Postcode en Woonplaats: | Jaar van eerste inschrijving: | Datum: 7 nov. 2003 programmeren | Naam docent:

opgave 1 i

$map :: (a \to b) \to [a] \to [b]$
$ord :: Char \to Int$

$\Rightarrow$ ~~map~~ $(a \to b) \to [a] \to [b] = (a \to b) \to ([a] \to [b])$,
Char is niet van de vorm $a \to b$,
dus $a = Char$, $b = Int$:

1/4

$map\ ord :: [Char] \to [Int]$

$filter :: (g \to Bool) \to [g] \to [g]$

$\Rightarrow$ als we nemen $g = a$, dan hebben we ook $g = b$ en tegelijkertijd $b = Bool$, dus $g = Bool$, maar daarmee leg je restricties en dat willen we niet.
We kunnen wel $a = (g \to Bool)$ nemen $\Rightarrow$
$b = ([g] \to [g])$:

1/4

$map\ filter :: [(g \to Bool)] \to [([g] \to [g])]$

$foldr :: (c \to d \to d) \to d \to [c] \to d$
$(++) :: [e] \to [e] \to [e]$
$[\,] :: [f]$

$\Rightarrow$ we kunnen alleen nemen $c = [e]$, dan geldt dat $d = [e]$ ::

$foldr\ (++) :: [e] \to [[e]] \to [e]$

1/4

nu kunnen we alleen $[f] = [e]$ $\Rightarrow$

$foldr\ (++)\ [\,] :: [[f]] \to [f]$

$even :: Int \to Bool$
$[x * x \mid x \leftarrow [2,4,7]] :: [Int]$

$\Rightarrow$ we kunnen slechts $a = Int$ nemen, dit levert dan geldt:

$\frac{1}{4}$ filter even [x * x | x ← [2, 4, 7]] :: [Int]

map :: (a → b) → [a] → [b]
map :: (p → q) → [p] → [q]

map . map : we passen de ~~tweede~~ rechter map toe
en vervolgens de linker over het
resultaat

de rechter map levert iets van type
(p → q) → [p] → [q] = (p → q) → ([p] → [q])

⇒ er zijn twee mogelijkheden:
- (a → b) = (p → q) → ([p] → [q])
- (a → b) = (p → q)

⇒ tweede is meest algemeen:

map :: (p → q) → [p] → [q]
map . map :: (p → q) → ([[p]] → [[q]])

ii neem h :: a
⇒ (f . g) :: a → c ⇒ g :: a → b
f :: b → c

(f . g) . h :: c
⇒ compl f g h :: c

(g . h) :: b
f . (g . h) :: c ⇒ compr f g h :: c

compl f g h = (f . g) . h = f(g(h)) =

f (g . h) = f . (g . h) = compr f g h

⇒ compl = compr

compl f g :: a → c

Postcode en Woonplaats:	Jaar van eerste inschrijving:	Datum: Naam docent:

```
compl :: (b -> c) -> (a -> b) -> a -> c

compr f g :: a -> c
compr f :: (a -> b) -> a -> c
compr :: (b -> c) -> (a -> b) -> a -> c
```

opgave 4

inductiehypothese:

```
filter p (filter q xs) = filter (\x -> (p x && q x)) xs
```

basisgeval:

```
filter p (filter q []) = filter p []
= [] = filter (\x -> (p x && q x)) []
```

inductiestap: Aan te tonen: ...

• geval q y && p y

```
filter p (filter q (y:xs))
=
filter p (y:(filter q xs))
=
y:(filter p (filter q xs))
= (* inductiehypothese *)
y:(filter (\x -> (p x && q x)) xs)
=
filter (\x -> (p x && q x)) (y:xs)
```

• geval not (q y)

```
filter p (filter q (y:xs))
=
filter p (filter q xs)
= (* inductiehypothese *)
```

=

filter (\x -> p x && q x) (y:xs)

