mf werkexemplaar

Van recursie naar iteratie

Maarten Fokkinga, 20 mei 1986

We geven verscheidene manieren om enkelvoudig recursieve functie-definities naar iteratieve om te zetten. Hieruit resulteert een "standaard techniek" voor programmeren. Dit verhaal is geinspireerd op --en een navertelling en uitbreiding van-- de hand-outs bij het college Functionele talen, [vd Hoeven 1986].

* * *

We beschouwen inductief gedefinieerde data, d.w.z. de waarden van het volgende (Miranda-)type.

$$T * ::= a \mid C * (T *)$$

(Hierbij is 'a' mnemonisch voor "atom" en C voor "cons" of "constructie van".) We kunnen (T*) interpreteren als "lijsten van *"; dan is a de lege lijst en is C de op-kop-van operator. Definieren we het type Void ::= empty dan kunnen we (T Void) interpreteren als de Natuurlijke getallen; dan is a de nul en is C de successor-functie.

We bekijken nu het prototype van een functie die

inductief over (T E) is gedefinieerd, voor een of ander type E.

$$f\ a = a'$$
$$f\ (C\ x\ Y) = C'\ x\ (f\ Y)$$

De definitie-vorm is enkelvoudig recursief (maar niet iteratief (= tail recursive)). De faculteitsfunktie en het geaccumuleerd product over een lijst zijn volgens dit schema te definieren; en nog vele, vele andere functies ook.

Zij T' (afhankelijk van E, maar dat onderdrukken we in het vervolg) het resultaattype van f, dan moet $a' :: T'$ en $C' :: E \to T' \to T'$.

Merk op dat $f\ (C\ x_1 \ldots (C\ x_n\ a)\ldots) = (C'\ x_1 \ldots (C'\ x_n\ a')\ldots)$ en dat volgens (de herschrijfsemantiek en de definitie van) f het resultaat <u>outside-in</u> ofwel <u>top-down</u> "gevormd" wordt:

$$f\ (C\ x_1 (C\ x_2 \ldots (C\ x_n\ a)..))$$
$$\Rightarrow C'\ x_1\ (f\ (C\ x_2 \ldots (C\ x_n\ a)))$$
$$\Rightarrow C'\ x_1\ (C'\ x_2\ (f\ ---))$$
$$\Rightarrow C'\ x_1\ (C'\ x_2\ (C'\ x_3\ (f\ ---)))$$

etc.

We kunnen ook zeggen dat volgens (de herschrijfsemantiek en de definitie van) f het argument outside-in ofwel top-down "verwerkt" wordt. [Voor een lijst-argument is top-down verwerking "van voor naar achter"; voor een getal argument n is top-down verwerking "naar nul toe".]

We geven nu drie iteratieve (= tail recursieve) ~~van~~ definities voor f; elk van hen heeft z'n eigen voor- en nadelen. Notatie: $Y \preceq X$ betekent $X = (C\ x_1 \dots (C\ x_n\ Y) \dots)$ voor een of andere $x_1, \dots, x_n$.

Versie 1.

$$f_1\ X = g_1\ a'\ a$$

$$\underline{\text{where}}\ \|\ \forall Y: a \preceq Y \preceq X:\ \ g_1\ (f\ Y)\ Y = f\ X$$

$$g_1\ r\ Y = r, \quad Y = X$$

$$\quad = g_1\ (C'\ x\ r)\ (C\ x\ Y), \quad Y \precneqq X \text{ zeg } X = C\ x_1 \cdots (C\ x\ Y).$$

