1 Sissejuhatus

1.1 Programmianalüüs

Programmianalüüs annab staatilised kompileerimis-aegsed tehnikad mille abil saab ennustada turvalisi ning arvutatavaid lähenemisi väärtustele või käitumistele mis võivad esile kerkida dünaamiliselt programmi käimise ajal. Peamiseks eesmärgiks on võimaldada kompilaatoritel genereerida koodi mis väldiks liigset arvutamist - olemasolevate tulemuste taaskasutamine, korduvate arvutuste väljaviimine tsüklitest, ning arvutuste vältimine mille tulemust hiljem vaja ei lähe.

Ligikaudsed vastused.
read(x); (if x>0 then y:=1 else (y:=2;S)); z:=y
Intuitiivselt: väärtused saavad olla 1 või 2

1.2 While keel

Lihtne, imperatiivne keel mis koosneb lausest.

Syntactic categories
$a \in$ **AExp** aritmeetilised avaldised
$b \in$ **BExp** tõeväärtus avaldised
$S \in$ **Stmt** laused

$x, y \in$ **Var** muutujad
$n \in$ **Num** numbrid
$\ell \in$ **Lab** märgendid (*labels*)

$op_a \in$ $\mathbf{Op}_a$ aritmeetilised operaatorid
$op_b \in$ $\mathbf{Op}_b$ tõeväärtusoperaatorid
$op_r \in$ $\mathbf{Op}_r$ suhteoperaatorid(*relational*)

Keele süntaks on kirjeldatud abstraktse süntaksiga:
$a ::= x \mid n \mid a_1\ op_a\ a_2$
$b ::= true \mid false \mid not\ b \mid b_1\ op_b\ b_2 \mid a_1\ op_r\ a_2$
$S ::= [x := a]^\ell \mid [skip]^\ell \mid S_1; S_2 \mid if [b]^\ell\ then\ S_1\ else\ S_2 \mid while\ [b]^\ell\ do\ S$

1.3 Reaching Definitions Analysis

$[y := x]^1;\ [z := 1]^2;\ \mathit{while}[y > 1]^3\ \mathit{do}\ ([z := z * y]^4;\ [y := y - 1]^5);\ [y := 0]^6$

ℓ	$RD_{entry}(\ell)$	$RD_{exit}(\ell)$
1	(x,?),(y,?),(z,?)	(x,?),(y,1),(z,?)
2	(x,?),(y,1),(z,?)	(x,?),(y,1),(z,2)
3	(x,?),(y,1),(y,5),(z,2),(z,4)	(x,?),(y,1),(y,5),(z,2),(z,4)
4	(x,?),(y,1),(y,5),(z,2),(z,4)	(x,?),(y,1),(y,5),(z,4)
5	(x,?),(y,1),(y,5),(z,4)	(x,?),(y,5),(z,4)
6	(x,?),(y,1),(y,5),(z,2),(z,4)	(x,?),(y,6),(z,2),(z,4)

Täisinformatsioon programmi kohta paarides $RD = (RD_{entry}, RD_{exit})$

1.4 Andmevooanalüüs

Andmevooanalüüsis võetakse programmi kui graafi. Tipud on elementaarblokid ning kaared kirjeldavad võimalikke viise kuidas kontroll võib liikuda ühest blokist teise.

1.4.1 Equational Approach

$[y := x]^1;\ [z := 1]^2;\ \mathit{while}[y > 1]^3\ \mathit{do}\ ([z := z * y]^4;\ [y := y - 1]^5);\ [y := 0]^6$

I: tipu väljundinformatsiooni vastavusse panek sama tipu sisendinformatsiooniga

$RD_{exit}(1) = (RD_{entry}(1) \setminus \{(y, \ell) | \ell \in \mathbf{Lab}\}) \cup \{(y, 1)\}$
$RD_{exit}(2) = (RD_{entry}(2) \setminus \{(z, \ell) | \ell \in \mathbf{Lab}\}) \cup \{(z, 2)\}$
$RD_{exit}(3) = RD_{entry}(3)$
$RD_{exit}(4) = (RD_{entry}(4) \setminus \{(z, \ell) | \ell \in \mathbf{Lab}\}) \cup \{(z, 4)\}$
$RD_{exit}(5) = (RD_{entry}(5) \setminus \{(y, \ell) | \ell \in \mathbf{Lab}\}) \cup \{(y, 5)\}$
$RD_{exit}(6) = (RD_{entry}(6) \setminus \{(y, \ell) | \ell \in \mathbf{Lab}\}) \cup \{(y, 6)\}$

II: tipu sisendinformatsiooni vastavusse panek nende tippude väljundinformatsiooniga kust leidub kaar antud tippu

$RD_{entry}(2) = RD_{exit}(1)$
$RD_{entry}(3) = RD_{exit}(2) \cup RD_{exit}(5)$
$RD_{entry}(4) = RD_{exit}(3)$
$RD_{entry}(5) = RD_{exit}(4)$
$RD_{entry}(6) = RD_{exit}(3)$

$RD_{entry}(1) = \{(x, ?), (y, ?), (z, ?)\}$

1.4.2 The Constraint Based Approach

"Equational Approach“i alternatiiv. Idee seisneb kaasaarvamiste (inclusions or inequations or constraints) väljavõtmises
$[y := x]^1;\ [z := 1]^2;\ \mathit{while}[y > 1]^3\ \mathit{do}\ ([z := z * y]^4;\ [y := y - 1]^5);\ [y := 0]^6$

$RD_{exit}(1) \supseteq RD_{entry}(1) \setminus \{(y, \ell) | \ell \in \mathbf{Lab}\}$
$RD_{exit}(1) \supseteq \{(y, 1)\}$
$RD_{exit}(2) \supseteq RD_{entry}(2) \setminus \{(z, \ell) | \ell \in \mathbf{Lab}\}$
$RD_{exit}(2) \supseteq \{(z, 2\}$
$RD_{exit}(3) \supseteq RD_{entry}(3)$
$RD_{exit}(4) \supseteq RD_{entry}(4) \setminus \{(z, \ell) | \ell \in \mathbf{Lab}\}$
$RD_{exit}(4) \supseteq \{(z, 4)\}$
$RD_{exit}(5) \supseteq RD_{entry}(5) \setminus \{(y, \ell) | \ell \in \mathbf{Lab}\}$
$RD_{exit}(5) \supseteq \{(y, 5)\}$
$RD_{exit}(6) \supseteq RD_{entry}(6) \setminus \{(y, \ell) | \ell \in \mathbf{Lab}\}$
$RD_{exit}(6) \supseteq \{(y, 6)\}$

ja constraintid mis väljendavad veelgi otsesemalt programmi tööd
$RD_{entry}(2) \supseteq RD_{exit}(1)$
$RD_{entry}(3) \supseteq RD_{exit}(2)$
$RD_{entry}(3) \supseteq RD_{exit}(5)$
$RD_{entry}(5) \supseteq RD_{exit}(4)$
$RD_{entry}(6) \supseteq RD_{exit}(3)$

Üldiselt on meil constraint $RD_{entry}(\ell) \supseteq RD_{exit}(\ell')$ kui on võimalik liikuda elementaarblokist $\ell' -> \ell$.

1.5 Teisendused (Transformations)

Üks põhilistest programmi analüüsi eesmärkidest on programmi transformeerimine selliselt, et me saavutaksime parema *perfomance*. Reaching definitions kasutamine transformatsioonis - Constant Folding.
1. Asendada muutuja mõnes avaldises konstandiga kui on teada, et see tõesti kunagi ei muutu
2. Avaldiste lihtsustamine osalise väljaarvutamise teel: alamavaldised mis ei sisalda muutujaid saab arvutada.

$RD \vdash S \triangleright S'$

$[ass_1]\ RD \vdash [x := a]^\ell \triangleright [x := a[y \to n]]^\ell$
$if \begin{cases} y \in FV(a) \wedge (y, ? \notin RD_{entry}(\ell) \wedge \\ \forall (z, \ell') \in RD_{entry}(\ell) : (z = y) \Rightarrow [...]^{\ell'} is [y := n]^{\ell'}) \end{cases}$

$[ass_2]\ RD \vdash [x := a]^\ell \triangleright [x := n]^\ell$
$if FV(a) = \emptyset \wedge a \notin Num \wedge$ a *evaluates to* n

$$[seq_1]\ \frac{RD \vdash S_1 \triangleright S_1'}{RD \vdash S_1; S_2 \triangleright S_1'; S_2}$$

$$[seq_2]\ \frac{RD \vdash S_2 \triangleright S_2'}{RD \vdash S_1; S_2 \triangleright S_1; S_2'}$$

$$[if_1]\ \frac{RD \vdash S_1 \triangleright S_1'}{RD \vdash if[b]^\ell\ then\ S_1\ else\ S_2 \triangleright if[b]^\ell\ then\ S_1'\ else\ S_2}$$

$$[if_2]\ \frac{RD \vdash S_2 \triangleright S_2'}{RD \vdash if[b]^\ell\ then\ S_1\ else\ S_2 \triangleright if[b]^\ell\ then\ S_1\ else\ S_2'}$$

$$[wh]\ \frac{RD \vdash S \triangleright S'}{RD \vdash while[b]^\ell\ do\ S \triangleright while[b]^\ell\ do\ S'}$$

Edukad teisendused võivad olla kallid kui peale transformeerimist on vaja Reaching Definitions ümber arvutada

lahendus: kui RD on lahendus S_i jaoks ja $RD \vdash S_i \triangleright S_{i+1}$ siis RD on lahendus ka S_{i+1} jaoks - transformeerimisel muudeti ainult selliseid asju mida Reaching Definitions analüüs ei jälginud.

