Programmide teisendamine

Burstall ja Darlington

Jaak Pruulmann

jjpp@meso.ee

Motivatsioon

"Head programmid" on:

- lühikesed
- selged ja arusaadavad
- kergesti muudetavad
- suhteliselt ebaefektiivsed

Küsimused:

- kas ja kuidas saaks olemasoleva programmi teisendada efektiivsemale kujule?
- kuidas seda teha süstemaatiliselt, automaatselt?

Näide 1: Skalaarkorrutis

Olgu meil skalaarkorrutis $\cdot$, definitisiooniga

$$x \cdot y = \sum_{i=n}^{n} x_i y_i.$$

Tahame arvutada $a \cdot b + c \cdot d$.
Rekursiivselt, meil on:

$$dot(x, y, n) \Leftarrow \textbf{ if } n = 0 \textbf{ then } 0 \textbf{ else } dot(x, y, n-1) + x[n]y[n] \textbf{ fi}.$$

Tahame:

$$f(a, b, c, d, n) \Leftarrow dot(a, b, n) + dot(c, d, n)$$

Näide 1: Skalaarkorrutis (2)

$$
\begin{aligned}
f(a,b,c,d,n) \quad \Leftarrow \quad & \textbf{if } n = 0 \textbf{ then } 0 \textbf{ else} \\
& dot(a,b,n-1) + a[n]b[n] \textbf{ fi} \\
& + \textbf{if } n = 0 \textbf{ then } 0 \textbf{ else} \\
& dot(c,d,n-1) + c[n]d[n] \textbf{ fi} \\
\Leftarrow \quad & \textbf{if } n = 0 \textbf{ then } 0 + 0 \textbf{ else} \\
& (dot(a,b,n-1) + a[n]b[n]) \\
& +(dot(c,d,n-1) + c[n]d[n]) \textbf{ fi} \\
\Leftarrow \quad & \textbf{if } n = 0 \textbf{ then } 0 \textbf{ else} \\
& (dot(a,b,n-1) + (dot(c,d,n-1)) \\
& +a[n]b[n] + c[n]d[n] \textbf{ fi} \\
\Leftarrow \quad & \textbf{if } n = 0 \textbf{ then } 0 \textbf{ else} \\
& f(a,b,c,d,n-1) + a[n]b[n] + c[n]d[n] \textbf{ fi}
\end{aligned}
$$

Teisendusreeglid: tähistus

Kasutame pisut teistsugust kuju:

$$dot(x, y, n) \Leftarrow \textbf{if } n = 0 \textbf{ then } 0 \textbf{ else } dot(x, y, n-1) + x[n]y[n] \textbf{ fi}.$$

asemel

$$\begin{aligned} dot(x, y, 0) &\Leftarrow 0 \\ dot(x, y, n+1) &\Leftarrow dot(x, y, n) + x[n+1]y[n+1] \end{aligned}$$

jne. Fibonacci arve arvutav funktsioon f oleks näiteks:

$$f(0) \Leftarrow 1, f(1) \Leftarrow 1, f(x+2) \Leftarrow f(x+1) + f(x)$$

Teisendusreeglid: definitsioonid

avaldised võivad lisaks primitiivfunktsioonidele, parameetritele ja rekursiivsetele funktsioonidele sisaldada ka **where**-lauset: $E\ \mathbf{where}\ < u, \ldots, w >= F$ või $E\ \mathbf{where}\ u = F$, kus E ja F on avaldised ja $u, \ldots, w$ on kohalikud muutujad.

vasakpoolsed avaldised on kujul $f(e_1, \ldots, e_n), n \geq 0$, kus e_i võivad sisaldada ainult parameeter-muutujaid või konstruktor-funktsioone (peamiselt: ei või sisaldada where't).

parempoolsed avaldised on tavalised avaldised.

