Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

College 2010-2011
6. Data Structuren & Bomen

Doaitse Swierstra

Utrecht University

September 28, 2010

College 6: Datastructuren en bomen

Optionele waarden

Een optionele waarde van een bepaald type is

- ofwel ontbrekend,
- ofwel een waarde van dat type.

Datatype van optionele waarden:

> **data** *Maybe a* = *Nothing* | *Just a*.

Optionele waarden

Een optionele waarde van een bepaald type is

- ofwel ontbrekend,
- ofwel een waarde van dat type.

Datatype van optionele waarden:

> **data** *Maybe a* = *Nothing* | *Just a*.

Nothing en *Just* zijn constructorfuncties:

> *Nothing* :: *Maybe a*
> *Just* :: *a*→*Maybe a*.

Optionele waarden

Een optionele waarde van een bepaald type is

- ofwel ontbrekend,
- ofwel een waarde van dat type.

Datatype van optionele waarden:

> **data** *Maybe a* = *Nothing* | *Just a*.

Nothing en *Just* zijn constructorfuncties:

> *Nothing* :: *Maybe a*
> *Just* :: *a*→*Maybe a*.

☞ *Maybe* is geen type, maar een typeconstructor:
een "functie" van types naar types.

Typedefinities

```
data Maybe a = Nothing | Just a
```

Typedefinities

```
data Maybe a = Nothing | Just a
```

Een datatypedefinitie bestaat uit

Typedefinities

data[1] *Maybe* *a* = *Nothing* | *Just* *a*

Een datatypedefinitie bestaat uit

1. het gereserveerde woord **data**;

Typedefinities

$$\textbf{data}^{[1]}\ \mathit{Maybe}^{[2]}\ a \quad = \quad \mathit{Nothing} \quad | \quad \mathit{Just}\ \ a$$

Een datatypedefinitie bestaat uit

1. het gereserveerde woord **data**;
2. een typeconstructornaam (beginnend met een hoofdletter);

Typedefinities

$$\textbf{data}\,[1]\ \mathit{Maybe}\,[2]\ a\,[3] = \mathit{Nothing} \mid \mathit{Just}\ a$$

Een datatypedefinitie bestaat uit

1. het gereserveerde woord **data**;
2. een typeconstructornaam (beginnend met een hoofdletter);
3. nul of meer typeparameters (beginnend met een kleine letter);

Typedefinities

data[1] *Maybe*[2] *a*[3] =[4] *Nothing* | *Just* *a*

Een datatypedefinitie bestaat uit

1. het gereserveerde woord **data**;
2. een typeconstructornaam (beginnend met een hoofdletter);
3. nul of meer typeparameters (beginnend met een kleine letter);
4. het gereserveerde symbool $=$;

Typedefinities

data[1] *Maybe*[2] *a*[3] =[4] *Nothing* |[5] *Just* *a*

Een datatypedefinitie bestaat uit

1. het gereserveerde woord **data**;
2. een typeconstructornaam (beginnend met een hoofdletter);
3. nul of meer typeparameters (beginnend met een kleine letter);
4. het gereserveerde symbool $=$;
5. nul of meer alternatieven, gescheiden door het gereserveerde symbool | en elk bestaande uit

Typedefinities

data[1] *Maybe*[2] *a*[3] =[4] *Nothing*[6] |[5] *Just*[6] *a*

Een datatypedefinitie bestaat uit

1. het gereserveerde woord **data**;
2. een typeconstructornaam (beginnend met een hoofdletter);
3. nul of meer typeparameters (beginnend met een kleine letter);
4. het gereserveerde symbool $=$;
5. nul of meer alternatieven, gescheiden door het gereserveerde symbool | en elk bestaande uit
   6. een constructornaam (beginnend met een hoofdletter),

Typedefinities

data[1] *Maybe*[2] *a*[3] =[4] *Nothing*[6] |[5] *Just*[6] *a*[7]

Een datatypedefinitie bestaat uit

1. het gereserveerde woord **data**;
2. een typeconstructornaam (beginnend met een hoofdletter);
3. nul of meer typeparameters (beginnend met een kleine letter);
4. het gereserveerde symbool $=$;
5. nul of meer alternatieven, gescheiden door het gereserveerde symbool | en elk bestaande uit
   6. een constructornaam (beginnend met een hoofdletter),
   7. nul of meer argumenttypes voor de constructor.

Ons tweede data type

```
data Richting = Noord
              | Oost
              | Zuid
              | West
```

Ons tweede data type

$$\begin{aligned}\textbf{data}\ \mathit{Richting} = {} & \mathit{Noord} \\ & \mid \mathit{Oost} \\ & \mid \mathit{Zuid} \\ & \mid \mathit{West}\end{aligned}$$

Een data type definitie bevat:

1. het keyword **data**

Ons tweede data type

$$\begin{aligned}\textbf{data}\ \mathit{Richting} = {} & \mathit{Noord} \\ & \mid \mathit{Oost} \\ & \mid \mathit{Zuid} \\ & \mid \mathit{West}\end{aligned}$$

Een data type definitie bevat:

1. het keyword **data**
2. een naam, die begint met een hoofdletter

Ons tweede data type

$$\begin{aligned}\textbf{data}\ \mathit{Richting} = {} & \mathit{Noord} \\ & \mid \mathit{Oost} \\ & \mid \mathit{Zuid} \\ & \mid \mathit{West}\end{aligned}$$

Een data type definitie bevat:

1. het keyword **data**
2. een naam, die begint met een hoofdletter
3. een $=$-symbool

Ons tweede data type

$$\begin{aligned}\textbf{data}\ \mathit{Richting} = {} & \mathit{Noord} \\ & \mid \mathit{Oost} \\ & \mid \mathit{Zuid} \\ & \mid \mathit{West}\end{aligned}$$

Een data type definitie bevat:

1. het keyword **data**
2. een naam, die begint met een hoofdletter
3. een $=$-symbool
4. een aantal alternatieven, gescheiden door een $\mid$-symbool

Ons tweede data type

```
data Richting = Noord
              | Oost
              | Zuid
              | West
```

Een data type definitie bevat:

1. het keyword **data**
2. een naam, die begint met een hoofdletter
3. een $=$-symbool
4. een aantal alternatieven, gescheiden door een $|$-symbool

Ons tweede data type

```
data Richting = Noord
              | Oost
              | Zuid
              | West
```

Een data type definitie bevat:

1. het keyword **data**
2. een naam, die begint met een hoofdletter
3. een $=$-symbool
4. een aantal alternatieven, gescheiden door een $|$-symbool

Eigenlijk zijn *Bool*-eans ook zo gedefiniëerd:

```
data Bool = False
          | True
```

Constructoren kunnen voorkomen in patronen

We mogen de op deze manier geïntroduceerde nieuwe constructoren gebruiken in patronen:

$$
\begin{aligned}
&\mathit{keerom}\ \mathit{Noord} = \mathit{Zuid}\\
&\mathit{keerom}\ \mathit{Zuid} = \mathit{Noord}\\
&\mathit{keerom}\ \mathit{Oost} = \mathit{West}\\
&\mathit{keerom}\ \mathit{West} = \mathit{Oost}
\end{aligned}
$$

Constructoren kunnen voorkomen in patronen

We mogen de op deze manier geïntroduceerde nieuwe constructoren gebruiken in patronen:

$$
\begin{aligned}
&\mathit{keerom}\ \mathit{Noord} = \mathit{Zuid}\\
&\mathit{keerom}\ \mathit{Zuid} = \mathit{Noord}\\
&\mathit{keerom}\ \mathit{Oost} = \mathit{West}\\
&\mathit{keerom}\ \mathit{West} = \mathit{Oost}
\end{aligned}
$$

Het lijken waarden die op de parameterplaatsen staan, maar technisch gezien zijn het patronen, bestaande uit een enkele waarde (zoals $[\,]$ als patroon slaagt als het argument de lege lijst is).

Voorbeeld van gebruik *Maybe*

Een mooi voorbeeld van het gebruik van *Maybe* is in de functie *lookup*:

$$
\begin{array}{l}
lookup :: Eq\ a \Rightarrow a \rightarrow [(a,b)] \rightarrow Maybe\ b \\
lookup\ \_\ [\,] \qquad\qquad\qquad\qquad\qquad\quad = Nothing \\
lookup\ k\ ((x,y) : xys) \mid k \equiv x \qquad\quad = Just\ y \\
\qquad\qquad\qquad\qquad\qquad\quad\ \mid otherwise = lookup\ k\ xys
\end{array}
$$

Voorbeeld van gebruik *Maybe*

Een mooi voorbeeld van het gebruik van *Maybe* is in de functie *lookup*:

```
lookup :: Eq a⇒a→[(a,b)]→Maybe b
lookup _ []                          = Nothing
lookup k ((x,y):xys) | k ≡ x         = Just y
                     | otherwise     = lookup k xys
```

Maybe kan gebruikt worden om verschillende pogingen te combineren, waarbij zodra er één slaagt, de rest niet meer geprobeerd wordt:

```
ofanders :: Maybe a→Maybe a→Maybe a
Nothing `ofanders` x = x
y       `ofanders` _ = y   -- hier is y dus gelijk aan Just ...
```

Zoeken in een aantal lijsten

Een voorbeeld van het gebruik is als we een aantal lijsten achter elkaar willen proberen:

$$
\begin{aligned}
&(<|>) :: \mathit{Maybe}\ a \to \mathit{Maybe}\ a \to \mathit{Maybe}\ a \\
&\mathit{Nothing} <|> x = x \\
&y \qquad\quad <|> \_ = y \\
&\mathit{zoekOpIn}\ \mathit{lijsten}\ \mathit{key} = \mathit{foldr}\ (<|>) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathit{Nothing} \\
&\qquad\qquad\qquad\qquad\qquad\qquad (\mathit{map}\ (\mathit{lookup}\ \mathit{key})\ \mathit{lijsten})
\end{aligned}
$$

Zoeken in een aantal lijsten

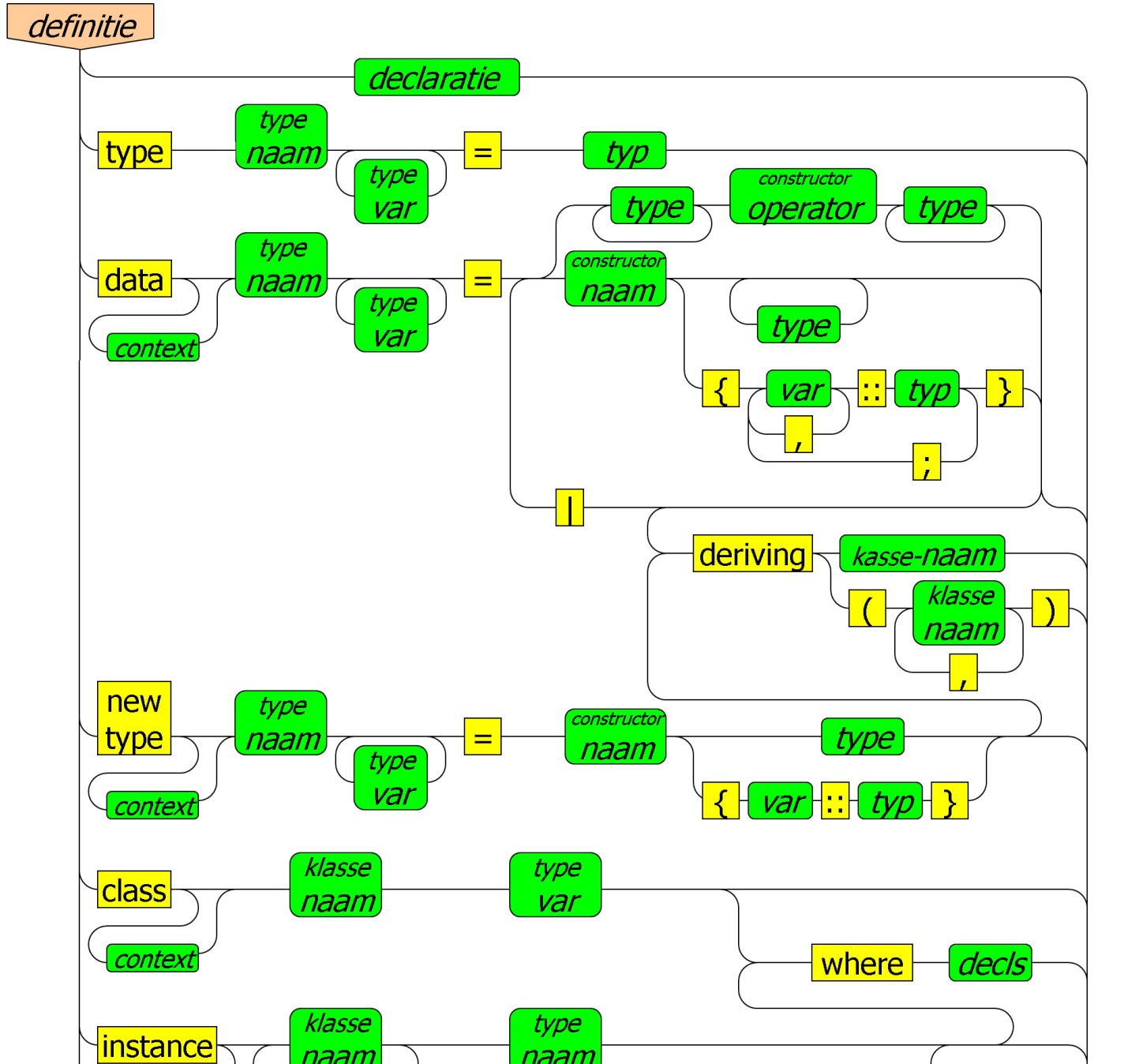
Een voorbeeld van het gebruik is als we een aantal lijsten achter elkaar willen proberen:

```
(<|>) :: Maybe a -> Maybe a -> Maybe a
Nothing <|> x = x
y       <|> _ = y

zoekOpIn lijsten key = foldr (<|>)
                             Nothing
                             (map (lookup key) lijsten)
```

In tegenstelling to wat je zou denken stopt het zoeken zodra je de waarde gevonden hebt. Lazy evaluation zorgt hier weer voor!

Syntax Diagram



Typedefinities: voorbeelden

```
data Bool        = False | True
data Either a b  = Left a | Right b
data Pair  a b   = Pair a b
data Fork a      = Fork a a
```

Typedefinities: voorbeelden

```
data Bool        = False | True
data Either a b  = Left a | Right b
data Pair  a b   = Pair a b
data Fork a      = Fork a a
```

```
False ::                Bool
True  ::                Bool
Left  :: a →            Either a b
Right :: b →            Either a b
Pair  :: a → b →        Pair a b
Fork  :: a → a →        Fork a
```

Typedefinities: voorbeelden

```
data Bool       = False | True
data Either a b = Left a | Right b
data Pair  a b  = Pair a b
data F
```

aantal parameters is gelijk aan aantal componenten

```
False :: Bool
True  :: Bool
Left  :: a → Either a b
Right :: b → Either a b
Pair  :: a → b → Pair a b
Fork  :: a → a → Fork a
```

(A) (B)

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Ingebouwde types: lijsten

Lijsttypes gedragen zich alsof ze gedefinieerd zijn met

data $[\,]\ a = [\,] \mid (:)\ a\ ([\,]\ a)$.