2 Andmevoo analüüs

2.1 Intraprocedural Analysis

$init$: **Stmt** $\rightarrow$ **Lab**
$init([x := a]^\ell) = \ell$
$init([skip]^\ell) = \ell$
$init(S_1; S_2) = init(S_1)$
$init(if[b]^\ell \ then \ S_1 \ else \ S_2) = \ell$
$init(while[b]^\ell \ do \ S) = \ell$

$final$: **Stmt** $\rightarrow P($**Lab**$)$
$final([x := a]^\ell) = \{\ell\}$
$final([skip]^\ell) = \{\ell\}$
$final(S_1; S_2) = final(S_2)$
$final(if[b]^\ell \ then \ S_1 \ else \ S_2) = final(S_1) \cup final(S_2)$
$final(while[b]^\ell \ do \ S) = \{\ell\}$

$blocks$: **Stmt** $\rightarrow P($**Blocks**$)$
$blocks([x := a]^\ell) = \{[x := a]\}$
$blocks([skip]^\ell) = \{[skip]\}$
$blocks(S_1; S_2) = blocks(S_1) \cup blocks(S_2)$
$blocks(if[b]^\ell \ then \ S_1 \ else \ S_2) = \{[b]^\ell\} \cup blocks(S_1) \cup blocks(S_2)$
$blocks(while[b]^\ell \ do \ S) = \{[b]^\ell\} \cup blocks(S)$

$labels$: **Stmt** $\rightarrow P($**Lab**$)$ $labels(S) = \{\ell | [B]^\ell \in blocks(S)\}$

Voog ja pööratud voog
$flow$: **Stmt** $\rightarrow P($**Lab** $\times$ **Lab**$)$
$flow([x := a]^{\ell}) = \emptyset$
$flow([skip]^{\ell}) = \emptyset$
$flow(S_1; S_2) = flow(S_1) \cup flow(S_2) \cup \{(\ell, init(S_2)) | \ell \in final(S_1)\}$
$flow(while[b]^{\ell}\ then\ S_1\ else\ S_2) =$
$flow(S_1) \cup flow(S_2) \cup \{(\ell, init(S_1)), (\ell, init(S_2))\}$
$flow(while[b]^{\ell}\ do\ S) = flow(S) \cup \{(\ell, init(S))\} \cup \{(\ell', \ell) | \ell' \in final(S)\}$

$flow^R$: **Stmt** $\rightarrow P($**Lab** $\times$ **Lab**$)$
on defineeritud kui $flow^R(S) = \{(\ell, \ell') | (\ell', \ell) \in flow(S)\}$

Näiteprogramm
$[z := 1]^1; while[x > 0]^2 do([z := z * y]^3; [x := x - 1]^4)$

$labels(S) = \{init(S)\} \cup \{\ell | (\ell, \ell') \in flow(S)\} \cup \{\ell' | (\ell, \ell') \in flow(S)\}$

Isoleeritud sisendpunktid $\forall \ell \in$ **Lab** : $(\ell, init(S_*)) \notin flow(S_*)$

Isoleeritud väljundpunktid $\forall \ell_1 \in final(S_*) \forall \ell_2 \in$ **Lab** : $(\ell_1, \ell_2) \notin flow(S_*)$

2.1.1 Kättesaadavate avaldiste analüüs (Available Expressions)

Iga programmipunkti kohta millised avaldised kindlalt on juba välja arvutatud, mida hiljem ei muudeta ühelgi võimalikul teel mis viivad antud porgrammipunkti. Analüüsi tulemusena saadud informatsiooni saab kasutada üleliigsete arvutuste vältimiseks.
$[x := a + b]^1;\ [y := a * b]^2;\ while[y > a + b]^3\ do\ ([a := a + 1]^4;\ [x := a + b]^5);$
a+b on iga kord kättesaadav kui täitmine jõuab tsükli testini (label 3)

$kill_{AE}([x := a]^\ell) = \{a' \in \mathbf{AExp}_* | x \in FV(a')\}$
$kill_{AE}([skip]^\ell) = \emptyset$
$kill_{AE}([b]^\ell) = \emptyset$

$gen_{AE}([x := a]^\ell) = \{a' \in \mathbf{AExp}(a) | x \notin FV(a')\}$
$gen_{AE}([skip]^\ell) = \emptyset$
$gen_{AE}([b]^l) = \mathbf{AExp}(b)$

Andmevoovõrdused
$$AE_{entry}(\ell) = \begin{cases} \emptyset & kui\ \ell = init(S_*) \\ \bigcup AE_{exit}(\ell') | (\ell', \ell) \in flow(S_*) & muudel\ juhtudel \end{cases}$$
$AE_{exit}(\ell) = (AE_{entry}(\ell) \setminus kill_{AE}(B^\ell)) \cap gen_{AE}(B^\ell) \quad kus\ B^\ell \in blocks(S_*)$
Avaldis loetakse kättesaadavaks kui teda teel ei tapeta.

$[z := x + y]^\ell; while[true]^{\ell'}\ do\ [skip]^{\ell''}$

$AE_{entry}(\ell) = \emptyset$
$AE_{entry}(\ell') = AE_{exit}(\ell) \cap AE_{exit}(\ell'')$
$AE_{entry}(\ell'') = AE_{exit}(\ell')$

$AE_{exit}(\ell) = AE_{entry}(\ell) \cup \{x + y\}$
$AE_{exit}(\ell') = AE_{entry}(\ell')$
$AE_{exit}(\ell'') = AE_{entry}(\ell'')$

Peale lihtustamist saame $AE_{entry}(\ell') = \{x + y\} \cap AE_{entry}(\ell')$

Näide
$[x := a + b]^1; [y := a * b]^2; while[y > a + b]^3\ do\ ([a := a + 1]^4; [x := a + b]^5)$

2.1.2 Reaching Definitions Analysis

Õigem oleks nimetada "Reaching assignments".
Iga programmipunkti kohta, millised omistamised võivad olla teostatud ja mitte ülekirjutatud kui jõutakse antud punkti mööda mõnda teed. Peamine rakendus on seoste (direct links) tekitamiseks blokkide vahel kus luuakse väärtusi ning kus neid kasutatakse.

Näide
$[x := 5]^1; [y := 1]^2; \mathit{while} [x > 1]^3 \ \mathit{do}\ ([y := x * y]^4; [x := x - 1]^5)$

$\mathit{kill}_{RD}([x := a]^\ell) = \{(x, ?)\} \cup \{(x, \ell) | B^{\ell'} \ \mathit{is\ an\ assignment\ to}\ x\ \mathit{in}\ S_*\}$
$\mathit{kill}_{RD}([\mathit{skip}]^\ell) = \emptyset$
$\mathit{kill}_{RD}([b]^\ell) = \emptyset$

$\mathit{gen}_{RD}([x := a]^\ell) = \{(x, \ell)\}$
$\mathit{gen}_{RD}([\mathit{skip}]^\ell) = \emptyset$
$\mathit{gen}_{RD}([b]^\ell) = \emptyset$

Andmevoovõrdused
$$RD_{entry}(\ell) = \begin{cases} \{(x, ?) | x \in FV(S_*)\} & \mathit{kui}\ \ell = \mathit{init}(S_*) \\ \bigcup RD_{exit}(\ell') | (\ell', \ell) \in \mathit{flow}(S_*)\ \mathit{muudel\ juhtudel} \end{cases}$$
$$RD_{exit}(\ell) = (RD_{entry}(\ell) \setminus \mathit{kill}_{RD}(B^\ell)) \cup \mathit{gen}_{RD}(B^\ell) \quad \mathit{kus}\ B^\ell \in \mathit{blocks}(S_*)$$

$[z := x + y]^\ell; \mathit{while} [\mathit{true}]^{\ell'}\ \mathit{do}\ [\mathit{skip}]^{\ell''}$

$RD_{entry}(\ell) = \{(x, ?), (y, ?), (z, ?)\}$
$RD_{entry}(\ell') = RD_{exit}(\ell) \cup RD_{exit}(\ell'')$
$RD_{entry}(\ell'') = RD_{exit}(\ell')$

$RD_{exit}(\ell) = (RD_{entry}(\ell) \setminus \{(z, ?)\}) \cup \{(x, \ell)\}$
$RD_{exit}(\ell') = (RD_{entry}(\ell')$
$RD_{exit}(\ell'') = (RD_{entry}(\ell'')$

Peale lihtsustamist saame $RD_{entry}(\ell') = \{(x, ?), (y, ?), (z, \ell)\} \cup RD_{entry}(\ell')$

2.1.3 Very Busy Expression Analysis

Avaldis on "very busy"kui märgendist väljudes, ükskõik millist teed valides antud märgendist seda avaldist kasutatakse enne kui mõnda muutujat mis sellesse avaldisse kuulub muudetakse. Analüüsi tulemusel saadud informatsiooni saab kasutada optimiseerimiseks - mõned avaldised bloki seest saab välja arvutada ning neid väärtuseid hiljem kasutada.

