Terminator

allan.reinhold@gmail.com

November 20, 2010

Kokkuvõte

Termineerumisanalüüs on osa programmianalüüsist ja on seotud peatumis probleemiga (ingl. Halting problem). Programmi termineerumisanalüüsiga üritame näidata, kas programmis toimub kindel termineerumine.

Terminator on raja- ja kontekstitundlik tarkvara analüsaator, tema ülesandeks on välja selgitada termineerumise argument, kasutades binaarulatuse analüüsi.

1 Sissejuhatus

Programmi termineerumine on keskse tähtsusega tagamaks süsteemi igakordset töötamist. Antud töös antakse ülevaade Terminatori nimelisest programmist, mida saab iseloomustada kui raja- ja kontekstitundlikku (ingl. path-sensitive and context-sensitive) tarkvara analüsaatorit. Terminator näitab läbilaskevõimet suuretel rakendustel, mille lähtekood võib olla 20000 rida või rohkem. Arvestatakse sisemisemisi tsükleid (ingl. nested loops), viitu (ingl. pointers), funktsiooni viitu (ingl. function pointers). Eriliseks teeb selle rakenduse omadus, et see suudab tööjaotust vajadusel korrigeerida kahe ülesande vahel – termineerumis argumendi konstrueerimise ja termineerumis argumendi kontrollimise vahel.

Reaktiivsüsteemid, näiteks operatsioonisüsteemid, mailiserverid, andmebaasimootorid, sellised süsteemid koosnevad reeglina paljudest komponentidest. Komponentide ülesanne on alati termineeruda. Juhud, mil termineeruvust ei toimu saab süsteeme nimetada, mittevastavateks süsteemideks ehk ei toimu see, mida on eelnevalt oodatud.

Klassikaliseks lähenemisteks antud probleemil on koostada avaldis, mis kirjeldaks teatud seisundi (ingl. state) taset (ingl. rank) ja iga võimaliku sammuga kahandada seda. Seisundi tasemefunktsiooni konstrueerimine on keeruline, kuna seda on vaja rakendada tervele programmile. Terminatori eeliseks on selle takistuse suhteliselt kerge lahendus. Selline etapp koostatakse läbivaatluse vajaduste alusel, kasutades selleks vaid mõnda rada terves programmis. Eesmärk pole konstrueerida ainult üks termineerumise argument vaid hulk oletatavaid võimalike termineeruvaid argumente. Võib esineda ka halbu oletusi. See ülemhulk peab teisisõnu olema hulk, milles esinevad oletuslikud lõplike avaldisteni jõudmise funktsioonid.

Termineerumisanalüüsiga üritame näidata, kas programmis toimub kindel termineerumine. Termineerumisanalüüs on osa programmianalüüsist ja on seotud peatumis probleemiga (ingl. Halting problem). Kuna peatumis probleem

oma olemuselt on lahendamatu, tõestatud aastal 1936. Alan Turingu poolt, saab väita, et termineerumisanalüüs ei saa alati anda loodetud tulemust (Toimub termineerumine/Ei toimu termineerumist). Turingu tõestusest saab siiski järeldada, et enamik programmidele saab termineerumisanalüüsi siiski tulemuslikult rakendada aga pole vahet kui keerulise tööriistaga katseid tehakse, leidub ikkagi vähemalt üks programm, mille termineerumist pole võimalik välja selgitada.

Käesolevas töös kirjeldame termineerumisanalüüsi üldiselt, uurime Terminatori nimelist programmi, binaarulatuseanalüüsi tööpõhimõtteid. Toome näiteid programmi terminerumisest. Töö aluseks on võetud suurel määral esimene viide, sellepärast puuduvad konkreetsed viited ka selle allika kohta.

2 Termineerumine

Programmi termineerumis probleemi (tuntud ka - ühtse peatumis probleemina, ingl. uniform halting problem), kasutades tänapäevast sõnastust saab seda defineerida järgnevalt. [2] Piiratud aja sees jõuda otsusele, kas programm lõpetab töö alati või potensiaalselt võib igavesti töötada.

