Werkexemplaar

Fair nondeterministic merge considered necessary

Maarten Fokkinga, 17 november 1985

Een verzameling kan in een functionele programmeertaal gerepresenteerd worden als een opsomming (lijst) van zijn elementen. We laten zien dat het werken met die representaties veel prettiger gaat als de "eerlijke nondeterministische meng"-operator in de taal aanwezig is. De gevolgen, die de aanwezigheid van die operator heeft op het redeneren over de korrektheid van programma's, worden kort beschouwd.

* * *

1. Inleiding

Het is elders al eerder opgemerkt dat de "fair nondeterministic merge"-operator nodig is om in een functionele taal operating systems te programmeren, zie bijvoorbeeld [Henderson 1982] en [Turner 1984]. Dit is wel begrijpelijk omdat bij stelsels communicerende processen nondeterminisme uit de aard der zaak aanwezig is en dus ook wel in de taal aanwezig moet zijn. Desondanks pogen veel auteurs de invoering

van nondeterminisme in een functionele programmeertaal uit te stellen, omdat het begrip '(semantische) gelijkheid' verloren dreigt te gaan en daarmee één van de voordelen van functionele talen: de mogelijkheid om korrektheidsargumentaties te formuleren in termen van semantische gelijkheid van expressies.

Wij zullen vanuit een heel ander gezichtspunt de noodzaak van de zogenaamde fair nondeterministic merge beargumenteren. Namelijk, die operator is nodig om op een juist abstractie-niveau met (representaties van) verzamelingen te werken in een functionele taal. Met name willen we overspecificatie vermijden (t.a.v. de wiskundige vormgeving van de manipulaties met verzamelingen); voorbeelden daarvan zijn de volgorde der generatoren in een verzamelingsabstractie, de volgorde der operanden in een verzamelingsvereniging en de keuze van een specifieke inductieve definitie uit een stel --wiskundig gezien-- gelijkwaardige definities.

We zullen ook kort ingaan op de gevolgen van de invoering van die nondeterministische operator. Tot mijn grote verrassing lijkt het zo te zijn dat er wél een geschikte "semantische gelijkheid" is te defi-

nieren, die aan alle eisen van "gelijkheid" voldoet en derhalve geschikt is om korrektheidsredeneringen aan op te hangen. Als voorbeeld zullen we de bekende Broek-Ackermann anomalie beschouwen.

In het bovenstaande kan het woord "verzameling" overal vervangen worden door "bag" (i.e. "multi-set"). Een bag is "een verzameling waarin de elementen een multipliciteit" hebben, waarin elementen dus méérmalen in voor kunnen komen. Eenvoudigheids halve zullen we steeds het woord "verzameling" blijven gebruiken.

2. Verzamelingen

Het begrip verzameling wordt in de wiskunde en bij informele algoritmen zeer veel gebruikt, en het verdient dan ook aanbeveling om beweringen met verzamelingen zo natuurlijk mogelijk in functionele programmeertalen te laten verlopen. Het spreekt vanzelf dat een verzameling op verlerlei manieren gerepresenteerd kan worden. Bijvoorbeeld, een verzameling A van getallen kan gerepresenteerd worden als een functie $f: nbr \to bool$ zo dat $f\,n = true$ als $n \in A$ en $f\,n = false$ als $n \notin A$; of als een lijst L

zó dat $L\hat{}n = true/false$ als $n \in A / n \notin A$; of als een lijst L zo dat n in L voorkomt precies wanneer $n \in A$; etcetera. Wij zullen in het vervolg van dit verhaal alleen deze laatste representatie beschouwen; dus een verzameling wordt gerepresenteerd door een of andere opsomming van zijn elementen.

Beschouw nu een of andere verzameling A en de verzameling B van lijsten over A. Voor B kunnen we verlerlei inductieve definities geven; we laten er hier enkele zien.

(1) B is de kleinste verzameling met
   (i) $[] \in B$
   (ii) $a:b \in B$ voor elke $a \in A$ en $b \in B$.

(2) B is de kleinste verzameling met
   $B = \{[]\} \cup \{a:b \mid a \in A,\ b \in B\}$.

