vec{x}) \right),$$

where $c_i(\vec{x})$ are (strict or non-strict) inequalities between linear expressions in variables $x_1, \ldots, x_n$. We would like to obtain the equivalent formula without (existential) quantificators. We start by eliminating $\exists x_n$ and then proceed by eliminating $\exists x_{n-1}, \exists x_{n-2}, \ldots, \exists x_1$. We rearrange inequalities in $\bigwedge_{1 \leq i \leq N} c_i(\vec{x})$, to be of the form

$$\begin{aligned} \exists x_1 . \exists x_2 . \ldots \exists x_n . \bigwedge C' \wedge \left(\bigwedge_{1 \leq i \leq l'} x_n > a_i \right) \wedge \left(\bigwedge_{l' < i \leq l} x_n \geq a_i \right) \\ \wedge \left(\bigwedge_{1 \leq i < m'} x_n \leq b_i \right) \wedge \left(\bigwedge_{m' \leq i \leq m} x_n < b_i \right), \end{aligned} \tag{4.9}$$

where inequalities in C' and linear expressions a_i, b_i do not contain the variable x_n. The equivalent formula without x_n is

$$\begin{aligned} \exists x_1 . \exists x_2 . \ldots \exists x_{n-1} . \bigwedge C' \wedge \left(\bigwedge_{1 \leq i \leq l'} \bigwedge_{1 \leq j \leq m} a_i < b_j \right) \wedge \\ \left(\bigwedge_{1 \leq i < m'} \bigwedge_{l' < j \leq l} a_j \leq b_i \right) \wedge \left(\bigwedge_{m' \leq i \leq m} \bigwedge_{1 \leq j \leq l} a_j < b_i \right). \end{aligned} \tag{4.10}$$

We iteratively apply the procedure and obtain the equivalent quantifier-free formula. This algorithm causes an exponential blow up in the size of the formula and is consequently rather slow and has a high space complexity. There are two main measures of the size of the formula: number of existential quantifiers n (which corresponds to dimension), and number of conjuncts N (which corresponds to the constraint set size). To obtain some sort of a worst-case asymptotic formula, we consider how big the blow-up is after eliminating only one quantifier. We observe, that by eliminating $\exists x_n$, we obtain a conjunction of at most $\mathcal{O}(N^2)$ linear inequalities. After we do that for every existential quantifier, we get un upper bound $\mathcal{O}(N^{2^n})$. The good news is

it is the dimension that is the exponent, and in the examples we later consider this number is relatively small, whereas the constraint set can be huge.

However only when we combine the two procedures to obtain an optimised constraint set in the iterative algorithm for finding fixed points from Section 4, the sets are small enough for the computer to deal with. Still, in the worst case, we apply quantifier elimination $\mathcal{O}(|C|^2)$ (quadratically) many times in the iterative algorithm for finding fixed points, which sums up to be quite time consuming. We comment more on that in Section 7, where we present the results of the implementations.

5 μ-terms

A nice source of monotonic piecewise linear functions are Łukasiewicz μ-terms as presented in the article [13]. They are terms in Łukasiewicz many-valued (also called "fuzzy") logic (see [8]) extended with the operations of least and greatest fixed point. The motivation for considering such terms comes from specification logics for probabilistic systems. Let us take a look at the definition of a μ-term.

We use the following grammar

$$t ::= x \mid 0 \mid 1 \mid rt \mid t \sqcup t \mid t \sqcap t \mid t \oplus t \mid t \odot t \mid \mu x.t \mid \nu x.t,$$

where r is a real (sometimes restricted to rational) number between 0 and 1, x ranges over a countably infinite set of variables, $\sqcup$ and $\sqcap$ are Łukasiewicz weak disjunction and Łukasiewicz weak conjunction respectively, $\oplus$ and $\odot$ are Łukasiewicz strong disjunction and Łukasiewicz strong conjunction respectively, μ denotes the least fixed point and ν denotes the greatest fixed point. We write $t(x_1, \ldots x_n)$ to mean all variables of the term t are contained in the set $\{x_1, x_2, \ldots x_n\}$ and we interpret such a term t as a function $[0,1]^n \to [0,1]$, by defining the *value* $t(\vec{r})$ of $t(x_1, \ldots, x_n)$ applied to a vector $(r_1, \ldots, r_n) \in [0,1]^n$ in the following way:

$$\begin{aligned}
x_i(\vec{r}) = r_i,\ 0(\vec{r}) = 0,\ 1(\vec{r}) = 1 &\qquad \text{variables and constants} \\
(qt)(\vec{r}) = q \cdot t(\vec{r}) &\qquad \text{scalar multiplication} \\
(t_1 \sqcup t_2)(\vec{r}) = \max\{t_1(\vec{r}), t_2(\vec{r})\} &\qquad \text{Łukasiewicz weak disjunction} \\
(t_1 \sqcap t_2)(\vec{r}) = \min\{t_1(\vec{r}), t_2(\vec{r})\} &\qquad \text{Łukasiewicz weak conjunction} \\
(t_1 \oplus t_2)(\vec{r}) = \min\{t_1(\vec{r}) + t_2(\vec{r}), 1\} &\qquad \text{Łukasiewicz strong disjunction} \\
(t_1 \odot t_2)(\vec{r}) = \max\{t_1(\vec{r}) + t_2(\vec{r}) - 1, 0\} &\qquad \text{Łukasiewicz strong conjunction}
\end{aligned}$$

We say that a μ-term t is *rational*, if all scalars q appearing in scalar multiplications in t are rational numbers. The connectives above, every term $t(x_1, \ldots, x_n)$ defines a continuous function from $[0,1]^n$ to $[0,1]$. In order to obtain least and greatest fixed points to the function, we can use the Brouwer fixed point theorem ([12]). However, solutions of fixed point expressions are not necessarily themselves continuous as shown by the following example.

Example 5.1. *Consider the μ-term $x \oplus y$ (taken from [13]) and its least fixed point $\mu y.(x \oplus y)$. This term defines a function, which we might naturally express as $\mu y. \min\{x + y, 1\}$. We show that this function is not continuous. Now we consider*

two different cases, depending on the minimum. If the minimum is at $x+y$, to find the fixed point, we need to solve the equation $y = x + y$, which has a solution $x = 0$. If however the minimum is at 1, we have $y = 1$, but this is only under the condition, that $1 \leq x + y$. Since $y = 1$, we get the condition $x > 0$. When we put the two conditions together, we have

$$\mu y.(x \oplus y) = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases}$$

This example means that Brouwer's theorem cannot be invoked to interpret terms with iterated fixed point constructions, since Brouwer's theorem only applies to continuous functions.

We extend the table defining the *value* $t(\vec{r})$ of $t(x_1, \ldots, x_n)$ applied to a vector $(r_1, \ldots, r_n) \in [0,1]^n$ with the following :

$$(\mu y.t(x_1, \ldots x_n, y))(\vec{r}) = \mathrm{lfp}(r' \mapsto t(\vec{r}, r'))$$

$$(\nu y.t(x_1, \ldots x_n, y))(\vec{r}) = \mathrm{gfp}(r' \mapsto t(\vec{r}, r'))$$

where lfp and gfp are the operators returning the least and greatest fixed point respectively. Fixed points bind the variables as seen in the Section 2.4 on fixed points. A μ-term t is *closed*, if all its variables are bound by fixed points. A closed (rational) term t denotes a function from $[0,1]^0$ to $[0,1]$, i.e. it denotes a (rational) number in $[0,1]$. But in order for all this to be well defined and and the fixed points to exist and be unique by the Knaster-Tarski theorem 2.10, we need to show, that terms are monotonic functions (in all arguments).

Proposition 5.2. *A μ-term $t(x_1, \ldots, x_n)$ always defines a monotonic function from $[0,1]^n$ to $[0,1]$.*

The proof is not included in [13] and is therefore given here in detail.

Proof. The proof goes by induction on the structure of t. Suppose $\vec{r}, \vec{s} \in [0,1]^n$ with $\vec{r} \leq \vec{s}$, i.e. for every i, with $1 \leq i \leq n$ it holds that $r_i \leq s_i$. Now consider the following cases.

- If $t = x_i$, then $t(\vec{r}) = r_i \leq s_i = t(\vec{s})$.
- If $t = 0$, then $t(\vec{r}) = 0 = t(\vec{s})$.
- If $t = 1$, then $t(\vec{r}) = 1 = t(\vec{s})$.
- If $t = qt'$, then $t(\vec{r}) = qt'(\vec{r}) \leq qt'(\vec{s}) = t(\vec{s})$.
- If $t = t_1 \sqcup t_2$, we have by induction hypothesis $t_1(\vec{r}) \leq t_1(\vec{s})$ and $t_2(\vec{r}) \leq t_2(\vec{s})$. Without loss of generality, we can assume, that $t(\vec{r}) = t_1(\vec{r})$. So we have $t(\vec{r}) = t_1(\vec{r}) \leq t_1(\vec{s}) \leq \max\{t_1(\vec{s}), t_2(\vec{s})\} = t(\vec{s})$. Therefore $t(\vec{r}) \leq t(\vec{s})$ holds.
- If $t = t_1 \sqcap t_2$, we have by induction hypothesis $t_1(\vec{r}) \leq t_1(\vec{s})$ and $t_2(\vec{r}) \leq t_2(\vec{s})$. By definition of the minimum, we have $t(\vec{r}) \leq t_1(\vec{r}) \leq t_1(\vec{s})$ and $t(\vec{r}) \leq t_2(\vec{r}) \leq t_2(\vec{s})$. So we have $t(\vec{r}) \leq \min\{t_1(\vec{s}), t_2(\vec{s})\} = t(\vec{s})$. Therefore $t(\vec{r}) \leq t(\vec{s})$ holds.

- If $t = t_1 \oplus t_2$, we have by induction hypothesis $t_1(\vec{r}) \leq t_1(\vec{s})$ and $t_2(\vec{r}) \leq t_2(\vec{s})$. Then $t(\vec{r}) = \min\{t_1(\vec{r}) + t_2(\vec{r}), 1\} \leq \min\{t_1(\vec{s}) + t_2(\vec{s}), 1\} = t(\vec{s})$.
- If $t = t_1 \odot t_2$, we have by induction hypothesis $t_1(\vec{r}) \leq t_1(\vec{s})$ and $t_2(\vec{r}) \leq t_2(\vec{s})$. By definition of t, we have $t(\vec{r}) = \max\{t_1(\vec{r}) + t_2(\vec{r}) - 1, 0\}$. Now we need to consider the following cases.
  - If $t(\vec{r}) = 0$ and $t(\vec{s}) = 0$, then we trivially have $t(\vec{r}) \leq t(\vec{s})$.
  - If $t(\vec{r}) = 0$ and $t(\vec{s}) > 0$, then $t(\vec{r}) \leq t(\vec{s})$.
  - If $t(\vec{r}) > 0$, then $t(\vec{r}) = t_1(\vec{r}) + t_2(\vec{r}) - 1 \leq t_1(\vec{s}) + t_2(\vec{s}) - 1 = t(\vec{s})$.
- If $t = \mu x_n.t'$ or $t = \nu x_n.t'$, the appropriate function is monotonic due to Proposition 2.12. □

So by the Knaster-Tarski theorem 2.10, least and greatest fixed points of μ-terms exist and are unique. We motivated μ-terms as a source of piecewise linear functions. Accordingly, we next show that μ-terms indeed define piecewise linear functions. We do that by representing their graph as a formula in first-order theory of linear arithmetic. Because we have already proven in Proposition 3.8, that such formulas represent only piecewise linear functions, this property is sufficient.

Proposition 5.3. *For every Łukasiewicz μ-term $t(x_1, \ldots, x_n)$, its graph*

$$\{(\vec{x}, y) \in [0,1]^{n+1} | t(\vec{x}) = y\}$$

is definable by a fomula $F_t(x_1, \ldots, x_n, y)$ in the first-order theory of linear arithmetic.