Zelf kun je zo'n definitie niet schrijven, want $[\,]$ is normaliter geen geldige (type)constructornaam.

Ingebouwde types: lijsten

Lijsttypes gedragen zich alsof ze gedefinieerd zijn met

data $[\,]\ a = [\,] \mid (:)\ a\ ([\,]\ a)$.

Zelf kun je zo'n definitie niet schrijven, want $[\,]$ is normaliter geen geldige (type)constructornaam.

In plaats van $[\,]\ a$ schrijven we $[a]$:

data $[a] = [\,] \mid a : [a]$.

Ingebouwde types: lijsten

Lijsttypes gedragen zich alsof ze gedefinieerd zijn met

data $[\,]\ a = [\,] \mid (:)\ a\ ([\,]\ a)$.

Zelf kun je zo'n definitie niet schrijven, want $[\,]$ is normaliter geen geldige (type)constructornaam.

In plaats van $[\,]\ a$ schrijven we $[a]$:

data $[a] = [\,] \mid a : [a]$.

Constructoroperatoren moeten met een dubbele punt beginnen.

Ingebouwde types: tupels

Ook voor tupels (paren, drietupels, viertupels enz.) is er speciale syntaxis, maar paartypes gedragen zich alsof ze gedefinieerd zijn met

data $(,)\ a\ b = (,)\ a\ b$.

Ingebouwde types: tupels

Ook voor tupels (paren, drietupels, viertupels enz.) is er speciale syntaxis, maar paartypes gedragen zich alsof ze gedefinieerd zijn met

data $(,)\ a\ b = (,)\ a\ b$.

In plaats van $(,)\ a\ b$ mag je ook (a,b) schrijven:

data $(a,b) = (a,b)$.

Ingebouwde types: tupels

Ook voor tupels (paren, drietupels, viertupels enz.) is er speciale syntaxis, maar paartypes gedragen zich alsof ze gedefinieerd zijn met

> **data** $(,)\ a\ b = (,)\ a\ b$.

In plaats van $(,)\ a\ b$ mag je ook (a,b) schrijven:

> **data** $(a,b) = (a,b)$.

Voor drietupels is er $(,,)$, voor viertupels $(,,,)$ enz.

Ingebouwde types: lege tupels

Het type van lege tupels, $()$, gedraagt zich alsof het gedefinieerd is met

```
data () = ().
```

Ingebouwde types: gehele getallen en lettertekens

Het type van gehele getallen, Int, gedraagt zich alsof het gedefinieerd is met

data $Int = \ldots \mid (-2) \mid (-1) \mid 0 \mid 1 \mid 2 \mid \ldots.$

Ingebouwde types: gehele getallen en lettertekens

Het type van gehele getallen, *Int*, gedraagt zich alsof het gedefinieerd is met

data *Int* $= \ldots \mid (-2) \mid (-1) \mid 0 \mid 1 \mid 2 \mid \ldots.$

Het type van lettertekens, *Char*, gedraagt zich alsof het gedefinieerd is met

data *Char* $= \ldots \mid$ `'a'` $\mid$ `'b'` $\mid$ `'c'` $\mid \ldots$

Optionele waarden als functieresultaat

Zoeken in een lijst van paren bestaande uit een zoeksleutel van type a en een waarde van type b:

$$
\begin{array}{lll}
lookup :: Eq\ a \Rightarrow a \rightarrow [(a,b)] \rightarrow Maybe\ b & & \\
lookup & \_ \quad [\,] & = Nothing \\
lookup & k \quad ((x,y) : xys) & \\
 & \mid k \equiv x & = Just\ y \\
 & \mid otherwise & = lookup\ k\ xys
\end{array}
$$

Optionele waarden als functieresultaat

Zoeken in een lijst van paren bestaande uit een zoeksleutel van type a en een waarde van type b:

$$
\begin{array}{lll}
lookup :: Eq\ a \Rightarrow a \rightarrow [(a,b)] \rightarrow Maybe\ b & & \\
lookup & \_ & [\,] = Nothing \\
lookup & k & ((x,y) : xys) \\
 & \mid k \equiv x & = Just\ y \\
 & \mid otherwise & = lookup\ k\ xys
\end{array}
$$

Bijvoorbeeld:

$$
\begin{array}{l}
lookup\ 2\ [(2,\texttt{'a'}),(3,\texttt{'b'})] \rightsquigarrow Just\ \texttt{'a'} \\
lookup\ 3\ [(2,\texttt{'a'}),(3,\texttt{'b'})] \rightsquigarrow Just\ \texttt{'b'} \\
lookup\ 5\ [(2,\texttt{'a'}),(3,\texttt{'b'})] \rightsquigarrow Nothing
\end{array}
$$

Berekeningen met defaultwaarden

De functie *fromMaybe* neemt een optionele waarde en produceert

- de waarde, als die er is, of anders
- een gegeven defaultwaarde.

Berekeningen met defaultwaarden

De functie *fromMaybe* neemt een optionele waarde en produceert

- de waarde, als die er is, of anders
- een gegeven defaultwaarde.

$$
\begin{aligned}
&\mathit{fromMaybe} :: a \to \mathit{Maybe}\ a \to a \\
&\mathit{fromMaybe}\quad \_\quad (\mathit{Just}\ x) = x \\
&\mathit{fromMaybe}\quad v\quad \mathit{Nothing} = v
\end{aligned}
$$

Berekeningen met defaultwaarden

De functie *fromMaybe* neemt een optionele waarde en produceert

- de waarde, als die er is, of anders
- een gegeven defaultwaarde.

$$
\begin{aligned}
&\mathit{fromMaybe} :: a \to \mathit{Maybe}\ a \to a \\
&\mathit{fromMaybe}\ \ \_\ \ (\mathit{Just}\ x) = x \\
&\mathit{fromMaybe}\ \ v\ \ \mathit{Nothing} = v
\end{aligned}
$$

Ook bestaat:

$$\mathit{maybe} :: b \to (a \to b) \to \mathit{Maybe}\ a \to b$$

Berekeningen met defaultwaarden

De functie *fromMaybe* neemt een optionele waarde en produceert

- de waarde, als die er is, of anders
- een gegeven defaultwaarde.

$$
\begin{aligned}
&\mathit{fromMaybe} :: a \to \mathit{Maybe}\ a \to a \\
&\mathit{fromMaybe}\quad \_\quad (\mathit{Just}\ x) = x \\
&\mathit{fromMaybe}\quad v\quad \mathit{Nothing} = v
\end{aligned}
$$

Ook bestaat:

$$\mathit{maybe} :: b \to (a \to b) \to \mathit{Maybe}\ a \to b$$

Probeer eens de overeenkomst met *foldr* te herkennen!.

Voorbeeld van gebruik van *Maybe*

Met behulp van *fromMaybe* schrijven we een variatie op *lookup*:

$$
\begin{aligned}
&lookup' :: Int \to [(Int, Char)] \to Char \\
&lookup' \quad k \qquad xys = fromMaybe\ \texttt{'?'}\ (lookup\ k\ xys),
\end{aligned}
$$

Voorbeeld van gebruik van *Maybe*

Met behulp van *fromMaybe* schrijven we een variatie op *lookup*:

$$lookup' :: Int \to [(Int, Char)] \to Char$$
$$lookup' \quad k \qquad xys = fromMaybe\ \texttt{'?'}\ (lookup\ k\ xys),$$

of korter:

$$lookup'\ k = fromMaybe\ \texttt{'?'} \circ lookup\ k.$$

Voorbeeld van gebruik van *Maybe*

Met behulp van *fromMaybe* schrijven we een variatie op *lookup*:

$$\begin{aligned} &lookup' :: Int \to [(Int, Char)] \to Char \\ &lookup' \quad k \qquad xys = fromMaybe\ \texttt{'?'}\ (lookup\ k\ xys), \end{aligned}$$

of korter:

$$lookup'\ k = fromMaybe\ \texttt{'?'} \circ lookup\ k.$$

Bijvoorbeeld:

$$\begin{aligned} &lookup'\ 2\ [(2,\texttt{'a'}),(3,\texttt{'b'})] \rightsquigarrow \texttt{'a'} \\ &lookup'\ 3\ [(2,\texttt{'a'}),(3,\texttt{'b'})] \rightsquigarrow \texttt{'b'} \\ &lookup'\ 5\ [(2,\texttt{'a'}),(3,\texttt{'b'})] \rightsquigarrow \texttt{'?'}. \end{aligned}$$

Patronen

$$id\ x = x$$

$$\begin{aligned} fromMaybe\ \_\ (Just\ x) &= x \\ fromMaybe\ v\ Nothing &= v \end{aligned}$$

Patronen

$$id\ x = x$$

$$\begin{aligned} fromMaybe\ \_\ (Just\ x) &= x \\ fromMaybe\ v\ Nothing &= v \end{aligned}$$

In patronen aan de linkerkant van een functiedefinitie mogen nu voorkomen:

Patronen

$$id\ x^{\boxed{1}} = x$$

$$\begin{aligned} fromMaybe\ \_\ \ (Just\ \ x^{\boxed{1}}) &= x \\ fromMaybe\ v^{\boxed{1}}\ Nothing\ \ \ &= v \end{aligned}$$

In patronen aan de linkerkant van een functiedefinitie mogen nu voorkomen:

- $\boxed{1}$ variabelen,

Patronen

$$id\ x^{\boxed{1}} = x$$

$$\begin{aligned} fromMaybe\ \_^{\boxed{2}}\ (Just\quad x^{\boxed{1}}) &= x \\ fromMaybe\ v^{\boxed{1}}\ Nothing\quad\ \ &= v \end{aligned}$$

In patronen aan de linkerkant van een functiedefinitie mogen nu voorkomen:

1. variabelen,
2. wildcards,

Patronen

$$id\ x^{\boxed{1}} = x$$

$$\begin{aligned} fromMaybe\ \_^{\boxed{2}}\ (Just^{\boxed{3}}\ x^{\boxed{1}}) &= x \\ fromMaybe\ v^{\boxed{1}}\ Nothing^{\boxed{3}} &= v \end{aligned}$$

In patronen aan de linkerkant van een functiedefinitie mogen nu voorkomen:

1. variabelen,
2. wildcards,
3. constructors,

Bomen

Een belangrijke groep datastructuren is die van bomen.

Bomen komen voor in vele vormen. Bijvoorbeeld,

```
data Tree a = Leaf
            | Node (Tree a) a (Tree a)
```

voor het type van bomen met waarden in de knopen.

Bomen

Een belangrijke groep datastructuren is die van bomen.

Bomen komen voor in vele vormen. Bijvoorbeeld,

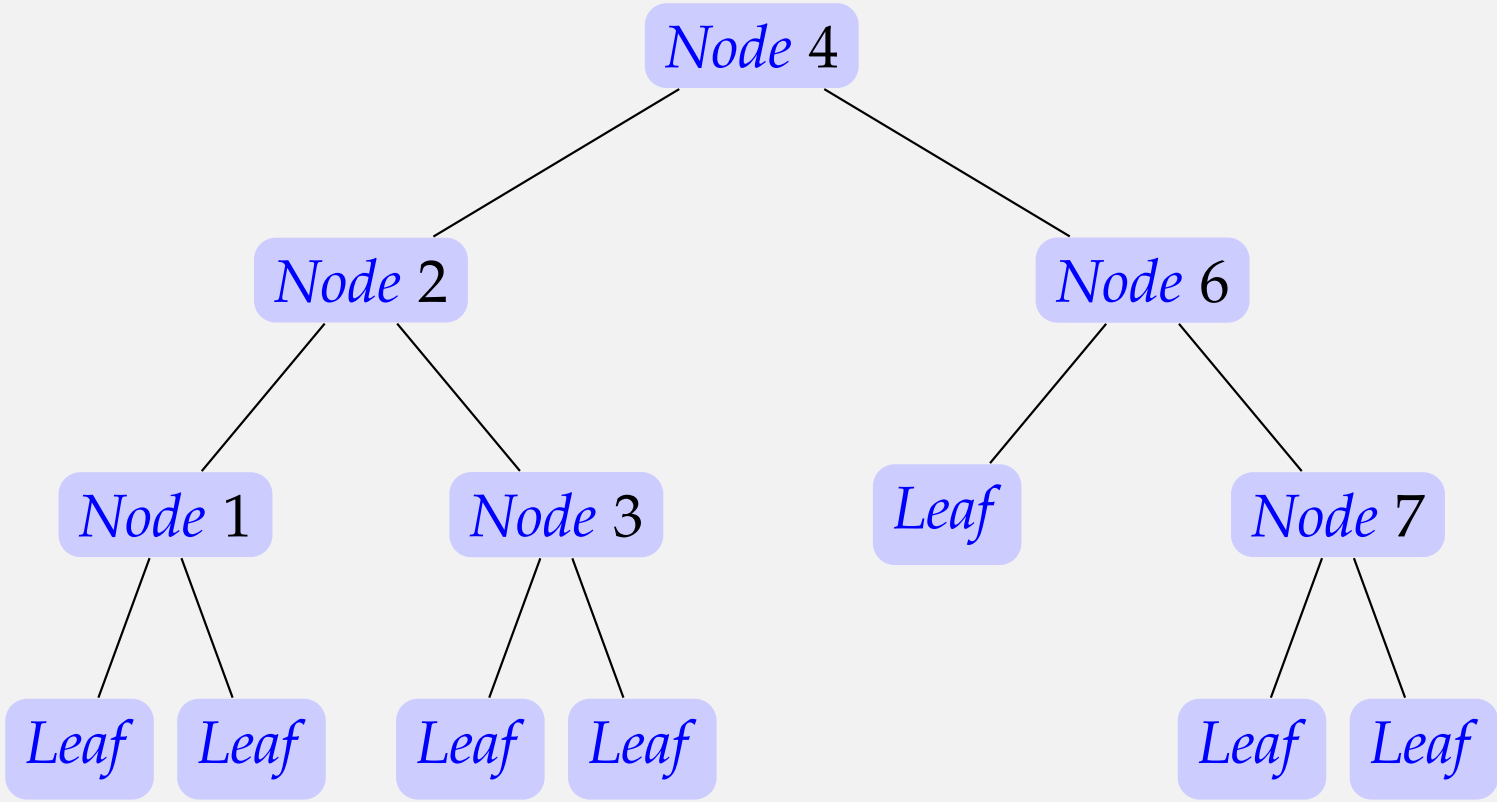
```
data Tree a = Leaf
            | Node (Tree a) a (Tree a)
```

voor het type van bomen met waarden in de knopen.

