

# **Adatt Source Files**

Sat Jun 5 18:44:00 2021

# Contents

<b>1</b>	<b>Files</b>	<b>4</b>
1.1	Data/Collection/DistinctListSet.ad . . . . .	4
1.2	Data/Collection/FunSetBool.ad . . . . .	6
1.3	Data/Collection/ListBag.ad . . . . .	8
1.4	Data/Collection/Queue.ad . . . . .	12
1.5	Data/Collection/Stack.ad . . . . .	14
1.6	Data/Lens.ad . . . . .	16
1.7	Data/List.ad . . . . .	22
1.8	Data/Maybe.ad . . . . .	24
1.9	Prelude.ad . . . . .	24
1.10	Prelude/BasicClasses.ad . . . . .	25
1.11	Prelude/Bool.ad . . . . .	32
1.12	Prelude/Char.ad . . . . .	34
1.13	Prelude/Collection.ad . . . . .	35
1.14	Prelude/Default.ad . . . . .	40
1.15	Prelude/Either.ad . . . . .	41
1.16	Prelude/Eq.ad . . . . .	42
1.17	Prelude/Foldable.ad . . . . .	45
1.18	Prelude/Function.ad . . . . .	50
1.19	Prelude/Functor.ad . . . . .	51
1.20	Prelude/List.ad . . . . .	51
1.21	Prelude/ListCollection.ad . . . . .	57
1.22	Prelude/Literal.ad . . . . .	58
1.23	Prelude/Maybe.ad . . . . .	62
1.24	Prelude/Monad.ad . . . . .	63
1.25	Prelude/Num.ad . . . . .	66
1.26	Prelude/Ord.ad . . . . .	69
1.27	Prelude/String.ad . . . . .	72
1.28	Prelude/Template.ad . . . . .	72
1.29	Prelude/Text.ad . . . . .	77
1.30	Prelude/Tuple.ad . . . . .	78
1.31	Prelude/Unit.ad . . . . .	81
<b>2</b>	<b>Modules</b>	<b>82</b>
<b>3</b>	<b>Indices</b>	<b>85</b>
3.1	Classes . . . . .	85
3.1.1	B . . . . .	85
3.1.2	C . . . . .	86
3.1.3	D . . . . .	88
3.1.4	E . . . . .	88
3.1.5	F . . . . .	94
3.1.6	I . . . . .	97
3.1.7	L . . . . .	97
3.1.8	M . . . . .	98
3.1.9	N . . . . .	99
3.1.10	O . . . . .	100
3.1.11	S . . . . .	101
3.2	Constants . . . . .	102
3.2.1	( . . . . .	102
3.2.2	A . . . . .	106
3.2.3	B . . . . .	108

Contents

3.2.4	C	108
3.2.5	D	111
3.2.6	E	113
3.2.7	F	115
3.2.8	G	118
3.2.9	I	119
3.2.10	J	120
3.2.11	L	121
3.2.12	M	122
3.2.13	N	125
3.2.14	O	126
3.2.15	P	127
3.2.16	R	128
3.2.17	S	129
3.2.18	T	132
3.2.19	U	133
3.2.20	V	134
3.2.21	Z	135
3.3	Templates	135
3.3.1	B	135
3.3.2	C	136
3.3.3	D	136
3.3.4	G	136
3.3.5	I	137
3.3.6	T	137
3.4	Types	137
3.4.1	B	137
3.4.2	C	138
3.4.3	D	138
3.4.4	E	138
3.4.5	F	138
3.4.6	G	138
3.4.7	I	138
3.4.8	L	139
3.4.9	M	139
3.4.10	N	139
3.4.11	O	140
3.4.12	P	140
3.4.13	Q	140
3.4.14	S	140
3.4.15	T	141
3.4.16	U	142

# 1 Files

## 1.1 Data/Collection/DistinctListSet.ad

### Outline

- module Data.Collection.DistinctListSet ..... lines 1 - 96
  - DistinctListSet (datatype) ..... line 7
  - Semigroup (DistinctListSet 'e) (typeclass-instance) ..... lines 9 - 12
    - \*  $\langle \rangle$  (definition) ..... lines 10, 11
  - Monoid (DistinctListSet 'e) (typeclass-instance) ..... lines 14 - 16
    - \* empty (definition) ..... line 15
  - Collection (DistinctListSet 'e) 'e (typeclass-instance) ..... lines 18 - 27
    - \* insert (definition) ..... lines 19, 20
    - \* fromList (definition) ..... line 22
    - \* isEmptyP (definition) ..... line 25
    - \* memberP (definition) ..... line 26
  - CollectionWithMembershipTest (DistinctListSet 'e) 'e (typeclass-instance) .. lines 29 - 32
    - \* occur (definition) ..... line 30
    - \* member (definition) ..... line 31
  - Foldable (DistinctListSet 'e) 'e (typeclass-instance) ..... lines 34 - 43
    - \* toList (definition) ..... line 35
    - \* foldl (definition) ..... line 38
    - \* foldr (definition) ..... line 39
    - \* size (definition) ..... line 40
    - \* null (definition) ..... line 41
    - \* elem (definition) ..... line 42
  - CollectionWithEmptynessTest (DistinctListSet 'e) 'e (typeclass-instance) .. lines 46 - 49
    - \* isEmpty (definition) ..... line 47
  - CollectionWithDelete (DistinctListSet 'e) 'e (typeclass-instance) ..... lines 52 - 70
    - \* delete (definition) ..... line 53
    - \* filter (definition) ..... line 55
    - \* partition (definition) ..... lines 57 - 62
    - \* difference (definition) ..... lines 64, 65
    - \* intersection (definition) ..... lines 67, 68
  - CollectionWithSubsetTest (DistinctListSet 'e) 'e (typeclass-instance) .... lines 72 - 82
    - \* isSubsetOf (definition) ..... lines 73, 74
    - \* isProperSubsetOf (definition) ..... line 76
    - \* isSetEquiv (definition) ..... line 78
    - \* disjoint (definition) ..... lines 80, 81
  - FiniteCollection (DistinctListSet 'e) 'e (typeclass-instance) ..... lines 84, 85
  - SetLike (DistinctListSet 'e) 'e (typeclass-instance) ..... lines 87, 88
  - Set (DistinctListSet 'e) 'e (typeclass-instance) ..... lines 90, 91
  - FiniteSet (DistinctListSet 'e) 'e (typeclass-instance) ..... lines 93, 94

