



Vrije Universiteit Brussel

Inleiding tot de λ -calculus

Prof. dr. Olga De Troyer



Overzicht

- Inleiding
- λ -expressies
- Currying
- Vrije en gebonden variabelen; combinator
- Substitutie en β -gelijkheid
- λ -expressies en rekenen
- Programmeerconstructies
- Recursiviteit en fixpunten



λ -calculus

- Geïntroduceerd door Alonzo Church en Stephen Kleene in de jaren 1930s om het begrip functie abstract te kunnen bestuderen
- Ook gebruikt om het begrip berekenbaarheid te formaliseren
 - Om na te gaan welke functies al dan niet berekenbaar zijn, of nog, welke (wiskundige) problemen oplosbaar zijn, en welke niet

(Een functie f is berekenbaar desda er bestaat een programma P dat de functie f berekent, m.a.w. P stop voor input a en geeft b als output als f gedefinieerd is voor a en anders stop P nooit).



λ -calculus & Programmeertalen

- Hoewel ontwikkeld vóór het bestaan van computertalen is het formalisme de basis van **functionele programmeertalen** zoals Lisp en Scheme
 - Basisconcept van deze programmeertalen zijn nl functies
- Invloed op programmeertalen:
 - “Call by name” mechanism
 - Parameters worden pas geëvalueerd indien nodig
 - Hogere orde functies
 - functie met functies als parameters of
 - Output is een functie.

1938 Zuse bouwde eerste computer Z1 of de Zuses.

Computers werden pas in de 2^{de} wereldoorlog gebruikt om codes te kraken

Call by name en hogere orde functies in programmeertalen zijn afkomstig van Lambda Calculus



Functies

- Een numerieke functie is op twee manieren te definiëren
 - Extensioneel (verzameling theoretisch)
 - Intentioneel (functievoorschrift)
- Voorbeeld:
 - $f(x) = 2 + x$ functievoorschrift

verzameling theoretisch:

{..., (-1, 1), (0, 2), (1, 3), ... }

extensie



- $f(x) = 2 + x$

→ In λ -achtige notatie:

parameter $\lambda x. (+2)x$ voorschrift

– Te lezen als:

Om de waarde van $f(x)$ te berekenen voor de parameter x :

- Pas de functie $+2$ toe op x : dit resulteert in de functie ' $+2$ '
 $+2(x) = x + 2$

- Pas de functie ' $+2$ ' toe op de parameter x

- Merk op: prefix notatie

– Merk op dat er geen naam nodig is voor de functie.



λ -calculus - basis

- 2 fundamentele bewerkingen:
 - **Abstractie**: het maken van een functievoorschrift
 - **Toepassen** (of aanroepen) van het functievoorschrift op een parameter.



Syntax of λ -calculus

Definitie λ -expressies

Laat V een verzameling variabelen zijn $V = \{x, y, z, \dots\}$

De verzameling van λ -expressies wordt als volgt gedefinieerd:

1. alle variabelen (elementen van V) zijn λ -expressies
2. Als M en N λ -expressies zijn dan is
 $(M)N$ een λ -expressie (toepassing)
3. Als $x \in V$ en M λ -expressie dan is
 $\lambda x.M$ een λ -expressie (abstractie)

M.a.w.:

definitie van functie in λ -calculus: λ formele-parameter.functievoorschrift

Aanroep van een functie in λ -calculus: (functie) actuele-parameter

De verzameling van λ -expressies wordt genoteerd als Λ .



Voorbeelden λ -expressies

$\lambda x.x$

$\lambda x.\lambda y.(y)x$

$(\lambda y.(x)y) \lambda x.(u)x$

$\lambda x. \lambda y.x$

$\lambda f. \lambda x.(f)x$

$\lambda f. \lambda x.(f)(f)x$



Syntax of λ -calculus

Toepassing van rechts naar links:

$(P)(Q)x$ komt overeen met $P(Q(x))$
 (in klassieke functienotatie)

$((P)Q)x$ komt overeen met eerst P toepassen op Q en daarna het resultaat op x



Currying

Maar 1 parameter voor functies? Beperking?