Termineerumist on kõige mugavam defineerida, kui seostame seda protseduuri otseselt paariga, kus on programmi esialgsed konfiguratsioonid ning relatsioonid. Relatsioonid täpsustavad üleminekuid programmi täitmise ajal. Programmi täitmisest saab mõelda kui ühest programmi konfiguratsioonist läbi konfiguratsiooni muutumise, mis on lubatud programmi üleminekurelatsioonide poolt. Programm on termineeruv kui kõik ta täitmised on lõplikud. Toimub mitte termineerumine, kui leidub vähemalt üks täitmine, mis pole termineeruv. [2]

Kui tõestada programmi termineerumist, tuleb tõestada, et programmi ülemineku relatsioon on fundeeritud. [2]

3 Terminator

Terminatori raskeim osa on termineerumis argumendi kontrollimine, sest kuna see argument pole enam ainult üks tasemefunktsioon vaid terve hulk tasemefunktsioone. Üksiku tasemefunktsiooniga tuleb näidata, et tase väheneb eelseisundist järelseisundisse peale iga ülemineku sammu täitmist. Terminatori seadistustes pole piisav näidata ühte üleminekut, selle asemel tuleb arvestada kõigi üleminekutega, et igale üleminekul kahaneb üks tasemefunktsioon eelseisundi ja järelseisundi vahel. Tuleb esmalt leida kõik seisundi paarid s_1 ja s_2, et s_1 seisund oleks kättesaadav programmi algseisundist ja s_2 vastavalt s_1-st. Peale seda tuleb näidata, et ühe tasemefunktsiooni väärtus väheneb s_1-st s_2-le. Töös on seda ülesannet vastavalt nimetatud binaarulatuse analüüsiks.

Terminator kordab tsüklis kahte protseduuri: binaarulatuse analüüsi kontrolli, kasutades selleks termineerumisargumendi kandidaate; samal ajal kui tasemefunktsioonisünteesmootor konstrueerib kasvavalt uusi termineerumisargumente. Terminaatori aluseks olevat algoritmi võime vaadata jooniselt1.

Joonis1.
T := ∅; (* termineerumise argument, fundeeritud seoste ühend *)
repeat

(* T binaarse ulatuse kontroll *)
if $R_I^+ \subseteq$ T then
report "Termineerumine"
else
p := binaarseos, nii et $p \subseteq R_I^+$ ja $p \nsubseteq T$
(* tasemefunktsiooni leidmise koht p jaoks, kui see leidub *)
if p pole fundeeritud seos then
report "Mitte termineerumine"
else
(* Termineerumis argumendi konstrueerimine *)
W := tasemerelatsioon, st. $p \subseteq W$ ja W on fundeeritud
T := T $\cup$ W
end.

Binaarulatuse analüüs (ingl. binary reachablilty analysis) kontrollib kas R_I^+ kuulub hulka T. Kui kontroll nurjub, tagastatakse binaarne relatsioon, binaarset relatsiooni saab esitleda kui käskude hulka (tsüklis) ehk programmi. Järgnevad tasemefunktsiooni (ingl. ranking function) ja vastava tasemerelatsiooni (ingl. ranking relation) W konstrueerimise tehnikad.

3.1 Binaarulatuse analüüs

Binaarulatuse relatsiooni R_I^+ kui sulundit(ingl. closure) R keelatud seisunditest kuni ulatuvate seisunditeni, saab esitleda järgnevalt (kus R on programmi P ülemineku relatsioon ja I on algolekute hulgad):

$R_I^+ \triangleq \{s_1, s_2 | \exists s_0 \in I.(s_0, s_1 \in R^* \wedge (s_1, s_2) \in R^+\}$

R_I^+ koosneb seisundi paaridest (s_1,s_2) nii, et s_2 on kättesaadav seisundist s_1 vähemalt ühe sammuga ja s_1 ise on kättesaadav algolekust s_0.