Proof. We again prove by induction on the structure of t. Since the formulas are more or less self-explanatory, we do not need additional comments on them. Not all cases are done in [13], but here we present all of them.

- If $t = 0$, then $F_t(y)$ is the formula $0 \leq y \wedge y \leq 0$.
- If $t = 1$, then $F_t(y)$ is the formula $1 \leq y \wedge y \leq 1$.
- If $t = x$, then $F_t(x, y)$ is the formula $(0 \leq y) \wedge (y \leq 1) \wedge (0 \leq x) \wedge (x \leq 1) \wedge (x \leq y) \wedge (y \leq x)$.
- If $t = r \cdot t'$, then $F_t(\vec{x}, y)$ is the formula $F_{t'}(\vec{x}, \frac{y}{r})$.
- If $t = t_1 \sqcup t_2$, then $F_t(\vec{x}, y)$ is the formula $\exists z_1, z_2.F_{t_1}(\vec{x}, z_1) \wedge F_{t_2}(\vec{x}, z_2) \wedge ((z_1 \leq z_2 \wedge y = z_2) \vee (z_2 \leq z_1 \wedge y = z_1))$.
- If $t = t_1 \sqcap t_2$, then $F_t(\vec{x}, y)$ is the formula $\exists z_1, z_2.F_{t_1}(\vec{x}, z_1) \wedge F_{t_2}(\vec{x}, z_2) \wedge ((z_1 \leq z_2 \wedge y = z_1) \vee (z_2 \leq z_1 \wedge y = z_2))$.
- If $t = t_1 \oplus t_2$, then $F_t(\vec{x}, y)$ is the formula $\exists z_1, z_2.F_{t_1}(\vec{x}, z_1) \wedge F_{t_2}(\vec{x}, z_2) \wedge ((z_1 + z_2 \leq 1 \wedge y = z_1 + z_2) \vee (1 \leq z_1 + z_2 \wedge y = 1))$.
- If $t = t_1 \odot t_2$, then $F_t(\vec{x}, y)$ is the formula $\exists z_1, z_2.F_{t_1}(\vec{x}, z_1) \wedge F_{t_2}(\vec{x}, z_2) \wedge ((0 \leq z_1 + z_2 - 1 \wedge y = z_1 + z_2 - 1) \vee (z_1 + z_2 - 1 \leq \wedge y = 0))$.

- If $t = \mu x_{n+1}.t_1$, then $F_t(\vec{x}, y)$ is the formula $F_{t_1}(\vec{x}, y, y) \wedge \forall z.(F_{t_1}(\vec{x}, z, z) \implies y \leq z)$.
- If $t = \nu x_{n+1}.t_1$, then $F_t(\vec{x}, y)$ is the formula $F_{t_1}(\vec{x}, y, y) \wedge \forall z.(F_{t_1}(\vec{x}, z, z) \implies z \leq y)$.

□

Thus we see, that μ-terms are in fact a source of monotone piecewise linear functions. The reverse statement is still an open question:

Question 5.4. *Is every monotone (rational) piecewise linear function $f\colon [0,1]^n \to [0,1]$ definable by a (rational) μ-term $t(x_1, \ldots, x_n)$?*

McNaughton's Theorem (see [11]) famously classifies the functions defined by Łukasiewicz formulas without scalar multiplication as the continuous piecewise linear functions on $[0,1]$ with integer coefficients. A positive answer to the Question 5.4 would provide an analogus result for the Łukasiewicz μ-calculus.

5.1 Algorithm for evaluating μ-terms

We now present an algorithm for computing the function defined by a μ-term t, which works recursively following the structure of t. We have seen that every μ-term defines a monotone piecewise linear function, and our algorithm will compute this function as a local algorithm in the sense of 3.10. This means that least-fixed-point expressions can be computed using the algorithm we gave in Section 4, and greatest-fixed-point expressions can be computed using a dual algorithm (by the principle of symmetry). The other operations of the calculus are dealt with as described below. In [13], the cases are not considered in such detail.

Input: A rational μ-term $t(x_1, \ldots, x_n)$ and a vector of rationals $(r_1, \ldots, r_n) \in [0,1]^n$.
Output: A rational conditioned linear expression $C \vdash e$ in variables $x_1, \ldots x_n$ with the following properties:

(P1) $C(r_1, \ldots, r_n)$ holds,

(P2) for all $s_1, \ldots, s_n \in \mathbb{R}$, if $C(\vec{s})$ holds, then $s_1, \ldots, s_n \in [0,1]$ and $e(\vec{s}) = \mu x_{n+1}.t(\vec{s}, x_{n+1})$.

The algorithm works recursively on the structure of the term t.
if $t = 0$ **then**
  return $\emptyset \vdash 0$
end if
if $t = 1$ **then**
  return $\emptyset \vdash 1$
end if
if $t = x$ **then**
  return $\{0 \leq x, x \leq 1\} \vdash x$ (range constraints need to be present)
end if
if $t = rt_1$ **then**

recursively compute $C_1 \vdash e_1$ for t_1.
return $C_1 \vdash re_1$
end if
if $t = t_1 \sqcup t_2$ **then**
recursively compute $C_1 \vdash e_1$ for t_1 and $C_2 \vdash e_2$ for t_2.
if $e_1(\vec{r}) \leq e_2(\vec{r})$ **then**
return $C_1 \cup C_2 \cup \{e_1 \leq e_2\} \vdash e_2$
else
return $C_1 \cup C_2 \cup \{e_1 \geq e_2\} \vdash e_1$
end if
end if
if $t = t_1 \sqcap t_2$ **then**
recursively compute $C_1 \vdash e_1$ for t_1 and $C_2 \vdash e_2$ for t_2.
if $e_1(\vec{r}) \leq e_2(\vec{r})$ **then**
return $C_1 \cup C_2 \cup \{e_1 \leq e_2\} \vdash e_1$
else
return $C_1 \cup C_2 \cup \{e_1 \geq e_2\} \vdash e_2$
end if
end if
if $t = t_1 \oplus t_2$ **then**
recursively compute $C_1 \vdash e_1$ for t_1 and $C_2 \vdash e_2$ for t_2.
if $e_1(\vec{r}) + e_2(\vec{r}) \leq 1$ **then**
return $C_1 \cup C_2 \cup \{e_1 + e_2 \leq q\} \vdash e_1 + e_2$
else
return $C_1 \cup C_2 \cup \{e_1 + e_2 \geq 1\} \vdash 1$
end if
end if
if $t = t_1 \odot t_2$ **then**
recursively compute $C_1 \vdash e_1$ for t_1 and $C_2 \vdash e_2$ for t_2.
if $e_1(\vec{r}) + e_2(\vec{r}) - 1 \geq 0$ **then**
return $C_1 \cup C_2 \cup \{e_1 + e_2 - 1 \geq 0\} \vdash e_1 + e_2 - 1$
else
return $C_1 \cup C_2 \cup \{e_1 + e_2 - 1 \leq 0\} \vdash 0$
end if
end if
if $t = \mu x_{n+1}.t_1$or $\nu x_{n+1}.t_1$ **then**
Run algorithm for finding fixed points from Section 4 with a recursive call to evaluate t_1 as the input to local algorithm. The greatest fixed point is computed using a dual algorithm. Then **return** the result.
end if

6 Implementation of the algorithm

We implemented the algorithm for finding fixed points of monotone piecewise linear functions via evaluating μ-terms. The implementation subsumes all four versions of the algorithm: the original one from [13], the modified one, and each of those versions

equipped with quantifier elimination optimisation. The entire implementation and results are an original contribution.

The full source code of the implementation in python 3 is accessible at `https://bitbucket.org/AnjaPetkovic/evaulating-mu-terms`. We strongly recommend to examine the README.md file before running the code, since there is a short tutorial about the specifications of the implementation. Here, we present only some ideas of the implementation and how the algorithm works. We are always working with μ-terms instead of piecewise linear functions themselves, so the code is adapted for the purpose of evaluating terms.

6.1 The class of μ-terms

μ-terms are generated using the muTerm class. A muTerm has a simple tree structure with the following attributes: operator (the outmost type of the term), subterm1, subterm2, value and name. The value is only relevant for constants and rational multiplication, whereas the name is only relevant for variables and operators μ and ν. The default value for unnecessary attributes is None. To construct a term $\mu y.(x \odot y)$ we need to write the following code.

```
term = muTerm('mu',
muTerm('.',
muTerm('var',None,None,'x'),
muTerm('var',None,None,'y'),
 None, None),
None, None, None, 'x')
```

The names of the variables can be strings or numbers. It is convenient to have canonical names for bound variables. Therefore for closed terms, we apply the following convention: the outer variable is named var(1), then var(2) etc. For example the cannonical names for variables in term $\mu x.\mu y.(x \odot y)$ are

$$\mu \text{ var}(1).\mu \text{ var}(2).(\text{ var}(1) \odot \text{ var}(2)).$$

Because this is not usually the way we input a μ-term, we have a function *rename_variables* to take care of it for us.

It is also useful for testing, to adapt the μ-terms in an equivalent form according to the golden lemma of μ-calculus 2.13 to eliminate all consecutive (nested) μs and νs. We achieve that by calling the function *eliminate_consequent_mu*.

6.2 Generating examples

Since meaningful examples that arise from probabilistic model-checking problems are difficult to construct, we test the algorithm on random μ-terms. We consider two versions of terms: floating point and (exact) rational. In the first version, a rational number represented as a floating point number (for which the computuation is approximate due to floating point issues), in the second version, we use python's class *Fraction*, to represent rational numbers as fractions, and the computations are done precisely. The deficiencies of floating point is discussed with other results in Section 7.

To measure the algorithm's performance, we need to compare examples with the same number of fixed points. For convenience, we put all the fixed points on the outside, i.e. the randomly constructed examples are of the form

$$\mu x_1.\nu x_2.\mu x_3.\ldots\mu x_n.t(x_1,\ldots,x_n),$$

where t does not itself contain any fixed point constructions. Of course the least and greatest fixed points do not necessarily interleave, but if they do not, we can apply the function to eliminate consequent μs and νs and leave us with an equivalent, but smaller term. However in the file *random_example_generator.py* we can find functions that generate μ-terms with fixed points interleaving and μ-terms with consequent μs and νs. Those last ones are used for testing that the results are the same by all versions of the algorithm and the same as for the reduced μ-term after applying *eliminate_consequent_mu* function.

Because we are generating rational terms, we need to provide rational scalars for scalar multiplication. There are two possible approaches to this task. First, we can choose a random real number on $[0,1]$ and convert it to an exact rational (by taking its finite representation in the computer). However we would like to have examples with small denominators, so we take a different approach. If at any point we choose a scalar, we do it by the following procedure:

$m := 2$
while True **do**
  uniformly at random choose an integer i such that $1 \leq i \leq m$.
  if $i = m$ **then**
    uniformly at random choose an integer j such that $1 \leq i \leq m$.
    return $\frac{j}{m}$
  end if
  $m := m + 1$
end while

This procedure provides us with a fraction of small rationals. We encounter the problem, when fraction is in lowest term. Therefore there are many different ways to obtain a certain rational number. For example $1 = \frac{2}{2} = \frac{3}{3} = \cdots$. However if we regard fractions in their rudimentary form as a pair of integers, we obtain the following distribution: Let $k, m \in \mathbb{Z}_+$, then

$$\mathbb{P}\left(X = \frac{k}{m}\right) = \left(\prod_{n=2}^{m-1}\left(1 - \frac{1}{n}\right)\right)\frac{1}{m^2} = \frac{1}{m-1}\cdot\frac{1}{m^2}.$$

To obtain this formula, we think in the following terms. For m to be chosen as a denominator, all $2, 3, \ldots m-1$ must not be chosen, so we get a product of expressions $(\left(1 - \frac{1}{n}\right))$. Then at iteration m, the correct denominator is chosen with probability $\frac{1}{m}$ and again the nominator is chosen with the same probability as the denominator. If we still desire to calculate the probability of X being a concrete rational number, we need to calculate the sum

$$\mathbb{P}\left(X = \frac{k}{m}\right) = \sum_{n=1}^{\infty}\mathbb{P}\left(X = \frac{nk}{nm}\right) = \sum_{n=1}^{\infty}\frac{1}{nm-1}\cdot\frac{1}{n^2m^2},$$

where $k, m \in \mathbb{Z}_+$ such that $\gcd(k, m) = 1$. The procedure for randomly generating scalars can never generate 0. However, as it is later on described, we can generate a term 0 via choosing it as an operator.

