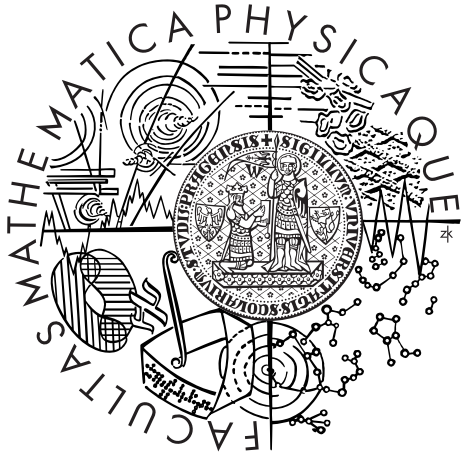
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Elif Garajová

Intervalový solver nelineárních podmínek

Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Milan Hladík, Ph.D.
Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2014

Rada by som predovšetkým poďakovala vedúcemu mojej práce Mgr. Milanovi Hladíkovi, Ph.D. za jeho ochotu a cenné rady. Ďalej ďakujem mojej rodine a priateľom za podporu počas štúdia. Špeciálne poďakovanie patrí Ing. Miroslavovi Radovi za jeho trpezlivosť a podnetné pripomienky pri tvorbe tejto práce.

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Intervalový solver nelineárních podmínek

Autor: Elif Garajová

Katedra: Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Milan Hladík, Ph.D., Katedra aplikované matematiky

Abstrakt: Práce se zabývá algoritmem Sivia (Set Inverter via Interval Analysis) pro řešení spojitého problému splňování omezujících podmínek pomocí intervalových metod a propagačních technik. Prezentovány jsou základní vlastnosti algoritmu, a také oprava uváděného odhadu jeho složitosti. Dále jsou navrhnuta vylepšení týkající se testování splnění omezujících podmínek a optimalizace počtu intervalových boxů popisujících získané řešení problému. Představeny jsou též tzv. kontraktory používané pro zvýšení efektivity algoritmu Sivia prostřednictvím redukce zpracovávaných intervalových boxů. Uvedené algoritmy byly implementované jako knihovna funkcí jazyka Matlab tvořící solver nelineárních podmínek s možností jednoduché vizualizace získaného řešení. Součástí práce je také porovnání základních kontraktorů na konkrétních příkladech.

Klíčová slova: problém splňování omezujících podmínek, intervalová data, propagační techniky, Sivia

Title: Interval solver for nonlinear constraints

Author: Elif Garajová

Department: Department of Applied Mathematics

Supervisor: Mgr. Milan Hladík, Ph.D., Department of Applied Mathematics

Abstract: The thesis is focused on the Sivia algorithm (Set Inverter via Interval Analysis) designed for solving a continuous constraint satisfaction problem using interval methods and propagation techniques. Basic properties of the algorithm are derived, including the correction of its presented complexity bound. Some improvements concerning the testing of constraint satisfaction and optimization of the number of interval boxes describing the solution are proposed. The thesis also introduces contractors used to enhance the effectivity of the Sivia algorithm by reducing the interval boxes processed. Presented algorithms were implemented in a solver for nonlinear constraints with a simple visualization of the result using the Matlab language. A comparison of basic contractors on specific examples is given.

Keywords: constraint satisfaction problem, interval data, propagation techniques, Sivia

Názov práce: Intervalový solver nelineárních podmínek

Autor: Elif Garajová

Katedra: Katedra aplikované matematiky

Vedúci bakalárskej práce: Mgr. Milan Hladík, Ph.D., Katedra aplikované matematiky

Abstrakt: Práca sa zaoberá algoritmom Sivia pre riešenie spojitého problému splňovania obmedzujúcich podmienok pomocou intervalových metód a propagačných techník. Prezentované sú základné vlastnosti algoritmu, ako aj oprava uvádzaného odhadu jeho zložitosti. Ďalej sú navrhnuté vylepšenia týkajúce sa testovania splnenia obmedzujúcich podmienok a optimalizácie počtu intervalových boxov popisujúcich získané riešenie problému. Predstavené sú tiež tzv. kontraktory používané pre zvýšenie efektivity algoritmu Sivia prostredníctvom redukcie spracovávaných intervalových boxov. Uvedené algoritmy boli implementované ako knižnica funkcií jazyka Matlab tvoriaca solver nelineárnych podmienok s možnosťou jednoduchej vizualizácie získaného riešenia. Súčasťou práce je tiež porovnanie základných kontraktorov na konkrétnych príkladoch.

Kľúčové slová: problém splňovania obmedzujúcich podmienok, intervalové dáta, propagačné techniky, Sivia

Obsah

1 Úvod

1.1 Motivácia

S rýchlym rozvojom počítačov sa čoraz častejšie stretávame s ich využitím v rôznych vedných oblastiach. Súčasná technika umožňuje rýchlejšie a presnejšie vyhodnocovanie náročných výpočtov, a tak vznikajú stále nové algoritmy pre riešenie praktických, ale i teoretických problémov.

Počítačová aritmetika má však proti reálnej stále jeden veľký nedostatok, a tým je obmedzená presnosť. Nemožnosť reprezentovania čísel ich presným tvarom vedie k zaokrúhľovacím chybám, ktoré sa môžu v priebehu výpočtu zväčšovať a spôsobiť tak, že získaný výsledok bude ďaleko od reálneho riešenia skúmaného problému. Tento problém sa pritom netýka iba iracionálnych čísel – počítačová aritmetika totiž pre reprezentáciu čísel používa binárnu sústavu a mnohé racionálne čísla po prevode získajú nekonečný binárny rozvoj. Ak chceme mať istotu, že získaný výsledok bude v určitom zmysle dobrou aproximáciou skutočného riešenia problému, musíme použiť iný prístup.

Jedným z možných riešení tohto nedostatku je využitie intervalov. Reálne číslo tak nebudeme reprezentovať zaokrúhlenou hodnotou, ale intervalom, v ktorom sa dané číslo určite nachádza. Na reprezentáciu hraníc tohto intervalu už obmedzená presnosť stačí. Rovnako prispôsobíme aritmetické operácie a funkcie tak, aby boli vo výsledku zahrnuté všetky požadované riešenia.

Počiatky intervalovej analýzy sa datujú do druhej polovice 20. storočia, rozšírila sa však najmä v posledných dvoch desaťročiach. Okrem umožnenia reprezentácie reálnych čísel v aritmetike s obmedzenou presnosťou našli intervaly svoje miesto aj pri modelovaní problémov s nepresnými vstupnými dátami. V praxi často nedokážeme požadované hodnoty určiť presne, ale vieme nájsť medze, v ktorých sa pohybujú. Miesto odhadu presnej hodnoty tak môžeme použiť interval, ktorý nameranú hodnotu s istotou obsahuje.

Jednou z oblastí využitia intervalových metód je riešenie problémov, ktoré dokážeme modelovať pomocou nelineárnych rovníc a nerovníc nad premennými nadobúdajúcimi hodnoty z daných intervalov. Intervalová aritmetika nám umožňuje vytvoriť vnútorný a vonkajší odhad takto popísanej množiny. Získaný výsledok teda nie je iba akýmsi nepresným odhadom riešenia problému, ale s určitosťou obsahuje všetky hodnoty, ktoré do samotnej množiny riešení patria.

1.2 Cieľ práce

Cieľom tejto práce je implementovať intervalový solver spojitých problémov splňovania obmedzujúcich podmienok v prostredí MATLABu. Obmedzujúce podmienky riešené v tejto úlohe sú reprezentované formou nelineárnych rovníc a nerovníc alebo ako tzv. testovacie funkcie.

Základný algoritmus bude vylepšený pomocou propagačných techník vo forme intervalových kontraktorov pre efektívnejšie odstránenie bodov nepatriacich do množiny riešení daného problému. Ďalej navrhneme metódu pre zníženie počtu intervalových boxov generovaných implementovaným algoritmom.

1.3 Súvisiace práce

Na úvod predstavíme niektoré existujúce programy riešiace problematiku splňovania nelineárnych obmedzujúcich podmienok so spojitými premennými. Tieto balíky funkcií využívajú prevažne metódy intervalovej analýzy, avšak niektoré z nich prinášajú tiež rôzne techniky z iných oblastí zamerané na riešenie špecifickejších problémov.

Alias Knižnica Alias (An Algorithms Library of Interval Analysis for equation Systems, Merlet (2004)) je súbor algoritmov založených na intervalovej analýze implementovaných v jazyku C++ s rozhraním pre použitie v Maple. Knižnica je vyvíjaná v rámci projektu COPRIN a zameriava sa na optimalizačné problémy a riešenie sústav rovníc a nerovníc. Okrem obecných algoritmov poskytuje tiež niektoré špecializované metódy, napríklad pre riešenie trigonometrických rovníc alebo výpočet koreňov polynómov v jednej premennej.

Quimper Balík aplikácií Quimper (QUick Interval Modeling and Programming in a bounded-ERror context, Chabert – Jaulin (2009)) zahŕňa nástroj pre riešenie spojitých problémov s obmedzujúcimi podmienkami s možnosťou numerickej alebo grafickej reprezentácie výsledku založený na intervalovej knižnici Ibex pre jazyk C++. Prináša tiež vlastný jazyk pre popis problému a požadovaného spôsobu jeho riešenia. Časť balíku je zameraná na problémy z oblasti riešenia diferenciálnych rovníc.

RealPaver Softvérový balík RealPaver (Granvilliers – Benhamou, 2006) je nástroj pre modelovanie a riešenie optimalizačných problémov a problémov s nelineárnymi podmienkami. Balík je implementovaný v jazyku `C` a prostredníctvom kombinácie techník intervalovej analýzy a splňovania obmedzujúcich podmienok poskytuje aproximáciu množiny riešení problému pomocou intervalových boxov. Intervalovú aritmetiku pre RealPaver zabezpečuje knižnica Gaol.

RSolver Balík RSolver (Ratschan, 2006) je zameraný na riešenie problémov s kvantifikovanými obmedzujúcimi podmienkami, teda obmedzujúcimi podmienkami bližšie špecifikovanými pomocou existenčných a univerzálnych kvantifikátorov. RSolver je implementovaný v jazyku OCaml s podporou knižnice smathlib. Balík tiež obsahuje samostatné grafické rozhranie pre vizualizáciu získaných výsledkov.

Mnohé z existujúcich programov pre riešenie problémov splňovania obmedzujúcich podmienok fungujú ako samostatné aplikácie. Pre tvorbu solveru vytvoreného v tejto práci bol zvolený jazyk MATLAB s cieľom umožniť použitie programu priamo z výpočtového prostredia MATLAB, ktoré má v komunite zaoberajúcej sa intervalovou analýzou široké zastúpenie. Pre podporu intervalovej aritmetiky je použitá sada nástrojov Intlab umožňujúca garantované intervalové výpočty.

Kvôli možnosti širokého využitia programu sme sa zamerali na implementáciu algoritmov a techník pracujúcich s obecnými nelineárnymi podmienkami bez požiadavkov na tvar zadanej úlohy. Pre lepšiu predstavu o tvare a vlastnostiach

množiny riešení daného problému sú súčasťou solveru aj základné vizualizačné funkcie. V rámci tejto práce sme tiež navrhli niektoré vylepšenia základného implementovaného algoritmu Sivia, bližšie popísané v sekcii 3.6. Ďalšie rozšírenie solveru prináša využitie algoritmu Sivia pre vizualizáciu presných aritmetických operácií na množine komplexných intervalov.

1.4 Štruktúra práce

Teoretická časť práce je venovaná popisu implementovaných algoritmov a ich vlastností, ako aj potrebnému teoretickému základu. Súčasťou práce je užívateľská dokumentácia, v ktorej je bližšie popísané spustenie balíku a používanie jednotlivých funkcií. V programátorskej dokumentácii poskytujeme úvod do práce s tzv. Mex-súbormi, ktoré umožňujú prepojiť zdrojový kód napísaný v jazyku C++ s funkciami používanými v MATLABe.

V nasledujúcej kapitole definujeme základné pojmy intervalovej analýzy, poskytneme úvod do reálnej intervalovej aritmetiky a popíšeme problém závislosti vyplývajúci z jej používania. Uvedieme tiež definíciu problému splňovania obmedzujúcich podmienok a s ním súvisiacich pojmov. Tretia kapitola je venovaná analýze riešeného problému a popisu algoritmu Sivia, ktorý ho rieši. Ďalej odvodíme niektoré základné vlastnosti tohto algoritmu a navrhneme možnosti jeho vylepšenia. Štvrtá kapitola uvádza rôzne druhy techník, na ktorých sú založené kontraktory používané pre redukciu intervalových boxov, s ktorými algoritmus Sivia pracuje. Predstavíme štyri typy kontraktorov s rôznymi možnosťami ich implementácie a zavedieme pojem lokálnej konzistencie, na základe ktorého je možné teoretické porovnanie vlastností týchto kontraktorov. Praktické porovnanie implementovaných verzií kontraktorov poskytne kapitola 5. Poslednú kapitolu tvorí úvod do komplexnej intervalovej aritmetiky a zhrnutie rôznych prístupov k definícii komplexného intervalu. Na záver uvedieme možnosti vizualizácie komplexnej intervalovej aritmetiky prevodom na problém splňovania obmedzujúcich podmienok.

2 Základné pojmy

Kapitola najprv definuje základné pojmy súvisiace s intervalmi. Ďalej je zavedená reálna intervalová aritmetika a intervalové analógie reálnych funkcií. Záver kapitoly je venovaný formalizácii problému splňovania obmedzujúcich podmienok.

2.1 Intervaly a intervalová aritmetika

2.1.1 Interval a súvisiace pojmy

V úvode tejto sekcie definujeme reálne intervaly, ich charakteristiky a príslušné značenie. Pokiaľ nebude uvedené inak, bude pojem „interval“ označovať práve množinu vymedzenú definíciou 2.1.

Definícia 2.1 (Reálny interval)**.** Nech $\underline{x}, \overline{x} \in \mathbb{R}$ a nech platí $\underline{x} \leq \overline{x}$. Potom množinu

$$\boldsymbol{x} = [\underline{x}, \overline{x}] := \{x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x}\}$$

nazveme *reálnym intervalom*. Hodnotu $\underline{x}$, resp. $\overline{x}$ nazývame *dolnou*, resp. *hornou hranicou* intervalu $\boldsymbol{x}$. Množinu všetkých reálnych intervalov budeme značiť symbolom $\mathbb{IR}$.

Definícia 2.2. Buď $\boldsymbol{x} \in \mathbb{IR}$. Ak platí $\underline{x} = \overline{x}$, nazveme interval *degenerovaný*.

V určitých prípadoch môže byť výhodné uvažovať aj *neobmedzené* intervaly tvaru $[\underline{x}, \infty), (-\infty, \overline{x}]$, príp. $(-\infty, \infty)$, ktoré definícia 2.1 nezahŕňa. Napríklad podmienku v tvare nerovnosti $f(x) \leq 0$ môže byť užitočné previesť pridaním doplnkovej premennej na rovnosť tvaru $f(x) + s = 0$, $s \in [0, \infty)$. Táto úprava je však možná iba v prípade, že nie je vyžadovaná obmedzenosť intervalov. Niekedy sa medzi intervaly zaraďuje aj prázdna množina, napríklad v súvislosti s operáciou intervalového prieniku.

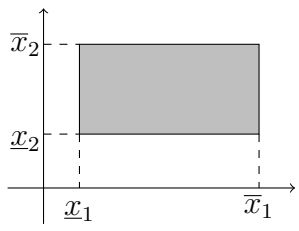
Definícia 2.3. Buď $\boldsymbol{x} \in \mathbb{IR}$, definujeme nasledujúce pojmy:

(a) *stred* intervalu $x^c := \frac{1}{2}(\overline{x} + \underline{x})$,

(b) *polomer* intervalu $x^\Delta := \frac{1}{2}(\overline{x} - \underline{x})$,

(c) *šírka* intervalu $w(\boldsymbol{x}) := \overline{x} - \underline{x}$.

Definícia 2.4 (Intervalový box)**.** Reálny *intervalový vektor* alebo *box* je karteziánskym súčinom n reálnych intervalov $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, značíme $\boldsymbol{x} = \boldsymbol{x}_1 \times \cdots \times \boldsymbol{x}_n$ alebo skrátene $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$. Množinu všetkých n-zložkových intervalových vektorov značíme $\mathbb{IR}^n$. *Objem* intervalového boxu $\boldsymbol{x}$ definujeme ako $\prod_{i=1}^{n} w(\boldsymbol{x}_i)$.



Obr. 2.1: Intervalový box v $\mathbb{IR}^2$

2.1.2 Množinové operácie

Definícia 2.5 (Množinové operácie)**.** Nech $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{IR}$, potom ich *prienik*, *zjednotenie* a *obal* definujeme nasledovne:

$$\begin{aligned}\boldsymbol{x} \cap \boldsymbol{y} &:= \{z \mid z \in \boldsymbol{x} \wedge z \in \boldsymbol{y}\} \\ \boldsymbol{x} \cup \boldsymbol{y} &:= \{z \mid z \in \boldsymbol{x} \vee z \in \boldsymbol{y}\} \\ \boldsymbol{x} \sqcup \boldsymbol{y} &:= \left[\min(\underline{x}, \underline{y}), \max(\overline{x}, \overline{y})\right]\end{aligned}$$

V prípade, že je prienik intervalov $\boldsymbol{x}$ a $\boldsymbol{y}$ neprázdny, môžeme ho ekvivalentne vyjadriť ako interval $[\max(\underline{x}, \underline{y}), \min(\overline{x}, \overline{y})]$.

Nakoľko zjednotenie dvoch intervalov $\boldsymbol{x}$ a $\boldsymbol{y}$ nemusí tvoriť súvislú množinu, využíva sa častejšie intervalový obal, ktorý je najmenším intervalom (vzhľadom k inklúzii) obsahujúcim množinu $\boldsymbol{x} \cup \boldsymbol{y}$.

2.1.3 Porovnávanie intervalov

Porovnávanie prvkov na množine reálnych intervalov zavedieme pomocou usporiadania $\leq$ a ekvivalencie $=$ na množine reálnych čísel. Dva reálne intervaly $\boldsymbol{x} = [\underline{x}, \overline{x}]$ a $\boldsymbol{y} = [\underline{y}, \overline{y}]$ *sa rovnajú* práve vtedy, keď popisujú rovnakú množinu, teda platí

$$\boldsymbol{x} = \boldsymbol{y} \;\Leftrightarrow\; \underline{x} = \underline{y} \;\wedge\; \overline{x} = \overline{y}.$$

Interval $\boldsymbol{x}$ *je menší alebo rovný* intervalu $\boldsymbol{y}$, ak pre každú dvojicu prvkov (x, y), kde $x \in \boldsymbol{x}, y \in \boldsymbol{y}$, platí $x \leq y$, zjednodušene

$$\boldsymbol{x} \leq \boldsymbol{y} \Leftrightarrow \overline{x} \leq \underline{y}.$$

Obdobne môžeme definovať reláciu $\geq$ a ostré nerovnosti.

Narozdiel od usporiadania $\leq$ na množine reálnych čísel, nie je takto definované usporiadanie na reálnych intervaloch úplné, pretože existujú neporovnateľné prvky. Napríklad pre intervaly $[0, 1]$ a $[0, 2]$ neplatí vzťah $[0, 1] \leq [0, 2]$ ani $[0, 2] \leq [0, 1]$.

2.1.4 Intervalová aritmetika

Aritmetické operácie známe napríklad z reálnych čísel je možné intuitívne rozšíriť aj na reálne intervaly. Pri počítaní s intervalmi budeme požadovať, aby výsledok zahŕňal hodnoty operácií pre všetky možné voľby prvkov z intervalov a zároveň aby výsledkom každej intervalovej operácie bol opäť interval. Spojitý obraz intervalov popisuje Veta 2.1, k zobecneniu pre intervalové boxy vedie veta 2.2.

Veta 2.1 (Rudin (1976, kap. 4))**.** *Buď $f : \mathbb{R} \to \mathbb{R}$ funkcia spojitá na intervale $\boldsymbol{x}$. Potom obraz f na $\boldsymbol{x}$ je interval.*

Veta 2.2 (Rudin (1976, kap. 4))**.** *Spojitý obraz kompaktnej množiny je kompaktná množina. Spojitý obraz súvislej množiny je súvislá množina.*

V $\mathbb{R}^n$ sú kompaktné práve uzavreté a obmedzené množiny, špeciálne teda platí veta 2.2 pre obrazy intervalových boxov. Ďalej vieme, že neprázdne súvislé kompaktné podmnožiny $\mathbb{R}$ sú práve uzavreté intervaly. Pre každú spojitú funkciu

$f : \mathbb{R}^n \to \mathbb{R}$ teda platí, že obrazom intervalového boxu je interval, môžeme preto zaviesť definíciu príslušnej intervalovej funkcie nasledovne:

$$f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \{f(x_1, \ldots, x_n) \mid \forall i \in \{1, \ldots, n\} : x_i \in \boldsymbol{x}_i\}.$$

V prípade nespojitých funkcií môžeme miesto výslednej množiny uvažovať jej *intervalový obal*, teda najmenší interval vzhľadom k inklúzii, ktorý danú množinu obsahuje.

Pre základné aritmetické operácie je možné odvodiť vzorce, ktoré popisujú výsledok príslušnej intervalovej operácie pomocou dolných a horných hraníc operandov.

$$\boldsymbol{x} + \boldsymbol{y} = \left[\underline{x} + \underline{y}, \overline{x} + \overline{y}\right]$$
$$\boldsymbol{x} - \boldsymbol{y} = \left[\underline{x} - \overline{y}, \overline{x} - \underline{y}\right]$$

Výsledné množiny pre operácie násobenia a delenia sú zložitejšie, avšak pri znalosti hraníc operandov môžeme niektoré možnosti vopred vylúčiť. Pri delení predpokladáme, že interval v menovateli neobsahuje nulu – niekedy však môže byť užitočné definovať tzv. *rozšírenú intervalovú aritmetiku*, ktorá okrem iného umožňuje aj delenie intervalom obsahujúcim 0 (pre podrobnejší popis viď napríklad Moore et al. (2009, s. 109)).

$$\boldsymbol{x} \cdot \boldsymbol{y} = [\min S, \max S] \qquad S := \{\underline{x} \cdot \underline{y}, \underline{x} \cdot \overline{y}, \overline{x} \cdot \underline{y}, \overline{x} \cdot \overline{y}\}$$
$$\frac{\boldsymbol{x}}{\boldsymbol{y}} = \boldsymbol{x} \cdot \frac{1}{\boldsymbol{y}} \qquad \frac{1}{\boldsymbol{y}} := \left[\frac{1}{\overline{y}}, \frac{1}{\underline{y}}\right], 0 \notin \boldsymbol{y}$$

Poznámka. Pri výpočtoch s obmedzenou presnosťou používame v intervalovej aritmetike tzv. *zaokrúhľovanie smerom von*, dolnú hranicu teda zaokrúhľujeme nadol a hornú hranicu nahor, aby sme zaručili, že výsledný interval bude obsahovať všetky požadované hodnoty.

Analógiu zložitejších reálnych funkcií predstavuje na množine intervalov koncept intervalového rozšírenia.

Definícia 2.6 (Intervalové rozšírenie)**.** Funkcia $[f] : \mathbb{IR}^n \to \mathbb{IR}$ je *intervalovým rozšírením* funkcie $f : \mathbb{R}^n \to \mathbb{R}$, ak je obraz $[f](\boldsymbol{x})$ intervalového vektoru $\boldsymbol{x}$ nadmnožinou $\{f(x) \mid x \in \boldsymbol{x}\}$.

Príklad. Uvažujme funkciu $f(x, y) = x + y$, potom funkcia $[f_1](\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} + \boldsymbol{y}$ využívajúca intervalovú aritmetiku spĺňa definíciu 2.6. Ak pripustíme neobmedzené intervaly, je intervalovým rozšírením f je tiež konštantná funkcia $[f_2](\boldsymbol{x}, \boldsymbol{y}) = [-\infty, \infty]$.

Intervalové rozšírenie reálnej funkcie nie je určené jednoznačne. Rozšírenie nazveme *minimálne*, ak je obrazom každého intervalového boxu najmenší možný interval. Pre monotónne spojité funkcie dokážeme získať minimálny obraz intervalu pomocou dolnej a hornej hranice vzoru. Napríklad ak je f rastúca, potom platí $f(\boldsymbol{x}) = [f(\underline{x}), f(\overline{x})]$. V obecnom prípade budeme najčastejšie používať rozšírenie, ktoré získame nahradením reálnych funkcií príslušnými intervalovými funkciami, tzv. *prirodzené* intervalové rozšírenie. Tvrdenie 2.3 popisuje jednu z jeho významných vlastností.

Tvrdenie 2.3. *Buď $[f]$ prirodzené intervalové rozšírenie funkcie $f : \mathbb{R}^n \to \mathbb{R}$ a $\boldsymbol{x} \in \mathbb{IR}^n$. Ak je $f(x_1, \ldots, x_n)$ spojitá na $\boldsymbol{x}$ a každá premenná x_i sa v analytickom predpise f vyskytuje nanajvýš raz, potom platí $f(\boldsymbol{x}) = [f](\boldsymbol{x})$.*

Dôkaz. viď Moore et al. (2009, kap. 6) □

2.1.5 Problém závislosti

Nájdenie presného obrazu intervalu pri danej reálnej funkcii je obecne NP-ťažký problém (Gaganov, 1985). Narozdiel od prípadu s unikátnym výskytom premenných, môžeme pri opakovanom výskyte premenných v predpise funkcie použitím prirodzeného rozšírenia dostať nadhodnotený obraz intervalu. Príčinou je tzv. *problém závislosti*, ktorý ilustruje nasledujúci príklad.

Príklad. Uvažujme reálnu funkciu zadanú predpisom $f(x) = x^2 - x$ a interval $\boldsymbol{x} = [-1, 1]$. Potom vyhodnotením prirodzeného intervalového rozšírenia dostaneme $[f](\boldsymbol{x}) = [-1, 1]^2 - [-1, 1] = [-1, 2]$. Úpravou predpisu funkcie na ekvivalentný tvar $g(x) = (x - \frac{1}{2})^2 - \frac{1}{4}$ a vyhodnotením prirodzeného rozšírenia $[g](\boldsymbol{x})$ však dostaneme optimálny obraz $[-\frac{1}{4}, 2]$.

Funkcie, ktoré sú v reálnej aritmetike ekvivalentné, môžu dávať pri použití intervalovej aritmetiky rôzne výsledky. Vo funkcii $[f]$ totiž vyberáme hodnotu x nezávisle pre každý výskyt premennej, do obrazu rozšírenia sa tak dostane napríklad hodnota $0^2 - 1 = -1$, ktorá ale do obrazu reálnej funkcie f nepatrí.