Binaarulatuse analüüs on protseduur, mis kontrollib, kas R_I^+ sisaldub etteantud binaarrelatsioonis T: $R_I^+ \subseteq$ T. Juhul kui sisaldumist ei toimu, eksisteerib järjestus(ingl. *sequence*) programmi lauseid: $\tau_1 ... \tau_i ... \tau_n$ koos käivitamis järjestusega: s_1->τ_1 s_1... s_{i-1}->τ_i $s_i ... s_{(n-1)}$->$\tau_n s_n$ nii, et seisundi paar (s_i,s_n) ei sisaldu binaarrelatsioonis T: (s_i,s_n)$\in R_I^+$ ja (s_i,s_n)$\notin$T. Kui p on relatsioon, mis koosneb kõigist seisundi paaridest (s_i,s_n), mis on ühendatud ülal kireldatud käivitus järjestusega. Relatsioon p on kontranäidustus sisalduvusest (ingl. *inclusion*) $R_I^+ \subseteq$ T. Seega: $p \subseteq R_I^+$ ja p$\notin$T. Terminator rakendab taseme süntees meetodit, et proovida konstrueerida W tasemefunktsioon p-st. W koosneb seisundi paaridest koos kahaneva positiivse tasemega. Seega konstrueeritud tasemerelatsioon W sisaldab relatsiooni p ning on fundeeritud.

Terminator kasutab binaarulatust, et otsida võimalikke mitte fundeeritud radu ja üritab seda tõestada tasemefunktsiooni süntees meetodiga, et rajad on fundeeritud. Näiteks, kui leidub (a) tsüklit läbivaks rajaks $1- > 2- > 3- > 7$, saab seda esitada kas teise programmina (b) või relatsioonina(c) Q. Vaata joonist2.

Joonis2.
(a) Mitme rajaga tsükkel

```
1: do {
2:       if (z>x) {
3:            x++;
4:       } else {
```

```
5:              z++;
6:      }
7: } while (x<y);
```

(b) Tsükkel, esindamaks rada 1->2->3->7 (a)-s.

```
1: do {
2:      assume(z>x);
3:          x++;
7:      } while (x<y);
```

(c)Relatsioon Q esindab (a) tsükli läbimise rada 1->2->3->7. Avaldist (x_0, y_0, z_0) kasutatakse, et esindada
programmi seisundeid ennem tsüklit ja (x_1, y_1, z_1) esindada pärast tsüklit.
$Q((x_0, y_0, z_0), (x_1, y_1, z_1)) \triangleq z_0 > x_0 \wedge x_1 = x_0{+}1 \wedge y_1 = y_0 \wedge z_1 = y_1 \wedge x_1 < y_1$

Et tõestada järgneva programmi termineeruvust, tuleb kasvavalt konstrueerima relatsioon T, mis lõpuks peab sisaldama programmi binaarulatuse relatsiooni. Selleks tuleb konstrueerida alamrelatsioonid T_l koos seisundi paaridega samas programmis, kus leidub üks neist: 7, 12, 28, 33. Need asukohad moodustavad lõikepunktide hulga.

Joonis 3.

```
1 int Ack(int x, int y)
2 {
3       if (x>0) {
4           int n;
5           if (y>0) {
6               y−;
7               n = Ack(x,y);
8           } else {
9               n = 1;
10          }
11          x−;
12          return Ack(x,n);
13      } else {
14          return y+1;
15      }
16 }
17
18 void main()
19 {
20      int x = nondet();
21      int y = nondet();
22
23      int * p = &y;
24      int * q = &x;
25      bool b = true;
26
27      while(x<100 && 100<y && b)
28      {
```

```
29          if (p==q) {
30              int k = Ack(nondet(),nondet());
31              (*p)++;
32              while((k–)>100)
33              {
34                  if (nondet()) {p = &y;}
35                  if (nondet()) {p = &x;}
36                  if (!b) {k++;}
37              }
38          } else {
39              (*q)–;
40              (*p)–;
41              if (nondet()) {p = &y;}
42              if (nondet()) {p = &x;}
43          }
44      b = nondet();
45      }
46 }
```

Selgitus: nondet-ga näidatakse mitte deterministlikult valitud täisarve(ingl. Integer) ja tõeväärtusi(ingl.
Boolean).

