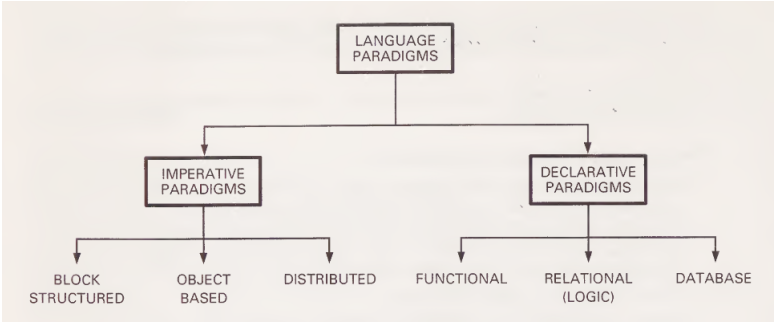
Declaratief Programmeren

Thomas van Binsbergen

Universiteit van Amsterdam

February 28, 2023

Verhoudingen tussen de belangrijkste programmeerparadigma's?



D. Appleby, Programming Languages: Paradigm and Practice. Diagram based on Wegner's classification from 1988.

Blok Declaratief programmeren & Haskell

1. Declaratief programmeren
   - Expressies
   - Algebraïsche data representatie
   - Patronen en patroonherkenning
   - Wiskundige variabelen
   - Clausules en recursie

Declaratief programmeren

Programmeren door wiskundige objecten te beschrijven mbv regels of vergelijkingen

- Wat wordt er zoal beschreven?
  - Functies (functioneel programmeren),
  - Relaties (logisch programming),
  - De structuur van data (database tabellen, webpagina's, JSON objecten, etc.)
  - etc.

Declaratief programmeren

Programmeren door wiskundige objecten te beschrijven mbv regels of vergelijkingen

- Wat wordt er zoal beschreven?
  - Functies (functioneel programmeren),
  - Relaties (logisch programming),
  - De structuur van data (database tabellen, webpagina's, JSON objecten, etc.)
  - etc.
- Declaratieve paradigma's:
  - Functioneel programmeren: functies met algebraïsche data als invoer en uitvoer
  - Logisch programmeren: relaties met axioma's, inferentie regels en te bewijzen stellingen
  - Database talen: tabellen, bevragingen en manipulatie

Declaratief programmeren

Programmeren door wiskundige objecten te beschrijven mbv regels of vergelijkingen

- Wat wordt er zoal beschreven?
  - Functies (functioneel programmeren),
  - Relaties (logisch programming),
  - De structuur van data (database tabellen, webpagina's, JSON objecten, etc.)
  - etc.
- Declaratieve paradigma's:
  - Functioneel programmeren: functies met algebraïsche data als invoer en uitvoer
  - Logisch programmeren: relaties met axioma's, inferentie regels en te bewijzen stellingen
  - Database talen: tabellen, bevragingen en manipulatie
- Variabelen zijn **aanduidingen** gebonden aan waardes *(wiskundige variabelen)*

Declaratief programmeren

Programmeren door wiskundige objecten te beschrijven mbv regels of vergelijkingen

- Wat wordt er zoal beschreven?
  - Functies (functioneel programmeren),
  - Relaties (logisch programming),
  - De structuur van data (database tabellen, webpagina's, JSON objecten, etc.)
  - etc.
- Declaratieve paradigma's:
  - Functioneel programmeren: functies met algebraïsche data als invoer en uitvoer
  - Logisch programmeren: relaties met axioma's, inferentie regels en te bewijzen stellingen
  - Database talen: tabellen, bevragingen en manipulatie
- Variabelen zijn **aanduidingen** gebonden aan waardes *(wiskundige variabelen)*
- **Recursie** ipv iteratie – bv een functie die gedefiniëerd is in termen van zichzelf

Algebraïsche expressies en algebraïsche data representatie

Een *expressie* bestaat uit een waarde, een variabele, of is een toepassing van een *operator* op nul, één of meerdere expressies, en beschrijft zo een berekening die een waarde oplevert.

```
6 * 7      (-3) <= 2      6 * (5 + x)      42
```

Algebraïsche expressies en algebraïsche data representatie

Een *expressie* bestaat uit een waarde, een variabele, of is een toepassing van een *operator* op nul, één of meerdere expressies, en beschrijft zo een berekening die een waarde oplevert.

```
6 * 7      (-3) <= 2        6 * (5 + x)      42

let x = 2 in 6 * (5 + x)        if (-3) <= 2 then 6 * 7 else 42
```