Poznámka. Pre intervalovú druhú mocninu $\boldsymbol{x}^2$ a súčin $\boldsymbol{x} \cdot \boldsymbol{x}$ neplatí očakávaný vzťah $\boldsymbol{x}^2 = \boldsymbol{x} \cdot \boldsymbol{x}$. V prípade súčinu totiž nie je zachytená závislosť medzi dvomi výskytmi premennej x. Podľa definície platí:

$$\boldsymbol{x}^2 = \{x^2 \mid x \in \boldsymbol{x}\},$$
$$\boldsymbol{x} \cdot \boldsymbol{x} = \{x \cdot y \mid x \in \boldsymbol{x}, y \in \boldsymbol{x}\}.$$

Podobne vzťah $\boldsymbol{x} - \boldsymbol{x} = [0, 0]$ platí iba pre degenerované intervaly.

2.2 Splňovanie obmedzujúcich podmienok

Koncept obmedzujúcich podmienok umožňuje formalizovať problémy z rôznych oblastí prírodných a technických vied, kde sa vyskytujú úlohy popisujúce množinu objektov, ktorej prvky musia spĺňať zadané podmienky. K takýmto problémom často vedie výskum napríklad v oblasti umelej inteligencie, logiky, optimalizácie, ale aj v biológii či diskrétnej matematike. Medzi známe problémy, ktoré je možné formulovať ako problémy splňovania obmedzujúcich podmienok, patrí napríklad riešenie Sudoku, farbenie mapy alebo rozvrhovanie výroby.

Obecne ide o problémy s premennými, ktoré môžu nadobúdať hodnoty z vopred určených definičných oborov. Napríklad pri riešení dobre známej logickej úlohy Sudoku tvorí jednotlivé premenné 81 polí siete a definičným oborom premennej je množina prirodzených čísel $\{1, \ldots, 9\}$. Obmedzujúcou podmienkou potom rozumieme reláciu popisujúcu požadované vzťahy medzi premennými (napríklad nerovnosť priradených hodnôt pre každú premennú v rovnakom riadku Sudoku).

Uvedený príklad spadá do množiny klasických diskrétnych problémov splňovania obmedzujúcich podmienok s konečnými definičnými obormi premenných. V praxi však často potrebujeme tiež modelovať problémy, kde môžu premenné nadobúdať hodnoty z danej obmedzenej, ale nekonečnej množiny. Jednoduchým príkladom je riešenie sústavy nelineárnych nerovníc s voľbou hodnôt premenných z intervalov reálnych čísel. Tieto typy problémov sa tiež často vyskytujú v robotike a lokalizácii (viď napr. Jaulin (2006)).

Definícia 2.7. *Problém splňovania obmedzujúcich podmienok* definujeme ako trojicu (X, D, C), kde

- $X = \{x_1, \ldots, x_n\}$ je konečná množina premenných,
- $D = \{D_1, \ldots, D_n\}$ je konečná množina definičných oborov premenných,
- $C = \{c_1, \ldots, c_m\}$ je konečná množina obmedzujúcich podmienok (relácií nad množinou premenných).

Problém splňovania obmedzujúcich podmienok sa často označuje skratkou CSP (angl. constraint satisfaction problem). Pre účely tejto práce budeme *obmedzujúcou podmienkou* rozumieť podmienku $f(x_1, \ldots, x_n) \circ 0$, kde $x_1, \ldots, x_n$ patria do množiny premenných X, $\circ$ je relácia z množiny $\{\leq, \geq, =\}$ a $f : \mathbb{R}^n \to \mathbb{R}$. Ako *definičné obory* jednotlivých premenných uvažujeme reálne intervaly. V takomto prípade hovoríme o spojitom numerickom CSP. Pokiaľ nebude uvedené inak, budeme ďalej skratka CSP označovať práve tento typ problému.

Definícia 2.8 (Riešenie CSP)**.** *Ohodnotením* premenných nazveme zobrazenie, ktoré každej premennej $x_i \in X$ priradí hodnotu z príslušného definičného oboru D_i. Ohodnotenie je *konzistentné* vzhľadom k obmedzujúcej podmienke c_j, pokiaľ ju neporušuje. *Riešením* CSP nazveme každé ohodnotenie premenných, ktoré je konzistentné vzhľadom ku každej podmienke $c \in C$.

3 Algoritmus SIVIA

V tejto kapitole popíšeme niekoľko rôznych prístupov k riešeniu spojitého problému splňovania obmedzujúcich podmienok spolu s ich výhodami a nevýhodami. Budeme sa bližšie venovať intervalovému prístupu a predstavíme algoritmus Sivia založený na tomto prístupe s niektorými jeho modifikáciami a vylepšeniami a odvodíme základné vlastnosti tohto algoritmu. Ďalej sa budeme venovať projekcii riešenia CSP popísaného pomocou množín viacrozmerných intervalových boxov do nižšej dimenzie.

3.1 Riešenie spojitého CSP

3.1.1 Prístupy k riešeniu spojitého CSP

V závislosti na požadovaných vlastnostiach získaného riešenia a požiadavkách na priebeh výpočtu je niekoľko možných prístupov k splňovaniu spojitých obmedzujúcich podmienok (Benhamou – Granvilliers, 2006).

Z pohľadu *počítačovej algebry* sú spojité obmedzujúce podmienky formule logiky prvého rádu nad reálnymi číslami. Počítačová algebra využíva algoritmy pre symbolické výpočty, pomocou ktorých sa snaží previesť zadaný problém na jednoduchší problém v rovnakej triede ekvivalencie. Presné výpočty sú garantované za podmienky použitia algebraických čísel alebo racionálnych čísel s nekonečnou presnosťou.

Programovanie s obmedzujúcimi podmienkami sa zaoberá splnením relácií medzi premennými. Proces riešenia problému prebieha rozsiahlym prehľadávaním, ktoré môže byť za účelom zvýšenia efektivity kombinované s propagačnými technikami zabezpečujúcimi odstránenie časti hodnôt, ktoré nie sú súčasťou riešenia. Tieto obecné techniky je možné využiť pre široké spektrum rôznorodých problémov.

V *numerickej analýze* interpretujeme obmedzujúce podmienky ako rovnosti a nerovnosti medzi reálnymi funkciami. Často sa využíva linearizácia nelineárnych podmienok, avšak kvôli obmedzenej presnosti a zaokrúhľovacím chybám môže v praxi dôjsť k nepresnostiam v získanom riešení.

Intervalová analýza rieši tento problém nahradením zaokrúhlených reálnych čísel intervalmi. Intervalové algoritmy dokážu pomocou intervalovej aritmetiky a správneho zaokrúhľovania garantovať obsiahnutie presného riešenia problému vo výslednej aproximácii.

3.1.2 Intervaly a riešenie CSP

Pri tvorbe tejto práce sme sa rozhodli pre využitie intervalovej analýzy, ktorá poskytuje garantované výsledky aj pri obmedzenej presnosti čísel v počítačovej aritmetike. Získané výsledky síce môže byť oproti práci s reálnymi číslami nadhodnotené, avšak napriek tomu nám dokáže poskytnúť mnoho informácií o presnom riešení. Túto skutočnosť čiastočne ilustruje nasledujúci príklad s využitím prirodzeného intervalového rozšírenia funkcie pre zistenie, či je daná funkcia v niektorom bode definičného oboru nulová.

Príklad. Uvažujme reprezentáciu reálnych čísel s presnosťou obmedzenou na jedno desatinné miesto. Buď $f(x) = x(x-1) - \sin(x)$ definovaná na intervale $\boldsymbol{x} = [2,3]$. Potom pre prirodzené intervalové rozšírenie f platí

$$\begin{aligned}[f](\boldsymbol{x}) &= [2,3]([2,3]-1) - \sin([2,3]) \\ &= [2,3] \cdot [1,2] - [0.1,1] \\ &= [2,6] - [0.1,1] = [1,5.9]\end{aligned}$$

Z definície intervalového rozšírenia vieme, že $f(\boldsymbol{x}) \subseteq [f](\boldsymbol{x})$, dokázali sme teda, že funkcia f nemá na intervale $[2,3]$ žiadny nulový bod.

Intervalové výpočty sú však často časovo náročnejšie než výpočty s presnými číslami. Pokúsime sa preto intervalové algoritmy zrýchliť využitím propagačných techník z oblasti programovania s obmedzujúcimi podmienkami. Zameriame sa predovšetkým na obecné techniky, ktoré nekladú žiadne dodatočné požiadavky na typ použitých nelineárnych podmienok alebo tvar riešeného problému.

3.2 Základná myšlienka algoritmu

Algoritmus založený na intervalovom prístupe k riešeniu spojitého CSP bude pracovať s celými definičnými obormi premenných a bude využívať výpočty pomocou intervalovej aritmetiky. Vstupom algoritmu je množina obmedzujúcich podmienok a zoznam definičných oborov jednotlivých premenných, výstupom budú tri množiny intervalových boxov popisujúcich riešenie problému.

Zo zadania problému dostaneme intervalový box $\boldsymbol{x}$ tvorený definičnými obormi premenných. Tento box vložíme do množiny nespracovaných boxov. Každý prvok tejto množiny spracujeme podľa jednej z nasledujúcich možností:

(a) ukážeme, že každý prvok $\boldsymbol{x}$ je riešením CSP,

(b) ukážeme, že $\boldsymbol{x}$ neobsahuje žiadne riešenie CSP,

(c) o $\boldsymbol{x}$ sa nám nepodarí preukázať žiadnu z predchádzajúcich možností.

V prípade (a) môžeme priradenie jednotlivých intervalov z $\boldsymbol{x}$ príslušným premenným prehlásiť za riešenie. V prípade (b) sme ukázali, že žiadne z možných ohodnotení premenných nie je konzistentné so všetkými podmienkami, preto je množina riešení problému v tomto boxe prázdna. V prípade (c) box $\boldsymbol{x}$ rozdelíme na niekoľko menších častí, ktoré vložíme do množiny nespracovaných boxov.

Pre zaistenie konečnosti algoritmu zvolíme minimálnu šírku ε najdlhšej strany boxu, ktorý ešte budeme v bode (c) ďalej deliť. Všetky boxy, ktorých každá strana bude menšia než ε a nepodarí sa nám o nich preukázať možnosť (a) ani (b), zaradíme do množiny nerozhodnutých boxov.

Zjednotenie boxov, u ktorých sme preukázali možnosť (a), bude tvoriť *vnútorný odhad* množiny skutočných riešení CSP a pridaním množiny nerozhodnutých boxov získame *vonkajší odhad*. Tieto odhady nám môžu poskytnúť informácie o riešiteľnosti zadanej úlohy – ak je vonkajší odhad prázdny, CSP nemá riešenie a ak je naopak vnútorný odhad neprázdny, potom určite nejaké riešenie daného CSP existuje.

3.3 Testovanie splnenia podmienok

Algoritmus počas behu testuje, či daný intervalový box spĺňa obmedzujúce podmienky. Pre tento test budeme používať prirodzené intervalové rozšírenie definované v sekcii 2.1.4. Uvažujme funkciu $f : \mathbb{R}^n \to \mathbb{R}$ zadanú predpisom, jej prirodzené intervalové rozšírenie $[f] : \mathbb{IR}^n \to \mathbb{IR}$ a obmedzujúcu podmienku $f(x_1, \ldots, x_n) = 0$. Označme $\mathcal{S}$ množinu boxov, ktoré sú riešením daného CSP, $\mathcal{N}$ množinu boxov neobsahujúcich žiadne riešenie a $\mathcal{B}$ množinu nerozhodnutých boxov. Pre daný n-rozmerný box $\boldsymbol{x}$ a zvolený parameter ε vyhodnotíme test nasledovne:

(a) ak platí $[f](\boldsymbol{x}) = 0$ (ekviv. $[f](\boldsymbol{x}) \subseteq [0, 0]$), pridaj $\boldsymbol{x}$ do množiny $\mathcal{S}$,

(b) ak $0 \notin [f](\boldsymbol{x})$ (ekviv. $[f](\boldsymbol{x}) \cap [0, 0] = \emptyset$), pridaj $\boldsymbol{x}$ do množiny $\mathcal{N}$,

(c) ak sme zatiaľ box $\boldsymbol{x}$ nezaradili a je vo všetkých rozmeroch menší než ε, pridaj $\boldsymbol{x}$ do množiny $\mathcal{B}$,

(d) inak $\boldsymbol{x}$ rozdeľ a otestuj vzniknuté boxy.

Podobne pre podmienku tvaru $f(x_1, \ldots, x_n) \leq 0$ môžeme body (a) a (b) upraviť na test:

(a) $[f](\boldsymbol{x}) \leq 0$ (ekviv. $[f](\boldsymbol{x}) \subseteq (-\infty, 0]$),

(b) $[f](\boldsymbol{x}) > 0$ (ekviv. $[f](\boldsymbol{x}) \cap (-\infty, 0] = \emptyset$).

Uvažujme CSP s premennými $(x_1, \ldots, x_n) = x$ a ich definičnými obormi $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ a obmedzujúcimi podmienkami v tvare

$$f_1(x) = 0, \ldots, f_k(x) = 0,$$
$$f_{k+1}(x) \leq 0, \ldots, f_m(x) \leq 0$$

pre nejaké $k \in \{0, \ldots, m\}$ (podmienky s nerovnosťou $\geq$ môžeme upraviť vynásobením hodnotou -1). Priraďme každej z podmienok $f_1, \ldots, f_k$ degenerovaný interval $[0, 0]$ a podmienkam $f_{k+1}, \ldots, f_m$ neobmedzený interval $(-\infty, 0]$ a označme tieto intervaly $(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m)$. Týmto krokom upravíme každú obmedzujúcu podmienku na ekvivalentný tvar $f_i(x) \in \boldsymbol{y}_i$. S týmto tvarom pracuje algoritmus 3.1, ktorý je známy pod anglickou skratkou SIVIA (Set Inverter via Interval Analysis, Jaulin – Walter (1993a)).

Algoritmus využíva pomocnú funkciu `dividebox`, ktorá rozdelí vstupný box na niekoľko menších. Primárne budeme používať delenie podľa najdlhšej strany, avšak sekcia 3.4.2 popisuje ďalšie možné prístupy. Funkcia ďalej prijíma parameter ε a zaisťuje, aby sa box nikdy nedelil podľa strany s menšou šírkou než je tento parameter. Pre box $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ dostaneme operáciou delenia na polovicu podľa strany s indexom j boxy

$$\boldsymbol{x}_l = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{j-1}, [\underline{x_j}, x_j^c], \boldsymbol{x}_{j+1}, \ldots, \boldsymbol{x}_n),$$
$$\boldsymbol{x}_r = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{j-1}, [x_j^c, \overline{x_j}], \boldsymbol{x}_{j+1}, \ldots, \boldsymbol{x}_n).$$

Boxy vzniknuté rozdelením testovaného boxu sú uložené na zásobník a otestované v ďalších krokoch algoritmu. Podmienkou pre ukončenie algoritmu je spracovanie všetkých boxov na zásobníku.

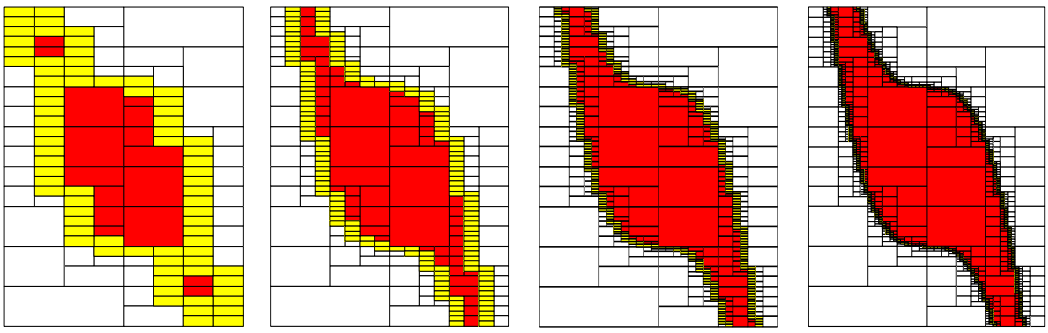
Algoritmus 3.1 Set Inverter via Interval Analysis (`cspsivia`)

Vstup: $\forall i \in \{1,\ldots,m\}: \ f_i : \mathbb{R}^n \to \mathbb{R}$, $\boldsymbol{y} \in \mathbb{IR}^m$, $\boldsymbol{x} \in \mathbb{IR}^n$
Výstup: množiny $\mathcal{S}, \mathcal{N}, \mathcal{B}$ popisujúce riešenie daného problému

1: $\mathcal{S} \leftarrow \emptyset, \mathcal{N} \leftarrow \emptyset, \mathcal{B} \leftarrow \emptyset$
2: Stack.`push`($\boldsymbol{x}$)
3: **while** Stack $\neq \emptyset$ **do**
4: $\quad \boldsymbol{x} \leftarrow$ Stack.`pop`()
5: $\quad$ **if** $\exists i \in \{1,\ldots,m\} : [f_i](\boldsymbol{x}) \cap \boldsymbol{y}_i = \emptyset$ **then**
6: $\quad\quad \mathcal{N} \leftarrow \mathcal{N} \cup \{\boldsymbol{x}\}$
7: $\quad$ **else if** $\forall i \in \{1,\ldots,m\} : [f_i](\boldsymbol{x}) \subseteq \boldsymbol{y}_i$ **then**
8: $\quad\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{\boldsymbol{x}\}$
9: $\quad$ **else if** $\max(\{w(\boldsymbol{x}_i) \mid i \in \{1,\ldots,n\}) < \varepsilon$ **then**
10: $\quad\quad \mathcal{B} \leftarrow \mathcal{B} \cup \{\boldsymbol{x}\}$
11: $\quad$ **else**
12: $\quad\quad \mathbb{X}_{new} \leftarrow$ `dividebox`($\boldsymbol{x}, \varepsilon$)
13: $\quad\quad$ **for each** $\boldsymbol{x}' \in \mathbb{X}_{new}$ **do**
14: $\quad\quad\quad$ Stack.`push`($\boldsymbol{x}'$)

Definícia 3.1 (Dláždenie)**.** Buď $\boldsymbol{x} \in \mathbb{IR}^n$, potom zjednotenie disjunktných boxov (až na hranicu) nenulového objemu, ktoré pokrýva $\boldsymbol{x}$, nazveme jeho *dláždením*. Dláždenie boxu $\boldsymbol{x}$ nazveme *pravidelné*, pokiaľ je možné ho získať z pôvodného boxu konečným aplikovaním operácie voľby boxu a delenia na polovicu podľa najdlhšej strany s najnižším indexom.

Výsledkom algoritmu SIVIA je pravidelné dláždenie vstupného boxu $\boldsymbol{x}$ tvorené zjednotením množín $\mathcal{S}$ (červené boxy na obrázku 3.1), $\mathcal{N}$ (biele boxy) a $\mathcal{B}$ (žlté boxy). Označme $\mathcal{R}$ množinu riešení zadaného CSP, potom platí $\mathcal{S} \subseteq \mathcal{R} \subseteq (\mathcal{S} \cup \mathcal{B})$ a $\mathcal{R} \cap \mathcal{N} = \emptyset$.



(a) $\varepsilon = 1$ (b) $\varepsilon = 0.5$ (c) $\varepsilon = 0.25$ (d) $\varepsilon = 0.125$

Obr. 3.1: Vplyv parametru ε na presnosť dláždenia

3.3.1 Vlastnosti algoritmu

V tejto sekcii dokážeme, že algoritmus 3.1 dobehne v konečnom počte krokov a na výstupe vydá výsledok, v ktorom budú obsiahnuté všetky riešenia CSP. Ďalej uvedieme poznatky o konvergencii zjednotenia boxov v množine $\mathcal{S} \cup \mathcal{B}$ vydanej na výstupe algoritmu SIVIA ku skutočnej množine riešení pre $\varepsilon \to 0$. V závere odvodíme dosiahnutú časovú a priestorovú zložitosť algoritmu.

Lemma 3.1 (Konečnosť)**.** *Algoritmus 3.1 vydá výsledok vždy po konečnom počte krokov.*

Dôkaz. Ak by algoritmus nedobehol v konečnom čase, znamenalo by to, že na zásobník v nekonečne mnoho krokoch pridávame nové boxy (počet boxov, ktoré vzniknú použitím funkcie `dividebox`, je konečný). K tomu dôjde iba v prípade, v nekonečne mnoho krokoch nájdeme nezaradený box, ktorý má aspoň jednu dostatočne dlhú stranu. Pre každé $\varepsilon > 0$ však v konečnom počte krokov dostaneme z pôvodného boxu operáciou delenia podľa najdlhšej strany množinu boxov, ktoré budú mať šírku každého intervalu menšiu než ε. Tento prípad teda nemôže nastať. □

Lemma 3.2 (Korektnosť)**.** *Boxy z množín $\mathcal{S}$ a $\mathcal{B}$ na výstupe algoritmu 3.1 obsahujú všetky riešenia zadaného CSP.*

Dôkaz. Algoritmus by nepracoval korektne, ak by zaradil nejakú hodnotu x_0 z pôvodného boxu $\boldsymbol{x}_0$, ktorá by bola riešením CSP, do množiny $\mathcal{N}$ neobsahujúcej žiadne riešenia. Z definície 2.6 platí pre reálnu funkciu f a jej prirodzené intervalové rozšírenie $[f]$ vzťah

$$f(x_0) \in f(\boldsymbol{x}_0) \subseteq [f](\boldsymbol{x}_0).$$

Pre zaradenie boxu $\boldsymbol{x}$ obsahujúceho bod x_0 do množiny $\mathcal{N}$ musí existovať podmienka (bez újmy na obecnosti uvažujme tvar $f(x) = 0$), pre ktorú $0 \notin [f](\boldsymbol{x})$. Z uvedenej inklúzie ale plynie $f(x_0) \neq 0$, čo je v spore s tvrdením, že vektor x_0 je riešením daného CSP a že ohodnotenie, ktoré určuje, je konzistentné so všetkými obmedzujúcimi podmienkami. □

Pre overenie vlastnosti konvergencie potrebujeme zaviesť koncept vzdialenosti medzi dvomi množinami. Pre definíciu tejto vzdialenosti použijeme pojem metriky a metrického priestoru.

Definícia 3.2 (Metrický priestor)**.** Dvojicu (X, ρ), kde X je množina prvkov a $\rho : X \times X \to \mathbb{R}$, nazveme *metrickým priestorom*, ak pre každú trojicu prvkov $x, y, z \in X$ platí:

(a) $\rho(x, y) \geq 0$, (nezápornosť)

(b) $\rho(x, y) = \rho(y, x)$, (symetria)

(c) $\rho(x, y) = 0 \Leftrightarrow x = y$, (totožnosť)

(d) $\rho(x, y) \leq \rho(x, z) + \rho(z, y)$. (trojuholníková nerovnosť)

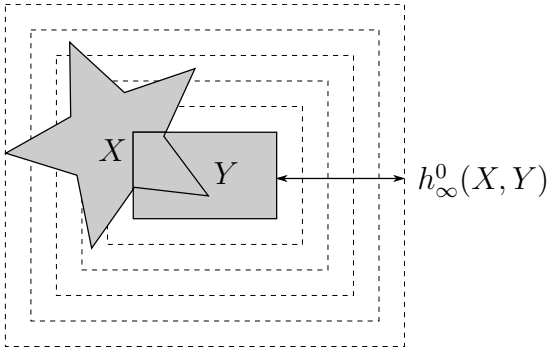
Funkciu ρ nazývame *metrikou* na priestore (X, ρ).

Uvažujme metrický priestor $\mathbb{R}^n$ s maximovou metrikou:

$$\rho_{max}(x, y) \coloneqq \max_{i \in \{1,\ldots,n\}} |x_i - y_i|.$$

Nech $B(o, 1)$ označuje jednotkovú guľu v priestore $(\mathbb{R}^n, \rho_{max})$, zavedieme *blízkosť* dvoch kompaktných množín $X, Y \subseteq \mathbb{R}^n$ ako

$$h^0_\infty(X, Y) \coloneqq \inf\{r \in \mathbb{R}^+ \mid X \subseteq Y + rB(o, 1)\}.$$



Obr. 3.2: Blízkosť dvoch množín v priestore $(\mathbb{R}^2, \rho_{max})$

Jednou z najznámejších metrík na priestore kompaktných podmnožín $\mathbb{R}^n$ je tzv. *Hausdorffova metrika*, ktorá je definovaná ako

$$h_\infty(X,Y) \coloneqq \max\{h^0_\infty(X,Y), h^0_\infty(Y,X)\}.$$

Ďalší možný prístup k definícii vzdialenosti množín poskytuje tzv. *doplnková Hausdorffova semi-metrika*. Doplnkovú Hausdorffovu semi-metriku môžeme zaviesť ako

$$\overline{h}_\infty(X,Y) \coloneqq h_\infty(\mathbb{R}^n/X, \mathbb{R}^n/Y),$$

kde $\mathbb{R}^n/X$ označuje množinový doplnok X v $\mathbb{R}^n$.

Príklad. (Jaulin et al., 2001) Uvažujme dostatočne malé $\varepsilon > 0$ a tri množiny $X = [1,7]$, $Y = [1,7] \cup [9-\varepsilon, 9]$ a $Z = [1,5] \cup [5+\varepsilon, 7]$. Potom vzdialenosť množín X a Y v zavedených metrikách je $h_\infty(X,Y) = 2$, $\overline{h}_\infty(X,Y) = \frac{\varepsilon}{2}$, vzdialenosť množín X a Z je $h_\infty(X,Z) = \frac{\varepsilon}{2}$, $\overline{h}_\infty(X,Z) = 2$.

Na základe uvedených definícií zavedieme metriku m_∞, ktorú použijeme pre popis konvergencie algoritmu Sivia:

$$m_\infty(X,Y) \coloneqq \max\{h_\infty(X,Y), \overline{h}_\infty(X,Y)\}.$$

Ďalej zavedieme na metrickom priestore niektoré pojmy, ktoré nám umožnia formalizovať popis konvergencie algoritmu Sivia vo vete 3.3.

Definícia 3.3. Nech (X,ρ) a (Y,σ) sú metrické priestory a $f : X \to Y$. Funkcia f je *spojitá* práve vtedy, keď spĺňa podmienku:

$$(\forall x \in X)(\forall \varepsilon > 0)(\exists \delta > 0)(\forall x' \in X) : \rho(x,x') < \delta \Rightarrow \sigma(f(x), f(x')) < \varepsilon.$$

Definícia 3.4. Buď (X,ρ) metrický priestor, $\{x_n\}$ postupnosť prvkov X a $x \in X$. Potom postupnosť $\{x_n\}$ *konverguje k* x, ak platí:

$$(\forall \varepsilon > 0)(\exists N \in \mathbb{N}) : n > N \Rightarrow \rho(x_n, x) < \varepsilon.$$

Definícia 3.5. Buď $X \subseteq \mathbb{R}^n$, potom X je *plná*, ak je rovná uzáveru svojho vnútrajšku, teda platí $X = \mathrm{clo}(\mathrm{int}(X))$.