$if[a > b]^1\ then\ ([x := b - a]^2; [y := a - b]^3)\ else\ ([y := b - a]^4; [x := a - b]^5)$

$kill_{VB}([x := a]^\ell) = \{a' \in \mathbf{AExp}_* | x \in FV(a')\}$
$kill_{VB}([skip]^\ell) = \emptyset$
$kill_{VB}([b]^\ell) = \emptyset$

$gen_{VB}([x := a]^\ell) = \mathbf{AExp}(a)$
$gen_{VB}([skip]^\ell) = \emptyset$
$gen_{VB}([b]^\ell) = \mathbf{AExp}(b)$

Andmevoovõrdused
$$VB_{exit}(\ell) = \begin{cases} \emptyset & kui\ \ell = init(S_*) \\ \bigcap VB_{entry}(\ell') | (\ell', \ell) \in flow^R(S_*) & muudel\ juhtudel \end{cases}$$
$$VB_{entry}(\ell) = (VB_{exit}(\ell) \setminus kill_{VB}(B^\ell)) \cup gen_{VB}(B^\ell) \quad kus\ B^\ell \in blocks(S_*)$$

$(while[x > 1]^\ell\ do\ [skip]^{\ell'}); [x := x + 1]^{\ell''}$

$VB_{entry}(\ell) = VB_{exit}(\ell)$
$VB_{entry}(\ell') = VB_{exit}(\ell')$
$VB_{entry}(\ell'') = \{x + 1\}$

$VB_{exit}(\ell) = VB_{entry}(\ell') \cap VB_{entry}(\ell'')$
$VB_{exit}(\ell') = VB_{entry}(\ell)$
$VB_{exit}(\ell'') = \emptyset$

l jaoks saame $VB_{exit}(\ell) = VB_{exit}(\ell) \cap \{x + 1\}$

2.1.4 Live Variables analysis

Muutuja on elus märgendist väljumisel kui leidub tee sellest märgendist muutuja kasutamiseni kus seda muutujat ei defineerita ümber. Analüüs määrabki iga programmi punkti kohta millised muutujad võivad olla elus selle punkti väljundis. Tegu on tagurpidi analüüsiga. Analüüsi saab kasutada "Surnud koodi ellimineerimiseks kui muutuja ei ole bloki lõpus elus ja selle bloki sisuks on omistamine võib selle bloki ellimineerida.

$[x := 2]^1; [y := 4]^2; [x := 1]^3; (if [y > x]^4 then [z := y]^5 else [z := y * y]^6); [x := z]^7$

$kill_{LV}([x := a]^\ell) = \{x\}$
$kill_{LV}([skip]^\ell) = \emptyset$
$kill_{LV}([b]^\ell) = \emptyset$

$gen_{LV}([x := a]^\ell) = \mathbf{FV}(a)$
$gen_{LV}([skip]^\ell) = \emptyset$
$gen_{LV}([b]^\ell) = \mathbf{FV}(b)$

Andmevoovõrdused
$LV_{exit}(\ell) = \begin{cases} \emptyset & kui\ \ell \in final(S_*) \\ \bigcup\{LV_{entry}(\ell') | (\ell', \ell) \in flow^R(S_*)\} & muudel\ juhtudel \end{cases}$
$LV_{entry}(\ell) = (LV_{exit}(\ell) \setminus kill_{LV}(B^\ell)) \cup gen_{LV}(B^\ell) \quad kus\ B^\ell \in blocks(S_*)$

$(while [x > 1]^\ell do [skip]^{\ell'}); [x := x + 1]^{\ell''}$

$LV_{entry}(\ell) = LV_{exit}(\ell) \cup \{x\}$
$LV_{entry}(\ell') = LV_{exit}(\ell')$
$LV_{entry}(\ell'') = \{x\}$

$LV_{exit}(\ell) = LV_{entry}(\ell') \cup LV_{entry}(\ell'')$
$LV_{exit}(\ell') = LV_{entry}(\ell')$
$LV_{exit}(\ell'') = \emptyset$

l jaoks saame $LV_{exit}(\ell) = LV_{exit}(\ell) \cup \{x\}$

2.1.5 Tuletatud andmevoo informatsioon (Derived Data Flow Information)

Sageli lingitakse märgend, kus luuakse väärtusi, teiste märgendite, mis neid väärtuseid kasutavad, külge.
Linke, mis iga muutuja kasutamise kohta seovad kõik omistamised mis jõuavad selle kasutamiseni nimetatakse Use-Definition chains või ud-chains.
Linke, mis iga omistamise kohta seovad kõik kasutused kutsutakse Definition-Use chains ehk du-chains.

definition clear path: $\ell_1, ..., \ell_n$ muutuja x jaoks kui ükski label ei omista x'ile uut väärtust ja kui ℓ_n kasutab x.

$$
\begin{aligned}
clear(x, \ell, \ell') = \exists \ell_1, ..., \ell_n : \\
&(\ell_1 = \ell) \wedge (\ell_n = \ell' \wedge (n > 0) \wedge \\
&(\forall i \in \{1, ..., n-1\} : (\ell_1, \ell_{i+1}) \in flow(S_*)) \wedge \\
&(\forall i \in \{1, ..., n-1\} : \neg def(x, \ell_i)) \wedge use(x, \ell_n)
\end{aligned}
$$

$$use(x, \ell) = (\exists B : [B]^\ell \in blocks(S_*) \wedge x \in gen_{LV}([B]^\ell))$$

$$def(x, \ell) = (\exists B : [B]^\ell \in blocks(S_*) \wedge x \in kill_{LV}([B]^\ell))$$

Defineerime funktsioonid

$$
\begin{aligned}
ud(x, \ell') = &\{\ell | def(x, \ell) \wedge \exists \ell'' : (\ell, \ell'') \in flow(S_*) \wedge clear(x, \ell'', \ell')\} \\
&\cup \{? | clear(x, init(S_*), \ell')\}
\end{aligned}
$$

$$
du(x, \ell) = \begin{cases} \{\ell' | def(x, \ell) \wedge \exists \ell'' : (\ell, \ell'') \in flow(S_*) \wedge clear(x, \ell'', \ell')\} & kui\ \ell \neq ? \\ \{\ell' | clear(x, init(S_*), \ell')\} & kui\ \ell = ? \end{cases}
$$

Näide:
$[x := 0]^1; [x := 3]^2; (if [z = x]^3 then [z := 0]^4 else [z := x]^5); [y := x]^6; [x := y + z]^7$

$ud(x, \ell)$	x	y	z
1	∅	∅	∅
2	∅	∅	∅
3	{2}	∅	{?}
4	∅	∅	∅
5	{2}	∅	∅
6	{2}	∅	∅
7	∅	{6}	{4,5}

$du(x, \ell)$	x	y	z
1	∅	∅	∅
2	{3,5,6}	∅	∅
3	∅	∅	∅
4	∅	∅	{7}
5	∅	∅	{7}
6	∅	{7}	∅
7	∅	∅	∅
?	∅	∅	{3}

ud- ja du-ahelaid saab kasutada "Surnud koodi ellimineerimiseks"ja "Koodi tõstmiseks"(bloki tõstmine ettepoole)

2.2 Theoretical Properties

Eesmärk on näidata, et "Live Variables" analüüs on korrektne

2.2.1 Struktuurne operatsioonsemantika

$\sigma \in \mathbf{State} = \mathbf{Var} \to \mathbf{Z}$
Üleminekud kujul $\langle S, \sigma\rangle \to \sigma'$ ja $\langle S, \sigma\rangle \to \langle S', \sigma'\rangle$

Lemma
(*i*) $kui \langle S, \sigma\rangle \to \sigma'$ $siis$ $final(S) = \{init(S)\}$
(*ii*) $kui \langle S, \sigma\rangle \to \langle S', \sigma'\rangle$ $siis$ $final(S) \supseteq final(S')$
(*iii*) $kui \langle S, \sigma\rangle \to \langle S', \sigma'\rangle$ $siis$ $flow(S) \supseteq flow(S')$
(*iv*) $kui \langle S, \sigma\rangle \to \langle S', \sigma'\rangle$ $siis$ $blocks(S) \supseteq blocks(S')$ ja kui S on kooskõlalise märgistusega siis seda on ka S'

