TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Simmo Saan

Abstraktsete domeenide omaduspõhine testimine

Bakalaureusetöö (9 EAP)

Juhendaja: Vesal Vojdani, PhD
Juhendaja: Kalmer Apinis, PhD

Tartu 2018

Abstraktsete domeenide omaduspõhine testimine

Lühikokkuvõte:

One or two sentences providing a basic introduction to the field, comprehensible to a scientist in any discipline.

Two to three sentences of more detailed background, comprehensible to scientists in related disciplines.

One sentence clearly stating the general problem being addressed by this particular study.

One sentence summarising the main result (with the words “here we show´´ or their equivalent).

Two or three sentences explaining what the main result reveals in direct comparison to what was thought to be the case previously, or how the main result adds to previous knowledge.

One or two sentences to put the results into a more general context.

Two or three sentences to provide a broader perspective, readily comprehensible to a scientist in any discipline, may be included in the first paragraph if the editor considers that the accessibility of the paper is significantly enhanced by their inclusion.

Võtmesõnad:

List of keywords

CERCS:

CERCS kood ja nimetus: `https://www.etis.ee/Portal/Classifiers/Details/d3717f7b-bec8-4cd9-8ea4-c89cd56ca46e`

Property-based Testing of Abstract Domains

Abstract:

Inglisekeelne lühikokkuvõte

Keywords:

List of keywords

CERCS:

CERCS code and name: https://www.etis.ee/Portal/Classifiers/Details/d3717f7b-bec8-4cd9-8ea4-c89cd56ca46e

Sisukord

Unsolved issues

Sissejuhatus

What is it in simple terms (title)?

Why should anyone care?

What was my contribution?

What you are doing in each section (a sentence or two per section)

Tip: if it's hard for you to start writing, then try to split it to smaller parts, e.g. if the title is "Type Inference for a Cryptographic Protocol Prover Tool" then the "What is it" can be divided into "what is type inference", "what is cryptographic protocol" and "what is the prover tool". These three can also be split to smaller parts etc.

Valdkonna kirjeldus

Alamsektsioon sissejuhatusega kokku

Veel rohkem taustast, alustada kaugemalt: *static vs dynamic analysis*

Staatiline programmianalüüs on võimalikult automaatne protsess, mis programmi lähtekoodi põhjal järeldab midagi selle programmi käitumise kohta. Staatilist analüüsi teostatakse mitmel põhjusel. Esiteks, programmi optimeerimise eesmärgil teostatakse analüüsi kompilaatorites, leidmaks kohti programmikoodis, mida on automaatse muudatusega võimalik optimeerida, ilma et sellest muutuks programmi käitumine. Teiseks, programmist vigade leidmise eesmärgil, leidmaks vigu, ilma et oleks tarvis programm käivitada ja vigane olukord esile kutsuda. Kolmandaks, programmi korrektsuse näitamiseks, veendumaks, et programm kindlasti käitub oodatud veatul moel.

Kolmas punkt ei erine teisest arusaadavalt

Mainida alternatiivseid staatilise analüüsi meetodeid

Nimetada ja viidata olemasolevatele analüsaatoritele

Andmevooanalüüs (*data-flow analysis*) on üks intuitiivne meetod staatilise programmianalüüsi teostamiseks. Selle keskseks ideeks on võimalike programmi seisundite, sh sageli muutujate võimalike seisundite, määramine selle programmi punktides. Andmevooanalüüs kasutab programmi juhtimisvoograafi (*control-flow graph*), et järgida programmi seisundi muutumist selle võimalike töövoogude jooksul.

Üldiselt pole võimalik staatilise analüüsiga alati täpselt määrata programmi seisundit, sest see oleks samaväärne programmi käivitamisega. Seetõttu vaadeldakse ligikaudseid seisundeid, mis vastavad konkreetsetele seisunditele. Sellist ligikaudsete seisundite uurimist nimetatakse abstraktseks interpretatsiooniks ja see põhineb rangel teoreetilisel alusel, millel on ligikaudsusele vaatamata head omadused. Nimelt, abstraktse interpretatsiooni

teooria lubab, et analüüs on korrektne, st kui uuritavast programmist otsitavat tüüpi viga ei leita, siis võib olla kindel, et seda seal päriselt ka ei ole.

Abstraktse interpretatsiooni korrektsuseks on vajalik, et vaadeldavad ligikaudsed programmi seisundid, mis moodustavadki abstraktse domeeni, rahuldaks teatud algebralisi omadusi, mis võimaldavad andmevooanalüüsi teostada sobiva võrrandisüsteemi lahendamise teel. Seetõttu on hädavajalik, et staatilist analüüsi teostav programm, analüsaator, ise oleks implementeeritud korrektselt, sest vastasel juhul pole teostatavate analüüside tulemused usaldusväärsed ja korrektsed.

