werkexemplaar.

Een omzetting van LREC- naar L-ITER definities

Maarten Fokkinga, 2 mei 1986.

LREC en L-ITER zijn recursie-schema's ~~die~~ voor functies op lijsten; L-ITER is de iteratieve ofwel tail-recursieve vorm, LREC is de niet-iteratieve lineaire recursieve vorm en beide plegen recursie op de staart van de argumentlijst. Wij geven een omzetting van LREC-definities naar L-ITER-definities, die gelijkwaardige functies oplevert wanneer hun domein tot eindige totale lijsten beperkt wordt.

* * *

Het hierna volgende is geïnspireerd op de uitspraak in de hand-outs bij het college Functionele Programmeertalen, [vd Hoeven 1986, pag RECLST 3]:

> "..., maar mogelijk is [de omzetting van LREC naar L-ITER] in het algemeen ook niet. Het vinden van een alternatieve definitie van double in L-ITER-vorm bijvoorbeeld schept onoverkomelijke problemen."

We zullen zien dat de bedoelde functie double eenvoudig naar L-ITER-vorm is om te zetten.

We zeggen dat een definitie van LREC vorm is als hij de volgende vorm heeft (voor een of andere "startwaarde" a en "successorfunctie" S):

f.1: f [] = a

f.2: f (x:X) = S x (f X)

Als voorbeeld hiervan wordt de volgende definitie gegeven.

db.1: db [] = []

db.2: db (x:X) = [x] ++ db X ++ db X

Voorts zeggen we dat een definitie van L-ITER vorm is als hij de volgende vorm heeft (voor een of andere 'afwerking' h en "successorfunctie" S'):

f'.1: f' r [] = h r

f'.2: f' r (x:X) = f' (S' r x) X

De parameter r bevat, operationeel gesproken, de relevante informatie over de alreeds verwerkte elementen van de oorspronkelijke lijst (~~in~~ uit de hoofdaanroep), en afwerking h ~~destilleerd~~ destilleert hieruit de gewenste eindwaarde. In veel gevallen is r het alreeds gevormde partiele resultaat; bij r worden steeds volgende lijstelementen geaccumuleerd.

Wanneer we S en S' als infix operatoren schrijven dan geldt:

$$f\,[x_1,\dots,x_n] = x_1\ S\ (x_2\ S\ (x_3\ \dots\ (x_n\ S\ a)\dots))$$

$$f'\ a'\ [x_1,\dots,x_n] = h((\dots((a'\ S'\ x_1)\ S'\ x_2)\ \dots\ x_{n-1})\ S'\ x_n)$$

Dus de twee schema's verschillen voornamelijk door de rechts-associatieve behandeling van S onder LREC terwijl S' onder L-ITER links-associtief behandeld wordt.

Het probleem is nu zó danige keuzen voor S' en a' en h' te maken dat geldt:

$$(*) \qquad f\ X = f'\ a'\ X$$

voor alle X. Wij zullen een oplossing geven zodanig dat $(*)$ geldt voor alle eindige totale X, (ofwel voor alle X mits f maar strict is in X). Daartoe defini-eren we

$h.1$: $\quad h\ r = r\ a$

$a'.1$: $\quad a' = \lambda R.R$

$S'.1$: $\quad S'\ r\ y = \lambda R.\ r\ (S\ y\ R)$

<u>Lemma</u> Voor alle r en eindige Y geldt $f'\ r\ Y = r\ (f\ Y)$.

<u>bewijs</u> met inductie naar $\#Y$.

$Y=[]$: $\quad f'\ r\ Y =\{\text{def. } Y\}= f'\ r\ [] =\{f'.1\}= h\ r =\{h.1\}= r\ a$
$\qquad =\{f.1\}= r\ (f\ []) =\{Y\}= r\ (f\ Y)$.

$Y=y:Y'$: $\quad f'\ r\ Y =\{Y\}= f'\ r\ (y:Y') =\{f'.2\}= f'\ (S'\ r\ y)\ Y'$
$\qquad =\{S'\}= f'\ (\lambda R.\ r\ (S\ y\ R))\ Y' =\{\text{ind.hyp.}\}= (\lambda R.\ r\ (S\ y\ R))(f\ Y')$
$\qquad =\{\beta\}= r\ (S\ y\ (f\ Y')) =\{f.2\}= r\ (f\ (y:Y')) =\{Y\}= r\ (f\ Y)$.

QED.

<u>Stelling</u> Zij h, a' en S' gekozen als boven. Dan geldt voor alle eindige Y: $\quad f\ Y = f'\ a'\ Y$.

<u>bewijs</u>

$f\ Y = (\lambda R.R)\ (f\ Y) =\{a'\}= a'\ (f\ Y) =\{\text{lemma}\}= f'\ a'\ Y$.

QED.

In het bijzonder volgt nu de L-ITER definitie voor double:

$db'.1$: $\quad db'\ r\ [] = r\ []$

$db'.2$: $\quad db'\ r\ (x:X) = db'\ (\lambda R.\ r\ ([x]{+}{+}R{+}{+}R))\ X$

want hiervoor geldt nu: $\quad db\ X = db'\ (\lambda R.R)\ X$.

* * *

In het bewijs van het lemma is gebruik gemaakt van de eindigheid van de lijstparameter (doordat het principe van volledige inductie is toegepast). Inderdaad, voor oneindige of niet-totale lijstargumenten Y gaat de stelling niet op. Bijvoorbeeld, kies $S = \lambda x,X.\ x:X$ zodat f de identiteit op lijsten is en goedgedefinieerd is voor oneindige lijsten. De bijhorende f' bouwt weliswaar in de r-parameter de oneindige lijst "identiek" na, maar dit resultaat wordt nooit als functieresultaat opgeleverd. Er geldt voor alle L-ITER gedefinieerde functies f' dat

$$f' r (x_1 : x_2 : \ldots : x_n : \perp) = \perp$$

* * *

De L-ITER definitievorm heeft een voordeel boven de (gelijkwaardige) LREC definitievorm, indien de r-parameter van L-ITER een getal (of waarheidswaarde of, algemener, van grondtype) is. Want dan kan, met eager evaluatie voor de r-parameter, een aanroep van f' in constante ruimte geevalueerd worden, terwijl de evaluatie van f een stapelruimte vergt die lineair aangroeit met de lengte van het lijstargument. (Wanneer de r-parameter lazy wordt geevalueerd*) vergt de opslag van die parameter een ruimte ~~die~~ ~~...~~ in $\Theta(\#Y)$.)

*) of van functie type is, zoals bij bovenstaande omzetting

De LREC vorm heeft echter een voordeel boven de L-ITER-vorm, indien $S \times R = \cdots x \cdots : \cdots R \cdots$. Want dan vindt onder lazy evaluation echte streamprocessing plaats, terwijl dat bij de L-ITER vorm niet het geval is.

Het is dus maar de vraag of de omzetting van db naar db' practisch enige winst oplevert.

Literatuur

vd Hoeven, G.F. , Hand-outs by het college Functionele Programmeertalen, T.H. Twente, april 1986.