Tõestused
[i, ass] $\langle [x := a]^\ell, \sigma\rangle \to \sigma[x \mapsto A[\![a]\!]\sigma]$
ja me saame
$final([x := a]^\ell) = \{\ell\} = \{init([x := a]^\ell)\}$
...
[ii, seq] $\langle S_1; S_2, \sigma\rangle \to \langle S_1'; S_2, \sigma'\rangle$ sest $\langle S_1, \sigma\rangle \to \langle S_1', \sigma'\rangle$
ja me saame
$final(S_1; S_2) = final(S_2) = final(S_1'; S_2)$
...
[iii, seq] $\langle S_1; S_2, \sigma\rangle \to \langle S_1'; S_2, \sigma'\rangle$ sest $\langle S_1, \sigma\rangle \to \langle S_1', \sigma'\rangle$
ja me saame

$$\begin{aligned} flow(S_1; S_2) &= flow(S_1) \cup flow(S_2) \cup \{(\ell, init(S_2)) | \ell \in final(S_1)\} \\ &\supseteq flow(S_1') \cup flow(S_2) \cup \{(\ell, init(S_2)) | \ell \in final(S_1)\} \\ &\supseteq flow(S_1') \cup flow(S_2) \cup \{(\ell, init(S_2)) | \ell \in final(S_1')\} \\ &= flow(S_1'; S_2) \end{aligned}$$

2.2.2 "Live Variables" analüüsi korrektsus

võrduste süsteem kooskõlalise märgistusega programmil S_* viitame edaspidi kui $LV^{=}(S_*)$. Seda annab muuta ning saada kitsenduste süsteem $LV^{\subseteq}(S_*)$ kujul

$$LV_{exit}(\ell) \supseteq \begin{cases} \emptyset & kui\ \ell \in final(S_*) \\ \bigcup\{LV_{entry}(\ell')|(\ell',\ell) \in flow^R(S_*)\} & muudel\,juhtudel \end{cases}$$

$$LV_{entry}(\ell) \supseteq (LV_{exit}(\ell) \setminus kill_{LV}(B^\ell)) \cup gen_{LV}(B^\ell) \quad kus B^\ell \in blocks(S_*)$$

$$live_{entry}, live_{exit} : \mathbf{Lab}_* \to P(\mathbf{Var}_*)$$

Me ütleme, et *live* on $LV^{=}(S)$ lahendiks ja kirjutame
$live \models LV^{=}(S)$
kui funktsioonid rahuldavad võrdusi; Sama moodi kirjutame
$live \models LV^{\subseteq}(S)$
kui *live* on $LV^{\subseteq}(S)$ lahendiks.

Lemma 2.15
Olgu S_* kooskõlalise märgistusega programm. Kui $live \models LV^{=}(S_*)$ siis ka $live \models LV^{\subseteq}(S)$
Tõestus
Kui $live \models LV^{=}(S)$ siis selgelt ka $live \models LV^{\subseteq}(S)$ sest $\supseteq$ sisaldab ka $=$.

Lemma 2.16
Kui $live \models LV^{\subseteq}(S_1)$ (S_1 on kooskõlalise märgistusega) ja $flow(S_1) \supseteq flow(S_2)$ ja $blocks(S_1) \supseteq blocks(S_2)$ siis $live \models LV^{\subseteq}(S_2)$ (S_2 on kooskõlalise märgistusega)
Tõestus
Kui S_1 on kooskõlalise märgistusega ja $blocks(S_1) \supseteq blocks(S_2)$ siis samuti ka S_2 on kooskõlalise märgistusega. Kui $live \models LV^{\subseteq}(S_1)$ siis *live* rahuldab ka kõik kitsendused $LV^{\subseteq}(S_2)$ ja seetõttu $live \models LV^{\subseteq}(S_2)$

Järeldus 2.17
Kui $live \models LV^{\subseteq}(S)$ (S on kooskõlalise märgistusega) ja kui $\langle S, \sigma\rangle \to \langle S', \sigma'\rangle$ siis samuti ka $live \models LV^{\subseteq}(S')$

Lemma 2.18
Kui $live \models LV^{\subseteq}(S)$ (S on kooskõlalise märgistusega) siis iga paari $(\ell, \ell') \in flow((S)$ on meil $live_{exit}(\ell) \supseteq live_{entry}(\ell')$

Võtame V (kõigi live muutujate hulk) ja defineerime korrektsuse suhte (relation):
$\sigma_1 \sim_V \sigma_2$ kui $\forall x \in V : \sigma_1(x) = \sigma_2(x)$
See suhe väljendab seda, et kõigil praktilistel eesmärkidel on kaks seisundit σ_1 ja σ_2 võrdsed : loevad ainult live muutujad.

Näide $[x := y + z]^\ell$ *ja* $V_1 = \{y, z\}$ *ja* $V_2 = \{x\}$ Siis $\sigma_1 \sim_{V_1} \sigma_2$ tähendab $\sigma_1(y) = \sigma_2(y) \wedge \sigma_1(z) = \sigma_2(z)$ ja $\sigma_1 \sim_{V_2} \sigma_2$ tähendab $\sigma_1(x) = \sigma_2(x)$
Võtame, et $\langle [x := y + z]^\ell, \sigma_1 \rangle \rightarrow \sigma_1'$ ja $\langle [x := y + z]^\ell, \sigma_2 \rangle \rightarrow \sigma_2'$. Nüüd $\sigma_1 \sim_{V_1} \sigma_2$ kindlustab et $\sigma_1' \sim_{V_2} \sigma_2'$ Kui V_2 on live muutujate hulk peale $[x := y + z]^\ell$ siis V_1 on live muutujate hulk enne $[x := y + z]^\ell$

$N(\ell) = live_{entry}(\ell)$
$X(\ell) = live_{exit}(\ell)$

Lemma 2.20
Oletame, et $live \models LV^{\subseteq}(S)$ (S on kooskõlalise märgistusega) siis $\sigma_1 \sim_{X(\ell)} \sigma_2$ -st järeldub (implies) $\sigma_1 \sim_{N(\ell')} \sigma_2$ iga paari $(\ell, \ell') \in flow(S)$ kohta.

Põhieesmärk. Näitab. et semantiliselt korrektne *liveness* informatsioon on säilitatud igal sammul: (i) juhul kui programm koheselt ei termineeru ja (ii) juhul kui programm koheselt termineerub

Teoreem 2.21
Kui $live \models LV^{\subseteq}(S)$ (S on kooskõlalise märgistusega) siis
(i) *kui* $\langle S, \sigma_1\rangle \rightarrow \langle S', \sigma_1'\rangle$ *ja* $\sigma_1 \sim_{N(init(S))} \sigma_2$ *siis leidub selline* σ_2' *et* $\langle S, \sigma_2\rangle \rightarrow \langle S', \sigma_2'\rangle$ *ja* $\sigma_1' \sim_{N(init(S'))} \sigma_2'$
(ii) *kui* $\langle S, \sigma_1\rangle \rightarrow \sigma_1'$ *ja* $\sigma_1 \sim_{N(init(S))} \sigma_2$ *siis leidub selline* σ_2' *et* $\langle S, \sigma_2\rangle \rightarrow \sigma_2'$ *ja* $\sigma_1' \sim_{X(init(S))} \sigma_2'$

Tõestus
[ass] $\langle [x := a]^{\ell}, \sigma_1\rangle \rightarrow \sigma_1[x \mapsto A[\![a]\!]\sigma_1]$ ja kitsenduste süsteemi vormis saame
$N(\ell) = live_{entry}(\ell) = (live_{exit}(\ell) \setminus \{x\}) \cup FV(a) = (X(\ell) \setminus \{x\}) \cup FV(a)$
seega $\sigma_1 \sim_{N(\ell)} \sigma_2$ järeldub $A[\![a]\!]\sigma_1 = A[\![a]\!]\sigma_2$
sest a väärtus on mõjutatud ainult temas leiduvatest muutujatest. Seega võttes $\sigma_2' = \sigma_2[x \mapsto A[\![a]\!]\sigma_2]$ on meil $\sigma_1'(x) = \sigma_2'(x)$ ja seega $\sigma_1' \sim_{X(\ell)} \sigma_2'$ nagu nõutud.

2.3 Monotoonsed raamistikud

Hoolimata programmianalüüside erinevustest on seal piisavalt palju sarnasusi selleks, et pidada tõenäoliseks, et neil on ühine alusraamistik.
Kõik neli klassikalist analüüsi tegelevad võrdustega kooskõlalise märgistusega programmil S_* ja on kujul

$$Analysis_\circ(\ell) = \begin{cases} \iota & kui\ \ell \in E \\ \bigsqcup\{Analysis_\bullet(\ell') | (\ell', \ell) \in F\} & muudel\ juhtudel \end{cases}$$
$$Analysis_\bullet(\ell) = f_\ell(Analysis_\circ(\ell))$$

* $\bigsqcup$ on $\bigcap$ või $\bigcup$
* F on kas $flow(S_*)$ või $flow^R(S_*)$
* E on $\{init(S_*)\}$ või $final(S_*)$
* ι kirjeldab init või final analüüsi informatiooni
* f_ℓ on teisendusfunktsioon seotud $B^\ell \in blocks(S_*)$

Available Expressions : päripidine analüüs, suurim lahendus
Reaching Definitions : päripidine analüüs, vähim lahendus
Very Busy Expressions : tagurpidi analüüs, suurim lahendus
Live Variables: tagurpidi analüüs, vähim lahendus