- Functie met meerdere parameters steeds voor te stellen via functies met 1 parameter

F: domein → co-domein

Waarbij co-domein zelf weer functies kan bevatten

m.a.w: $f: A \times B \rightarrow C$ wordt
 $g: A \rightarrow (B \rightarrow C)$ functie
 $g(a) = f_a$ en $f_a(b) = f(a,b)$

Voorbeeld:

plus(2,3) wordt $+_2 = +_2$ en $+_2(3) = 5$



Currying

In het algemeen:

$$A_1 \times \dots \times A_n \rightarrow B \text{ wordt } A_1 \rightarrow (A_2 \rightarrow \dots (A_n \rightarrow B))$$

Deze techniek wordt **currying** genoemd, genoemd naar een Amerikaanse wiskundige Curry.

- **Voordeel:**

- Theorie te beperken tot functies met 1 argument

- **Nadeel:**

- Leesbaarheid

Voorbeeld: $\lambda x. \lambda y. (y)x$

is functie met 2 argumenten die zijn 2de argument toepast op zijn eerste argument of m.a.w. $g(x,f) = f(x)$



Currying in Scheme

(lambda (x y)
(+ x y))

Currying →

(lambda (x)
(lambda (y)
(+ x y)))

((lambda (x y)
(+ x y)) 3 4)

((lambda (x)
lambda (y)
(+ x y))) 3) 4)

geeft bij evaluatie (x gebonden
aan 3):

((lambda (y)
(+ 3 y)) 4)



Binden van variabelen

Voor een λ -expressie van de vorm $\lambda x.M$ zegt men:

- λx bindt x in de λ -expressie M
- Het bereik van de binding is M , d.w.z. dat alle nog niet gebonden voorkomens van x in $\lambda x.M$ gebonden zijn

Voorbeeld:

$$((\lambda x.\lambda z.((x)z)(\lambda x.x)z)\lambda y.y)\lambda x.x$$



Vrije en gebonden variabelen

In een λ -expressie heten alle voorkomen van variabelen die niet gebonden zijn **vrij**

Definitie

Laat M een λ -expressie zijn. Dan wordt de **verzameling $VV(M)$ van vrije variabelen van M** als volgt gedefinieerd:

- Voor alle variabelen $x \in V$: $VV(x) = \{x\}$
- $VV(\lambda x.M) = VV(M) \setminus \{x\}$
- $VV((M)N) = VV(M) \cup VV(N)$

Definitie

Een λ -expressie zonder vrije variabelen heet **gesloten** of een **combinator**. De verzameling van combinatoren wordt genoteerd als Λ_0



Substitutie - intuïtief

- Uitwerking van $(\lambda x.M)P$
 - Intuïtief: door elk voorkomen van de parameter x te vervangen door P

Om dit formeel te definiëren moeten we eerst de substitutie definiëren.



Substitutie - definitie

Definitie

Zij P en M twee λ -expressies, en x een variabele $x \in V$.

De substitutie $\{P/x\}M$ van P voor x in M , is als volgt gedefinieerd:

$$\{P/x\}x = P \quad (\text{S1})$$

$$\{P/x\}y = y \text{ als } y \in V \setminus \{x\} \quad (\text{S2})$$

$$\{P/x\}(F)Q = (\{P/x\}F)\{P/x\}Q \quad (\text{S3})$$

$$\{P/x\} \lambda x. M = \lambda x. M \quad (\text{S4})$$

$$\{P/x\} \lambda y. M = \lambda y. \{P/x\} M \text{ als } y \neq x \text{ en } y \notin VV(P) \quad (\text{S5})$$

$$\begin{aligned} \{P/x\} \lambda y. M &= \lambda z. \{P/x\} \{z/y\} M \\ \text{als } y \neq x \text{ en } z \notin VV(P) \text{ en } z \notin VV(M) &\quad (\text{S6}) \end{aligned}$$