Omaduspõhine testimine on testimismeetod, mis on sobib hästi programmi loogika matemaatiliste omaduse kontrollimiseks. Selleks kirjeldatakse kontrollitavad omadused predikaatidena ja neile juhuslike argumentide genereerimise metoodika. Nende kombineerimisel genereeritakse soovitud kogus juhuslike argumentide komplekte, millel leitakse predikaatide väärtused, kinnitades omaduse kehtimist või kummutades selle. Lisaks toetab omaduspõhise testimise raamistik leitud vääravate testjuhtude lihtsustamist.

Siduda analüsaatori korrektsus omaduspõhise testimise kasutamisega

Konkreetselt Goblint-ist

1 Abstraktsed domeenid

Pikem üleminek sissejuhatusest domeenide juurde

Andmevooanalüüsiga püütakse võimalikult täpselt määrata programmi seisundit igas programmi punktis.

Definitsioon 1.1. Domeeniks nimetatakse programmi kõikvõimalike seisundite hulka [1].

Selline informaalne definitsioon on ebapiisav mingisuguse teooria arendamiseks, mistõttu tegelikult vaadetakse domeene, mis moodustavad täieliku võre.

1.1 Andmevooanalüüs

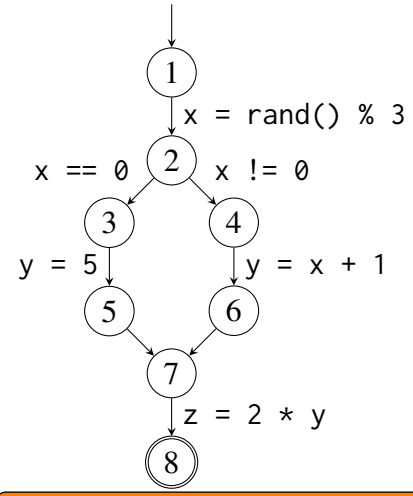
Andmevooanalüüsi ja abstraktse interpretatsiooni põhimõtete selgitamiseks parim viis on näite läbitegemine. Selleks olgu edaspidi vaatluse all programm jooniselt 1, kus on kõrvuti C-keelse lähtekoodi jupp ja selle juhtimisvoograaf, mille tippudes on programmi punktid, milles seisundeid uuritakse, ja servadel vastavad laused, mida täidetakse. Järgnevalt on käsitsi analüüsitud selle programmi täisarvuliste muutujate võimalike väärtusi igas programmi punktis:

- Punktis 1 pole muutujaid deklareeritud, seega nende väärtustest ei saa rääkida.
- Punktis 2 muutuja `x` väärtust üheselt määrata pole võimalik, sest see tuleb juhuarvu funktsioonist `rand()`, kuid jäägiga jagamise tulemusena kuulub see kindlasti hulka $\{0, 1, 2\}$[1]. Lisaks on vahepeal deklareeritud väärtustamata muutuja `y`, millel C keele semantikas vaikeväärtust pole ja seetõttu võib selle väärtus olla suvaline täisarv, st suvaline hulgast $\mathbb{Z}$ [2].
- Punktis 3, kus tingimus oli tõene, on `x` kindlasti ainult 0.
- Punktis 4, kus tingimus oli väär, saab eelnevast hulgast välistatud nulli eemaldamisel öelda, et muutuja `x` väärtus on hulgas $\{1, 2\}$.
- Punktis 5 on täpselt teada, et `y` on 5.
- Punktis 6, teades võimalikke muutuja `x` väärtusi, saab öelda, et muutuja `y` väärtus tehte tulemusena on hulgas $\{2, 3\}$.

[1] C keele semantika on siinkohal keerukas, kuid (üldiselt võimalikud) negatiivsed jäägid on antud juhul välistatud, sest `rand()` funktsioon ei tagasta negatiivseid arve.

```c
int x = rand() % 3;
int y;
if (x == 0)
    y = 5;
else
    y = x + 1;
int z = 2 * y;
```

kood eraldi faili?



tikz eraldi .tex faili?

(a) C-keelne lähtekood

(b) Juhtimisvoograaf

Joonis 1. Tsüklita programmi näidis.

- Punktis 7, kus kaks võimalikku tingimuslause haru uuesti kokku saavad, tuleb arvestada mõlema haru võimalike väärtustega, seega muutuja y väärtus kuulub hulka $\{2, 3, 5\}$.
- Punktis 8, teades võimalikke y väärtusi, saab öelda, et muutuja z väärtus tehte tulemusena on hulgas $\{4, 6, 10\}$.

Tehtud analüüs on võimalikult täpne, mis oleks ka automatiseeritud analüüsi eesmärgiks. Iseenesest poleks vale mõnes programmi punktis mõne muutujaga seostada suuremat võimalike väärtuste hulka, kuid sellisest analüüsist oleks vähem kasu, sest see tooks sisse ebavajalikku ebatäpsust. Järgnevalt ongi eesmärk matemaatiliselt formaliseerida sellise hea analüüsi teostamine, mis omakorda oleks aluseks analüüsi teoreetiliseks uurimiseks ja automatiseerimiseks.