## Content

```

1 module Data.Collection.DistinctListSet (
2   DistinctListSet(..)
3 ) where
4
5 import qualified Data.List as List
6
7 datatype DistinctListSet 'a := DistinctListSet ['a]
8
9 instance Eq 'e => Semigroup (DistinctListSet 'e) where
10   define (<>) (DistinctListSet l1) (DistinctListSet l2) :=
11     DistinctListSet (l1 <> (List.filter (fn x -> not (List.elem x l1)) l2))
12 end-instance
13
14 instance Eq 'e => Monoid (DistinctListSet 'e) where
15   define mempty := DistinctListSet []
16 end-instance
17
18 instance Eq 'e => Collection (DistinctListSet 'e) 'e where
19   define insert e (DistinctListSet l) :=
20     DistinctListSet (if elem e l then l else e:l)
21
22   define fromList l := DistinctListSet (List.nub l)
23   {-# inline fromList #-}
24
25   define non-exec isEmptyP (DistinctListSet l) := bool2prop (null l)
26   define non-exec memberP e (DistinctListSet l) := List.elemP e l
27 end-instance
28
29 instance Eq 'e => CollectionWithMembershipTest (DistinctListSet 'e) 'e where
30   define occur e dl := if member e dl then 1 else 0
31   define member e (DistinctListSet l) := elem e l
32 end-instance
33
34 instance Foldable (DistinctListSet 'e) 'e where
35   define toList (DistinctListSet dl) := dl
36   {-# inline toList #-}
37
38   define foldl f acc (DistinctListSet dl) := List.foldl f acc dl
39   define foldr f acc (DistinctListSet dl) := List.foldr f acc dl
40   define size (DistinctListSet dl) := List.length dl
41   define null (DistinctListSet dl) := List.null dl
42   define elem e (DistinctListSet dl) := List.elem e dl
43 end-instance
44
45
46 instance Eq 'e => CollectionWithEmptynessTest (DistinctListSet 'e) 'e where
47   define isEmpty := null
48   {-# inline isEmpty #-}
49 end-instance
50
51
52 instance Eq 'e => CollectionWithDelete (DistinctListSet 'e) 'e where
53   define delete e (DistinctListSet l) := DistinctListSet (List.removeFirst ((==) e) l)
54
55   define filter p (DistinctListSet l) := DistinctListSet (List.filter p l)
56
57   define partition p (DistinctListSet l) :=

```

```

58   let
59     val (l1, l2) = List.partition p l
60   in
61     (DistinctListSet l1, DistinctListSet l2)
62   end
63
64   define difference (DistinctListSet l1) (DistinctListSet l2) :=
65     DistinctListSet (List.filter (fn x -> not (List.elem x l2)) l1)
66
67   define intersection (DistinctListSet l1) (DistinctListSet l2) :=
68     DistinctListSet (List.filter (fn x -> List.elem x l2) l1)
69
70   end-instance
71
72   instance Eq 'e => CollectionWithSubsetTest (DistinctListSet 'e) 'e where
73     define isSubsetOf (DistinctListSet l1) (DistinctListSet l2) :=
74       List.all (fn x -> List.elem x l2) l1
75
76     define isProperSubsetOf xs ys := isSubsetOf xs ys && (size xs < size ys)
77
78     define isSetEquiv xs ys := isSubsetOf xs ys && (size xs == size ys)
79
80     define disjoint (DistinctListSet l1) (DistinctListSet l2) :=
81       List.all (fn x -> not (List.elem x l2)) l1
82   end-instance
83
84   instance Eq 'e => FiniteCollection (DistinctListSet 'e) 'e where
85   end-instance
86
87   instance Eq 'e => SetLike (DistinctListSet 'e) 'e where
88   end-instance
89
90   instance Eq 'e => Set (DistinctListSet 'e) 'e where
91   end-instance
92
93   instance Eq 'e => FiniteSet (DistinctListSet 'e) 'e where
94   end-instance
95
96   end-module

```

## 1.2 Data/Collection/FunSetBool.ad

### Outline

- module Data.Collection.FunSetBool ..... lines 1 - 52
  - FunSetBool (datatype) ..... line 6
  - Semigroup (FunSetBool 'e) (typeclass-instance) ..... lines 8 - 11
    - \* <> (definition) ..... lines 9, 10
  - Monoid (FunSetBool 'e) (typeclass-instance) ..... lines 13 - 15
    - \* mempty (definition) ..... line 14
  - Collection (FunSetBool 'e) 'e (typeclass-instance) ..... lines 17 - 27
    - \* insert (definition) ..... lines 18, 19
    - \* fromList (definition) ..... line 21
    - \* isEmptyP (definition) ..... lines 24, 25
    - \* memberP (definition) ..... line 26

- CollectionWithMembershipTest (FunSetBool 'e) 'e (typeclass-instance) . . . . .	lines 29 - 32
* occur (definition) . . . . .	line 30
* member (definition) . . . . .	line 31
- CollectionWithDelete (FunSetBool 'e) 'e (typeclass-instance) . . . . .	lines 34 - 47
* delete (definition) . . . . .	line 35
* filter (definition) . . . . .	line 37
* partition (definition) . . . . .	line 39
* difference (definition) . . . . .	lines 41, 42
* intersection (definition) . . . . .	lines 44, 45
- SetLike (FunSetBool 'e) 'e (typeclass-instance) . . . . .	lines 49, 50

