INLEIDING TOT DE λ-CALCULUS DEEL 3

VRIJE UNIVERSITEIT BRUSSEL

"ISZERO" IN λ-CALCULUS

$\textbf{\textit{iszero}} \equiv \lambda n.((n)\ \lambda x.\textbf{\textit{false}})\textbf{\textit{true}}$

$\lambda x.\textbf{\textit{false}}$: geeft steeds false terug

$((n)\ \lambda x.\textbf{\textit{false}})\textbf{\textit{true}}$: als n een church getal, dan (betekenis church getal) n keer toepassen van de functie $\lambda x.\textbf{\textit{false}}$ op het argument true: geeft steeds false

- maar bij n= c_0 wordt de functie niet toegepast, maar wordt het argument (true) terug gegeven

Dus:

$(\textbf{\textit{iszero}})\textbf{\textit{c}}_0 =_\beta \textbf{\textit{true}}$

$(\textbf{\textit{iszero}})\textbf{\textit{c}}_n =_\beta \textbf{\textit{false}}$ for $n>0$

VUB VRIJE UNIVERSITEIT BRUSSEL

"IS-ZERO" IN λ-CALCULUS

$(\textbf{\textit{iszero}})\textbf{\textit{c}}_0 =_\beta \textbf{\textit{true}}$

Bewijs

$(\lambda n.((n)\ \lambda x.\textbf{\textit{false}})\textbf{\textit{true}})\ \textbf{\textit{c}}_0 =_\beta ((\textbf{\textit{c}}_0)\ \lambda x.\textbf{\textit{false}})\textbf{\textit{true}}$

$\equiv ((\lambda f.\lambda x.x)\ \lambda x.\textbf{\textit{false}})\textbf{\textit{true}}$ (def $\textbf{\textit{c}}_0$)

$=_\beta (\lambda x.x)\ \textbf{\textit{true}}$

$=_\beta \textbf{\textit{true}}$

$(\textbf{\textit{iszero}})\textbf{\textit{c}}_n =_\beta \textbf{\textit{false}}$ for $n>0$

Bewijs

$(\lambda n.((n)\ \lambda x.\textbf{\textit{false}})\textbf{\textit{true}})\ \textbf{\textit{c}}_n =_\beta ((\textbf{\textit{c}}_n)\ \lambda x.\textbf{\textit{false}})\textbf{\textit{true}}$

$=_\beta (\lambda x.\textbf{\textit{false}})^n\textbf{\textit{true}}$ *(lemma "x": $((c_n)\mathrm{f})^m\ \mathrm{y} =_\beta (\mathrm{f})^{n \times m}\ \mathrm{y}$)*

$=_\beta \textbf{\textit{false}}$

"CAR EN CDR"

car $\equiv \lambda a.\lambda d.a$

cdr $\equiv \lambda a.\lambda d.d$

car geeft 1ste argument terug; cdr geeft 2de argument terug

cons $\equiv \lambda a.\lambda d.\lambda z.((z)a)d$

Intuitief: geef een car en geef een cdr en er wordt een λ-expressie terug gegeven, die verwacht 1 argument, λz. *wanneer toegepast, dit* argument zal toepassen op op de beide (eerdere) argumenten car a en cdr d.

((**cons**) A) B $\equiv ((\lambda a.\lambda d.\lambda z.((z)a)d)A)B =_{\beta} \lambda z.((z)A)B$

CONS

$(((\textbf{\textit{cons}})\ A)\ B)\ \textbf{\textit{car}} =_\beta A$

$(((\textbf{\textit{cons}})\ A)\ B)\ \textbf{\textit{cdr}} =_\beta B$

Bewijs

$$\begin{aligned}(((\textbf{\textit{cons}})\ A)\ B)\ \textbf{\textit{car}} &=_\beta (\lambda z.((z)A)B)\ car \\ &=_\beta ((car)A)B \\ &\equiv ((\lambda a.\lambda d.a)A)B \\ &=_\beta (\lambda d.A)B \\ &=_\beta A\end{aligned}$$

$$\begin{aligned}(((\textbf{cons})\ A)\ B)\ \textbf{cdr} &=\beta\ (\lambda z.((z)A)B)\ cxr \\ &=\beta\ ((cdr)A)B \\ &\equiv ((\lambda a.\lambda d.d)A)B \\ &=\beta\ (\lambda d.d)B \\ &=\beta\ B\end{aligned}$$