iga l kohta koostatakse lõplik argument T^l kui T-de ühend. $T \triangleq T^7 \bigcup T^{12} \bigcup T^{28} \bigcup T^{33} \bigcup T^{DIFF}$
Kus T^{DIFF} on suur hulk triviaalselt fundeeritud relatsioone, asukohtadel näiteks:
$(s,t)|s(pc) = 3 \bigwedge t(pc) = 2, (s,t)|s(pc) = 3 \bigwedge t(pc) = 5$, jne.
Täpsemalt: $T^{DIFF} \triangleq \bigcup i \neq j (s,t)|s(pc) = j \bigwedge t(pc) = j$
Konstrueeritud relatsioonid T^l on loetletud järgnevas tabelis, vt. Joonist4.
Selgitus: Termineerumis argument (binaar relatsioonide fundeeritud ühend T^l_k lõikepunkti asukoha l seisundite vahel) on kasvavalt konstrueeritud ja seejärel kontrollitud. Tõendamaks temineerumist joonisel 1 toodud programmile.

Joonis4.

l	Fundeeritud binaarrelatsioonid T^l_k
7	$T^7_0(s,t) \triangleq false$
	$T^7_1(s,t) \triangleq t(y) > -1 \bigwedge t(y) \leq s(y) - 1$
12	$T^{12}_0(s,t) \triangleq false$
	$T^{12}_1(s,t) \triangleq t(x) > -1 \bigwedge t(x) \leq s(x) - 1$
28	$T^{28}_0(s,t) \triangleq false$
	$T^{28}_1(s,t) \triangleq t(y) > 100 \bigwedge t(y) \leq s(y) - 1$
	$T^{28}_2(s,t) \triangleq t(x) < 100 \bigwedge t(x) \geq s(x) + 1$
33	$T^{33}_0(s,t) \triangleq false$
	$T^{33}_1(s,t) \triangleq t(k) > 100 \bigwedge t(k) \leq s(k) - 1$

Tähele tasub panna, et $T^{28} = T^{28}_0 \bigcup T^{28}_1 \bigcup T^{28}_2$ on ühend fundeeritud relatsioonidest, mida aga ei saa tähendada ühendi enda kohta. Analüüs alustab relatsiooniga T^l_0, tühi relatsioon väärtustatud $false$ väärtusega ja lisab tasemeväärtused relatsioonidele T^l_1, T^l_2 jne. seni kuni pole enam rohkem kontranäiteid l jaoks, see tähendab, et T^l moodustatud nende ühendist sisaldab nüüd igat seisundi paari (s_i, s_n) asukohal l, nii et s_i on kättesaadav seisundist s_1 main

täitmisel ja s_n on kättesaadav seisundist s_i kasutades mittetühje täitmis sammude järjendit.

Binaarulatuse analüüsija on disainitud, et toetada programme koos viitade aliastega, nagu on näha Joonisel 3. Maksab tähele panna, et termineeruvus argument asukohal 28 (T^{28}), ei täpsusta relatsiooni x, y, p ja q vahel. See on eraldumis suhte näide. Binaarulatuse analüüsija jälgib $*p$ aliast, aga termineerumise argument ei täpsusta aliaste kasutamise suhteid. Termineeruvus argumendi konstruktsioonil pole vaja neid jälgida. Kui tõestada, et T^{33} sisaldab seisundi kõiki paare asukohal 33, binaarulatuse analüüsija peab tõestama, et $b == true$ on programm muutumatu asukohal 36. Seega hõlmab see programmi muutumatuse sünteesi, standard ulatuvuse analüüsi ülesannet.

Kui kommenteerida kood realt 11 Joonisel3, siis Terminator ei suuda tõestada programmi termineeruvust, annab tulemuseks järgneva programmi läbiva raja (tulemus antud koodiridade kaupa).

$Stem = 20- > 21- > 23- > 24- > 25- > 27- > 29- > 39- > 40- > 41- > 42- > 44- > 27- > 29- > 30- > 3- > 5- > 9- > 12$

$Cycle = 3- > 5- > 6- > 7- > 3- > 5- > 9- > 12$ Töös on nimetatud rada "Lassoks", mis sisaldub kahest osast, Stem (lasso vars) ja Cycle (lasso silmus).