Buď $f : \mathbb{R}^n \to \mathbb{R}^m$, potom pre $Y \subseteq \mathbb{R}^m$ označuje f^{-1} zobrazenie zadané vzťahom $f^{-1}(Y) \coloneqq \{x \in \mathbb{R}^n \mid f(x) \in Y\}$.

Veta 3.3 (Konvergencia)**.** *Buď $f : \mathbb{R}^n \to \mathbb{R}^m$, $\boldsymbol{y} \in \mathbb{IR}^n$ a $\mathcal{S}$, $\mathcal{B}$ zjednotenia intervalových boxov v množinách, ktoré sú výstupom algoritmu* Sivia *pre problém s obmedzujúcimi podmienkami $f(x) \in \boldsymbol{y}$ a označme $\mathcal{R}$ množinu riešení tohto problému. Ak f^{-1} je m_∞-spojitá na $\boldsymbol{y}$, potom pre $\varepsilon \to 0$ platí:*

(a) $\mathcal{B} \xrightarrow{\supseteq} \partial\mathcal{R}$,

(b) $\mathcal{S} \xrightarrow{\subseteq} \mathcal{R}$ *(ak je* $\mathcal{R}$ *plná),*

(c) $\mathcal{S} \cup \mathcal{B} \xrightarrow{\supseteq} \mathcal{R}$,

kde $\partial\mathcal{R}$ označuje hranicu množiny $\mathcal{R}$ a $\xrightarrow{\supseteq}$, resp. $\xrightarrow{\subseteq}$ označuje h_∞-konvergenciu zvonku, resp. zvnútra množiny.

Dôkaz. viď Jaulin – Walter (1993a) □

V prvej časti vety 3.4 predstavíme a dokážeme opravený odhad počtu iterácií hlavného cyklu algoritmu Sivia pre vyriešenie CSP s presnosťou ε. Druhá časť vety udáva odhad potrebnej veľkosti zásobníku pre uloženie spracovávaných boxov.

Veta 3.4. *Uvažujme spojitý problém splňovania obmedzujúcich podmienok s premennými $x_1, \ldots, x_n$ a definičnými obormi $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$.*

(a) Počet iterácií hlavného cyklu algoritmu Sivia *potrebných pre vyriešenie tohto problému so zvolenou presnosťou ε je nanajvýš*

$$2\left(\frac{2 \cdot \max_i\{w(\boldsymbol{x}_i)\}}{\varepsilon} - 1\right)^n + 1.$$

(b) Maximálny počet boxov uložených na zásobníku v priebehu algoritmu je zhora obmedzený hodnotou výrazu

$$n(\log_2(w(\boldsymbol{x})) - \log_2(\varepsilon) + 1).$$

Dôkaz. Časť (a): Počet iterácií algoritmu je rovný počtu spracovaných boxov, teda $2 \cdot d + 1$, kde d označuje počet delení vykonaných v priebehu algoritmu. Nech $\boldsymbol{x}_i$ je strana pôvodného boxu s najväčšou šírkou, označme l šírku najmenšej strany boxu patriaceho do výsledného dláždenia, ktorý vznikol delením $\boldsymbol{x}_i$. V najhoršom prípade bola $\boldsymbol{x}_i$ rozdelená iba na intervaly dĺžky $l < \varepsilon$. Buď k počet týchto intervalov, potom platí $k \cdot l = w(\boldsymbol{x}_i)$. Najmenšia strana, ktorú ešte v priebehu algoritmu Sivia môžeme rozdeliť, má šírku ε, platí preto nerovnosť $\frac{\varepsilon}{2} \leq l$. Spojením odvodených rovností a nerovníc dostaneme vzťah:

$$\begin{aligned} k &\leq \frac{w(\boldsymbol{x}_i)}{\frac{\varepsilon}{2}}, \\ k &\leq \frac{2 \cdot w(\boldsymbol{x}_i)}{\varepsilon}. \end{aligned}$$

Algoritmus pripúšťa možnosť takéhoto delenia pre každú premennú, pre rozdelenie jedného intervalu na k častí potrebujeme $k-1$ delení. Horný odhad na celkový počet delení je teda:

$$d \leq \left(\frac{2 \cdot \max_i\{w(\boldsymbol{x}_i)\}}{\varepsilon} - 1\right)^n.$$

Pre dôkaz časti (b) viď Jaulin – Walter (1993b). □

3.4 Modifikácie algoritmu

3.4.1 Testy inklúzie

Pre fungovanie algoritmu 3.1 sme predpokladali, že vieme vytvoriť prirodzené intervalové rozšírenie každej funkcie použitej v obmedzujúcich podmienkach. Ak máme funkciu zadanú predpisom, ktorý používa základné aritmetické operácie $(+, -, \cdot, /)$, prípadne niektoré ďalšie operácie a funkcie (mocniny, logaritmy, goniometrické funkcie, ...), ktoré dokážeme jednoducho uviesť aj do intervalovej aritmetiky, môžeme príslušné funkcie priamočiaro nahradiť. Avšak s menšími úpravami je možné algoritmus Sivia použiť aj pre zložitejšie alebo implicitne zadané funkcie.

Definícia 3.6 (Test inklúzie)**.** Funkcia $[t] : \mathbb{IR}^n \to \{-1, 0, 1\}$ je *test inklúzie* pre množinu $\mathbb{S}$, ak platí:

$$[t](\boldsymbol{x}) = 1 \Rightarrow \boldsymbol{x} \subseteq \mathbb{S},$$
$$[t](\boldsymbol{x}) = 0 \Rightarrow \boldsymbol{x} \cap \mathbb{S} = \emptyset.$$

Ak je $[t](\boldsymbol{x}) = -1$, nedokážeme o vzťahu $\boldsymbol{x}$ a množiny $\mathbb{S}$ rozhodnúť.

V prípade, že pre zadanú funkciu dokážeme vytvoriť test inklúzie (napríklad na základe nejakého typu intervalového rozšírenia), môžeme tento test v algoritme Sivia využiť k rozdeleniu boxov do jednotlivých výsledných množín. Algoritmus 3.1 zobecníme úpravou riadkov 5 a 7 na algoritmus 3.2.

Algoritmus 3.2 Sivia s využitím testu inklúzie (`cspsiviatest`)

Vstup: $\forall i \in \{1, \ldots, m\} : [t_i] : \mathbb{IR}^n \to \{-1, 0, 1\}$, $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$
Výstup: množiny $\mathcal{S}, \mathcal{N}, \mathcal{B}$ popisujúce riešenie daného problému

```
1: S ← ∅, N ← ∅, B ← ∅
2: Stack.push(x)
3: while Stack ≠ ∅ do
4:    x ← Stack.pop()
5:    if ∃i ∈ {1,...,m} : [t_i](x) = 0 then
6:       N ← N ∪ {x}
7:    else if ∀i ∈ {1,...,m} : [t_i](x) = 1 then
8:       S ← S ∪ {x}
9:    else if max({w(x_i) | i ∈ {1,...,n}}) < ε then
10:      B ← B ∪ {x}
11:   else
12:      X_new ← dividebox(x, ε)
13:      for each x' ∈ X_new do
14:         Stack.push(x')
```

Testy inklúzie nám umožňujú rozšíriť množinu funkcií, s ktorými dokáže algoritmus Sivia pracovať. Pri ich použití však strácame informácie o predpise funkcie využívané technikami, ktoré zvyšujú efektivitu tohoto algoritmu (viď kapitolu 4).

3.4.2 Delenie intervalových boxov

Jedným z faktorov, ktoré môžu ovplyvniť počet iterácií algoritmu a štruktúru výsledného dláždenia, je spôsob implementácie funkcie `dividebox`. Základným prístupom je deliť box vždy na polovice podľa zvolenej súradnice. Ďalšou možnosťou je delenie boxu na viac častí, a to buď podľa jednej alebo podľa viacerých súradníc.

Významná je tiež stratégia výberu strany, podľa ktorej bude box rozdelený. Medzi najpoužívanejšie prístupy patria:

- delenie podľa definičného oboru premennej, ktorý má najväčšiu šírku,
- delenie podľa definičného oboru premennej zvolenej podľa pevne daného poradia s cyklickým opakovaním,
- využívanie rôznych pomocných kritérií, ktoré určia najvhodnejšiu premennú (viď napr. Kearfott – Novoa (1990)).

3.5 Projekcia riešenia

Formulácia úlohy s obmedzujúcimi podmienkami niekedy vyžaduje zavedenie dodatočných pomocných premenných, ktoré sú potrebné pre výpočet, avšak v samotnom riešení ich hodnoty nie sú významné. V takomto prípade môže byť žiadúce vytvoriť projekciu riešenia na podmnožinu premenných obsiahnutých v CSP. V tejto práci sa budeme venovať projekciám do priestoru dimenzie 1 a 2, ktoré sú vhodné pre vizualizáciu výsledných dát.

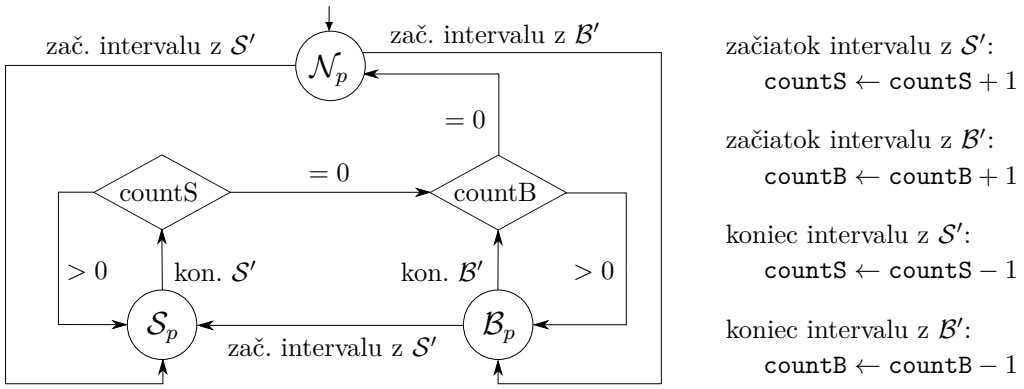
Pripomeňme, že množina $\mathcal{S}$ obsahuje intervalové boxy, ktoré sú podmnožinou množiny riešení CSP, množina $\mathcal{N}$ obsahuje boxy nepatriace do riešenia a $\mathcal{B}$ je množina boxov, ktoré nebolo možné pri danej presnosti zaradiť. Vytvoríme množiny $\mathcal{S}'$, $\mathcal{N}'$ a $\mathcal{B}'$ tak, že z každého boxu ponecháme iba súradnice zodpovedajúce premenným, ktoré budú súčasťou výsledku projekcie. Po tejto úprave môžu vzniknúť skupiny boxov, ktoré sa prekrývajú. Označme $\mathcal{S}_p$, $\mathcal{N}_p$, $\mathcal{B}_p$ množiny boxov, ktoré budú výsledkom korektnej projekcie pôvodného dláždenia do dimenzie 1 (zjednotenie troch množín disjunktných boxov pokrývajúce pôvodný box). Zavedieme nasledujúce pravidlá pre prieniky prekrývajúcich sa boxov:

- prienik boxu $\boldsymbol{x} \in \mathcal{S}'$ s akýmkoľvek iným boxom $\boldsymbol{y}$ patrí do množiny $\mathcal{S}_p$,
- prienik boxu $\boldsymbol{x} \in \mathcal{B}'$ s boxom $\boldsymbol{y} \in \mathcal{B}' \cup \mathcal{N}'$ patrí do množiny $\mathcal{B}_p$,
- prienik boxov $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{N}'$ patrí do množiny $\mathcal{N}_p$.

Na základe týchto podmienok môžeme vytvoriť jednoduchý algoritmus pre projekciu riešenia CSP na jednu premennú. Jednotlivé horné a dolné hranice intervalov v množinách $\mathcal{S}'$, $\mathcal{N}'$ a $\mathcal{B}'$ zoradíme vzostupne a ku každej hranici uložíme informáciu o tom, do ktorej množiny patrí príslušný interval. Takto zoradený zoznam hraníc bude tvoriť zoznam udalostí, ktoré musíme spracovať. Pri každej udalosti aktualizujeme informáciu o tom, či je aktívny nejaký interval z množiny $\mathcal{S}'$, $\mathcal{N}'$ alebo $\mathcal{B}'$ a podľa uvedených pravidiel priradíme vzniknuté úseky intervalov do jednej z výsledných množín. Aby sa do výsledku nedostali prebytočné

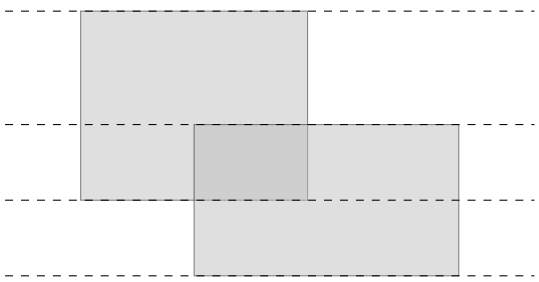
degenerované intervaly, budeme radiť zoznam tak, aby sa najprv spracovali dolné hranice (počítadlo aktívnych intervalov nedosiahne nulovú hodnotu, pokiaľ to nebude nutné).

Diagram na obrázku 3.3 naznačuje priebeh algoritmu pre projekciu do dimenzie 1 pomocou troch stavov $\mathcal{S}_p, \mathcal{N}_p, \mathcal{B}_p$ a prechodov medzi nimi na základe rozhodovacích podmienok. Ďalej doplníme nasldujúce detaily: Počiatočným stavom je $\mathcal{N}_p$. Ku stavom $\mathcal{S}_p$ a $\mathcal{B}_p$ sú priradené premenné `countS` a `countB`, v ktorých je uložený počet aktívnych intervalov z množín $\mathcal{S}'$ a $\mathcal{B}'$. Postupne spracúvame udalosti tvorené množinou zoradených začiatkov a koncov intervalov – pri udalosti začiatku intervalu zvýšime príslušné počítadlo, pri udalosti konca intervalu počítadlo opäť znížime. Pri vstupe do stavu začína nový interval v danej množine, pri prechode do iného stavu interval ukončíme a pridáme ho do výslednej množiny. Zo stavu $\mathcal{S}_p$ pokračujeme do iného stavu pri splnení podmienky $\texttt{countS} = 0$, zo stavu $\mathcal{B}_p$ prechádzame do stavu $\mathcal{N}_p$, ak je $\texttt{countB} = 0$ a do stavu $\mathcal{S}_p$, ak spracujeme začiatok nového intervalu z množiny $\mathcal{S}'$.



Obr. 3.3: Projekcia riešenia CSP na jednu premennú (`proj1d`)

Výhodou tohoto algoritmu je minimálny možný počet intervalov vo výstupných množinách. Algoritmus `proj1d` využijeme pri tvorbe algoritmu pre projekciu na dve premenné. Základom bude geometrický algoritmus, ktorý rozdelí pôvodný skúmaný box na pásy podľa dolných a horníc hraníc v jednej zvolenej súradnici tak, že v každom vzniknutom páse bude možné uvažovať o dvojrozmerných boxoch ako o intervaloch (viď obr. 3.4).



Obr. 3.4: Delenie priestoru na pásy

V každom takto získanom páse rozdelíme jednotlivé intervaly do množín $\mathcal{S}'$, $\mathcal{N}'$, $\mathcal{B}'$. Pre zjednodušenie vytvoríme zoznam intervalov v jednotlivých množinách pre aktuálny pás, pri spracovaní ďalšieho pásu stačí tento zoznam aktualizovať

na základe začiatkov a koncov intervalov na danej úrovni. Pre získanie projekcie boxov v každom samostatnom páse použijeme algoritmus pre projekciu do dimenzie 1. Intervaly, ktoré dostaneme ako výstp funkcie `proj1d`, rozšírime na celý pás pridaním druhej súradnice, čím získame dvojrozmerné intervalové boxy. Po spracovaní všetkých pásov tvorí zjednotenie získaných výsledkov požadovanú projekciu dláždenia do dimenzie 2. Celý priebeh dvojrozmernej projekcie dláždenia popisuje algoritmus 3.3. Zoznamy stripeS a stripeB obsahujú intervaly v aktuálnom páse.

Algoritmus 3.3 Projekcia riešenia CSP na dve premenné (`proj2d`)

Vstup: množiny $\mathcal{S}, \mathcal{N}, \mathcal{B}$ popisujúce riešenie CSP, indexy $i, j \in \{1, \ldots, n\}$
Výstup: množiny $\mathcal{S}_p, \mathcal{N}_p, \mathcal{B}_p$ popisujúce projekciu riešenia na premenné x_i, x_j

1: $\mathcal{S}' \leftarrow \{(\boldsymbol{x}_i, \boldsymbol{x}_j) \mid (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \mathcal{S}\}$; $\mathcal{B}' \leftarrow \{(\boldsymbol{x}_i, \boldsymbol{x}_j) \mid (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \mathcal{B}\}$
2: $\mathbb{E} \leftarrow$ `sort`$([y \mid y = \underline{x}_1 \vee y = \overline{x}_1, (x_1, x_2) \in \mathcal{S}' \cup \mathcal{B}'])$
3: stripeS $\leftarrow \emptyset$; stripeB $\leftarrow \emptyset$
4: $j \leftarrow 1$
5: $l =$ `length`$(\mathbb{E})$
6: **while** $j < l$ **do**
7: $\quad v \leftarrow \mathbb{E}(j)$
8: $\quad$ **while** $\mathbb{E}(j) = v$ **do**
9: $\quad\quad$ **if** $\mathbb{E}(j)$ je hranica boxu $\boldsymbol{x}$ z $\mathcal{S}'$ **then**
10: $\quad\quad\quad$ **if** $\mathbb{E}(j)$ je dolná hranica **then**
11: $\quad\quad\quad\quad$ stripeS $\leftarrow$ stripeS $\cup\, \boldsymbol{x}$
12: $\quad\quad\quad$ **else**
13: $\quad\quad\quad\quad$ stripeS $\leftarrow$ stripeS $\setminus\, \boldsymbol{x}$
14: $\quad\quad$ **else if** $\mathbb{E}(j)$ je hranica boxu $\boldsymbol{x}$ z $\mathcal{B}'$ **then**
15: $\quad\quad\quad$ **if** $\mathbb{E}(j)$ je dolná hranica **then**
16: $\quad\quad\quad\quad$ stripeB $\leftarrow$ stripeB $\cup\, \boldsymbol{x}$
17: $\quad\quad\quad$ **else**
18: $\quad\quad\quad\quad$ stripeB $\leftarrow$ stripeB $\setminus\, \boldsymbol{x}$
19: $\quad\quad j \leftarrow j + 1$
20: $\quad \mathcal{S}, \mathcal{N}, \mathcal{B} \leftarrow$ `proj1d`(stripeS, `[]`, stripeB)
21: $\quad$ rozšír intervaly v $\mathcal{S}, \mathcal{N}, \mathcal{B}$ na celý pás a vlož do výsledných množín

3.6 Vylepšenia algoritmu

Nasledujúce sekcie prezentujú dva spôsoby vylepšenia algoritmu Sivia, ktoré boli v rámci tejto práce navrhnuté. Prvým z nich je jednoduchá úprava testovania splnenia obmedzujúcich podmienok, pomocou ktorej zabránime nepotrebnému opakovanému vyhodnocovaniu niektorých podmienok. V druhej sekcii sa budeme venovať zníženiu počtu intervalových boxov popisujúcich riešenie CSP pri zachovaní pravidelnosti výsledného dláždenia. Navrhnutý spôsob je možné implementovať ako súčasť algoritmu Sivia, a nevyžaduje tak žiadne dodatočné spracovanie výsledku. Efektivita navrhnutého algoritmu je tiež otestovaná na niekoľkých konkrétnych príkladoch.

3.6.1 Testovanie riešenia

Uvažujme spojité CSP s obmedzujúcimi podmienkami v tvare $f_i(x) \in \boldsymbol{y}_i$ pre $i \in \{1, \ldots, m\}$, kde x je vektor n premenných s definičnými obormi danými vektorom $\boldsymbol{x} \in \mathbb{IR}^n$. V riadku 7 základnej verzie algoritmu Sivia testujeme splnenie podmienky

$$\forall i \in \{1, \ldots, m\} : [f_i](\boldsymbol{x}) \subseteq \boldsymbol{y}_i.$$

V každej iterácii hlavného cyklu tak potrebujeme vyhodnotiť m intervalových rozšírení $[f_i]$ reálnych funkcií f_i nad príslušným intervalovým boxom $\boldsymbol{x}$.

V niektorých prípadoch je však možné použiť výsledok testu pre zjednodušenie jeho vyhodnocovania v ďalších iteráciách. Napríklad vieme, že ak skúmaný box spĺňa obmedzujúcu podmienku, potom ju určite spĺňajú aj všetky boxy, ktoré sú jeho podmnožinou. Ak teda zistíme, že daný box spĺňa časť obmedzujúcich podmienok a o ostatných podmienkach nedokážeme rozhodnúť, môžeme si túto informáciu uložiť. Pri rozhodovaní o boxoch vzniknutých jeho rozdelením stačí overovať iba tie podmienky, o ktorých sme v predchádzajúcich krokoch nedokázali rozhodnúť.

Algoritmus 3.1 upravíme tak, aby bola v každom kroku (okrem počiatočného) dostupná informácia o splnení podmienok pre intervalový box, z ktorého vznikol aktuálne spracovávaný box. Vytvoríme nový pomocný zásobník, na ktorý budeme pri delení boxu ukladať pole logických hodnôt, ktorého prvky určujú, či je príslušná podmienka pre daný box splnená. Riadok 7 upravíme tak, že budeme testovať iba podmienky s hodnotou `false` v uloženom poli.

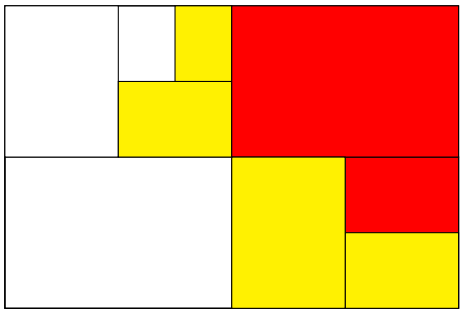
3.6.2 Spájanie intervalových boxov

Výstupom algoritmu Sivia môže byť veľké množstvo intervalových boxov. V tejto sekcii navrhneme spôsob, ako vytvoriť ekvivalentný popis množiny riešení zadaného CSP pomocou menšieho počtu boxov.

Uvažujme intervalový box $\boldsymbol{x} \in \mathbb{IR}^n$, operáciou delenia na polovice podľa najdlhšej strany s najnižším indexom dostaneme dva boxy:

$$\boldsymbol{x}_l = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{j-1}, [\underline{x_j}, x_j^c], \boldsymbol{x}_{j+1}, \ldots, \boldsymbol{x}_n),$$
$$\boldsymbol{x}_r = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{j-1}, [x_j^c, \overline{x_j}], \boldsymbol{x}_{j+1}, \ldots, \boldsymbol{x}_n).$$

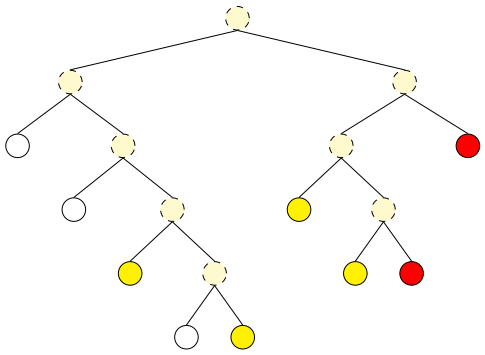
Box $\boldsymbol{x}_l$ nazveme *ľavým synom* pôvodného boxu $\boldsymbol{x}$ a box $\boldsymbol{x}_r$ jeho *pravým synom*. Opakovaným výberom jedného alebo oboch synov a ich ďalším delením dostaneme pravidelné dláždenie boxu $\boldsymbol{x}$.



Obr. 3.5: Pravidelné dláždenie intervalového boxu

Proces vytvárania dláždenia pomocou operácie delenia na polovice môžeme reprezentovať pomocou binárneho stromu. Koreň stromu bude tvoriť box $\boldsymbol{x}$ a jeho potomkami budú uzly s jeho ľavým synom $\boldsymbol{x}_l$ a pravým synom $\boldsymbol{x}_r$. Každý vnútorný uzol zodpovedá boxu, ktorý bol rozdelený. Listy budú reprezentovať boxy, ktoré sú súčasťou výsledného dláždenia. Obrázok 3.6 je ukážkou stromovej reprezentácie dláždenia vytvoreného na obrázku 3.5.

V listoch môžu byť tiež uložené ďalšie informácie. V prípade algoritmu Sivia, kde je dláždenie tvorené zjednotením troch množín, môže byť užitočné ukladať do listov informáciu o príslušnosti daného boxu do jednej z týchto množín.



Obr. 3.6: Reprezentácia dláždenia binárnym stromom

Ľavý syn $\boldsymbol{x}_l$ a pravý syn $\boldsymbol{x}_r$ každého uzlu zodpovedajú intervalovým boxom, ktoré sú v dláždení susedné. Môžeme preto zaviesť operáciu *zjednotenia* týchto dvoch uzlov, ktorej výsledkom bude nahradenie dvojice $\boldsymbol{x}_l$ a $\boldsymbol{x}_r$ boxom, ktorý zodpovedá ich rodičovskému uzlu. V prípade, že sú v strome okrem samotných boxov patriacich do dláždenia uložené aj ďalšie vlastnosti, potrebujeme zaručiť, aby pri zjednotení nedošlo k strate informácií.

Pri behu algoritmu 3.1 môže dôjsť k prípadu, že rozdelíme box na niekoľko menších častí, no pri zadanej presnosti nedokážeme rozhodnúť o príslušnosti žiadnej jeho časti k množine riešení daného CSP. V takomto prípade môže byť žiadúce túto skupinu boxov vo výslednom dláždení nahradiť pôvodným boxom, a tak znížiť počet boxov potrebných pre popis množiny riešení. Tento problém vyriešime opakovaným použitím operácie zjednotenia susedných uzlov v strome reprezentujúcom priebeh algoritmu. Samotný algoritmus Sivia upravíme tak, aby miesto zaradenia boxu do množiny predal informáciu o zaradení do rodičovského uzlu. Po obdržaní tejto informácie od všetkých synov môžeme rozhodnúť, či dokážeme získať informáciu o zaradení rodičovského uzlu do množiny (ak patria všetky uzly synov do rovnakej množiny) a prípadne pokračovať v spájaní na vyšších vrstvách alebo zaradíme synov do jednotlivých množín rovnako ako v pôvodnom algoritme.

Počty boxov vygenerovaných základnou verziou algoritmu Sivia a upravenou verziou založenou na popísanom algoritme spájania boxov sú pre zvolené problémy k dispozícii v tabuľke 3.1. V posledných dvoch stĺpcoch tabuľky sú uvedené počty boxov v jednotlivých množinách v tvare (patriace do riešenia/nepatriace do riešenia/nerozhodnuté).