Substitutie - toelichting

Toelichting regel (S6):

$$\{P/x\} \lambda y. M = \lambda z. \{P/x\} \{z/y\} M$$

waarbij y vrije variabele in P

Voorbeeld: $\{\lambda u.(y)u/x\} \lambda y.(\textcolor{blue}{x})y$

P staat voor $\lambda u.(y)u$

M staat voor $(\textcolor{blue}{x})y$

Vóór substitutie: y is vrij in P : $\lambda u.(\textcolor{red}{y})u$

Bij een directe substitutie zou y gebonden worden:

$$\{P/x\} \lambda y. M = \lambda y. (\lambda u.(\textcolor{red}{y})u)y$$

Daarom eerst y vervangen door (bijvoorbeeld) z in M : $\lambda z.(\textcolor{blue}{x})z$

Dan x vervangen door P . Het correct antwoord:

$$\lambda z. (\lambda u. (y)u)z$$



Voorbeeld substitutie

$\{\lambda u.(y)u/x\} \lambda y.(x)y$

$\{P/x\}M$

$P \equiv \lambda u.(y)u$, $M \equiv \lambda y.(x)y$ en $y \in VV(P)$ dus (S6):

$\lambda z.\{\lambda u.(y)u/x\}\{z/y\}(x)y$ dan (S3):

$\lambda z.\{\lambda u.(y)u/x\}(\{z/y\}x)\{z/y\}y$ dan (S2):

$\lambda z.\{\lambda u.(y)u/x\}(x)\{z/y\}y$ dan (S1):

$\lambda z.\{\lambda u.(y)u/x\}(x)z$ dan (S3):

$\lambda z.(\{\lambda u.(y)u/x\}x)\{\lambda u.(y)u/x\}z$ dan (S2):

$\lambda z.(\{\lambda u.(y)u/x\}x)z$ dan (S1):

$\lambda z.(\lambda u.(y)u)z$



β -gelijkheid - definitie

Definitie

De β -gelijkheid relatie $=_\beta \subseteq \Lambda \times \Lambda$ is gedefinieerd als volgt:

1. $(\lambda x.M)P =_\beta \{P/x\}M$ (β)
2. $\lambda x.M =_\beta \lambda z.\{z/x\}M$ als $z \notin VV(M)$ (α)
3. $M =_\beta M$ (reflexief)
4. $M =_\beta N$ dan ook $N =_\beta M$ (symmetrisch)
5. $M =_\beta N$ en $N =_\beta L$ dan ook $M =_\beta L$ (transitief)
6. $M =_\beta M'$ en $P =_\beta P'$ dan $(M)P =_\beta (M')P'$ (congruent)
7. $M =_\beta M'$ dan $\lambda x.M =_\beta \lambda x.M'$ (congruent)

Axioma's 1 en 2 worden de β - en α -axioma's genoemd (historisch was β het 2de axioma)

Het β -axioma geeft de intuïtieve betekenis van het toepassen van een functie op een actuele parameter, nl. formele parameter vervangen door de actuele parameter

Het α -axioma laat toe om een parameter te herbenoemen

Axioma's 3, 4 en 5 zeggen dat $=_\beta$ een *equivalentie* relatie is

Axioma 6 & 7 zeggen dat het niet belangrijk is waar we beginnen met uitwerken, eerst de parameter P (nl. $(M)P =_\beta (M')P'$) of eerst de functie M (nl. $(M)P =_\beta (M')P$)



Rekenen in λ -calculus

- λ -calculus kent geen constanten
 - Toch kan men de natuurlijke getallen voorstellen d.m.v. de λ -calculus
 - We kunnen zelf een volledige programmeertaal maken met:
 - Constanten, rekenkundige functies
 - Boolean, if-then constructies
 - Recursie
 -

*Definitie*

Beschouw λ -expressies F en M . De expressie $(F)^n M$ wordt inductief gedefinieerd als volgt:

$$(F)^0 M \equiv M$$

$$(F)^{1+n} M \equiv (F)(F)^n M$$



Rekenen in λ -calculus

Lemma (“+”)

Beschouw λ -expressies F en M .