1.2 Võred

Domeenide kirjeldamiseks kasutatakse matemaatilisi struktuure, mida nimetatakse võredeks. Kuigi võreteooria on arvestatav matemaatika haru, siis sügavamale laskumata on siin toodud vajalikud mõisted võredest aru saamiseks. Järgnevad eestikeelsed definitsioonid on refereeritud V. Laane loengukonspektist aines „Võreteooria“ [3], kuid tähistused on kohandatud programmianalüüsi kirjandusele omaseks [4:17]:

Definitsioon 1.2. Osaliselt järjestatud hulk on paar $(A, \sqsubseteq)$, kus A on hulk, millel on defineeritud binaarne seos $\sqsubseteq$, mis iga $a, b, c \in A$ korral rahuldab järgnevaid tingimusi:

- $a \sqsubseteq a$ (refleksiivsus),
- kui $a \sqsubseteq b$ ja $b \sqsubseteq c$, siis $a \sqsubseteq c$ (transitiivsus),
- kui $a \sqsubseteq b$ ja $b \sqsubseteq a$, siis $a = b$ (antisümmeetrilisus).

Domeenide puhul kasutatakse osalist järjestust seisundite täpsuse võrdlemiseks. Kokkuleppeliselt järjestatakse seisundid täpsemast ebatäpsema suunas — kirjutis $a \sqsubseteq b$ tähendab, et seisund a kirjeldab programmi olekut täpsemalt või sama täpselt kui seisund b. Teiste sõnadega, alati kui programmi olekut kirjeldab a, siis saab seda kirjeldada ka b-ga.
Järgnevalt olgu $(A, \sqsubseteq)$ osaliselt järjestatud hulk ja $X \subseteq A$.

Definitsioon 1.3. Elementi c nimetatakse hulga X **ülemiseks tõkkeks**, kui iga $x \in X$ korral $x \sqsubseteq c$. Vähimat ülemist tõket nimetatakse **ülemiseks rajaks**, st X-i iga ülemise tõkke d korral $c \sqsubseteq d$.

Definitsioon 1.4. Elementi c nimetatakse hulga X **alumiseks tõkkeks**, kui iga $x \in X$ korral $c \sqsubseteq x$. Suurimat alumist tõket nimetatakse **alumiseks rajaks**, st X-i iga alumise tõkke d korral $d \sqsubseteq c$.

Hulga X ülemist ja alumist raja tähistatakse vastavalt $\bigsqcup X$ ja $\bigsqcap X$. Kui $X = \{a, b\}$, siis tähistatakse ülemine ja alumine raja vastavalt $a \sqcup b$ ja $a \sqcap b$.

Domeenide puhul kasutatakse ülemisi tõkkeid mitme seisundi ühendamiseks, nii nagu seda oli vaja teha joonise 1 programmi punktis 7. Seejuures tahetakse loomulikult täpseimat ühendatud seisundit, mis ongi ühendatavate seisundite ülemine raja.

Milleks vaja alumist raja?

Definitsioon 1.5. Täielik võre on osaliselt järjestatud hulk, mille igal alamhulgal leidub ülemine ja alumine raja.

Täielikus võres $(A, \sqsubseteq)$ leidub vähim element $\bot = \bigsqcap A$ ja suurim element $\top = \bigsqcup A$, mida nimetatakse vastavalt *bottom*iks ja *top*iks.

Domeenidelt nõutaksegi, et need oleks täielikud võred, sest sellega kaasnevad ülal kirjeldatud võimalused seisunditega töötamiseks. Domeeni element $\top$ kirjeldab kõige üldisemat seisundit ehk seisundit, kus pole programmi oleku kohta mitte midagi teada. Element $\bot$ kirjeldab võimatut seisundit ehk tüüpiliselt seda, et programmi täitmisel mitte kunagi uuritavasse punkti ei jõutagi.

1.3 Alamhulkade domeen

Iga hulga S korral saab vaadelda selle kõigi alamhulkade hulka $\mathcal{P}(S)$, millel on loomulik osaline järjestus $\subseteq$. Osutub, et see on ka täielik võre, kus

- $a \sqsubseteq b \Leftrightarrow a \subseteq b$,

- $a \sqcup b = a \cup b$ ja $a \sqcap b = a \cap b$,
- $\bot = \varnothing$ ja $\top = S$.

Hulk X kirjeldab väärtust z parajasti siis, kui $z \in X$. Sellega on defineeritud, kuidas peaks seostama domeeni elemente konkreetsete võimalike väärtustega.

Description relation — kas üldse mõtet mainida? Vb jätta Galois ühenduste juurde.

Ülal käsitsi tehtud joonise 1 programmi analüüsis oligi iga muutuja väärtusi analüüsitud alamhulkade domeeni $\mathbb{D} = (\mathcal{P}(\mathbb{Z}), \subseteq)$ abil. Programmi punktis 7 toimunud muutuja y seisundite ühendamine on võrede terminites $\{5\} \sqcup \{2, 3\}$. Antud programmi analüüsimiseks on selle domeeniga vaja seostada kahte tüüpi tehted:

Konstantide abstraheerimine Lause `y = 5` abstraktseks teostamiseks, st muutujaga y domeeni elemendi seostamiseks, on esinev konstant vaja sobivalt abstraheerida. Täisarvude alamhulkade domeenis sobib konstantse väärtuse a jaoks ilmselt element $\{a\}$.