## Content

```

1 module Data.Collection.FunSetBool (
2   FunSetBool(..)
3 ) where
4
5 -- A datatype of sets given by its characteristic function
6 datatype FunSetBool 'a := FunSetBool ('a -> Bool)
7
8 instance Eq 'e => Semigroup (FunSetBool 'e) where
9   define (<>) (FunSetBool f1) (FunSetBool f2) :=
10     FunSetBool (fn x -> f1 x || f2 x)
11 end-instance
12
13 instance Eq 'e => Monoid (FunSetBool 'e) where
14   define mempty := FunSetBool (constant False)
15 end-instance
16
17 instance Eq 'e => Collection (FunSetBool 'e) 'e where
18   define insert e (FunSetBool f) :=
19     FunSetBool (fn x -> (x == e) || f x)
20
21   define fromList l := FunSetBool (fn x -> elem x l)
22   {-# inline fromList #-}
23
24   define non-exec isEmptyP (FunSetBool f) :=
25     (forallP e. bool2prop (not (f e)))
26   define non-exec memberP e (FunSetBool f) := bool2prop (f e)
27 end-instance
28
29 instance Eq 'e => CollectionWithMembershipTest (FunSetBool 'e) 'e where
30   define occur e dl := if member e dl then 1 else 0
31   define member e (FunSetBool f) := f e
32 end-instance
33
34 instance Eq 'e => CollectionWithDelete (FunSetBool 'e) 'e where
35   define delete e (FunSetBool f) := FunSetBool (fn x -> (x /= e) && f e)
36
37   define filter p (FunSetBool f) := FunSetBool (fn x -> (p x) && (f x))
38
39   define partition p s := (filter p s, filter (not . p) s)
40
41   define difference (FunSetBool f1) (FunSetBool f2) :=
42     FunSetBool (fn x -> f1 x && not (f2 x))
43

```

```

44 define intersection (FunSetBool f1) (FunSetBool f2) :=
45   FunSetBool (fn x -> (f1 x) && (f2 x))
46
47 end-instance
48
49 instance Eq 'e => SetLike (FunSetBool 'e) 'e where
50 end-instance
51
52 end-module

```

## 1.3 Data/Collection/ListBag.ad

### Outline

- module Data.Collection.ListBag ..... lines 1 - 155
  - ListBag (datatype) ..... line 8
  - emptyListBag (declaration + definition) ..... lines 10, 11
  - Functor ListBag (typeclass-instance) .....lines 13 - 15
    - \* fmap (definition) .....line 14
  - Semigroup (ListBag 'a) (typeclass-instance) .....lines 17 - 19
    - \* <> (definition) .....line 18
  - Monoid (ListBag 'a) (typeclass-instance) .....lines 21 - 24
    - \* mempty (definition) .....line 22
  - CollectionWithEmptinessTest (ListBag 'e) 'e (typeclass-instance) .....lines 26 - 28
    - \* isEmpty (definition) .....line 27
  - Collection (ListBag 'e) 'e (typeclass-instance) .....lines 30 - 41
    - \* insert (definition) .....line 31
    - \* fromList (definition) .....line 34
    - \* isEmptyP (definition) .....line 37
    - \* memberP (definition) .....line 38
  - Foldable (ListBag 'e) 'e (typeclass-instance) .....lines 43 - 45
    - \* toList (definition) .....line 44
  - CollectionWithMembershipTest (ListBag 'e) 'e (typeclass-instance) .....lines 47 - 54
    - \* occur (definition) .....line 48
    - \* member (definition) .....line 51
  - FiniteCollection (ListBag 'e) 'e (typeclass-instance) ..... lines 56, 57
  - listIntersection (declaration + definition) .....lines 61 - 68
  - CollectionWithDelete (ListBag 'e) 'e (typeclass-instance) .....lines 70 - 90
    - \* delete (definition) .....line 71
    - \* filter (definition) .....line 74
    - \* partition (definition) .....lines 77 - 82
    - \* difference (definition) .....line 85
    - \* intersection (definition) .....line 88
  - listSubsetOf (declaration + definition) ..... lines 93 - 100
  - listSetEquiv (declaration + definition) ..... lines 103 - 111
  - listDisjoint (declaration + definition) ..... lines 113 - 120
  - CollectionWithSubsetTest (ListBag 'e) 'e (typeclass-instance) ..... lines 123 - 135
    - \* isSubsetOf (definition) .....line 124



* isProperSubsetOf (definition) .....	line 127
* isSetEquiv (definition) .....	line 130
* disjoint (definition) .....	line 133
- EqP (ListBag 'e) (typeclass-instance) .....	lines 137 - 139
* === (definition) .....	line 138
- Eq (ListBag 'e) (typeclass-instance) .....	lines 141 - 144
* == (definition) .....	line 142
- BagLike (ListBag 'e) 'e (typeclass-instance) .....	lines 146, 147
- Bag (ListBag 'e) 'e (typeclass-instance) .....	lines 149, 150
- FiniteBag (ListBag 'e) 'e (typeclass-instance) .....	lines 152, 153

## Content

```

1 module Data.Collection.ListBag (
2   ListBag
3 ) where
4
5 import qualified Data.List as List
6
7
8 datatype ListBag 'a := ListBag ['a]
9
10 declare emptyListBag :: ListBag 'a
11 define emptyListBag := ListBag []
12
13 instance Functor ListBag where
14   define fmap f (ListBag l) := ListBag (map f l)
15 end-instance
16
17 instance Semigroup (ListBag 'a) where
18   define (<>) (ListBag xs) (ListBag ys) := ListBag (xs ++ ys)
19 end-instance
20
21 instance Monoid (ListBag 'a) where
22   define mempty := emptyListBag
23   {-# inline mempty #-}
24 end-instance
25
26 instance CollectionWithEmptynessTest (ListBag 'e) 'e where
27   define isEmpty (ListBag l) := (null l)
28 end-instance
29
30 instance Collection (ListBag 'e) 'e where
31   define insert e (ListBag l) := ListBag (e:l)
32   {-# inline insert #-}
33
34   define fromList l := ListBag l
35   {-# inline fromList #-}
36
37   define non-exec isEmptyP (ListBag l) := bool2prop (null l)
38   define non-exec memberP e (ListBag l) := List.elemP e l
39   {-# inline memberP #-}
40
41 end-instance
42
43 instance Foldable (ListBag 'e) 'e where