Een voorbeeld van een expressie die een waarde van dit type bouwt is

```
Node (Node (Node Leaf 1 Leaf) 2 (Node Leaf 3 Leaf))
     4
     (Node Leaf 6 (Node Leaf 7 Leaf)).
```

Bomen: intuïtie



Node (*Node* (*Node Leaf* 1 *Leaf*) 2 (*Node Leaf* 3 *Leaf*))
4
(*Node Leaf* 6 (*Node Leaf* 7 *Leaf*))

Meer boomtypes

Meer boomtypes

Bomen met waarden in de bladeren:

$$\begin{aligned}\textbf{data}\ \mathit{Tree'}\ a = {} & \mathit{Leaf'}\ a \\ & \mid \mathit{Node'}\ (\mathit{Tree'}\ a)\ (\mathit{Tree'}\ a).\end{aligned}$$

Meer boomtypes

Bomen met waarden in de bladeren:

```
data Tree' a = Leaf' a
             | Node' (Tree' a) (Tree' a).
```

Bomen met waarden in de zowel de knopen als de bladeren:

```
data Tree'' a b = Leaf'' a
                | Node'' (Tree'' a b) b (Tree'' a b).
```

Meer boomtypes

Bomen met waarden in de bladeren:

```
data Tree' a = Leaf' a
             | Node' (Tree' a) (Tree' a).
```

Bomen met waarden in de zowel de knopen als de bladeren:

```
data Tree'' a b = Leaf'' a
                | Node'' (Tree'' a b) b (Tree'' a b).
```

Bomen met drie takken in elke knoop:

```
data Tree''' a = Leaf'''
               | Node''' a (Tree''' a) (Tree''' a) (Tree''' a).
```

Meer boomtypes

Bomen met waarden in de bladeren:

```
data Tree' a = Leaf' a
             | Node' (Tree' a) (Tree' a).
```

Bomen met waarden in de zowel de knopen als de bladeren:

```
data Tree'' a b = Leaf'' a
                | Node'' (Tree'' a b) b (Tree'' a b).
```

Bomen met drie takken in elke knoop:

```
data Tree''' a = Leaf'''
               | Node''' a (Tree''' a) (Tree''' a) (Tree''' a).
```

Bomen met willekeurig veel takken in elke knoop (rhododendrons):

```
data RoseTree a = RoseLeaf
                | RoseNode a [RoseTree a].
```

Typeconstructoren van hogere orde

Net als functies functies als argument kunnen meekrijgen, kunnen typeconstructoren typeconstructoren als argument meekrijgen:

```
data GTree f a = GLeaf | GNode a (f (GTree f a)).
```

Typeconstructoren van hogere orde

Net als functies functies als argument kunnen meekrijgen, kunnen typeconstructoren typeconstructoren als argument meekrijgen:

$$\textbf{data}\ GTree\ f\ a = GLeaf \mid GNode\ a\ (f\ (GTree\ f\ a)).$$

Bijvoorbeeld:

Typeconstructoren van hogere orde

Net als functies functies als argument kunnen meekrijgen, kunnen typeconstructoren typeconstructoren als argument meekrijgen:

$$\textbf{data}\ \mathit{GTree}\ f\ a = \mathit{GLeaf} \mid \mathit{GNode}\ a\ (f\ (\mathit{GTree}\ f\ a)).$$

Bijvoorbeeld:

Bomen met willekeurig veel takken in elke knoop:

$$\textbf{type}\ \mathit{RoseTree}\ a = \mathit{GTree}\ [\,]\ a.$$

Typeconstructoren van hogere orde

Net als functies functies als argument kunnen meekrijgen, kunnen typeconstructoren typeconstructoren als argument meekrijgen:

$$\textbf{data}\ \mathit{GTree}\ f\ a = \mathit{GLeaf} \mid \mathit{GNode}\ a\ (f\ (\mathit{GTree}\ f\ a)).$$

Bijvoorbeeld:

Bomen met willekeurig veel takken in elke knoop:

$$\textbf{type}\ \mathit{RoseTree}\ a = \mathit{GTree}\ [\,]\ a.$$

Bomen met twee takken in elke knoop:

$$\textbf{type}\ \mathit{BinTree}\ a = \mathit{GTree}\ \mathit{Fork}\ a.$$

Functies over bomen

Recursieve datastructuren leiden tot recursieve functies.

$$\textbf{data}\ \mathit{Tree}\ a = \mathit{Leaf} \mid \mathit{Node}\ (\mathit{Tree}\ a)\ a\ (\mathit{Tree}\ a)$$

Functies over bomen worden typisch geschreven door voor elke constructor een geval te definiëren:

$$\begin{aligned} &\mathit{size} :: \mathit{Tree}\ a && \to \mathit{Int} \\ &\mathit{size}\ \ \mathit{Leaf} && = 0 \\ &\mathit{size}\ \ (\mathit{Node}\ l\ \_\ r) && = \mathit{size}\ l + 1 + \mathit{size}\ r. \end{aligned}$$

Functies over bomen

Recursieve datastructuren leiden tot recursieve functies.

$$\textbf{data}\ \mathit{Tree}\ a = \mathit{Leaf} \mid \mathit{Node}\ (\mathit{Tree}\ a)\ a\ (\mathit{Tree}\ a)$$

Functies over bomen worden typisch geschreven door voor elke constructor een geval te definiëren:

$$\begin{array}{lll} \mathit{size} :: \mathit{Tree}\ a & \to \mathit{Int} \\ \mathit{size}\ \ \mathit{Leaf} & = 0 \\ \mathit{size}\ \ (\mathit{Node}\ l\ \_\ r) & = \mathit{size}\ l + 1 + \mathit{size}\ r. \end{array}$$

☞ De functie *size* is polymorf in het type van de elementen in de boom.

Functies over bomen

Recursieve datastructuren leiden tot recursieve functies.

$$\textbf{data}\ \mathit{Tree}\ a = \mathit{Leaf} \mid \mathit{Node}\ (\mathit{Tree}\ a)\ a\ (\mathit{Tree}\ a)$$

Functies over bomen worden typisch geschreven door voor elke constructor een geval te definiëren:

$$\begin{aligned} &\mathit{size} :: \mathit{Tree}\ a && \to \mathit{Int} \\ &\mathit{size}\ \ \mathit{Leaf} && = 0 \\ &\mathit{size}\ \ (\mathit{Node}\ l\ \_\ r) && = \mathit{size}\ l + 1 + \mathit{size}\ r. \end{aligned}$$

☞ De functie *size* is polymorf in het type van de elementen in de boom.

☞ De haakjes in het tweede patroon zijn **echt nodig**

Zoeken in lijsten

Zoeken in lijsten is duur:

$$
\begin{array}{lllll}
elem :: Eq\ a \Rightarrow a \rightarrow [a] & & & \rightarrow Bool \\
elem & \_ & [\,] & = False \\
elem & e & (x : xs) & = x \equiv e \vee elem\ e\ xs.
\end{array}
$$

Zoeken in lijsten

Zoeken in lijsten is duur:

$$
\begin{array}{lll}
elem :: Eq\ a \Rightarrow a \rightarrow [a] & & \rightarrow Bool \\
elem & \_ \quad [\,] & = False \\
elem & e \quad (x : xs) & = x \equiv e \vee elem\ e\ xs.
\end{array}
$$

Als het gezochte element niet in de lijst voorkomt, komen we daar pas achter als we de hele lijst doorlopen hebben.

Zoeken in gesorteerde lijsten

Zoeken in gesorteerde lijsten is, gemiddeld, iets goedkoper:

$$
\begin{array}{lllll}
elem_{Ord} :: Ord\ a \Rightarrow a \rightarrow [a] & & & & \rightarrow Bool \\
elem_{Ord} & \_ & [\,] & & = False \\
elem_{Ord} & e & (x : xs) & \mid e \equiv x & = True \\
 & & & \mid e < x & = False \\
 & & & \mid otherwise & = e\ `elem_{Ord}`\ xs.
\end{array}
$$

Als we eenmaal een waarde tegengekomen zijn die groter is dan de gezochte waarde, heeft verder zoeken geen zin.

Zoeken in gesorteerde lijsten

Zoeken in gesorteerde lijsten is, gemiddeld, iets goedkoper:

$$
\begin{array}{lllll}
elem_{Ord} :: Ord\ a \Rightarrow a \rightarrow [a] & & & & \rightarrow Bool \\
elem_{Ord} & \_ & [\,] & & = False \\
elem_{Ord} & e & (x : xs) & \mid e \equiv x & = True \\
 & & & \mid e < x & = False \\
 & & & \mid otherwise & = e\ `elem_{Ord}`\ xs.
\end{array}
$$

Als we eenmaal een waarde tegengekomen zijn die groter is dan de gezochte waarde, heeft verder zoeken geen zin.

Echter: als het gezochte element wel in de lijst zit, moeten we nog steeds die lijst helemaal tot aan de positie van het element aflopen.

Zoekbomen

Zoeken in zogenaamde zoekbomen is, in het algemeen, goedkoper.

De waarden waarin gezocht wordt, worden opgeslagen in een boom:

$$\begin{aligned}\textbf{data}\ \mathit{Tree}\ a = {} & \mathit{Leaf} \\ & \mid \mathit{Node}\ (\mathit{Tree}\ a)\ a\ (\mathit{Tree}\ a).\end{aligned}$$

Zoekbomen

Zoeken in zogenaamde zoekbomen is, in het algemeen, goedkoper.

De waarden waarin gezocht wordt, worden opgeslagen in een boom:

```
data Tree a = Leaf
            | Node (Tree a) a (Tree a).
```

Bij het bouwen van de boom zorgen we ervoor dat

- alle waarden in de linkerdeelboom van een knoop kleiner dan of gelijk zijn aan de waarde in de knoop, en
- alle waarden in de rechterdeelboom van een knoop groter zijn dan de waarde in de knoop.

Zoeken in zoekbomen

Zoeken naar een waarde in een zoekboom is heel eenvoudig:

- als de gezochte waarde gelijk is aan de waarde in een knoop, dan hebben we de waarde gevonden;
- als de gezochte waarde kleiner is dan de waarde in een knoop, dan zoeken we verder in de linkerdeelboom;
- als de gezochte waarde groter is dan de waarde in een knoop, dan zoeken we verder in de rechterdeelboom.

Zoeken in zoekbomen

Zoeken naar een waarde in een zoekboom is heel eenvoudig:

- als de gezochte waarde gelijk is aan de waarde in een knoop, dan hebben we de waarde gevonden;
- als de gezochte waarde kleiner is dan de waarde in een knoop, dan zoeken we verder in de linkerdeelboom;
- als de gezochte waarde groter is dan de waarde in een knoop, dan zoeken we verder in de rechterdeelboom.

$$
\begin{array}{lll}
elem_{Tree} :: Ord\ a \Rightarrow a \rightarrow Tree\ a & & \rightarrow Bool \\
elem_{Tree} & \_ \quad Leaf & = False \\
elem_{Tree} & e \quad (Node\ l\ x\ r) & \\
\quad \mid e \equiv x & & = True \\
\quad \mid e < x & & = elem_{Tree}\ e\ l \\
\quad \mid otherwise & & = elem_{Tree}\ e\ r.
\end{array}
$$

Opbouwen van een zoekboom

We kunnen een zoekboom opbouwen door te beginnen met een enkel blad en alle elementen één voor één toe te voegen:

$$
\begin{aligned}
&toSearchTree :: Ord\ a \Rightarrow [a] \rightarrow Tree\ a \\
&toSearchTree\ [\,] \quad\quad = Leaf \\
&toSearchTree\ (x : xs) = insert_{Tree}\ x\ (toSearchTree\ xs)
\end{aligned}
$$

Opbouwen van een zoekboom

We kunnen een zoekboom opbouwen door te beginnen met een enkel blad en alle elementen één voor één toe te voegen:

$$
\begin{aligned}
&toSearchTree :: Ord\ a \Rightarrow [a] \rightarrow Tree\ a \\
&toSearchTree\ [\,] \quad\quad = Leaf \\
&toSearchTree\ (x : xs) = insert_{Tree}\ x\ (toSearchTree\ xs)
\end{aligned}
$$

of, met behulp van *foldr*:

$$
\begin{aligned}
&toSearchTree :: Ord\ a \Rightarrow [a] \rightarrow Tree\ a \\
&toSearchTree = foldr\ insert_{Tree}\ Leaf\,.
\end{aligned}
$$

Opbouwen van een zoekboom

We kunnen een zoekboom opbouwen door te beginnen met een enkel blad en alle elementen één voor één toe te voegen:

$$
\begin{aligned}
&\mathit{toSearchTree} :: \mathit{Ord}\ a \Rightarrow [a] \rightarrow \mathit{Tree}\ a \\
&\mathit{toSearchTree}\ [\,] \quad\quad = \mathit{Leaf} \\
&\mathit{toSearchTree}\ (x : xs) = \mathit{insert}_{\mathit{Tree}}\ x\ (\mathit{toSearchTree}\ xs)
\end{aligned}
$$

of, met behulp van *foldr*:

$$
\begin{aligned}
&\mathit{toSearchTree} :: \mathit{Ord}\ a \Rightarrow [a] \rightarrow \mathit{Tree}\ a \\
&\mathit{toSearchTree} = \mathit{foldr}\ \mathit{insert}_{\mathit{Tree}}\ \mathit{Leaf}\,.
\end{aligned}
$$

☞ Maar: hoe is $\mathit{insert}_{\mathit{Tree}}$ gedefinieerd?

LiElementen invoegen in een zoekboom Bij het invoegen van een element in een zoekboom moeten we erop letten de zoekboomeigenschap in stand te houden:

$$
\begin{aligned}
&insert_{Tree} :: Ord\ a \Rightarrow a \rightarrow Tree\ a \qquad\qquad \rightarrow Tree\ a \\
&insert_{Tree}\ e\ Leaf = Node\ Leaf\ e\ Leaf \\
&insert_{Tree}\ e\ (Node\ l\ x\ r) \\
&\quad \mid e \leqslant x \qquad = Node\ (insert_{Tree}\ e\ l)\ x\ r \\
&\quad \mid otherwise = Node\ l \qquad\qquad\qquad x\ (insert_{Tree}\ e\ r).
\end{aligned}
$$

LiElementen invoegen in een zoekboom Bij het invoegen van een element in een zoekboom moeten we erop letten de zoekboomeigenschap in stand te houden:

$$
\begin{aligned}
&insert_{Tree} :: Ord\ a \Rightarrow a \rightarrow Tree\ a \qquad\qquad \rightarrow Tree\ a \\
&insert_{Tree}\ e\ Leaf = Node\ Leaf\ e\ Leaf \\
&insert_{Tree}\ e\ (Node\ l\ x\ r) \\
&\quad \mid e \leqslant x \qquad = Node\ (insert_{Tree}\ e\ l)\ x\ r \\
&\quad \mid otherwise = Node\ l \qquad\qquad\qquad x\ (insert_{Tree}\ e\ r).
\end{aligned}
$$

☞ In bomen die gebouwd worden met $insert_{Tree}$ kunnen elementen dubbel voorkomen. Hoe kunnen we dit voorkomen?