Abstraktne aritmeetika Lause `y = x + 1` abstraktseks teostamisks on vaja teostada liitmine muutuja x seisundi $\{1, 2\}$ ja konstandi seisundi $\{1\}$ vahel. Täisarvude alamhulkade domeenis saab elementide $A, B \in \mathcal{P}(\mathbb{Z})$ liitmise defineerida kui

$$A + B = \{a + b \mid a \in A, b \in B\}$$

ning sarnaselt võib defineerida ülejäänud tehted.

Nende matemaatiliste vahenditega ongi võimalik näites tehtu süstematiseerida. Siiski kirjeldab kasutatud domeen korraga ainult ühe muutuja väärtuseid, kuid muutujatevahelisi toiminguid otseselt mitte — seda peab ikkagi domeeniväliselt teostama. Oleks veelgi parem, kui tervet programmi olekut, st kõigi muutujate seisundeid korraga, saaks vaadelda ühe keerukama domeeni elementidena. Selleks võetaksegi kasutusele kujutused.

1.4 Kujutusdomeen

Olgu Var programmi muutujate hulk ja Val domeen, milles vaadeldakse üksiku muutuja seisundit. Sel juhul saab vaadelda abstraktsete muutujate väärtustuste domeeni [4:45]

$$\mathbb{D} = (\mathsf{Var} \to \mathsf{Val})_\bot = (\mathsf{Var} \to \mathsf{Val}) \cup \{\bot\}.$$

Domeeni elementideks on kujutused muutujate hulgast nende abstraktsete väärtuste hulka, mis on väga analoogilised konkreetsete muutujate väärtustustega funktsionaalselt kirjeldatuna. Element $\bot$ on tehislikult lisatud, et domeen moodustaks täieliku võre, ja tähendab saavutamatut (ingl. *unreachable*) programmi punkti. Osaline järjestus selles domeenis on defineeritud järgnevaga:

$$D_1 \sqsubseteq D_2 \iff D_1 = \bot \quad \vee \quad \forall x \in \mathsf{Var}\ D_1(x) \sqsubseteq D_2(x).$$

$\sqcup, \sqcap, \top$?

Joonisel 1 tehtud näites olid muutujad $\mathsf{Var} = \{\mathsf{x}, \mathsf{y}, \mathsf{z}\}$ ja väärtused domeenis $\mathsf{Val} = \mathcal{P}(\mathbb{Z})$. Sellises kujutuste domeenis saabki kirjeldada tervet seisundit korraga ning sama analüüsi lõpptulemus oleks järgmine:

- Punktis 1: $\{\mathsf{x} \mapsto \top, \mathsf{y} \mapsto \top, \mathsf{z} \mapsto \top\}$.
- Punktis 2: $\{\mathsf{x} \mapsto \{0, 1, 2\}, \mathsf{y} \mapsto \top, \mathsf{z} \mapsto \top\}$.
- Punktis 3: $\{\mathsf{x} \mapsto \{0\}, \mathsf{y} \mapsto \top, \mathsf{z} \mapsto \top\}$.
- Punktis 4: $\{\mathsf{x} \mapsto \{1, 2\}, \mathsf{y} \mapsto \top, \mathsf{z} \mapsto \top\}$.
- Punktis 5: $\{\mathsf{x} \mapsto \{0\}, \mathsf{y} \mapsto \{5\}, \mathsf{z} \mapsto \top\}$.
- Punktis 6: $\{\mathsf{x} \mapsto \{1, 2\}, \mathsf{y} \mapsto \{2, 3\}, \mathsf{z} \mapsto \top\}$.
- Punktis 7: $\{\mathsf{x} \mapsto \{0, 1, 2\}, \mathsf{y} \mapsto \{2, 3, 5\}, \mathsf{z} \mapsto \top\}$.
- Punktis 8: $\{\mathsf{x} \mapsto \{0, 1, 2\}, \mathsf{y} \mapsto \{2, 3, 5\}, \mathsf{z} \mapsto \{4, 6, 10\}\}$.

See loetelu kuidagi kompaktsemaks?

Kasutades kogu olekut kirjeldavaid domeeni elemente, on programmi laused ja avaldised juhtimisvoograafis kirjeldatavad funktsioonidena lähteseisundist sihtseisundisse. Neid nimetatakse **üleminekufunktsioonideks** (ingl. *transfer function*) ja samas näites võivad need olla järgnevad:

$$
\begin{aligned}
tf_{1,2}(d) &= d[\mathsf{x} \mapsto \{0, 1, 2\}], \\
tf_{2,3}(d) = d[\mathsf{x} \mapsto \{0\}], &\qquad tf_{2,4}(d) = d[\mathsf{x} \mapsto d(\mathsf{x}) \setminus \{0\}], \\
tf_{3,5}(d) = d[\mathsf{y} \mapsto \{5\}], &\qquad tf_{4,6}(d) = d[\mathsf{y} \mapsto d(\mathsf{x}) + \{1\}], \\
tf_{7,8}(d) &= d[\mathsf{z} \mapsto \{2\} * d(\mathsf{y})],
\end{aligned}
$$

kus kirjutis $d[\mathsf{x} \mapsto \{0\}]$ tähendab funktsiooni uuendamist (ingl. *function update*) ehk sama funktsiooni d, mis argumendil x omab nüüd uut väärtust $\{0\}$. Aritmeetilised tehted hulkade vahel on sellised nagu nad on sisemises domeenis Val.