```

```

44   define toList (ListBag l) := l
45 end-instance
46
47 instance CollectionWithMembershipTest (ListBag 'e) 'e where
48   define occur e (ListBag l) := count e l
49   {-# inline occur #-}
50
51   define member := elem
52   {-# inline member #-}
53
54 end-instance
55
56 instance Eq 'e => FiniteCollection (ListBag 'e) 'e where
57 end-instance
58
59
60
61 declare listIntersection :: Eq 'a => ['a] -> ['a] -> ['a]
62 define rec listIntersection [] _ := []
63   | listIntersection _ [] := []
64   | listIntersection (x:xs) l :=
65     case (takeFirst ((==) x) l) of
66       Nothing -> listIntersection xs l
67       | Just (l', _) -> x:(listIntersection xs l')
68     end
69
70 instance Eq 'e => CollectionWithDelete (ListBag 'e) 'e where
71   define delete e (ListBag l) := ListBag (removeFirst ((==) e) l)
72   {-# inline delete #-}
73
74   define filter p (ListBag l) := ListBag (filter p l)
75   {-# inline filter #-}
76
77   define partition p (ListBag l) :=
78     let
79       val (lt, lf) = List.partition p l
80     in
81       (ListBag lt, ListBag lf)
82     end
83   {-# inline partition #-}
84
85   define difference := foldl (fn l e -> delete e l)
86   {-# inline difference #-}
87
88   define intersection (ListBag l1) (ListBag l2) := ListBag (listIntersection l1 l2)
89   {-# inline intersection #-}
90 end-instance
91
92
93 declare listSubsetOf :: Eq 'a => ['a] -> ['a] -> Bool
94 define rec listSubsetOf [] _ := True
95   | listSubsetOf _ [] := False
96   | listSubsetOf (x:xs) l :=
97     case (takeFirst ((==) x) l) of
98       Nothing -> False
99       | Just (l', _) -> listSubsetOf xs l'
100     end
101
102

```

```

103 declare listSetEquiv :: Eq 'a => ['a] -> ['a] -> Bool
104 define rec listSetEquiv [] [] := True
105     | listSetEquiv [] _ := False
106     | listSetEquiv _ [] := False
107     | listSetEquiv (x:xs) l :=
108         case (takeFirst ((==) x) l) of
109             Nothing -> False
110             | Just (l', _) -> listSetEquiv xs l'
111     end
112
113 declare listDisjoint :: Eq 'a => ['a] -> ['a] -> Bool
114 define rec listDisjoint [] _ := True
115     | listDisjoint _ [] := True
116     | listDisjoint (x:xs) l :=
117         case (takeFirst ((==) x) l) of
118             Nothing -> listDisjoint xs l
119             | Just _ -> False
120     end
121
122
123 instance Eq 'e => CollectionWithSubsetTest (ListBag 'e) 'e where
124     define isSubsetOf (ListBag xs) (ListBag ys) := listSubsetOf xs ys
125     {-# inline isSubsetOf #-}
126
127     define isProperSubsetOf (ListBag xs) (ListBag ys) := listSubsetOf xs ys && (length xs < length ys)
128     {-# inline isProperSubsetOf #-}
129
130     define isSetEquiv (ListBag xs) (ListBag ys) := listSetEquiv xs ys
131     {-# inline isSetEquiv #-}
132
133     define disjoint (ListBag xs) (ListBag ys) := listDisjoint xs ys
134     {-# inline disjoint #-}
135 end-instance
136
137 instance Eq 'e => EqP (ListBag 'e) where
138     define non-exec (===) q1 q2 := bool2prop (isSetEquiv q1 q2)
139 end-instance
140
141 instance Eq 'e => Eq (ListBag 'e) where
142     define (==) := isSetEquiv
143     {-# inline (==) #-}
144 end-instance
145
146 instance BagLike (ListBag 'e) 'e where
147 end-instance
148
149 instance Eq 'e => Bag (ListBag 'e) 'e where
150 end-instance
151
152 instance Eq 'e => FiniteBag (ListBag 'e) 'e where
153 end-instance
154
155 end-module

```

## 1.4 Data/Collection/Queue.ad

### Outline

- module Data.Collection.Queue ..... lines 1 - 91
  - Queue (datatype) .....line 10
  - emptyQueue (declaration + definition) ..... lines 12, 13
  - enqueue (declaration + definition) ..... lines 17, 18
  - dequeue (declaration + definition) .....lines 20 - 23
  - queueToList (declaration + definition) ..... lines 25, 26
  - queueFromList (declaration + definition) ..... lines 28, 29
  - Functor Queue (typeclass-instance) .....lines 31 - 33
    - \* fmap (definition) .....line 32
  - EqP (Queue 'a) (typeclass-instance) .....lines 35 - 37
    - \* === (definition) .....line 36
  - Eq (Queue 'a) (typeclass-instance) .....lines 39 - 41
    - \* == (definition) .....line 40
  - Ord (Queue 'a) (typeclass-instance) .....lines 43 - 45
    - \* compare (definition) .....line 44
  - Semigroup (Queue 'a) (typeclass-instance) .....lines 47 - 49
    - \* <> (definition) .....line 48
  - Monoid (Queue 'a) (typeclass-instance) .....lines 51 - 54
    - \* mempty (definition) .....line 52
  - CollectionWithEmptinessTest (Queue 'e) 'e (typeclass-instance) .....lines 56 - 58
    - \* isEmpty (definition) .....line 57
  - Collection (Queue 'e) 'e (typeclass-instance) .....lines 60 - 69
    - \* insert (definition) .....line 61
    - \* fromList (definition) .....line 64
    - \* isEmptyP (definition) .....line 67
    - \* memberP (definition) .....line 68
  - Foldable (Queue 'e) 'e (typeclass-instance) .....lines 71 - 80
    - \* toList (definition) .....line 72
    - \* size (definition) .....line 75
    - \* null (definition) .....line 76
    - \* elem (definition) .....line 79
  - CollectionWithMembershipTest (Queue 'e) 'e (typeclass-instance) .....lines 82 - 86
    - \* occur (definition) .....line 83
    - \* member (definition) .....line 84
  - FiniteCollection (Queue 'e) 'e (typeclass-instance) ..... lines 88, 89