The algorithm has the property that, although it terminates with probability 1, the expected number of iterations of the while loop until termination is infinity. Equivalently, the denominator has infinite expectation. Interestingly enough, we never get long delays in randomly choosing a scalar.

To generate random terms, we randomly choose operators. But because we do not want the terms to get too big, every fifth operator needs to add an unused variable to the term. When we reach an assigned number of free variables, we close the term with μs and νs. And thus provide a random closed term. We only need to rename its variables, to be used in further process.

This procedure of generating random terms is clearly ad hoc. Nevertheless it proved sufficient in practice to generate examples that led to interesting behaviour of the algorithm.

6.3 Algorithm evaluate

The algorithm for evaluating μ-terms (and in its most interesting part, calculating fixed points of monotonic piecewise linear functions) deals with conditioned linear expressions of the form $C \vdash e$. Since sets are inconvenient to maintain in the computer, we use lists.

Every linear expression

$$q_0 + q_1 x_1 + \ldots, q_n x_n$$

is represented by a list of the form

$$[q_0, q_1, \ldots, q_n].$$

Because the variables are named in a canonical way, we store the i-th coefficient of the linear expression at the i-th place (starting the counting from 0). When we want to apply a linear expression to a vector $\vec{r} \in [0, 1]^n$, we merely do the scalar product of the two lists. But we need to be careful, since the lists, that represent linear expressions, have one extra element, q_0, whereas $\vec{r}$ has exactly n components.

The inequalities between linear expressions $e_1 \triangleleft e_2$ (where $\triangleleft$ denotes $<$, $\leq$, $>$ or $\geq$) are stored as a triple

$$(\triangleleft, e_1, e_2),$$

where e_1 and e_2 are of course the lists like above and $\triangleleft$ is given by a string. A conditioned linear expression $C \vdash e$ is a pair (C, e), where C is represented by a list of inequalities (triples) and e is a list (as appropriate for a linear expression).

The implementation follows the algorithm algorithm from Section 5.1 for evaluating μ-terms using given data structures described above. When inequalities have to be rearranged as for example in equation (4.1), we do the appropriate manipulations on the lists. We also have to be careful when using division, in order to make sure the resulting lists are still of the same types and dealing with exact rationals.

There are 4 different versions of the evaluating algorithm. The original version, adapted from the article [13], is subsumed in the function *evaluate*, that takes three

arguments: the μ-term, the vector $\vec{r}$ and a boolean determining whether we would like to test the performance of the algorithm in this example or not.

Next we can add optimization of the constraint sets via quantifier elimination to the original version of the algorithm. At every point, when we change a set of constraints, we optimize it via the procedure described above in Section 4.3 for getting rid of redundant inequalities. This shrinks the constraint sets and enables further computations, when we would otherwise run out of memory. To use this procedure, we call the function *evaluate_ optimized*.

The last two versions follow the same pattern. First we evaluate μ-terms using the modified algorithm for calculating fixed points, as described in Section 4.2.1. Second, we again combine this modified algorithm with optimisation of the constraint sets via quantifier elimination. This is done by calling the function *evaluate_ optimized_ direct* with an additional argument *quantifier_ elimination*, which is a boolean, that determines whether the constraint sets are being optimized or not.

6.4 Testing the performance

The main purpose of implementation is testing the performance of the algorithm (in all four versions). There is always a question as to what are good performance measures for such procedures. There seems to be many options, from the basis size and the conditon size, that we have already encountered in Section 4.2.2, to the actual amount of time, the algorithm spends on computing the result. Since we can only run the algorithm on one vector at the time, calculating the basis size is practically impossible for bigger terms (with a lot of fixed points to compute). We also need to find a properties that will highlight the differences between different versions of the algorithm.

We have therefore decided to measure the following performance properties:

- number of repetitions of the main loop in the fixed-point algorithm (e.g., in the case of the third implementation (modified algorithm without redundancy removal), the "loop" marked in the algorithm in Section 4). In the implementation, we call this the "while" loop,
- actual time spent on the computation (in seconds),
- maximal size of a constraint set encountered during the computation.

The first two properties measure how slow is the process, whereas the third property is a measurement of space complexity. We expect the two to be correlated. In particular, one expects large constraint sets to take time to process.

In order to perform the tests, we have constucted so called benchmark examples, a list of μ-terms with specific properties, namely we determine how many μs and νs are at the beginning of a μ-term (see Section 6.2 for details). We have constructed hundreds of terms, that all have the same number of μs and νs, which either interleave or appear in a consecutive order (first all μs, then all the νs). To check that the programs work correctly, we have then run all four versions of the algorithm on the examples and made sure, that we get the same results. For the terms with consecutive fixed points, we hve checked, that the terms, reduced by the

function *eliminate_consequent_mu*, also produce the same results. However by "the same results" we mean the linear expression (in our case the number obtained from evaluating closed term) and not the set of constraints.

While measuring the performance, we have run the algorithms on examples with the total number of μs and νs ranging from 0 to 9, making 10 examples for each number of fixed points, and then we looked at the average performance, the median and the worst case. Even though an average out of 10 does not give us very steady results, we can see certain trends in the performance. The reason for such small number of fixed points is, that the algorithms (especially the original one from [13]) are so slow and space-consuming, that we simply cannot run any bigger examples on an ordinary computer. If however we apply quantifier elimination to a modified version of the algorithm, we can get further than that (up to 15 fixed points), although the computation is still very slow and thus time-consuming.

The testing is done in the file *test.py*, graphs of performance are generated by the script in *graphs.py* and results are compared by the script *comparison.py*.

7 Results

First, let us consider the issues, when the terms are not presented with exact rational numbers. Because we have to check certain equalities between rational numbers in the algorithm (for example $q_{n+1} = 1$), which cannot be properly done if we are not dealing with exact rationals, the algorithm does not necessarily terminate. Such examples are written in the file *problematic_examples.py*. Therefore we only deal with the version, where we use exact rational numbers, modelled by the python's module *Fractions*.

In order to evaluate the performance of each version of the algorithm, we take a look at the following graphs of performance properties according to the number of fixed points in the term being evaluated. We note, that we have graphs in logarithmic (base 10) scale, to demonstrate the exponential growth.

In Figure 5 we show how many iterations are needed in average according to the number of fixed points. There is only one line because all four versions of the algorithm perform exactly the same number of iterations. In fact, when we take a look on the raw performance data, we observe, that on all examples we have tested the algorithm on, the number of iterations is the same in all versions. It does not follow from this that the different versions of the algorithm will always take the same number of iterations. There is sufficient nondeterminism in the algorithms (for example, in the modified version, in the choice of constraint N, or supremum term b_j) that it is very plausible that the number of iterations does not always agree in general. However, we have not found a single example for which a difference occurs in practice. In Figure 5 we have an even further confirmation of that fact, when we have three lines, that represent averages, medians and maximums of numbers of iterations and they are the same for all versions of the algorithm. However we spot a peculiar behaviour, when we go from 8 fixed points to 9 and the average number of iterations drops. We believe this is due to the very small number of examples and is consequently noise in the trend. To have more reliable data, we would need to process many more examples, as previously discussed in Section 6.4. Still, overall we

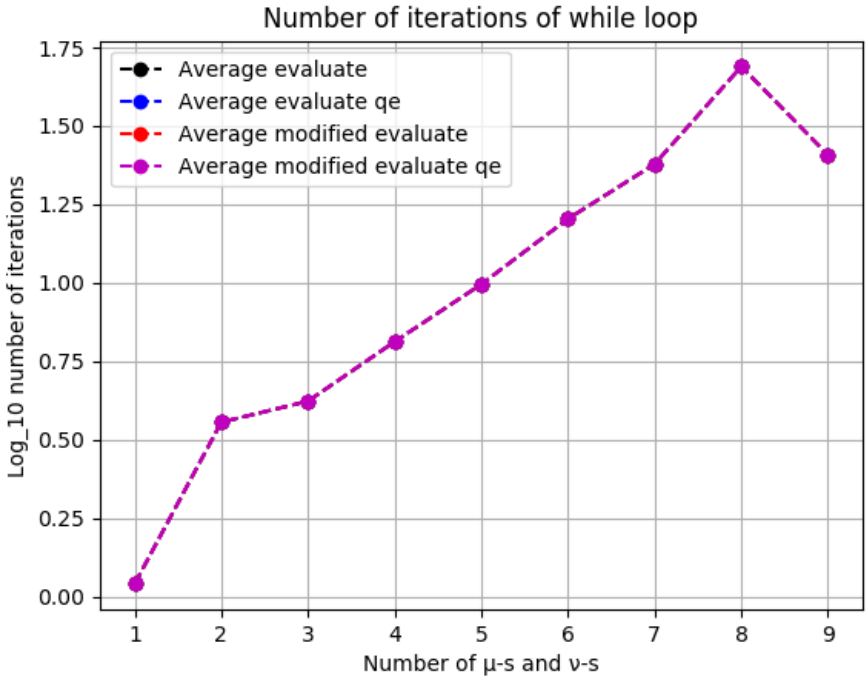


Figure 5: Graph of averages of number of loop iterations.

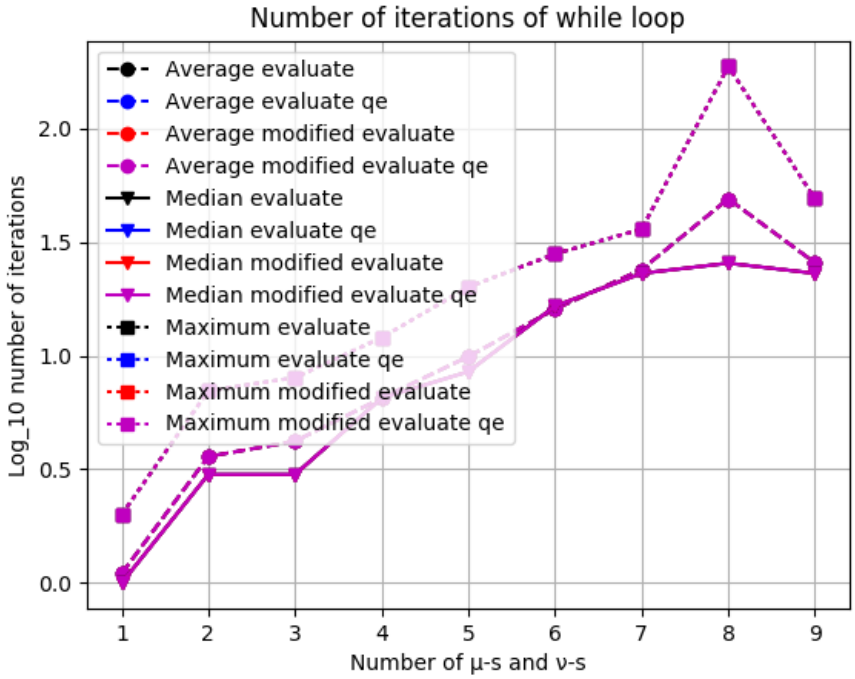


Figure 6: Graph of averages, medians and maximums of number of loop iterations.

can see a distinct exponentional growth of the number of iterations in all cases. Such exponential behaviour is presumably unavoidable (due to the problem subsuming

the solution of simple stochastic games [4], for which the best known algorithm is exponential).