• geval q y && (not (p y))

filter p (filter q (y:xs))
=
filter p ~~(filter~~ (y : (filter q xs)))
=
filter p (filter q xs)
= (* inductiehypothese *)
filter (\x -> p x && q x) xs
=
filter (\x -> p x && q x) (y:xs)

opgave 2 i

```
elements :: Eq a => [(a,Int)] -> [a]
elements [] = []
elements ((p,q) : lst)          = p : elements lst
  | elem p xs = xs
  | otherwise = (p : xs)
      where xs = elements lst
```

[4/4]

Hierin kan p sowieso niet meer voorkomen (vanwege definitie van [illegible] [illegible] van [illegible])

ii

```
multip :: Eq a => a -> [(a,Int)] -> Int
multip x [] = 0
multip x ((p,q) : lst)
  | x == p = q
  | otherwise = multip x lst
```

1/2

iii

```
addmultip :: Eq a => a -> Int -> [(a,Int)] -> [(a,Int)]
addmultip x n [] = [(x,n)]
addmultip x n ((p,q) : lst)
  | x == p = ((p, q+n) : lst)
  | otherwise = ((p,q) : (addmultip x n lst))
```

1/2

1

Opgave 3 i

```
voegtoe :: a -> [a] -> [[a]]
voegtoe x [] = [[x]]
voegtoe x (y:ys) = ((x:(y:ys)) : (map (y:) (voegtoe x ys)))
```

3/4

Postcode en Woonplaats: | Jaar van eerste inschrijving: | Datum: Naam docent:

ii perms :: [a] → [[a]]

perms [] = [[]]

$\frac{1}{2}$ perms (x : xs) = concat (map (voegtoe x) (perms xs))

Opgave 5 i FEM-programma (Γ, A) heeft waarde n als er ten aanzien van Γ herschrijfstappen bestaan van A naar n een herschrijfvrij

$\frac{1}{4}$ +

ii als A te herschrijven is, kunnen we A herschrijven naar $B_1, \ldots, B_m$.

Zijn $B_1, \ldots, B_m$ niet gelijk, dan kunnen we verder herschrijven tot $C_1, \ldots, C_m = C$.

Is C een normaalvorm en gelijk aan waarde n, dan heeft (Γ, A) waarde n.

Is C geen normaalvorm, ~~Da~~ dan zijn er herschrijfstappen vanuit C naar een term D, met D een normaalvorm.

Alle herschrijfrijen zullen in D eindigen.

Het programma heeft dan de waarde van D. ~~[illegible]~~

C en D zijn FEM-termen, en een FEM-term kan hooguit één getal als waarde hebben. Daardoor heeft een FEM-programma (Γ, A) ook hooguit één getal als waarde.

Geef helderdere afleiding uit het gegeven feit

iii a u k i 5

$\frac{1}{4}$ ⇝ k 5 (i 5)

⇝ 5 ?

b (u k i 5) : $\sigma_1 \to \sigma_2 \to N \to N$

u x y z : $\sigma_1 \to \sigma_2 \to N \to N$

Te ver uit de richting

⇒ ~~[illegible]~~ x : N ~~x : σ~~ x : σ_1 y : σ_2

(x z (y z)) : $\sigma_1 \to \sigma_2 \to N \to N$

⇓ (x z) : τ_1 → x : σ_1 ⇒ $\sigma_1 = N \to \tau_1$

((x z) (y z)) : ~~$\sigma_1 \to \sigma_2 \to N \to N$~~ (y z) : τ_2 → $\sigma_2 = N \to \tau_2$

k x y = x i x = x

k : $\sigma_1 \to \sigma_2 \to \sigma_1$ i : $\sigma_1 \to \sigma_1$

Opgave 6 i ~~type TypSter~~

```
data TypSter = Nul | X | Y | S TypSter
             | P TypSter
             | T TypSter TypSter TypSter

parserSter :: String -> TypSter
```

ii

```
eval :: TypSter -> Int -> Int -> Int
eval Nul   x y = 0
eval X     x y = x
eval Y     x y = y
eval (S s) x y = 1 + (eval s x y)
eval (P p) x y
  | w == 0         = 0
  | otherwise      = w-1
     where w = eval p x y
eval T a b c x y
  | w == 0         = eval b x y
  | otherwise      = eval c x y
  where w = eval a x y
```