LiElementen invoegen in een zoekboom Bij het invoegen van een element in een zoekboom moeten we erop letten de zoekboomeigenschap in stand te houden:

$$
\begin{aligned}
&insert_{Tree} :: Ord\ a \Rightarrow a \rightarrow Tree\ a \qquad\qquad \rightarrow Tree\ a \\
&insert_{Tree}\ e\ Leaf = Node\ Leaf\ e\ Leaf \\
&insert_{Tree}\ e\ (Node\ l\ x\ r) \\
&\quad \mid e \leqslant x \qquad = Node\ (insert_{Tree}\ e\ l)\ x\ r \\
&\quad \mid otherwise = Node\ l \qquad\qquad\qquad x\ (insert_{Tree}\ e\ r).
\end{aligned}
$$

☞ In bomen die gebouwd worden met $insert_{Tree}$ kunnen elementen dubbel voorkomen. Hoe kunnen we dit voorkomen?

$$
\begin{aligned}
&insert_{Tree}\ e\ (Node\ l\ x\ r) \\
&\quad \mid e \equiv x \qquad = Node\ l\ x\ r \\
&\quad \mid e < x \qquad = Node\ (insert_{Tree}\ e\ l)\ x\ r \\
&\quad \mid otherwise = Node\ l \qquad\qquad\qquad x\ (insert_{Tree}\ e\ r)
\end{aligned}
$$

LiElementen invoegen in een zoekboom Bij het invoegen van een element in een zoekboom moeten we erop letten de zoekboomeigenschap in stand te houden:

$$
\begin{aligned}
&insert_{Tree} :: Ord\ a \Rightarrow a \rightarrow Tree\ a \qquad\qquad \rightarrow Tree\ a \\
&insert_{Tree}\ e\ Leaf = Node\ Leaf\ e\ Leaf \\
&insert_{Tree}\ e\ (Node\ l\ x\ r) \\
&\quad \mid e \leqslant x \qquad = Node\ (insert_{Tree}\ e\ l)\ x\ r \\
&\quad \mid otherwise = Node\ l \qquad\qquad\qquad x\ (insert_{Tree}\ e\ r).
\end{aligned}
$$

☞ In bomen die gebouwd worden met $insert_{Tree}$ kunnen elementen dubbel voorkomen. Hoe kunnen we dit voorkomen?

$$
\begin{aligned}
&insert_{Tree}\ e\ t@(Node\ l\ x\ r) \\
&\quad \mid e \equiv x \qquad = t \\
&\quad \mid e < x \qquad = Node\ (insert_{Tree}\ e\ l)\ x\ r \\
&\quad \mid otherwise = Node\ l \qquad\qquad\qquad x\ (insert_{Tree}\ e\ r)
\end{aligned}
$$

Verwijderen van elementen uit een zoekboom

Bij het verwijderen van een element uit een zoekboom doet zich een complicatie voor:

$$
\begin{aligned}
&delete_{Tree} :: Ord\ a \Rightarrow a \rightarrow Tree\ a \rightarrow Tree\ a \\
&delete_{Tree} \quad\quad\quad\quad\ \_ \quad Leaf \quad = Leaf \\
&delete_{Tree} \quad\quad\quad\quad\ e \quad (Node\ l\ x\ r) \\
&\quad \mid e \equiv x \quad\quad = \textbf{???} \\
&\quad \mid e < x \quad\quad = Node\ (delete\ e\ l)\ x\ r \\
&\quad \mid otherwise = Node\ l\ x\ (delete\ e\ r).
\end{aligned}
$$

Verwijderen van elementen uit een zoekboom

Bij het verwijderen van een element uit een zoekboom doet zich een complicatie voor:

$$
\begin{aligned}
&delete_{Tree} :: Ord\ a \Rightarrow a \rightarrow Tree\ a \rightarrow Tree\ a \\
&delete_{Tree} \quad\quad\quad\quad\ \_ \quad Leaf \quad = Leaf \\
&delete_{Tree} \quad\quad\quad\quad\ e \quad (Node\ l\ x\ r) \\
&\quad \mid e \equiv x \quad\quad = \mathbf{???} \\
&\quad \mid e < x \quad\quad = Node\ (delete\ e\ l)\ x\ r \\
&\quad \mid otherwise = Node\ l\ x\ (delete\ e\ r).
\end{aligned}
$$

We missen nu een waarde om de plaats in te nemen van het weg te gooien element: er zit niets anders op dan de linker- en rechterdeelboom samen te voegen.

Verwijderen van elementen uit een zoekboom

Bij het verwijderen van een element uit een zoekboom doet zich een complicatie voor:

```
deleteTree :: Ord a ⇒ a → Tree a → Tree a
deleteTree _ Leaf = Leaf
deleteTree e (Node l x r)
    | e ≡ x     = joinTree l r
    | e < x     = Node (delete e l) x r
    | otherwise = Node l x (delete e r).
```

We missen nu een waarde om de plaats in te nemen van het weg te gooien element: er zit niets anders op dan de linker- en rechterdeelboom samen te voegen.

Samenvoegen van zoekbomen

$$
\begin{array}{lllll}
join_{Tree} & :: Tree\ a \to Tree\ a \to Tree\ a & & & \\
join_{Tree} & Leaf & r & = & r \\
join_{Tree} & l & Leaf & = & l \\
join_{Tree} & l & r & = & \textbf{let}\ (x, l') = maxFrom\ l \\
 & & & & \quad \textbf{in}\ \ Node\ l'\ x\ r
\end{array}
$$

Samenvoegen van zoekbomen

$$
\begin{array}{lllll}
join_{Tree} :: Tree\ a \to Tree\ a \to Tree\ a & & & & \\
join_{Tree} & Leaf & r & = & r \\
join_{Tree} & l & Leaf & = & l \\
join_{Tree} & l & r & = & \textbf{let}\ (x, l') = maxFrom\ l \\
 & & & & \quad \textbf{in}\ \ Node\ l'\ x\ r
\end{array}
$$

De functie *maxFrom* haalt het grootste element uit een boom en levert het samen met de rest van de boom op.

Samenvoegen van zoekbomen

$$
\begin{array}{lll}
join_{Tree} :: Tree\ a \to Tree\ a \to Tree\ a & & \\
join_{Tree} & Leaf & r = r \\
join_{Tree} & l & Leaf = l \\
join_{Tree} & l & r = \textbf{let}\ (x, l') = maxFrom\ l \\
 & & \quad \textbf{in}\ \ Node\ l'\ x\ r
\end{array}
$$

De functie *maxFrom* haalt het grootste element uit een boom en levert het samen met de rest van de boom op:

$$
\begin{array}{lll}
maxFrom :: Tree\ a & & \to (a, Tree\ a) \\
maxFrom & Leaf & = error\ \texttt{"maxFrom: Leaf"} \\
maxFrom & (Node\ l\ x\ Leaf) & = (x, l) \\
maxFrom & (Node\ l\ x\ r) & = \textbf{let}\ (y, r') = maxFrom\ r \\
 & & \quad \textbf{in}\ \ (y, Node\ l\ x\ r').
\end{array}
$$

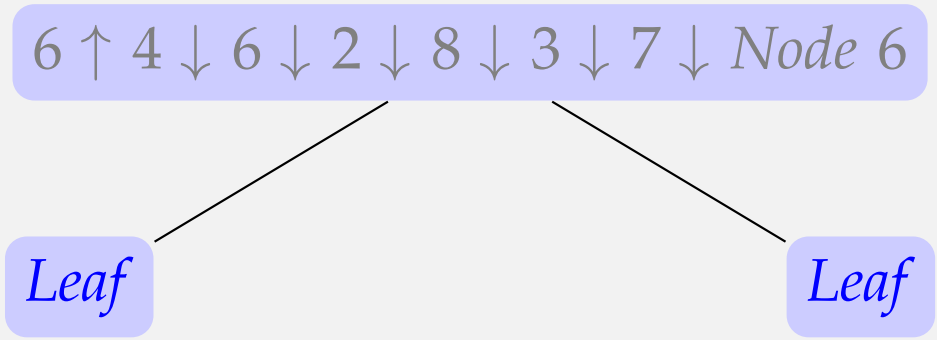
Zoekbomen: animatie

$$6 \uparrow 4 \downarrow 6 \downarrow 2 \downarrow 8 \downarrow 3 \downarrow 7 \downarrow 6 \downarrow \mathit{Leaf}$$

$$(\downarrow) = \mathit{insert}_{\mathit{Tree}}$$
$$(\uparrow) = \mathit{delete}_{\mathit{Tree}}$$

$$(\bowtie) = \mathit{join}_{\mathit{Tree}}$$
$$\mathit{mf} = \mathit{maxFrom}$$

Zoekbomen: animatie



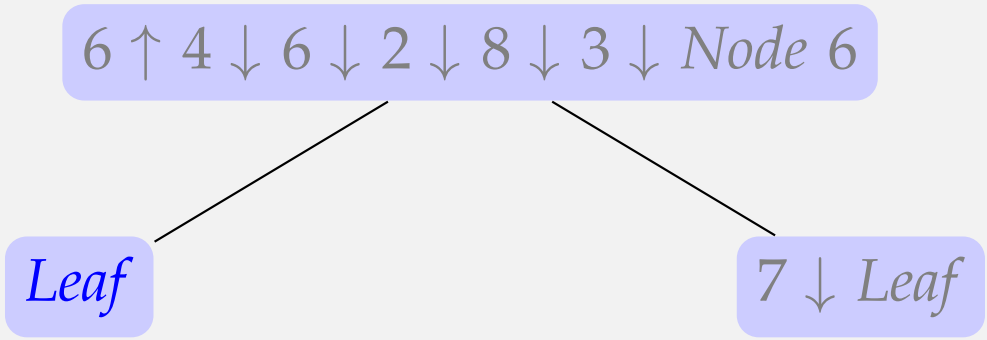
$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Zoekbomen: animatie



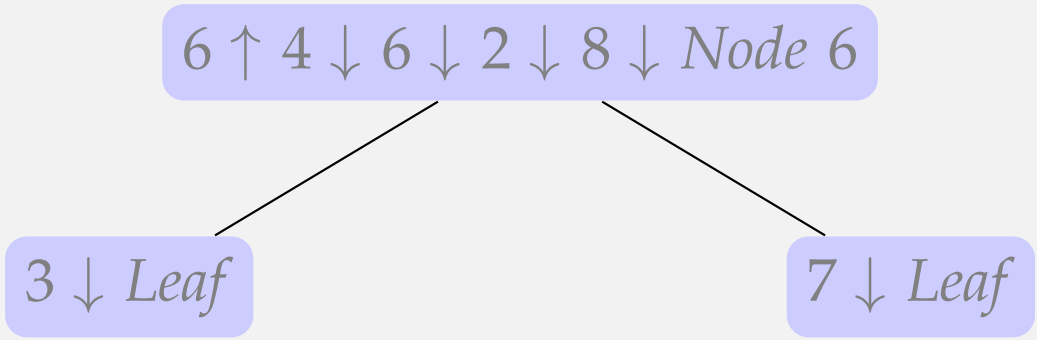
$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Zoekbomen: animatie



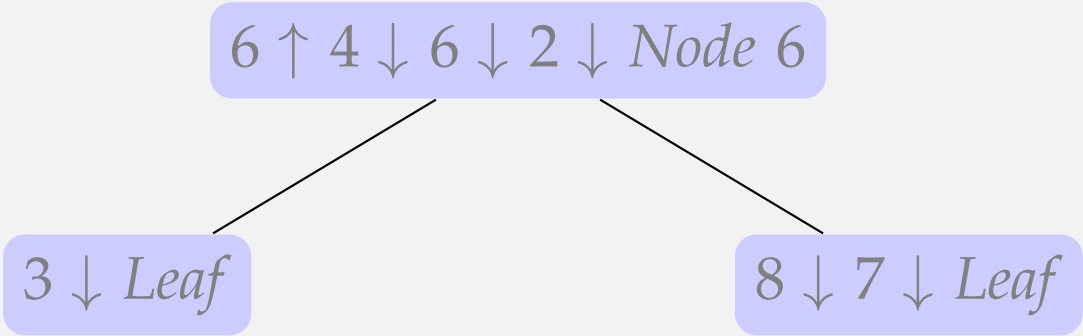
$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Zoekbomen: animatie



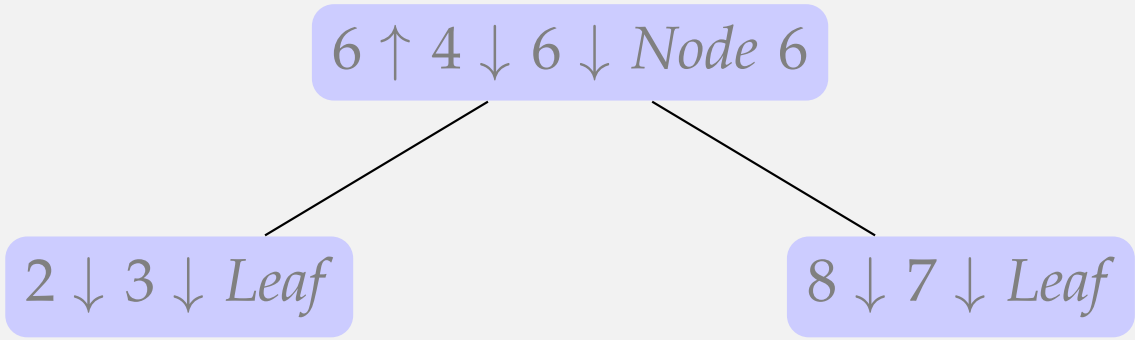
$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Zoekbomen: animatie



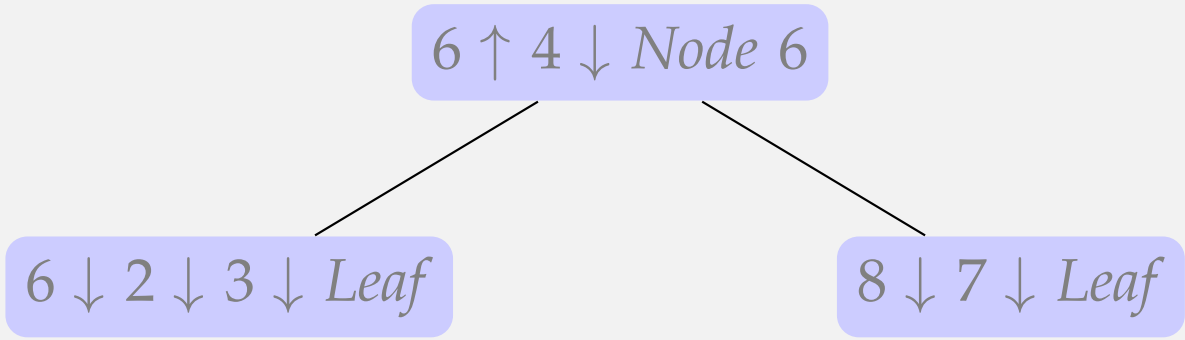
$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