Voor alle $n, m \in \mathbb{N}$ geldt dat

$$(F)^{n+m}M \equiv (F)^n(F)^mM$$

Bewijs: per inductie op n

Geval $n = 0$

$$(F)^{0+m}M \equiv (F)^mM$$

$$\text{neem } M' \equiv (F)^mM$$

$$\equiv M' \equiv (F)^0M' \quad (def)$$

$$\equiv (F)^0(F)^mM$$



Rekenen in λ -calculus

Stel lemma geldt voor het **geval $n = k$ (IH)**;
Nu bewijzen voor **het geval $n = k+1$ of $1+k$** :

$$\begin{aligned} (F)^{1+k+m} M &= (F)^{1+(k+m)} M \\ &\equiv (F)(F)^{(k+m)} M \quad (\text{def}) \\ &\equiv (F)(F)^k (F)^m M \quad (IH: \text{inductiehypothese}) \\ &\equiv (F)(F)^k M' \quad (M' \equiv (F)^m M) \\ &\equiv (F)^{1+k} M' \quad (\text{def}) \\ &\equiv (F)^{1+k} (F)^m M \quad (M' \equiv (F)^m M). \end{aligned}$$



Rekenen in λ -calculus

Church getallen

Voorstelling natuurlijke getallen:

- 0 als $\lambda f. \lambda x. x$ → Een functie en een parameter
- 1 als $\lambda f. \lambda x. (f)x$ → Functie 1 keer toegepast op parameter
- 2 als $\lambda f. \lambda x. (f)(f)x$ → Functie 2 keer toegepast op parameter
- enz.

Definitie

De zogenaamde **Church getallen** c_n ($n \in \mathbb{N}$) worden gedefinieerd als volgt:

$$c_n = \lambda f. \lambda x. (f)^n x$$

Functievoorschrift met 2 argumenten; het 1ste argument (een functie) wordt een aantal keer (n) toegepast op het 2^{de} argument (de parameter voor de functie).

Dus de natuurlijke getallen worden voorgesteld door welbepaalde lambda expressies.



Rekenen in λ -calculus

λ -definieerbaar

Definitie

Een numerieke functie $f: \mathbb{N}^p \rightarrow \mathbb{N}$ met $p \in \mathbb{N}$ parameters is **λ -definieerbaar** als er een combinator F bestaat zodat

$$(((F)c_{n1})c_{n2}) \dots c_{np} =_{\beta} c_{f(n1, n2, \dots, np)}$$

m.a.w. er bestaat een combinator waarvan het effect op de Church getallen hetzelfde is als de operatie toegepast op de 'echte' getallen

Voorbeeld: de optelling is λ -definieerbaar als er een lambda expressie (combinator) *plus* bestaat zodat:

$$((\text{plus}) c_{n1}) c_{n2} =_{\beta} c_{+(n1, n2)}$$

$$\text{bv: } ((\text{plus}) c_2) c_3 =_{\beta} c_5$$



Rekenen in λ -calculus “successor”

succ(n) = n+1 is λ -definieerbaar

Bewijs:

Zoek **succ** zodat: $(\text{succ})c_n =_{\beta} c_{n+1}$

We hebben dus nodig:

$$(\text{succ})c_0 \equiv (\text{succ}) \lambda f. \lambda x. x =_{\beta} \lambda f. \lambda x. (f)x \equiv c_1$$

$$(\text{succ})c_1 \equiv (\text{succ}) \lambda f. \lambda x. (f)x =_{\beta} \lambda f. \lambda x. (f)(f)x \equiv c_2$$

In het algemeen moet **(succ)** dus een extra “aanroep” van f toevoegen:

$$(\text{succ})c_n =_{\beta} \lambda f. \lambda x. (\textcolor{red}{f}) \text{ voorschrift}_n$$

Hoe vinden we nu *voorschrift*_n?