ANDERE NUMERIEKE FUNCTIES ALS λ-EXPRESSIES

pred(n) = n - 1 als n > 0

Idee : paren maken van de vorm $(\boldsymbol{c}_{n}, \boldsymbol{c}_{n-1})$ of ((cons) $\boldsymbol{c}_{n})$ $\boldsymbol{c}_{n-1}$)

Dan pred van $\boldsymbol{c}_n$ definieren als de cdr van het paar $(\boldsymbol{c}_{n}, \boldsymbol{c}_{n-1})$, *nl*

(((cons) $\boldsymbol{c}_n$) $\boldsymbol{c}_{n-1}$) cdr

Hoe creeren we van $(\boldsymbol{c}_{n}, \boldsymbol{c}_{n-1})$ **->** $(\boldsymbol{c}_{n+1}, \boldsymbol{c}_{n})$

(nextp)((cons) $\boldsymbol{c}_n$) $\boldsymbol{c}_{n-1}$ = ((cons) $\boldsymbol{c}_{n+1}$) $\boldsymbol{c}_n$

nextp ≡ λ*p*.((**cons**)(***succ***)(*p*)***car***) (*p*)***car***

≡ λ*p*. λ*z*. ((z) λ*n*. λ*f* . λ*x* (*f*)((*n*)*f*)*x*)(*p*)**car**)(*p*)**car**

ANDERE NUMERIEKE FUNCTIES ALS λ-EXPRESSIES

Maar hoe creeren we nu die cons cell?

Om $(\boldsymbol{c}_{n,}\ \boldsymbol{c}_{n-1})$ te construeren, volstaat het om, vertrekkend

vanuit $(\boldsymbol{c}_{0,}\ \boldsymbol{c}_{0})$ *,* *n* keer **nextp** toe te passen

Een functie *n* keer toepassen -> n^{de} Church getal

Of $(\boldsymbol{c}_{n,}\ \boldsymbol{c}_{n-1})$ genereren door $((\boldsymbol{c}_n)\mathbf{nextp})((\text{cons})\ \boldsymbol{c}_0)\ \boldsymbol{c}_0$

$\boldsymbol{pred} \equiv \lambda n.(\ ((\text{n})\text{nextp})\ ((\text{cons})\boldsymbol{c}_o)\boldsymbol{c}_o\ )\ \boldsymbol{cdr}$

$\boldsymbol{minus} \equiv \lambda m.\lambda n.((m)\boldsymbol{pred})\ n$

m keer ***pred*** toepassen op *n,*

m.a.w. *m* keer -1 toepassen op *n*, dus *n - m*

RECURSIE

We missen nu nog recursiviteit voor een volwaardige programmeertaal!

Eerst introductie van fixpunten

FIXPUNTEN

Voor een numerieke functie f: $D \rightarrow D$ heet $x \in$ D een *fixpunt* van f als en slechts als $f(x) = x$

Bv. 0 is fixpunt voor *2, nl 0 *2 = 0

Voor een λ-expressie wordt dit:

Definitie

$X \in \Lambda$ is een fixpunt van $F \in \Lambda$ als en slechts als

$$(F)X =_\beta X$$

Stelling

1. Iedere λ-expressie heeft een fixpunt:

   $\forall F \in \Lambda\ \exists X: (F)X =_{\beta} X$

2. Er is een fixpunt combinator

   $\boldsymbol{Y} \equiv \lambda f.(\lambda x.(f)(x)x)\ \lambda x.(f)(x)x$

   waarvoor geldt dat

   $\forall F \in \Lambda : (F)(\boldsymbol{Y})F =_{\beta} (\boldsymbol{Y})\, F$

   m.a.w. deze **Y** laat toe om het fixpunt te berekenen voor een willekeurige F, nl. het fixpunt is (**Y**)F .

FIXPUNTEN

Bewijs deel 1

Definieer $\boldsymbol{W} \equiv \lambda x.(F)(x)x$

en $\boldsymbol{X} \equiv (W)W$

Dan geldt $\boldsymbol{X} \equiv (W)W \equiv (\lambda x.(F)(x)x)W$

$=_\beta (F)(W)W \equiv (F)\boldsymbol{X}$

En dus is $\boldsymbol{X}$ een fixpunt van F

Bewijs deel 2

$(\boldsymbol{Y})F \equiv (\lambda f.(\lambda x.(f)(x)x)\ \lambda x.(f)(x)x)F$

$=_\beta (\lambda x.(F)(x)x)\ \lambda x.(F)(x)x$

$\equiv (W)W$

$\equiv \boldsymbol{X}$

en dus is $(\boldsymbol{Y})F$ een fixpunt van F.