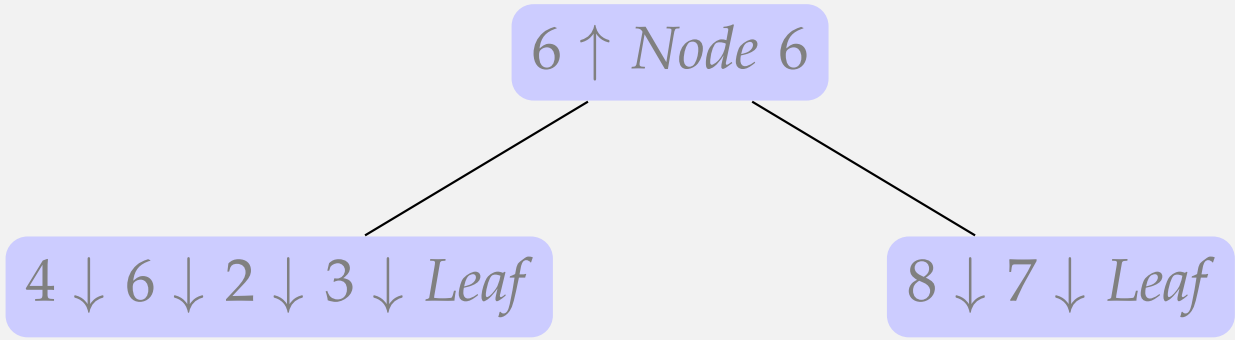
Zoekbomen: animatie



$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

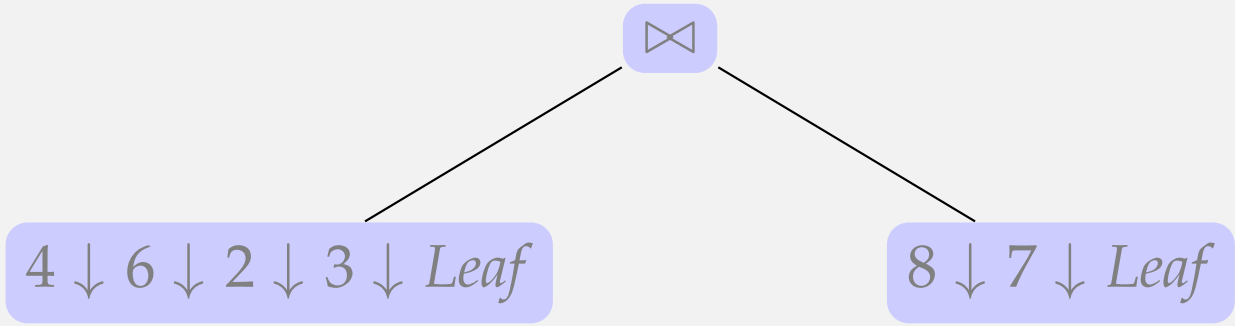
Zoekbomen: animatie



$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

Zoekbomen: animatie



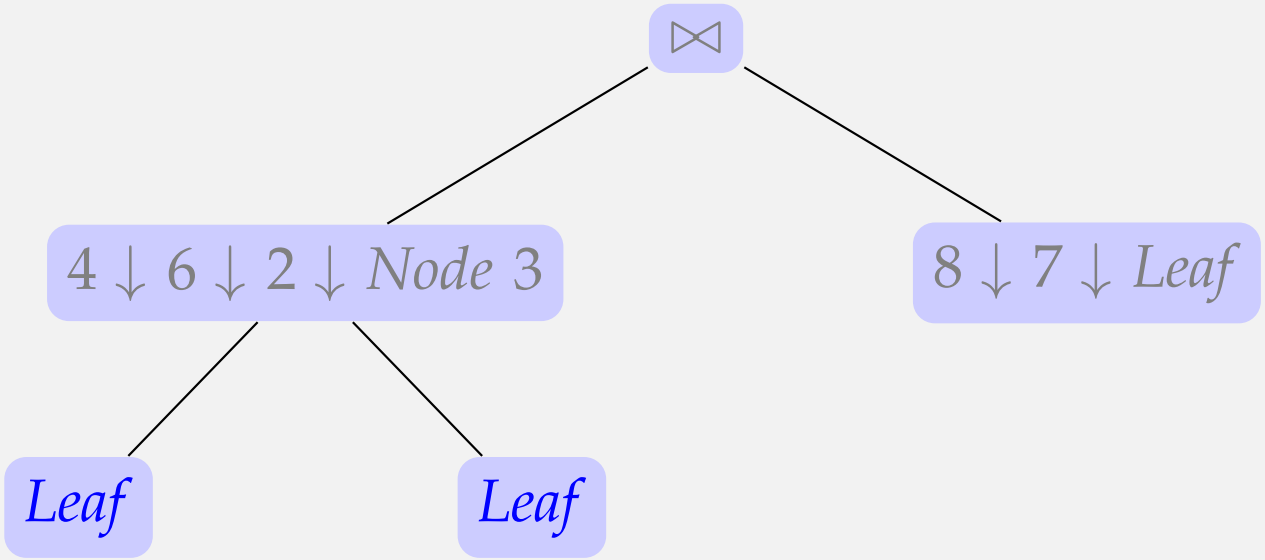
$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

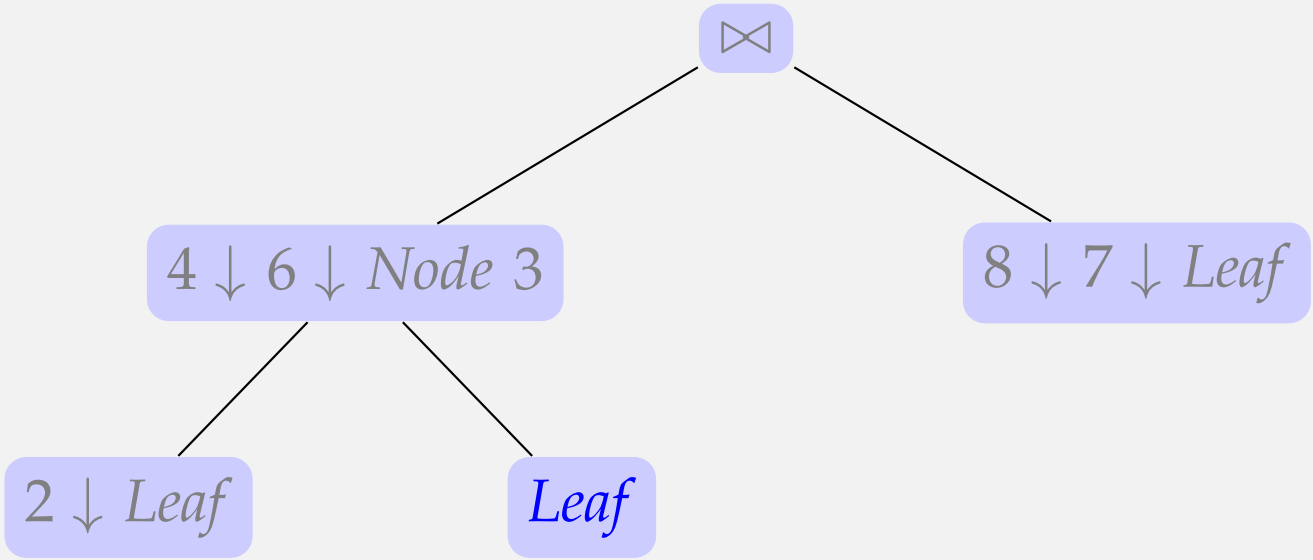
Zoekbomen: animatie



$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

Zoekbomen: animatie



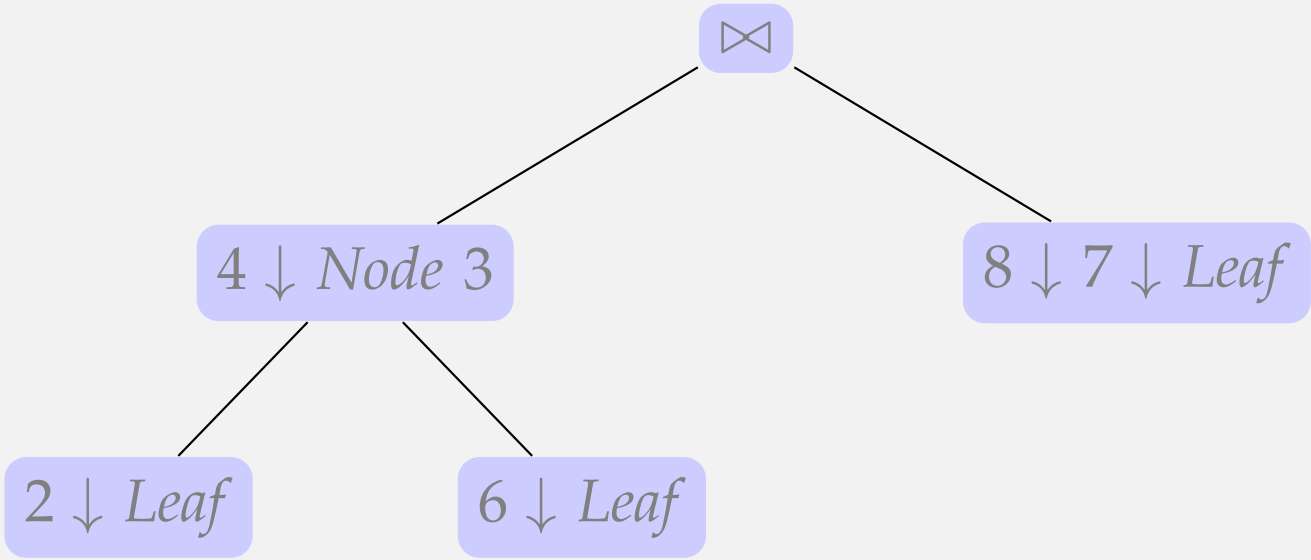
$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

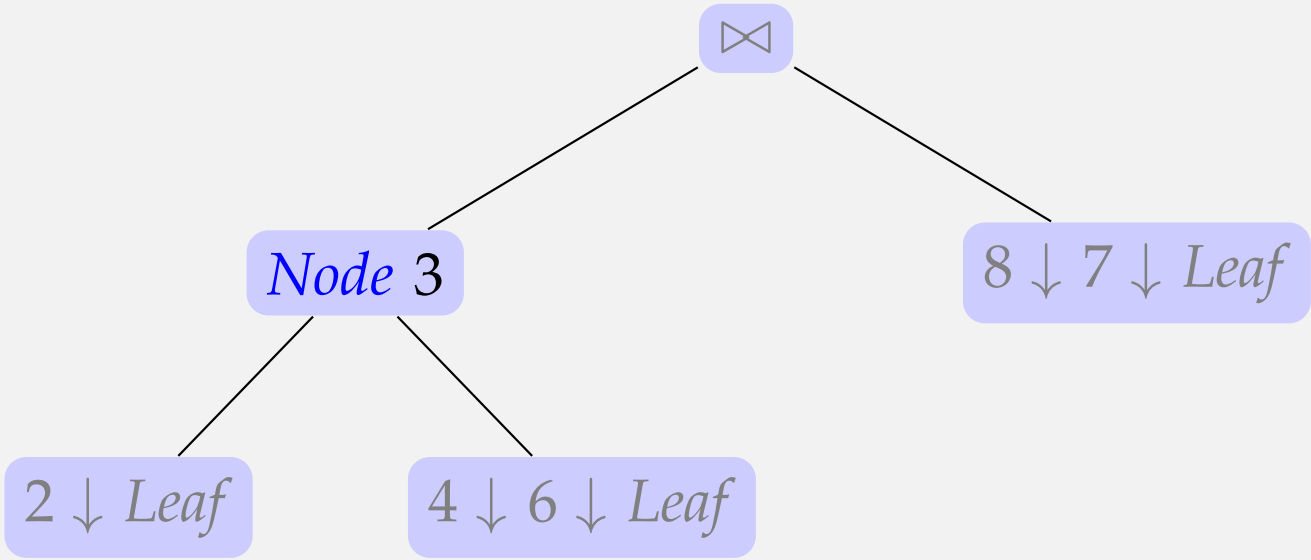
Zoekbomen: animatie



$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

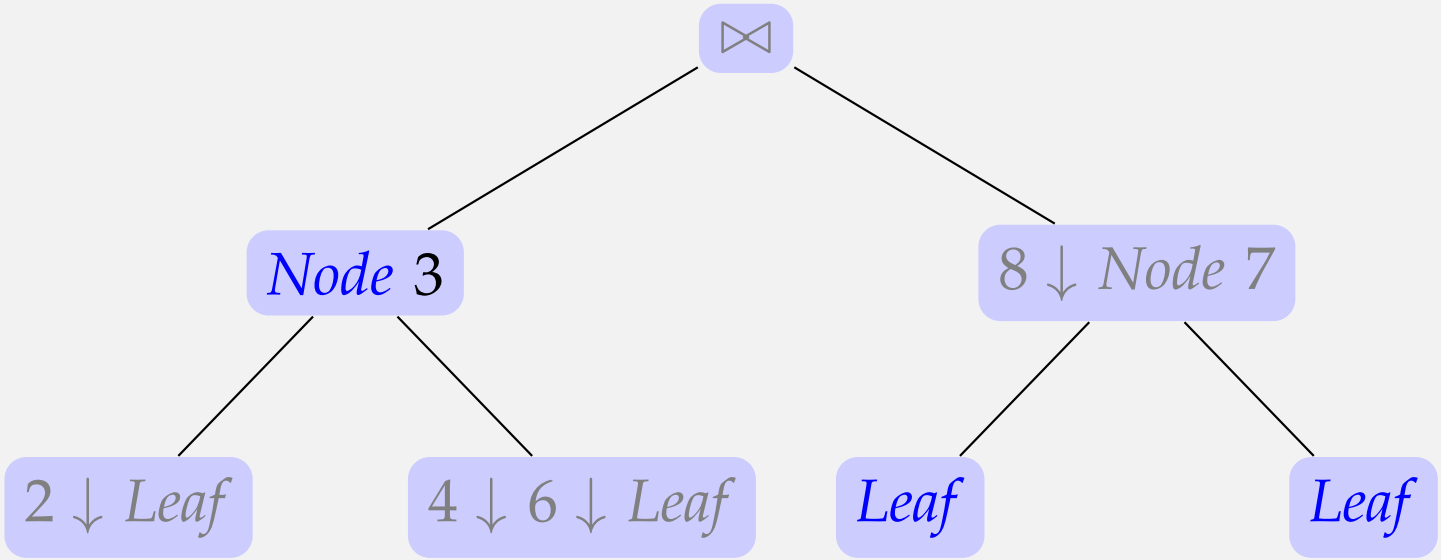
Zoekbomen: animatie



$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

Zoekbomen: animatie



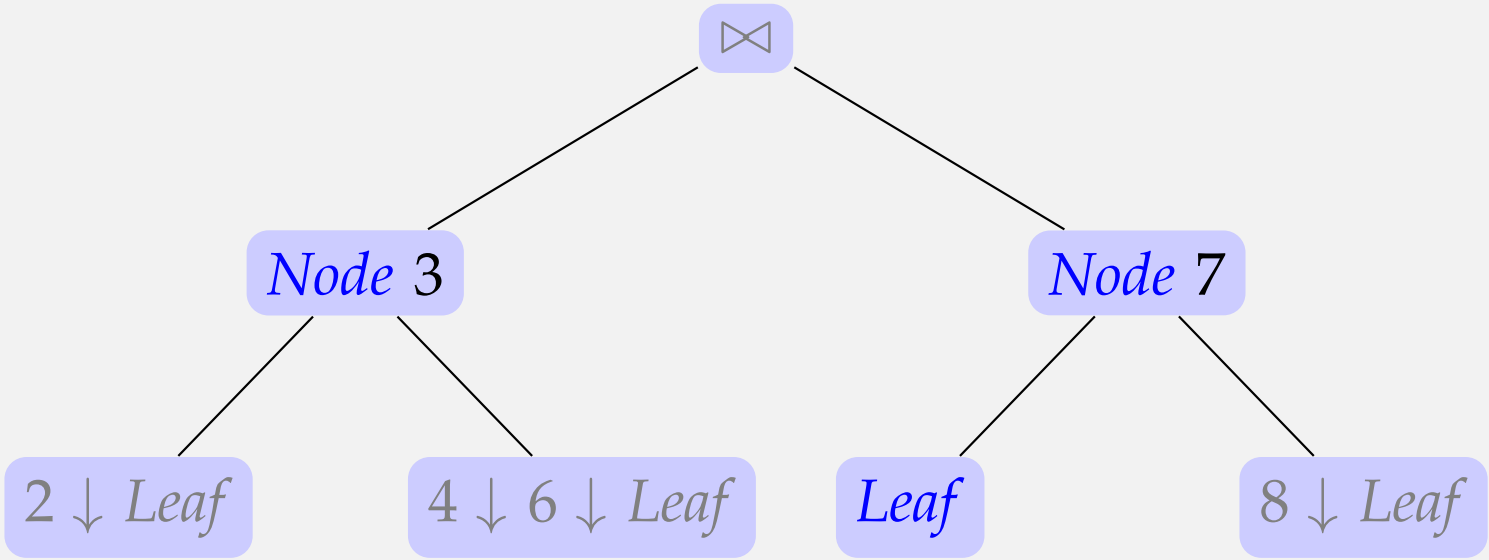
$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Zoekbomen: animatie