≡ Wordt hier gebruikt om het gebruik van een definitie aan te geven



Rekenen in λ -calculus “successor”

Voorschrift_n vinden:

$$(\mathbf{succ})c_n =_{\beta} \lambda f. \lambda x. (\textcolor{red}{f}) \text{ voorschrift}_n$$

Merk op dat voorschrift_n staat voor $(f)^n x$

en $c_n \equiv \lambda f. \lambda x. (\textcolor{yellow}{(f)^n x})$

Dus als we de $\lambda f. \lambda x.$ kunnen wegwerken hebben we het voorschrift:

$$\begin{aligned} ((c_n) f) x &\equiv ((\lambda f. \lambda x. (\textcolor{yellow}{(f)^n x})) f) x \\ &=_{\beta} (\lambda x. (\textcolor{yellow}{(f)^n x})) x \\ &=_{\beta} (\textcolor{green}{f})^n x \\ &\equiv \text{voorschrift}_n \end{aligned}$$

Dus $(\mathbf{succ})c_n =_{\beta} \lambda f. \lambda x. (\textcolor{red}{f}) ((c_n) f) x$ of
 $\mathbf{succ} \equiv \lambda n. \lambda f. \lambda x. (\textcolor{red}{f}) ((\textcolor{blue}{n}) f) x$



Rekenen in λ -calculus “successor”

Dus $\mathbf{succ}(n) = n+1$ is λ -definieerbaar en
 $\mathbf{succ} \equiv \lambda n. \lambda f. \lambda x. (f)((n)f)x$

Check (enkel voor c_0):

$$\begin{aligned} (\mathbf{succ})c_0 &\equiv (\lambda n. \lambda f. \lambda x. (f)((n)f)x) c_0 && \text{(def } c_0\text{)} \\ &\equiv (\lambda n. \lambda f. \lambda x. (f)((n)f)x) \color{green}{\lambda f. \lambda x. x} \\ &= (\color{red}{\lambda n. \lambda f. \lambda x. (f)((n)f)x}) \color{red}{\lambda f. \lambda x. x} \\ &= \beta \lambda f. \lambda x. (f)((\color{red}{\lambda f. \lambda x. x})f)x && (\beta) \\ &= \lambda f. \lambda x. (f)(\color{teal}{\lambda f. \lambda x. x})x \\ &= \lambda f. \lambda x. (f)(\color{green}{\lambda x. x})x \\ &= \beta \lambda f. \lambda x. (f) \color{teal}{x} && (\beta) \\ &\equiv c_1 \end{aligned}$$

Bv examenvraag: bewijs dat $(\mathbf{succ})c_n = c_{n+1}$



Rekenen in λ -calculus Optelling

Is de optelling λ -definieerbaar?

Ja en de combinator is:

$$\text{plus} = \lambda n. \lambda m. \lambda f. \lambda x. ((n)f)((m)f)x$$

Intuitief (gebaseerd op definitie van church getallen):