* "Päripidiste analüüside"korral F on $flow(S_*)$ ja $Analysis_\circ$ arvestab sisendtingimusi ning $Analysis_\bullet$ arvestab väljundtingimusi. Võrduste süsteem eeldab, et S_* omab isoleeritud sisendpunkte
* "Tagurpidi analüüside"korral F on $flow^R(S_*)$ ja $Analysis_\circ$ arvestab väljundtingimusi ning $Analysis_\bullet$ arvestab sisendtingimusi. Võrduste süsteem eeldab, et S_* omab isoleeritud väljundpunkte

* Kui $\bigsqcup$ on $\bigcap$ nõuame me suurimat hulka mis lahendab võrdused ja me saame määrata tingimused mis on rahuldatud kõigil teedel mis jõuavad (või lahkuvad) märgendi sisendisse (või väljundist). Neid analüüse nimetatakse *must analyses*
* Kui $\bigsqcup$ on $\bigcup$ nõuame me vähimat hulka mis lahendab võrdused ja me saame määrata tingimused mis on rahuldatud vähemalt ühel teel mis jõuab (või lahkub) märgendi sisendisse (või väljundist). Neid analüüse nimetatakse *may analyses*

2.3.1 Basic Definitions

Tingimuste ruumid (Property spaces) Oluline koostisosa raamistikus on "tingimuste ruum", L - esitab nii andmevoo informatsiooni kui on ka kasutusel kombinatsiooni operaatorina, $\bigsqcup : P(L) \to L$ mis kombineerib informatsiooni programmi erinevatelt teedelt.
$\sqcup : L \times L \to L$ on defineeritud kui $l_1 \sqcup l_2 = \bigsqcup\{l_1, l_2\}$ ja $\bigsqcup \emptyset$ jaoks kirjutame $\bot$

Teisendusfunktsioonid (Transfer functions) Teine oluline raamistiku koostisosa - teisendusfunktsioonid, $f_\ell : L \to L \; kus \; l \in \mathbf{Lab}_*$
Need funktsioonid peavad olema monotoonsed ehk $l \sqsubseteq l'$ järeldub $f_\ell(l) \sqsubseteq f_\ell(l')$
Selleks, et kontrollida teisendusfunktioone me nõuame, et hulk $\mathcal{F}$ sisaldaks monotoonseid funktsioone üle L ning täidaks järgmisi tingimusi:
* $\mathcal{F}$ sisaldab kõik teisendusfunktsioonid f_ℓ
* $\mathcal{F}$ sisaldab identsusfunktsiooni id ja
* $\mathcal{F}$ on kinnine funktsioonide kompositiooni suhtes

Kokkuvõttes: Monotoonne Raamistik koosneb:
* täielikust võrest, L, mis rahuldab Kasvava Ahela Tingimust (Ascending Chain Condition) ja me kirjutame $\bigsqcup$ vähima ülemise tõkke (least upper bound) operaatori jaoks
* monotoonsete funktsioonide hulgast $\mathcal{F}$ hulgast L hulka L mis sisaldab identsusfunktsiooni ja mis on kinnine funktsioonide kompositsiooni suhtes.

Nõuame ka, et kõik funktsioonid $f \in \mathcal{F}$ peavad olema distributiivsed:
$f(l_1 \sqcup l_2) = f(l_1) \sqcup f(l_2)$

Nõuded (Instances) Andmevoovõrdustest on näha, et ainult monotoonsest raamistikust ei piisa analüüsi kirjeldamiseks. Selleks defineerime me nõude, et analüüs monotoonsest raamistikust koosneb:
* täielikust võrest, L
* funktsioonide ruumist, $\mathcal{F}$
* lõplikust (finite) voost, F, mis tavaliselt on $flow(S_*)$ või $flow^R(S_*)$
* lõplikust hulgast äärmuslikest (extremal) märgenditest, E mis tavaliselt on $\{init(S_*)\}$ või $final(S_*)$
* äärmuslikust väärtusest $\iota \in L$ märkimaks äärmuslikke märgendeid
* ja kujutisest f. märgendite hulgast(F'ist ja E'st) teisendusfunktsioonideks $\mathcal{F}$'is

$$Analysis_\circ(\ell) = \bigsqcup\{Analysis_\bullet(\ell')|(\ell',\ell) \in F\} \sqcup \iota_E^\ell$$
$$kus\ \iota_E^\ell = \begin{cases} \iota & kui\ \ell \in E \\ \bot & kui\ \ell \notin E \end{cases}$$
$$Analysis_\bullet(\ell) = f_\ell(Analysis_\circ(\ell))$$

Kitsendused
$$Analysis_\circ(\ell) \sqsupseteq \bigsqcup\{Analysis_\bullet(\ell')|(\ell',\ell) \in F\} \sqcup \iota_E^\ell$$
$$kus\ \iota_E^\ell = \begin{cases} \iota & kui\ \ell \in E \\ \bot & kui\ \ell \notin E \end{cases}$$
$$Analysis_\bullet(\ell) \sqsupseteq f_\ell(Analysis_\circ(\ell))$$

	Available Expressions	Reaching Definitions	Very Busy Expressions	Live Variables
L	$P(\mathbf{AExp}_*)$	$P(\mathbf{Var}_* \times \mathbf{Lab}_*^?)$	$P(\mathbf{AExp}_*)$	$P(\mathbf{Var}_*)$
$\sqsubseteq$	$\supseteq$	$\subseteq$	$\supseteq$	$\subseteq$
$\bigsqcup$	$\bigcap$	$\bigcup$	$\bigcap$	$\bigcup$
$\bot$	$\mathbf{AExp}_*$	$\emptyset$	$\mathbf{AExp}_*$	$\emptyset$
ι	$\emptyset$	$\{(x,?) \mid x \in FV(S_*)\}$	$\emptyset$	$\emptyset$
E	$\{init(S_*)\}$	$\{init(S_*)\}$	$final(S_*)$	$final(S_*)$
F	$flow(S_*)$	$flow(S_*)$	$flow^R(S_*)$	$flow^R(S_*)$
$\mathcal{F}$	$\{f : L \to L \mid \exists l_k, l_g : f(l) = (l \setminus l_k) \cup l_g\}$			
f_ℓ	$f_\ell(l) = (l \setminus kill([B]^\ell)) \cup gen([B]^\ell)$ *kus* $[B]^\ell \in blocks(S_*)$			

Lemma 2.25
Kõik neli andmevooanalüüsi on monotoonsed raamistikud.

Tõestus
Tõestuseks, et analüüsid on Monotoonsed Raamistikud, peame kinnitama, et $\mathcal{F}$'il on vajalikud tingimused: eeldame, et $l \sqsubseteq l'$ Siis $(l \setminus l_k) \sqsubseteq (l' \setminus l_k)$ ja seetõttu $((l \setminus l_k) \cup l_g) \sqsubseteq ((l' \setminus l_k) \cup l_g)$ ja järelikult $f(l) \sqsubseteq f(l')$ nagu nõutud.
See arvutus on õige hoolimata sellest kas $\sqsubseteq$ on $\subseteq$ või $\supseteq$
Identideedi funktioon on F'is: on saadi kui võeti l_k ja l_g võrdseks $\emptyset$'ga.
Funktsioonid F'is on kinnised kompositsiooni suhtes: Oletame, et
$f(l) = (l \setminus l_k) \cup l_g$ ja $f'(l) = (l \setminus l'_k) \cup l'_g$

Arvutame
$$(f \circ f')(l) = (((l \setminus l'_k) \cup l'_g) \setminus l_k) \cup l_g$$
$$= (l \setminus (l'_k \cup l_k)) \cup ((l'_g \setminus l_k) \cup l_g)$$

Niisiis $(f \circ f')(l) = (l \setminus l''_k) \cup l''_g$ kus $l''_k = l'_k \cup l_k$ ja $l''_g = (l'_g \setminus l_k) \cup l_g$

Näide

$[x := a + b]^1;\ [y := a * b]^2;\ while [y > a + b]^3\ do\ ([a := a + 1]^4;\ [x := a + b]^5)$

Täielik huvipakkuv võre $(P(\{a + b, a * b, a + 1\}), \supseteq)$

milles vähim element on $\{a + b, a * b, a + 1\}$

Voog : $\{(1, 2), (2, 3), (3, 4), (4, 5), (5, 3)\}$

Äärmuslikud märgendid: $\{1\}$

Äärmuslik väärtus: $\emptyset$

Ja teisendusfunktsioon, mis on seotud märgentidega:

$f_1^{AE}(Y) = Y \cup \{a + b\}$

$f_2^{AE}(Y) = Y \cup \{a * b\}$

$f_3^{AE}(Y) = Y \cup \{a + b\}$

$f_4^{AE}(Y) = Y \setminus \{a + b, a * b, a + 1\}$

$f_5^{AE}(Y) = Y \cup \{a + b\}$

kus $Y \subseteq \{a + b, a * b, a + 1\}$

2.4 Võrduste lahendamine (Equation Solving)

Kuidas seda raamistikku nüüd kasutada? Kaks lähenemist: esiteks iteratiivne algoritm ja teiseks analüüs mis levitab analüüsi informatsiooni mööda programmi teid (path).

2.4.1 Maksimaalse Püsipunkti lahendus (Maximal Fixed Point, MFP)

Üldine iteratiivne algoritm mis arvutab vähima lahenduse andmevoo võrdustele.