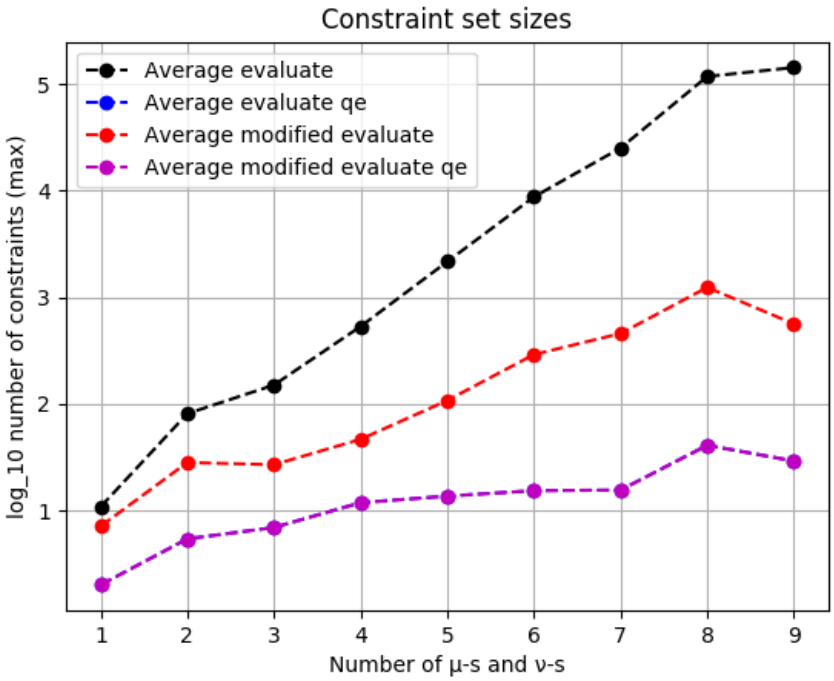


Figure 7: Graph of averages of constraint set sizes.

We now take a look at the average size of constraint sets in Figure 7. The number of inequalities inside the largest constraint set significantly differs between the initial version of the algorithm (evaluate) and the modified version. For the first, we see an exponential growth, but for the second, the growth appears to be a slower exponential growth (with smaller basis), which is expected, since we have eliminated adding quite a few constraints. Again we attribute the down trend at 9 fixed points to a small number of examples. The slowest growth and consequently the smallest number of constraints is obtained when we use optimization via quantifier elimination. We observe, that in both versions, that use quantifier elimination, we get exactly the same results, as seen in Figure 8. This reinforces our belief, that the procedure via quantifier elimination really gives us the smallest number of constraints possible as stated in Question 4.3. We can be sure, that the averages give us a good insight on the behaviour, since we can compare the versions of the algorithms with medians and maximums in Figure 9 and get roughly the same results. In order to get the feeling of just how much the constraint sets get smaller if optimised via quantifier elimination, we take a look at the largest constraint set for 9 fixed points. With the original algorithm, the largest constraint set contains 409385 constraints, with the modified version it contains 2208 constraints and after applying quantifier elimination it contains only 62 inequalities.

When we compare, how much time is spent on computation per example, we are slightly surprised by the results. In Figure 10 we see an average time spent on the examples for all four versions of the algorithm. It is interesting how quanti-

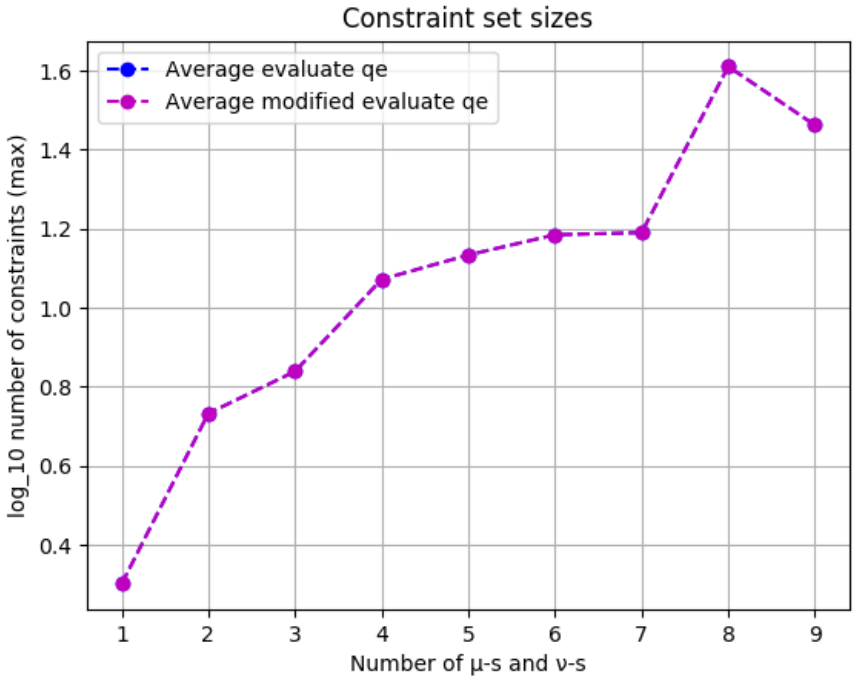


Figure 8: Graph of averages of constraint set sizes for versions with quantifier elimination.

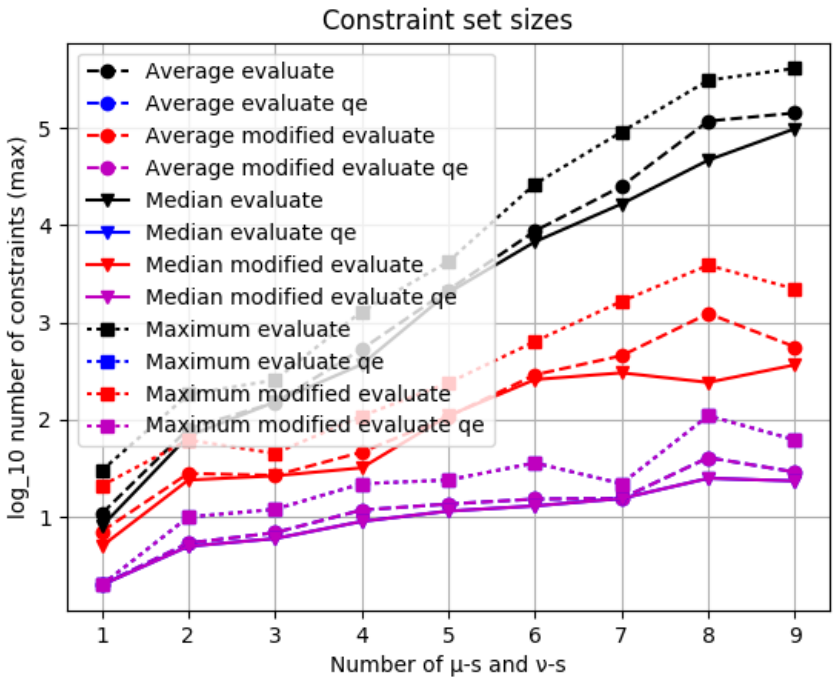


Figure 9: Graph of maximums, averages and medians of constraint set sizes.

fier elimination anihilates any progress in saving time on computations by making constraint sets smaller. Since with quantifier elimination optimisation there are not nearly as many constraints in the sets as in the initial version of the algorithm, we would expect the procedure to be much faster as the iterations through the sets are faster. But it seems that quantifier elimination slows the process down substantially and there seems to be some sort of trade off between space complexity and time complexity in this case. We can again observe an exponential growth in the time spent on computations for all versions apart from the modified one without qunatifier elimination, which seems to be the fastest and represents the best trade off between time and space requirements. We can observe the same in the graph that shows averages and maximums in Figure 11 and in the graph that illustrates the medians in Figure 12.

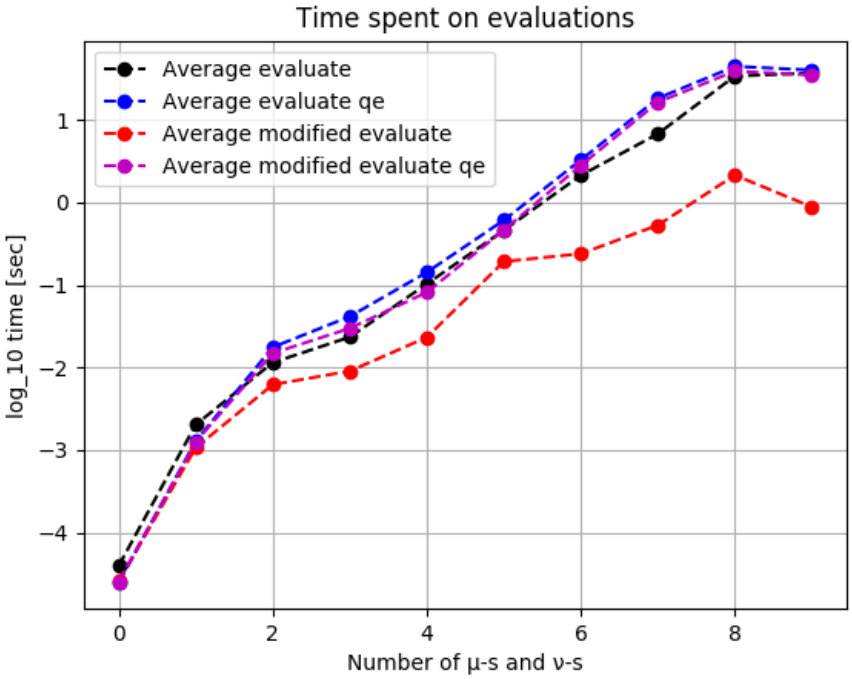


Figure 10: Graph of averages of time spent on evaluations of terms.

Overall, we can conclude, that our modification of the algorithm is in fact useful, but if we have space issues, we still may have to use quantifier elimination optimisation of the constraint sets, which has a serious negative effect on the time spent per computation.

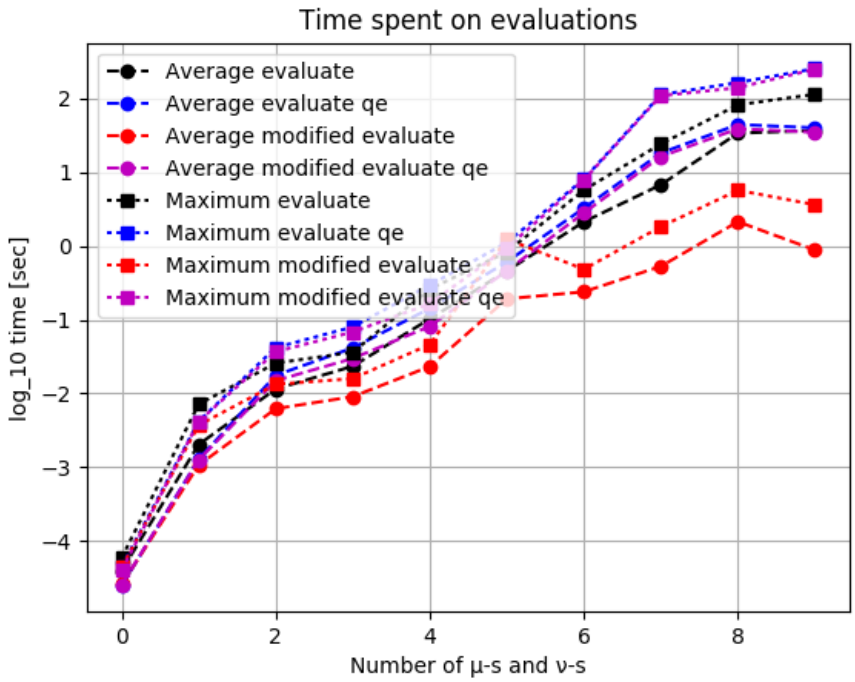


Figure 11: Graph of maximums and averages of time spent on evaluations of terms.