n keer toepassen van f op

m keer toepassen van f op x

=> in totaal n+m keer toepassen van f op x



Rekenen in λ -calculus

Optelling

$$\text{plus} \equiv \lambda n. \lambda m. \lambda f. \lambda x. ((n)f)((m)f)x$$

Bewijs

$$\begin{aligned} ((\text{plus}) c_n) c_m &\equiv ((\lambda n. \lambda m. \lambda f. \lambda x. ((n)f)((m)f)x) c_n) c_m && (\text{def plus}) \\ &\equiv_{\beta} (\lambda m. \lambda f. \lambda x. ((c_n)f)((m)f)x) c_m && (\beta) \\ &\equiv (\lambda m. \lambda f. \lambda x. ((\lambda f. \lambda x. (f^n x)f)((m)f)x) c_m && (\text{def } c_n) \\ &\equiv (\lambda m. \lambda f. \lambda x. ((\lambda f. \lambda x. (f^n x)f)((m)f)x) c_m && (\beta) \\ &\equiv_{\beta} (\lambda m. \lambda f. \lambda x. (\lambda x. (f^n x)((m)f)x) c_m && (\beta) \\ &\equiv (\lambda m. \lambda f. \lambda x. (\lambda x. (f^n x)((m)f)x) c_m && (\beta) \\ &\equiv (\lambda m. \lambda f. \lambda x. (f^n((m)f)x) c_m && (\beta) \\ &\equiv (\lambda m. \lambda f. \lambda x. (f^n((m)f)x) c_m && (\beta) \\ &\equiv_{\beta} \lambda f. \lambda x. (f^n((c_m)f)x) && (\beta) \\ &\equiv \lambda f. \lambda x. (f^n((\lambda f. \lambda x. (f^m x)f)x) && (\text{def } c_m) \\ &\equiv \lambda f. \lambda x. (f^n((\lambda f. \lambda x. (f^m x)f)x) && (\beta) \\ &\equiv_{\beta} \lambda f. \lambda x. (f^n(\lambda x. (f^m x)x) && (\beta) \\ &\equiv \lambda f. \lambda x. (f^n(\lambda x. (f^m x)x) && (\beta) \\ &\equiv_{\beta} \lambda f. \lambda x. (f^n(f^m x) && (\beta) \\ &\equiv \lambda f. \lambda x. (f^{n+m}x) && (\text{Lemma "+"}) \\ &\equiv C \end{aligned}$$



Rekenen in λ -calculus

Volgend doel: vermenigvuldiging

Eerst lemma

Lemma ("x")

Voor alle $n, m \in \mathbb{N}$ geldt dat

$$\underbrace{((c_n)f)^m}_{{m \text{ keer } h \text{ elkaar}}} y =_{\beta} (f)^{n \times m} y$$

Soort copierfunctie.



Rekenen in λ -calculus

TB: $((c_n)f)^m y =_{\beta} (f)^{nxm} y$

Bewijs per inductie over m

1. Geval m = 0

$$\begin{aligned} ((c_n)f)^0 y &\equiv y && \text{(def)} \\ &\equiv (f)^0 y && \text{(def)} \\ &= (f)^{nx0} y \end{aligned}$$



Rekenen in λ -calculus

Vervolg bewijs $((\mathbf{c}_n)f)^m y =_{\beta} (f)^{n \times m} y$

2. Stel bewezen voor geval $m=k$ (IH),
dan nu bewijs voor het geval $m= k+1$

$$\begin{aligned} ((\mathbf{c}_n)f)^{k+1} y &\equiv ((\mathbf{c}_n)f) ((\mathbf{c}_n)f)^k y \\ &\equiv ((\mathbf{c}_n)f) (f)^{n \times k} y && \text{inductiehypothese} \\ &\equiv ((\lambda f. \lambda x. (f)^n x) f) (f)^{n \times k} y && (\text{def } \mathbf{c}_n) \\ &=_{\beta} (\lambda x. (f)^n x) (f)^{n \times k} y && (\beta) \\ &=_{\beta} (f)^n (f)^{n \times k} y && (\beta) \\ &\equiv (f)^{n+n \times k} y && (\text{lemma "+"}) \\ &= (f)^{n \times (1+k)} y \end{aligned}$$



Rekenen in λ -calculus vermenigvuldiging

Is de vermenigvuldiging λ -definieerbaar?