4 Täielik näide.

Järgevalt näidatakse Terminaatorit kasutades järgneva programmi termineerumine. Programmis on üks tsükkel, seega otsitakse termineerumise argumenti programmi ühele asukohale (programmi asukoht 5) vt. Joonist 5.

Joonis5

```
1 void main()
2 {
3       int x=nondet(), y=nondet(), z=nondet();
4       if (y>0) {
5             do {
6                   if (nondet()) {
7                         x = x + y;
8                   } else {
9                         z = x - y;
10                  }
11            } while (x<y && y<z);
13      }
14 }
```

Esimene iteratsioon: Alustatakse termineerumise tõendamist andes termineerumise argumendile väärtuseks tühja argumendi, st. $T^5(s,t) = false$. Et kontrollida termineerumise argumendi T^5 õigsust, püüab binaarulatuse protseduur tõendada selle programmi töökindlust (ingl. $safety$), mida ta konstrueerib. Antud juhul nõrk binaarulatus pole piisavalt võimas, sest tuleb teada, et $y > 0$ tsükli analüüsi vältel. Sellepärast, asutakse kohe tugeva binaarulatuse juurde. Binaarulatuse analüüsi rakendamine genereeris uue programmi, mille järel sooritati analüüs hoopis sellele. Uus programm näeb välja selline vt. Joonist6.

Joonis6

```
0.1 int 'pc = 0;
0.2 int 'x, 'y, 'z;
1 void main()
2 {
3           int x=nondet(), y=nondet(), z=nondet();
4           if (y>0) {
5                 do {
5.1                     if ('pc==5) {
5.2                         if (!(false)) {
5.3                             ERROR: skip;
5.4                         }
5.5                     }
5.6                     if ('pc==0) {
5.7                         if (nondet()) {
5.8                             x = x;
5.9                             ' y = y;
5.10                             'z = z;
5.11                             'pc = 5;
5.12                         }
5.13                     }
6                        if (nondet()) {
7                             x = x + y;
8                         } else {
9                             z = x - y;
10                        }
11                } while (x<y && y<z);
13          }
14 }
```

Real 5.2 kasutatakse tingimust $T^5 = false$, ajutine töökindlus kontrollija leiab, et programmi asukoht '$ERROR$' (st. rida 5.3) on kättesaadav, ühe kontranäite olemasolul: $3- > 4- > 5- > 5.1- > 5.6- > 5.7- > 5.8- > 5.9- > 5.10- > 5.11- > 6- > 7- > 8- > 10- > 11- > 5- > 5.1- > 5.2- > 5.3$. Selle kontranäite saab jaotada vastavalt stem-ks ja cycle-ks. Jaotus tehakse ära jõudes reale 5.8,

mille tulemuseks jääb:

$stem = 3- > 4- > 5$

$cycle = 6- > 7- > 8- > 10- > 11- > 5$

Stem ja cycle esindavad järgnevaid matemaatilisi relatsioone R_{stem} ja R_{cycle}:

$R_{stem}((x_0, y_0, z_0), (x_1, y_1, z_1)) \triangleq y_1 > 0 \bigwedge x1 = x0 \bigwedge z1 = z0$

$R_{cycle}((x_0, y_0, z_0), (x_1, y_1, z_1)) \triangleq x_1 = x_0 + y_0 \bigwedge z_1 = z_0 \bigwedge y_1 = y_0 \bigwedge x_1 < y_1 \bigwedge y_1 < z_1$

Kontranäide, mis edastatakse tasemefunktsioonisünteesmootorile on järgmine:

$p_1(s_1, s_2) \triangleq \exists s_0 \in I.(s_0, s_1) \in Rstem \bigwedge (s_1, s_2) \in Rcycle$

Tasemefunktsioonisünteesmootor saab tõestada, et see relatsioon on fundeeritud ja tagastab järgneva tasemefunktsiooni:

$T_1^5(s, t) \triangleq t(x) < t(y) \bigwedge t(x) \geq s(x) + 1$

See tähendab, et T_1^5 on fundeeritud ja $p_1 \subseteq T_1^5$. Seejärel on T_1^5 lisatud termineerumise argumentidele lisaks:

$T^5(s,t) \triangleq T_1^5(s,t) \bigvee false$

Teine iteratsioon: taaskäivitatakse protseduur, kus üritatakse tõestada termineerumist asukohas 5 ja praegune termineerumise argument T^5 on võrdne $T^5(s,t) \triangleq T_1^5(s,t) \bigvee false$ ja kus tasemefunktsioon on järgnev: $T_1^5(s,t) \triangleq t(x) < t(y) \bigwedge t(x) \geq s(x) + 1$. Binaarulatuse protseduur toodab seejärel järgneva programmi, vt Joonist7.