Teisendusreeglid: reeglid (1)

i *Definitsioon* toob sisse uue võrduse, mille vasak pool pole ühegi varem olemas olnud võrduse instants (väärtustus?). Näiteks:
$f(a,b,c,d,n) \Leftarrow dot(a,b,n) + dot(c,d,n)$.

ii *Instansseerimine (instantation)* lisab võrduse, mille vasak pool on asenduse abil saadud mõnest olemasolevast. Näiteks:
$f(a,b,c,d,0) \Leftarrow dot(a,b,0) + dot(c,d,0)$.

Teisendusreeglid: reeglid (2)

iii *Avamine (unfolding).* Kui $E \Leftarrow E'$ ja $F \Leftarrow F'$ on võrdused ja F' sees esineb E instants, siis asendatakse see F'' saamiseks vastava E' instantsiga. Saadud võrdus $F \Leftarrow F''$ lisatakse võrduste hulka. Näiteks:

$$
\begin{aligned}
dot(x, y, n+1) &\Leftarrow dot(x, y, n) + x[n+1]y[n+1](E \Leftarrow E') \\
f(a, b, c, d, n+1) &\Leftarrow dot(a, b, n+1) + dot(c, d, n+1)(F \Leftarrow F') \\
f(a, b, c, d, n+1) &\Leftarrow dot(a, b, n) + a[n+1]b[n+1] \\
&\quad + dot(c, d, n) + c[n+1]d[n+1](F \Leftarrow F'').
\end{aligned}
$$

Teisendusreeglid: reeglid (3)

iv *Taandamine (folding).* Kui $E \Leftarrow E'$ ja $F \Leftarrow F'$ on võrdused ja F' sees esineb E' instants, siis asendatakse see F'' saamiseks vastava E instantsiga. Saadud võrdus $F \Leftarrow F''$ lisatakse võrduste hulka. Näiteks:

$$
\begin{aligned}
f(a,b,c,d,n) &\Leftarrow dot(a,b,n) + dot(c,d,n) \quad (E \Leftarrow E') \\
f(a,b,c,d,n+1) &\Leftarrow dot(a,b,n) + dot(c,d,n) \quad (F \Leftarrow F') \\
&\quad + a[n+1]b[n+1] + c[n+1]d[n+1] \\
f(a,b,c,d,n+1) &\Leftarrow f(a,b,c,d,n) \quad (F \Leftarrow F'') \\
&\quad + a[n+1]b[n+1] + c[n+1]d[n+1]
\end{aligned}
$$

Teisendusreeglid: reeglid (4)

v *Abstraktsioon.* Me võime lisada where-lause, tuletades olemasolevast võrdusest $E \Leftarrow E'$ uue võrduse

$$E \Leftarrow E'[u_1/F_1, \ldots, u_n/F_n] \textbf{ where } < u_1, \ldots, u_n > = < F_1, \ldots, F_n > .$$

vi *Seadused.* Me võime võrdustest tuletada uusi, teisendades nende paremat poolt meie primitiividele kehtivate seaduste abil (assotsiatiivsus, kommutatiivsus jms).

Teisendusreeglid: strateegia (algoritm?)

(a) Lisa tarvilikud definitsioonid

(b) *Instansseeri*

(c) *Ava* iga instansseering.
Igal avamisel

(d) Proovi rakendada *seaduseid* ja *abstraktsiooni*.

(e) Taanda korduvalt.

(Sammud (a) ja (b) vajavad kasutaja sekkumist, (c) ja (d) on täielikult automatiseeritavad)

Näide 2: Fibonacci arvud

1.	$f(0)$	$\Leftarrow 1$	antud
2.	$f(1)$	$\Leftarrow 1$	antud
3.	$f(n)$	$\Leftarrow f(x+1)+f(x)$	antud
4.	$g(x)$	$\Leftarrow < f(x+1), f(x) >$	definitsioon (H!)
5.	$g(0)$	$\Leftarrow < f(1), f(0) >$	instansseerimine
		$\Leftarrow < 1, 1 >$	avamine (1. ja 2.)
6.	$g(x+1)$	$\Leftarrow < f(x+2), f(x+1) >$	4. instansseering
		$\Leftarrow < f(x+1)+f(x), f(x+1)$	avamine (3)
		$\Leftarrow < u+v, u >$ **where**	abstraktsioon
		$< u, v > = < f(x+1), f(x) >$	
		$\Leftarrow < u+v, u >$ **where**	taandamine (4)
		$< u, v > = g(x)$	