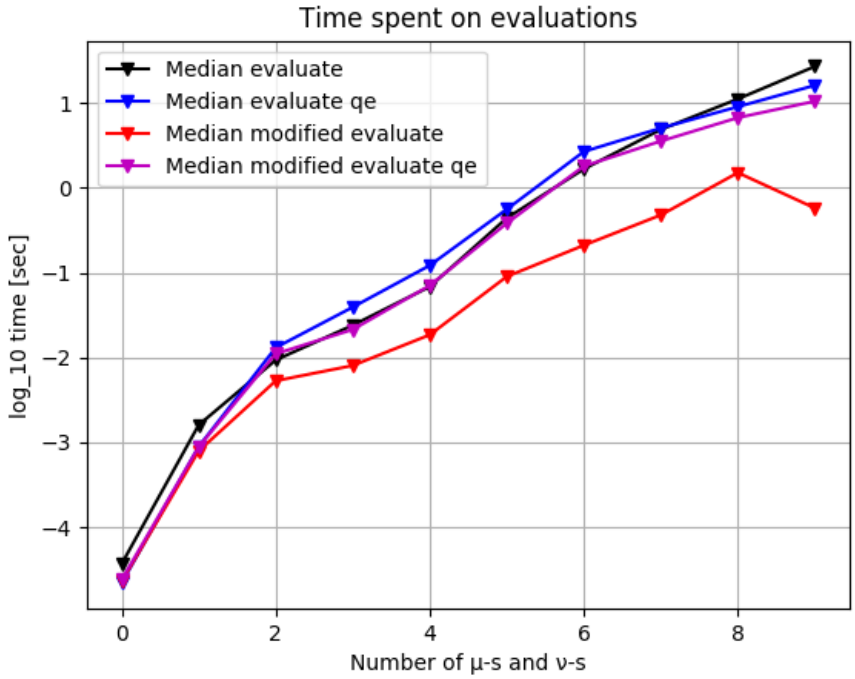


Figure 12: Graph of medians of time spent on evaluations of terms.

8 Conclusion and further work

In terms of both time and space complexity the modified algorithm is a serious improvement on the original one from [13]. This problem subsumes alhorithmic problems such as the solution of "simple stochastic games" [4], for which the best known algorithms are all exponential (although there is no proven hardness result). It would be interesting to obtain more experimental evidence comparing our best algorithm (the modified one) with known algorithms for simple stochastic games, and for quantified model checking. We could still aim to enhance the performance, perhaps starting with a more efficient implementation of the optimisation of the constraint sets.

There are still two open questions to be tackled. A positive answer to Question 4.3 on the minimality of the constraint set obtained via quantifier elimination optimization, would give us grounds on which we may compare constraint sets and would also yield an interesting fact about the facets of "generalised" polytopes (when we have the strict inequalities together with the non-strict ones) as a consequence. It is very plausible that the answer to this question exists in the literature. However, we have not been able to find it.

We believe that the answer to Question 5.4 on whether or not μ-terms subsume all monotone (rational) piecewise linear functions is positive. Such an answer would definitely give μ-terms an even grater meaning and consequently make our implementations even more useful.

Besides the two questions, that were also mentioned in the previous chapters, we offer some additional ideas for futher advancement. On the more theoretical side, we could investigate the μ-terms determine some sort of "distributivity" laws for fixed point operators over the rest of operators, i.e. we could examine for which of the operators $\circ \in \{\sqcup, \sqcap, \oplus, \odot\}$ are equalities of the form

$$\mu x.(t_1 \circ t_2) = (\mu x.t_1) \circ (\mu x.t_2),$$

where t_1 and t_2 are terms, that involve variable x, ture. The idea is to use such distributivity laws to pre-process terms, allowing for potential more efficient evaluation.

If however we focus on the implementation, one could implement full quantifier elimination on arbitrary formulas from the first-order theory of linear arithmetic and use it to compute evaluation of μ-term. This would be done via converting the μ-term to a fomula and performing quantifier elimination in order to compare this approach to the iterative algorithm described in Section 4. It is to be expected that using a general quantifier elimination algorithm in this way will result to a less efficient computation, because quantifier elimination is known to be a very costly procedure.

Another issue is obtaining some theoretical bounds on the time/space efficiency of the iterative algorithm. It seems very hard to analyse the time and space requirements with any degree of precision. It would be good to have at least heuristic mathematical explanations for the observed exponential time behaviour. We could also investigate theoretical bounds on what complexity class the problem of computing the value of a μ-term lies in. It subsumes the problem of solving simple

stochastic games [4], which is known to be in $NP \cap coNP$, but it is not known whether it lies in P. Is the problem of computing the value of a μ-term also in $NP \cap coNP$?

We could also investigate in developing versions of the iterative algorithm that efficiently solve systems of simultaneous fixed-point equation with the same fixed point (least or greatest), see [1] for further information on how to achieve that. This constrasts with the current algorithm that works one variable at a time. Once this is done, it will be possible to apply the algorithm to model-checking problems, and to compare it to standard specialist algorithms for such problems.

Perhaps Microsoft's Z3 (see `https://github.com/Z3Prover/z3`) could be used for solving those systems of equations. But it is highly nontrivial given that we are working with simultaneous equations involving complicated piecewise linear functions. It would be easier to essentially modify the iterative algorithm so that the linear solution finding part which solves $x_{n+1} = e(x_1, ..., x_{n+1})$ (where e is a linear expression), does the relevant linear algebra to solve several variables simultaneously. One would then need to adjust all the "find next approximation" subprocedure appropriately.

9 Razširjeni povzetek v slovenščini

V tem razdelku podajamo daljši povzetek v slovenščini. Osredotočili se bomo na definicije pojmov, zanimive trditve in ključne izreke, dokaze pa bomo spuščali, saj so podrobno podani v angleškem jeziku. Prevodi angleških pojmov so bolj ali manj dobesedni in se morda ne skladajo s konvencijo.

9.1 Uvod

V tem projektu se ukvarjamo z monotonimi odsekovno linearnimi funkcijami in različnimi algoritmi za računanje njihovih najmanjših in največjih fiksnih točk. S fiksnimi točkami so se matematiki ukvarjali v različnih kontekstih, od Banachovega skrčitvenega načela [9], kjer gledamo konvergenco, do Brouwerjevega izreka o fiksni točki [12], ki se osredotoča na topološki vidik in preučuje funkcije na kroglah. Mi si bomo pogledali fiksne točke v smislu izreka Knaster-Tarski [1].

Problem računanja fiksnih točk montonih odsekovno linearnih funkcij izvira iz specifikacije in verifikacije sistemov, ki se vedejo nedeterministično in verjetnostno. Ker so funkcije, ki pridejo iz teh sistemov, nezvezne, se ne moremo omejiti le na zvezne odsekovno linearne funkcije, ampak upoštevamo tudi nezvezne.

Glavna literatura pri tem projektu je članek, ki sta ga napisala M. Mio in A. Simpson [13]. V njem najdemo prvotno različico algoritma za računanje fiksnih točk monotonih odsekovno linearnih funkcij. Mi pa smo vpeljali nekaj posodobitev, da bi izboljšali njegovo učinkovitost. Vse te spremembe, kot tudi dokaz, da so pravilne in da se posodobljen algoritem vedno ustavi, so originalno delo. Tudi optimizacija preko eliminacije kvantifikatorjev je nova in torej originalen prispevek.

9.2 Mreže, negibne točke in izrek Knaster-Tarski

V tem razdelku smo si pomagali predvsem s knjigo [1]. Uporabljali bomo notacijo $(E, \leq)$ za urejeno množico, kjer je $\leq$ delna urejenost. Ko bo relacija očitna iz konteksta, bomo oznako zanjo spuščali. Spomnimo se, da je element $e \in E$ *zgornja meja* za X, kjer je $X \subseteq E$, če za vsak $x \in X$ velja $x \leq e$. Podobno je e *spodnja meja*, če velja $e \leq x$ za vse $x \in X$. Element $e \in E$ je *najmanjša zgornja meja* množice $X \subseteq E$, če je najmanjši med vsemi zgornjimi mejami, tj. če sta izpolnjena pogoja

- $\forall x \in X,\ x \leq e$,
- če velja $\forall x \in X,\ x \leq f$ za nek $f \in E$, potem je $e \leq f$.

Podobno velja za *največjo spodnjo mejo*. Najmanjša zgornja meja je enolična in jo označimo z $\bigvee X$ (oziroma $\bigwedge X$ za največjo spodnjo mejo).

Definicija 9.1. Naj bo $(E, \leq)$ takšna urejena množica, da ima za vsaka dva elementa $x, y \in E$, množica $\{x, y\}$ najmanjšo zgornjo mejo $x \vee y$ in največjo spodnjo mejo $x \wedge y$. Potem je E *mreža*. Če ima vsaka podmnožica $X \subseteq E$ najmanjšo zgornjo mejo $\bigvee X$ in največjo spodnjo mejo $\bigwedge X$, potem pravimo, da je E *polna mreža*.

V posebnem ima polna mreža najmanši element $\bigvee \emptyset$ in največji element $\bigwedge \emptyset$.

Primer 9.2. Kanonični primer polne mreže je potenčna množica z relacijo inkluzije. Naj bo S poljubna množica in $\mathcal{P}(S)$ njena potenčna množica. Tedaj je $(\mathcal{P}(S), \subseteq)$ delna urejenost in za poljubno podmnožico $X \subseteq \mathcal{P}(S)$ imamo $\bigvee X = \bigcup X$ in $\bigwedge X = \bigcap X$.

Opazimo, da sta pojma spodnje meje in zgornje meje zelo simetrična. Velja, da ima poljubna lastnost, ki velja za najmanjše zgornje meje, svojo dualno formulacijo za največjo spodnjo mejo in torej nam lastnosti ni potrebno zopet dokazati. Temu pravimo *princip simetrije*.

Sedaj pa lahko podamo tudi formalno definicijo monotonosti funkcije med delno urejenima množicama.

Definicija 9.3. Naj bosta $(E, \leq_E)$ in $(F, \leq_F)$ urejeni množici. Funkcija $f\colon E \to F$ je *monotona*, če

$$\forall x, y \in E, x \leq_E y \implies f(x) \leq_F f(y).$$

Definicija se sklada z definicijo monotonosti, ki jo že poznamo na funkcijah v $\mathbb{R}$, vendar pa smo navajeni razlikovati med naraščajočimi in padajočimi funkcijami. Za nas po zgornji definiciji 9.3 monotonost pomeni zgolj naraščanje in bomo termin monotonost tudi uporabljali v tem ožjem smislu.

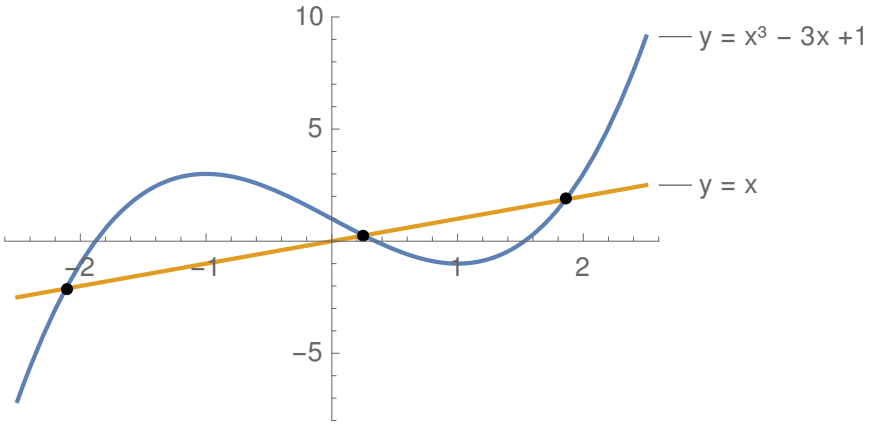
Osredotočimo se na funkcije iz urejene množice E nazaj vase in definirajmo *fiksno točko*.

