Programmeerparadigmas

Programmeerparadigmas	
Richting	Informatica
Jaar	Bachelor Informatica Keuzevakken

Bespreking

Het vak programmeerparadigma's heeft net zoals praktisch alle vakken van professor Janssens een open boek examen. Het leermateriaal bestaat uit een reeks slides en deze gaan over de belangrijkste programmeerparadigma's. Extra aandacht gaat uit naar functioneel programmeren en logisch programmeren. Net zoals bij het vak Algoritmen en complexiteit moet je steeds zoveel mogelijk je aandacht te houden bij de lessen en ervoor te zorgen dat je zoveel mogelijk extra informatie bij de slides bijschrijft. Het materiaal dat behandelt wordt is heel wat minder abstract dan bij Algoritmen en complexiteit en de cursus is beter. Vooral het gedeelte over logisch programmeren is goed uitgelegd. Probeer de lambda-calculus goed te begrijpen aangezien de meeste vragen op het examen hierover gaan. Het examen is opgedeeld in een theoretisch examen en een praktijkexamen. Op het theoretisch examen worden enkele inzichtsvragen gesteld over de materie, daarom kan het zijn dat het antwoord je niet meteen te binnen schiet maar dat na enkele keren alle vragen te lezen alles duidelijk wordt. Begin daarom ook niet meteen vraag 1 op te lossen maar lees rustig eerst alles door. Het praktijkexamen is analoog met de theoretische vragen van de oefeningensessies en zijn niet enorm eenvoudig maar ook niet supermoeilijk. Ook hier geldt dat je best eerst alle vragen enkele keren overleest vooraleer te antwoorden. Laat je ook niet van de wijs brengen door de ietwat moeilijke formulering van de vragen want meestal is het antwoord vanzelfsprekend. Besteed ook bij de oefeningen extra aandacht aan functioneel programmeren aangezien het gedeelte over logisch programmeren gemakkelijker is en beter in de cursus staat uitgelegd dan het gedeelte over functioneel programmeren.

Puntenverdeling

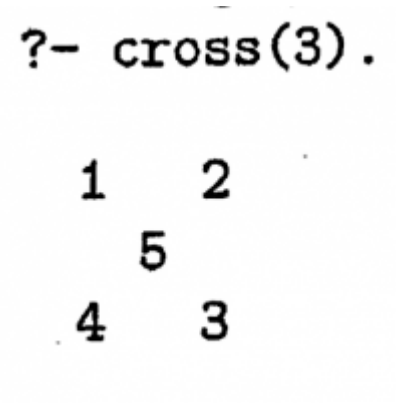
9 punten op theorie, 9 punten op praktijk, 2 permanente evaluatie.

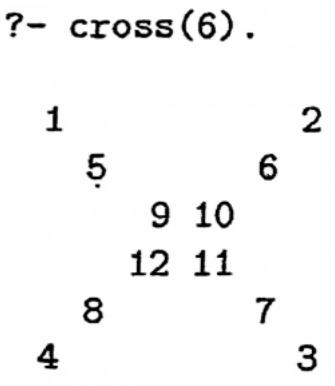
Examenvragen

Academiejaar 2016 - 2017 - 1ste zittijd

1. Are the following equalities valid in the (untyped) lambda-calculus?
    1. `(λz.λy.(y)z)(λx.(x)y)y = λy.(y)(y)y`
    2. `((λu.λy.u)(z)x)λu.z = (λy.(y)z)λy.(y)x`
2. Define rules for normal order evaluation (work out applications from the outer brackets to the inner ones, left to right). Hint: use the syntac to distinguish between (1) a term in normal form (nf), (2) a term in normal that is not an application (nanf), (3) a term that is not an application (na). Hence `nf ::= λx.nf | nanf` etc.
3. Give S, χ, C such that `|- λx : X.λy : Y.λz : Z.(xz)(yz) : S |`$_{\chi}$ `C` and give a solution for the typing of `λx : X.λy : Y.λz : Z.(xz)(yz)` in the lambda calculus with the booleans and natural numbers.
4. Define a **Haskell** function that will calculate the scalar product of two lists of integers. The scalar product of two lists $x_1,...,x_n$ and $y_1,...,y_n$ is defined as:
   $\Sigma\ x_i * y_i$
   You can assume that the lists have the same length. You can use functions from the prelude.
5.
    1. Define a **Haskell** function that merges two sorted lists into a single sorted list. Example: `merge [1,3,5] [2,4,6]` should result in the list `[1,2,3,4,5,6]`
    2. Using your merge-function, define a function that sorts an unsorted list.
6. Consider the following definition of binary trees in **Haskell**: `data Tree a = Leaf a | Node (Tree a) (Tree a)`
   Define a function that verifies whether a binary tree is balanced. We here define a binary tree to be balanced if the height of the left subtree and the height of the right subtree is equal, and if the left subtree is balanced, and if the right subtree is balanced.