We beschouwen (1) en (2) als notationele varianten van elkaar. Bij de volgende definities laten we steeds de zinssnede "B is de kleinste verzameling met" achterwege. Def's (3),(4),(5) lijken erg op (2).

(3) $B = \{[]\} \cup \{a:b \mid b \in B,\ a \in A\}$

(4) $B = \{a:b \mid a \in A,\ b \in B\} \cup \{[\,]\}$

(5) $B = \{a:b \mid b \in A,\ a \in B\} \cup \{[\,]\}$

De volgende definities zien er echt anders uit, maar definieren nog steeds dezelfde verzameling B.

(6) $B = \{[\,]\} \cup \{[a] \mid a \in A\} \cup \{b +\!\!+ b' \mid b, b' \in B\}$

(7) $B = \{[\,]\} \cup \{b +\!\!+ [a] +\!\!+ b' \mid b, b' \in B,\ a \in A\}$

En voor (6) en (7) kunnen we nog de volgorde der summanden en de volgorde der generatoren varieren (zodat we er nog 22 definities bijkrijgen!).

Zij nu rA een of andere representatie van A, dwz een of andere opsomming (lijst) van de elementen van A. We kunnen dan proberen ~~een~~ representaties rB voor B te definieren door in de definities de verzamelingsnotaties te vertalen in de lijstnotaties: dus $\{[\,]\}$ wordt $[[\,]]$ en $\{\ldots \mid a \in A, b \in B\}$ wordt $\{\ldots \mid a \leftarrow rA;\ b \leftarrow rB\}$ en $\cup$ wordt $+\!\!+$. Weliswaar introduceren we dan een specifieke volgorde in de lijsten, maar dat zou niet bezwaarlijk mogen zijn gezien de afspraak dat iedere opsomming (lijst) mag dienen als representatie van een verzameling. Dus we vinden dan de volgende mogelijke definities.

(2') $rB2 = [[\,]] +\!\!+ \{a:b \mid a \leftarrow rA;\ b \leftarrow rB2\}$

(3') $rB3 = [[\,]] +\!\!+ \{a:b \mid b \leftarrow rB3;\ a \leftarrow rA\}$

(4') $rB4 = \{a:b \mid a \leftarrow rA;\ b \leftarrow rB4\} +\!\!+ [[\,]]$

(5') $rB5 = \{a:b \mid b \leftarrow rB5;\ a \leftarrow rA\} +\!\!+ [[\,]]$

(6') $rB6 = [[\,]] +\!\!+ \{[a] \mid a \leftarrow rA\} +\!\!+ \{b +\!\!+ b' \mid b, b' \leftarrow rB_6\}$

(7') $rB7 = [[\,]] +\!\!+ \{b +\!\!+ [a] +\!\!+ b' \mid b, b' \leftarrow rB_7;\ a \leftarrow rA\}$

Etcetera. (Hierbij vatten we de generator $b, b' \leftarrow rB$ op als een afkorting van $b \leftarrow rB;\ b' \leftarrow rB$. Dus $b, b' \leftarrow rB$ moet niet verward worden met $[b, b'] \leftarrow rB$.) De geoefende programmeur zal echter onmiddellijk zien dat de meeste van de bovenstaande definities falen: ~~de~~ Sommige gedefinieerde rBi ~~is~~ zijn géén opsomming van B. Bijvoorbeeld voor $A = \{a_0, a_1\}$ en $rA = [a_0, a_1]$ vinden we dat alléén rB_2 en rB_7 een representatie van B zijn:

$rB_2 = [[\,], [a_0], [a_0, a_0], [a_0, a_0, a_0], \ldots]$

$rB3 =$ een lijst v. alle lijsten met a_0 en a_1 als element

$rB_4 = [\,[a_0, a_0, a_0, \ldots\ldots]$ ---?--- $= \ldots (a_0 \ldots) = \perp$

$rB_5 = [$ ---?--- $= \perp$

$rB_6 = [[\,], [a_0], [a_1], [\,], [a_0], [a_1], [\,], [a_0], [a_1], \ldots]$

$rB_7 = rB2$

De conclusie is duidelijk: de programmeur moet geweldig overspecifieeren en de volgorde der summanden

en generatoren specifiek heizen en ook de vorm van de inductieve definitie van de verzameling die gerepresenteerd moet worden.