Pomocou navrhnutého algoritmu najčastejšie spojíme boxy, o ktorých algoritmus Sivia nedokázal pri požadovanej presnosti rozhodnúť ani po ich rozdelení. Kvôli nadhodnocovaniu obrazu funkcie, ku ktorému dochádza pri využití intervalovej aritmetiky, však môžeme týmto spôsobom v niektorých prípadoch spájať aj

Podmienky	Definičný obor	ε	počet boxov (základný alg.)	počet boxov (so spájaním)
$x^2 + y^2 \in [9, 16]$	$[-5, 5] \times [-5, 5]$	0.1	476/524/720	476/524/540
$\sin(x) + y \geq x$	$[-10, 10] \times [0, 10]$	0.01	1915/1704/2633	1627/1423/1660
$x \cdot y + z^3 \geq 2$	$[-5, 5] \times [0, 20] \times [0, 1]$	0.25	395/296/1423	395/296/542
$(x^2 - 1)^2 - y^2(3 + 2y) \leq 0$	$[-10, 10] \times [-10, 10]$	0.1	400/428/664	368/396/418
$x \cdot x + y(y + z) \geq 0$	$[-5, 5] \times [-5, 5] \times [-5, 5]$	0.25	2524/2072/6716	2332/1664/3212

Tabuľka 3.1: Počty boxov v riešení pri použití algoritmu pre ich spájanie

boxy, ktoré sú podmnožinou riešenia alebo boxy nepatriace do riešenia. Ak napríklad uvážime obmedzujúcu podmienku $x \cdot x \geq 0$ na definičnom obore $[-1, 1]$, potom v prvom kroku nedokážeme o zaradení počiatočného intervalu rozhodnúť. Po rozdelení na $[-1, 0]$ a $[0, 1]$ už však oba intervaly danú podmienku spĺňajú.

4 Kontraktory

Táto kapitola predstaví možnosti rozšírenia základného algoritmu SIVIA pomocou tzv. kontraktorov. V úvode stručne popíšeme rôzne spôsoby tvorby kontraktorov a ich výhody a nevýhody. Ďalšie sekcie budú venované konkrétnym druhom kontraktorov založených na propagačných technikách (dopredný a spätný kontraktor a BoxNarrow) a možnostiam ich kombinovania pre dosiahnutie lepších výsledkov. Zároveň uvedieme pojem lokálnej konzistencie, ktorý nám umožní porovnať teoretické vlastnosti uvedených algoritmov.

4.1 Propagačné a iné techniky

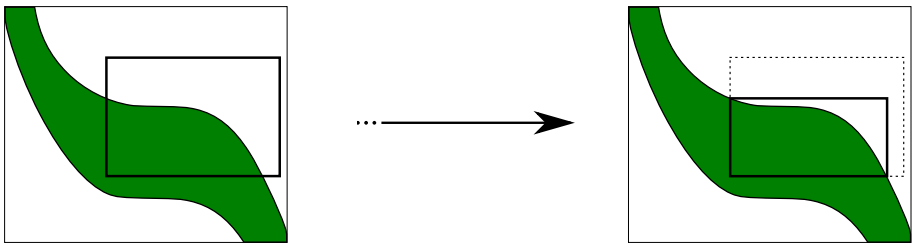
Použitie kontraktorov je jedným z hlavných spôsobov zlepšenia efektivity základného algoritmu SIVIA. Pred testovaním a delením boxu sa pokúsime orezať hodnoty, ktoré nie sú riešením daného CSP a ďalej budeme pracovať s takto získaným boxom. Výhodou je možnosť znížiť počet potrebných iterácií algoritmu pre dosiahnutie požadovanej presnosti, avšak za cenu zložitejšieho vyhodnocovania a straty pravidelnosti získaného dláždenia.

Definícia 4.1. Buď $\mathbb{S} \subseteq \mathbb{R}^n$, potom funkciu $C : \mathbb{IR}^n \to \mathbb{IR}^n$ nazveme *kontraktor* pre množinu $\mathbb{S}$, ak pre každé $\boldsymbol{x} \in \mathbb{IR}^n$ platí:

(a) $C(\boldsymbol{x}) \subseteq \boldsymbol{x}$,

(b) $C(\boldsymbol{x}) \cap \mathbb{S} = \boldsymbol{x} \cap \mathbb{S}$.

Podmienku (a) z definície 4.1 nazývame vlastnosť *kontrakcie* (výsledný box musí byť nanajvýš tak veľký, ako pôvodný) a podmienku (b) vlastnosť *korektnosti* (nestratíme žiadne riešenie).



Obr. 4.1: Použitie kontraktoru

V tejto práci budú primárne popísané propagačné techniky, teda techniky využívajúce polynomiálne algoritmy, ktoré podľa jednotlivých obmedzujúcich podmienok redukujú definičné obory použitých premenných. Propagácia obmedzujúcich podmienok zahŕňa metódy založené na zakázaní určitých hodnôt alebo kombinácií hodnôt premenných, ktoré porušujú nejakú podmnožinu obmedzujúcich podmienok.

Propagačné techniky môžeme posudzovať na základe tzv. iteračných pravidiel alebo na základe lokálnych konzistencií. *Iteračné pravidlá* určujú vlastnosti

typu a poradia operácií použitých pre kontrakciu, teda popisujú priebeh propagácie. *Lokálne konzistencie* naopak popisujú vlastnosti, ktoré musí problém spĺňať po ukončení propagácie. Jedným zo základných druhov lokálnych konzistencií je hranová konzistencia. Z hranovej konzistencie môžeme odvodiť rôzne druhy intervalových konzistencií, ktoré poskytujú aproximáciu množiny v definícii 4.2. Zároveň požadujeme, aby takto definované množiny boli nadmnožinou hranovo konzistentnej množiny, čím zaručíme, že výsledok bude obsahovať všetky ohodnotenia konzistentné s danou podmienkou.

Definícia 4.2 (Hranová konzistencia)**.** Nech $x_1, \ldots, x_n$ sú premenné, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ príslušné definičné obory a c obmedzujúca podmienka v tvare $f(x_1, \ldots, x_n) = 0$. Definičný obor $\boldsymbol{x}_i$ premennej x_i je s podmienkou c *hranovo konzistentný* (angl. *arc consistent*), ak platí, že $\boldsymbol{x}_i$ je množina

$$\{x_i \in \boldsymbol{x}_i \mid \exists x_1 \in \boldsymbol{x}_1, \ldots, \exists x_{i-1} \in \boldsymbol{x}_{i-1}, \exists x_{i+1} \in \boldsymbol{x}_{i+1}, \ldots, \exists x_n \in \boldsymbol{x}_n : f(x_1, \ldots, x_n) = 0\}.$$

V nasledujúcich sekciách budú predstavené dva základné typy kontraktorov – – dopredný a spätný konktrator (a s ním súvisiaca konzistencia vzhľadom k obalu) a kontraktor BoxNarrow (a konzistencia vzhľadom k boxu), ako aj rôzne spôsoby ich implementácie. Ďalej uvedieme možnosti ich kombinácie pre dosiahnutie lepšej efektivity redukcie definičných oborov. Výber podmienok a volanie kontraktorov popisuje algoritmus 4.1, kde $\mathrm{var}(c_i)$ označuje množinu premenných s výskytom v obmedzujúcej podmienke c_i a funkcia `kontraktor` predstavuje kontraktor pre množinu riešení CSP využívajúci podmienku c k zmenšeniu boxu $\boldsymbol{x}$.

Algoritmus 4.1 Obecný cyklus pre použitie kontraktorov (`contract`)

Vstup: obmedzujúce podmienky $C = \{c_1, \ldots, c_m\}$, box $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$
Výstup: kontrahovaný box $\boldsymbol{x}$

```
1: C' ← C
2: while C' ≠ ∅ and x ≠ ∅ do
3:    c ← obmedzujúca podmienka z množiny C'
4:    y ← kontraktor(c, x)
5:    if x ≠ y then
6:       C' ← C' ∪ {c_i ∈ C | ∃j : x_j ∈ var(c_i) ∧ x_j ≠ y_j}
7:       x ← y
8:    else
9:       C' ← C'\{c}
```

Výber obmedzujúcej podmienky v riadku 3 je ľubovoľný, avšak existujú stratégie voľby podmienok, ktoré môžu proces kontrakcie zefektívniť (Goualard – Jermann, 2008). Po použití zvoleného kontraktoru na danú podmienku pridávame v riadku 6 do množiny aktívnych podmienok všetky obmedzujúce podmienky obsahujúce niektorú z premenných, ktorých definičné obory sa v poslednej iterácii cyklu zmenili. V praxi je možné miesto rovnosti testovať dostatočné zkontrahovanie boxu (napr. $\boldsymbol{y}$ je aspoň o 10 % menší než $\boldsymbol{x}$).

Ďalšie typy kontraktorov, ktoré však pracujú efektívne najmä pre špecifické typy problémov, zahŕňajú intervalové zobecnenia algoritmov pracujúcich s reálnou aritmetikou (Gaussova eliminácia, Gauss-Seidelov algoritmus, Newtonova

metóda) alebo techniky využívajúce linearizáciu obmedzujúcich podmienok a algoritmy lineárneho programovania. Hlavnou nevýhodou týchto kontraktorov sú dodatočné požiadavky na typ obmedzujúcich podmienok alebo tvar celej sústavy a s tým súvisiaca obmedzenejšia množina problémov, ktoré dokážu riešiť (napr. kontraktory pre štvorcové alebo lineárne systémy). Mnohé z nich však prinášajú možnosť pracovať s celou množinou obmedzujúcich podmienok naraz a zvýšiť tak mieru dosiahnutej kontrakcie. Prehľad rôznych druhov kontraktorov poskytujú napríklad Jaulin et al. (2001, kap. 4).

4.2 Dopredný a spätný kontraktor

4.2.1 Rozklad na primitívne podmienky

Dopredný a spätný kontraktor (Benhamou et al., 1999) je založený na rozklade predpisu obmedzujúcej podmienky na jednoduché podvýrazy, ich vyhodnocovaní pomocou intervalovej aritmetiky a využití týchto hodnôt pre získanie nových definičných oborov jednotlivých premenných.

Definícia 4.3 (Primitívna podmienka)**.** Obmedzujúcu podmienku nazveme *primitívnou*, ak obsahuje nanajvýš jeden operátor $(+, -, \cdot, /)$ alebo elementárnu funkciu (napr. $\sin, \cos, \exp, \dots$).

Príklad. Uvažujme obmedzujúcu podmienku $x \cdot y + \frac{x}{z} = 0$. Túto podmienku môžeme rozložiť na primitívne podmienky zavedením pomocných premenných nasledovne:

$$\begin{aligned} a_1 &= x \cdot y \\ a_2 &= \frac{x}{z} \\ a_3 &= a_1 + a_2 \\ a_3 &= 0 \end{aligned}$$

Základný algoritmus kontraktoru pracuje v dvoch fázach – doprednej a spätnej. V doprednej fáze dochádza k rozkladu obmedzujúcej podmienky na primitívne podmienky a k ich vyhodnocovaniu intervalovou aritmetikou podľa priradených definičných oborov. V spätnej fáze prechádzame primitívne podmienky v opačnom poradí, izolujeme jednotlivé premenné a redukujeme ich definičné obory na základe hodnôt vypočítaných v doprednej fáze. Aby bola zaistená vlastnosť kontrakcie, v spätnej fáze tvoríme prienik získanej hodnoty s pôvodným definičným oborom premennej. Vlastnosť korektnosti je splnená z princípu využitia intervalovej aritmetiky.

Príklad. Uvažujme obmedzujúcu podmienku $x + y \cdot z = 0$ s definičnými obormi premenných $[-3, 2] \times [1, 2] \times [1, 2]$. Dopredná fáza kontraktoru prebehne v dvoch krokoch:

$$\begin{aligned} a_1 = y \cdot z \ &= [1, 2] \cdot [1, 2] = [1, 4] \\ a_2 = x + a_1 &= [-3, 2] + [1, 4] = [-2, 6] \end{aligned}$$

Na záver doplníme podmienku $a_2 = a_2 \cap [0, 0] = [-2, 6] \cap [0, 0] = [0, 0]$ vyjadrujúcu splnenie rovnosti.

V druhej fáze izolujeme jednotlivé premenné a dopočítame nové hodnoty ich definičných oborov:

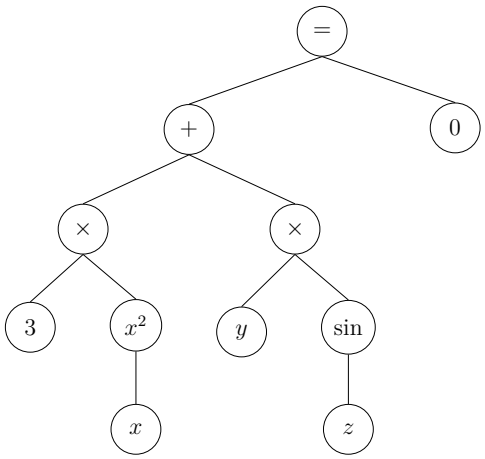
$$\begin{aligned} a_1 &= (a_2 - x) \cap a_1 = ([0,0] - [-3,2]) \cap [1,4] = [1,3] \\ x &= (a_2 - a_1) \cap x = ([0,0] - [1,3]) \cap [-3,2] = [-3,-1] \\ y &= \frac{a_1}{z} \cap y = \frac{[1,3]}{[1,2]} \cap [1,2] = [1,2] \\ z &= \frac{a_1}{y} \cap z = \frac{[1,3]}{[1,2]} \cap [1,2] = [1,2] \end{aligned}$$

Získali sme nové definičné obory premenných $[-3,-1] \times [1,2] \times [1,2]$.

Poznámka. Pre niektoré inverzné operácie použité v spätnej fáze môže byť výhodnejšie ponechať nespojitý obraz ako zjednotenie intervalov a vytvoriť intervalový obal až po prieniku s pôvodným intervalom. Napríklad ak je $\cos^{-1}(x)$ neprázdna množina, dostaneme vďaka periodicite funkcie cos ako intervalový obal celú množinu $\mathbb{R}$ a v podmienke typu $y = \cos^{-1}(x) \cap y$ by nedošlo k žiadnej kontrakcii. Množinu $\cos^{-1}(x) \cap y$ pre dané definičné obory $\boldsymbol{x}, \boldsymbol{y}$ preto definujeme ako najmenší interval obsahujúci $\{y \in \boldsymbol{y} \mid \cos(y) \in \boldsymbol{x}\}$.

4.2.2 Strom výrazu

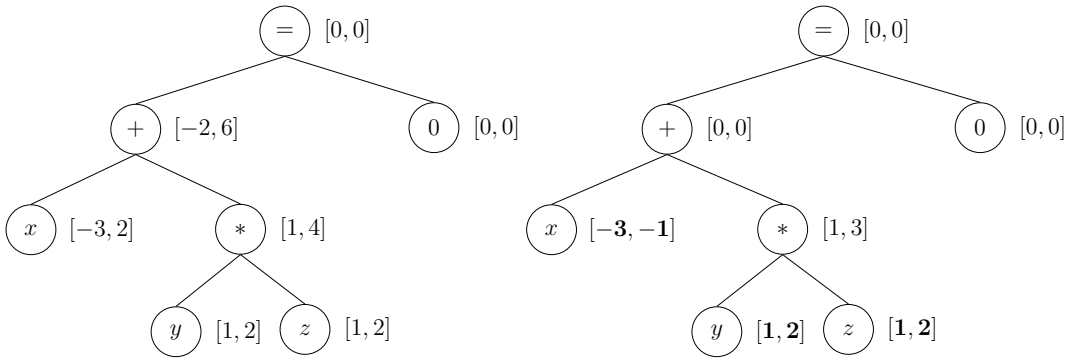
Alternatívny spôsob implementácie dopredného a spätného kontraktoru bez potreby rozkladu na primitívne podmienky umožňuje tzv. strom algebraického výrazu. Strom výrazu je zakorenený binárny (pri operáciach vyššej arity obecne n-árny) strom, v ktorom listy reprezentujú konštanty a premenné použité v danom výraze a vnútorné uzly reprezentujú operátory a funkcie.



Obr. 4.2: Strom výrazu pre podmienku $3x^2 + y \cdot \sin(z) = 0$

Pre prevod obmedzujúcej podmienky na strom výrazu využijeme algoritmus Shunting yard (Dijkstra, 1961) používaný na vyhodnotenie aritmetických výrazov v infixovej notácii. Algoritmus umožňuje v lineárnom čase previesť výraz do tzv. postfixovej notácie, ktorá je jednoduchšia na vyhodnotenie. Pridaním zásobníku, na ktorý budeme v priebehu prevodu ukladať časti stromu, dostaneme modifikovaný algoritmus umožňujúci postavenie stromu algebraického výrazu.

Vo vytvorenom strome rozlišujeme dva typy uzlov – vnútorné uzly (vrátane koreňa) a listy. Vnútorné uzly stromu nahrádzajú pomocné premenné, ktoré sme zavádzali pri rozklade na primitívne podmienky a zároveň reprezentujú operácie s nimi spojené. Listy reprezentujú konštanty a premenné v predpise obmedzujúcej podmienky. V každom uzle je uložený interval – pri inicializácii nastavíme intervaly vo vnútorných uzloch na $(-\infty, \infty)$, intervaly v uzloch s premennými na ich definičné obory a intervaly v uzloch s konštantami na degenerované intervaly ekvivalentné ich hodnote.



Obr. 4.3: Dopredná a spätná fáza v strome $(x + y \cdot z = 0)$

Pri doprednej fáze postupujeme od listov do koreňa a postupne priraďujeme hodnoty intervalom vo vnútorných uzloch na základe hodnôt v synoch a funkcie v danom uzle. Pre uzol s reláciou rovnosti použijeme rovnako ako v prípade rozkladu operáciu prieniku. Pre popis algoritmu doprednéhо a spätného kontraktoru zavedieme nasledujúce značenie:

- r.children je množina synov uzlu r,
- r.parent je rodičovský uzol r,
- r.siblings je množina synov rodiča uzlu r okrem samotného uzlu r.

Pre každý uzol r je v premennej r.interval uložená aktuálna hodnota intervalu pre premennú alebo konštantu, ktorú daný uzol reprezentuje. Doprednú fázu kontraktoru implementovaného pomocou stromu výrazu popisuje algoritmus 4.2.

Algoritmus 4.2 Dopredná fáza (`forwardphase`)

Vstup: strom výrazu s koreňom r, box $\boldsymbol{x}$
Výstup: ohodnotený strom výrazu s koreňom r

```
1: switch (r)
2:    case r je vnútorný uzol s funkciou f:
3:       for each r' ∈ r.children do
4:          forwardphase(r', x)
5:       r.interval ← [f](r.children.interval) // použi hodnoty intervalov v synoch
6:    case r je konštanta k:
7:       r.interval ← [k, k]
8:    case r je premenná x_i:
9:       r.interval ← x_i
10: end switch
```

Spätná fáza začína v koreni a postupuje k listom. Princíp je rovnaký ako pri rozklade na primitívne podmienky, opäť osamostatníme jednotlivé premenné a pomocou inverzných operácií dopočítame nové hodnoty intervalov, intervaly v uzloch aktualizujeme prienikom pôvodnej a novej hodnoty. Pokiaľ je v niektorom uzle prienik prázdny, nie sú pôvodné definičné obory so zadanou obmedzujúcou podmienkou konzistentné a úloha nemá riešenie.

Algoritmus 4.3 Spätná fáza (`backwardphase`)

Vstup: strom výrazu s koreňom r, box $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$
Výstup: kontrahovaný box $\boldsymbol{x}$

1: **switch** (r)
2: **case** r je vnútorný uzol alebo koreň**:**
3: **for each** $r' \in r$.children **do**
4: new $\leftarrow [f]^{-1}(r'.\text{parent}, r'.\text{siblings})$ // $[f]^{-1}$ je inverzná operácia k $[f]$
5: $r'.\text{interval} \leftarrow r'.\text{interval} \cap \text{new}$
6: $\boldsymbol{x} \leftarrow$ `backwardphase`$(r', \boldsymbol{x})$
7: **case** r je premenná x_i**:**
8: $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i \cap r.\text{interval}$
9: **end switch**

Spojením doprednej a spätnej fázy dostaneme dopredný a spätný kontraktor (viď algoritmus 4.4). Vyhodnotenie každej fázy vrátane tvorby stromu výrazu prebehne v čase lineárnom v dĺžke spracovávanej obmedzujúcej podmienky, algoritmus teda spĺňa požiadavok na polynomialitu kontraktoru.

Algoritmus 4.4 Dopredný a spätný kontraktor (`fbprop`)

Vstup: obmedzujúca podmienka c, box $\boldsymbol{x}$
Výstup: kontrahovaný box $\boldsymbol{x}$

1: $r \leftarrow$ `shuntingyard`(c)
2: $r \leftarrow$ `forwardphase`$(r, \boldsymbol{x})$
3: $\boldsymbol{x} \leftarrow$ `backwardphase`(r)

Definícia 4.4. Nech $x_1, \ldots, x_n$ sú premenné, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ príslušné definičné obory a c obmedzujúca podmienka v tvare $f(x_1, \ldots, x_n) = 0$. Definičný obor $\boldsymbol{x}_i$ premennej x_i je s podmienkou c *konzistentný vzhľadom k obalu* (angl. *hull consistent*), ak platí, že $\boldsymbol{x}_i$ je intervalový obal množiny

$$\{x_i \in \boldsymbol{x}_i \mid \exists x_1 \in \boldsymbol{x}_1, \ldots, \exists x_{i-1} \in \boldsymbol{x}_{i-1}, \\ \exists x_{i+1} \in \boldsymbol{x}_{i+1}, \ldots, \exists x_n \in \boldsymbol{x}_n : f(x_1, \ldots, x_n) = 0\}.$$

Veta 4.1 (Trombettoni et al. (2010))**.** *Buď c obmedzujúca podmienka v tvare $f(x) = 0$, kde f je spojitá na svojom definičnom obore $\boldsymbol{x} \in \mathbb{IR}^n$. Ak má každá premenná v predpise f unikátny výskyt, potom je na výstupe dopredného a spätného kontraktoru definičný obor každej premennej konzistentný s podmienkou c vzhľadom k obalu.*

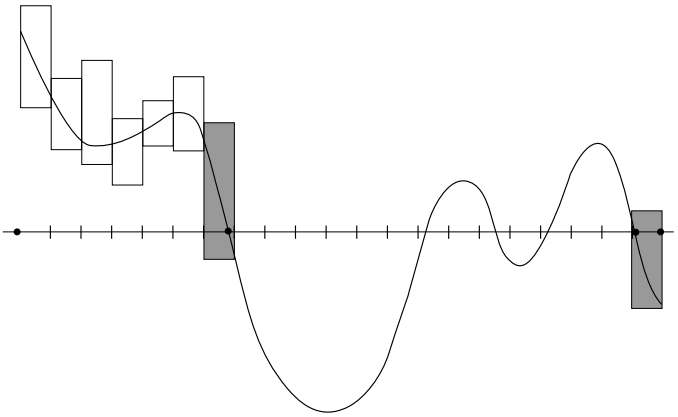
Vďaka tejto vlastnosti je algoritmus dopredného a spätného kontraktoru tiež často označovaný ako HC3 (v základnej variante), resp. HC4 (vo variante so stromom výrazu).

4.3 Kontraktor BoxNarrow

4.3.1 Metóda orezávania

Kontraktor BoxNarrow (Benhamou et al., 1994) spracúva podobne ako dopredný a spätný kontraktor jednu obmedzujúcu podmienku, avšak pracuje v nej iba s jednou zvolenou premennou. Ostatné premenné sú nahradené ich definičnými obormi, čím dostaneme obmedzujúcu podmienku s nelineárnou intervalovou funkciou jednej premennej.

Uvažujme obmedzujúcu podmienku v tvare $f(x) = 0$. Kontraktor sa snaží redukovať definičný obor zvolenej premennej x_i nájdením najľavejšieho a najpravejšieho nulového bodu príslušnej intervalovej funkcie $[f](x_i)$. Pri implementácii metódou orezávania rozdelí kontraktor definičný obor spracovávanej premennej na menšie disjunktné intervaly (až na hranice).



Obr. 4.4: Použitie kontraktoru BoxNarrow s metódou orezávania

Získané intervaly zoradíme od najmenšieho po najväčší a prechádzame v tomto poradí. Na každom intervale vyhodnotíme funkciu $[f]$ pomocou intervalovej aritmetiky a skontrolujeme, či sa v obraze nachádza 0. Ak 0 do intervalu nepatrí, opakujeme test pre ďalší interval v poradí. Ak sme našli interval obsahujúci 0, môžeme tento považovať za novú dolnú hranicu definičného oboru premennej x_i. Ďalej pokračujeme rovnakým postupom od najväčšieho intervalu, kým nenájdeme novú hornú hranicu.

Príklad. Uvažujme obmedzujúcu podmienku $x_1^2 - x_1 \cdot x_2 = 0$ na definičnom obore $[1, 10] \times [4, 50]$. Vytvoríme funkciu premennej x_1 nahradením premennej x_2 jej definičným oborom v predpise funkcie v obmedzujúcej podmienke:

$$f_1(x_1) = x_1^2 - x_1 \cdot [4, 50].$$

Pre túto funkciu nájdeme jej najľavejší a najpravejší nulový bod na intervale $[1, 10]$ postupným orezávaním hodnôt, čím dostaneme interval $[4, 10]$. Ďalej vytvoríme funkciu premennej x_2, v ktorej môžeme použiť už získaný redukovaný definičný obor premennej x_1.

$$f_2(x_2) = [4, 10]^2 - [4, 10] \cdot x_2$$

V druhom kroku hľadáme krajné nulové body $f_2(x_2)$ na definičnom obore $[4, 50]$. Hodnota $x_2 = 4$ je nulovým bodom, voľbou hodnoty 4 v prvom aj druhom intervale dostaneme rovnosť $4^2 - 4 \cdot 4 = 0$. Najpravejším nulovým bodom f_2 je $x_2 = 25$, novým definičným oborom druhej premennej je teda interval $[4, 25]$.

Kvôli prevodu presných čísel na intervaly v obmedzenej reprezentácii a nadhodnocovaniu obrazu pri intervalových výpočtoch sa môže stať, že algoritmus ukončí hľadanie aj v prípade, že sa v danom podintervale nenachádza nulový bod funkcie. Napríklad funkcia $x \cdot x + 1$ nemá na intervale $[-1, 1]$ žiadny nulový bod, avšak po dosadení a vyhodnotení intervalovou aritmetikou dostaneme obraz $[0, 2]$. V takomto prípade hovoríme o tzv. *kvazi-nulových bodoch* a v ďalšom texte budeme pod pojmom nulového bodu chápať aj tieto body.