INPUT: An instance of a Monotone Framework: $(L,\mathcal{F},F,E,\iota,f.)$
OUTPUT: $MFP_\circ, MFP_\bullet$
METHOD:
Setp 1: Initialisation (of W and Analysis)
W:=nil;
for all (ℓ, ℓ') in F do
    W:=cons((ℓ, ℓ'), W);
for all ℓ in F or E do
    if $\ell \in$ E then Analysis[ℓ]:=ι
        else Analysis[ℓ]:=$\bot_L$;

Setp 2: Iteration (updating W and Analysis)
while W $\neq$ nil do
    ℓ:=fst(head(W)); ℓ':=snd(head(W));
    W:=tail(W);
    if f_ℓ(Analysis[ℓ]) $\not\sqsubseteq$ Analysis[ℓ'] then
        Analysis[ℓ']:=Analysis[ℓ'] $\sqcup$ f_l(Analysis[ℓ]);
        for all ℓ'' with (ℓ',ℓ'') in F do
            W:=cons((ℓ',ℓ''),W);

Step 3: Presenting the result ($MFP_\circ$ *and* $MFP_\bullet$)
for all l in F or E do
    $MFP_\circ(\ell)$:=Analysis[ℓ];
    $MFP_\bullet(\ell)$:= f_ℓ(Analysis[ℓ]);

$[x := a + b]^1$; $[y := a * b]^2$; *while* $[y > a + b]^3$ *do* $([a := a + 1]^4$; $[x := a + b]^5)$;
W = ((2,3),(3,4),(4,5),(5,3))
U = {a+b,a*b,a+1}

Analysis[ℓ]

	W	l=1	l=2	l=3	l=4	l=5
1	((1,2),W)	∅	U	U	U	U
2	((2,3),W)	∅	{a+b}	U	U	U
3	((3,4),W)	∅	{a+b}	{a+b,a*b}	U	U
4	((4,5),W)	∅	{a+b}	{a+b,a*b}	{a+b,a*b}	U
5	((5,3),W)	∅	{a+b}	{a+b,a*b}	{a+b,a*b}	∅
6	((3,4),W)	∅	{a+b}	{a+b}	{a+b,a*b}	∅
7	((4,5),W)	∅	{a+b}	{a+b}	{a+b}	∅
8	((2,3),...)	∅	{a+b}	{a+b}	{a+b}	∅
9	((3,4),...)	∅	{a+b}	{a+b}	{a+b}	∅
10	((4,5),...)	∅	{a+b}	{a+b}	{a+b}	∅
11	((5,3))	∅	{a+b}	{a+b}	{a+b}	∅
12	()	∅	{a+b}	{a+b}	{a+b}	∅

Lahendus

l	$AE_{entry}(l)$	$AE_{exit}(l)$
1	∅	{a+b}
2	{a+b}	{a+b, a*b}
2	{a+b}	{a+b}
2	{a+b}	∅
2	∅	{a+b}

2.4.2 Meet Over all Paths (MOP) solution

Teed (Paths)
Olgu meil monotoonse raamistiku instants $(L,\mathcal{F},F,E,\iota,f.)$. Kasutame notatsiooni $\vec{\ell} = [\ell_1, ..., \ell_n]$ märgendite reale kus $n \geq 0$. Defineerime kaks teed: Tee kus ei ole ole sees märgentit ℓ:
$path_\circ(\ell) = \{[\ell_1, ..., \ell_{n-1}] | n \geq 1 \wedge \forall i \leq n : (\ell_i, \ell_{i+1}) \in F \wedge \ell_n = l \wedge \ell_1 \in E\}$
ja tee kus on märgend l sees:
$path_\bullet(\ell) = \{[\ell_1, ..., \ell_n] | n \geq 1 \wedge \forall i \leq n : (\ell_i, \ell_{i+1}) \in F \wedge \ell_n = l \wedge \ell_1 \in E\}$

Tee $\vec{\ell} = [\ell_1, ..., \ell_n]$ jaoks defineerime teisendusfunktsiooni
$f_{\vec{\ell}} = f_{\ell_n} \circ ... \circ f_{\ell_1} \circ id$

$MOP_\circ(\ell) = \bigsqcup\{f_{\vec{\ell}}(\iota) | \vec{\ell} \in path_\circ(\ell)\}$
$MOP_\bullet(\ell) = \bigsqcup\{f_{\vec{\ell}}(\iota) | \vec{\ell} \in path_\bullet(\ell)\}$

MOP vs MFP
MFP lahendus läheneb turvaliselt MOP lahendusele (MFP $\sqsupseteq$ MOP)
Juhtudel ($\bigcap, \rightarrow, \uparrow$) või ($\bigcap, \leftarrow, \uparrow$) on MFP lahendus alamhulk MOP lahendusele
Juhtudel ($\bigcup, \rightarrow, \downarrow$) või ($\bigcup, \leftarrow, \downarrow$) on MFP lahendus ülemhulk MOP lahendusele

Lemma 2.32
Olgu MFP ja MOP lahendused monotoonse raamistiku instantsile
$(L, \mathcal{F}, F, E, \iota, f.)$, siis
$MFP_\circ \sqsupseteq MOP_\circ$ and $MFP_\bullet \sqsupseteq MOP_\bullet$

Tõestus
$\forall \ell : MOP_\bullet(\ell) \sqsubseteq f_\ell(MOP_\circ(\ell))$
$\forall \ell : MFP_\bullet(\ell) \sqsubseteq f_\ell(MFP_\circ(\ell))$

Nüüd peame näitama, et $\forall \ell : MOP_\circ(\ell) \sqsubseteq MFP_\circ(\ell)$
Paneme tähele, et $MFP_\circ$ on vähim püsipunkt funktsioonile mis on defineeritud kui
$F(A_\circ)(\ell) = (\bigsqcup\{f_{\ell'}(A_\circ(\ell'))|(\ell', \ell) \in F\}) \sqcup \iota^l_E$

Piirame teede pikkust mida kasutati $MOP_\circ$ arvutamiseks, iga $n \geq 0$:
$MOP^n_\circ(\ell) = \bigsqcup\{f_{\vec{\ell}}(\iota)|\vec{\ell} \in path_\circ(\ell), |\vec{\ell}| < n\}$

Nüüd $MOP_\circ(\ell) = \bigsqcup_n MOP^n_\circ(\ell)$ ja selles, et tõestada $MFP_\circ \sqsupseteq MOP_\circ$ piisab kui tõestada
$\forall n : MFP_\circ \sqsupseteq MOP^n_\circ$

Juht $MFP_\circ \sqsupseteq MOP^0_\circ$ on triviaalne.
$$\begin{aligned}
MFP_\circ(\ell) &= F(MFP_\circ)(\ell) \\
&= (\bigsqcup\{f_{\ell'}(MFP_\circ(\ell'))|(\ell', \ell) \in F\}) \sqcup \iota^\ell_E \\
&\sqsupseteq (\bigsqcup\{f_{\ell'}(MOP^n_\circ(\ell'))|(\ell', \ell) \in F\}) \sqcup \iota^\ell_E \\
&= (\bigsqcup\{f_{\ell'}(\bigsqcup\{f_{\vec{\ell}}(\iota)|\vec{\ell} \in path_\circ(\ell'), |\vec{\ell}| < n\})|(\ell', \ell) \in F\}) \sqcup \iota^\ell_E \\
&\sqsupseteq (\bigsqcup\{\bigsqcup\{f_{\ell'}(f_{\vec{\ell}}(\iota))|\vec{\ell} \in path_\circ(\ell'), |\vec{\ell}| < n\}|(\ell', \ell) \in F\}) \sqcup \iota^\ell_E \\
&= (\bigsqcup\{f_{\vec{\ell}}(\iota)|\vec{\ell} \in path_\circ(\ell), 1 \leq |\vec{\ell}| \leq n\}) \sqcup \iota^\ell_E \\
&= MOP^{n+1}_\circ(\ell)
\end{aligned}$$

2.5 Interprocedural Analysis

Andmevoo analüüsid olid intraprotseduraalsed - tegelesid lihtsa keelega kus ei olnud funktsioone ega protseduure. Interprotseduraalne analüüs arvestab ka funktsioone ning protseduure.
Laiendame While keelt lisades top-level deklaratsioonid nii et protseduuridel oleks call-by-value ja call-by-result parameetrid.