Funktsiooni uuendamise def?

Abstraktse aritmeetika jaoks eraldi operaatorid: $\oplus, \odot, \otimes$?

Seejuures üleminekufunktsioonid saab keele semantika ning lausete ja avaldiste struktuuri mallide järgi mehaaniliselt defineerida ehk konkreetse programmi analüüsil toimub see automatiseeritult.

Kujutusdomeeni kasutamisega on võimalik lihtsam lihtsam ühte muutujat korraga kirjeldav domeen laiendada kõigile muutujatele korraga. Seejuures muutub võimalikuks järgida andmete liikumist muutujate vahel, mis vastab juba paremini andmevooanalüüsi nimele.

1.5 Algoritmiline analüüs

Võrede, nende abstraktsete tehete ja üleminekufunktsioonide abil on formaliseeritud suur osa esialgses näites intuitiivselt tehtust, kuid täielikult automatiseeritud analüüsini on jäänud veel viimane samm, mis võimaldaks kogu analüüsi algoritmiliselt kirjeldada. Kõigepealt peavad paigas olema kaks asja:

1. Domeen $\mathbb{D}$, mille abil analüüsi teostatakse ja mis on võimeline kirjeldama uuritavaid programmi omadusi.
2. Üleminekufunktsioonide defineerimise protsess programmeerimiskeele lausetest.

Nende olemasolul saab konkreetse programmi analüüsi teostada järgnevate sammudega:

1. Iga programmi punkti i jaoks võtta kasutusele üks muutuja x_i, mis vastab otsitavale programmi abstraktsele seisundile selles punktis.

2. Iga juhtimisvoograafi serva $k \to i$ jaoks defineerida fikseeritud protsessi abil üleminekufunktsioon $tf_{k,i}$. Nõnda saab iga serva jaoks moodustada võrratuse $x_i \sqsupseteq tf_{k,i}(x_k)$. See tähendab, et analüüsis otsitav seisund võib olla ebatäpsem kui konkreetne üleminek ette näeb.

   Lisaks üleminekutele koostatakse võrratus ka programmi alguspunkti, milleks olgu x_1, jaoks kujul $x_1 \sqsupseteq \imath$, kus $\imath$ kirjeldab algseisundit.

3. Neist võrratustest moodustub süsteem, kus iga muutuja on vasakul täpselt ühel korral, selleks vajadusel võrratusi parema poole ülemraja järgi ühendades:

$$\begin{cases} x_1 \sqsupseteq f_1(x_1, \ldots, x_n), \\ \vdots \\ x_n \sqsupseteq f_n(x_1, \ldots, x_n), \end{cases}$$

   kus $f_1, \ldots, f_n$ on moodustatud võrratuste parematest pooltest (üleminekufunktsioonidest ja nende ülemrajadest).

   Täpsemalt defineerida üleminekufunktsioonide ülemraja?

   Sama saab lühemalt kirja panna, kui vaadelda hoopis täielikku võre $\mathbb{D}^n$, mille elementide positsioonil i on programmi punkti i seisund. Selle võre tehted on defineeritud punktiviisiliselt.

Otsekorrutise domeeni eraldi defineerimine?

Sellises võres on muutujaks $\bar{x} = (x_1, \ldots, x_n)$ ning funktsioonid ühendatud kokku ühte funktsiooni $\bar{f}(\bar{x}) = (f_1(\bar{x}), \ldots, f_n(\bar{x}))$. Sellega on kogu eelnev võrratuste süsteem ühendatud üheks ainsaks võrratuseks $\bar{x} \sqsupseteq \bar{f}(\bar{x})$ [4:21].

Lisaks eeldatakse, et funktsioonid f_i ja seega kaudselt funktsioon $\bar{f}$ on monotoonsed, mis tähendab, et nad säilitavad rakendamisel täpsusega määratud järjestust [4:20].

Definitsioon 1.6. Olgu $\mathbb{D}_1$ ja $\mathbb{D}_2$ osaliselt järjestatud hulgad. Funktsiooni $f : \mathbb{D}_1 \to \mathbb{D}_2$ nimetatakse **monotoonseks**, kui iga $a, b \in \mathbb{D}_1$ korral

$$a \sqsubseteq b \quad \Longrightarrow \quad f(a) \sqsubseteq f(b) \text{ [5:23].}$$

4. Saadud võrratusele (ehk kaudselt algsele võrratuste süsteemile) tuleb leida vähim lahend. Selle iga lahend on üks korrektne analüüs, mis iga programmi punkti kohta sisaldab selle abstraktset seisundit. Vähim sobiv lahend on ühtlasi täpseim võimalik korrektne analüüs, ehk see, millest enim kasu oleks.