Algoritmus 4.5 Kontraktor BoxNarrow (`boxnarrow`)

Vstup: obmedzujúca podmienka $f(x_1, \ldots, x_n) = 0$, box $\boldsymbol{x}$, presnosť ε
Výstup: kontrahovaný box $\boldsymbol{x}$

```
1: for i = 1 to n do
2:    f_i(x_i) ← [f](x_1, ..., x_{i-1}, x_i, x_{i+1}, ..., x_n)
3:    parts ← dividebox(x_i, w(x_i)/ε)
4:    for each p ∈ parts do // postup od dolnej hranice
5:       if 0 ∈ [f_i](p) then
6:          x_i ← p
7:          break
8:    for each p ∈ reverse(parts) do // postup od hornej hranice
9:       if 0 ∈ [f_i](p) then
10:         x̄_i ← p̄
11:         break
```

V algoritme 4.5 predpokladáme, že množina parts predstavuje zoradený zoznam, ktorý prechádzame v danom poradí. V najhoršom prípade môžeme v cykle prejsť celý definičný obor premennej x_i, pre redukciu oboru každej z n premenných tak potrebujeme nanajvýš $\frac{w(\boldsymbol{x}_i)}{\varepsilon}$ iterácií.

4.3.2 Intervalová Newtonova metóda

Pre hľadanie nulových bodov nelineárnej reálnej funkcie je tiež možné použiť tzv. *Newtonovu metódu*. V tejto sekcii odvodíme intervalovú analógiu Newtonovej metódy jednej premennej, ktorú je možné použiť ako podprocedúru kontraktoru BoxNarrow. Bližšie o intervalovej Newtonovej metóde jednej premennej, jej modifikáciách a komplexnejších odvodených metódach pojednávajú Hansen – Walster (2004).

Veta 4.2 (Veta o strednej hodnote)**.** *Buď $f(x)$ reálna funkcia spojitá na intervale $[a, b]$, ktorá má deriváciu v každom bode otvoreného intervalu (a, b). Potom existuje bod $c \in (a, b)$, pre ktorý platí $f'(c) = \frac{f(b)-f(a)}{b-a}$.*

Uvažujme $f(x) : \boldsymbol{x} = [\underline{x}, \overline{x}] \to \mathbb{R}$ spojite diferencovateľnú funkciu jednej reálnej premennej. Potom podľa vety 4.2 platí

$$f(b) = f(a) + f'(c)(b - a)$$

pre nejaké $c \in (a, b) \subseteq \boldsymbol{x}$. Predpokladajme, že existuje bod $x_0 \in \boldsymbol{x}$, pre ktorý platí $f(x_0) = 0$ a zvoľme $x \in \boldsymbol{x}$ ľubovoľne. Potom existuje $c \in \boldsymbol{x}$ tak, že platí:

$$0 = f(x_0) = f(x) + f'(c)(x_0 - x)$$
$$x_0 = x - \frac{f(x)}{f'(c)}$$

Uvažujme $[f']$ intervalové rozšírenie funkcie f', definujeme *intervalový Newtonov operátor* $N(\boldsymbol{x})$ nasledovne:

$$N(\boldsymbol{x}) \coloneqq x^c - \frac{f(x^c)}{[f'](\boldsymbol{x})}$$

Ďalej definujeme k-ty *Newtonov krok* pre $k \geq 1$:

$$\boldsymbol{x}_k \coloneqq \boldsymbol{x}_{k-1} \cap N(\boldsymbol{x}_{k-1})$$

Označme $\{\boldsymbol{x}_n\}_0^\infty$ postupnosť intervalových boxov vytvorenú iterovaním Newtonovho kroku. Nasledujúce dve tvrdenia zhŕňajú vlastnosti tejto postupnosti.

Veta 4.3 (Korektnosť INM)**.** *Ak existuje bod $x_0 \in \boldsymbol{x}_0$, pre ktorý platí $f(x_0) = 0$, potom $x_0 \in \boldsymbol{x}_k$ aj pre každé $k \geq 1$.*

Veta 4.4 (Konvergencia INM)**.** *Buď $x_0 \in \boldsymbol{x}_0$ bod, pre ktorý platí $f(x_0) = 0$ a nech $0 \notin [f'](\boldsymbol{x}_0)$. Potom $\{\boldsymbol{x}_n\}_0^\infty$ je zanorená postupnosť konvergujúca k x_0.*

Poznámka. Pri hľadaní nulových bodov funkcie pomocou intervalovej Newtonovej metódy môže byť výhodné definovať operáciu delenia pomocou rozšírenej intervalovej aritmetiky. Pre nedegenerovaný interval $\boldsymbol{x}$ obsahujúci 0 definujeme prevrátenú hodnotu v rozšírenej intervalovej aritmetike:

(a) ak $\underline{x} = 0$ a $\overline{x} > 0$, potom $\dfrac{1}{\boldsymbol{x}} \coloneqq \left[\dfrac{1}{\overline{x}}, \infty\right)$,

(b) ak $\underline{x} < 0$ a $\overline{x} = 0$, potom $\dfrac{1}{\boldsymbol{x}} \coloneqq \left(-\infty, \dfrac{1}{\underline{x}}\right]$,

(c) ak $\underline{x} < 0$ a $\overline{x} > 0$, potom $\dfrac{1}{\boldsymbol{x}} \coloneqq \left(-\infty, \dfrac{1}{\underline{x}}\right] \cup \left[\dfrac{1}{\overline{x}}, \infty\right)$.

Rozšírený podiel dvoch intervalov $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{IR}$, $0 \in \boldsymbol{x}$ potom zavedieme ako

$$\frac{\boldsymbol{y}}{\boldsymbol{x}} \coloneqq \boldsymbol{y} \cdot \frac{1}{\boldsymbol{x}}$$

Pre delenie intervalom neobsahujúcim 0 používame naďalej definíciu podľa klasickej intervalovej aritmetiky. Operácia delenia degenerovaným intervalom $[0, 0]$ zostáva nedefinovaná.

Na základe odvodenej metódy môžeme kontraktor BoxNarrow upraviť tak, aby pre redukciu definičného oboru zvolenej premennej využíval intervalovú Newtonovu metódu implementovanú v procedúre `leftnarrow` pre nájdenie novej dolnej hranice intervalu $\boldsymbol{x}_i$ (viď algoritmus 4.6) a obdobne vytvorenú procedúru `rightnarrow` pre nájdenie novej hornej hranice. Funkciu `intnewton` tvorí cyklus,

v ktorom je vykonávaný Newtonov krok (so základnou intervalovou aritmetikou), pokým nedosiahneme požadovanú presnosť riešenia.

Algoritmus 4.6 Podprocedúra kontraktoru BoxNarrow (`leftnarrow`)

Vstup: $f_i : \mathbb{IR} \to \mathbb{IR}, \boldsymbol{x}_i \in \mathbb{IR}$, presnosť ε
Výstup: kontrahovaný interval $\boldsymbol{x}_i$

```
1: if 0 ∉ [f_i](x_i) then
2:    return ∅
3: x_i ← intnewton(x_i, f_i, ε)
4: if w(x_i) < ε then
5:    return x_i
6: (x_{i_1}, x_{i_2}) ← dividebox(x_i)
7: x_i ← leftnarrow(x_{i_1}, [f_i], ε)
8: if x_i = ∅ then
9:    return leftnarrow(x_{i_2}, [f_i], ε)
10: else
11:    return x_i
```

Definícia 4.5. Nech $x_1, \ldots, x_n$ sú premenné, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ príslušné definičné obory, c obmedzujúca podmienka v tvare $f(x_1, \ldots, x_n) = 0$ a $[f]$ intervalové rozšírenie funkcie f. Definičný obor $\boldsymbol{x}_i$ premennej x_i je s podmienkou c *konzistentný vzhľadom k boxu* (angl. *box consistent*), ak platí, že $\boldsymbol{x}_i$ je intervalový obal množiny

$$\{x_i \in \boldsymbol{x}_i \mid 0 \in [f](\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, [x_i], \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_n)\},$$

kde $[x_i]$ označuje najmenší reprezentovateľný interval (pri obmedzenej presnosti) obsahujúci x_i.

Veta 4.5 (Trombettoni et al. (2010))**.** *Buď c obmedzujúca podmienka v tvare $f(x) = 0$ s definičným oborom $\boldsymbol{x} \in \mathbb{IR}^n$. Ak je f spojitá, jej predpis obsahuje nanajvýš jednu premennú s viacnásobným výskytom, potom je vo výstupe kontraktoru BoxNarrow (s presnosťou rovnou šírke najmenšieho reprezentovateľného intervalu) definičný obor každej premennej konzistentný s podmienkou c vzhľadom k obalu. V obecnom prípade je získaný definičný obor konzistentný s podmienkou c vzhľadom k boxu.*

4.4 Kombinácia kontraktorov

Algoritmus dopredného a spätného kontraktoru predstavuje rýchly a efektívny spôsob redukcie definičných oborov obmedzujúcich podmienok, v ktorých sa každá premenná vyskytuje nanajvýš jedenkrát. Technika používaná v kontraktore BoxNarrow čiastočne rieši problém závislosti premenných a umožňuje efektívnu redukciu definičných obor v prípade, že má niektorá z premenných viacnásobný výskyt. Redukcia pomocou kontraktoru BoxNarrow je však výpočtovo náročnejšia než pri použití dopredného a spätného kontraktoru, ktorý pri unikátnom výskyte premenných vydá na výstupe rovnako dobrý výsledok ako BoxNarrow.

Tieto vlastnosti vedú k myšlienke vytvorenia nového algoritmu s rýchlosťou dopredného a spätného kontraktoru pre jednoduchšie problémy a efektivitou kontraktoru BoxNarrow pre zložitejšie. Využitie priameho vyhodnocovania pomocou intervalovej aritmetiky je preferované voči zložitejšej intervalovej Newtonovej metóde pre kontrakciu definičných oborov premenných s unikátnym výskytom. Kontraktor BoxNarrow je použitý iba v prípade, ak nie je použitie dopredného a spätného kontraktoru postačujúce (viacnásobný výskyt premenných).

Kontraktor založený na tejto myšlienke po prvýkrát predstavili Benhamou et al. (1999) pod označením BC4 ako kombináciu algoritmov HC4 (dopredný a spätný kontraktor so stromovou reprezentáciou) a BC3 (BoxNarrow s intervalovou Newtonovou metódou). V algoritme 4.7 uvažujeme upravenú verziu funkcie `boxnarrow`, ktorá redukuje definičný obor iba pre zvolenú premennú.

Algoritmus 4.7 Kombinovaný kontraktor BC4

Vstup: $\boldsymbol{x} \in \mathbb{IR}^n$, obmedzujúce podmienky $C = \{c_1, \ldots, c_m\}$
Výstup: kontrahovaný box $\boldsymbol{x}$

1: **repeat**
2: $\quad \boldsymbol{y} \leftarrow \boldsymbol{x}$
3: $\quad$ **do**
4: $\quad\quad$ cont $\leftarrow$ **false**
5: $\quad\quad$ **for each** $c \in C$ **do**
6: $\quad\quad\quad X_1 \leftarrow \{x \in \mathrm{var}(c) \mid x$ má unikátny výskyt v $c\}$
7: $\quad\quad\quad \boldsymbol{z} \leftarrow \boldsymbol{x}$
8: $\quad\quad\quad \boldsymbol{x} \leftarrow \mathtt{fbprop}(\boldsymbol{x}, c)$
9: $\quad\quad\quad$ cont $\leftarrow \boldsymbol{x} \neq \emptyset$ **and** $((\exists i : \boldsymbol{x}_i \neq \boldsymbol{z}_i$ **and** $x_i \in X_1)$ **or** cont)
10: $\quad$ **while** cont
11: $\quad$ **if** $\boldsymbol{x} \neq \emptyset$ **then**
12: $\quad\quad X_2 \leftarrow \{x \in \mathrm{var}(c) \mid x$ má v c viacnásobný výskyt$\}$
13: $\quad\quad$ **for each** $x \in X_2$ **do**
14: $\quad\quad\quad \boldsymbol{x} \leftarrow \mathtt{boxnarrow}(\boldsymbol{x}, c, x)$
15: **until** $\boldsymbol{x} = \boldsymbol{y}$ **or** $\boldsymbol{x} = \emptyset$

Ďalšie možné rozšírenie algoritmu 4.7 uvádza Granvilliers (2000) v podobe algoritmu BC5, ktorý k doprednému a spätnému kontraktoru a BoxNarrow pridáva intervalovú Gauss-Seidelovu metódu. Táto metóda sa používa pre riešenie štvorcových sústav lineárnych rovníc, preto je potrebná linearizácia obmedzujúcich podmienok a prevod na štvorcovú sústavu.

4.5 Kontraktor MoHC

4.5.1 Obraz monotónnych funkcií

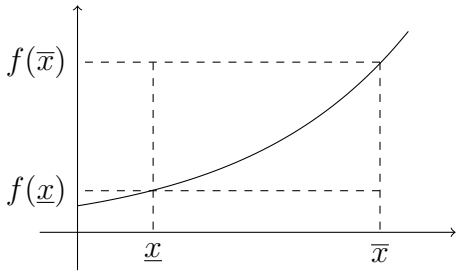
Kontraktor MoHC (angl. Monotonic Hull Consistency, Araya et al. (2009)) je založený na využití monotónnosti funkcií v obmedzujúcich podmienkam vzhľadom k niektorým premenným. Buď f reálna funkcia n premenných $x_1, \ldots, x_n$ na definičnom obore $\boldsymbol{x} \in \mathbb{IR}^n$. Ak má f na intervale $\boldsymbol{x}_i$ kladnú parciálnu deriváciu podľa x_i, potom je *rastúca v premennej* x_i, ak má túto parciálnu deriváciu zápornú, je

klesajúca v premennej x_i. Funkcia je *monotónna v premennej* x_i, ak je v nej rastúca alebo klesajúca a funkcia je *monotónna*, ak je monotónna v každej premennej. V priebehu algoritmu MoHC budeme detekovať monotónnosť funkcie na základe jej parciálnych derivácií na definičnom obore. Metódy pre výpočet parciálnych derivácií funkcie na intervale popisuje napríklad Araya (2010, kap. 2.2.4).

Uvažujme spojitú reálnu funkciu $f(x)$ jednej premennej s definičným oborom $\boldsymbol{x} \in \mathbb{IR}$. Optimálny intervalový obraz tejto funkcie tvorí množina

$$f(\boldsymbol{x}) = \{f(x) \mid x \in \boldsymbol{x}\} = \left[\min_{x \in \boldsymbol{x}} f(x), \max_{x \in \boldsymbol{x}} f(x)\right].$$

Ak je f monotónna, maximum a minimum nadobúda v krajných bodoch intervalu. Pre rastúcu funkciu f dostaneme popis optimálneho obrazu $f(\boldsymbol{x}) = [f(\underline{x}), f(\overline{x})]$, ak je f klesajúca naopak platí $f(\boldsymbol{x}) = [f(\overline{x}), f(\underline{x})]$. Takto môžeme pre monotónne funkcie dostať tesnejší obraz, než napríklad použitím prirodzeného intervalového rozšírenia.



Obr. 4.5: Optimálny obraz monotónnej funkcie

Vlastnosť monotónnosti je možné využiť aj pre nájdenie obrazu funkcie viacerých premenných. Pre funkciu f a jej premennú x s definičným oborom $\boldsymbol{x}$ zavedieme nasledujúce značenie:

- ak je f rastúca v x, potom $x^+ \coloneqq \overline{x}$ a $x^- \coloneqq \underline{x}$,
- ak je f klesajúca v x, potom $x^+ \coloneqq \underline{x}$ a $x^- \coloneqq \overline{x}$.

Veta 4.6 (Araya (2010))**.** *Buď* $f(x_1, \ldots, x_n)$ *spojitá reálna funkcia na definičnom obore* $\boldsymbol{x} \in \mathbb{IR}^n$*, ktorá je na* $\boldsymbol{x}$ *monotónna vo všetkých premenných. Potom optimálny obraz funkcie* f *na* $\boldsymbol{x}$ *je rovný*

$$f(\boldsymbol{x}) = \left[f(x_1^-, \ldots, x_n^-), f(x_1^+, \ldots, x_n^+)\right].$$

Monotónnosť funkcií je možné využiť pre nájdenie tesnejšieho obrazu funkcie aj v prípade, že je rastúca alebo klesajúca iba v niektorých premenných. Pre takýto typ funkcií definujeme intervalové rozšírenie založené na monotónnosti funkcie v jednotlivých premenných.

Definícia 4.6 (Rozšírenie podľa monotónnosti)**.** Buď $f(x_1, \ldots, x_n)$ reálna funkcia s definičným oborom $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ a $[f]$ jej prirodzené intervalové rozšírenie. Označme $\boldsymbol{x}^-$ (resp. $\boldsymbol{x}^+$) box, ktorý vznikne z $\boldsymbol{x}$ nahradením definičného oboru každej premennej, v ktorej je f rastúca, hodnotou $\underline{x}$ (resp. $\overline{x}$) a každej premennej, v ktorej je f klesajúca, hodnotou $\overline{x}$ (resp. $\underline{x}$). Potom jej *rozšírenie podľa monotónnosti* na obore $\boldsymbol{x}$ definujeme ako:

$$[f]_m(\boldsymbol{x}) \coloneqq \left[\underline{[f](\boldsymbol{x}^-)}, \overline{[f](\boldsymbol{x}^+)}\right]$$

Rozšírenie podľa monotónnosti dáva obecne aspoň tak tesný odhad obrazu funkcie na intervale, ako prirodzené intervalové rozšírenie. Pre funkcie monotónne vo všetkých premenných s viacnásobným výskytom platí silnejšie tvrdenie, ktoré popisuje veta 4.7.

Veta 4.7 (Araya (2010))**.** *Buď f spojitá reálna funkcia na definičnom obore $\boldsymbol{x} \in \mathbb{IR}^n$, ktorá je na $\boldsymbol{x}$ monotónna vo všetkých premenných s viacnásobným výskytom v jej predpise. Potom intervalové rozšírenie f podľa monotónnosti dáva optimálny obraz f na $\boldsymbol{x}$.*

4.5.2 Algoritmus kontraktoru

Uvažujme obmedzujúcu podmienku $f(x_1, \ldots, x_n) = 0$ a definičný obor jej premenných $\boldsymbol{x} \in \mathbb{IR}^n$. Označme X_1 množinu všetkých premenných s unikátnym výskytom v predpise f a X_2 množinu všetkých premenných s viacnásobným výskytom. Kontraktor MoHC pre redukciu definičných oborov premenných v tejto obmedzujúcej podmienke popisuje algoritmus 4.8.

Algoritmus 4.8 Kontraktor MoHC (`mohc`)

Vstup: $\boldsymbol{x} \in \mathbb{IR}^n, f : \mathbb{IR}^n \to \mathbb{IR}$, množiny premenných X_1 a X_2, τ, presnosť ε
Výstup: kontrahovaný box $\boldsymbol{x}$

1: $\boldsymbol{x} \leftarrow$ `fbprop`$(f(x_1, \ldots, x_n) = 0, \boldsymbol{x})$
2: **if** $X_2 \neq \emptyset$ **and** $\rho[f] < \tau$ **then**
3: $\quad \mathbf{g} \leftarrow$ `getgradient`$(f, X_2, \boldsymbol{x})$
4: $\quad f_{max}, f_{min}, Y, X_2 \leftarrow$ `getmonovars`$(f, X_2, \boldsymbol{x}, \mathbf{g})$
5: $\quad \boldsymbol{x} \leftarrow$ `fbprop`$(f_{min}(x_1, \ldots, x_n) \leq 0, \boldsymbol{x})$
6: $\quad \boldsymbol{x} \leftarrow$ `fbprop`$(f_{max}(x_1, \ldots, x_n) \geq 0, \boldsymbol{x})$
7: $\quad \boldsymbol{x} \leftarrow$ `monoboxnarrow`$(\boldsymbol{x}, f_{max}, f_{min}, Y, \mathbf{g}, \varepsilon)$

Kontraktor MoHC využíva pre redukciu dopredný a spätný kontraktor a modifikovanú verziu kontraktoru BoxNarrow pre monotónne funkcie. Po úvodnom volaní dopredného a spätného kontraktoru zistíme, či predpis zadanej obmedzujúcej funkcie obsahuje premenné s viacnásobným výskytom. Zároveň je možné použiť kontrolu podmienky $\rho[f] < \tau$, kde hodnota $\rho[f]$ udáva pomer obrazu založeného na rozšírení podľa monotónnosti a obrazu získaného pomocou prirodzeného rozšírenia a parameter τ je určený užívateľom a rozhoduje o frekvencii volania procedúr založených na monotónnosti funkcií.

Pomocná funkcia `getgradient` vracia zoznam parciálnych derivácií funkcie f na množine $\boldsymbol{x}$ podľa premenných s viacnásobným výskytom v predpise f. Ďalšou pomocnou funkciou je funkcia `getmonovars`, ktorá zabezpečuje výber monotónnych premenných s viacnásobným výskytom na základe získaných parciálnych derivácií. Ak je niektorá premenná monotónna, je presunutá do novej množiny Y, inak je ponechaná v množine X_2. Funkcia `getmonovars` ďalej vracia dve funkcie f_{max} a f_{min}, kde sú premenné, v ktorých je f rastúca nahradené ich dolnými (f_{min}), resp. hornými (f_{max}) hranicami a obdobne pre premenné, v ktorých f klesá.

Volanie kontraktoru `fbprop` v riadku 5 a 6 zabezpečuje redukciu definičných oborov pre premenné s unikátnym výskytom a pre premenné s viacnásobným výskytom, u ktorých nebola zistená monotónnosť. K redukcii oborov monotónnych

premenných s viacnásobným výskytom v predpise f je určená funkcia `monoboxnarrow`, ktorá je upravenou verziou kontraktoru BoxNarrow.

Algoritmus 4.9 BoxNarrow s využitím monotónnosti (`monoboxnarrow`)

Vstup: $\boldsymbol{x} \in \mathbb{IR}^n$, f_{max}, f_{min}, množina monotónnych premenných X, intervalový gradient $\mathbf{g}$, presnosť ε
Výstup: kontrahovaný box $\boldsymbol{x} \in \mathbb{IR}^n$

```
1: for each x_i ∈ X do
2:   if [f_min](x) < 0 and applyfmax[i] then
3:     if g_i > 0 then // f je rastúca v premennej x_i
4:       leftnarmax(x_i, f_max^{x_i}, g_i, ε)
5:     else if g_i < 0 then // f je klesajúca v premennej x_i
6:       rightnarmax(x_i, f_max^{x_i}, g_i, ε)
7:   if [f_max](x) > 0 and applyfmin[i] then
8:     if g_i > 0 then
9:       rightnarmin(x_i, f_min^{x_i}, g_i, ε)
10:    else if g_i < 0 then
11:      leftnarmin(x_i, f_min^{x_i}, g_i, ε)
```

Funkcie s prefixom `leftnar` a `rightnar` sú určené na hľadanie tesnejšej dolnej, resp. hornej hranice definičného oboru premennej. Všetky štyri pomocné funkcie používajú rovnako ako v prípade kontraktoru BoxNarrow pre nájdenie novej hranice intervalovú Newtonovu metódu, líšia sa iba vo funkcii, s ktorou pracujú (podľa toho, či je f v premennej x_i rastúca alebo klesajúca). Booleovské polia applyfmax a applyfmin obsahujú pre každú monotónnu premennú hodnotu označujúcu, či ďalšie volanie pomocnej funkcie môže znovu zlepšiť príslušnú hranicu jej definičného oboru. Tieto polia sú inicializované hodnotou *true* pre každú premennú a menia sa pri volaní pomocných funkcií. Konkrétnu implementáciu pomocných funkcií, ako aj ďalšie modifikácie a vylepšenia kontraktoru MoHC popisujú Araya et al. (2009).

5 Porovnanie kontraktorov

Táto sekcia je venovaná porovnaniu efektivity implementovaných algoritmov na konkrétnych príkladoch. Testované budú dva základné kontraktory (dopredný a spätný kontraktor, BoxNarrow), ktoré sú často využívané ako jadro komplexnejších kontraktorov. Pre porovnanie uvedieme tiež výsledky dosiahnuté základnou variantou algoritmu Sivia bez použitia kontraktorov.

5.1 Návrh testu

Porovnanie kontraktorov prebehne na niekoľkých spojitých CSP s rôznym počtom premenných a obmedzujúcich podmienok. Problémy boli zvolené tak, aby zahŕňali širšiu škálu nelineárnych funkcií (polynómy, logaritmy, goniometrické funkcie apod.). Časť problémov je prevzatá z bežne používaných testov v intervalovej analýze (viď Merlet (2007)), ďalšie pridané problémy poukazujú na rozdielne vlastnosti testovaných algoritmov. Algoritmy sú porovnávané na základe doby výpočtu daného problému, počtu potrebných iterácií hlavného cyklu algoritmu Sivia a počtu výsledných intervalových boxov popisujúcich riešenie problému. Nasleduje zoznam použitých problémov s ich krátkym popisom:

RumpUni Exponenciálna nerovnica jednej premennej, ktorej všetky riešenia sú obsiahnuté v intervale $[0, 1.2565]$.
Definičný obor: $[-10^8, 10^8]$
Obmedzujúca podmienka: $e^x - 2x - 1 \leq 0$

LogSqrt Sústava dvoch nerovníc obsahujúca logaritmus a odmocninu, riešením je $x = 1$ a $y \in [0.5, 10]$.
Definičný obor: $[0, 1] \times [0, 10]$
Obmedzujúce podmienky: $\log x + \sqrt{x} \geq x, y + y \geq x$

Ring Sústava dvoch nerovníc dvoch premenných popisujúca medzikružie s polomermi $\sqrt{0.5}$ a $\sqrt{2}$.
Definičný obor: $[-10, 10] \times [-10, 10]$
Obmedzujúce podmienky: $x^2 + y^2 \leq 2, x^2 + y^2 \geq 0.5$

Empty Sústava dvoch polynomiálnych nerovníc dvoch premenných s prázdnou množinou riešení.
Definičný obor: $[0, 10^5] \times [0, 10^5]$
Obmedzujúce podmienky: $x^3 + y \geq 2, x^3 + y \leq 1$

NumAn Problém z oblasti numerickej analýzy obsahujúci racionálnu lomenú funkciu, sústava má na zadanom intervale dve riešenia (približne $0.2807, 125.12$ a $0.3889, 147.278$).
Definičný obor: $[-10^3, 0.99] \times [-10^8, 10^8]$

Obmedzujúce podmienky:

$$x - 0.0000179297550y^2 = 0$$
$$y - \frac{90}{1-x} = 0$$

Wings Sústava dvoch polynomiálnych nerovníc popisujúca nesúvislú množinu s dvomi komponentami v rovine.
Definičný obor: $[-5,5] \times [-5,5]$
Obmedzujúce podmienky: $x^2 - y \geq 2, x^2 + y^2 \leq 3$

Cube Sústava troch nerovníc popisujúca kocku v priestore dimenzie 3 s hranou dĺžky 10.
Definičný obor: $[-10,10] \times [-10,10] \times [-10,10]$
Obmedzujúce podmienky: $|x| \leq 5, |y| \leq 5, |z| \leq 5$

Trig Sústava troch rovníc s tromi premennými obsahujúca základné goniometrické funkcie, úloha má jedno riešenie $x = 0, y = \frac{1}{3}, z = 0$.
Definičný obor: $[-10^8, 10^8] \times [-10^8, 10^8] \times [-10^8, 10^8]$
Obmedzujúce podmienky:

$$-\frac{1}{81}\cos x + \frac{1}{9}y^2 + \frac{1}{3}\sin z - x = 0$$
$$\frac{1}{3}\sin x + \frac{1}{3}\cos z - y = 0$$
$$-\frac{1}{9}\cos x + \frac{1}{3}y + \frac{1}{6}\sin z - z = 0$$

Sum4 Sústava štyroch rovníc, v každej sa vyskytuje iba jedna premenná, úloha má na danom definičnom obore 16 riešení.
Definičný obor: $[-10^6, 10^6] \times [-10^6, 10^6] \times [-10^6, 10^6] \times [-10^6, 10^6]$
Obmedzujúce podmienky:

$$w \cdot w + w + w + w = 3$$
$$x \cdot x + x + x + x = 4$$
$$y \cdot y + y + y + y = 5$$
$$z \cdot z + z + z + z = 6$$

Chemk Sústava 4 rovníc so 4 neznámymi, úloha má na danom definičnom obore jedno riešenie (približne $w = 0.3845, x = 0.0002, y = 2.5217 \cdot 10^{-8}, z = 0.1479$).
Definičný obor: $[0,1] \times [0,1] \times [0,1] \times [0,1]$
Obmedzujúce podmienky:

$$x^2 - y = 0$$
$$w^2 - z = 0$$
$$2.177 \cdot 10^7 y - 1.697 \cdot 10^7 yw + 0.55xw + 0.45x - w = 0$$
$$1.585 \cdot 10^{14} yw + 4.126 \cdot 10^7 xz - 8.282 \cdot 10^6 xw + 2.284 \cdot 10^7 z\ w -$$
$$-1.918 \cdot 10^7 z + 48.4w - 27.73 = 0$$

5.2 Výsledky testu

Tabuľka 5.1 uvádza doby výpočtu riešenia testových problémov pre jednotlivé algoritmy. Namerané časy sú uvedené v sekundách, pri každom probléme je tiež špecifikovaná požadovaná presnosť riešenia. Algoritmus SIVIA a kontraktor BoxNarrow boli implementované priamo v jazyku MATLAB, zatiaľ čo dopredný a spätný kontraktor tvorí MEX-súbor jazyka C++. Pri všetkých testoch bol použitý procesor Intel Core i5-2410M 2.30 GHz.