Joonis7

```
0.1 int 'pc = 0;
0.2 int 'x, 'y, 'z;
1 void main()
2 {
3       int x=nondet(), y=nondet(), z=nondet();
4       if (y>0) {
5           do {
5.1             if ('pc==5) {
5.2                 if (!( (x<y && x>='x)
5.3                 || false
5.4                 ));
5.5             }
5.6             if ('pc==0) {
5.7                 if (nondet()) {
5.8                     'x = x;
5.9                     'y = y;
5.10                    'z = z;
5.11                    'pc = 5;
5.12                }
5.13            }
6               if (nondet()) {
7                   x = x + y;
8               } else {
9                   z = x - y;
10              }
11          } while (x<y && y<z);
13      }
14 }
```

Ajutine töökindlus kontrollija leiab, et '$ERROR$' on kättesaadav ja produtseerib järgneva kontranäite:

$stem = 3 -> 4 -> 5$

$cycle = 6 -> 8 -> 9 -> 10 -> 11 -> 5$

R_{stem}ja R_{cycle} vastavad relatsioonid:

$R_{stem}((x_0,y_0,z_0),(x_1,y_1,z_1)) \triangleq y_1 > 0 \bigwedge x_1 = x_0 \bigwedge z_1 = z_0$

$R_{cycle}((x_0,y_0,z_0),(x_1,y_1,z_1)) \triangleq x_1 = x_0 - y_0 \bigwedge x_1 = x_0 \bigwedge y_1 = y_0 \bigwedge x_1 < y_1 \bigwedge y_1 < z_1$

Teine kontranäite relatsioon p_2 defineeritakse järgnevalt:

$p_2(s_1,s_2) \triangleq \exists s_0 \in I.(s_0,s_1) \in Rstem \bigwedge (s_1,s_2) \in Rcycle$

$p_2$on ka fundeeritud – produtseeritud tasemerelatsioon on:
$T_2^5(s,t) \triangleq t(y) < t(z) \bigwedge t(z) \leq s(z)$–1
Termineerumise argumenti täiendatakse jällegi vastavalt:
$T^5(s,t) \triangleq T_2^5(s,t) \bigvee T_1^5(s,t) \bigvee false$
Kolmas iteratsioon: Nüüd ollakse valmis tõendama järgneva termineerumis argumendi õigsust:
$T^5(s,t) \triangleq T_2^5(s,t) \bigvee T_1^5(s,t) \bigvee false$
Binaarulatuse protseduur seejärel produtseerib järgneva programmi, vt Joonist8.

Joonis8

```
0.1 int 'pc = 0;
0.2 int 'x, 'y, 'z;
1 void main()
2 {
3       int x=nondet(), y=nondet(), z=nondet();
4       if (y>0) {
5           do {
5.1             if ('pc==5) {
5.2                 if (!( (y<z && z<='z)
5.3                 || (x<y && x>='x)
5.4                 || false
5.5                 ));
5.6             }
5.7             if ('pc==0) {
5.8                 if (nondet()) {
5.9                     'x = x;
5.10                    'y = y;
5.11                    'z = z;
5.12                    'pc = 5;
5.13                }
5.14            }
6               if (nondet()) {
7                   x = x + y;
8               } else {
9               z = x - y;
10              }
11          } while (x<y && y<z);
13      }
14 }
```