Algebraïsche expressies en algebraïsche data representatie

Een *expressie* bestaat uit een waarde, een variabele, of is een toepassing van een *operator* op nul, één of meerdere expressies, en beschrijft zo een berekening die een waarde oplevert.

```
6 * 7      (-3) <= 2          6 * (5 + x)      42

let x = 2 in 6 * (5 + x)          if (-3) <= 2 then 6 * 7 else 42
```

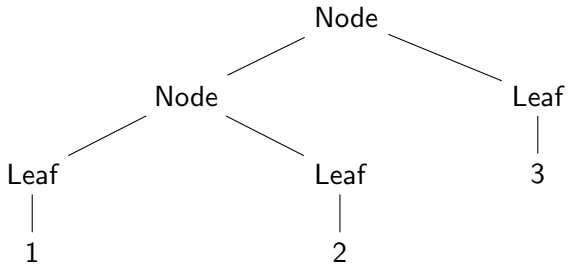
Soorten waardes:

- Primitief/atomair.
  In Haskell: Char, Bool, Int, Integer, Float, Double, ...
- Composiet/samengesteld.
  In Haskell: Tuples, Lists, en door de gebruiker geïntroduceerde algebraïsche datatypes

Termen

Een *term* bestaat uit een waarde, of is een toepassing van een *term constructor* op nul, één of meerdere termen, en beschrijft zo een samengestelde waarde

- Een expressie beschrijft een berekening, een term beschrijft een (samengestelde) waarde

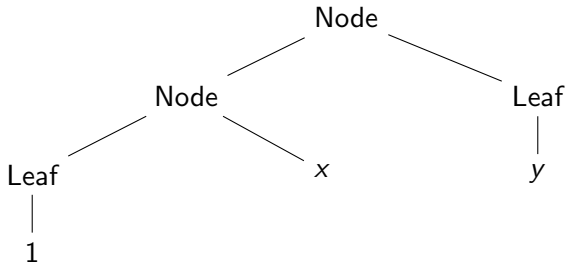



```
data BinIntTree = Node BinIntTree BinIntTree | Leaf Int
my_tree = Node (Node (Leaf 1) (Leaf 2)) (Leaf 3)
```

Patronen

Een *patroon* bestaat uit een waarde, een variable, of is een toepassing van een *term constructor* op nul, één of meerdere patronen, en beschrijft zo een een verzameling van waardes

- Een patroon wordt gebruikt voor branching in declaratieve talen




```
case my_tree of
  Node (Node (Leaf 1) x) (Leaf y) -> -- eval some expression with x and y
  _                               -> -- eval some other expression
```

Patroonherkenning

Tijdens patroonherkenning wordt onderzocht of een bepaalde term binnen een patroon valt:

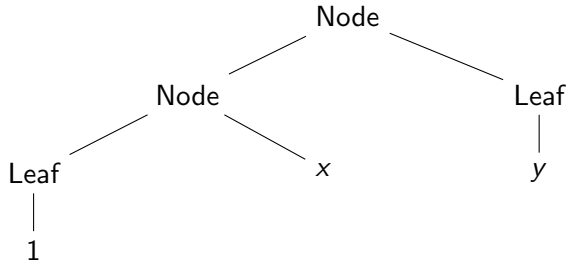


Figure: Patroon

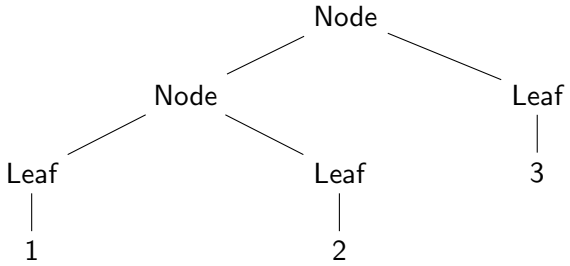


Figure: Term

- Het antwoord komt in de vorm van nul, één of meerdere *substituties*:
  In dit geval één: $\{x \mapsto (Leaf\ 2), y \mapsto 3\}$

Patroonherkenning

Tijdens patroonherkenning wordt onderzocht of een bepaalde term binnen een patroon valt:

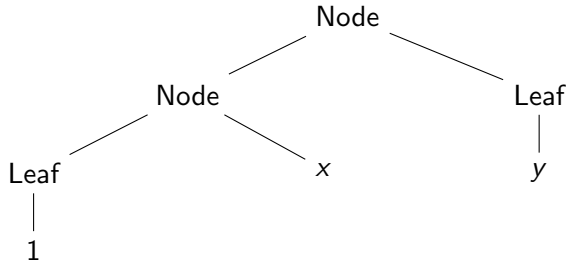


Figure: Patroon

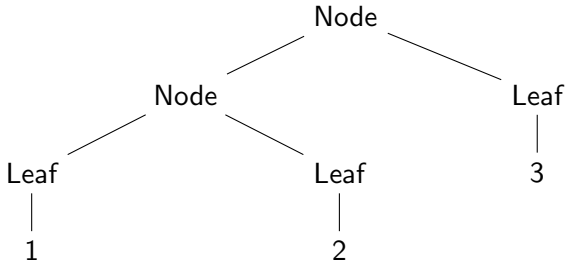


Figure: Term

- Het antwoord komt in de vorm van nul, één of meerdere *substituties*:
  In dit geval één: $\{x \mapsto (\mathit{Leaf}\ 2), y \mapsto 3\}$
- Patroonherkenning wordt toegepast op waardes, expressies worden (mogelijk) geëvalueerd
  In Haskell worden expressies gevalueerd tot Weak Head Normal Form

Wiskundige variabelen

Declaratieve programmeertalen maken over het algemeen gebruik van *wiskundige variabelen*:

- Zijn direct gebonden aan een waarde (als onderdeel van een substitutie)
- Verwijzen conceptueel gezien *niet* naar een aanpasbaar stuk geheugen of register
- Scoping regels: welke substituties zijn er actief tijdens de evaluatie van een expressie?
- Substituties kunnen elkaar vervangen, bijv. door `let ... in ...` te gebruiken:

```
let x = 42
 in let my_inc x = x + 1
 in let y = my_inc x
 in let x = my_inc y
 in (x, y)          -- wat zijn de waardes van de expressies x en y?
```

- Immutable data: operaties op datastructuren maken kopieën, geen aanpassingen

Clausules en recursie

Haskell

```haskell
sum_tree :: BinIntTree -> Int
sum_tree (Leaf i)   = i                          -- eerste clausule
sum_tree (Node l r) = sum_tree l + sum_tree r -- tweede clausule
```

Clausules en recursie

Haskell

```haskell
sum_tree :: BinIntTree -> Int
sum_tree (Leaf i)   = i                          -- eerste clausule
sum_tree (Node l r) = sum_tree l + sum_tree r -- tweede clausule
```

Prolog

```prolog
sum_tree(leaf(I), I).                                   % eerste clausule
sum_tree(node(L,R), Z) :- sum_tree(L,X)                 % tweede clausule
                         ,sum_tree(R,Y)
                         ,Z is X + Y.

?- sum_tree(node(node(leaf(1),leaf(2)),leaf(3)),X).
X = 6.
```

Clausules en recursie

Haskell

```haskell
sum_tree :: BinIntTree -> Int
sum_tree (Leaf i)   = i                          -- eerste clausule
sum_tree (Node l r) = sum_tree l + sum_tree r -- tweede clausule
```

Prolog

```prolog
sum_tree(leaf(I), I).                                   % eerste clausule
sum_tree(node(L,R), Z) :- sum_tree(L,X)                 % tweede clausule
                         ,sum_tree(R,Y)
                         ,Z is X + Y.

?- sum_tree(node(node(leaf(1),leaf(2)),leaf(3)),X).
X = 6.
```

- Beide definities zijn recursief

Clausules en recursie

Haskell

```haskell
sum_tree :: BinIntTree -> Int
sum_tree (Leaf i)   = i                          -- eerste clausule
sum_tree (Node l r) = sum_tree l + sum_tree r -- tweede clausule
```

Prolog

```prolog
sum_tree(leaf(I), I).                                   % eerste clausule
sum_tree(node(L,R), Z) :- sum_tree(L,X)                 % tweede clausule
                         ,sum_tree(R,Y)
                         ,Z is X + Y.

?- sum_tree(node(node(leaf(1),leaf(2)),leaf(3)),X).
X = 6.
```

- Beide definities zijn recursief
- De eerste clausule in de Prolog code toont een non-linear patroon

Blok Declaratief programmeren & Haskell

1. Declaratief programmeren – belangrijkste concepten
2. Haskell – introductie
3. Haskell – geavanceerd
4. Relationeel/logisch programmeren – belangrijkste concepten & prolog
5. Haskell – ontwerpkeuzes
6. Functioneel programmeren – ontwerpkeuzes