### Content

```

1 module Data.Collection.Queue (
2   Queue,
3   emptyQueue,
4   enqueue,
5   dequeue
6 ) where

```

```

7
8 import qualified Data.List as List
9
10 datatype Queue 'a := Queue ['a] ['a]
11
12 declare emptyQueue :: Queue 'a
13 define emptyQueue := Queue [] []
14
15 {-# allow-similar-names "enqueue", "dequeue" #-}
16
17 declare enqueue :: 'a -> Queue 'a -> Queue 'a
18 define enqueue e (Queue l1 l2) := Queue l1 (e:l2)
19
20 declare dequeue :: Queue 'a -> Maybe ('a, Queue 'a)
21 define rec dequeue (Queue (e:l1) l2) := Just (e, Queue l1 l2)
22     | dequeue (Queue [] []) := Nothing
23     | dequeue (Queue [] l2) := dequeue (Queue (reverse l2) [])
24
25 declare queueToList :: Queue 'a -> ['a]
26 define queueToList (Queue l1 l2) := l1 ++ reverse l2
27
28 declare queueFromList :: ['a] -> Queue 'a
29 define queueFromList l := Queue l []
30
31 instance Functor Queue where
32     define fmap f (Queue l1 l2) := Queue (map f l1) (map f l2)
33 end-instance
34
35 instance EqP 'a => EqP (Queue 'a) where
36     define non-exec (===) q1 q2 := (queueToList q1) === (queueToList q2)
37 end-instance
38
39 instance Eq 'a => Eq (Queue 'a) where
40     define (==) q1 q2 := (queueToList q1) == (queueToList q2)
41 end-instance
42
43 instance Ord 'a => Ord (Queue 'a) where
44     define compare q1 q2 := compare (queueToList q1) (queueToList q2)
45 end-instance
46
47 instance Semigroup (Queue 'a) where
48     define (<>) q1 q2 := queueFromList ((queueToList q1) ++ (queueToList q2))
49 end-instance
50
51 instance Monoid (Queue 'a) where
52     define mempty := emptyQueue
53     {-# inline mempty #-}
54 end-instance
55
56 instance CollectionWithEmptynessTest (Queue 'e) 'e where
57     define isEmpty (Queue l1 l2) := (null l1 && null l2)
58 end-instance
59
60 instance Collection (Queue 'e) 'e where
61     define insert := enqueue
62     {-# inline insert #-}
63
64     define fromList := queueFromList
65     {-# inline fromList #-}

```

```

66
67   define non-exec isEmptyP q := bool2prop (isEmpty q)
68   define non-exec memberP e (Queue l1 l2) := List.elemP e l1 ∨ List.elemP e l2
69 end-instance
70
71 instance Foldable (Queue 'e) 'e where
72   define toList := queueToList
73   {-# inline toList #-}
74
75   define size (Queue l1 l2) := length l1 + length l2
76   define null := isEmpty
77   {-# inline null #-}
78
79   define elem e (Queue l1 l2) := elem e l1 || elem e l2
80 end-instance
81
82 instance Eq 'e => CollectionWithMembershipTest (Queue 'e) 'e where
83   define occur e (Queue l1 l2) := count e l1 + count e l2
84   define member := elem
85   {-# inline member #-}
86 end-instance
87
88 instance Eq 'e => FiniteCollection (Queue 'e) 'e where
89 end-instance
90
91 end-module

```

## 1.5 Data/Collection/Stack.ad

### Outline

- module Data.Collection.Stack ..... lines 1 - 82
  - Stack (datatype) .....line 10
  - emptyStack (declaration + definition) ..... lines 12, 13
  - push (declaration + definition) ..... lines 15, 16
  - pop (declaration + definition) .....lines 18 - 20
  - stackToList (declaration + definition) ..... lines 22, 23
  - stackFromList (declaration + definition) ..... lines 25, 26
  - Functor Stack (typeclass-instance) .....lines 28 - 30
    - \* fmap (definition) .....line 29
  - EqP (Stack 'a) (typeclass-instance) .....lines 32 - 34
    - \* === (definition) .....line 33
  - Eq (Stack 'a) (typeclass-instance) .....lines 36 - 38
    - \* == (definition) .....line 37
  - Ord (Stack 'a) (typeclass-instance) .....lines 40 - 42
    - \* compare (definition) .....line 41
  - Semigroup (Stack 'a) (typeclass-instance) .....lines 44 - 46
    - \* <> (definition) .....line 45
  - Monoid (Stack 'a) (typeclass-instance) .....lines 48 - 51
    - \* mempty (definition) .....line 49
  - CollectionWithEmptynessTest (Stack 'e) 'e (typeclass-instance) .....lines 53 - 55
    - \* isEmpty (definition) .....line 54

- Collection (Stack 'e) 'e (typeclass-instance) .....	lines 57 - 66
* insert (definition) .....	line 58
* fromList (definition) .....	line 61
* isEmptyP (definition) .....	line 64
* memberP (definition) .....	line 65
- Foldable (Stack 'e) 'e (typeclass-instance) .....	lines 68 - 71
* toList (definition) .....	line 69
- CollectionWithMembershipTest (Stack 'e) 'e (typeclass-instance) .....	lines 73 - 77
* occur (definition) .....	line 74
* member (definition) .....	line 75
- FiniteCollection (Stack 'e) 'e (typeclass-instance) .....	lines 79, 80

## Content

```

1 module Data.Collection.Stack (
2   Stack,
3   emptyStack,
4   push,
5   pop
6 ) where
7
8 import qualified Data.List as List
9
10 datatype Stack 'a := Stack ['a]
11
12 declare emptyStack :: Stack 'a
13 define emptyStack := Stack []
14
15 declare push :: 'a -> Stack 'a -> Stack 'a
16 define push e (Stack l) := Stack (e:l)
17
18 declare pop :: Stack 'a -> Maybe ('a, Stack 'a)
19 define pop (Stack (e:l)) := Just (e, Stack l)
20   | pop (Stack []) := Nothing
21
22 declare stackToList :: Stack 'a -> ['a]
23 define stackToList (Stack l) := l
24
25 declare stackFromList :: ['a] -> Stack 'a
26 define stackFromList l := Stack l
27
28 instance Functor Stack where
29   define fmap f (Stack l) := Stack (map f l)
30 end-instance
31
32 instance EqP 'a => EqP (Stack 'a) where
33   define non-exec (===) q1 q2 := (stackToList q1) === (stackToList q2)
34 end-instance
35
36 instance Eq 'a => Eq (Stack 'a) where
37   define (==) q1 q2 := (stackToList q1) == (stackToList q2)
38 end-instance
39
40 instance Ord 'a => Ord (Stack 'a) where
41   define compare q1 q2 := compare (stackToList q1) (stackToList q2)
42 end-instance