Asukoht '$ERROR$' pole kättesaadav selles programmis ja seega lõplik termineerumise argument on programmi asukohal 5 on leitud:
$T^5(s,t) \triangleq T_2^5(s,t) \bigvee T_1^5(s,t) \bigvee false$
Kus:
$T_1^5(s,t) \triangleq t(x) < t(y) \bigvee t(x) \geq s(x) + 1$
$T_2^5(s,t) \triangleq t(y) < t(z) \bigvee t(z) \leq s(z)$–1
Kuna programmis on vaid üks lõikepunkt võib lõplikuks termineerumise argumendiks lugelda järgnevat:
$R_I^+ \subseteq T_2^5 \bigcup T_1^5(s,t) \bigcup TDIFF$

5 Terminatori projektist

Terminatori projektiga tehti algust aastal 2004 ning projektis keskendutakse arendamaks automaatseid meetodeid tõendamaks programmi termineerumist ja üldiseid elulikkuse (ingl. liveness) väärtuseid. Praegune versioon Terminatorist on raja- ja kontekstitundlik tõendaja, mis toetab samaaegselt programmi sisemisi tsükkleid, rekursiooni, viitu(ja kõrvalmõjusid), funktsiooni viitu, jne. Terminatorit on väga edukalt rakendatud Windowsi seadme draiveritel, mis sisaldavad kuni 35000 rida koodi. Ilmnesid mõningad juhtumid, kus draiverite moodulid ei termineerunud enam, ehk need vead olid ka teada draiverite arendajatele aga sellest pikemalt võib lugeda esimesest viitest. [3]

Peamine eesmärk Terminatori projektis on arendada automaatsed tehnikad ja tööriistad (tööriistad rakendavad tehnikaid), et kasutada neid tööstuslikus tarkvaras ning näidata, et tarkvaras ei toimuks hangumist. [3]

6 Kokkuvõte

Termineerumisanalüüs on osa programmianalüüsist ja on seotud peatumis probleemiga (ingl. Halting problem), mida 1936 aastal tõestas Alan Turing. Termineerumis analüüsi ülesandeks on näidata, kas etteantud programm termineerub. Turingu tõestusest saab siiski järeldada, et enamik programmidele saab termineerumisanalüüsi siiski tulemuslikult rakendada aga pole vahet kui keerulise tööriistaga katseid tehakse, leidub ikkagi vähemalt üks programm, mille termineerumist pole võimalik välja selgitada. Töös on tutvustatud lühidalt Terminatori teadus projekti ning metoodikaid. Terminatorit ja binaarulatuse analüüsi saab iseloomustada järgnevalt: Skalaarsus: Binaarulatuse analüüs rakendab täiustatud abstraktset kontranäite vormi, mis suudab käidelda igat binaarulatuse päringut. Rakendatavus: Terminator toetab enamus keele tunnusjooni: tsüklid, aliased, viidad, funktsiooniviidad, jne. See on võimalik tänu binaarulatusele, mis käitleb selliseid detaile iseseisvalt. Automaatsus: Terminator on täielikult automaatiseeritud. Ta ei vaja kasutaja poolset lisainfot (tasemefunktsioone, tõendamis vihjeid), kuna selle suudab produtseerida kontranäite täiustus mehhanism, mis võimendab binaarulatust. Täpsus: Terminator rakendab täpset raja- ja kontekstitundlikku programmianalüüsi. Kontranäite genereerimine: Terminator annab kontranäiteid nurjunud termineerumis tõendusele, see on jällegi tänu binaarulatuse analüüsile.

Terminator saavutab edu nihutades koormust termineerumis argumendi konstrueerimiselt selle kontrollimisele. Terminator konstrueerib termineerumis argumenti, mis on disjunktsioon fundeeritud relatsioonidest. Iga selline relatsioon on leitud vajadustele, mis on tekkinud programmi ühekordsest ja kiirest läbimisest. Keerukaim osa on täita binaarulatuse analüüsil, kus tuleb kontrollida, et argumentide disjunktsioon kataks täielikult programmi läbimis seisundid.

7 Viited

1. Byron Cook , Andreas Podelski ja Andrey Rybalchenko. Termination Proofs for Systems Code *

2. Byron Cook. Principles of the program termination.

3. Terminator Research Project.

http://research.microsoft.com/en-us/um/cambridge/projects/terminator/default.htm