Problém	ε	SIVIA bez kontraktoru	Dopredný a spätný kontr.	Kontr. BoxNarrow
LogSqrt	0.1	2.746880	2.820861	**0.527249**
Rump	0.01	0.500589	**0.453978**	3.315279
Ring	0.5	**0.597691**	0.753629	5.400360
Empty	0.2	0.217248	**0.024391**	0.730911
NumAn	0.01	62.381886	**0.467183**	17.280301
Wings	0.2	**0.746831**	1.219425	14.03759
Cube	0.1	913.919072	**0.044959**	0.750350
Trig	0.1	3021.063782	**0.105020**	4.931279
Sum4	0.1	44.905380	26.607079	**5.930470**
Chemk	0.01	187.642470	**0.816312**	33.104256

Tabuľka 5.1: Porovnanie kontraktorov podľa doby výpočtu

Z uvedených dát plynie, že najlepšie výsledky vzhľadom na dobu výpočtu dosahuje na testových problémoch dopredný a spätný kontraktor. Hlavnou výhodou tohto kontraktoru je jeho jednoduchý priebeh (opakované vyhodnocovanie primitívnych výrazov pomocou intervalovej aritmetiky) a možnosť redukovať definičné obory všetkých premenných počas jedného prechodu stromom výrazu. Kontraktor BoxNarrow dosiahol lepšie výsledky v niektorých problémoch s opakovaným výskytom jednej premennej, avšak pri opakovanom výskyte viacerých premenných nie je redukcia získaná použitím tohto kontraktoru a jeho zložitejším redukčným algoritmom dostatočná. Význam použitia kontraktorov je viditeľný hlavne pri riešení problémov vo vyššej dimenzii a s malým počtom riešení (napr. Chemk, Trig), v ktorých kontraktory dosahujú výrazne lepšie výsledky než základný algoritmus SIVIA.

Problém	ε	SIVIA bez kontraktoru	Dopredný a spätný kontr.	Kontr. BoxNarrow
LogSqrt	0.1	1373 (0/441/246)	489 (0/490/122)	1 (1/1/0)
Rump	0.01	205 (25/71/7)	53 (23/17/4)	47 (20/2/4)
Ring	0.5	279 (16/68/56)	95 (12/32/36)	95 (12/32/36)
Empty	0.2	75 (0/38/0)	1 (0/1/0)	1 (0/1/0)
NumAn	0.01	23393 (0/9115/2582)	3 (0/12/2)	3 (0/12/2)
Wings	0.2	359 (14/82/84)	159 (18/108/62)	159 (18/112/62)
Cube	0.1	361951 (8/81120/99848)	1 (1/6/0)	1 (1/6/0)
Trig	0.1	305599 (0/152796/4)	1 (0/2/1)	1 (0/6/1)
Sum4	0.1	6967 (0/3376/108)	2029 (0/1342/30)	31 (0/110/16)
Chemk	0.01	20453 (0/7945/2282)	15 (0/42/1)	27 (0/51/5)

Tabuľka 5.2: Porovnanie kontraktorov podľa počtu iterácií a popisu riešenia

V tabuľke 5.2 je uvedený počet iterácií hlavného cyklu algoritmu SIVIA potrebných pre získanie popisu riešenia testovaných problémov s požadovanou presnosťou ε. V zátvorke je uvedený počet intervalových boxov použitých príslušným

algoritmom pre popis riešenia problému. Počet boxov je uvádzaný v tvare (podmnožina riešenia/boxy nepatriace do riešenia/nezaradené boxy). Zo získaných dát vidíme, že kontraktory dokážu pri niektorých problémoch už v prvej iterácii algoritmu zistiť, že je množina riešení prázdna (problém Empty) alebo nájsť jeho približné či presné riešenie (LogSqrt, Cube, Trig). V jednej iterácii cyklu sa pri použití kontraktorov vykonáva väčší počet operácií, nižší počet iterácií tak nemusí nutne znamenať kratšiu dobu výpočtu. Výhodou kontraktorov však môže byť v určitom zmysle presnejší popis riešenia, obsahujúci napríklad menší počet nezaradených boxov.

6 Využitie: komplexné intervaly

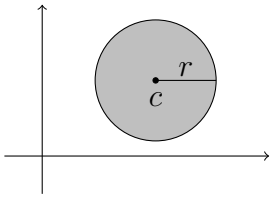
Táto kapitola bude venovaná predstaveniu ďalšej možnosti využitia algoritmu Sivia – vizualizácii základných aritmetických operácií nad množinou komplexných intervalov. V úvode predstavíme rôzne spôsoby definovania komplexných intervalov, zavedieme na nich základnú aritmetiku a stručne popíšeme jej vlastnosti. Pre obdĺžnikové komplexné intervaly potom odvodíme spojitý problém splňovania obmedzujúcich podmienok, ktorý bude popisovať presné operácie násobenia a delenia.

6.1 Reprezentácia komplexných intervalov

Podobne ako je možné rozšíriť množinu reálnych čísel na množinu komplexných čísel, môže byť užitočné zaviesť pojem intervalu komplexných čísel a aritmetické operácie nad množinou týchto intervalov. Existujú tri najčastejšie používané prístupy k definícii komplexného intervalu. Všetky tri prístupy popisujú komplexné intervaly ako uzavreté a obmedzené podmnožiny komplexnej roviny, líšia sa však v konkrétnej podobe týchto množín a vo vlastnostiach komplexnej intervalovej aritmetiky na nich definovanej.

6.1.1 Kruhové komplexné intervaly

Definícia 6.1 (Gargantini – Henrici (1971))**.** Buď $c \in \mathbb{C}, r \in \mathbb{R}, r \geq 0$. *(Kruhový) komplexný interval* je množina $\mathbf{z} \coloneqq \{z \in \mathbb{C} \mid |z - c| \leq r\}$. Hodnotu c nazveme *stred* komplexného intervalu $\mathbf{z}$ a r jeho *polomer*.



Obr. 6.1: Kruhový komplexný interval

Ďalej zavedieme základné aritmetické operácie nad množinou kruhových komplexných intervalov. Súčet a rozdiel dvoch komplexných intervalov $\boldsymbol{x} = (c_x, r_x)$, $\boldsymbol{y} = (c_y, r_y)$ definujeme priamočiaro ako intervaly

$$\boldsymbol{x} + \boldsymbol{y} \coloneqq (c_x + c_y, r_x + r_y),$$
$$\boldsymbol{x} - \boldsymbol{y} \coloneqq (c_x - c_y, r_x + r_y).$$

Definícia 6.2. Buď $x = a + bi \in \mathbb{C}$, potom *absolútnu hodnotu* x definujeme ako $|x| \coloneqq \sqrt{a^2 + b^2}$.

Definícia 6.3. Buď $x = a + bi \in \mathbb{C}$, potom *komplexne združené číslo* k x definujeme ako $a - bi$ a značíme ho $\mathrm{conj}(x)$.

Pre súčin dvoch kruhových komplexných intervalov sa najčastejšie používa nasledujúca definícia:

$$\boldsymbol{x} \cdot \boldsymbol{y} \coloneqq (c_x \cdot c_y, |c_x| r_y + |c_y| r_x + r_x r_y).$$

Podiel $\frac{\boldsymbol{x}}{\boldsymbol{y}}$ definujeme ako súčin $\boldsymbol{x} \cdot \frac{1}{\boldsymbol{y}}$ pomocou prevrátenej hodnoty $\boldsymbol{y}$ (predpokladáme $0 \notin \boldsymbol{y}$):

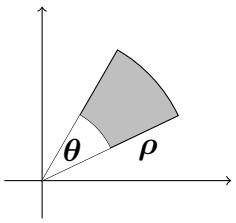
$$\frac{1}{\boldsymbol{y}} \coloneqq \left(\frac{\mathrm{conj}(c_y)}{|c_y|^2 - r_y^2}, \frac{r_y}{|c_y|^2 - r_y^2} \right).$$

Takto definovaný súčin, resp. podiel je obecne nadmnožinou množiny všetkých súčinov, resp. podielov jednotlivých prvkov z $\boldsymbol{x}$ a $\boldsymbol{y}$.

6.1.2 Polárne komplexné intervaly

Druhá definícia komplexných intervalov je založená na myšlienke prevodu na polárny tvar, ktorý sa používa ako alternatívna reprezentácia komplexných čísel. Polárny tvar popisuje komplexné číslo pomocou jeho vzdialenosti ρ od počiatku a uhlu φ, ktorý jeho spojnica s počiatkom zviera s kladnou reálnou osou. Komplexné číslo z potom môžeme uviesť v tvare $z = \rho e^{\varphi i}$.

Definícia 6.4 (Candau et al. (2006))**.** Nech $\boldsymbol{\rho}, \boldsymbol{\theta} \in \mathbb{IR}$, $\boldsymbol{\rho} \geq 0$, potom *(polárny) komplexný interval* definujeme ako množinu $\mathbf{z} \coloneqq \{\rho e^{i\theta} \in \mathbb{C} \mid \rho \in \boldsymbol{\rho}, \theta \in \boldsymbol{\theta}\}$.



Obr. 6.2: Polárny komplexný interval

Interval $\boldsymbol{\rho}$ nazývame *magnitúda* komplexného intervalu $\mathbf{z}$ a interval $\boldsymbol{\theta}$ je jeho *uhol*. Výhodou polárnej reprezentácie je uzavretosť na operácie násobenia a delenia, pri sčítaní a odčítaní už však dochádza k nadhodnoteniu výsledného obrazu.

Definíciu 6.4 môžeme upraviť, aby bola polárna reprezentácia komplexného intervalu jednoznačná, pridaním podmienok

$$\overline{\theta} - \underline{\theta} \leq 2\pi,$$
$$0 \leq \underline{\theta} < 2\pi,$$
$$0 \leq \overline{\theta} < 4\pi.$$

Súčet (rozdiel) polárnych komplexných intervalov nemôžeme zaviesť jednoducho ako množinu všetkých súčtov (rozdielov) prvkov jednotlivých intervalov, pretože takto vytvorená množina obecne nemá tvar požadovaný definíciou 6.4. Jednou z možností, ako definovať súčet a rozdiel, je použitie obdĺžnikových komplexných intervalov (Klatte – Ullrich, 1980). Pri prevodoch však potrebujeme vytvárať obaly množín, s ktorými pracujeme, a tak môže byť získaný výsledok iba veľmi hrubým odhadom popisovanej množiny.

Druhou možnosťou je definícia súčtu komplexných intervalov $\boldsymbol{x}, \boldsymbol{y}$ ako najmenšieho prvku z množiny polárnych komplexných intervalov, ktorý obsahuje množinu $\{x + y \mid x \in \boldsymbol{x}, y \in \boldsymbol{y}\}$, podobne pre rozdiel intervalov. Charakterizáciu tejto množiny bližšie popisujú Candau et al. (2006).

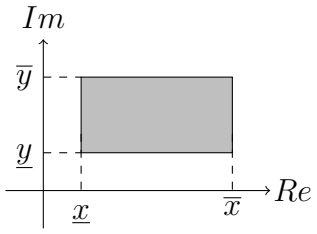
Operácie súčinu a podielu dvoch polárnych komplexných intervalov $\boldsymbol{x} = (\boldsymbol{\rho}_x, \boldsymbol{\theta}_x)$ a $\boldsymbol{y} = (\boldsymbol{\rho}_y, \boldsymbol{\theta}_y)$ zavedieme nasledovne:

$$\boldsymbol{x} \cdot \boldsymbol{y} \coloneqq \{x \cdot y \mid x \in \boldsymbol{x}, y \in \boldsymbol{y}\} = \left(\boldsymbol{\rho}_x \cdot \boldsymbol{\rho}_y, \boldsymbol{\theta}_x + \boldsymbol{\theta}_y\right),$$
$$\frac{\boldsymbol{x}}{\boldsymbol{y}} \coloneqq \left\{\frac{x}{y} \mid x \in \boldsymbol{x}, y \in \boldsymbol{y}\right\} = \left(\frac{\boldsymbol{\rho}_x}{\boldsymbol{\rho}_y}, \boldsymbol{\theta}_x - \boldsymbol{\theta}_y\right).$$

6.1.3 Obdĺžnikové komplexné intervaly

Tretia definícia komplexných intervalov pochádza z najbežnejšej reprezentácie komplexných čísel v tvare $a + bi$ alebo ekvivalentne ako usporiadanej dvojice reálnych čísel (a, b).

Definícia 6.5 (Boche (1966))**.** *(Obdĺžnikový) komplexný interval* $\boldsymbol{z}$ je usporiadaná dvojica reálnych intervalov $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{IR}^2$. Množinu všetkých komplexných intervalov budeme značiť symbolom $\mathbb{IC}$. Komplexný interval $([0, 0], [1, 1])$ označujeme $\mathbf{i}$. Interval $\boldsymbol{x}$ je *reálna časť* komplexného intervalu $\boldsymbol{z}$ (značenie $\boldsymbol{x} = Re(\boldsymbol{z})$) a $\boldsymbol{y}$ je jeho imaginárna časť (značenie $\boldsymbol{y} = Im(\boldsymbol{z})$).



Obr. 6.3: Obdĺžnikový komplexný interval

Zavedením komplexnej jednotky i pre komplexné intervaly môžeme interval $(\boldsymbol{x}, \boldsymbol{y})$ zapísať v tvare $(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} + \boldsymbol{y}i = \{x + yi \mid x \in \boldsymbol{x}, y \in \boldsymbol{y}\}$. Geometricky tvorí komplexný interval uzavretý obdĺžnik v komplexnej rovine rovnobežný s osami (alebo reálny intervalový box v $\mathbb{R}^2$).

Keďže komplexné intervaly definujeme v rovine, neexistuje na množine komplexných intervalov prirodzené lineárne usporiadanie. Môžeme však podobne ako pre reálne intervaly definovať rovnosť:

$$(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}', \boldsymbol{y}') \Leftrightarrow \boldsymbol{x} = \boldsymbol{x}' \wedge \boldsymbol{y} = \boldsymbol{y}'.$$

Súčet a rozdiel dvoch komplexných intervalov $\boldsymbol{x} = (\mathbf{a}, \mathbf{b}), \boldsymbol{y} = (\mathbf{c}, \mathbf{d})$ je definovaný nasledovne:

$$\boldsymbol{x} + \boldsymbol{y} = (\mathbf{a} + \mathbf{c}, \mathbf{b} + \mathbf{d}),$$
$$\boldsymbol{x} - \boldsymbol{y} = (\mathbf{a} - \mathbf{c}, \mathbf{b} - \mathbf{d}).$$

Operácie násobenia a delenia definujeme podobne ako na množine komplexných čísel (predpokladáme $0 \notin \mathbf{c}^2 + \mathbf{d}^2$):

$$\begin{aligned} \boldsymbol{x} \cdot \boldsymbol{y} &= (\mathbf{ac} - \mathbf{bd}, \mathbf{ad} + \mathbf{bc}), \\ \frac{\boldsymbol{x}}{\boldsymbol{y}} &= \left(\frac{\mathbf{ac} + \mathbf{bd}}{\mathbf{c}^2 + \mathbf{d}^2}, \frac{\mathbf{bc} - \mathbf{ad}}{\mathbf{c}^2 + \mathbf{d}^2} \right). \end{aligned}$$

Definícia 6.6. Buď $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{IC}$, potom jeho *komplexne združený interval* definujeme ako komplexný interval $(\boldsymbol{x}, -\boldsymbol{y})$ a *opačný interval* ako $(-\boldsymbol{x}, -\boldsymbol{y})$.

Geometricky predstavuje komplexne združený interval obdĺžnik, ktorý je k danému komplexnému intervalu symetrický podľa reálnej osi. Opačný interval je obraz intervalu stredovo symetrický podľa počiatku.

Na množine komplexných intervalov neplatia niektoré vlastnosti komplexne združených prvkov známe z komplexných čísel. Napríklad súčet komplexného intervalu a jeho komplexne združeného intervalu tvorí reálny interval iba v prípade, že je jeho imaginárnou časťou degenerovaný interval:

$$(\boldsymbol{x}, \boldsymbol{y}) + (\boldsymbol{x}, -\boldsymbol{y}) = (2\boldsymbol{x}, \boldsymbol{y} - \boldsymbol{y}).$$

Podobne pre súčin komplexných intervalov platí:

$$(\boldsymbol{x}, \boldsymbol{y}) \cdot (\boldsymbol{x}, -\boldsymbol{y}) = (\boldsymbol{x}\boldsymbol{x} + \boldsymbol{y}\boldsymbol{y}, -\boldsymbol{x}\boldsymbol{y} + \boldsymbol{y}\boldsymbol{x}) \neq (\boldsymbol{x}^2 + \boldsymbol{y}^2, 0).$$

6.2 Súvislosť s algoritmom SIVIA

Označme množinu $\{x \circ y \mid x \in \boldsymbol{x}, y \in \boldsymbol{y}\}$ ako presný obraz komplexných intervalov $\boldsymbol{x}, \boldsymbol{y}$ pri operácii $\circ$. Pri počítaní s obdĺžnikovými komplexnými intervalmi platí, že výsledok operácie súčtu alebo rozdielu dvoch intervalov je rovný skutočnému ich presnému obrazu. Táto vlastnosť je už však porušená pre operácie súčinu a podielu, keďže presný obraz týchto operácií obecne netvorí obdĺžnik.

Presný obraz súčinu dvoch komplexných intervalov $\boldsymbol{x}, \boldsymbol{y}$ môžeme popísať nasledovne:

$$\begin{aligned} &\{x \cdot y \mid x \in \boldsymbol{x}, y \in \boldsymbol{y}\} \\ =&\{(a + bi) \cdot (c + di) \mid (a + bi) \in \boldsymbol{x}, (c + di) \in \boldsymbol{y}\} \\ =&\{(ac - bd) + (ad + bc)i \mid (a + bi) \in \boldsymbol{x}, (c + di) \in \boldsymbol{y}\} \end{aligned}$$

Podobne môžeme odvodiť presný popis množiny všetkých podielov, ktoré vzniknú delením hodnôt z dvoch komplexných intervalov:

$$\begin{aligned} &\left\{ \frac{x}{y} \;\middle|\; x \in \boldsymbol{x}, y \in \boldsymbol{y} \right\} \\ =&\left\{ \frac{a + bi}{c + di} \;\middle|\; (a + bi) \in \boldsymbol{x}, (c + di) \in \boldsymbol{y} \right\} \\ =&\left\{ \frac{ac + bd}{c^2 + d^2} + \frac{bc - ad}{c^2 + d^2} i \;\middle|\; (a + bi) \in \boldsymbol{x}, (c + di) \in \boldsymbol{y} \right\} \end{aligned}$$

Túto množinu môžeme s ľubovoľnou presnosťou vykresliť pomocou algoritmu SIVIA prevodom získaného popisu množiny na problém splňovania obmedzujúcich podmienok. Zavedieme dve nové premenné e, f, ktoré budú reprezentovať reálnu a imaginárnu časť súčinu, resp. podielu. Pre tieto premenné nájdeme počiatočné definičné obory vynásobením komplexných intervalov podľa definície. Obmedzujúce podmienky dostaneme rozdeleným odvodeného popisu prvkov v množine na reálnu a imaginárnu časť, tieto položíme rovné premenným e a f.

Formulácia problému aproximácie množiny všetkých súčinov prvkov dvoch komplexných intervalov $\boldsymbol{x} = (\mathbf{a}, \mathbf{b}), \boldsymbol{y} = (\mathbf{c}, \mathbf{d})$ je nasledovná:

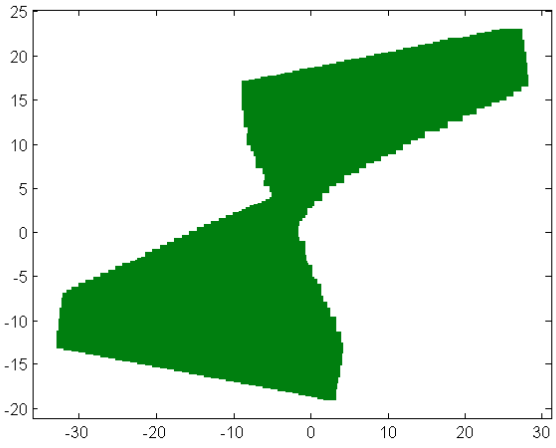
- premenné: a, b, c, d, e, f,
- definičné obory: $(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, Re(\boldsymbol{x} \cdot \boldsymbol{y}), Im(\boldsymbol{x} \cdot \boldsymbol{y}))$,
- obmedzujúce podmienky:

$$\begin{aligned} ac - bd &= e, \\ ad + bc &= f. \end{aligned}$$

Podobne môžeme odvodiť popis CSP pre množinu všetkých podielov:

- premenné: a, b, c, d, e, f,
- definičné obory: $(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, Re\left(\frac{\boldsymbol{x}}{\boldsymbol{y}}\right), Im\left(\frac{\boldsymbol{x}}{\boldsymbol{y}}\right))$,
- obmedzujúce podmienky:

$$\begin{aligned} \frac{ac + bd}{c^2 + d^2} &= e, \\ \frac{bc - ad}{c^2 + d^2} &= f. \end{aligned}$$



Obr. 6.4: Násobenie komplexných intervalov $([-1, 5], [2, 3]) \cdot ([-6, 6], [1, 1])$

7 Užívateľská dokumentácia

7.1 Inštalácia solveru

Pre používanie solveru stačí pridať adresár *cspsivia* do cesty MATLABu. Pridanie nových adresárov k ceste umožňuje funkcia `addpath`. Pre pridanie súborov na začiatok zoznamu ciest, v ktorých MATLAB hľadá používané triedy a funkcie, použijeme základné volanie v tvare

```
addpath('cesta1', ..., 'cestaN').
```

Táto funkcia tiež prijíma voliteľný parameter pre určenie pozície v zozname – ak za zvolené cesty doplníme reťazec '-end', umiestnia sa pridávané cesty na koniec zoznamu. Pre pridanie adresára spolu so všetkými jeho podadresármi je možné použiť variantu

```
addpath(genpath('cesta k adresáru')).
```

Adresár *cspsivia* obsahuje vygenerovaný MEX-súbor potrebný pre používanie dopredného a spätného kontraktoru pre Windows a MATLAB 2013a (32-bit). Pre využitie tohto kontraktoru na inej platforme, prípadne v inej verzii MATLABu, je nutné vygenerovať príslušný MEX-súbor (viď sekciu 7.5.2) využívajúci priloženú časť intervalovej knižnice Boost.

Pre správne fungovanie solveru je tiež potrebná sada nástrojov Intlab pre prácu s intervalmi a intervalovou aritmetikou v prostredí MATLABu, ktorú je možné získať na stránke `http://www.ti3.tuhh.de/rump/intlab/`.

7.2 Popis vstupov a výstupov

V tejto sekcii poskytneme stručný úvod k dátovým typom a funkciám MATLABu a Intlabu potrebným pre používanie vytvoreného solveru. Inštaláciu Intlabu a prácu s ďalšími funkciami bližšie popisuje napríklad Horáček (2011, kap. 11).

Pre predávanie definičných oborov premenných v CSP budeme používať jednu z troch funkcií určených k vytvoreniu intervalu:

- pomocou dolnej a hornej hranice: `infsup(`$\underline{x}, \overline{x}$`)`,
- pomocou stredu a polomeru: `midrad(`x^c, x^Δ`)`,
- ako degenerovaný interval: `intval(`x`)`.

V prípade viacerých premenných predávame vektor intervalov, ktorý vytvoríme pomocou hranatých zátvoriek, jednotlivé intervaly oddeľujeme medzerami, prípadne čiarkami:

```
[ interval1 interval2 ... intervalN ].
```

K jednotlivým prvkom vektoru môžeme pristupovať pomocou okrúhlych zátvoriek, i-tý prvok vektoru x označuje $x(i)$.

Obmedzujúce podmienky sa funkciám predávajú buď ako textové reťazce, ktoré sa v MATLABe zadávajú v apostrofoch alebo pomocou anonymných funkcií.