   Saab näidata, et see on samaväärne võrrandi $\bar{x} = \bar{f}(\bar{x})$ vähima lahendi leidmisega, mida nimetatakse ka vähima püsipunkti leidmiseks. Selleks on kõige naiivne iteratiivne algoritm, mis alustab väärtusest $\bar{x}_0 = \overline{\bot} = (\bot, \ldots, \bot)$ ja järjest iga $\bar{x}_i$ korral arvutab $\bar{x}_{i+1} = \bar{f}(\bar{x}_i)$ kuni lõpuks jõutakse nii kaugele, et mingi i korral $\bar{x}_i = \bar{x}_{i+1}$, st funktsiooni väärtused stabiliseeruvad ja funktsiooni edasi rakendamine tulemust ei muuda. Leitud lahendist ongi võimalik välja lugeda analüüsi tulemused iga programmi punkti jaoks [4:21].

Joonise 1 programmi analüüsimisel oleks võrratuste ja võrrandite süsteemid järgnevad:

$$\begin{cases} x_1 \sqsupseteq \top \\ x_2 \sqsupseteq tf_{1,2}(x_1) \\ x_3 \sqsupseteq tf_{2,3}(x_2) \\ x_4 \sqsupseteq tf_{2,4}(x_2) \\ x_5 \sqsupseteq tf_{3,5}(x_3) \\ x_6 \sqsupseteq tf_{4,6}(x_4) \\ \left.\begin{array}{l} x_7 \sqsupseteq x_5 \\ x_7 \sqsupseteq x_6 \end{array}\right\} \Rightarrow x_7 \sqsupseteq x_5 \sqcup x_6 \\ x_8 \sqsupseteq tf_{7,8}(x_7) \end{cases} \quad \text{ja} \quad \begin{cases} x_1 = \top \\ x_2 = tf_{1,2}(x_1) = x_1[\mathsf{x} \mapsto \{0,1,2\}] \\ x_3 = tf_{2,3}(x_2) = x_2[\mathsf{x} \mapsto \{0\}] \\ x_4 = tf_{2,4}(x_2) = x_2[\mathsf{x} \mapsto d(\mathsf{x}) \setminus \{0\}] \\ x_5 = tf_{3,5}(x_3) = x_3[\mathsf{y} \mapsto \{5\}] \\ x_6 = tf_{4,6}(x_4) = x_4[\mathsf{y} \mapsto d(\mathsf{x}) + \{1\}] \\ x_7 = x_5 \sqcup x_6 \\ x_8 = tf_{7,8}(x_7) = x_7[\mathsf{z} \mapsto \{2\} * d(\mathsf{y})] \end{cases}.$$

Nende iteratiivne lahendamine on samm-sammult toodud tabelis 1. On näha, et algoritmiline analüüs jõudis oodatud tulemuseni. Nagu näha, siis naiivne iteratsioon on üsna

ebaefektiivne viis vähima püsipunkti leidmiseks, kuna väärtuste stabiliseerumine võib võtta palju samme, kusjuures iga kord arvutatakse uuesti ka need väärtused, mis enam niikuinii ei muutu. Loomulikult on olemas mitmeid palju efektiivsemaid meetodeid, näiteks *round-robin* iteratsioon ja *worklist* algoritm [4:24,83].

Algoritmide nimesid kuidagi tõlkida?

Sellegipoolest pole vaadeldud analüüs kuigi praktiline, sest võimalike arvuliste väärtuste hulgad võivad olla suured või lausa lõpmatud. Üldiselt pole võimalik lõpmatuid hulki programmis efektiivselt esitada ja nendega opereerida. Seetõttu loobutakse ülimast täpsusest alamhulkade kujul ja kasutatakse ebatäpsemaid domeene väärtuste kirjeldamiseks, mida on efektiivselt võimalik analüsaatorisse implementeerida. Alamhulkade domeene saab siiski muudeks analüüsideks mõistlikult kasutada ning neil on oluline roll domeenide abstraheerimise uurimisel.

1.6 Intervalldomeen

Intervalldomeen on domeen, milles täisarvude väärtuste abstraheerimiseks kasutatakse arvtelje lõike. Selline abstraktsioon on arvuhulkadega võrreldes efektiivsem, olles samaaegselt praktikas ka piisavalt täpne.

Definitsioon 1.7. Intervalldomeeniks [4:55] nimetatakse hulka

$$\mathbb{I} = \{[l, u] \mid l \in \mathbb{Z} \cup \{-\infty\}, u \in \mathbb{Z} \cup \{+\infty\}, l \leq u\},$$

millel on osaline järjestus

$$[l_1, u_1] \sqsubseteq [l_2, u_2] \quad \Longleftrightarrow \quad l_2 \leq l_1 \wedge u_1 \leq u_2.$$

Ülejäänud domeenid ka definitsiooni kujul?