```

```

43
44 instance Semigroup (Stack 'a) where
45   define (<>) q1 q2 := stackFromList ((stackToList q1) ++ (stackToList q2))
46 end-instance
47
48 instance Monoid (Stack 'a) where
49   define mempty := emptyStack
50   {-# inline mempty #-}
51 end-instance
52
53 instance CollectionWithEmptynessTest (Stack 'e) 'e where
54   define isEmpty (Stack l) := (null l)
55 end-instance
56
57 instance Collection (Stack 'e) 'e where
58   define insert := push
59   {-# inline insert #-}
60
61   define fromList := stackFromList
62   {-# inline fromList #-}
63
64   define non-exec isEmptyP q := bool2prop (isEmpty q)
65   define non-exec memberP e (Stack l) := List.elemP e l
66 end-instance
67
68 instance Foldable (Stack 'e) 'e where
69   define toList := stackToList
70   {-# inline toList #-}
71 end-instance
72
73 instance Eq 'e => CollectionWithMembershipTest (Stack 'e) 'e where
74   define occur e (Stack l) := count e l
75   define member := elem
76   {-# inline member #-}
77 end-instance
78
79 instance Eq 'e => FiniteCollection (Stack 'e) 'e where
80 end-instance
81
82 end-module

```

## 1.6 Data/Lens.ad

### Outline

- module Data.Lens ..... lines 2 - 259
  - Lens (recordtype) .....lines 41 - 44
  - SimpleLens (type) .....line 47
  - Getter (recordtype) .....lines 50 - 52
  - to (declaration + definition) ..... lines 55, 56
  - Setter (recordtype) .....lines 60 - 62
  - SimpleSetter (type) .....line 64
  - IsGetter (typeclass) .....lines 68 - 74
    - \* view (declaration) .....line 69
    - \* toGetter (declaration + definition) ..... lines 71, 72



- ^. (alias) .....	line 76
- IsSetter (typeclass) .....	lines 79 - 85
* set (declaration) .....	line 80
* toSetter (declaration + definition) .....	lines 82, 83
- IsGetter (Getter 'a 'b) 'a 'b (typeclass-instance) .....	lines 87 - 90
* view (definition) .....	line 88
- IsGetter (Lens 's 't 'a 'b) 's 'a (typeclass-instance) .....	lines 92 - 95
* view (definition) .....	line 93
- simpleLensToGetter (declaration + definition) .....	lines 98, 99
- IsSetter (Setter 's 't 'b) 's 't 'b (typeclass-instance) .....	lines 102 - 105
* set (definition) .....	line 103
- IsSetter (Lens 's 't 'a 'b) 's 't 'b (typeclass-instance) .....	lines 107 - 110
* set (definition) .....	line 108
- combineGetter (declaration + definition) .....	lines 116, 117
- Dot (Getter 'a 'b) (Getter 'b2 'c) (Getter 'a 'c) (typeclass-instance) ..	lines 120 - 123
* . (definition) .....	line 121
- Dot (Getter 'a 'b) (Lens 'b2 'x 'c 'y) (Getter 'a 'c) (typeclass-instance)	lines 125 - 128
* . (definition) .....	line 126
- Dot (Lens 'a 'x 'b 'y) (Getter 'b2 'c) (Getter 'a 'c) (typeclass-instance)	lines 130 - 133
* . (definition) .....	line 131
- combineSetter (declaration + definition) .....	lines 136 - 141
- Dot (Lens 's1 't1 'a1 'b1) (Setter 's2 't2 'b2) (Setter 's1 't1 'b2)	
(typeclass-instance) .....	lines 144 - 147
* . (definition) .....	line 145
- Dot (Lens 's1 't1 'a1 'b1) (Lens 's2 't2 'a2 'b2) (Lens 's1 't1 'a2 'b2)	
(typeclass-instance) .....	lines 149 - 152
* . (definition) .....	line 150
- BuildTupleElemLensClass (template) .....	lines 167 - 171
- BuildTupleElemLenses (template) .....	lines 173 - 189
- generate-code .....	line 191
- BuildRecordFieldLens (template) .....	lines 224 - 238
- BuildRecordFieldLensesSuffix (template) .....	lines 241 - 247
- BuildRecordFieldLensesPrefix (template) .....	lines 249 - 255
- BuildRecordFieldLenses (template) .....	line 257

## Content

```

1 -- very poor man's lenses
2 module Data.Lens (
3   -- Basic Defintions for Lenses, Getters and Setters
4   Getter(..),
5   Setter(..),
6   Lens(..),
7   SimpleLens(..),
8   SimpleSetter(..),
9
10  IsGetter(..),
11  IsSetter(..),
12  to,
```

```

13  (^.),
14  simpleLensToGetter,
15
16  -- Tuple lenses
17  Elem1(..),
18  Elem2(..),
19  Elem3(..),
20  Elem4(..),
21  Elem5(..),
22  Elem6(..),
23  Elem7(..),
24  Elem8(..),
25  Elem9(..),
26
27  -- Templates for building lenses corresponding to record fields
28  BuildRecordFieldLens,
29  BuildRecordFieldLenses,
30  BuildRecordFieldLensesPrefix,
31  BuildRecordFieldLensesSuffix
32 ) where
33
34 {-# severity similar-names - #-}
35
36 -- | a very poor mans lens. Essentially it is a tuple of a get and set function. The getter-part can
37 -- get a certain part out of some bigger data. The setter sets this part to a new value. Setting might c
38 -- output. For example, it is possible to set the second element of a tuple to a different type than its
39 -- This changes the type of the output so. "Lens 's 't 'a 'b" describes functions for getting a value o
40 -- type 's and a for updating a value of type 's with a value of type 'b to get a value of type 't.
41 recordtype Lens 's 't 'a 'b := Lens <|
42     lensGet :: 's -> 'a,
43     lensSet :: 's -> 'b -> 't
44 |>
45
46 -- | very often, the type is not changed by setting. This is then called a simple lens.
47 type SimpleLens 's 'a := Lens 's 's 'a 'a
48
49 -- | a datatype for just the getter part of lenses
50 recordtype Getter 'a 'b := Getter <|
51     getterGet :: 'a -> 'b
52 |>
53
54 -- | constructor for getters mimicking Haskell naming
55 declare to :: ('a -> 'b) -> Getter 'a 'b
56 define to := Getter
57 {-# inline to #-}
58
59 -- | the setter part of a lens, however, Setters without Getters are quite rare
60 recordtype Setter 's 't 'b := Setter <|
61     setterSet :: 's -> 'b -> 't
62 |>
63
64 type SimpleSetter 's 'b := Setter 's 's 'b
65
66
67 -- | type-class for accessing Lenses and Getters with a common interface
68 class IsGetter 'v 'a 'b | 'v -> 'a 'b where
69     declare view :: 'v -> 'a -> 'b
70
71     declare toGetter :: 'v -> Getter 'a 'b