Anonymné funkcie sa uvádzajú zavináčom nasledovaným zoznamom premenných v prvej zátvorke a predpisom funkcie v druhej zátvorke. Jednotlivé premenné sú v zozname oddelené čiarkami:

```
@(x1, ..., xN)(predpis funkcie).
```

Niektoré funkcie požadujú na vstupe pole buniek obsahujúcich anonymné funkcie, textové reťazce apod. Pole buniek v MATLABe vytvoríme ako zoznam objektov oddelených čiarkami uzavretý v zložených zátvorkách:

```
{objekt1, ..., objektN}.
```

Prístup k jednotlivým bunkám umožňujú okrúhle zátvorky (napr. $x(i)$), pre prístup k dátam uloženým v bunkách použijeme zložené zátvorky (napr. $x\{i\}$).

7.2.1 Použitie funkcie `cspsivia`

Uvažujme úlohu nájdenia množiny všetkých bodov (x, y) v rovine, ktorých vzdialenosť od počiatku spadá do intervalu $[3, 4]$. Pre popísanie tejto úlohy ako problému splňovania obmedzujúcich podmienok potrebujeme určiť premenné, vzťahy medzi nimi a množiny hodnôt, ktoré budú môcť nadobúdať. Problém bude obsahovať dve *premenné* x a y, ktoré budú určovať súradnice bodu v rovine. Množina všetkých bodov danej vzdialenosti r od počiatku tvorí kružnicu, ktorú môžeme analyticky zapísať ako

$$x^2 + y^2 = r^2.$$

Všetky body, ktorých vzdialenosť od počiatku leží v intervale $[3, 4]$, teda môžeme popísať ako medzikružie dané dvomi nerovnicami:

$$x^2 + y^2 \geq 3^2,$$
$$x^2 + y^2 \leq 4^2.$$

Tieto nerovnice budú tvoriť *obmedzujúce podmienky* v našom probléme. Ďalej potrebujeme určiť *definičné obory* premenných. Vieme, že vzdialenosť každého bodu v hľadanej množine od počiatku bude nanajvýš 4, pri zahrnutí možných nepresností v aproximácii môžeme ako definičný obor oboch premenných zvoliť interval $[-4.5, 4.5]$.

Pre použitie algoritmu SIVIA (funkcia `cspsivia`) na vyriešenie tohto problému potrebujeme zvoliť niektoré ďalšie parametre. Jediným ďalším povinným parametrom je voľba presnosti výsledného riešenia. Túto hodnotu udáva parameter $\varepsilon > 0$, ktorý určuje najmenšiu možnú dĺžku strany boxu, ktorá sa bude ešte ďalej deliť.

Obmedzujúce podmienky predávame buď formou textových reťazcov (názvy premenných sú písmená anglickej abecedy):

```
>> f1 = 'x^2 + y^2 >= 9';
>> f2 = 'x^2 + y^2 <= 16';
```

alebo ako anonymné funkcie:

```
>> f1 = @(x,y)(x^2 + y^2 >= 9);
>> f2 = @(x,y)(x^2 + y^2 <= 16);
```

Ďalšou možnosťou je využitie testov inklúzie bližšie popísaných v sekcii 3.4.1. V tomto prípade je obmedzujúca podmienka vytvorená ako samostatná funkcia, ktorá sa predáva pomocou ukazovateľa. Testovacia funkcia prijíma na vstupe intervalový box, o ktorom chceme rozhodnúť a vracia hodnotu:

- 0, ak sa daný box neobsahuje žiadne riešenie,
- 1, ak je daný box podmnožinou množiny riešení,
- ľubovoľnú inú hodnotu, ak o boxe nedokáže rozhodnúť.

Definičné obory premenných zadávame ako intervalový vektor, v našom prípade

```
>> D = [infsup(-4.5, 4.5), infsup(-4.5, 4.5)];
```

Základná syntax pre použitie funkcie `cspsivia` na získanie aproximácie riešenia daného CSP s presnosťou 0.1 je

```
[S, N, B] = cspsivia({f1, f2}, D, 0.1);
```

Užívateľ má však tiež možnosť nastaviť ďalšie parametre upravujúce spôsob delenia boxov a použitý kontraktor. Prvý nepovinný parameter určuje voľbu kontraktoru z nasledujúcich možností:

- **none** (prednastavená hodnota) bez použitia kontraktoru,
- **fbprop** dopredný a spätný kontraktor,
- **boxnar** kontraktor BoxNarrow s metódou orezávania,
- **boxnarnewt** kontraktor BoxNarrow s Newtonovou metódou,
- **comb** kombinácia dopredného a spätného kontraktoru a BoxNarrow,
- **mohc** kontraktor Monotonic Hull Consistency.

Ďalším nepovinným parametrom je číselná hodnota určujúca, na koľko častí bude spracovávaný box rozdelený v každej iterácii algoritmu Sivia. V prípade, že hodnota nie je zadaná, bude box delený na polovice.

Nasledujúci nepovinný parameter určuje voľbu stratégie pre delenie strán. K dispozícii sú tri možnosti výberu:

- **lf** (longest first, prednastavené) delenie podľa najdlhšej strany,
- **sf** (shortest first) delenie podľa najkratšej strany,
- **rr** (round robin) cyklické striedanie strán v poradí od najnižšieho indexu.

Posledným nepovinným parametrom je logická hodnota, ktorá určuje, či sa má použiť algoritmus pre spájanie intervalových boxov (hodnota 1 znamená využitie algoritmu).

Poznámka. Vlastnosti MATLABu umožňujú predať pri volaní funkcie nepovinný parameter iba v prípade, že sú zadané všetky nepovinné parametre v poradí pred ním. Voľba stratégie delenia je preto možná iba so zadanou voľbou kontraktoru a počtom častí pre delenie, podobne pre voľbu počtu častí je nutné zadať voľbu kontraktoru. Pre použitie prednastavenej hodnoty je možné parameter ignorovať zadaním `[]`.

Výstupom funkcie `cspsivia` sú tri množiny intervalových boxov, v texte označované $\mathcal{S}, \mathcal{N}$ a $\mathcal{B}$. Prvá množina obsahuje boxy, ktorých zjednotenie tvorí vnútorný odhad skutočnej množiny riešení daného CSP. Zjednotením s množinou boxov $\mathcal{B}$ dostaneme vonkajší odhad skutočnej množiny riešení. Boxy obsiahnuté v množine $\mathcal{N}$ môžu pri použití kontraktorov obsahovať neplnodimenzionálne riešenie (aspoň jedna strana dĺžky 0), pri použití základnej verzie algoritmu SIVIA neobsahuje žiadne riešenie.

Ďalšie ukážkové príklady použitia funkcie `cspsivia` a ostatných funkcií je možné nájsť v súbore example.m, ktorý je súčasťou balíka.

7.2.2 Využívanie pomocných funkcií

`dividebox` Túto funkciu využíva algoritmus SIVIA na delenie intervalových boxov, ktoré nie je možné v danom kroku zaradiť do jednej z výsledných množín. Pre použitie mimo hlavného algoritmu stačí zadať intervalový box uložený v poli buniek, hodnotu ε, počet častí, na ktoré sa má box rozdeliť a voľbu strany pre delenie.

Hodnota ε určuje, aká má byť minimálna dĺžka strany, ktorá sa bude deliť. Ak sa má interval zvolenej strany rozdeliť v každom prípade, nastavíme $\varepsilon = 0$. Možnosti voľby strany pre delenie sú podobné, ako vo funkcii `cspsivia`, avšak stratégia round robin je nahradená delením podľa zadaného indexu strany.

Príklad delenia intervalového boxu $([-5, 5], [0, 1])$ podľa druhej súradnice na 5 častí bez ohľadu na šírku zvoleného intervalu:

```
dividebox({infsup(-5,5), infsup(0,1)}, 0, 5, 2);
```

`str2anon` Funkcia umožňuje vytvoriť numerické a booleovské anonymné funkcie zadané pomocou textových reťazcov. Za premenné sú považované reťazce tvorené písmenami anglickej abecedy, ktoré nie sú názvami podporovaných funkcií. Vo funkciách je možné používať základné aritmetické operácie a funkcie a relácie rovnosti a usporiadania v tvare `=`, `<=`, `>=`.

Príklad vytvorenia funkcie, ktorá vráti logickú hodnotu určujúcu, či je sínus zadaného argumentu rovný nule:

```
str2anon('sin(x) = 0');
```

`cspdb` Súbor cspdb.m obsahuje databázu niektorých známych typov kriviek a množín nimi popísaných. Tieto príklady môžu byť použité ako ukážka funkcionality balíku alebo pre vizualizáciu vlastností vybraných funkcií. Po spustení tohto súboru v MATLABe môžeme používať jednotlivé pomenované krivky. Každá krivka umožňuje použitie metódy `plotcurve`, ktorá pre zadanú presnosť a intervalový box vykreslí množinu ňou popísanú. Vlastné krivky je možné pridávať pomocou zoznamu obsahujúceho pole buniek s reťazcami popisujúcimi rovnice a nerovnice:

```
nazov = curve({podmienka1, ..., podmienkaM});
```

Príklad vizualizácie množiny v tvare nekonečna na boxe $[-5, 5] \times [-3, 3]$ s presnosťou $\varepsilon = 0.25$:

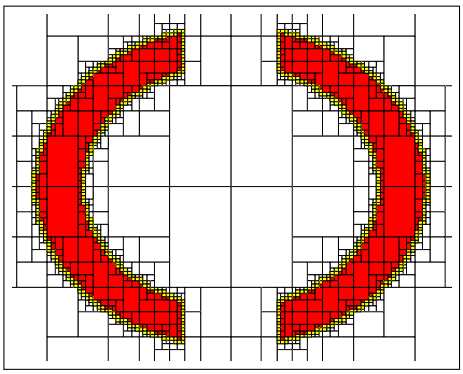
```
infty.plotcurve([infsup(-5,5), infsup(-3,3)], 0.25);
```

Dokumentáciu obsahujúcu popis ďalších funkcií a ich vstupov a výstupov je možné zobraziť priamo v MATLABe pomocou príkazu `help názov_funkcie` alebo príkazom `doc názov_funkcie` vo vstavanom prehliadači. Zoznam všetkých dostupných tried a funkcií so stručným popisom sa nachádza v prílohe A.

7.3 Projekcia a vizualizácia výsledkov

Balík obsahuje niekoľko funkcií umožňujúcich vizualizáciu získaných výsledkov priamo v prostredí MATLABu. Základným prostriedkom pre vizualizáciu dát je funkcia `plotboxes`, ktorá prijíma tri parametre:

- množinu dvojrozmerných intervalových boxov $\mathcal{S}$, ktorá reprezentuje vnútorný odhad množinu riešení zadaného problému,
- množinu dvojrozmerných intervalových boxov $\mathcal{N}$, v ktorej sa žiadne riešenie nenachádza (alebo sa v nej nachádzajú iba neplnodimenzionálne riešenia),
- množinu dvojrozmerných intervalových boxov $\mathcal{B}$, ktorá reprezentuje nepresnosť v aproximácii riešenia.



Obr. 7.1: Ukážka výstupu funkcie `plotboxes`

Funkcia ďalej prijíma nepovinný parameter, pomocou ktorého môže užívateľ nastaviť farby zobrazovaných boxov v jednotlivých množinách. Pri vynechaní parametru sa použije červená farba pre množinu $\mathcal{S}$, biela pre množinu $\mathcal{N}$ a žltá pre množinu $\mathcal{B}$. Zoznam zvolených farieb sa zadáva ako trojprvkové pole, teda vo formáte `plotboxes(S, N, B, {farba1, farba2, farba3})`. Posledným prijímaným parametrom je logická hodnota určujúca, či sa majú boxy vykresliť s ohraničením (hodnota 1) alebo bez ohraničenia.

MATLAB umožňuje tri spôsoby špecifikácie farieb – pomocou RGB hodnoty, krátkym názvom alebo dlhým názvom. RGB hodnota farby sa udáva trojzložkovým vektorom s hodnotami v intervale $[0, 1]$. Špecifikácia pomocou krátkeho alebo dlhého názvu je možná iba pre pomenované farby uvedené v tabuľke 7.1.

Pre vizualizáciu množiny riešení problémov s väčším počtom premenných je možné využiť funkcie pre projekciu do nižšej dimenzie. Projekciu do dimenzie 1 zabezpečuje funkcia `proj1d`. Táto funkcia prijíma 5 argumentov – tri množiny boxov ako v prípade funkcie `plotboxes`, index dimenzie pre výpočet projekcie a argument s logickou hodnotou 0 alebo 1, ktorá určuje, či sa má vygenerovať

Farba	Dlhý názov	Krátky názov	
azúrová	cyan	c	
purpurová	magenta	m	
žltá	yellow	y	
červená	red	r	
zelená	green	g	
modrá	blue	b	
čierna	black	k	
biela	white	w	

Tabuľka 7.1: Pomenované farby v MATLABe

aj grafická reprezentácia výsledku. Funkcia ďalej prijíma jeden voliteľný logický argument určujúci, či sa má grafická reprezentácia zobraziť miesto prednastavenej (vodorovnej) osi x na (zvislú) os y.

K projekcii do dimenzie 2 je určená funkciu `proj2d`. Rovnako ako `proj1d` prijíma táto funkcia tri množiny popisujúce riešenie problému. Ďalej je povinným parametrom dvojprvkový vektor indexov premenných, ktoré budú tvoriť výslednú projekciu a logický argument určujúci, či sa má výsledok tiež vizualizovať.

7.4 Komplexné intervaly

7.4.1 Kruhové komplexné intervaly

Sada nástrojov Intlab umožňuje prácu s kruhovými komplexnými intervalmi (viď sekciu 6.1.1). Rovnako ako pre reálne intervaly sú k dispozícii tri možnosti, ako zadať komplexný interval:

- keďže pracujeme s kruhovou reprezentáciou komplexných intervalov, je najčastejšie používanou možnosťou definovanie intervalu pomocou stredu a polomeru v tvare `midrad(`c`, `r`)`,
- ak chceme zadať degenerovaný komplexný interval, môžeme použiť syntax `intval(`x`)`,
- poslednou možnosťou je vytvorenie intervalu pomocou dolnej a hornej hranice, ktoré v tomto prípade určujú ľavý dolný a pravý horný roh obdĺžnika, príkazom `infsup(`$\underline{x}, \overline{x}$`)`. V tomto prípade sa však zadaný interval prevedie do kruhovej reprezentácie a bude vytvorený najmenší kruhový interval, ktorý zadaný obdĺžnik obsahuje.

Pre takto vytvorené intervaly je tiež definovaná komplexná intervalová aritmetika a ďalšie alternatívy reálnych intervalových funkcií, ktoré je možné používať rovnako ako pre reálne intervaly.

7.4.2 Obdĺžnikové komplexné intervaly

V rámci implementačnej časti tejto práce bola vytvorená trieda `rectcintval`, ktorá umožňuje ukladať objekty obdĺžnikových komplexných intervalov (viď sekciu 6.1.3). K vytvoreniu nového objektu komplexného intervalu je určený konšt-

ruktor tejto triedy s rovnakým názvom, ktorý prijíma dva argumenty – reálnu a imaginárnu časť komplexného intervalu:

```
rectcintval(infsup(0,1), infsup(0,1))
```

Pre takto vytvorené objekty komplexných intervalov sú dostupné základné aritmetické operácie (`+, -, *, /`) a testovanie na rovnosť (`=, ~=`).

Trieda tiež obsahuje ďalšie dva operátory `.*` a `./` využívajúce hlavnú funkciu `cspsivia` k získaniu tesnejšieho obrazu množiny násobkov alebo podielov dvoch komplexných intervalov. Tieto operátory vracajú pole s tromi množinami popisujúcimi výsledný obraz. Operátory zároveň zobrazujú grafickú reprezentáciu vypočítaného výsledku pomocou funkcie `plotboxes`.

7.5 Používanie MEX-súborov

7.5.1 Čo je MEX-súbor?

MEX je skratkou pojmu „MATLAB executable“, ktorý môže označovať:

- zdrojové MEX-súbory, teda súbory obsahujúce zdrojový kód jazyka C/C++ alebo Fortran s bránovou funkciou pre komunikáciu s MATLABom,
- moduly vzniknuté kompiláciou zdrojových súborov, tiež nazývané binárne MEX-súbory,
- hlavičkový súbor obsahujúci funkcie tvoriace rozhranie medzi MATLABom a zvoleným programovacím jazykom,
- skript, ktorý je súčasťou MATLABu a zabezpečuje (spolu s kompilátorom daného jazyka) o preklad zdrojového súboru.

V tomto texte budeme primárne používať označenie MEX-súbor pre zdrojové MEX-súbory jazyka C++. Tieto predstavujú jednoduchý spôsob ako vniesť objektový prístup a efektivitu C++ do skriptov napísaných v MATLABe. Po preložení takéhoto zdrojového súboru pomocou skriptu `mex` môže užívateľ volať vytvorenú funkciu priamo z prostredia MATLABu.

Okrem možnosti prekladu pomocou skriptu `mex`, ktorá je bližšie popísaná v sekcii 7.5.2, je tiež možné distribuovať už vygenerované súbory. Vytvorené súbory však musia pre správne fungovanie byť kompatibilné s používanou platformou a verziou MATLABu (viď sekcia 7.5.3).

7.5.2 Generovanie MEX-súborov

Zdrojové MEX-súbory obsahujú kód jazyka C/C++ alebo Fortran, preto je pre ich preklad potrebný podporovaný kompilátor daného jazyka. Zoznam kompilátorov podporovaných jednotlivými verziami MATLABu je k dispozícii na stránkach spoločnosti MathWorks (2014a). Kompilátor nastavíme pomocou príkazu `mex -setup` zadaného na príkazovom riadku MATLABu.

```
>> mex -setup

Welcome to mex -setup. This utility will help you set up a default compiler.
    For a list of supported compilers, see http://www.mathworks.com/support/
    compilers/R2013a/win32.html

Please choose your compiler for building \MEX-files:

Would you like mex to locate installed compilers [y]/n?

Select a compiler:
[1] Lcc-win32 C 2.4.1 in C:\PROGRA~2\MATLAB\R2013a\sys\lcc
[2] Microsoft Software Development Kit (SDK) 7.1 in C:\Program Files (x86)\
    Microsoft Visual Studio 10.0
[3] Microsoft Visual C++ 2012 in C:\Program Files (x86)\Microsoft Visual Studio
    11.0

[0] None
```

Obr. 7.2: Vzorový výstup nastavenia kompilátoru

Samotný preklad súboru sa spúšťa zadaním príkazu `mex názov_súboru` na príkazovom riadku MATLABu, pokiaľ sa súbor nachádza v aktuálnom pracovnom adresári, prípadne zadaním plnej cesty k súboru v príkaze. Po úspešnom preklade môžeme začať využívať vytvorenú funkciu ako bežnú funkciu MATLABu. Názov novej funkcie je rovnaký ako názov zdrojového súboru použitého pre jej vytvorenie, pokiaľ nie je pri preklade zvolený iný názov (viď tabuľku 7.2).

```
>> mex -I'C:\boost_library\' 'fbprop.cpp'
>> x = fbprop({'a >= 1'}, [infsup(-5.5, 5.5)])
>> intval x =
[    1.0000,    5.5000]
```

Obr. 7.3: Preklad a použitie MEX-súboru

Okrem jednoduchého príkazu pre preklad súboru je tiež možné využiť rôzne prepínače. Niektoré základné prepínače predstavuje tabuľka 7.2, kompletný zoznam prepínačov spolu s ich popisom sa nachádza v dokumentácii MATLABu (MathWorks, 2014b).

`-h`	zobrazenie nápovedy
`-I`*cesta*	zahrnutie dodatočných hlavičkových súborov
`-v`	zobrazenie detailných informácií o preklade
`-output` *názov*	zmena názvu výstupného súboru
`-outdir` *názov*	zmena výstupného adresára

Tabuľka 7.2: Prepínače skriptu `mex`

7.5.3 Použitie existujúcich MEX-súborov

Druhou možnosťou používania funkcií z MEX-súborov je priame využitie získaného binárneho MEX-súboru. Pred používaním súboru je potrebné sa uistiť, či je preložený súbor kompatibilný s platformou, na ktorej ho chceme spustiť. Platformu, na ktorej bol súbor vytvorený, určuje jeho prípona (viď tabuľku 7.3). V prípade potreby je možné zistiť príponu vhodnú pre aktuálnu platformu príkazom `mexext`.

Prípona	Platforma
mexw32	Windows (32-bit)
mexw64	Windows (64-bit)
mexa64	Linux (64-bit)
mexmaci64	Apple Mac (64-bit)

Tabuľka 7.3: Prípony MEX-súborov podľa platformy

Pre spustenie binárneho MEX-súboru je tiež nutné, aby boli k dispozícii všetky knižnice, ktoré program využíva. Binárne MEX-súbory sú väčšinou spätne kompatibilné, malo by byť preto možné spustiť súbor vytvorený pomocou staršej verzie MATLABu v novšej verzii, avšak nemusí to platiť naopak.

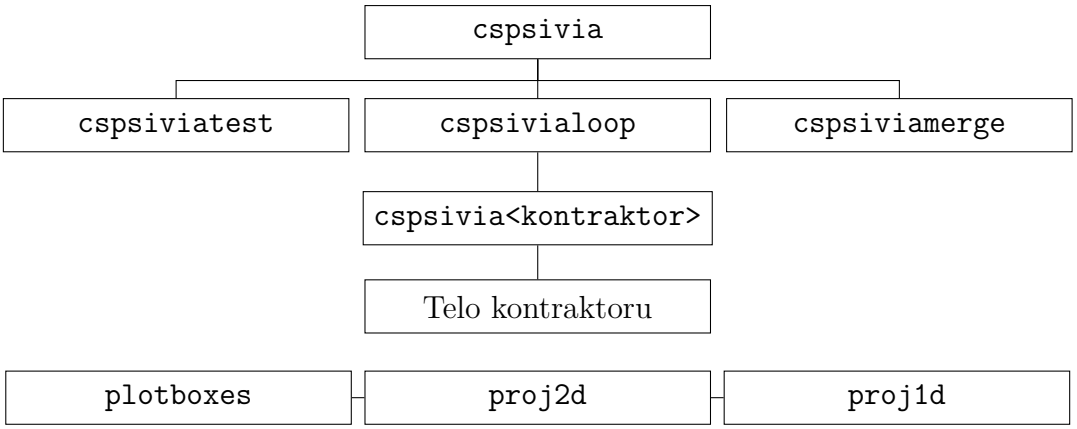
Pokiaľ je získaný súbor kompatibilný s platformou a verziou MATLABu a všetky potrebné knižnice sú dostupné, je možné jednoducho volať novú funkciu priamo z MATLABu pomocou názvu súboru (bez prípony) s príslušnými parametrami. Súbor spolu s potrebnými knižnicami musia byť uložené v adresári, ktorý je obsiahnutý v nastaveniach cesty MATLABu (funkcia `addpath`, prípadne voľba *File* > *SetPath*) alebo v aktuálnom pracovnom adresári.

8 Programátorská dokumentácia

8.1 Štruktúra programu

Implementovaný solver nelineárnych podmienok sa skladá z niekoľkých častí zabezpečujúcich spracovanie vstupu, samotný výpočet a zvýšenie jeho efektivity a prípadné ďalšie spracovanie výstupných dát. Súbory sú vytvorené prevažne v jazyku MATLAB, s výnimkou implementácie dopredného a spätného kontraktoru vo forme MEX-súboru v jazyku C++.

Základnou funkciou tvoriacou rozhranie pre spracovanie užívateľského vstupu je funkcia `cspsivia`. Táto funkcia sa stará o nastavenie nepovinných parametrov, získanie zoznamov premenných v zadaných obmedzeniach a vytvorenie objektov obmedzujúcich podmienok tak, aby mohli byť spracované ďalšími funkciami. Spracované obmedzujúce podmienky sú ukladané ako objekty triedy `constr`, ktoré okrem samotnej podmienky v podobe textového reťazca obsahujú jej normalizovaný tvar (s nulovou pravou stranou) ako anonymnú funkciu a použitú reláciu.



Obr. 8.1: Základná štruktúra implementovaného solveru

Telo hlavného algoritmu tvoria funkcie `cspsiviatest` a `cspsivialoop`. Prvá z nich sa použije v prípade, že užívateľ zadáva obmedzujúce podmienky pomocou testovacích funkcií (viď sekciu 3.4.1). V takomto prípade však nie je možné použiť implementované kontraktory. Funkciu `cspsivialoop` tvorí hlavný cyklus algoritmu SIVIA pre obmedzujúce podmienky predané ako textové reťazce alebo anonymné funkcie. Táto funkcia tiež zahŕňa nastavenie a volanie požadovaného kontraktoru. Ďalšou možnou volanou funkciou je `cspsiviamerge`, ktorá obsahuje algoritmus pre spájanie intervalových boxov vytvorených počas behu algoritmu SIVIA. Táto funkcia tiež znemožňuje použitie kontraktorov. Všetky verzie algoritmu využívajú pre delenie intervalových boxov funkciu `dividebox`.

Implementované kontraktory pracujú vždy iba s jednou obmedzujúcou podmienkou, ku každému kontraktoru je preto vytvorená funkcia zabezpečujúca voľbu jednotlivých podmienok (`cspsiviahull`, `cspsiviabox`, ...) a testovanie miery redukcie definičných oborov. Telá kontraktorov sú implementované v samostatných súboroch s ich príslušnými názvami: `fbprop` pre dopredný a spätný kontraktor, `boxnarrow{newton}` pre kontraktor BoxNarrow a `mohcrevise` pre kontraktor MoHC.

Balík ďalej obsahuje samostatné funkcie pre grafické spracovanie získaných dát. Vizualizáciu intervalových boxov popisujúcich riešenie daného CSP zabezpečuje funkcia `plotboxes`. Vytvorenie projekcie do jednej alebo dvoch dimenzií umožňuje dvojica funkcií `proj1d` a `proj2d`.

Súčasťou balíku je tiež trieda `rectcintval` umožňujúca základné aritmetické operácie nad obdĺžnikovými komplexnými číslami. Táto trieda využíva funkciu `cspsivia` pre výpočet presného násobenia a delenia a funkciu `plotboxes` pre vizualizáciu výsledku.

Zoznam všetkých implementovaných funkcií a tried s krátkym popisom je k dispozícii v prílohe A, bližší popis vstupov a výstupov obsahuje dokumentácia programu dostupná v prostredí MATLABu.

8.2 MEX-súbory

8.2.1 Štruktúra zdrojového MEX-súboru

V sekcii 7.5 sme sa venovali kompilovaniu MEX-súborov a ich používaniu. Táto kapitola poskytuje návod na vytvorenie zdrojového MEX-súboru v jazyku C++ a stručne popisuje niektoré potrebné knižnice. Pre tvorbu zdrojových MEX-súborov je tiež možné použiť jazyk Fortran.