Sellises domeenis

$$[l_1, u_1] \sqcup [l_2, u_2] = [\min\{l_1, l_2\}, \max\{u_1, u_2\}],$$
$$[l_1, u_1] \sqcap [l_2, u_2] = [\max\{l_1, l_2\}, \min\{u_1, u_2\}], \quad \text{kui } \max\{l_1, l_2\} \leq \min\{u_1, u_2\}.$$

Joonis intervallide $\sqcup, \sqcap$ kohta *a la* Seidl

Intervallide järjestus on määratud nende omavahelise sisalduvuse kaudu ning ülem- ja alamraja on vastavalt lõikude ühend ja ühisosa. Lõigu otspunktides on lubatud vastavad lõpmatust kirjeldavad väärtused, mis võimaldavad rääkida selles domeenis suurimast elemendist $\top = [-\infty, +\infty]$. Kuna alumine raja on ainult tinglikult defineeritud, siis intervalldomeen ei moodusta täielikku võre. See pole probleem, kuna pisut täiendatud domeen $\mathbb{I}_\perp = \mathbb{I} \cup \{\perp\}$ osutub täielikuks võreks, kui seda vaja peaks olema.

Tabel 1. Näiteprogrammi (joonisel 1) analüüsi süsteemi iteratiivse lahendamise sammud ja lahend.

i	0	1	2			3			4			5			6			7
			x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	
x_1	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	
x_2	$\bot$	$\bot$	$\{0,1,2\}$	$\top$	$\top$	$\{0,1,2\}$	$\top$	$\top$	$\{0,1,2\}$	$\top$	$\top$	$\{0,1,2\}$	$\top$	$\top$	$\{0,1,2\}$	$\top$	$\top$	
x_3	$\bot$	$\bot$	$\bot$			$\{0\}$	$\top$	$\top$	$\{0\}$	$\top$	$\top$	$\{0\}$	$\top$	$\top$	$\{0\}$	$\top$	$\top$	
x_4	$\bot$	$\bot$	$\bot$			$\{1,2\}$	$\top$	$\top$	$\{1,2\}$	$\top$	$\top$	$\{1,2\}$	$\top$	$\top$	$\{1,2\}$	$\top$	$\top$	
x_5	$\bot$	$\bot$	$\bot$			$\bot$			$\{0\}$	$\{5\}$	$\top$	$\{0\}$	$\{5\}$	$\top$	$\{0\}$	$\{5\}$	$\top$	sama
x_6	$\bot$	$\bot$	$\bot$			$\bot$			$\{1,2\}$	$\{2,3\}$	$\top$	$\{1,2\}$	$\{2,3\}$	$\top$	$\{1,2\}$	$\{2,3\}$	$\top$	
x_7	$\bot$	$\bot$	$\bot$			$\bot$			$\bot$			$\{0,1,2\}$	$\{2,3,5\}$	$\top$	$\{0,1,2\}$	$\{2,3,5\}$	$\top$	
x_8	$\bot$	$\bot$	$\bot$			$\bot$			$\bot$			$\bot$			$\{0,1,2\}$	$\{2,3,5\}$	$\{4,6,10\}$	

Massiivne tabel: kuidagi väiksemaks? Lisadesse?

Lõik $[l, u]$ kirjeldab täisarvu z parajasti siis, kui $l \leq z \leq u$.

Analoogiliselt alamhulkade domeeniga, tuuakse siin programmide analüüsimiseks sisse:

Konstantide abstraheerimine Konstantsele täisarvulisele väärtusele a vastab intervall $[a, a]$.

Abstraktne aritmeetika Intervallidel saab samuti defineerida erinevad aritmeetilised tehted, moodustades intervallaritmeetika. Näiteks lõikude $[l_1, u_1], [l_2, u_2] \in \mathbb{I}$ korral:

$$[l_1, u_1] + [l_2, u_2] = [l_1 + l_2, u_1 + u_2],$$
$$[l_1, u_1] \cdot [l_2, u_2] = [\min\{l_1 l_2, l_1 u_2, u_1 l_2, u_1 u_2\}, \max\{l_1 l_2, l_1 u_2, u_1 l_2, u_1 u_2\}].$$

Siin on tehete defineerimine keerulisem kui alamhulkade domeenis, nagu korrutamisest paistab, kusjuures lisaks peab defineerima juhud, kus mõned otspunktidest on $-\infty$ või ∞.

Kas peaks seda ka siin tegema?

Valides eelnevas joonisel 1 tehtud näites seekord $\mathsf{Val} = \mathbb{I}$, saab taaskord kirjeldada üleminekufunktsioonid, kasutades seekord intervallide operatsioone. Tabelis 2 on toodud samasuguse algoritmilise analüüsi lõpptulemus, kui seda oleks tehtud intervalldomeeni abil. Nagu näha, siis intervallide kasutamine on ebatäpsem kui alamhulkade kasutamine, sest kaob informatsioon selle kohta, kui mõni väärtus lõigu keskelt tegelikult ei saa esineda. Samas on analüüs ikkagi korrektne, sest ühtegi päriselt võimalikku väärtust pole ekslikult välistatud.