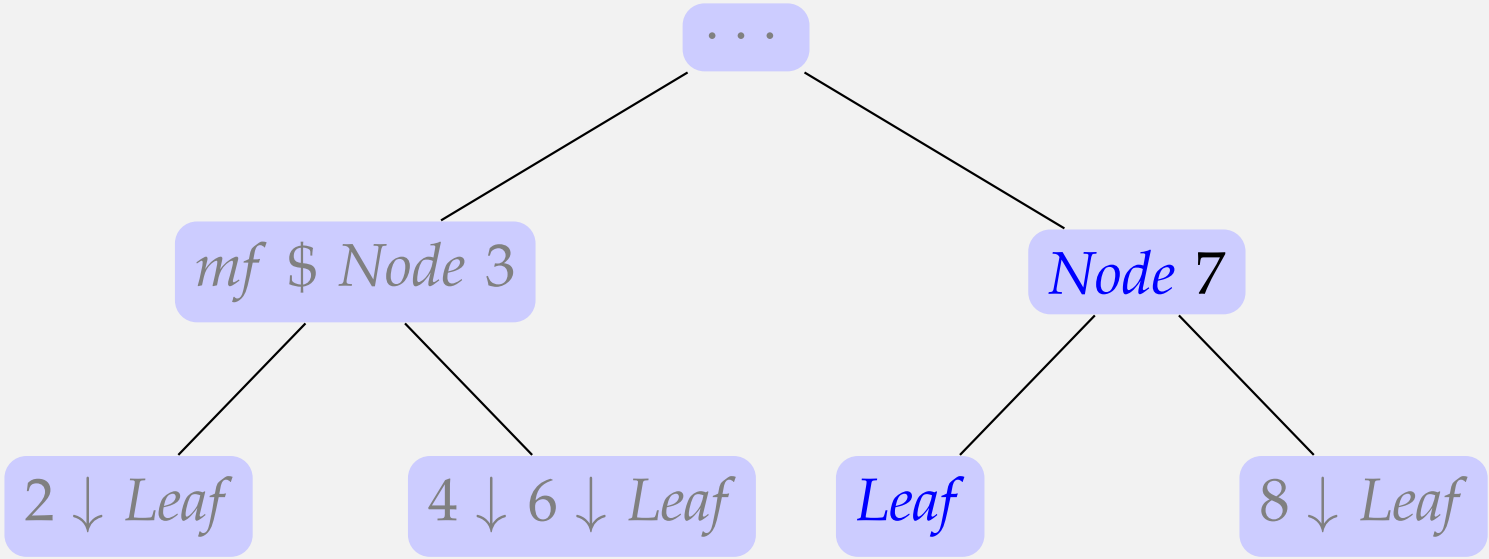
$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