7. Write a **Prolog** predicate *cross*, which takes as input a positive integer n, and draws a cross of size n (diagonally) composed of natural numbers counting up in an inward spiral.
   Example:





   You can assume that the argument will not be larger than 49. Therefore the printed numbers are limited to two decimals.
   Hint: Another way to look at the picture is as a set of shrinking rectangles sitting inside each other. This leads directly to an inductive definition of the picture in terms of the number that needs to be drawn in the upper left corner, the number of spaces before that number, and the number of spaces between the two numbers on the first line.

Academiejaar 2014 - 2015 - 2de zittijd

1.
   1. Give the normal form of the following expression:
      `((λx.λy.((zx)y)(zy))w)`
   2. The term `T = ((λa.λb.b(λx.(xx)λx.(xx)))λx.x)` (see "reduction strategies") has both an infinite and a finite reduction.
      Is it true that `T = λx.x`?
2. Is the following term T a valid term in the typed λ-calculus? Explain your answer:
   `T = (λy.(xy)(λz.xw))`

3. Page 6 of "Reduction Machine" - representation of `((λx.λy.(x(yx))2)[E,F])`.
    1. Give the representation of the term reached after one step.
    2. Same question but now starting from the representation of `((λx.λy.(x(yx))(zy))[E,F])`.
4. Write both recursive and non-recursive version of the **Haskell** function tupruns of type `[(Int, Char)] -> [Char]` that gives the following results:
   `Main> tupruns [(3, 'a'), (2: 'b')s (4: 'C')]`→`"aaabbcccc"`
   `Main> tupruns [(1, '#'),(0, '$')]`→`"#"`
   `Main> tupruns (zip [1..8] "abcdefgh")`→`"abbcccddddeeeeeffffffgggggggbhhhhhhh"`
5. Write a **Haskell** function `separate s` that returns a 2-tuple with the digits and non-digits in the string `s` separated, with the initial order preserved.
6. Write a Prolog predicate `diamond/1` which draws a diamond on the screen. The positive integer gives the size of the diamond.

```
?- diamond(2).
  1
 3 2
  4
?- diamond(3).
   1
  4 2
 7 5 3
  8 6
   9
```

```
?- diamond(8).
                1
              9   2
            17  10   3
          25  18  11   4
        33  26  19  12   5
      41  34  27  20  13   6
    49  42  35  28  21  14   7
  57  50  43  36  29  22  15   8
    58  51  44  37  30  23  16
      59  52  45  38  31  24
        60  53  46  39  32
          61  54  47  40
            62  55  48
              63  56
                64
```

Academiejaar 2009 - 2010 - 1ste zittijd

Theorie

1. Geef de normaalvorm van volgende expressies:
    1. (((λx.λy.λz.(yx)(xy))(uz))y)(((λx.λy.λz.(yx)(xy))(uz))y)
    2. (λx.(λy.(x(yy))λz.(x(zz)))(λu.λv.uw))(λx.(λy.(x(yy))λz.(x(zz)))(λu.λv.uw))
2. Een lijst [E1,E2,E3] kan voorgesteld worden als λz.((zE1)λz.((zE2)λz.((zE3)nil)))λz.((zE1)λz.((zE2)λz.((zE3)nil)))
    - Is dit een combinator? Waarom?
    - Indien niet, geef een combinator voor de operatie "vorm een lijst met 3 elementen".
3. Leg uit waarom zelf-applicatie niet mogelijk is in de getypeerde Lambda-calculus

Praktijk

1. Schrijf een programma dat opeenvolgende duplicaten van lijstelementen elimineert.
    1. In prolog: Verwachte werking

        ?- compress([a,a,a,a,b,c,c,a,a,d,e,e,e,e], X).