FIXPUNTEN

Gevolg

Beschouw de combinator $F \equiv \lambda f.\mathrm{body}_F$ en zijn fixpunt $X_F \equiv (\boldsymbol{Y})F$

dan is het fixpunt: $X_F =_\beta \{X_F / f\}\ \mathrm{body}_F$

Bewijs

Omdat X_F een fixpunt is van F is $(F)X_F =_\beta X_F$

Nu is $(F)X_F \equiv (\lambda f.\mathrm{body}_F)X_F$

$=_\beta \{X_F / f\}\mathrm{body}_F$ (β regel).

RECURSIVITEIT - VOORBEELD

Voorbeeld recursieve definitie in de wiskunde

n! = 1 als n = 0

n! = n (n-1)! als n >0

```
(define fac (lambda n)
(if  (zero? n) 1)
(* n (fac (pred n)))))
```

$(((\textbf{\textit{if}})\textbf{\textit{Cond}})A)B$

Letterlijk vertaling naar λ-calculus :

$\textbf{\textit{fac}} \equiv \lambda n.(((\textbf{\textit{if}})(\textbf{\textit{iszero}})n)\textbf{\textit{c}}_1)((\textbf{\textit{times}})n)(\textbf{\textit{fac}})\ (\textbf{\textit{pred}})n$

Deze definitie is echter circulair!: ***fac*** is gedefinieerd in termen van ***fac*** zelf! Mag niet!

VUB VRIJE UNIVERSITEIT BRUSSEL

RECURSIVITEIT - VOORBEELD

Oplossing: $\equiv$ vervangen door $=_\beta$

$\boldsymbol{fac} =_\beta \lambda n.(((\boldsymbol{if})(\boldsymbol{iszero})n)\boldsymbol{c_1})((\boldsymbol{times})n)(\boldsymbol{fac})\ (\boldsymbol{pred})n$

We zoeken dus een combinator ***fac*** zodanig

dat bovenstaand geldt

RECURSIVITEIT - VOORBEELD

$\textbf{\textit{fac}} =_\beta \lambda n.(((\textbf{\textit{if}})(\textbf{\textit{iszero}})n)\textbf{\textit{c}}_1)((\textbf{\textit{times}})n)(\textbf{\textit{fac}})\ (\textbf{\textit{pred}})n$

Introductie van een parameter

We kunnen dit herschrijven als volgt:

$\textbf{\textit{fac}} =_\beta (\lambda f.\lambda n.(((\textbf{\textit{if}})(\textbf{\textit{iszero}})n)\textbf{\textit{c}}_1)((\textbf{\textit{times}})n)(\textbf{\textit{f}})\ (\textbf{\textit{pred}})n)\ \textbf{\textit{fac}}$

Stel nu

$$\textbf{\textit{FAC}} \equiv \lambda f.\lambda n.(((\textbf{\textit{if}})(\textbf{\textit{iszero}})n)\textbf{\textit{c}}_1)((\textbf{\textit{times}})n)(\textbf{\textit{f}})\ (\textbf{\textit{pred}})n$$

We hebben dus een combinator gevonden waarvoor geldt:

$$\textbf{\textit{fac}} =_\beta (\textbf{\textit{FAC}})\ \textbf{\textit{fac}}$$

M.a.w. ***fac*** is het fixpunt voor ***FAC*** !!

We kunnen ***fac*** dus definieren als

$$\textbf{\textit{fac}} \equiv (\textbf{\textit{Y}})\ \textbf{\textit{FAC}}$$

RECURSIVITEIT

Samenvatting

$$\boldsymbol{fac} \equiv \lambda n.(((\boldsymbol{if})(\boldsymbol{iszero})n)\boldsymbol{c}_1)((\boldsymbol{times})n)(\boldsymbol{fac})\ (\boldsymbol{pred})n$$

$$\boldsymbol{FAC} \equiv \lambda f.\ \lambda\ n.(((\boldsymbol{if})(\boldsymbol{iszero})n)\boldsymbol{c}_1)((\boldsymbol{times})n)(f)\ (\boldsymbol{pred})n$$

$$\boldsymbol{Y} \equiv \lambda\ f.(\lambda\ x.(f)(x)x)\ \lambda\ x.(f)(x)x$$

$$\boldsymbol{fac} \equiv (\boldsymbol{Y})\ \boldsymbol{FAC}$$

Dit principe is te veralgemenen voor het definiëren van andere recursieve functies:

- Definieer de combinator in kleine letters
- Ga van de combinator in kleine letters naar combinator in grote letters (door introductie van een parameter)
- De combinator in kleine letters is fixpunt voor de combinator in grote letters.

EINDE LAMBDA CALCULUS