```

```

72   define toGetter := Getter . view
73   {‑# inline toGetter #‑}
74 end-class
75
76 alias (^.) := view
77
78 -- | type-class for accessing Lenses and Setters with a common interface
79 class IsSetter 'v 's 't 'b | 'v -> 's 't 'b where
80   declare set :: 'v -> 's -> 'b -> 't
81
82   declare toSetter :: 'v -> Setter 's 't 'b
83   define toSetter := Setter . set
84   {‑# inline toSetter #‑}
85 end-class
86
87 instance IsGetter (Getter 'a 'b) 'a 'b where
88   define view := getterGet
89   {‑# inline view #‑}
90 end-instance
91
92 instance IsGetter (Lens 's 't 'a 'b) 's 'a where
93   define view := lensGet
94   {‑# inline view #‑}
95 end-instance
96
97 -- | a version of toGetter for Lenses with extra type constraints
98 declare simpleLensToGetter :: SimpleLens 'a 'b -> Getter 'a 'b
99 define simpleLensToGetter sl := toGetter sl
100 {‑# inline simpleLensToGetter #‑}
101
102 instance IsSetter (Setter 's 't 'b) 's 't 'b where
103   define set := setterSet
104   {‑# inline set #‑}
105 end-instance
106
107 instance IsSetter (Lens 's 't 'a 'b) 's 't 'b where
108   define set := lensSet
109   {‑# inline set #‑}
110 end-instance
111
112 -- -----
113 -- combine lenses, getters and setters
114 -- -----
115
116 declare combineGetter :: (IsGetter 'g1 'a 'b, IsGetter 'g2 'b 'c) => 'g1 -> 'g2 -> 'a -> 'c
117 define combineGetter g1 g2 := view g2 . view g1
118 {‑# inline combineGetter #‑}
119
120 instance Dot (Getter 'a 'b) (Getter 'b2 'c) (Getter 'a 'c) | 'b2 -> 'b where
121   define (.) g1 g2 := Getter $ combineGetter g1 g2
122   {‑# inline (.) #‑}
123 end-instance
124
125 instance Dot (Getter 'a 'b) (Lens 'b2 'x 'c 'y) (Getter 'a 'c) | 'b2 -> 'b where
126   define (.) g1 g2 := Getter $ combineGetter g1 g2
127   {‑# inline (.) #‑}
128 end-instance
129
130 instance Dot (Lens 'a 'x 'b 'y) (Getter 'b2 'c) (Getter 'a 'c) | 'b2 -> 'b where

```

```

131   define (.) g1 g2 := Getter $ combineGetter g1 g2
132   {-# inline (.) #-}
133 end-instance
134
135
136 declare combineSetter :: (IsGetter 'g1 's1 'a, IsSetter 'g1 's1 't1 'b, IsSetter 'g2 'a 'b 'a2) => 'g1 -> 'a -> 'b
137 define combineSetter g1 g2 v1 x2 :=
138   let val v2 = view g1 v1
139       val v2' = set g2 v2 x2
140       val v1' = set g1 v1 v2'
141   in v1' end
142   {-# inline combineSetter #-}
143
144 instance Dot (Lens 's1 't1 'a1 'b1) (Setter 's2 't2 'b2) (Setter 's1 't1 'b2) | 'a1 -> 's2, 'b1 -> 't2 where
145   define (.) g1 g2 := Setter $ combineSetter g1 g2
146   {-# inline (.) #-}
147 end-instance
148
149 instance Dot (Lens 's1 't1 'a1 'b1) (Lens 's2 't2 'a2 'b2) (Lens 's1 't1 'a2 'b2) | 'a1 -> 's2, 'b1 -> 't2 where
150   define (.) g1 g2 := Lens (combineGetter g1 g2) (combineSetter g1 g2)
151   {-# inline (.) #-}
152 end-instance
153
154
155
156
157 -- -----
158 -- Lenses for tuples
159 -- -----
160
161 -- Tuples with distinct number of elements are different types in ADATT. Nevertheless
162 -- using the same syntax for accessing elements is useful. For this purpose type-classes
163 -- for accessing the i-th element of any tuple with at least i elements are introduced
164 -- and instantiated using code-generation
165
166
167 template BuildTupleElemLensClass (m :: Nat) := '''
168 class Elem $\{\{m\}\}$  \s \t \a \b | \s -> \a, \s \b -> \t where
169   declare elem $\{\{m\}\}$  :: Lens \s \t \a \b
170 end-class
171 '''
172
173 template BuildTupleElemLenses (n :: Nat) := '''
174  $\{\{$  name-scope {
175     var tyVars := !GetTypeVarsNumbered("a", n);
176     var xs := generateNames("x", n);
177 }
178  $\}\}$ 
179  $\{\{$ % for var m := 1 to n % $\}\}$ 
180  $\{\{$  var xs' := xs; xs'[m-1] := "_";
181     var xs'' := xs; xs''[m-1] := "y";
182     var tyVars' := tyVars; tyVars'[m-1] := "y";  $\}\}$ 
183 instance Elem $\{\{m\}\}$   $\{\{$ = !BuildTuple(tyVars) = $\}\}$  \x \z \y | \x ->  $\{\{$ = !BuildTuple(tyVars') = $\}\}$ , \z -> \y where
184   define elem $\{\{m\}\}$  := Lens # $\{\{m\}\}$  (fn  $\{\{$ = !BuildTuple(xs') = $\}\}$  y ->  $\{\{$ = !BuildTuple(xs'') = $\}\}$ )
185   \{-# inline elem $\{\{m\}\}$  #-\}
186 end-instance
187
188  $\{\{$ % end-for % $\}\}$ 
189 '''