Näide 2: Fibonacci arvud (2)

7. $f(x+2) \Leftarrow u+v$ **where** — abstraktsioon (3)
   $< u, v >=< f(x+1), f(x) >$
   $\Leftarrow u+v$ **where** — taandamine (4)
   $< u, v >= g(x)$

Kokku seega:

$$
\begin{aligned}
f(0) &\Leftarrow 1 \\
f(1) &\Leftarrow 1 \\
f(x+2) &\Leftarrow u+v \textbf{ where } < u, v >= g(x) \\
g(0) &\Leftarrow < 1, 1 > \\
g(x+1) &\Leftarrow < u+v, y > \textbf{ where } < u, v >= g(x)
\end{aligned}
$$

Reeglite rakendamise strateegiad

- praegune süsteem lubab põhimõtteliselt vabalt suvalisi reegleid suvalises järjekorras rakendada.
- tähelepanekud:
  i. peaaegu kõik optimeerivad teisendused koosnevad avamisest, lemmade abil ümber kirjutamisest ja taandamisest.
  ii. Assotsiatiivsuse, kommutatiivsuse ja **where**-abstraktsiooni kasutamise võib enamasti edasi lükata kuni vahetult taandamise ette.

"Jõuga taandamine" - seaduste või abstraktsiooni rakendamine vaid siis, kui see aitab taandada.

Reeglite rakendamise strateegiad (2)

Algoritm 1:

1. Tee suvaline avamine või lemma abil ümber kirjutamine. Suvaliselt vali järgmiseks kas samm 1 või 2.
2. Tee suvalisi "jõuga taandamisi," kuni saab.

Suvaliste valikute abil vaadatakse kõik võimalused läbi – kasutades vajadusel tagurdamist.

1. tähelepanek:
   iii juhul, kui on võimalik tagada, et avada ei tule lõputult, on kasulik kõik avamised enne taandamisi teha.

Reeglite rakendamise strateegiad (3)

Algoritm 2:

0. Ava kõik võrdused, kuni pole enam midagi avada.

   Iga optimeerimist vajava instantsi jaoks:

1. Tee kõikvõimalikud avamised
2. Suvaliselt tee kas (rakenda mõnd lemmat ja mine sammule 1) või (mine sammule 3)
3. Tee suvalisi "jõuga taandamisi," kuni saab.

Burstall-Darlingtoni süsteem

- seisuga jaanuar 1976.
- vajab kasutaja sekkumist. Ette tuleb anda:
  - algsele programmile lisatud definitisioonid
  - kasulikud lemmad, avaldised seaduste kohta
  - loend instantsidest, mille süsteem välja arvutama peaks (?)

Edasised arengud

- täitmispuu (-graaf) ja selle optimeerimine
- abifunktsioonide automaatne genereerimine

Iteratiivsele kujule viimine

Võrdused $f_1, \ldots, f_m$ on iteratiivsed, kui iga võrdus

- on kujul $f_i(x_1, \ldots, x_n) \Leftarrow E$ ja E ei sisalda funktsiooni f_i või
- on kujul $f_k(E_1, \ldots, E_n)$ ja $E_1, \ldots, E_n$ ei sisalda funktsiooni f_i või
- on tingimusavaldis, mille harud on ühel eelmistest kujudest

Teisendus pole automaatne, üldine "funktsioon" tuleb käsitsi defineerida. Tegemist on Boyer'i-Moore'i rekursiivsele kujule viimise teisenduste pöördoperatsioonidega.

Üledefineerimine

f(0) <= 0
f(n + 1) <= f(n)

Abstraktsed andmetüübid?

?