De correctheidsbewering is eenvoudig met inductie naar "de afstand van Y tot X" te bewijzen. Het nadeel van deze iteratieve versie is dat de guards "$Y \precneqq X$ zeg $X = (C\ x_1 \cdots (C\ x\ Y))$" en "$Y \preceq X$" weinig elegant en inefficient zijn. [Hoewel .... als deze iteratieve versie automatisch wordt afgeleid, dan zijn er best efficiente mogelijkheden voor de implementatie van de guards: het systeem heeft immers de representatie van X al in handen ....]. Merk op dat volgens g_1 het resultaat bottom-up / inside-out ~~bepaald~~ gevormd wordt, ~~(g_1 is immers iteratief!)~~: bij gegeven $X = (C\ x_1\ (C\ x_2 \cdots (C\ x_{n-1}\ (C\ x_n\ a)) \dots))$ geldt

$$g_1\ a'\ a$$

$$\Rightarrow g_1\ (C'\ x_n\ a')\quad (C\ x_n\ a)$$

$$\Rightarrow g_1\ (C'\ x_{n-1}\ (C'\ x_n\ a'))\quad (C\ x_{n-1}\ (C\ x_n\ a))$$

etc.

Voorts kunnen we zeggen dat volgens g het argument X bottom-up/inside-out verwerkt wordt. [Bij lijsten dus van achter naar voren! Bij getallen van nul naar 't getal toe.] Volgens f gebeurt dat top-down.

Versie 2. Bij deze versie wordt het argument wederom top-down verwerkt, en dit is dan het voordeel van deze versie boven de vorige.

$$f_2\ X = g_2\ (\lambda v.\ v)\ X$$

$$\underline{\text{where}}\ \|\ \forall x_1, \dots, x_n, Y:\ X = (C\ x_1 \cdots (C\ x_n\ Y) \cdots):$$

$$\|\ g_2\ (\lambda v.\ (C'\ x_1 \cdots (C'\ x_n\ v)))\ Y = f\ X$$

$$g_2\ r\ a = r\ a'$$

$$g_2\ r\ (C\ x\ Y) = g_2\ (\lambda v.\ r\ (C'\ x\ v))\ Y$$

De correctheidsbewering is eenvoudig met inductie naar de opbouw van Y te bewijzen. Het is opmerkelijk dat het resultaat top-down ~~wordt~~ gevormd lijkt te worden, terwijl g_2 tóch iteratief is! De truc is hier dat de uiteindelijke r-parameter inderdaad bottom-up wordt gevormd (de lambda-expressie), en dat in die r-parameter een top-down benadering van het uiteindelijke resultaat staat gerepresenteerd.

Een nadeel van deze versie is de inefficientie die door de functionele r-parameter wordt veroorzaakt: de winst ten opzichte van de oorspronkelijke niet-iteratieve f is nihil en zelfs negatief! Maar soms is C' zodanig dat de functies $(\lambda v.\ (C'\ x_1 \ldots (C'\ x_n\ v)\ldots))$ veel eenvoudiger gerepresenteerd kunnen worden. Bijvoorbeeld, als C' = de vermenigvuldiging van getallen, dan kunnen we die functie representeren door $(C' \ldots (C'(C'\ 1\ x_1)\ x_2) \ldots x_n)$.

Versie 3. We voltooien de representatie zoals zojuist gesuggereerd. We nemen dus aan dat C' commutatief en associatief is, en noodzakelijkerwijze dus E = T' en $C' :: T' \to T' \to T'$. De iteratieve definitie luidt nu:

$$
\begin{array}{l}
f_3\ X = g_3\ a\ X \\
\quad \underline{\text{where}}\ \|\ \forall x_1, \ldots, x_n, Y:\ X = (C\ x_1\ \ldots (C\ x_n\ Y)\ldots): \\
\quad\quad\quad \|\ g_3\ (C'\ x_n \ldots (C'\ x_1\ a)\ldots)\ Y = f\ X \\
\quad g_3\ r\ a = r \\
\quad g_3\ r\ (C\ x\ Y) = g_3\ (C'\ x\ r)\ Y \qquad \|\ \text{of: } g_3\ (C'\ r\ x)\ Y
\end{array}
$$

Van de drie iteratieve versies is dit de "beste"; dat is

op zich niet verwonderlijk omdat de correctheid berust op ~~extra~~ eigenschappen van C' die bij de vorige versies niet nodig zijn.