Ja, en de combinator is

$$\mathbf{times} \equiv \lambda n. \lambda m. \lambda f. (n)(m)f$$

Bewijs

$$((\mathbf{times}) c_n) c_m$$

$$\begin{aligned} &\equiv ((\lambda n. \lambda m. \lambda f. (n)(m)f) \mathbf{c}_n) \mathbf{c}_m && (\text{def times}) \\ &=_{\beta} (\lambda m. \lambda f. (\mathbf{c}_n)(m)f) \mathbf{c}_m && (\beta) \\ &\equiv (\lambda m. \lambda f. (\lambda f. \lambda x. (f)^n x)(m)f) \mathbf{c}_m && (\text{def } c_n) \\ &= (\lambda m. \lambda f. (\lambda f. \lambda x. (f)^n x)(\mathbf{m})f) \mathbf{c}_m \\ &=_{\beta} (\lambda m. \lambda f. \lambda x. ((m)f)^n x) \mathbf{c}_m && (\beta) \\ &= (\lambda m. \lambda f. \lambda x. ((\mathbf{m})f)^n x) \mathbf{c}_m \\ &=_{\beta} \lambda f. \lambda x. ((\mathbf{c}_m)f)^n x && (\beta) \\ &= \lambda f. \lambda x. ((\mathbf{c}_m)f)^n x \\ &=_{\beta} \lambda f. \lambda x. (\mathbf{f})^{mn} x && (\text{Lemma "x"}) \\ &\equiv c_{n \times m} \end{aligned}$$



“Boolean” expressies in λ -calculus

$$\mathbf{true} \equiv \lambda t. \lambda f. t$$

$$\mathbf{false} \equiv \lambda t. \lambda f. f$$

true geeft 1ste argument terug;

false geeft 2de argument terug

$$((\mathbf{true})A)B =_{\beta} A$$

$$((\mathbf{false})A)B =_{\beta} B$$

Bewijs:

$$((\lambda t. \lambda f. t) A) B =_{\beta} (\lambda f. A) B =_{\beta} A$$

$$((\lambda t. \lambda f. f) A) B =_{\beta} (\lambda f. f) B =_{\beta} B$$



“If” statement in λ -calculus

$$\text{if} \equiv \lambda c. \lambda d. \lambda e. ((c)d)e$$

Staat voor: if c then d else e

c (conditie) moet true of false geven

true $\Rightarrow ((c)d)e$ geeft 1ste argument terug, zijnde d

false $\Rightarrow ((c)d)e$ geeft 2de argument, zijnde e

$$(((\text{if})\text{true})A)B =_{\beta} A$$

Bewijs:

$$\begin{aligned} (((\lambda c. \lambda d. \lambda e. ((c)d)e) \text{true})A)B &=_{\beta} ((\lambda d. \lambda e. ((\text{true})d)e)A)B \\ &=_{\beta} (\lambda e. ((\text{true})A)e)B =_{\beta} ((\text{true})A)B =_{\beta} A \end{aligned}$$



“iszero” in λ -calculus

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

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

- $((n) \lambda x. \text{false}) \text{true}$: als n een church getal, dan (betekenis church getal) n keer toepassen van de functie $\lambda x. \text{false}$ 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:

$$\begin{aligned} (\text{iszero})c_0 &=_{\beta} \text{true} \\ (\text{iszero})c_n &=_{\beta} \text{false} \quad \text{for } n > 0 \end{aligned}$$



“is-zero” in λ -calculus

$$(iszero)c_0 =_{\beta} \text{true}$$

Bewijs

$$\begin{aligned} (\lambda n.((n) \lambda x. \text{false}) \text{true}) c_0 &=_{\beta} ((c_0) \lambda x. \text{false}) \text{true} \\ &\equiv ((\lambda f. \lambda x. x) \lambda x. \text{false}) \text{true} && (\text{def } c_0) \\ &=_{\beta} (\lambda x. x) \text{true} \\ &=_{\beta} \text{true} \end{aligned}$$

$$(iszero)c_n =_{\beta} \text{false} \quad \text{for } n > 0$$

Bewijs

$$\begin{aligned} (\lambda n.((n) \lambda x. \text{false}) \text{true}) c_n &=_{\beta} ((c_n) \lambda x. \text{false}) \text{true} \\ &\equiv (\lambda x. \text{false})^n \text{true} && (\text{lemma "x": } ((c_n)f)^m y =_{\beta} (f)^{nxm} y) \\ &=_{\beta} \text{false} \end{aligned}$$