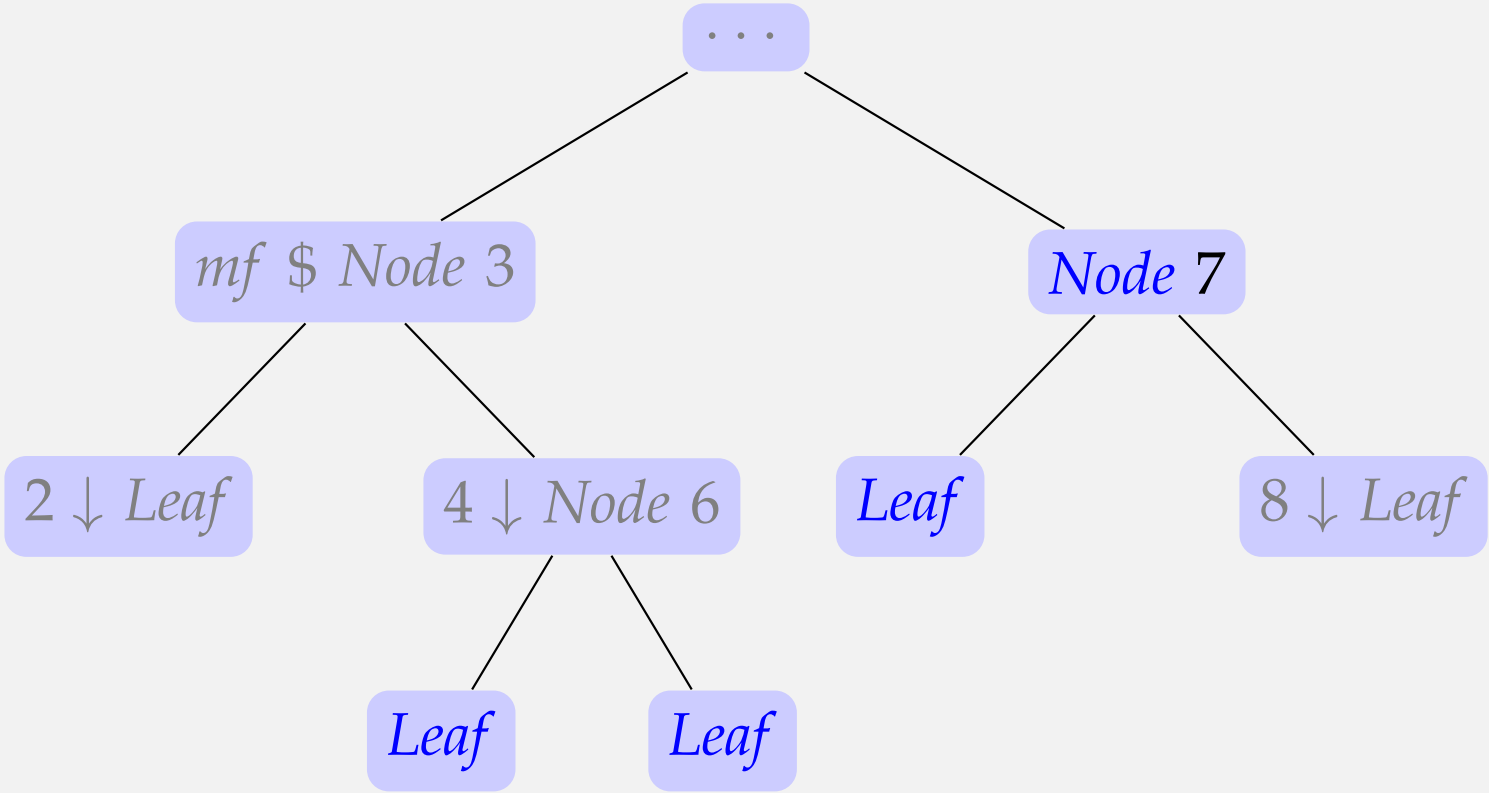
Zoekbomen: animatie



$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

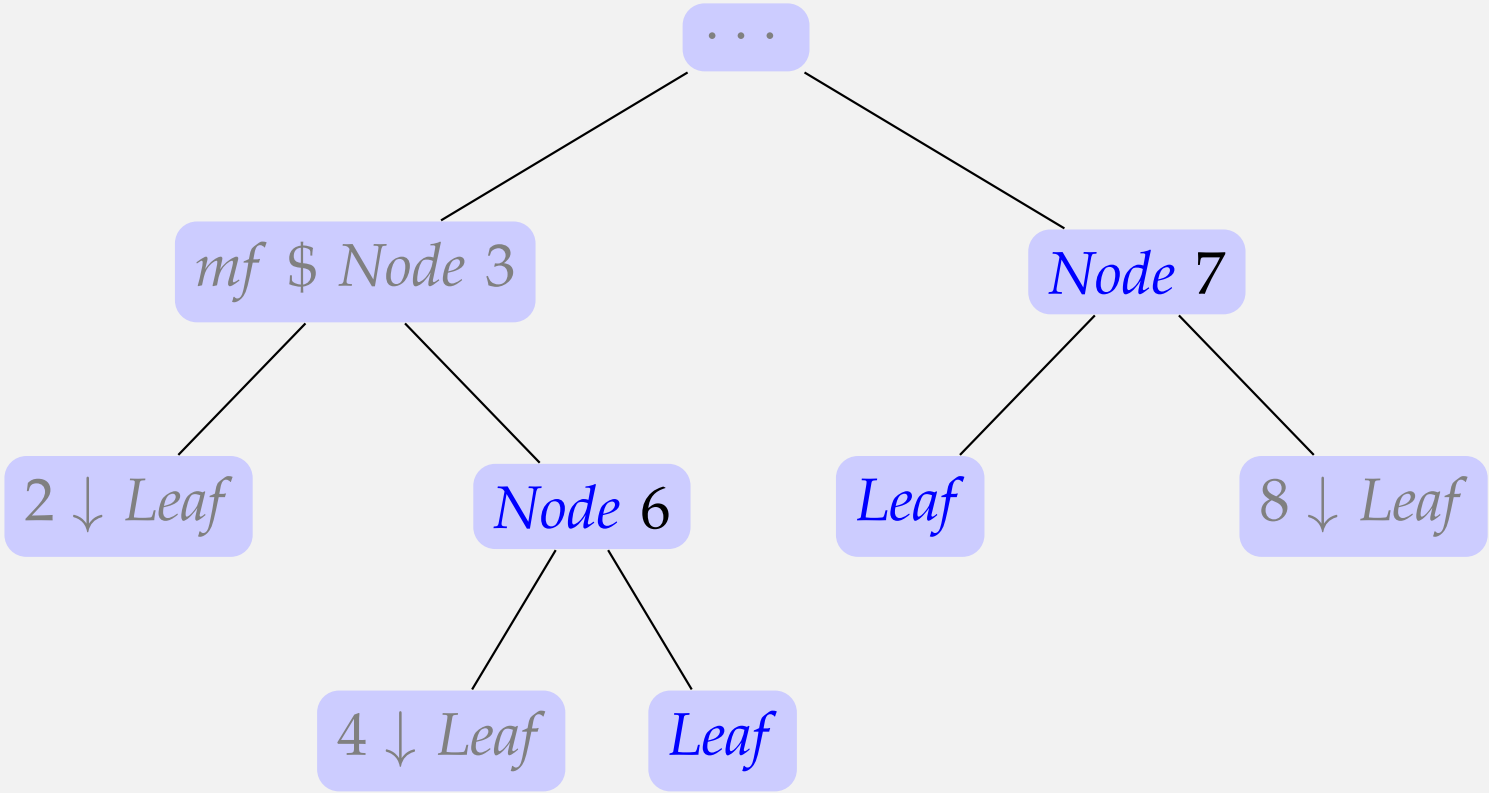
Zoekbomen: animatie



$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

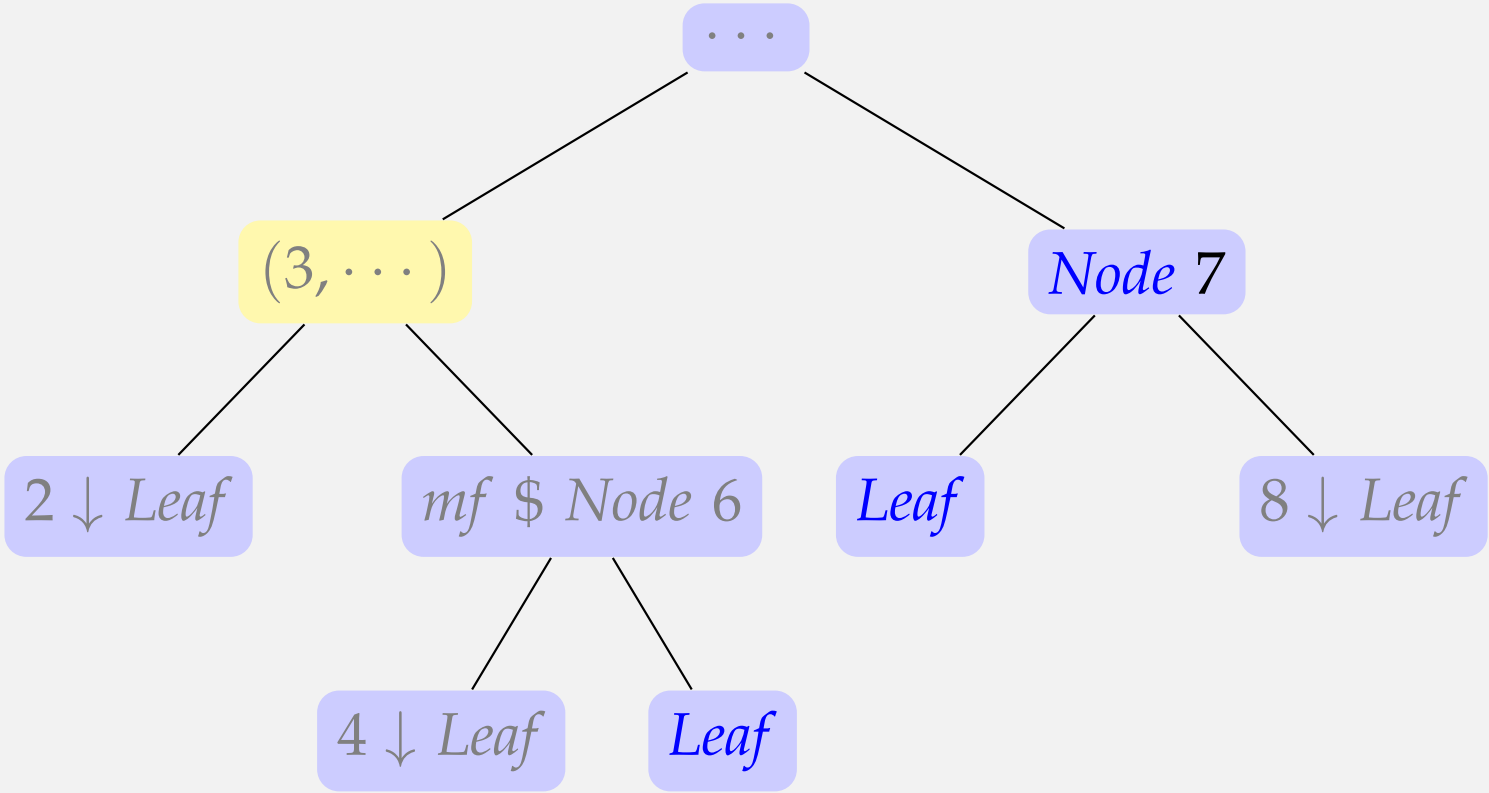
Zoekbomen: animatie



$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

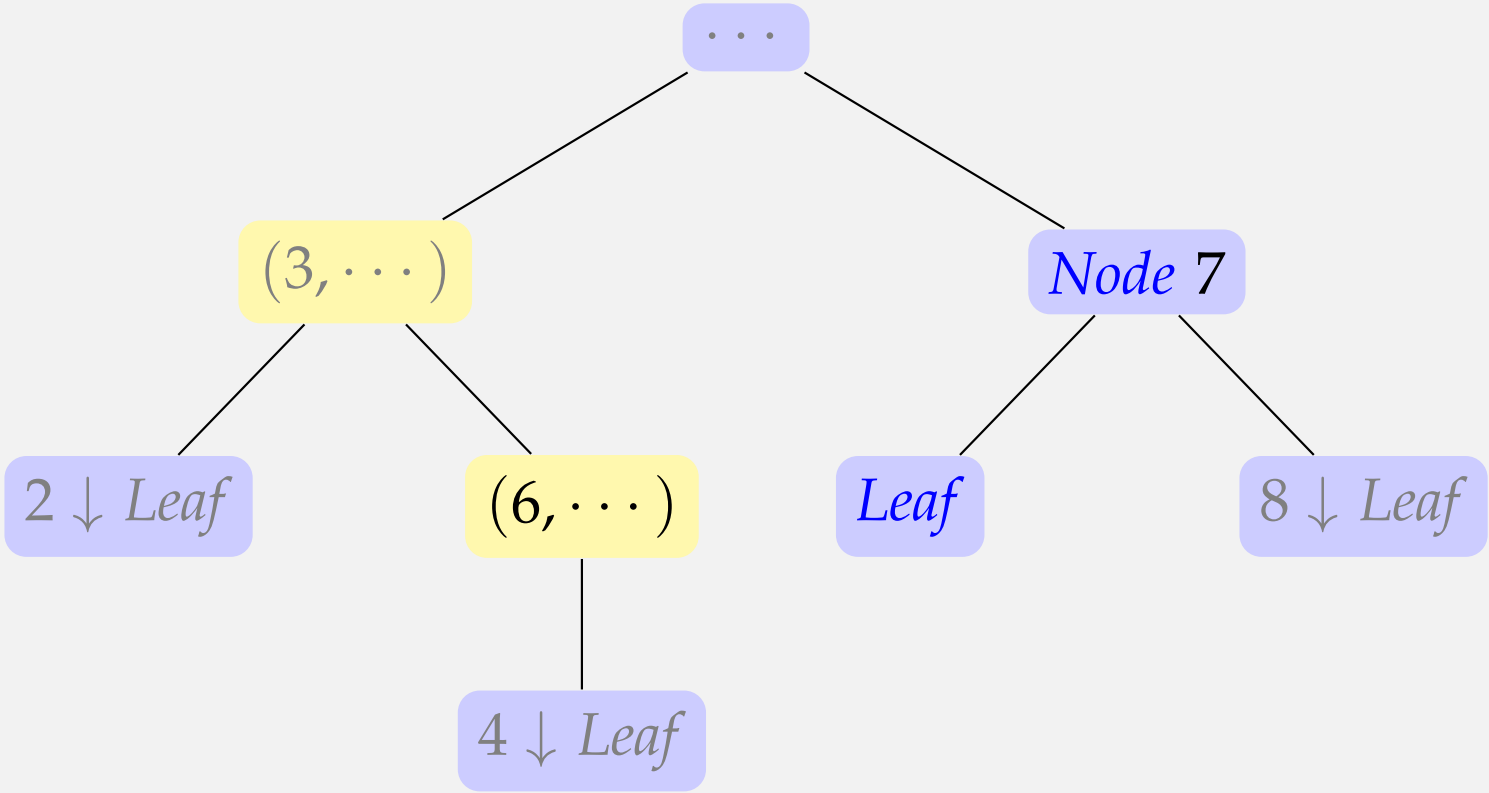
Zoekbomen: animatie



$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

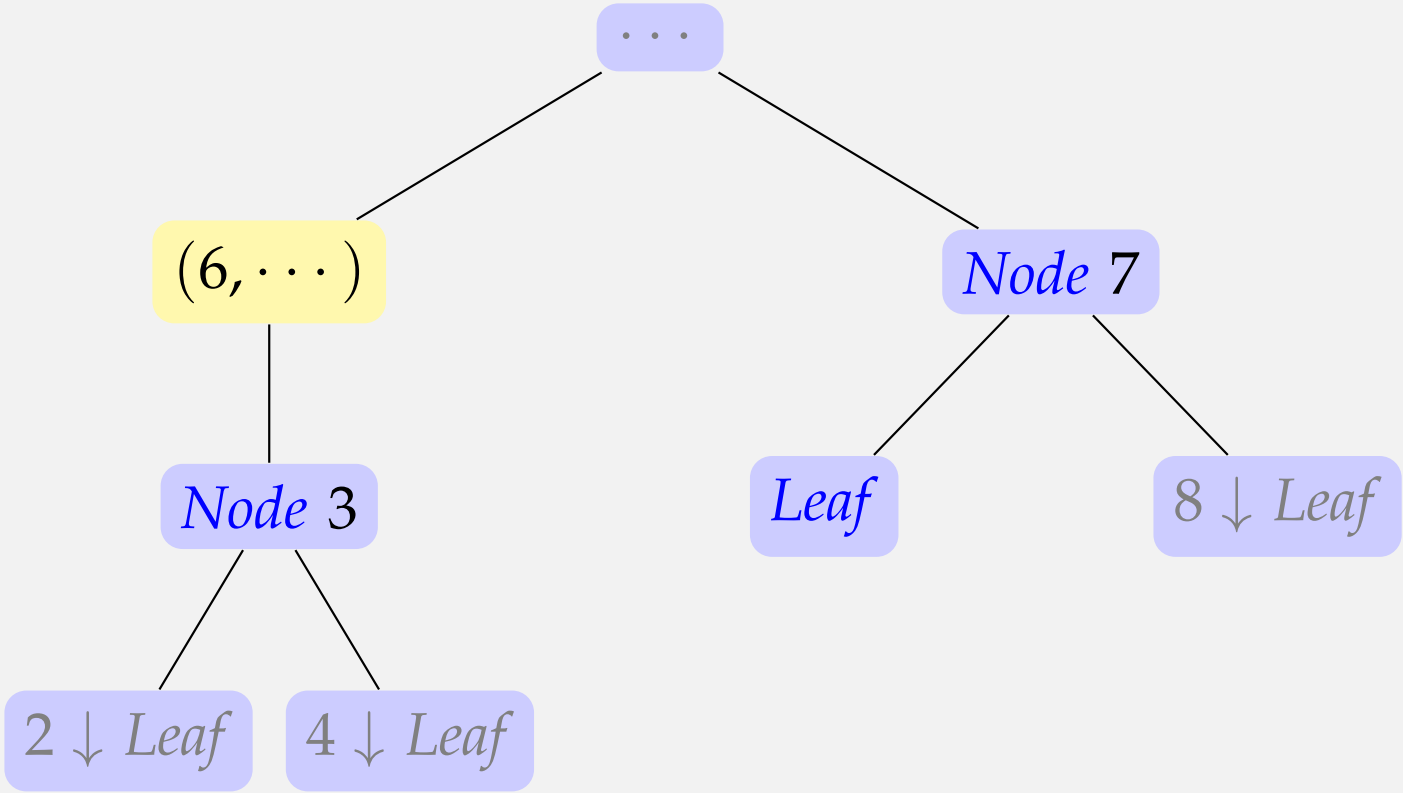
Zoekbomen: animatie



$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

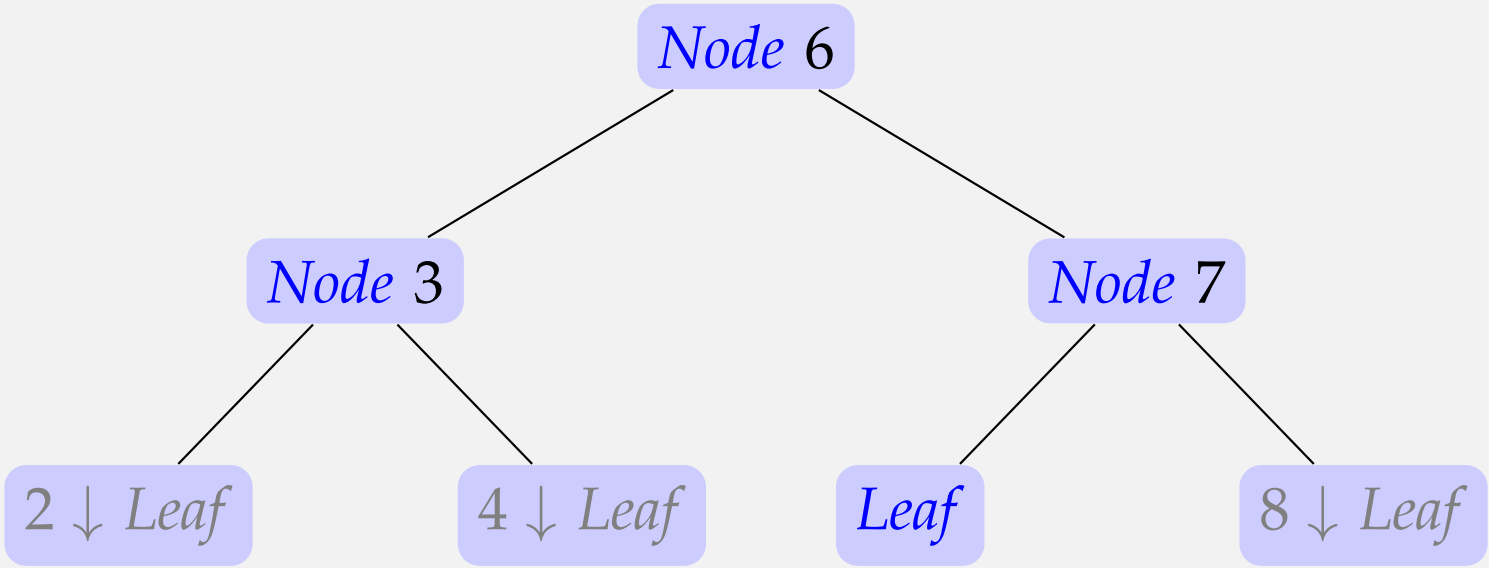
Zoekbomen: animatie



$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

Zoekbomen: animatie



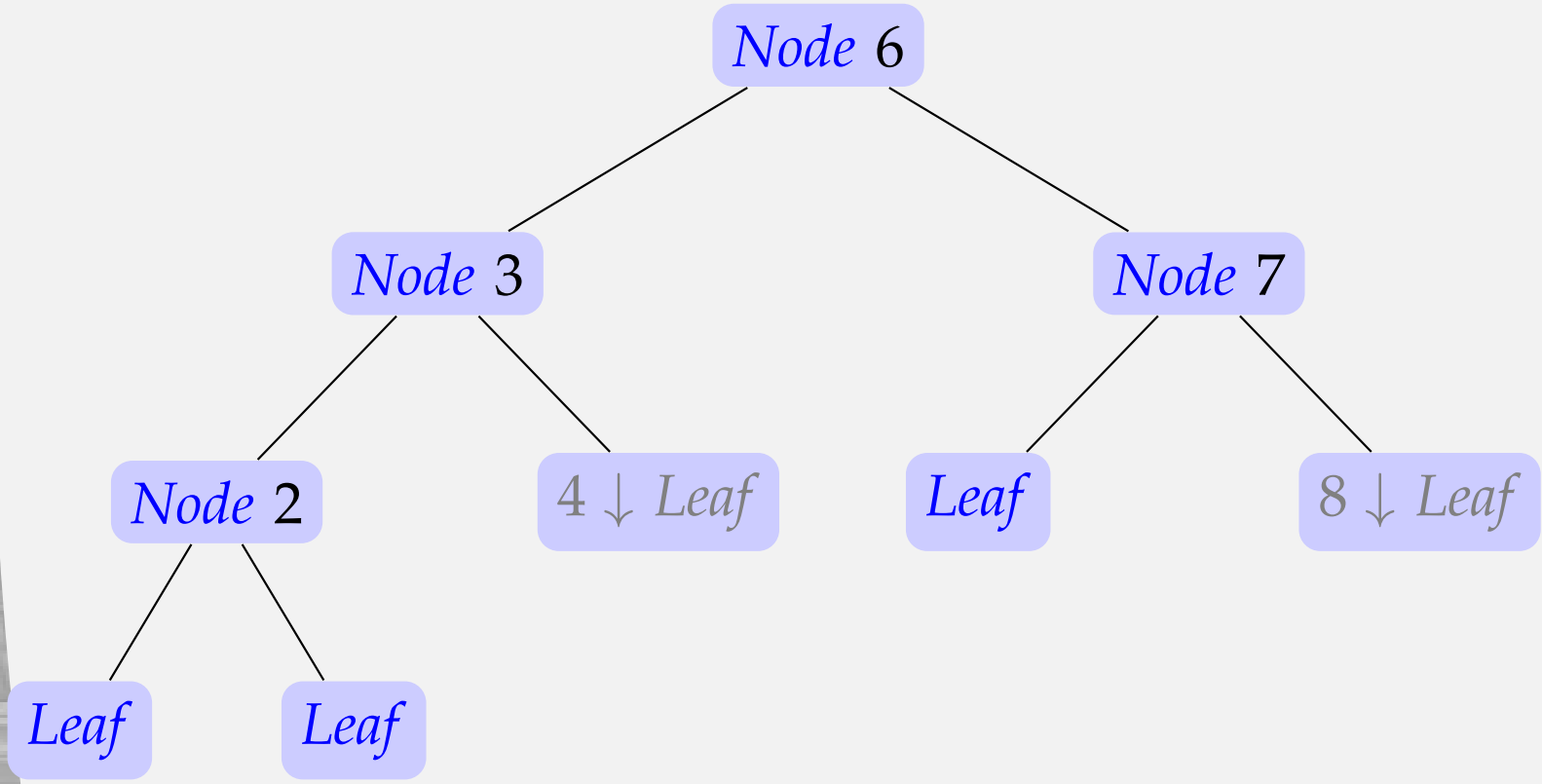
$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