De reden van het falen van sommige definities van rBi is tweeërlei. Enerzijds betekent in de verzamelingenleer dat bij de generator "$a \in A$, $b \in B$" álle paren (a, b) een keer aan bod komen, terwijl dat niet zonder meer gegarandeerd is bij de lijstgenerator "$a \leftarrow A$; $b \leftarrow B$". Anderzijds is er een extra moeilijkheid doordat bij een recursieve definitie van rB de generator $b \leftarrow$ rB mogelijk tot een divergente berekening leidt wanneer namelijk gepoogd wordt b te laten varieren over dat deel van rB dat nog niet gedefinieerd is (maar juist met behulp van die generator gedefinieerd wordt).

We zullen nu in de volgende sectie de fair nondeterministic merge introduceren en aan de taal toevoegen. Daarmee hebben we in feite een "union" voor verzamelingen en zijn we in staat een betekenis te definieren ~~en een betekenis~~ voor de komma-samenstelling van generatoren zó dat de oorspronkelijk als wishunde bedoelde definities (2) t/m (6) direct --of desnoods na enige transscriptie-- ook korrekte definities van een representatie voor B

zijn.

Oefening a. Definieer een functie merge zó dat bij representaties rA en rB van A en B, (merge rA rB) een representatie van $A \cup B$ is. Let er op dat A en B eventueel oneindig mogen zijn; ieder element van hen moet "binnen eindige tijd" een keer aan bod komen in (merge rA rB).

b. Ga na dat het gebruik van merge in plaats van ++ in definities (2') t/m (7') nog geen soelaas biedt. (en het gebruik van prod bij de generatoren)

c. Definieer mbv. merge een functie prod zodat (prod rA rB) een representatie van het cartesisch product $A * B$ is (bij gegeven representaties rA en rB van A en B).

3 Fair nondeterministic ~~union~~ merge

We nemen de fair nondeterministic merge in de taal op door de volgende reductie- en strategie-regels geldig te verklaren.

(U1) $(x{:}X) \cup Y \rightarrow x{:}(X \cup Y)$

(U2) $X \cup (y{:}Y) \rightarrow y{:}(X \cup Y)$

(U3) $[] \cup Y \rightarrow Y$

(U4) $X \cup [] \rightarrow X$

(U5) regel (U1) mag niet oneindig vaak toegepast worden als regel (U2) ook toepasbaar is (geworden) (op de betreffende redex/contractum); en net zo voor regel (U2) t.a.v. regel (U1).

De $\cup$-operator en de evaluatie volgens (U1) t/m (U5) is effectief (algoritmisch, mechanisch) realiseerbaar. De interpretator/reductie-machine hoeft slechts afwisselend aan e en e' in $(e \cup e')$ de gelegenheid tot een reductiestap te geven en, zodra daarbij een : of $[]$ op als hoofdoperator van de expressie verschijnt, de betreffende contractie uit te voeren.

Vermoeden Fair merge (union) is niet in de lambda-calculus uit te drukken. (Vergelijk [Barendregt 1981, Sectie 14.4].)

Dus deze merge is met recht een nieuw taalprimitivum te noemen. De operator is nondeterministisch omdat mogelijk (U1) en (U2) op eenzelfde expressie toepasbaar zijn (en toegepast mogen worden) terwijl de resultaten niet (semantisch) gelijk zijn. De operator is fair omdat alle elementen van de summanden een eerlijke kans krijgen: ze komen alle in iedere resultaat ooit voor.

Door de invoering van een nondeterministische operator zijn plotseklaps alle expressies nondeterministisch of wel multiple-valued geworden. Het begrip "semantische gelijkheid (van expressies)" moet nu aangepast worden om de meer-waardigheid korrekt in achting te nemen. We geven daarom de volgende definitie.

Definitie Gelijkheid, verfijning gesloten
De relaties $=$ en $\gg$ (op expressies) worden gedefinieerd als de kleinste relaties zó dat

$e = e'$ als e en e' {geen $\cup$ bevatten en} gelijk zijn in de oude, reeds bekende zin óf $e \gg e'$ en $e' \gg e$;

$e \gg e'$ als voor iedere berekening van e', zeg met resultaat r', er een berekening van e mogelijk is, met resultaat r, zó dat $r = r'$ of $r \gg r'$?