* * *

Voorbeeld: Faculteitsfunctie

We specialiseren de gegeven schema's voor de faculteitsfunctie. Zonder verdere uitleg sommen we hier de definities op; er valt niets nieuws te melden, hier.

$$
\begin{array}{l}
f\ 0 = 1 \\
f\ (n+1) = (n+1) * f\ n
\end{array}
$$

$$
\begin{array}{l}
f_1\ n = g_1\ 1\ 1 \\
\quad \underline{\text{where}}\ \|\ \forall i: 0 \le i \le n:\ g_1\ (f\ i)\ i = f\ n \\
\quad g_1\ r\ i = r, \quad i = n \\
\quad\quad\quad = g_1\ (i * r)\ (i+1), \quad i < n
\end{array}
$$

$$
\begin{array}{l}
f_2\ n = g_2\ (\lambda v.\ v)\ n \\
\quad \underline{\text{where}}\ \|\ \forall i: 0 \le i \le n:\ g_2\ (\lambda v.\ n * \cdots * (i+1) * v)\ i = f\ n \\
\quad g_2\ r\ 0 = r\ 1 \\
\quad g_2\ r\ (j+1) = g_2\ (\lambda v.\ r\ ((j+1) * v))\ j
\end{array}
$$

```
f3 n = g3 1 n
       where || ∀i: 0≤i≤n: g3 ((i+1)*···*n) i = f n
       g3 r 0 = r
       g3 r (j+1) = g3 ((j+1)*r) j
```

Toepassing: een "standaard techniek"

We formuleren het algemene geval nu voor lijsten en zullen dan zien hoe sommige lijstproblemen aangepakt moeten worden.

```
f [] = a'
f (x:Y) = C' x (f Y)
```

Bij de iteratieve versie f_1 hebben we al opgemerkt dat g_1 het argument bottom-up verwerkt, dat is: inside-out ofwel, bij lijsten, van achter naar voren. Dit kunnen we ook formuleren als van voren naar achter op de omgekeerde lijst. Dus we krijgen nu:

```
f1 X = g1 a' (reverse X)
       where || ∀Y,Z: X = Z++Y: g1 (f Y) (reverse Z) = f X
       g1 r [] = r
       g1 r (y:Z) = g1 (C' y r) Z
```


Stel nu dat een lijstverwerkingsprobleem gegeven is waarbij de lijst van vóór naar achter verwerkt moet worden. Op grond van bovenstaande transformatie van f naar f_1 kunnen we dan als volgt te werk gaan.

1. Los het probleem op met een recursieve functie (da's vaak gemakkelijk) die de argumentlijst van achter naar voren verwerkt:
   ```
   f [] = a'
   f (Y++[y]) = C' (f Y) y
   ```
2. De iteratieve versie (die de lijst van voor naar achter verwerkt!) luidt nu:
   ```
   f1 X = g1 a' X          -- zonder reversal !
          where || ∀Y,Z: X = Y++Z: g1 (f Y) Z = f X
          g1 r [] = r
          g1 r (y:Z) = g1 (C' r y) Z
   ```

In de correctheidsbewering $g_1\ (f\ Y)\ Z = f\ X$ duidt Y het alreeds verwerkte deel van de oorspronkelijke invoerlijst aan en Z het nog te verwerken restant. Omdat steeds Y aan de achterzijde wordt uitgebreid, moet f z'n argument van achter naar voren verwerken!

(Bovenstaande "standaard techniek" geeft een universele beschrijving van wat ikzelf al vele malen op een ad hoc heb gedaan. Had ik bovenstaande als standaard techniek gekend, dan had ik veel programma-ontwikkelingen aanzienlijk kunnen inkorten.)

Literatuur

Van der Hoeven, G.F., Hand-outs bij het college Functionele Talen, T.H. Twente mei 1986.