Süntaks
Programm P_* on kujul
begin $D_* S_*$ *end*

kus D_* on protseduuride deklaratsioonid kujul
$D ::= proc\ p(val\ x,\ res\ y)\ is^{\ell_n}\ S\ end^{\ell_x} \mid D\ D$

Lausete süntaks on kujul:
$S ::= ... \mid [call\ p(a,z)]^{\ell_c}_{\ell_r}$

Näide
begin proc fib(val z,u, res v) is[1]
if[z<3][2] then [v:=u+1][3]
else ([call fib(z-1,u,v)][4]5;[call fib(z-2,v,v)][6]7)
end[8];
[call fib(x,0,y)][9]10;
end

Voograafikud lausetele

$init([call\ p(a,z)]^{\ell_c}_{\ell_r}) = \ell_c$
$final([call\ p(a,z)]^{\ell_c}_{\ell_r}) = \{\ell_r\}$
$blocks([call\ p(a,z)]^{\ell_c}_{\ell_r}) = \{[call\ p(a,z)]^{\ell_c}_{\ell_r}\}$
$labels([call\ p(a,z)]^{\ell_c}_{\ell_r}) = \{\ell_c, \ell_r\}$
$flow([call\ p(a,z)]^{\ell_c}_{\ell_r}) = \{(\ell_c; \ell_n), (\ell_x; \ell_r)\}$
kui D_*-s leidub proc p(val x, res y) is$^{\ell_n}$ S end$^{\ell_x}$

Voograafikud programmidele

Olgu meil programm P_* kujul
begin $D_*\ S_*$ end
Iga protseduuri (proc p(val x, res y) is$^{\ell_n}$ S end$^{\ell_x}$) jaoks:

$init(p) = \ell_n$
$final(p) = \{\ell_x\}$
$blocks(p) = \{is^{\ell_n}, end^{\ell_x}\} \cup blocks(S)$
$labels(p) = \{\ell_n, \ell_x\} \cup labels(S)$
$flow(p) = \{(\ell_n, init(S))\} \cup flow(S) \cup \{(\ell, \ell_x) | \ell \in final(S)\}$

ja terve programmi P_* jaoks

$init_* = init(S_*)$
$final_* = final(S_*)$
$blocks_* = \bigcup\{blocks(p) | proc\ p(val\ x, res\ y)\ is^{\ell_n}\ S\ end^{\ell_x}\ is\ in\ D_*\}$
$\cup blocks(S_*)$
$labels_* = \bigcup\{labels(p) | proc\ p(val\ x, res\ y)\ is^{\ell_n}\ S\ end^{\ell_x}\ is\ in\ D_*\}$
$\cup labels(S_*)$
$flow_* = \bigcup\{flow(p) | proc\ p(val\ x, res\ y)\ is^{\ell_n}\ S\ end^{\ell_x}\ is\ in\ D_*\}$
$\cup flow(S_*)$

Protseduurisiseste voogude jaoks defineerime
$inter - flow_* = \{(\ell_c, \ell_n, \ell_x, \ell_r) | P_*\ sisaldab\ call\ p(a,z)]^{\ell_c}_{\ell_r}$
$ja\ proc\ p(val\ x, res\ y)\ is^{\ell_n}\ S\ end^{\ell_x}\}$

Päripidiste analüüside jaoks F=flow$_*$ ja E={init$_*$}, IF=inter-flow$_*$
Tagurpidi analüüside jaoks F=flow$_*^R$ ja E=final$_*$, IF=inter-flow$_*^R$

Näiteprogrammi

begin proc fib(val z,u, res v) is^{1}
if[z<3]^{2} then [v:=u+1]^{3}
else ([call fib(z-1,u,v)]$^{4}_{5}$;[call fib(z-2,v,v)]$^{6}_{7}$)
end^{8};
[call fib(x,0,y)]$^{9}_{10}$;
end

jaoks saame

$flow_* =$
$\{(1,2),(2,3),(3,8),(2,4),(4;1),(8;5),(5,6),(6;1),(8;7),(7,8),(9;1),(8;10)\}$
$inter - flow_* = \{(9,1,8,10),(4,1,8,5),(6,1,8,7)\}$
ning $init_* = 9$ ja $final_* = 10$

2.5.1 Struktuurne operatsioonisemantika

Keel peab lubama "local data"protseduurides - vaja vahet teha ühe muutuja esinemistel erinevates kohtades - lõplik hulk asukohti (aadreseid)
$\xi \in$ **Loc** - locations

Keskkond ρ seab muutujad käesolevas skoobis vastavusse nende asukohaga ning ladu ς (store) määrab nende asukohtade väärtused.
$\rho \in$ **Env** = $\mathbf{Var}_* \to$ **Loc** - evironments
$\varsigma \in$ **Store** = **Loc** $\to_{fin}$ **Z** - stores

Varem märgiti $\sigma \in$ **State** = $\mathbf{Var}_* \to$ **Z** konstrueeritakse nüüd $\sigma = \varsigma \circ \rho$
($\sigma = \varsigma(\rho(x))$). Nõuame, et see oleks täielik funktsioon (mitte osaline) ehk
$ran(\rho) \subseteq dom(\varsigma)$ kus
$ran(\rho) = \{\rho(x) | x \in \mathbf{Var}_*\}$
$dom(\varsigma) = \{\xi | \varsigma\ on\ defineeritud\ \xi - is\}$

Teisendused
$\rho \vdash_* \langle S, \varsigma\rangle \to \langle S', \varsigma'\rangle$
$\rho \vdash_* \langle S, \varsigma\rangle \to \varsigma'$

$\rho \vdash_* \langle x := a, \varsigma\rangle \to \varsigma[\rho(x) \mapsto A[\![a]\!](\varsigma \circ \rho)]'$
kui $\varsigma \circ \rho$ on täielik

Protseduuride väljakutsete jaoks:
$\rho \vdash_* \langle [call\ p(a,z)]^{\ell_c}_{\ell_r}, \varsigma\rangle \to$
$\quad \langle bind\ \rho_*[x \mapsto \xi_1, y \mapsto \xi_2]\ in\ S\ then\ z := y, \varsigma[\xi_1 \mapsto A[\![a]\!](\varsigma \circ \rho), \xi_2 \mapsto v]\rangle$
kus $\xi_1, \xi_2 \notin dom(\rho), v \in$ **Z**
ja proc p(val x, res y) is$^{\ell}_n$ S end$^{\ell}_x$ on defineeritud D_*-is

bind konstruktsiooni on meil vaja selleks, et kindlustada staatilised skoobi reeglid ja semantika:

$$\frac{\rho' \vdash_* \langle S, \varsigma\rangle \to \langle S', \varsigma'\rangle}{\rho' \vdash_* \langle bind\ \rho'\ in\ S\ then\ z := y, \varsigma\rangle \to \langle bind\ \rho'\ in\ S'\ then\ z := y, \varsigma'\rangle}$$

$$\frac{\rho' \vdash_* \langle S, \varsigma\rangle \to \varsigma'}{\rho' \vdash_* \langle bind\ \rho'\ in\ S\ then\ z := y, \varsigma\rangle \to \varsigma'[\rho(z) \mapsto \varsigma'(\rho'(y))]}$$

2.5.2 Intraprocedural vs Interprocedural Analysis

Iga protseduuri kutse jaoks on meil kaks teisendusfunktsiooni : f_{ℓ_c} ja f_{ℓ_r}
Iga protseduuri definitsiooni jaoks on meil kaks teisendusfunktsiooni f_{l_n} ja f_{l_x}

Saame võrduste süsteemi
$A_\bullet = f_\ell(A_\circ(\ell))$
$A_\circ = \bigsqcup\{A_\bullet(\ell')|(\ell', \ell) \in For(\ell'; \ell) \in F\} \sqcup \iota_E^l$
$kus\ \iota_E^\ell = \begin{cases} \iota & kui\ \ell \in E \\ \bot & kui\ \ell \notin E \end{cases}$

Valid paths
Defineerime MOP lahenduse ümber arvestades ainult korrektseid teid: MVP (Meet over all Valid Paths). Võttes programmi P_* - "begin D_* S_* end ütleme, et tee on täelik (Complete Path) alustades ℓ_1 kuni ℓ_2 programmis P_* kui selles on protseduuride sisendid ja väljundid vastavuses (protseduur naaseb punkti kust ta välja kutsuti).
$CP_{\ell_1,\ell_2} \to \ell_1$ (igakord kui $\ell_1 = \ell_2$)
$CP_{\ell_1,\ell_3} \to \ell_1, CP_{\ell_2,\ell_3}$ (iga $(\ell_1, \ell_2) \in F$)
$CP_{\ell_c,\ell} \to \ell_c, CP_{\ell_n,\ell_x}, CP_{\ell_r,\ell}$ (iga $(\ell_c, \ell_n, \ell_x, \ell_r) \in IF$)

Näide 2.35

begin proc fib(val z,u, res v) is[1]
if[z<3][2] then [v:=u+1][3]
else ([call fib(z-1,u,v)][4]$_5$;[call fib(z-2,v,v)][6]$_7$)
end[8];
[call fib(x,0,y)][9]$_{10}$;
end

$CP_{9,10} \rightarrow 9, CP_{1,8}, CP_{10,10}$
$CP_{10,10} \rightarrow 10$
$CP_{1,8} \rightarrow 1, CP_{2,8}$
$CP_{2,8} \rightarrow 2, CP_{3,8}$
$CP_{2,8} \rightarrow 2, CP_{4,8}$
$CP_{3,8} \rightarrow 3, CP_{8,8}$
$CP_{8,8} \rightarrow 8$
$CP_{4,8} \rightarrow 4, CP_{1,8}, CP_{5,8}$
$CP_{5,8} \rightarrow 5, CP_{6,8}$
$CP_{6,8} \rightarrow 6, CP_{1,8}, CP_{7,8}$
$CP_{7,8} \rightarrow 7, CP_{8,8}$