Voorts definieren we: $e = e'$ en $e \gg e'$ voor open expressies e en e' als voor alle gesloten instanties e_0, e_0' van e en e' geldt $e_0 = e_0'$ resp $e_0 \gg e_0'$. □

Bijvoorbeeld, er geldt

$\cup$ is commutatief en associatief,

$(x{:}X) \cup Y \gg x{:}(X \cup Y)$, $[] \cup Y \gg Y$

Als de gelijkheid zoals we die zojuist definieerden niet voldoet aan de "wetten van de gelijkheid", dan is die ~~ge~~ relatie weinig behulpzaam en zelfs misleidend by het redeneren over nondeterministische programma's.

Vermoeden De relatie = voldoet aan "de wetten voor de gelijkheid", met name

- reflexiviteit: $e = e$
- symmetrie: $e = e' \Rightarrow e' = e$
- transitiviteit: $e = e' = e'' \Rightarrow e = e''$
- congruentie: $e = e' \Rightarrow C[e] = C[e']$
  voor willekeurige expressie context $C[.]$
- substitutibiliteit: $e = e' \Rightarrow P(e) \Leftrightarrow P(e')$
  voor willekeurige P gedefd door
  $P ::= e = e' \mid e \gg e' \mid P \wedge P \mid \neg P \mid \exists x.P \mid \forall x.P$

Voorts geldt

- beta-regel: linker- en rechterlid v.e. def voldoen aan =
- merge-regel: redex $\gg$ contractum (voor regels (U1), .., (U4))

□

Als dit vermoeden juist is, kunnen we met = net zo redeneren als we met "gelijkheid" gewend zijn. De enige complicatie is dat er soms $\gg$-stappen gemaakt moeten worden. Voor $\gg$ hebben we het

volgende vermoeden.

Vermoeden De relatie $\gg$ voldoet aan

- reflexiviteit: $e \gg e$
- antisymmetrie: $e \gg e' \gg e \Rightarrow e = e'$
- transitiviteit: $e \gg e' \gg e'' \Rightarrow e \gg e''$
- monotonie: $e \gg e' \Rightarrow C[e] \gg C[e']$

(En voor de interactie tussen $\gg$ en predicaten weet ik zo gauw niets te verzinnen. $e \gg e' \Rightarrow (P(e) \Rightarrow P(e'))$ ~~en~~ $e \gg e' \Rightarrow (P(e') \Rightarrow P(e))$ lijken niet geldig te zijn.)

□

Tenslotte merken we nog op dat de fairness van regel (U5) nog niet in de eigenschappen van = en $\gg$ die we tot nu toe noemden, tot uitdrukking is gebracht. Zij echter r de afbeelding die nondeterministisch iedere verzameling afbeeldt op een opsomming van zijn elementen. Dan geldt

$$r(A) \cup r(B) \ll r(A \cup B)$$

(En als we i als de afbeelding nemen die aan iedere tweetal lijsten L en L' nondeterministisch een faire interleaving van L en L' toevoegt, dan geldt zelfs

$$L \cup L' = i(L, L') \quad .)$$

NB $\{x_i \mid i \in [1..]\} \cup [y] \gg [x_1, x_2, x_3, \ldots, x_n, y, x_{n+1}, \ldots]$
maar niet $\gg [x_1, x_2, x_3, \ldots]$

x in X $\Leftrightarrow x$ in $(X \cup Y)$
$\vee\ y \in Y$

We keren nu terug naar naar de programmering van bewerkingen op verzamelingen. We definieren

-- cartesisch product: $r(A*B) \gg r(A) * r(B)$

$[] * Y = [] $

$(x:X) * Y = \{[x,y] \mid y \leftarrow Y\} \cup (X*Y)$

$= \quad X*Y = \{[x,y] \mid x \leftarrow X;\ y \leftarrow Y\}$

-- notatie voor generatoren:

$p, q \leftarrow E$	staat voor	$[p,q] \leftarrow E*E$
$p \leftarrow E,\ q \leftarrow E'$	staat voor	$[p,q] \leftarrow E*E'$

We hoeven nu slechts in definities (2) t/m (7), en de 22 andere, sommige {}-haakjes in []-haakjes te wijzigen en de $\in$-tekens in $\leftarrow$-tekens. De definities die we aldus verkrijgen zijn alle korrekt: steeds geldt voor de gedefinieerde rB_i dat $rB_i \ll r(B)$, d.w.z. ieder element van de wiskundig gedefinieerde verzameling B komt zeker voor --met eindige index-- in elke --nondeterministisch gedefinieerde-- rB_i.