Tabel 2. Näiteprogrammi (joonisel 1) analüüsi lahend intervalldomeenis.

	x	y	z
x_1	$\top$	$\top$	$\top$
x_2	$[0, 2]$	$\top$	$\top$
x_3	$[0, 0]$	$\top$	$\top$
x_4	$[1, 2]$	$\top$	$\top$
x_5	$[0, 0]$	$[5, 5]$	$\top$
x_6	$[1, 2]$	$[2, 3]$	$\top$
x_7	$[0, 2]$	$[2, 5]$	$\top$
x_8	$[0, 2]$	$[2, 5]$	$[4, 10]$

2 Domeeni omadused

2.1 Võre omadused

Olgu $\mathbb{D}$ täielik võre, siis iga $a, b, c \in \mathbb{D}$ korral peavad kehtima järgnevad tingimused:

Osalise järjestuse omadused (definitsioonist 1.2)

- $a \sqsubseteq a$ (refleksiivsus);
- kui $a \sqsubseteq b$ ja $b \sqsubseteq c$, siis $a \sqsubseteq c$ (transitiivsus);
- kui $a \sqsubseteq b$ ja $b \sqsubseteq a$, siis $a = b$ (antisümmeetrilisus);

Rajade omadused

- $a \sqsubseteq a \sqcup b$ ja $b \sqsubseteq a \sqcup b$ (definitsioonist 1.3);
- $a \sqcap b \sqsubseteq a$ ja $a \sqcap b \sqsubseteq b$ (definitsioonist 1.4);

Rajade tehete omadused [3:6, 5:39]

- $(a \sqcup b) \sqcup c = a \sqcup (b \sqcup c)$ ja $(a \sqcap b) \sqcap c = a \sqcap (b \sqcap c)$ (assotsiatiivsus);
- $a \sqcup b = b \sqcup a$ ja $a \sqcap b = b \sqcap a$ (kommutatiivsus);
- $a \sqcup a = a$ ja $a \sqcap a = a$ (idempotentsus);
- $a \sqcup (a \sqcap b) = a$ ja $a \sqcap (a \sqcup b) = a$ (neelduvus);

Vähima ja suurima elemendi omadused

- $\bot \sqsubseteq a$;
- $a \sqsubseteq \top$;
- $a \sqcup \bot = a$;
- $a \sqcap \top = a$;

Järjestuse ja rajade tehete seosed Järgnevad on ekvivalentsed [5:39]:

1. $a \sqsubseteq b$,
2. $a \sqcup b = b$,
3. $a \sqcap b = a$.

2.2 Laiendamine ja kitsendamine

widening, narrowing

Omadused

- $a \sqcup b \sqsubseteq a \sqcup b$ [4:61];
- $a \sqcap b \sqsubseteq a \sqcap b \sqsubseteq a$ [4:66].

2.3 Abstraktsiooni korrektsus

[6:242]

Galois ühendused vms.

Kokkuvõte

what did you do?

What are the results?

future work?

Viidatud kirjandus

[1] Vojdani V. Mitmelõimeliste C-programmide kraasimine analüsaatoriga Goblin. Magistritöö. TÜ arvutiteaduse instituut, 2006. http://dspace.ut.ee/handle/10062/1253 (27. 02. 2018).

[2] ISO/IEC. ISO International Standard ISO/IEC 9899:2011. Information technology – Programming languages – C. N1570 Commitee Draft. Apr. 12, 2011. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf (03/30/2018).

[3] Laan V. Võreteooria. Loengukonspekt. TÜ. Sügis 2017. https://courses.ms.ut.ee/MTMM.00.039/2017_fall/uploads/Main/kon.pdf (27. 02. 2018).

[4] Seidl H., Wilhelm R., and Hack S. Foundations and Intraprocedural Optimization. *Compiler Design: Analysis and Transformation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–114. https://doi.org/10.1007/978-3-642-17548-0_1 (02/27/2018).

[5] Davey B. A. and Priestley H. A. Introduction to Lattices and Order. 2nd ed. Cambridge: Cambridge University Press, 2002.

[6] Cousot P. and Cousot R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Los Angeles, California: ACM Press, New York, NY, 1977, pp. 238–252. http://www.di.ens.fr/~cousot/publications.www/CousotCousot-POPL-77-ACM-p238--252-1977.pdf (03/25/2018).

[7] Vojdani V. et al. Static race detection for device drivers: the Goblint approach. *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016*. ACM, 2016, pp. 391–402.

[8] Might M. Order theory for computer scientists. http://matt.might.net/articles/partial-orders/ (03/03/2018).

Eemalda nocite

Lisad

I Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Simmo Saan**,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

   Abstraktsete domeenide omaduspõhine testimine

   mille juhendajad on Vesal Vojdani ja Kalmer Apinis

   1.1 reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

   1.2 üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, pp.kk.aaaa

Kuupäev