deel II: lambda calculus

quicksort in C

```
qsort( a, lo, hi ) int a[], hi, lo;
{ int h, l, p, t;
  if (lo < hi) {
    l = lo;
    h = hi;
    p = a[hi];

    do {
      while ((l < h) && (a[l] <= p))
          l = l+1;
      while ((h > l) && (a[h] >= p))
          h = h-1;
      if (l < h) {
          t = a[l];
```

```
        a[l] = a[h];
        a[h] = t;
      }
    }

    while (l < h);

    t = a[l];
    a[l] = a[hi];
    a[hi] = t;

    qsort( a, lo, l-1 );
    qsort( a, l+1, hi );
  }
}
```

quicksort in Haskell

```
qsort []     = []
qsort (x:xs) =
  qsort elts_lt_x ++ [x] ++ qsort elts_greq_x

  where
    elts_lt_x   = [y | y <- xs, y < x]
    elts_greq_x = [y | y <- xs, y >= x]
```

functioneel programmeren

- geen assignment
- functies als parameters
- meer abstractie

functionele programmeertalen

- Haskell
- ML
- Lisp
- Miranda
- Clean

lambda calculus

- Alonzo Church (jaren 1930)
- fundament van de wiskunde?

  nee: inconsistenties
- beperken tot functies
- basis van functioneel programmeren

abstractie

een definitie van een functie:

$$f : \mathsf{Nat} \to \mathsf{Nat}$$
$$f(x) = \mathsf{square}(x)$$

of:

$$f : \mathsf{Nat} \to \mathsf{Nat}$$
$$f : x \mapsto \mathsf{square}(x)$$

abstractie

een definitie van een functie:

$$f : \mathsf{Nat} \rightarrow \mathsf{Nat}$$
$$f(x) = \mathsf{square}(x)$$

of:

$$f : \mathsf{Nat} \rightarrow \mathsf{Nat}$$
$$f : x \mapsto \mathsf{square}(x)$$

in lambda notatie:

$$\lambda x.\, \mathsf{square}\, x$$

abstractie

de functie die x afbeeldt op M

$$\lambda x.\, M$$

applicatie

het toepassen van een functie op zijn argument:

$$(\lambda x.\,\mathsf{square}\,x)\,5$$

applicatie

het toepassen van een functie op zijn argument:

$$(\lambda x.\, \mathsf{square}\, x)\, \mathsf{5}$$

in het algemeen:

$$F\ M$$

applicatie

het toepassen van een functie op zijn argument:

$$(\lambda x.\, \mathsf{square}\, x)\, \mathsf{5}$$

in het algemeen:

$$F\ M$$

de toepassing niet het resultaat van het toepassen

lambda termen

- variabele x
- constante c
- abstractie $\lambda x.\, M$
- applicatie $F\, M$

haakjes

- $(M\ N\ P)$ in plaats van $((M\ N)\ P)$

  applicatie is associatief naar links

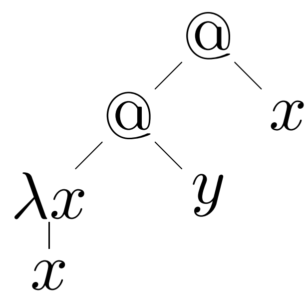
- $(\lambda x.\, \lambda y.\, M)$ in plaats van $(\lambda x.\, (\lambda y.\, M))$

- $(\lambda x.\, M\ N)$ in plaats van $(\lambda x.\, (M\ N))$

- $(M\ \lambda x.\, N)$ in plaats van $(M\ (\lambda x.\, N))$

termen als bomen

de term $(\lambda x.x)\,y\,x$ als boom:



gebonden variabelen

- in de wiskunde: $\int x^2 dx$

- in de logica: $\forall x.\, P(x) \rightarrow P(x)$

gebonden variabelen

- in de wiskunde: $\int x^2 dx$
- in de logica: $\forall x.\, P(x) \rightarrow P(x)$
- in de lambda calculus: $x\,(\lambda x.\, \lambda y.\, x\, y\, z)$

alpha conversie

de naam van een gebonden variabele doet er niet toe

en mag dus veranderd worden

alpha conversie

- $\lambda x.\, x = \lambda y.\, y$
- $\lambda x.\, \mathsf{plus}\, x\, y = \lambda z.\, \mathsf{plus}\, z\, y$
- $\lambda x.\, \mathsf{plus}\, x\, y \neq \lambda y.\, \mathsf{plus}\, y\, y$

Currying

geef de argumenten één voor één

$$\lambda x.\, \lambda y.\, \mathsf{plus}\, x\, y$$

idee: type

$$\mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}$$

in plaats van

$$\mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Nat}$$

beta reductie

voorbeeld:

$$(\lambda x.\, x)\, 3 \rightarrow_\beta 3$$

beta reductie

voorbeeld:

$$(\lambda x.\, x)\, 3 \to_\beta 3$$

beta reductieregel:

$$(\lambda x.\, M)\, N \to_\beta M[x := N]$$

substitutie

voorbeelden:

$$\begin{array}{lcl} (\lambda y.\, x\, y)[x := 3] & = & \lambda y.\, 3\, y \\ (\lambda x.\, x)[x := 3] & = & \lambda x.\, x \end{array}$$

niet goed:

$$(\lambda x.\, y)[y := x] = \lambda x.\, x$$