        X= [a,b,c,a,d,e]

    2. In haskell:

        *Main> compress ['a', 'a,' 'a', 'a', 'b', 'c', 'c', 'a', 'a', 'd', 'e', 'e', 'e']

        "abcade"

2. Schrijf een programma dat gegeven een begin en eindindex een slice uit een lijst kan extraheren
    1. In Prolog

        ?- slice([a,b,c,d,e,f,g,h,i,j,k], 3, 7, L).

        L = [c,d,e,f,g]

    2. In Haskell

        *Main> slice ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k'] 3 7

        "cdefg"

Academiejaar 2009 - 2010 - 2de zittijd

Theorie

1. Welke van volgende vergelijkingen is/zijn correct? Verklaar je antwoord:
    1. ((λu.λv.(vu)λx.(xx))λy.(y(yy)))=(λv.(λu.(vu)λx.(xx))λy.(y(yy)))((λu.λv.(vu)λx.(xx))λy.(y(yy)))=(λv.(λu.(vu)λx.(xx))λy.(y(yy)))
    2. ((λu.λv.(vu)λx.(xx))λy.(y(yy)))=((λu.(λu.uv)λx.(xx))λy.(y(yy)))((λu.λv.(vu)λx.(xx))λy.(y(yy)))=((λu.(λu.uv)λx.(xx))λy.(y(yy)))
2. (zie "Toekenning van typedescriptoren" in lambda calculus). Leg in je eigen woorden uit waarom het nodig is een generiek lijsttype gl in te voeren.
3. Geef enkele redenen waarom de volgorde van de clauses in een prolog programma belangrijk kan zijn, en illustreer dat telkens met een voorbeeld.

Praktijk

1. Schrijf een programma dat kan bepalen of een binairy tree symmetrisch is. Dit betekent dat de subtrees van de root elkaars spiegelbeeld zijn. Hierbij houden we geen rekening met de inhoud van de nodes, maar enkel met de structuur van de boom.
    1. Prolog : Verwachte werking

```
?- symmetric(t(a,t(b,nil,nil),t(c,nil,nil))).
true
```

1.
    1. In haskell:

```
 *main> symmetric(Tree 'a' Empty (Tree 'b' Empty Empty))
 False
```

1. Schrijf een programma dat alle integers binnen een bepaalde interval berekent.
    1. Prolog

```
?- range(4,9,L).
L = [4,5,6,7,8,9]
```

1.
    1. In Haskell zonder gebruik te maken van .. of enumfromto

```
main> range 4 9
[4,5,6,7,8,9]
```

Academiejaar 2006 - 2007 - 1ste zittijd

Theorie

1. Leg uit waarom het gebruik van specifieke reductieregels voor de verwerking van combinators de efficiëntie kan verhogen.
2. Een lijst [E1,E2,E3][E1,E2,E3] kan voor gesteld worden als λz.((zE1)λz.((wE2)λz.((zE3)nil)))λz.((zE1)λz.((wE2)λz.((zE3)nil))). Is dit een combinator? Geef een combinator voor de operatie ’vorm een lijst met 3 elementen’.
3. Geef een voorbeeld van een programma waar de betekenis van ’not’ afwijkt van de logische betekenis. Bewijs met behulp van een proof tree.

Oefeningen

1. Definieer een Haskell functie *hiLo :: [Double] →→ [Double]*. De functie krijgt als parameter een lijst van getallen die metingen zijn van de vochtigheidsgraad te Ukkel op een aantal opeenvolgende dagen. De functie moet zowel de gemiddelde vochtigheidsgraad op vochtige dagen als de gemiddelde vochtigheidsgraad op droge dagen berekenen. Een dag is vochtig als de gemeten vochtigheidsgraad meer dan 50% bedraagt, zo niet is het een droge dag. Je mag veronderstellen dat een library functie *avg :: [Double] -> Double* ter beschikking is om het gemiddelde van een lijst van getallen te berekenen.