Základnú štruktúru každého MEX-súboru tvorí bránová funkcia s pevne daným názvom `mexFunction` a parametrami `nlhs`, `plhs`, `nrhs` a `prhs`. Syntakticky je táto funkcia rovnaká ako klasické funkcie jazyka C++. Jej návratový typ je `void` – predávanie hodnôt do MATLABu zabezpečuje časť jej argumentov. Ďalej je nutné do programu zahrnúť knižnicu `mex.h`, ktorá obsahuje základné funkcie potrebné pre vytvorenie MEX-súboru.

```
1 #include "mex.h"
2
3 void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
4 {
5         // zdrojovy kod branovej funkcie
6 }
```

Obr. 8.2: Základná štruktúra MEX-súboru v C++

Význam jednotlivých parametrov bránovej funkcie je nasledovný (z angl. number, pointers, left-hand/right-hand side):

- **nlhs** počet výstupných parametrov funkcie
- **plhs** pole ukazovateľov na výstupné parametre
- **nrhs** počet vstupných parametrov funkcie
- **prhs** pole ukazovateľov na vstupné parametre

Aj pri nastavení premennej `nlhs` na 0 je možné vrátiť jeden výstupný parameter. Jeho hodnota bude priradená návratovej premennej `ans`.

Poznámka. Narozdiel od MATLABu využívajú MEX-funkcie indexovanie polí od 0 podľa konvencií jazyka C++.

Príklad. Pri volaní funkcie `f` v tvare `x = f(y, z)` je hodnota `nlhs` rovná 1, hodnota `nrhs` rovná 2, pole `prhs` obsahuje na indexoch 0 a 1 ukazovatele na premenné `y` a `z` a nastavením ukazovateľa na nultom mieste poľa `plhs` priradíme hodnotu premennej `x`.

V tele bránovej funkcie `mexFunction` je možné používať štandardné i vlastné funkcie jazyka C++, spracovávať vstupné parametre a nastavovať počet a hodnoty výstupných parametrov. V hlavičkovom súbore `mex.h` sa ďalej nachádzajú funkcie `mexCallMATLAB` a `mexEvalString` pre vykonávanie príkazov MATLABu. Volanie funkcií MATLABu z MEX-súborov vo svojej práci bližšie popisuje napríklad Getreuer (2010, kap. 6).

8.2.2 Knižnica `matrix.h` a dátový typ `mxArray`

Hlavičkový súbor `matrix.h` poskytuje ďalšie funkcie pre manipuláciu dát získaných z prostredia MATLABu. Všetky dáta, ktoré sú prenášané medzi MATLABom a funkciami jazyka C++ sú uložené v špeciálnom dátovom type `mxArray` spolu s príslušnými informáciami o ich pôvodnom type. Pre určenie skutočného typu dát, ktoré objekt `mxArray` obsahuje, je možné použiť funkcie knižnice `matrix.h` s prefixom `mxIs`.

Pre prístup k samotnému obsahu objektu je určená časť funkcií s prefixom `mxGet`, ďalšie umožňujú zistenie rozmerov alebo počtu prvkov daného poľa. Vytváranie vlastných objektov typu `mxArray`, napríklad pre návratové hodnoty premenných, obsluhujú funkcie s prefixom `mxCreate` a `mxSet`. K uvoľneniu dynamicky alokovanej pamäte slúži funkcia `mxDestroyArray`.

Po ukončení volania MEX-funkcie a návrate do MATLABu sa zavolá deštruktor pre každý objekt typu `mxArray`, ktorý nie je návratovým parametrom. Zároveň sa dealokuje pamäť, ktorá bola pridelená za behu programu pomocou príslušných funkcií knižnice `matrix.h`.

8.2.3 Ukážka MEX-súboru

V tejto sekcii nasleduje ukážka jednoduchého programu pre výpočet skalárneho súčinu dvoch vektorov. Príklad je doplnený vzorovým volaním vytvorenej funkcie z prostredia MATLABu. Ďalšie ukážky zdrojových súborov v jazykoch C a Fortran využívajúcich MEX-funkcie sú k dispozícii v priečinku `/extern/examples` v koreňovom adresári inštalácie MATLABu.

Každý MEX-súbor musí pre správne fungovanie obsahovať hlavičkový súbor `mex.h`, ktorý poskytuje rozhranie pre prepojenie programu s MATLABom. Ukážkový súbor ďalej obsahuje funkciu `dotProduct`, ktorá je klasickou funkciou jazyka C++ pre výpočet skalárneho súčinu dvoch vektorov.

```
1 #include "mex.h"
2
3 double dotProduct(double *v1, double *v2, int length)
4 {
5         double prod = 0;
6         for (int i = 0; i < length; i++)
7         prod += v1[i] * v2[i];
8         return prod;
9 }
```

Povinná je tiež bránová funkcia `mexFunction` starajúca sa o vstup a výstup. Pre výpis reťazca v okne MATLABu je použitá funkcia `mexPrintf`, ktorá je analógiou funkcie `printf` jazyka C, generovanie chybových hlášok obstaráva `mexErrMsgTxt`. Pripomeňme, že premenná `nrhs` obsahuje počet zadaných vstupných parametrov.

```
10 void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
11 {
12         mexPrintf("Program dotProduct\n");
13         double *vect1, *vect2;
14
15         if (nrhs != 2)
16                 mexErrMsgTxt("Nespravny pocet vstupnych parametrov!");
```

Nakoľko vytvorená funkcia `dotProduct` pracuje s ukazovateľom na vstavaný typ `double` jazyka C++, je potrebné získať ukazovatele na prvky poľa typu `mxArray` pomocou funkcie `mxGetPr`.

```
17         vect1 = mxGetPr(prhs[0]);
18         vect2 = mxGetPr(prhs[1]);
```

Na zistenie dĺžky vektora, teda počtu stĺpcov matice rádu $1 \times n$ slúži funkcia `mxGetN`, v prípade zisťovania počtu riadkov by sme použili obdobnú funkciu `mxGetM`. Ak sú dĺžky zadaných vektorov rôzne, vypíšeme chybovú hlášku.

```
19         if (mxGetN(prhs[0]) != mxGetN(prhs[1]))
20                 mexErrMsgTxt("Vektory maju roznu dlzku!");
```

Napokon zistíme požadovaný skalárny súčin a pomocou `mxCreateDoubleScalar` vytvoríme ukazovateľ na objekt typu `mxArray`, do ktorého výsledok uložíme a vložíme ho na nultý index návratového poľa `plhs`.

```
21         double result = dotProduct(vect1, vect2, mxGetN(prhs[0]));
22         plhs[0] = mxCreateDoubleScalar(result);
23 }
```

Takto vytvorený súbor môžeme skompilovať pomocou skriptu `mex` a použiť volaním priamo z príkazového riadku MATLABu s príslušnými vstupnými, prípadne výstupnými parametrami.

8.3 Použité intervalové knižnice

Pri tvorbe solveru nelineárnych podmienok boli použité dve knižnice podporujúce prácu s intervalmi a intervalovou aritmetikou. Základ programu vytvorený v jazyku MATLAB pracuje s nástrojovou sadou Intlab, MEX-súbor jazyka C++ využíva intervalovú časť knižnice Boost.

Intlab Sada nástrojov Intlab (Interval Laboratory) pre intervalové výpočty v MATLABe bola vytvorená v roku 1998 S. M. Rumpom. V súčasnosti okrem základnej reálnej intervalovej aritmetiky a intervalových rozšírení bežne používaných funkcií obsahuje tiež množstvo funkcií pre prácu s komplexnými intervalmi, intervalovými maticami, polynómami alebo pre optimalizáciu.

Pre riešenie problému splňovania obmedzujúcich podmienok pomocou intervalových metód sú významnou vlastnosťou tejto knižnice verifikované výpočty, ktoré zaručujú, že počas výpočtu nestratíme žiadne riešenie daného problému. Ďalším dôvodom pre voľbu tejto knižnice je jej jednoduchá použiteľnosť a široká užívateľská základňa.

Boost Sada Boost je súborom knižníc jazyka C++ obsahujúca balíky funkcií z rôznych oblastí. Časť Interval Arithmetic Library implementovaná v hlavičkovom súbore `interval.hpp` je určená pre prácu s intervalmi nad rôznymi dátovými typmi (napr. float, double). Podobne ako Intlab poskytuje široké spektrum intervalových rozšírení základných matematických funkcií a možnosť verifikovaných výpočtov. Ďalej obsahuje množinové funkcie, porovnávanie intervalov a ďalšie klasické intervalové funkcie. Táto knižnica tiež umožňuje nastavenie rôznych druhov zaokrúhľovania a pravidiel pre kontrolu prázdnych a nekonečných intervalov a neplatných hodnôt.

9 Záver

Hlavným výsledkom práce je knižnica funkcií jazyka MATLAB pre riešenie spojitých problémov splňovania nelineárnych podmienok pomocou intervalových metód a propagačných techník dostupná na priloženom CD. V rámci tvorby tejto knižnice bol tiež navrhnutý a implementovaný algoritmus pre spájanie intervalových boxov v dláždení reprezentujúcom riešenie problému, ktorý umožňuje optimalizáciu popisu riešenia. Súčasťou solveru je aj jednoduchá vizualizácia riešenia problému v rovine.

V teoretickej časti práce sme poskytli prehľad jednotlivých algoritmov spolu s ich vlastnosťami a modifikáciami. Prezentujeme tiež opravený odhad časovej zložitosti algoritmu SIVIA, ktorý je základom vytvoreného solveru. Pre zvýšenie efektivity tohto algoritmu boli použité 4 typy kontraktorov. Dva základné kontraktory boli porovnané na konkrétnych problémoch, ktoré v niektorých prípadoch ukazujú výrazné zrýchlenie výpočtu oproti algoritmu SIVIA. Uvažovali sme tiež o využití solveru CSP pre vizualizáciu základných aritmetických operácií na množine komplexných intervalov. Pre získanie samotnej vizualizácie je algoritmus SIVIA použiteľný, avšak kvôli časovej náročnosti tohto problému nie je v základnej verzii vhodný pre tvorbu interaktívnej aplikácie.

Práca prezentuje základný solver spojitých CSP s určitými vylepšeniami, zároveň však poskytuje ďalšie možnosti rozšírenia napríklad pridaním nových kontraktorov, návrhom algoritmov pre výber spracovávaných obmedzujúcich podmienok, návrhom pokročilejších funkcií pre grafickú reprezentáciu získaných intervalových dát alebo modifikáciami pre lepšiu použiteľnosť vo vizualizácii komplexnej intervalovej aritmetiky.

Zoznam použitej literatúry

Araya, I. *Exploiting Common Subexpressions and Monotonicity of Functions for Filtering Algorithms over Intervals*. PhD thesis, Université de Nice-Sophia Antipolis, Nice, France, March 2010.

Araya, I. – Neveu, B. – Trombettoni, G. An Interval Constraint Propagation Algorithm Exploiting Monotonicity. In *International workshop IntCP*, s. 65–83, 2009.

Benhamou, F. – McAllester, D. A. – Hentenryck, P. V. CLP(Intervals) Revisited. In Bruynooghe, M. (Ed.) *ILPS*, s. 124–138. MIT Press, 1994. ISBN 0-262-52191-1.

Benhamou, F. et al. Revising Hull and Box Consistency. In *International Conference on Logic Programming*, s. 230–244. MIT press, 1999.

Benhamou, F. – Granvilliers, L. Continuous and Interval Constraints. In Rossi, F. – Beek, P. – Walsh, T. (Ed.) *Handbook of Constraint Programming*, Foundations of Artificial Intelligence. Essex, UK: Elsevier Science Publishers Ltd., 2006. s. 571–603. ISBN 0-444-52726-5.

Boche, R. E. *Complex interval arithmetic with some applications*. Technical Report LMSC4-22-66-1, Lockheed Missiles and Space Co., Sunnyvale, CA, USA, 1966.

Candau, Y. et al. Complex Interval Arithmetic Using Polar Form. *Reliable Computing*. 2006, 12, 1, s. 1–20. ISSN 1385-3139.

Chabert, G. – Jaulin, L. Contractor Programming. *Artificial Intelligence*. July 2009, 173, 11, s. 1079–1100. ISSN 0004-3702.

Dijkstra, E. W. *Algol 60 translation: An Algol 60 translator for the X1 and Making a translator for Algol 60*. Technical Report 35, Mathematisch Centrum, Amsterdam, 1961.

Gaganov, A. Computation complexity of the range of a polynomial in several variables. *Cybernetics*. 1985, vol. 21, no. 4, s. 418–421. ISSN 0011-4235.

Gargantini, I. – Henrici, P. Circular arithmetic and the determination of polynomial zeros. *Numerische Mathematik*. 1971, 18, 4, s. 305–320. ISSN 0029-599X.

Getreuer, P. *Writing MATLAB C/MEX Code* [online]. April 2010. [cit. 24.03.2014]. Dostupné z: `http://www.getreuer.info/cmex.pdf`.

Goualard, F. – Jermann, C. A Reinforcement Learning Approach to Interval Constraint Propagation. *Constraints*. June 2008, 13, 1-2, s. 206–226. ISSN 1383-7133.

Granvilliers, L. Towards Cooperative Interval Narrowing. In Kirchner, H. – Ringeissen, C. (Ed.) *Frontiers of Combining Systems*, 1794 / *Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer, 2000. s. 18–31. ISBN 978-3-540-67281-4.

Granvilliers, L. – Benhamou, F. RealPaver: An Interval Solver using Constraint Satisfaction Techniques. *ACM Transactions on Mathematical Software*. 2006, 32, s. 138–156.

Hansen, E. – Walster, G. *Global Optimization Using Interval Analysis: Revised And Expanded*. Monographs and textbooks in pure and applied mathematics. Taylor & Francis, 2004. ISBN 0-8247-4059-9.

Horáček, J. *Přeurčené soustavy intervalových lineárních rovnic*. Diplomová práca, Univerzita Karlova v Praze, Praha, 2011.

Jaulin, L. Localization of an underwater robot using interval constraints propagation. In *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming, Nantes, France, September 2006 (CP 2006)*, Nantes, France, 2006.

Jaulin, L. et al. *Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer-Verlag, 2001. ISBN 1-85233-219-0.

Jaulin, L. – Walter, E. Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica*. 1993a, vol. 29, no. 4, s. 1053–1064. ISSN 0005-1098.

Jaulin, L. – Walter, E. Guaranteed nonlinear parameter estimation from bounded-error data via interval analysis. *Mathematics and Computers in Simulation*. 1993b, 35, 2, s. 123 – 137. ISSN 0378-4754.

Kearfott, R. B. – Novoa, M. I. Algorithm 681: INTBIS, a Portable Interval Newton/Bisection Package. *ACM Transactions on Mathematical Software*. June 1990, 16, 2, s. 152–157. ISSN 0098-3500.

Klatte, R. – Ullrich, C. Complex sector arithmetic. *Computing*. 1980, 24, 2-3, s. 139–148. ISSN 0010-485X.

MathWorks. *Supported and Compatible Compilers* [online]. 2014a. [cit. 24.03.2014]. Dostupné z: `http://www.mathworks.com/support/compilers/`.

MathWorks. *Build MEX-function from C/C++ or Fortran source code* [online]. 2014b. [cit. 24.03.2014]. Dostupné z: `http://www.mathworks.com/help/matlab/ref/mex.html`.

Merlet, J.-P. *An Algorithms Library of Interval Analysis for equation Systems* [online]. September 2004. [cit. 31.03.2014]. Dostupné z: `http://www-sop.inria.fr/coprin/logiciels/ALIAS/ALIAS.html`.

Merlet, J.-P. *The COPRIN examples page* [online]. December 2007. [cit. 20.05.2014]. Dostupné z: `http://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html`.

Moore, R. E. – Kearfott, R. B. – Cloud, M. J. *Introduction to Interval Analysis*. Society for Industrial and Applied Mathematics, 2009. ISBN 0-89871-669-1.

Ratschan, S. Efficient Solving of Quantified Inequality Constraints over the Real Numbers. *ACM Transactions on Computational Logic*. 2006, 7, 4, s. 723–748.

Rudin, W. *Principles of mathematical analysis*. McGraw-Hill Book Co., third edition, 1976. International Series in Pure and Applied Mathematics. ISBN 0-07-085613-3.

Trombettoni, G. et al. A Box-Consistency Contractor Based on Extremal Functions. In Cohen, D. (Ed.) *CP*, 6308 / *Lecture Notes in Computer Science*, s. 491–498. Springer, 2010. ISBN 978-3-642-15395-2.

Zoznam tabuliek

Zoznam algoritmov

Zoznam obrázkov

Zoznam použitých skratiek

CSP	Problém splňovania obmedzujúcich podmienok (angl. Constraint satisfaction problem)
Inm	Intervalová Newtonova metóda
HC	Konzistenia vzhľadom k obalu (angl. Hull consistency)
BC	Konzistencia vzhľadom k boxu (angl. Box consistency)
MoHC	Monotonic hull consistency
Sivia	algoritmus Set Inverter via Interval Analysis
Mex	skript alebo súbor typu Matlab Executable

A Zoznam tried a funkcií

`boxdiff` Výpočet rozdielu dvoch vnorených intervalových boxov. Funkcia prijíma dva intervalové boxy, druhý argument musí byť podmnožinou prvého. Rozdiel je reprezentovaný množinou nanajvýš $2d$ intervalových boxov (kde d je dimenzia pôvodných boxov). Táto funkcia je využívaná k získaniu boxov, v ktorých sa nenachádzajú žiadne riešenia CSP, pri redukcii definičných oborov pomocou kontraktorov.

`boxnarrow` Kontraktor BoxNarrow s metódou orezávania (viď sekciu 4.3.1). Funkcia využíva pomocnú funkciu dividebox pre delenie spracovávaného intervalu na časti na základe parametru ε. Redukcia definičného oboru prebieha na základe vyhodnotenia obrazu jednotlivých častí pomocou intervalovej aritmetiky.

`boxnarrownewton` Kontraktor BoxNarrow s využitím intervalovej Newtonovej metódy (viď sekciu 4.3.2). Obsahuje pomocné funkcie leftnarrow a rightnarrow pre redukciu intervalu a implementáciu intervalovej Newtonovej metódy jednej premennej.

`constr` Trieda pre ukladanie obmedzujúcich podmienok CSP. Pre každú podmienku je uložená jej reprezentácia pomocou textového reťazca a relácia rovnosti alebo nerovnosti v nej použitá. Objekt podmienky ďalej obsahuje ľavú stranu po prevode na normalizovaný tvar (pravá strana rovná 0) ako reťazec a anonymnú funkciu a interval, do ktorého má hodnota ľavej strany patriť.

`cspdb` Databáza niekoľkých množín popísaných pomocou známych kriviek reprezentovaných pomenovanými objektami triedy curve, vhodná napríklad pre ukážky vizualizácie množiny získanej pomocou algoritmu Sivia. Do databáze je možné manuálne pridávať ďalšie druhy kriviek.

`cspsivia` Základná funkcia solveru zabezpečujúca spracovanie vstupov od užívateľa a následné volanie funkcie `cspsivialoop` (základný algoritmus Sivia), `cspsiviamerge` (spájanie boxov) alebo `cspsiviatest` (testovacie funkcie) na základe zadaných parametrov. Funkcia vytvára objekty pre obmedzujúce podmienky v zadanom CSP a zoznamy premenných vyskytujúcich sa v jednotlivých podmienkach.

`cspsivia{hull|box|hullbox|mohc}` Pomocné funkcie zabezpečujúce voľbu obmedzujúcich podmienok pre spracovanie jednotlivými kontraktormi. Tieto funkcie tiež kontrolujú mieru kontrakcie definičných oborov po použití kontraktoru na danú podmienku a testujú prípadné zredukovanie boxu na prázdnu množinu, ak sa v ňom nenachádza žiadne riešenie úlohy.

`cspsivialoop` Hlavný cyklus základnej varianty algoritmu Sivia pre riešenie CSP (viď sekciu 3.3). Funkcia nastavuje a volá požadovaný kontraktor, testuje

splnenie obmedzujúcich podmienok a rozdeľuje boxy do výsledných množín popisujúcich riešenie zadaného CSP, prípadne volá pomocnú funkciu dividebox pre rozdelenie boxov, o ktorých nie je možné rozhodnúť.

`cspsiviamerge` Hlavný cyklus algoritmu SIVIA s úpravou pre čiastočné spájanie generovaných intervalových boxov (viď sekciu 3.6.2). Spájanie boxov je súčasťou samotného cyklu, boxy sa do výsledných množín pridávajú až po spojení. Pri použití algoritmu pre spájanie boxov nie je možné použitie kontraktorov, nakoľko takto vzniknuté dláždenie nie je pravidelné.

`cspsiviatest` Hlavný cyklus algoritmu SIVIA s podporou testovacích funkcií (viď sekciu 3.4.1). Vhodné pre obmedzujúce podmienky s komplexnejšími funkciami, pre ktoré nie je možné jednoducho odvodiť prirodzené intervalové rozšírenie. V tejto variante algoritmu nie je možné využívať kontraktory.

`curve` Jednoduchá trieda pre ukladanie kriviek a množín zadaných formou rovníc alebo nerovníc. Trieda obsahuje metódu pre vykreslenie objektu v intervalovom boxe pri zadanej presnosti pomocou algoritmu SIVIA.

`dividebox` Pomocná funkcia pre delenie intervalových boxov (viď sekciu 3.4.2). Prijíma parameter ε určujúci minimálnu šírku intervalu, ktorý je povolené deliť. Funkcia ďalej umožňuje voľbu strany, podľa ktorej bude box rozdelený a počet boxov získaných delením.

`fbprop` Dopredný a spätný kontraktor (viď sekciu 4.2) implementovaný ako MEX-súbor jazyka C++. Súčasťou kontraktoru je vytvorenie stromu algebraického výrazu pomocou algoritmu Shunting-Yard. Súbor ďalej obsahuje definície inverzných funkcií potrebných pre spätnú fázu kontraktoru.

`getmonovars` Pomocná funkcia kontraktoru MoHC pre výber monotónnych premenných na základe intervalového gradientu funkcie. Zároveň vytvára dve funkcie f_{min} a f_{max}, v ktorých sú výskyty monotónnych premenných vo funkcii f nahradené ich dolnými a hornými hranicami podľa toho, či je daná premenná rastúca alebo klesajúca.

`getvars` Funkcia pre získanie indexov premenných obsiahnutých v jednotlivých obmedzujúcich podmienkach. Súčasťou výstupu je aj zoznam premenných s unikátnym výskytom pre každú podmienku a zoznam názvov všetkých použitých premenných.

`intvalbox` Trieda pre ukladanie intervalových boxov s dodatočnými informáciami vzťahujúcimi sa k priebehu modifikovaného algoritmu SIVIA so spájaním boxov. Pre každý objekt je okrem jeho hodnoty možné uložiť zaradenie do výslednej množiny, počet delení potrebných pre jeho získanie (úroveň boxu v strome) a počet intervalových boxov, z ktorých bol vytvorený.

`isize` Pomocná funkcia pre výpočet objemu intervalového boxu. Objem boxu je definovaný ako súčin šírok jednotlivých intervalov. Vstup môže byť zadaný ako intervalový vektor alebo ako pole buniek obsahujúcich objekty intervalov.

`ivect` Pole s premenlivou dĺžkou pre ukladanie intervalových boxov. Pri naplnení kapacity sa pole automaticky rozšíri na dvojnásobnú veľkosť. Táto dátová štruktúra je využívaná pre rýchlejšie vkladanie prvkov do výsledných množín algoritmu Sivia popisujúcich riešenie CSP.

`mohcrevise` Hlavná časť kontraktoru MoHC (viď sekciu 4.5) využívajúca dopredný a spätný kontraktor a upravenú verziu kontraktoru BoxNarrow pre redukciu definičných oborov na základe monotónnosti funkcií. Súčasťou súboru je tiež pomocná funkcia pre získanie gradientu funkcie na intervalovom boxe a funkcia pre volanie dopredného a spätného kontraktoru s upravenou obmedzujúcou podmienkou (funkcie f_{min} a f_{max}).

`monoboxnarrow` Modifikácia kontraktoru BoxNarrow využívajúca monotónnosť funkcií v obmedzujúcich podmienkach. Funkcia využíva intervalovú Newtonovu metódu pre funkcie f_{min} a f_{max} pre výpočet redukovaného definičného oboru monotónnych premenných s viacnásobným výskytom v predpise obmedzujúcej podmienky. Súbor tiež obsahuje pomocné funkcie `leftnarrow` a `rightnarrow` pre získanie novej dolnej a hornej hranice definičného oboru.

`normeq` Jednoduchá funkcia umožňujúca prevod obmedzujúcej podmienky zadanej rovnicou alebo nerovnicou do tvaru $f(x) \circ 0$, kde $\circ \in \{\leq, \geq, =\}$. Funkcia je využívaná pri inicializácii algoritmu Sivia pre následné jednoduchšie spracovávanie obmedzujúcich podmienok a testovanie ich splnenia.

`plotboxes` Vizualizácia riešenia CSP získaného pomocou algoritmu Sivia. Funkcia prijíma tri množiny intervalových boxov dimenzie 2 popisujúce riešenie problému. Voliteľným parametrom je zoznam farieb použitých pre vykreslenie boxov v jednotlivých množínach.

`proj1d` Projekcia riešenia CSP do dimenzie 1 (viď sekciu 3.5). Prijíma tri množiny popisujúce riešenie CSP, index premennej, na ktorú sa má výsledok zobraziť a logickú hodnotu určujúcu, či sa má projekcia vizualizovať. Umožňuje tiež zobrazenie grafickej reprezentácie výsledku na zvislú os.

`proj2d` Projekcia riešenia CSP do dimenzie 2 (viď sekciu 3.5). Prijíma tri množiny popisujúce riešenie CSP, vektor obsahujúci dva indexy premenných, na ktoré sa má výsledok zobraziť a logickú hodnotu určujúcu, či sa má projekcia vizualizovať. Algoritmus využíva delenie dvojrozmerných boxov na pásy, v rámci ktorých využíva funkciu pre projekciu do dimenzie 1.

`rectcintval` Trieda implementujúca základné aritmetické operácie na množine obdĺžnikových komplexných intervalov (viď sekciu 6.1.3). Trieda využíva algoritmus Sivia pre nájdenie presnejšieho súčinu alebo podielu komplexných intervalov.

`spliteq` Pomocná funkcia pre rozdelenie obmedzujúcej podmienky na pravú a ľavú stranu a reláciu medzi nimi. Súčasťou výstupu je aj interval, do ktorého má patriť obraz funkcie pre podmienky v normalizovanom tvare $f(x) \circ 0$.

`stack` Zásobníková dátová štruktúra využitá pri implementácii algoritmu SIVIA. Trieda poskytuje základné metódy pre pridávanie a odoberanie prvku z vrcholu zásobníka, získanie zoznamu všetkých uložených prvkov alebo prvku na vrchole zásobníka. Implementovaná je tiež metóda pre zistenie, či je zásobník prázdny.

`str2anon` Prevod reťazca znakov na anonymnú funkciu. Reťazec môže obsahovať základné aritmetické operácie a funkcie podporované MATLABom. Zoznam premenných použitých v danom predpise je získavaný pomocou funkcie `symvar`.