Het ligt voor de hand om in het streven naar vermijding van overspecificatie nog een stap verder te gaan en afzonderlijke notaties in te voeren voor lijsten, bags (= lijsten waarin de volgorde er niet toe doet) en verzamelingen (= lijsten waarin de volgorde en multipliciteit er niet toe doet). Dat zullen we nu niet verder uitwerken.

~~feite is alleen $\{e, e', \ldots, e''\}$ een nieuwe taalconstructie omdat de andere hierboven al expliciet of als afkorting gedefinieerd zijn.~~

~~□~~

Overigens is een waarschuwing op zijn plaats: in wiskundige formuleringen met behulp van verzamelingen is de lidmaatschapstest nog al eens nodig, en die is helaas niet zonder meer als een totale functie te definieren voor bovengenoemde representatie.

4. Het nondeterminisme nader beschouwd

In deze sectie vragen we ons af of nondeterminisme wel noodzakelijk is, of ~~choice~~ union ook uitgedrukt kan worden in de ~~union~~ "choice", en beschouwen we als voorbeeld de Brock-Ackermann anomalie.

Nondeterminisme noodzakelijk?

Met behulp van $\overset{\cup}{\ast}$ konden we de verzamelingmanipulaties bevredigend programmeren. De vraag komt echter op of nondeterminisme wel noodzakelijk is. Nadere analyse van het falen van definities (2') t/m

(7') doet vermoeden dat de oorzaak is dat er geen "eerlijke" mengjing van lijsten geprogrammeerd kan worden die ook goed gaat voor partiele lijsten, dwz. voor lijsten waarvan de n-de tail $\bot$ is (voor een of andere $n \geq 0$).

Vermoeden

Zij merge zó danig dat (merge X Y) = een opsomming van alle elementen van X en Y, zelfs voor partiele X en Y. Dan is merge nondeterministisch in de volgende zin: er zijn expressies E_1 en E_1', E_2 en E_2' zo dat $E_1 = E_1'$, $E_2 = E_2'$ en toch (merge E_1 E_2) $\neq$ (merge E_1' E_2'). [We veronderstellen dat het effect van merge algoritmisch gerealiseerd wordt; dwz de evaluator moet algoritmisch zijn.] □

Een bewijs hiervan lijkt me niet eenvoudig. Waarschijnlijk moet algemene berekenbaarheidstheorie te hulp worden geroepen, en verloopt de redenering dan als volgt. Per aanname geldt

merge [x] $\bot$ = [x ....

Volgens berekenbaarheidstheorie zijn algoritmische functies monotoon, dus aangezien $\bot \sqsubseteq$ [y] is ook

[x,... = merge [x] $\bot \sqsubseteq$ merge [x] [y]

Derhalve: merge [x] [y] = [x, y ... . Net zo kunnen we dan afleiden: merge [x] [y] = [y, x, ... en

daarmee is het nondeterminisme, en dus de noodzaak ervan, aangetoond.

W Merge uitdrukbaar in choice?

Er zijn ook nog andere nondeterministische operatoren met een fairness-trekje. Met name de "faire" nondeterministische choice is er zo een. De vraag houdt op of we die verschillende operatoren in elkaar kunnen uitdrukken of dat ze alle als primitivum beschouwd moeten worden. Laten we hier ons beperken tot de choice. Beschouwd als primitivum moeten de reductieregels als volgt luiden.

(‖1) $(e \parallel e') \rightarrow e$ voor $e \neq \bot$ {of $e' = \bot$}

(‖2) $(e \parallel e') \rightarrow e'$ voor $e' \neq \bot$ {of $e = \bot$}

De "fairness" betekent hier dat zo mogelijk een non-$\bot$ alternatief wordt gekozen.