Definicija 9.4. Naj bo E poljubna množica in $f\colon E \to E$ a funkcija. *Fiksna (negibna) točka* funkcije f je takšen element $x \in E$, da velja $f(x) = x$. Množico vseh fiksnih točk funkcije f označimo s $\mathrm{Fix}(f)$.

Naj opozorimo, da sta za nas izraza *fiksna točka* in *negibna točka* sopomenki in ju bomo uporabljali izmenjujoče.

Če imamo funkcijo $f\colon \mathbb{R} \to \mathbb{R}$, njena fiksna točka leži na preseku grafa funkcije f in diagonale $y = x$.

Primer 9.5. Oglejmo si funkcijo $f(x) = x^3 - 3x + 1$ in njen graf na sliki 13. Opazimo,



Slika 13: Fiksne točke funkcije $f(x) = x^3 - 3x + 1$.

da ima f tri fiksne točke, torej $|\mathrm{Fix}(f)| = 3$. Čeprav so fiksne točke le x-koordinate, so zavoljo jasnosti označeni preseki z diagonalo.

Ko je E urejena množica, je $\mathrm{Fix}(f)$ urejena podmnožica in je v splošnem lahko prazna. Vendar pa ko je E polna mreža, fiksne točke gotovo obstajajo. Nas naj-

bolj zanimajo najmanjše in največje fiksne točke monotonih funkcij. Njihov obstoj zagotavlja izrek Knaster-Tarski.

Izrek 9.6 (Knaster-Tarski)**.** *Naj bo* $(E, \leq)$ *polna mreža in* $f\colon E \to E$ *monotona funkcija. Tedaj* $\bigwedge \mathrm{Fix}(f)$ *in* $\bigvee \mathrm{Fix}(f)$ *pripadata* $\mathrm{Fix}(f)$*.*

Preden gremo naprej, spoznajmo notacijo za najmanjše in največje negibne točke. Idejo si sposodimo pri predikatni logiki, kjer kvantifikatorja $\exists$ in $\forall$ vežeta spremenljivke. Tam uporabljamo notacijo $\exists x.expr[x]$, da povemo, da kvantifikator $\exists$ veže spremenljivko x v nekem izrazu $expr$, ki lahko vsebuje x. V izogib težavam z imeni spremenljivk, lahko vezane spremenljivke tako preimenujemo:

$$(\exists x.expr[x]) \to (\exists y.expr[y]).$$

Ko bi radi zapisali fiksne točke funkcije f, uporabimo naslednjo notacijo:

- najmanjšo fiksno točko zapišemo kot $\mu x.f(x)$,
- največjo fiksno točko zapišemo kot $\nu x.f(x)$.

Primera sta si med seboj dualna preko principa simetrije. Zopet je spremenljivka x vezana preko ekstremalne fiksne točke in jo lahko preimenujemo na enak način kot prej.

Imamo pa lahko tudi funkcije več spremenljivk. Naj bosta E in F polni mreži in $f\colon E \times F \to E$ monotona funkcija v obeh argumentih. Za poljuben $y \in F$ definiramo $f_y\colon E \to E$ z $f_y(x) = f(x, y)$. Z $\mu x.f(x, y)$ označimo funkcijo iz F v E definirano s predpisom $\mu x.f(x, y) = \mu x.f_y(x)$ (podobno $\nu x.f(x, y)$). Velja naslednja trditev.

Trditev 9.7. *Naj bosta* E *in* F *polni mreži. Če je* $f\colon E \times F \to E$ *monotona v obeh argumentih, potem sta* $\mu x.f(x, y)$ *in* $\nu x.f(x, y)$ *monotoni funkciji iz* F *v* E*.*

To dejstvo je pomembno, ko želimo narediti več zaporednih fiksnih točk, npr. $\mu x.\nu y.f(x, y)$. Tako nam izrek Knaster-Tarski 9.6 zagotavlja obstoj tudi fiksnih točk funkcije $\nu y.f(x, y)$.

Naslednja lema je tako zvana "zlata" lema μ-računa, saj se pogosto uporabi v dokazih. Omogoča nam, da zaporedne fiksne točke istega tipa (gnezdene fiksne točke) zamenjamo s preprostejšim izrazom.

Lema 9.8 (Zlata lema $\boldsymbol{\mu}$-računa)**.** *Naj bo* E *polna mreža in* $f\colon E \times E \to E$ *monotona funkcija v obeh argumentih. Tedaj velja*

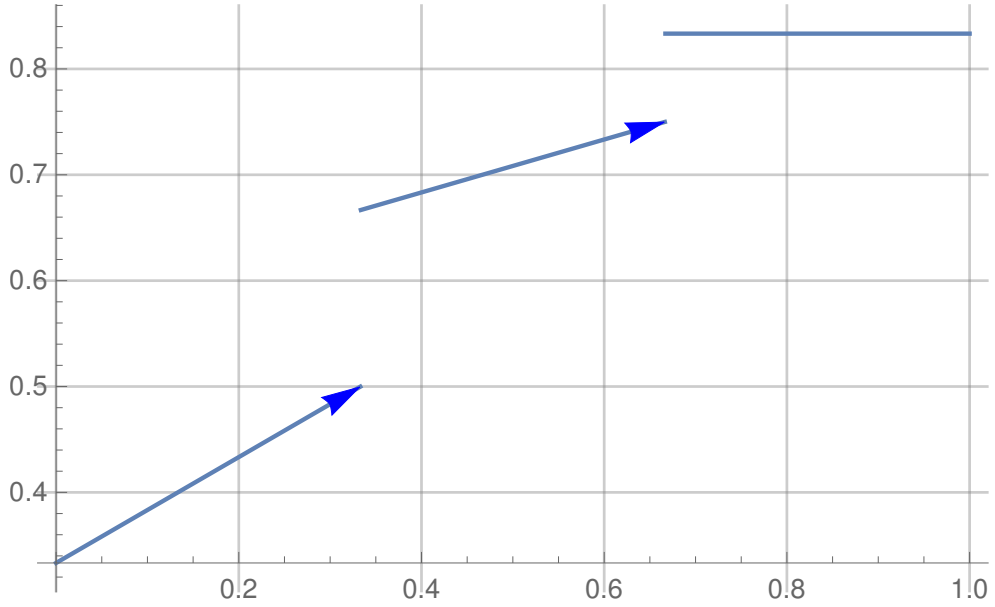
$$\mu x.\mu y.f(x, y) = \mu x.f(x, x) = \mu y.\mu x.f(x, y)$$

in

$$\nu x.\nu y.f(x, y) = \nu x.f(x, x) = \nu y.\nu x.f(x, y).$$

9.3 Odsekovno linearne funkcije

Ker se ukvarjamo z odsekovno linearnimi funkcijami, si oglejmo njihovo formalno definicijo. Beseda "odsekovno" že nakazuje, da bomo domeno razrezali na kose in na vsakega dali linearni predpis. Funkcija ni nujno zvezna, kot vidiomo na enodimenzionalnem primeru na sliki 14. Osredotočamo se na domeno $[0, 1]^n$. Da bi lahko razložili kako odsekovno linearne funkcije podamo, si poglejmo naslednje definicije.



Slika 14: Nezvezna monotona odsekovno linearna funkcija na $[0,1]$. Odprt interval je nakazan s puščico, saj ima funkcija enolično vrednost za vsak element domene.

Definicija 9.9. *Linearni izraz* v spremenljivkah $x_1, x_2, \ldots, x_n$ je izraz oblike

$$q_0 + q_1 x_1 + q_2 x_2 + \ldots + q_n x_n,$$

kjer so $q_0, \ldots, q_n$ realna števila. Pravimo, da je linearni izraz *racionalen* če so $q_0, \ldots, q_n$ racionalna števila.

Ker večinoma delamo z racionalnimi linearnimi izrazi, bomo besedo "racionalni" opuščali. Če pišemo $e(x_1, \ldots, x_n)$ to označuje linearni izraz v naslednjih spremenljivkah: $x_1, \ldots, x_n$. Linearni izrazi so zaprti za substitucijo.

Definicija 9.10. *Pogojnostni linearni izraz* je par $C \vdash e$, kjer je e linearni izraz in C končna množica strogih ali nestrogih neenakosti med linearnimi izrazi, tj. vsak element v C je ene od oblik

$$e_1 \leq e_2, \qquad e_1 < e_2. \tag{9.1}$$

Množici C v pogojnostnem linearnem izrazu pravimo tudi množica omejitev ali množica neenakosti.

Oznaka $C(\vec{r})$ pomeni konjunkcijo neenakosti, kjer zamenjamo spremenljivke v C z realnimi vrednostmi $r_1, \ldots, r_n$. Pogojnostni linearni izraz $C \vdash e$ predstavlja en kos odsekovno linearne funkcije. Domena kosa je množica vseh vektorjev $\vec{r}$, ki zadoščajo $C(\vec{r})$. Linearni izraz e podaja linearno funkcijo nad to domeno. Tako definirane domene so vedno konveksne, kar vidimo v dokazu trditve 3.3. Sedaj smo opremljeni, da podamo formalno definicijo odsekovno linearne funkcije

Definicija 9.11. Funkcija $f\colon [0,1]^n \to [0,1]$ je *odsekovno linearna*, če obstaja končna množica $\mathcal{F}$ pogojnostnih linearnih izrazov v spremenljivkah $x_1 \ldots, x_n$, da sta izpolnjena naslednja pogoja:

1. Za vse $\vec{r} \in [0,1]^n$, obstaja pogojnostni linearni izraz $(C \vdash e) \in \mathcal{F}$, da velja $C(\vec{r})$ in

2. za vse $\vec{r} \in [0,1]^n$ in vsak pogojnostni linearni izraz $(C \vdash e) \in \mathcal{F}$ velja, da če $C(\vec{r})$ velja, potem je $f(\vec{r}) = e(\vec{r})$.

Tedaj pravimo, da $\mathcal{F}$ *predstavlja* f.

Zavedajmo se, da dva pogojnostna linearna izraza $(C_1 \vdash e_1) \in \mathcal{F}$ in $(C_2 \vdash e_2) \in \mathcal{F}$ nimata nujno disjunktnih domen, a se morata e_1 in e_2 strinjati na morebitnem prekrivanju.

Primer 9.12. Zopet si oglejmo enodimezionalen primer na sliki 14. Funkcija je podana preko sledečih pogojnostnih linearnih izrazov:

$$\begin{aligned} 0 \leq x < \frac{1}{3} &\vdash \frac{1}{2}x + \frac{1}{3} \\ \frac{1}{3} \leq x < \frac{2}{3} &\vdash \frac{1}{4}x + \frac{7}{12} \\ \frac{2}{3} \leq x \leq 1 &\vdash \frac{5}{6}. \end{aligned}$$

Funkcija je nezvezna in ima tri kose.

V razdelku 3.2 smo dokazali, da lahko vsako monotono odsekovno linearno funkcijo zapišemo kot formulo v predikatni teoriji linearne aritmetike. Vendar pa sta oba pristopa, tako s formulo kot s pogonjonstnimi linearnimi izrazi, zelo prostorsko in časovno zahtevna. Zato je praktično predstaviti monotono odsekovno linearno funkcijo preko lokalnega algoritma, ki, če dobi neko točko v domeni, vrne primeren pogojnostni linearni izraz.

Definicija 9.13. *Lokalni algoritem* za odsekovno linearno funkcijo $f\colon [0,1]^n \to [0,1]$ je algoritem, ki za $r_1, \ldots r_n \in [0,1]$ vrne pogojnostni linearni izraz $C \vdash e$, za katerega velja

1. $C(r_1, \ldots, r_n)$ drži in
2. če za poljubne $s_1, \ldots s_n \in [0,1]$ velja $C(s_1, \ldots, s_n)$, potem je $f(s_1, \ldots s_n) = e(s_1, \ldots, s_n)$.

Še več, le končno mnogo različnih $C \vdash e$ lahko algoritem vrne.