Lihtne kontrollida, et tee [9,1,2,4,1,2,3,8,5,6,1,2,3,8,7,8,10] on Valid Path mis on genereeritud $CP_{9,10}$ poolt, samas kui tee [9,1,2,4,1,2,3,8,10] ei ole Valid Path

Tee on "valid path"kui see algab ekstreemsest märgendist ja kõik protseduuride väljumised langevad kokku sisenemistega. Kuid on võimalik, et osadesse protseduuridesse on sisenetud, ent mitte väljutud.
$VP_* \rightarrow VP_{\ell_1,\ell_2}$ (kui $\ell_1 \in E$ *ja* $\ell_2 \in \mathbf{Lab}_*$)
$VP_{\ell_1,\ell_2} \rightarrow \ell_1$ (kui $\ell_1 = \ell_2$)
$VP_{\ell_1,\ell_3} \rightarrow \ell_1, VP_{\ell_2,\ell_3}$ (kui $(\ell_1, \ell_2) \in F$)
$VP_{\ell_c,\ell} \rightarrow \ell_c, CP_{\ell_n,\ell_x}, VP_{\ell_r,\ell}$ (kui $(\ell_c, \ell_n, \ell_x, \ell_r) \in IF$)
$VP_{\ell_c,\ell} \rightarrow \ell_c, VP_{\ell_n,\ell}$ (kui $(\ell_c, \ell_n, \ell_x, \ell_r) \in IF$)

$vpath_\circ(\ell) = \{[\ell_1, ..., \ell_{n-1}] | n \geq 1 \wedge \ell_n = l \wedge [\ell_1, ..., \ell_n]$ *on valid path*$\}$
$vpath_\bullet(l) = \{[\ell_1, ..., \ell_n] | n \geq 1 \wedge \ell_n = l \wedge [\ell_1, ..., \ell_n]$ *on valid path*$\}$

Defineerime MVP lahenduse
$MVP_\circ(\ell) = \bigsqcup\{f_{\vec{\ell}}(\iota) | \vec{\ell} \in vpath_\circ(\ell)\}$
$MVP_\bullet(\ell) = \bigsqcup\{f_{\vec{\ell}}(\iota) | \vec{\ell} \in vpath_\bullet(\ell)\}$

2.5.3 Kontekst

Selleks, et vältida liigseid vigaseid teid - kodeerida valitud teede info andmevoo informatsioonile juurde.

$\delta \in \Delta$ - konteksti informatsioon
Intraprotseduraalne fragment
Kui meil on monotoonse raamistiku instants $(L, \mathcal{F}, F, E, \iota, f.)$ siis nüüd konstrueerime me "ilustatud monotoonse raamistku"(embellished Monotone Framework) instantsi, mis arvestab ka konteksti:
$(\widehat{L}, \widehat{\mathcal{F}}, F, E, \widehat{\iota}, \widehat{f}.)$

* $\widehat{L} = \Delta \to L$;
* teisendusfunktsionid hulgas $\widehat{\mathcal{F}}$ on monotoonsed;
* kõik teisendusfunktsioonid $\widehat{f_\ell}$ on antud kui $\widehat{f_\ell}(\widehat{l})(\delta) = f_\ell(\widehat{l}(\delta))$

Ignoreerides protseduure, saame uue kujuga andmevoovõrdused:
$A_\bullet(\ell) = \widehat{f_\ell}(A_\circ(\ell))$ (kõik märgendid va protseduuride kutsed)
$A_\circ(\ell) = \bigsqcup\{A_\bullet(\ell') | (\ell', \ell) \in F \text{ or } (\ell'; \ell) \in F\} \sqcup \widehat{\iota_E^\ell}$ (kõik märgendid)

Interprotseduraalne fragment Protseduuri "proc p(val x, res y)is$^{\ell_n}$ S end$^{\ell_x}$"jaoks on meil kaks teisendusfunktsiooni:
$\widehat{f_{\ell_n}}, \widehat{f_{\ell_x}} : (\Delta \to L) \to (\Delta \to L)$
Võtame neid kahte kui id funktsioone.

Protseduuride kutsete $(\ell_c, \ell_n, \ell_x, \ell_r) \in IF$ jaoks defineerime kaks teisendusfunktsiooni. Keskendume ainult päripidisele analüüsile kus P_* sisaldab $[call\ p(a, z)]_r^{\ell_c}$ ning samuti ka $proc\ p(val\ x, res\ y)\ is^{\ell_n}\ S\ end^{\ell_x}$

$\widehat{f^1_{\ell_n}} : (\Delta \to L) \to (\Delta \to L)$
ja seda kasutame võrduses:
$A_\bullet(\ell_c) = \widehat{f^1_{\ell_n}}(A_\circ(\ell_c))$ iga $(\ell_c, \ell_n, \ell_x, \ell_r) \in IF$ jaoks.

$\widehat{f^2_{\ell_c, \ell_r}} : (\Delta \to L) \times (\Delta \to L) \to (\Delta \to L)$
ja seda kasutame võrduses:
$A_\bullet(\ell_r) = \widehat{f^2_{\ell_c, \ell_r}}(A_\circ(\ell_c), A_\circ(\ell_r))$ iga $(\ell_c, \ell_n, \ell_x, \ell_r) \in IF$ jaoks.

2.5.4 Call Strings as Context

Peame valima hulga Δ (konteksti informatsioon) ja määrama ekstreemse väärtuse $\widehat{\iota}$. Samuti peame määrama kaks teisendusfunktsiooni. Kaks võimalust:

Call strings of unbounded length (piiramatu pikkusega) Esimese võimalusena lihtsalt kodeerime valitud tee. Põhiline huvi seisneb protseduuride kutsetes, seega salvestame voo kujul $(\ell_c; \ell_n)$. Formaalselt
$\Delta = \mathbf{Lab}^*$
$\widehat{\iota}(\delta) = \begin{cases} \iota & kui\ \delta = \Lambda \\ \bot & muudel\ juhtudel \end{cases}$
kus Λ on tühi järjestus märkimaks asjaolu, et kui programm alustab täitmist siis ei ole pooleliolevaid protseduuride kutseid.

Protseduuri kutse $(\ell_c, \ell_n, \ell_x, \ell_r) \in IF$ jaoks defineerime teisendusfunktsiooni $\widehat{f^1_{\ell_c}}(\widehat{l})([\delta, \ell]) = f^1_{l_c}(\widehat{l}(\delta))$ kus $[\delta, \ell]$ tähistab teed mis saadi ℓ_c lisamisel δ-le

$$\widehat{f^1_{\ell_c}}(\widehat{l})(\delta') = \begin{cases} f^1_{\ell_c}(\widehat{l}(\delta)) & kui\ \delta' = [\delta, \ell_c] \\ \bot & muudel\, juhtudel \end{cases}$$

Defineerime teisendusfunktsiooni ka protseduurist naasemise jaoks:
$\widehat{f^2_{\ell_c,\ell_r}}(\widehat{l}, \widehat{l'})(\delta) = f^2_{\ell_c,\ell_r}(\widehat{l}(\delta), \widehat{l'}([\delta, \ell_c]))$

Näide Fibonacci programmi juures pakuvad meile huvi sellised call stringid
Λ,[9],[9,4],[9,6],[9,4,4],[9,4,6],[9,6,4],[9,6,6],...
vastavalt sellele kui pooleli on 0,1,2,3,... protseduuri kutset.

Call strings of bounded length (piiratud pikkusega) Ilmselgelt võivad call string ahelad saavutada suvalise pikkuse, sest protseduuride kutsed võivad olla rekursiivsed. Piirame ära, salvestades ainult k viimast kutset:
$\Delta = \mathbf{Lab}^{\leq k}$

$$\widehat{\iota}(\delta) = \begin{cases} \iota & kui\ \delta = \Lambda \\ \bot & muudel\ juhtudel \end{cases}$$

Juhul kui k=0 on meil $\Delta = \{\Lambda\}$ ehk meil ei ole konteksti informatsiooni.

Näide

begin proc fib(val z,u, res v) is[1]
if[z<3][2] then [v:=u+1][3]
else ([call fib(z-1,u,v)][4]$_5$;[call fib(z-2,v,v)][6]$_7$)
end[8];
[call fib(x,0,y)][9]$_{10}$;
end

kui k=1, siis saame me huvipakkuvad call string'id sellised:
$\Lambda, [9], [4], [6]$

kui k=2, siis saame
$\Lambda, [9], [9,4], [9,6], [4,4], [4,6], [6,4].[6,6]$

Defineerime nüüd teisendusfunktsioonid:
$\widehat{f^1_{\ell_c}}(\widehat{l})(\delta') = \bigsqcup\{f^1_{\ell_c}(\widehat{l}(\delta))|\delta' = \lceil\delta, \ell_c\rceil_k\}$
$\widehat{f^2_{\ell_c,\ell_r}}(\widehat{l},\widehat{l'}) = f^2_{\ell_c,\ell_r}(\widehat{l}(\delta), \widehat{l'}(\lceil\delta, \ell_c\rceil_k))$