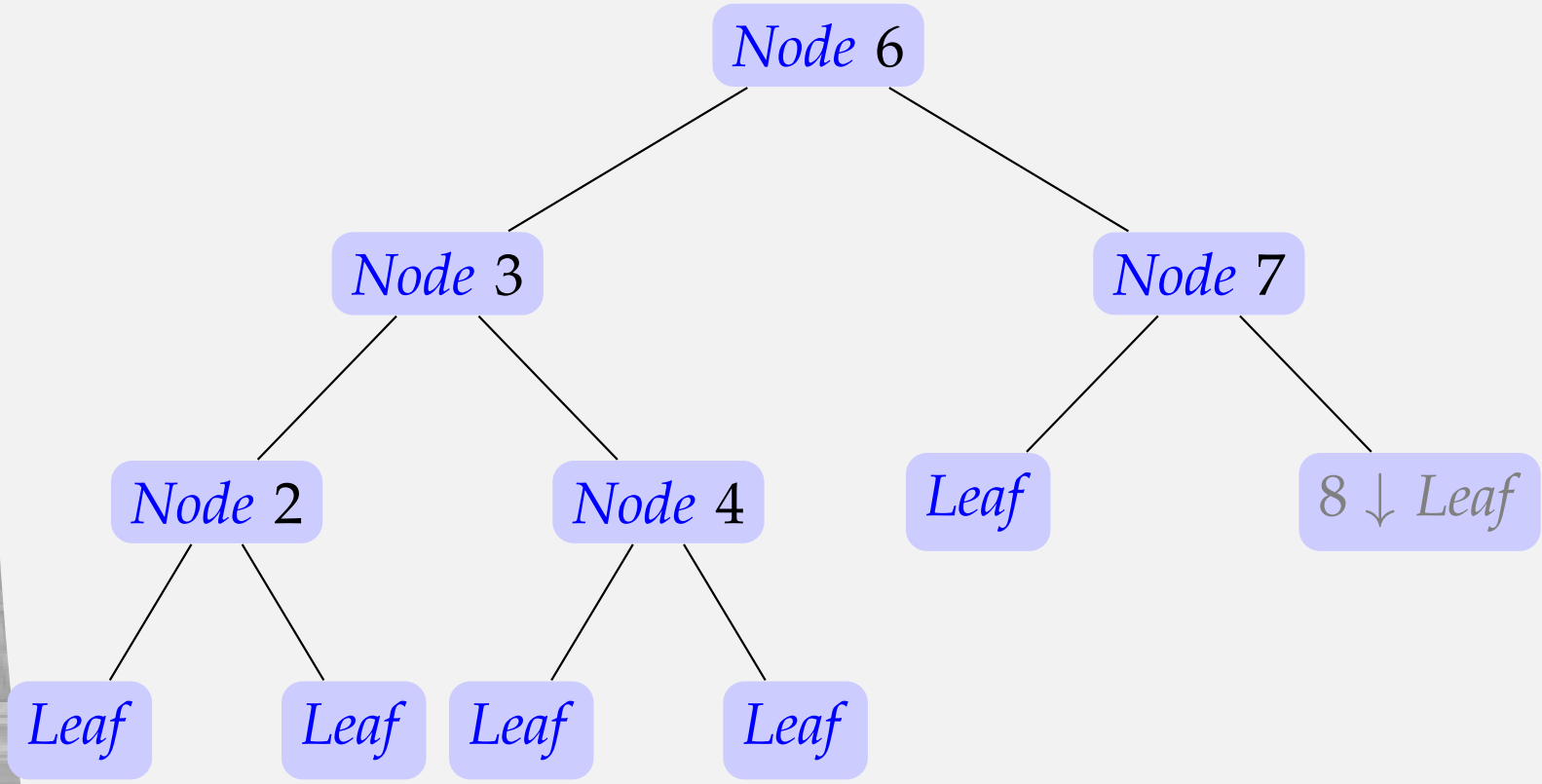
Zoekbomen: animatie



$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

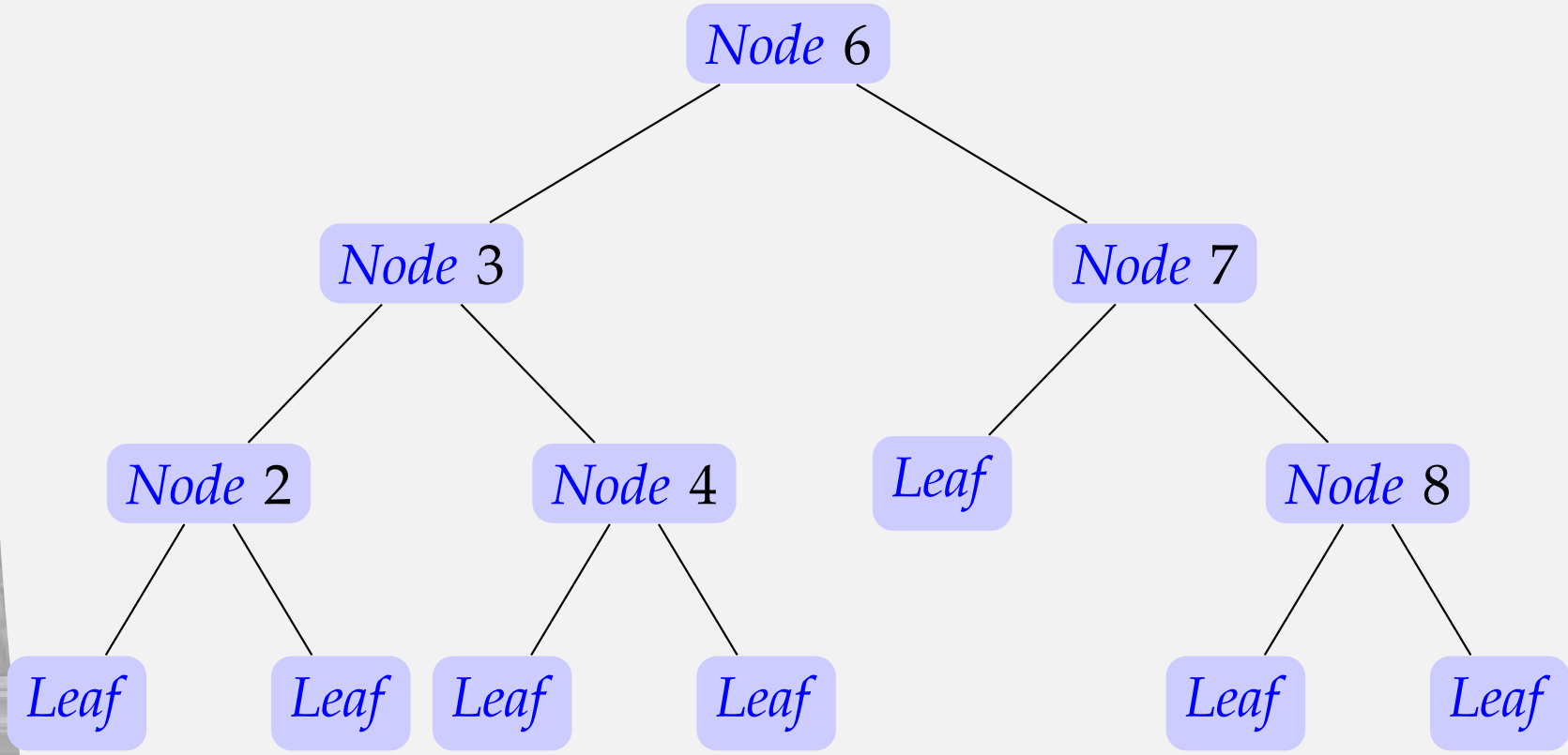
Zoekbomen: animatie



$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

Zoekbomen: animatie



$$(\downarrow) = insert_{Tree}$$
$$(\uparrow) = delete_{Tree}$$

$$(\bowtie) = join_{Tree}$$
$$mf = maxFrom$$

Het kan ook iets anders

$$
\begin{aligned}
&merge\ l\ r = (l \bowtie r)\ id \\
&((Node\ l\ x\ Leaf) \bowtie right)\ left = Node\ (left\ l)\ x\ right \\
&((Node\ l\ x\ r) \bowtie right)\ left = (r \bowtie right)\ (left \circ Node\ l\ x)
\end{aligned}
$$

Het kan ook iets anders

$$
\begin{aligned}
&merge\ l\ r = (l \bowtie r)\ id \\
&((Node\ l\ x\ Leaf) \bowtie right)\ left = Node\ (left\ l)\ x\ right \\
&((Node\ l\ x\ r) \bowtie right)\ left = (r \bowtie right)\ (left \circ Node\ l\ x)
\end{aligned}
$$

We hebben hier weer een accumulerende parameter ($left$) gebruikt, om het doorlopen stuk van de linker deelboom in te bewaren in de vorm van een functie die de nieuwe boom bouwt, zodra we zijn rechter onderhoek weten.

Scheve Bomen Voorkomen

$$groepeer :: [Tree\ a] \rightarrow [Tree\ a]$$

Scheve Bomen Voorkomen

Nog tenminste 4 elementen

$$groepeer :: [Tree\ a] \rightarrow [Tree\ a]$$
$$groepeer\ (l : Node\ \_\ v\ \_ : r : t : rest) = (Node\ l\ v\ r : t : groepeer\ rest)$$

Scheve Bomen Voorkomen

Klaar bij 3

$$
\begin{aligned}
&groepeer :: [Tree\ a] \rightarrow [Tree\ a] \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : t : rest) = (Node\ l\ v\ r : t : groepeer\ rest) \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : \quad [\,] \quad) = [Node\ l\ v\ r]
\end{aligned}
$$

Scheve Bomen Voorkomen

Een beetje scheef bij 2

$$
\begin{aligned}
&groepeer :: [Tree\ a] \rightarrow [Tree\ a] \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : t : rest) = (Node\ l\ v\ r : t : groepeer\ rest) \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : \quad [\,]\ \ ) = [Node\ l\ v\ r] \\
&groepeer\ (l : Node\ \_\ v\ r : \quad\ [\,]\ \ ) = [Node\ l\ v\ r]
\end{aligned}
$$

Scheve Bomen Voorkomen

Niets te doen bij 1 element

$$
\begin{aligned}
&groepeer :: [Tree\ a] \rightarrow [Tree\ a] \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : t : rest) = (Node\ l\ v\ r : t : groepeer\ rest) \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : \quad [\,]\quad) = [Node\ l\ v\ r] \\
&groepeer\ (l : Node\ \_\ v\ r : \quad [\,]\quad) = [Node\ l\ v\ r] \\
&groepeer\ (l : \quad [\,]\quad) = [l\quad]
\end{aligned}
$$

Scheve Bomen Voorkomen

$$
\begin{aligned}
&groepeer :: [Tree\ a] \rightarrow [Tree\ a] \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : t : rest) = (Node\ l\ v\ r : t : groepeer\ rest) \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : \quad [\,]\ ) = [Node\ l\ v\ r] \\
&groepeer\ (l : Node\ \_\ v\ r : \quad [\,]\ ) = [Node\ l\ v\ r] \\
&groepeer\ (l : \quad [\,]\ ) = [l\ ] \\
&verzamel = until\ ((\equiv 1) \circ length)\ groepeer
\end{aligned}
$$

Scheve Bomen Voorkomen

$$
\begin{aligned}
&groepeer :: [Tree\ a] \rightarrow [Tree\ a] \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : t : rest) = (Node\ l\ v\ r : t : groepeer\ rest) \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : \quad [\,]\ \ ) = [Node\ l\ v\ r] \\
&groepeer\ (l : Node\ \_\ v\ r : \quad [\,]\ \ ) = [Node\ l\ v\ r] \\
&groepeer\ (l : \quad [\,]\ \ ) = [l\ \ ] \\
&verzamel = until\ ((\equiv 1) \circ length)\ groepeer
\end{aligned}
$$

Nu kunnen we een boom bouwen door:

$$
\begin{aligned}
&lijst2Boom\ [\,] = Leaf \\
&lijst2Boom\ l\ \ = (\ldots)\ l
\end{aligned}
$$

Scheve Bomen Voorkomen

$$
\begin{aligned}
&groepeer :: [Tree\ a] \to [Tree\ a] \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : t : rest) = (Node\ l\ v\ r : t : groepeer\ rest) \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : \quad [\,]\ \ ) = [Node\ l\ v\ r] \\
&groepeer\ (l : Node\ \_\ v\ r : \quad\ [\,]\ \ ) = [Node\ l\ v\ r] \\
&groepeer\ (l : \quad\ [\,]\ \ ) = [l\ \ ] \\
&verzamel = until\ ((\equiv 1) \circ length)\ groepeer
\end{aligned}
$$

Nu kunnen we een boom bouwen door:

$$
\begin{aligned}
&lijst2Boom\ [\,] = Leaf \\
&lijst2Boom\ l\ \ = (\ldots \circ sort)\ l
\end{aligned}
$$

Scheve Bomen Voorkomen

$$
\begin{aligned}
&groepeer :: [Tree\ a] \rightarrow [Tree\ a] \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : t : rest) = (Node\ l\ v\ r : t : groepeer\ rest) \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : \quad [\,]\ \ ) = [Node\ l\ v\ r] \\
&groepeer\ (l : Node\ \_\ v\ r : \quad\ [\,]\ \ ) = [Node\ l\ v\ r] \\
&groepeer\ (l : \quad\ [\,]\ \ ) = [l\ \ ] \\
&verzamel = until\ ((\equiv 1) \circ length)\ groepeer
\end{aligned}
$$

Nu kunnen we een boom bouwen door:

$$
\begin{aligned}
&lijst2Boom\ [\,] = Leaf \\
&lijst2Boom\ l\ \ = (\ldots map\ (\lambda v \rightarrow Node\ Leaf\ v\ Leaf) \circ sort)\ l
\end{aligned}
$$

Scheve Bomen Voorkomen

$$
\begin{aligned}
&groepeer :: [Tree\ a] \to [Tree\ a] \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : t : rest) = (Node\ l\ v\ r : t : groepeer\ rest) \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : \quad [\,]\ ) = [Node\ l\ v\ r] \\
&groepeer\ (l : Node\ \_\ v\ r : \quad [\,]\ ) = [Node\ l\ v\ r] \\
&groepeer\ (l : \quad [\,]\ ) = [l\ ] \\
&verzamel = until\ ((\equiv 1) \circ length)\ groepeer
\end{aligned}
$$

Nu kunnen we een boom bouwen door:

$$
\begin{aligned}
&lijst2Boom\ [\,] = Leaf \\
&lijst2Boom\ l\ = (\ldots verzamel \circ map\ (\lambda v \to Node\ v\ Leaf\ Leaf) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \circ sort)\ l
\end{aligned}
$$

Scheve Bomen Voorkomen

$$
\begin{aligned}
&groepeer :: [Tree\ a] \to [Tree\ a] \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : t : rest) = (Node\ l\ v\ r : t : groepeer\ rest) \\
&groepeer\ (l : Node\ \_\ v\ \_ : r : \quad [\,]\ ) = [Node\ l\ v\ r] \\
&groepeer\ (l : Node\ \_\ v\ r : \quad [\,]\ ) = [Node\ l\ v\ r] \\
&groepeer\ (l : \quad [\,]\ ) = [l\ ] \\
&verzamel = until\ ((\equiv 1) \circ length)\ groepeer
\end{aligned}
$$

Nu kunnen we een boom bouwen door:

$$
\begin{aligned}
&lijst2Boom\ [\,] = Leaf \\
&lijst2Boom\ l\ = (head \circ verzamel \circ map\ (\lambda v \to Node\ Leaf\ v\ Leaf) \\
&\qquad\qquad\qquad\qquad \circ sort)\ l
\end{aligned}
$$