Dobro je opaziti, da lahko eksplicitno reprezentacijo f zlahka spremenimo v lokalni algoritem. Vendar pa obstajajo primeri, kjer je reprezentacija preko lokalnega algoritma bistveno bolj učinkovita (glej razdelek 5 o μ-termih).

9.4 Iterativni algoritem na primeru

Sedaj si oglejmo iterativni algoritem za računanje fiksnih točk monotonih odsekovno linearnih funkcij. Naj bo $f\colon [0,1]^{n+1} \to [0,1]$ odsekovno linearna funkcija, ki je monotona v zadnji spremenljivki. Ker sta po principu simetrije pojma najmanjše in največje fiksne točke dualna, se bomo osredotočili zgolj na najmanjše fiksne točke.

En od načinov za izračun fiksne točke bi bil, da pretvorimo funkcijo

$$\mu x_{n+1}.f(x_1, \ldots, x_n, x_{n+1})$$

v formulo iz predikatne teorije lineatne aritmetike uporabimo eliminacijo kvantifikatorjev [6]. Vendar pa je ta princip znan kot težek problem in v tem primeru precej

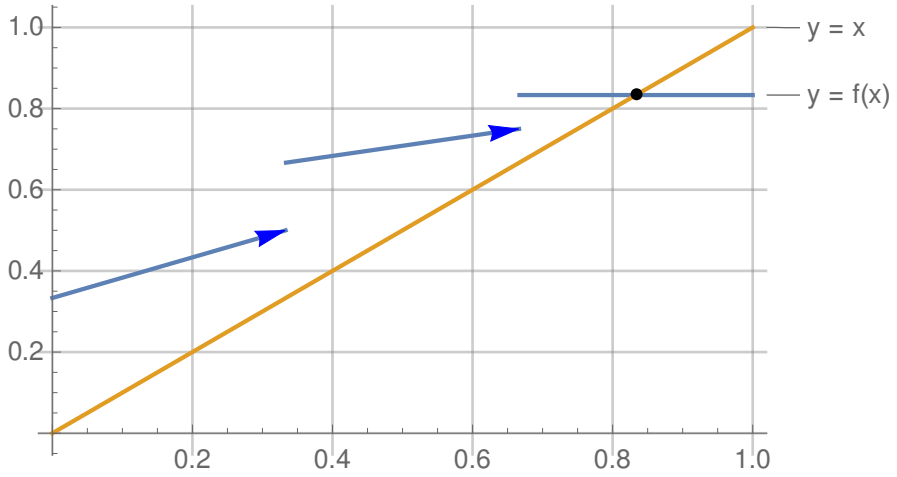
upočasni postopek izračuna. Zato predstavljamo alternativo, kjer namesto, da bi izračunali celotno družino pogojnostnih linearnih izrazov, algoritem deluje lokalno in priskrbi en sam pogojnostni linearni izraz za dani vektor $\vec{r}$. Ta lokalnost nam omogoča, da z odsekovno linearnimi funkcijami delamo preko lokalnih algoritmov, kot v definiciji 9.13.

Ker je algoritem precej tehničen in je v celoti opisan v razdelku 4, kjer je tudi dokazana pravilnost njegove različice, bomo na tem mestu le ilustrirali idejo algoritma na enodimenzionalnem primeru in komentirali razlike med originalno verzijo iz [13] in modificirano verzijo, ki je podana v razdelku 4.2.1.

Sledimo korakom modificirane različice algoritma iz razdelka 4.2.1. Fiksne točke računamo z iteracijo čez njihove približke. Začnemo z 0 za najmanjšo fiksno točko in z 1 za največjo fiksno točko. Naj bo funkcija $f\colon [0,1] \to [0,1]$ iz primera 9.12 podana z naslednjimi pogojnostnimi linearnimi izrazi:

$$\begin{aligned} 0 \leq x < \frac{1}{3} &\vdash \frac{1}{2}x + \frac{1}{3} \\ \frac{1}{3} \leq x < \frac{2}{3} &\vdash \frac{1}{4}x + \frac{7}{12} \\ \frac{2}{3} \leq x \leq 1 &\vdash \frac{5}{6}. \end{aligned}$$

Graf funkcije f in diagonale si lahko ogledamo na sliki 15.



Slika 15: Graf funkcije f in diagonale.

Očitno je f monotona in ima enolično fiksno točko v $\frac{5}{6}$.

Algoritem računa fiksno točko z iterativnim popravkom aproksimacije d. Pričnemo z $d = 0$ in najdemo linearen kos za $x = 0$, ki je podan s pogojnostnim linearnim izrazom $0 \leq x < \frac{1}{3} \vdash \frac{1}{2}x + \frac{1}{3}$. Enolična rešitev enačbe $x = \frac{1}{2}x + \frac{1}{3}$ je $x = \frac{2}{3}$, kar sega izven domene tega kosa. Zato zamenjamo d z novim približkom, ki ga dobimo iz $\frac{1}{2}x + \frac{1}{3}$, ko izračunamo v zgornji meji domene $x = \frac{1}{3}$. Torej je naslednja aproksimacija fiksne točke enaka $d = \frac{1}{2}$.

Sedaj pogledamo, kateremu kosu pripada $x = \frac{1}{2}$, kar je $\frac{1}{3} \leq x < \frac{2}{3} \vdash \frac{1}{4}x + \frac{7}{12}$. Rešitev enačbe $x = \frac{1}{4}x + \frac{7}{12}$ je $x = \frac{7}{9}$, kar zopet pogleda ven iz domene trenutnega

kosa. Torej najdemo naslednji približek z izračunom $\frac{1}{4}x + \frac{7}{12}$ v točki $x = \frac{2}{3}$ in dobimo $d = \frac{3}{4}$.

Nazadnje si ogledamo še zadnji kos domene za $x = \frac{3}{4}$, ki je podan z $\frac{2}{3} \leq x \leq 1 \vdash \frac{5}{6}$. Dobimo kandidata za fiksno točko $x = \frac{5}{6}$, ki pa tokrat je v domeni trenutnega linearnega kosa in tako $x = \frac{5}{6}$ proglasimo za najmanjšo fiksno točko funkcije f.

Čeprav smo na tem primeru pogledali vse kose domene, v splošnem temu ni tako. Splošen algoritem za iskanje fiksnih točk monotonih odsekovno linearnih funkcij z n argumenti je bistveno bolj kompliciran, saj iščemo fiksne točke linearnih izrazov in ne le števil.

9.5 Primerjava različic algoritma

Modificirana verzija algoritma se od originalne verzije iz [13] razlikuje v tem, da bolj spretno posodablja množice omejitev v pogojnostnih linearnih izrazih in jih tako ohranja manjše. Izkaže se, da prevelike množice neenakosti predstavljajo problem, zato smo razmišljali o tem, kako bi jih še bolj zmanjšali.

Na množice omejitev iz pogojnostnih linearnih izrazov lahko gledamo kot na predstavitev "posplošenih" konveksnih politopov [7], saj niso nujno zaprti zaradi strogih neenakosti. Če bi množice omejitev vsebovale le stroge neenakosti, bi imeli polno-dimenzionalne zaprte konveksne politope, za katere vemo, da imajo najmanjšo reprezentacijo preko lic (za vsako lice potrebujemo natanko eno neenakost) [7]. Dejstvo, da imamo opravka s posplošitvami politopov, nam daje upanje, da je veliko neenakosti v množici omejitev nepotrebnih.

Tega problema se lotimo preko eliminacije kvantifikatorjev [14]. Najprej uvedemo terminologijo, ki jo potrebujemo, da opišemo želene lastnosti.

Definicija 9.14. Naj bo $C(x_1, \ldots, x_n)$ množica linearnih neenakosti iz pogojnostnega linearnega izraza.

- Pravimo, da je neenakost $c \in C$ *nepotrebna* za množico omejitev C, če za vse $\vec{s} \in \mathbb{R}^n$ velja, da
$$(C \setminus c)(\vec{s}) \implies c(\vec{s}).$$
- Množica linearnih neenakosti $E(x_1, \ldots, x_n)$ je *ekvivalentna* množici C, če za vse $\vec{s} \in \mathbb{R}^n$ velja $C(\vec{s})$, če in samo če velja $E(\vec{s})$.

Za množico omejitev $C(x_1, \ldots, x_n)$ zgradimo njej ekvivalentno množico omejitev $E(x_1, \ldots, x_n)$ tako, da neenakosti dodajamo eno-po-eno in preverjamo, ali so nepotrebne. Denimo, da imamo na neki točki množico $E = \{e_1, e_2, \ldots, e_N\}$ in bi radi dodali novo neenakost $c \in C$ v E. Ta neenakost je nepotrebna, če dodajanje njene negacije pomeni, da dobimo prazno domeno, tj. če je formula

$$\exists x_1 . \exists x_2 . \ldots . \exists x_n . \left(\bigwedge_{1 \leq i \leq N} e_i(x_1, \ldots, x_n) \right) \wedge (\neg c(x_1, \ldots, x_n))$$

neresnična.

Ker je to zaprta formula v teoriji linearne aritmetike, lahko uporabimo eliminacijo kvantifikatorjev za izračun njene logične vrednosti. Če pa je c potrebna neenakost

in jo dodamo k obstoječi (minimalni) množici E, se lahko zgodi, da postanejo katere od neenakosti $e \in E$ nepotrebne, saj lahko sledijo iz konjunkcije neenakosti c in še nekaterih drugih neenakosti iz E. Da bi ohranili minimalnost množice E, moramo za vse neenakosti $e \in E$ preveriti, ali so nepotrebne za množico $E \cup \{c\}$.

Ta postopek je podrobno opisan v razdelku 4.3. Tam je tudi dokazan nalsednji izrek, ki zagotavlja, da nam postopek res vrne v nekem smislu optimalno množico neenakosti.

Izrek 9.15. *Naj bo $C(x_1, \ldots, x_n)$ množica omejitev. Zgornji postopek vrne podmnožico neenakosti $E(x_1, \ldots, x_n) \subseteq C(x_1, \ldots, x_n)$ z naslednjima lastnostma:*

1. *Množici omejitev C in E sta ekvivalentni, tj. za vsak $\vec{s} \in \mathbb{R}^n$, $C(\vec{s})$ drži če in samo če velja $E(\vec{s})$.*

2. *Vsaka neenakots $c \in E$ je potrebna v množici E, tj. obstaja vektor $\vec{s} \in \mathbb{R}^n$,da $(E \setminus c)(\vec{s})$ drži, vendar $c(\vec{s})$ ne drži. Ekvivalentno to pomeni, da v množici E ni nepotrebnih neenakosti.*

Torej smo si zagotovili minimalnost množice E v smislu, da v njej ni nepotrebnih neenakosti. Takšnih množic E lahko iz neke množice C dobimo več in prav vse ustrezajo kriterijem. Vprašanje pa ostaja, ali so vse enake kardinalnosti?

Vprašanje 9.16. *Naj bo $C(x_1, \ldots, x_n)$ množica omejitev. Ali zgornji algoritem vedno vrne minimalno (po kardinalnosti) ekvivalentno podmnožico omejitev*

$$E(x_1, \ldots, x_n) \subseteq C(x_1, \ldots, x_n),$$

tj. če obstaja še ena podmnožica $F \subseteq C$, ki je ekvivalentna množici C, ali potem sledi $|E| \leq |F|$?

Dopuščamo možnost, da je to vprašanje že kje rešeno in žal odgovora le mi nismo našli.

Tako se torej spopadamo s štirimi različnimi verzijami algoritma: originalna verzija iz članka [13], modificirana verzija iz razdelka 4.2.1 ter obe navedeni verziji opremljeni z optimizacijo preko eliminacije kvantifikatorjev. Vse te verzije smo eksperimentalno testirali z implementacijo.