Met deze choice-operator zijn guarded choices te modelleren:

if $e_1 \rightarrow e_1'$ ▯ $e_2 \rightarrow e_2'$ fi = (if e_1 then e_1' else Ω) ‖ (if e_2 then e_2' else Ω)

waarbij Ω staat voor (x where x=x), dus $\Omega = \bot$.

Vermoeden

Choice is niet uitdrukbaar in merge, en omgekeerd. □

Met name faalt de volgende definitie

$$x \,[\!]' \, y \;=\; hd\,([x] \cup [y])$$

omdat $(\bot \,[\!]' 0) = hd([\bot] \cup [0]) \gg hd\,[\bot, 0] = \bot$
terwijl $(\bot \,[\!]\, 0) = 0 \not\gg \bot$. Het probleem is om uit een element gelijk $\bot$ een lijst gelijk $\bot$ te vormen.
In SASL kom je ~~daarmee~~ nog een heel eind met behulp van de domeintesten number, logical, character, list en function, (aannemende dat function$(\bot) = \bot \neq$ function$(\lambda x.\bot)$ = true). In Twentel lukt het zeker niet.

De Brock-Ackermann anomalie.

Beschouw de volgende definitie.

$$\begin{aligned} wait\ [\,] &= [\,] \\ wait\ (x:[\,]) &= x:[\,] \\ wait\ (x:y:Z) &= x:y:\ wait\ Z \end{aligned}$$

Er geldt dan voor alle totale Z (niet eindigend in $\bot$) dat (wait Z) = Z. Als nu ook voor partiele Z (dus eindigend in $\bot$) geldt (wait Z) = Z, dan mogen we op grond van de (geldige) congruentie-wet

$$e = e' \;\Rightarrow\; C[e] = C[e']$$

overal (wait Z) en Z door elkaar vervangen onder behoud van gelijkheid. Er geldt echter wait$(z:\bot) = \bot \neq (z:\bot)$, dus de ~~vervanging~~ uitwisseling zal wel niet mogen. Inderdaad, definieer

$$\begin{aligned} Y &= \{0 \mid x \leftarrow X\} \cup \{1 \mid y \leftarrow Y\} \\ Y' &= wait\ (\{0 \mid x \leftarrow X\} \cup \{1 \mid y \leftarrow Y'\}) \end{aligned}$$

dan is voor length $X) = 0$; $Y = \bot = Y'$; voor length$(X) = 1$: $Y = (0:1:1:1\ldots) \neq \bot = Y'$; en voor length$(X) = n \geqslant 2$:

Y = 0: fair merge van $(n-1)$ nullen en oneindig veel enen,
Y' = 0: 0: fair merge van $(n-2)$ nullen en oneindig veel ene

(Overigens toonden Brock en Ackermann met de definitie van Y' aan dat, in de aanwezigheid van $\cup$, er geen functioneel verband is tussen input en output van expressies. Immers, bij ieder tweetal oneindige lijsten X en Y' geldt (wait $(\{0 \mid x \leftarrow X\} \cup \{1 \mid y \leftarrow Y'\})$) = een fair merge van nullen en enen, maar in de context van

$Y' =$ wait $(\{0 \mid x \leftarrow X\} \cup \{1 \mid y \leftarrow Y'\})$ geldt nog steeds dat X en Y' oneindige lijsten zijn, terwijl toch niet iedere fair merge van nullen en enen als resultaat van wait $(\{0 \mid x \leftarrow X\} \cup \{1 \mid y \leftarrow Y'\})$ kan verschijnen.) dat voorkomen van

5. Conclusie

Door de introductie van de fair nondeterministic merge zijn we in staat om manipulaties met verzamelingen zonder overspecificatie functioneel te programmeren. We hebben een aanzet gegeven voor een methode om over nondeterministische programma's te redeneren. Die methode is ook bij andere vormen van nondeterminisme toepasbaar.

Ik heb me bij de definities van $=$ en $\gg$ en $\cup$ en $\square$ laten leiden door wat ik me vaag herinnerde uit de literatuur. Er is vast veel meer bekend over dit onderwerp. Ik houd me aanbevolen voor goede literatuurverwijzingen.

—//—