2. Beschouw de volgende functie:

   > f = (-> case l of [] -> [] (x:xs) -> g x (f xs) where g = (l -> case l of [] -> [x] (y:ys)-> if (x>y) then (y : (g x ys)) else (x:y:ys))

   1. Wat is het resultaat van f[3,7,1,8,4]f[3,7,1,8,4] ?

   2. Welke taak voert deze functie ff uit (1 woord)? Beschrijf ook kort wat hulpfunctie gg doet.

3. Beschouw volgend Prolog programma:

```
student(student1).
student(student2, cse).
name(student1, [public, john, q]).
first_name(S, FN) :- student(S), name(S,[_, FN, _]).
foo([_, _, H|_], [foo, H]).
bar(X, Y) :- integer(X), Y is X*10.
bar([H|T], [10|T]).
```

   Wat is het resultaat van volgende queries?

```
?- student(X).
?- first_name(S, FN).
?- foo([a, b, c, d, e], X).
?- foo([1, 2, 3], [foo, 3]).
?- foo([a, 1], X).
?- p(X, f(Y)) = f(f(a), X).
?- bar(5, Y).
?- bar([1, 2, 3], Y).
?- bar([], Y).
```

Academiejaar 2007 - 2008 - 1ste zittijd

Theorie

1. Zijn deze vergelijkingen correct? Bewijs je antwoord.
   ((λu.λv.(v u) λx.(x x)) λy.(y (y y)))=(λv. (λu.(v u) λx.(x x)) λy.(y (y y)))((λu.λv.(vu)λx.(xx))λy.(y(yy)))=(λv.(λu.(vu)λx.(xx))λy.(y(yy)))

   ((λu.λv.(v u) λx.(x x)) λy.(y (y y)))=((λu. (λu.u v) λx.(x x)) λy.(y (y y)))((λu.λv.(vu)λx.(xx))λy.(y(yy)))=((λu.(λu.uv)λx.(xx))λy.(y(yy)))

2. Leg in je eigen woorden (niet meer dan 3/4 pagina) uit wat het effect is van de toepassing van de Y-combinator op een lambda-term E.

3. Geef enkele redenen waarom de volgorde van de clauses in een Prolog programma belangrijk kan zijn. Illustreer telkens met een voorbeeld.

Oefeningen

1. Haskell

    1. Wat is het type van volgende expressies? Houd rekening met eventuele class constraints. Indien er geen type kan afgeleid worden, verklaar dan waarom dat het geval is.

        > (a) -> [x] (b) y z -> (x,y:z:[]) (c) -> x + 5 (d) -> "Hello, world" (e) -> x ’a’ (f) -> x x (g) -> x + x

    2. Definieer een datatype Tuple dat, afhankelijk van de constructor, één, twee, drie of vier elementen van willekeurige type kan bevatten (er moeten dus vier constructors zijn, genaamd OneTuple, TwoTuple, ThreeTuple en FourTuple). Definieer bovendien ook vier functies getFirst, getSecond, getThird en getFourth, die ofwel "Just" het respectievelijke element uit een Tuple teruggeven, ofwel "Nothing" (bijvoorbeeld ingeval getFourth zou worden opgeroepen op een TwoTuple).
    Geef van deze functies ook de types.

2. Prolog

    1. Een multiway tree bestaat uit een root node en een (mogelijk lege) set successors, die eveneens multiway trees zijn. Een multiway tree is nooit leeg. In prolog kunnen we een multiway tree voorstellen als een term t(X,F)t(X,F), met XX de root, en FF de lijst van successors.
    Een mogelijk voorbeeldje ziet er dus uit als

        T=t(a,[t(f,[t(g,[])]),t(c,[]),t(b,[t(d,[]),t(e,[])])])
        T=t(a,[t(f,[t(g,[])]),t(c,[]),t(b,[t(d,[]),t(e,[])])])

        1. Schrijf een predicaat istree/1 dat waar is als en slechts als het gegeven argument een multiway tree is.

        2. Schrijf een predicaat nnodes/2 om het aantal nodes in een multiway tree te berekenen.