9.6 Implementacija in glavni rezultati

Celotna implementacija algoritmov v pythonu 3 in rezultati testiranj so originalen prispevek. Programska koda je v polnosti dostopna na repozitoriju `https://bitbucket.org/AnjaPetkovic/evaulating-mu-terms`. Proporočamo, da preučite datoteko README.md preden zaženete kodo, saj se tam nahajajo kratka navodila in specifikacije.

Implementacija ima tri glavne komponente: generiranje primerov, programska koda za izvajanje algoritmov (implementacija je narejena preko μ-termov, glej razdelek 5) in testiranje z analizo rezultatov. Pravilnost programov za izvajanje algoritmov smo med drugim preverili tako, da smo zaporedne pojavitve fiksne točke istega tipa (npr. samo μ in ne ν) nadomestili v skladu z zlato lemo μ-računa 9.8 in preverili, da poenostavljena funkcija in prvotna funkcija vrneta isti rezultat.

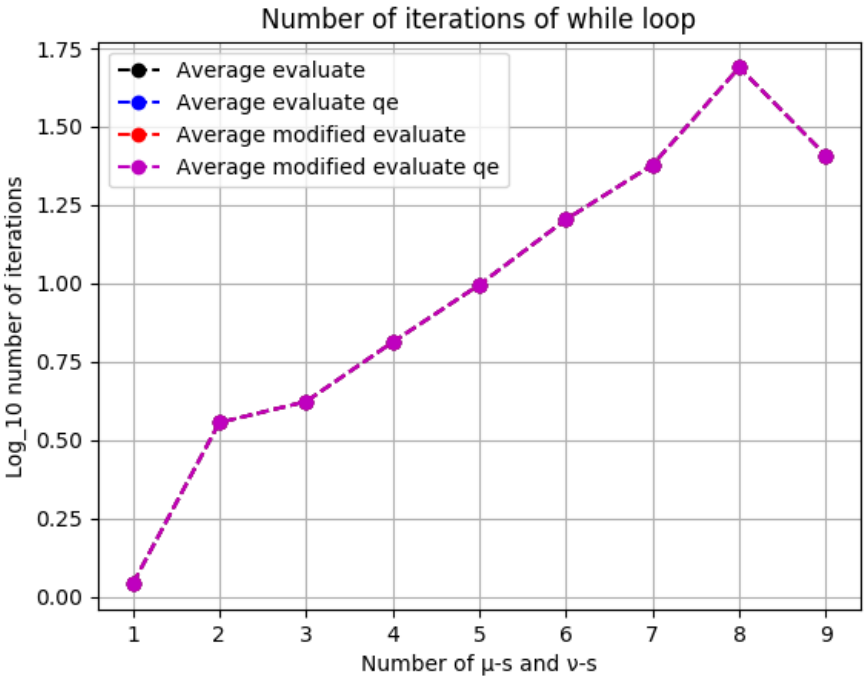
Najbolj zahteven del je seveda, ko gnezdimo fiksne točke eno za drugo, npr. $\mu x.\nu y.\mu z.f(x,y,z)$. Zanima nas obnašanje algoritmov v odvisnosti od števila gnezdenih fiksnih točk, ki se zaradi počasnosti algoritmov razpenja le med 0 in 9. Za vsako število fiksnih točk, smo vse štiri algoritme smo pognali na 10 naključno generiranih primerih in pri tem merili naslednje komponente:

- število iteracij (ponovitev zanke while), ko iteriramo preko približkov za fiksno točko,
- dejanski čas, ki ga računalnik porabi za izračun (v sekundah),
- velikost največje množice omejitev, ki jo srečamo med izračunom.

Prvi dve lastnosti merita, kako zelo počasen je proces, tretja lastnost pa meri prostorske zahteve. Pričakujemo, da sta ta dva pristopa korelirana, saj velike množice omejitev vzamejo veliko časa za procesiranje.

Rezultate smo prikazali na grafih v logaritemski skali, da demonstrirajo eksponentno rast.

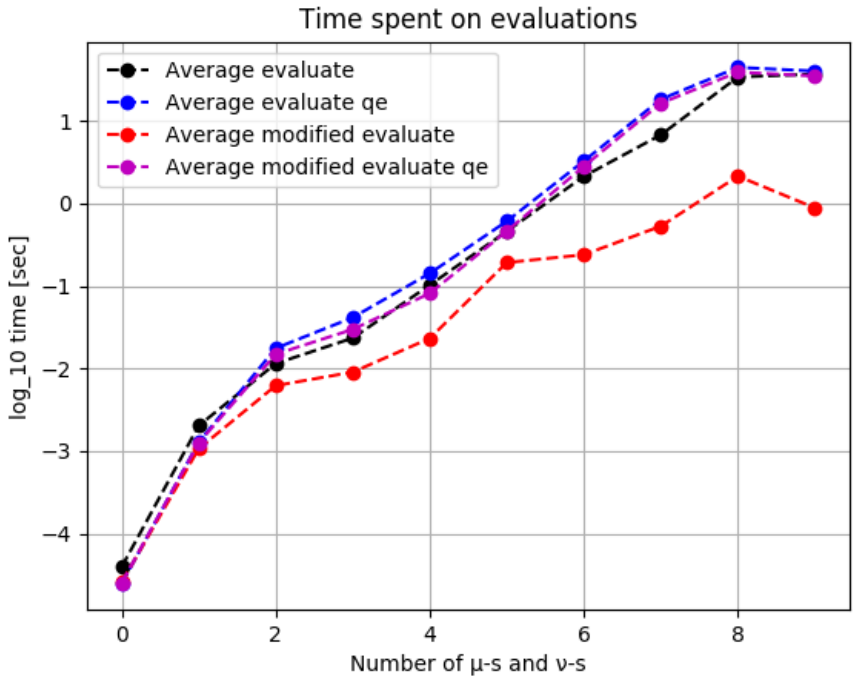
Na sliki 16 vidimo, koliko iteracij program povprečno potrebuje za izračun. Obstoj le ene črte nam pove, da vse štiri verzije algoritma potrebujejo natanko isto število iteracij. To še ne pomeni, da bodo algoritmi vedno delovali s povsem enakim številom iteracij. V algoritmih je dovolj nedeterminizma, da je povsem verjetno, da ni vedno tako. Le nam ni uspelo najti primera, kjer bi se to zgodilo.



Slika 16: Graf povprečja števila iteracij.

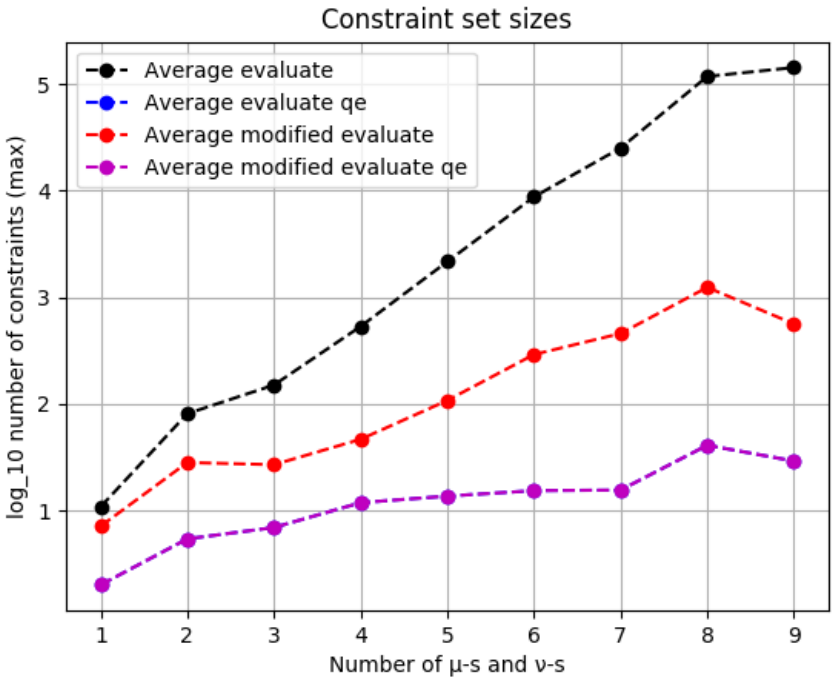
Graf na sliki 17 prikazuje povprečen čas, ki ga posamezna verzija algoritma porabi za izračun. Nad rezultatom smo nekoliko presenečeni, saj pričakujemo, da

bodo algoritmi, ki izvajajo eliminacijo kvantifikatorjev delovali hitreje, saj pregledujejo manjše množice. Vseeno pa je redukcija s pomočjo zahtevnega postopka eliminacije kvantifikatorjev tako draga, da nas stane ves privarčevani čas pri pregledovanju majnhih množic omejitev. Tako je modificirana verzija algoritma dejansko najhitrejša.



Slika 17: Graf povprečnega časa, ki ga posamezen algoritem porabi za izračun.

Za konec pa si oglejmo še graf na sliki 18, ki prikazuje velikosti množic omejitev. Opazimo, da obe verziji algoritma, ki uporabljata eliminacijo kvantifikatorjev, data identične rezultate, kar pomeni, da so množice omejitev enako velike, kar spodbuja naše sume, da ima vprašanje 9.16 pritrdilni odgovor. Po pričakovanjih ima največje množice omejitev originalni algoritem iz [13], nato mu sledi modificirana verzija in na koncu še optimizacija preko eliminacije kvantifikatorjev. Za boljši občutek o dejanskih velikostih (in posledično ogromnih razlikah med učinkovitostjo algoritmov), lahko preverimo, da je med testiranjem za 9 fiksnih točk največja množica omejitev za originalen algoritem vsebovala 409385 neenakosti, za modificiran algoritem 2208 neenakosti in za algoritem z optimizacijo preko eliminacije kvantifikatorjev le 62 neenakosti.



Slika 18: Graf povprečne največje velikosti množice omejitev.

References

[1] A. Arnold in D. Niwinsky, *Rudiments of μ-calculus*, Studies in Logic and the Foundations of Mathematics **146**, Elsevier, Amsterdam, 2001.

[2] C. Baier in J.-P. Katoen, *Principles of Model Checking*, The MIT Press, 2008.

[3] Barendregt, *The Lambda Calculus, Its Syntax and Semantics*, Studies in Logic and the Foundations of Mathematics **103**, Elsevier, North-Holland, 1985.

[4] A. Condon, *The complexity of stochastic games*, Information and Computation **96** (1992) 203–224.

[5] *Convex set*, [ogled 30. 8. 2017], dostopno na `https://en.wikipedia.org/wiki/Convex_set`.

[6] J. Ferrante in C. Rackoff, *A decision procedure for the first order theory of real addition with order*, SIAM Journal of Computing **4 (1)** (1975) 69 – 76.

[7] B. Grünbaum, *Convex Polytopes*, Graduate Texts in Mathematics **221**, Springer-Verlag New York, New York, 2003.

[8] P. Hajek, *Mathematics of Fuzzy Logic*, Trends in Logic - Studia Logica Library **4**, Kluwer academic publishers, Prague, 2001.

[9] I. Kaplansky, *Set Theory and Metric Spaces*, AMS Chelsea Publishing, American Mathematical Society; 2nd edition, England, 2001.

[10] B. L., S. M in S. S., *On a theory of computation and complexity over the real-numbers: Np-completeness, recursive functions and universal machines*, Bulletin of the AMS **21 (1)** (1989) 1–46.

[11] McNaughton, *A theorem about infinite-valued sentential logic*, The Journal of Symbolic Logic **16** (1951) 1–13.

[12] J. Milnor, *Analytic proofs of the "hairy ball theorem" and the brouwer fixed-point theorem*, Amer. Math. Monthly 85 **7** (1978) 521–524.

[13] M. Mio in A. Simpson, *Łukasiewicz μ-calculus*, Fundamenta Informaticae **150** (2017) 1–30.

[14] T. Nipkow, *Linear quantifier elimination*, Journal of Automated Reasoning **45**(2) (2010) 189–212, doi:10.1007/s10817-010-9183-0.