“car en cdr”

$$\mathbf{car} \equiv \lambda a. \lambda d. a$$

$$\mathbf{cdr} \equiv \lambda a. \lambda d. d$$

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

$$\mathbf{cons} \equiv \lambda a. \lambda d. \lambda z. ((z)a)d$$

Intuïtief: geef een car en geef een cdr en er wordt een λ -expressie terug gegeven, die ofwel car ofwel cdr terug geeft

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



“cons”

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

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

Bewijs

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



Andere numerieke functies als λ -expressies

$\text{nextp} \equiv \lambda p. ((\text{cons})(\text{succ})(p)\text{car}) (p)\text{car}$

p als cons

Voorbeeld p als $(1,0)$

$(p)\text{car}$

geeft de car van de cons p , dus 1

$(\text{succ})(p)\text{car}$

geeft de successor van de car van de cons p ,
dus 2

$((\text{cons})(\text{succ})(p)\text{car}) (p)\text{car}$

geeft nieuwe cons met 2 en 1, dus $(2,1)$



Andere numerieke functies als λ -expressies

pred $\equiv \lambda n. ((n)\text{nextp}) ((\text{cons})c_o)c_o) \text{ cdr}$

n keer nextp toepassen op (0,0)

Daarvan de cdr nemen

Vb: pred(3): 3 keer nextp van (0,0) = (3,2)

Dan cdr (3,2) is 2

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

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

$$Y = \lambda f. (\lambda x. (f)(x)x) \lambda x. (f)(x)x$$

waarvoor geldt dat

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

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



Bewijs deel 1

- Definieer $W \equiv \lambda x.(F)(x)x$
en $X \equiv (W)W$
- Dan geldt $X \equiv (W)W \equiv (\lambda x.(F)(x)x)W$
 $\equiv_{\beta} (F)(W)W \equiv (F)X$

En dus is X een fixpunt van F

Bewijs deel 2

- $(Y)F \equiv (\lambda f.(\lambda x.(f)(x)x) \lambda x.(f)(x)x)F$
 $\equiv_{\beta} (\lambda x.(F)(x)x) \lambda x.(F)(x)x$
 $\equiv (W)W$
 $\equiv X$

en dus is $(Y)F$ een fixpunt van F .

**Gevolg**

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

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

Bewijs

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

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

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



Recursiviteit - voorbeeld

Voorbeeld recursieve definitie in de wiskunde

$$\begin{aligned} n! &= 1 && \text{als } n = 0 \\ n! &= n \cdot (n-1)! && \text{als } n > 0 \end{aligned}$$

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

Letterlijk vertaling naar λ -calculus:

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

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



Recursiviteit - voorbeeld

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

$$\begin{aligned} \mathbf{fac} =_{\beta} & \lambda n. (((\mathbf{if})(\mathbf{iszzero})n)\mathbf{c}_1)((\mathbf{times})n)(\mathbf{fac}) \\ & (\mathbf{pred})n \end{aligned}$$

We zoeken dus een combinator **fac** zodanig dat bovenstaand geldt



Recursiviteit - voorbeeld

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

Introductie van
een parameter

We kunnen dit herschrijven als volgt:

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

Neem nu

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

We hebben dus een combinator gevonden waarvoor geldt:

$$\mathbf{fac} =_{\beta} (\mathbf{FAC}) \mathbf{fac}$$

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

We kunnen **fac** dus definieren als

$$\mathbf{fac} = (Y) \mathbf{FAC}$$



Recursiviteit

Samenvatting

$$\text{fac} \equiv \lambda n. (((\text{if})(\text{iszero})n)c_1)((\text{times})n)(\text{fac}) (\text{pred})n$$

$$\text{FAC} \equiv \lambda f. \lambda n. (((\text{if})(\text{iszero})n)c_1)((\text{times})n)(f) (\text{pred})n$$

$$Y \equiv \lambda f. (\lambda x. (f)(x)x) \lambda x. (f)(x)x$$

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

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