```

```

190
191 generate-code '''
192   {{ -- max number of tuple elements for which to generate classes and instances
193     var max := 9 }}
194   {{% for var m := 1 to max %}}
195   {{! BuildTupleElemLensClass(m) !}}
196
197   {{% end-for %}}
198   {{% for var m := 2 to max %}}
199   {{! BuildTupleElemLenses(m) !}}
200   {{% end-for %}}
201 '''
202
203
204 -----
205 -- Lenses for tuples
206 -----
207
208 -- It is useful to have lenses corresponding to record fields.
209 -- There are templates to automatically generate such lenses.
210
211 -- - BuildRecordFieldLens (field, lensName)
212 -- generate a lens with name "lensName" for the record field field
213
214 -- - BuildRecordFieldLensesSuffix (suffix, type)
215 -- generate lenses for all fields of record-type "type", whose name ends with given suffix. The lens name
216
217 -- - BuildRecordFieldLensesPrefix (prefix, type)
218 -- generate lenses for all fields of record-type "type", whose name starts with given prefix. The lens name
219
220 -- - BuildRecordFieldLenses (type)
221 -- short for "BuildRecordFieldLensesSuffix("_", type)
222
223
224 template BuildRecordFieldLens (field :: ConstID, lensName :: String) := '''
225 {{ var tyID := field.baseType;
226   var (_, tyWithArgs) := !TypeWithArgs(True, tyID);
227   var fs := tyID.fields;
228 }}
229 declare {{= lensName =}} :: Lens {{= tyWithArgs =}} _ _ _
230 define {{= lensName =}} := <|
231   {{= const lensGet =}} := {{= field =}},
232   {{= const lensSet =}} := fn r v -> <|
233     {{% foreach var fi in fs.indices %}}
234     {{= fs[fi] =}} := {{% if (fs[fi] == field) %}}v{{% else %}}{{=fs[fi]=}} r{{% end-if %}}{{% if (f
235     {{% end-foreach %}}
236   |>
237 |>
238 '''
239
240
241 template BuildRecordFieldLensesSuffix (suffix :: String, tyID :: TypeID) := '''
242 {{% foreach var f in tyID.fields %}}
243 {{% if (endsWith(suffix, f.name)) %}}
244 {{! BuildRecordFieldLens(f, dropEnd(suffix.length, f.name)) !}}
245
246 {{% end-if %}}
247 {{% end-foreach %}}'''
248

```

```

249 template BuildRecordFieldLensesPrefix (pre :: String, tyID :: TypeID) := '''
250 {{% foreach var f in tyID.fields %}}
251 {{% if (startsWith(pre, f.name)) %}}
252 {{! BuildRecordFieldLens(f, drop(pre.length, f.name)) !}}
253
254 {{% end-if %}}
255 {{% end-foreach %}}'''
256
257 template BuildRecordFieldLenses (tyID :: TypeID) := '''{{! BuildRecordFieldLensesSuffix("-", tyID) !}}'''
258
259 end-module
260
261
262

```

## 1.7 Data/List.ad

### Outline

- module Data.List ..... lines 1 - 82
  - replicate (declaration + definition) .....lines 16 - 18
  - take (declaration + definition) .....lines 20 - 23
  - drop (declaration + definition) .....lines 25 - 28
  - splitAt (declaration + definition) .....lines 30 - 38
  - splitAt\_take\_drop (property) .....line 40
  - takeWhile (declaration + definition) .....lines 43 - 46
  - dropWhile (declaration + definition) .....lines 48 - 51
  - span (declaration + definition) .....lines 53 - 60
  - break (declaration + definition) ..... lines 62, 63
  - lookup (declaration + definition) .....lines 66 - 69
  - zipWith (declaration + definition) .....lines 72 - 74
  - zip (declaration + definition) ..... lines 76, 77
  - unzip (declaration + definition) ..... lines 79, 80

### Content

```

1 module Data.List (
2   module Prelude.List,
3   replicate,
4   take,
5   drop,
6   splitAt,
7   takeWhile,
8   dropWhile,
9   break,
10  span,
11  lookup,
12  zip,
13  unzip
14 ) where
15
16 declare replicate :: Natural -> 'a -> ['a]
17 define rec replicate 0 _ := []
18           | replicate n a := a : (replicate (pred n) a)

```

```

19
20 declare take :: Natural -> ['a] -> ['a]
21 define rec take 0 _ := []
22     | take _ [] := []
23     | take n (x:xs) := x : (take (pred n) xs)
24
25 declare drop :: Natural -> ['a] -> ['a]
26 define rec drop 0 l := l
27     | drop _ [] := []
28     | drop n (_:xs) := drop (pred n) xs
29
30 declare splitAt :: Natural -> ['a] -> (['a], ['a])
31 define rec splitAt 0 l := ([], l)
32     | splitAt _ [] := ([], [])
33     | splitAt n (x:xs) :=
34         let
35             val (lt, ld) = (splitAt (pred n) xs)
36         in
37             (x:lt, ld)
38         end
39
40 property EqP 'a => splitAt_take_drop n (l :: ['a]) := ((splitAt n l) === (take n l, drop n l))
41
42
43 declare takeWhile :: ('a -> Bool) -> ['a] -> ['a]
44 define rec takeWhile _ [] := []
45     | takeWhile p (x:xs) :=
46         if (p x) then x : (takeWhile p xs) else []
47
48 declare dropWhile :: ('a -> Bool) -> ['a] -> ['a]
49 define rec dropWhile _ [] := []
50     | dropWhile p (x:xs) :=
51         if (p x) then (dropWhile p xs) else (x:xs)
52
53 declare span :: ('a -> Bool) -> ['a] -> (['a], ['a])
54 define rec span _ [] := ([], [])
55     | span p (x:xs) :=
56         if (p x) then
57             let val (ys, zs) = span p xs in
58                 (x:ys, zs)
59             end
60         else ([], x:xs)
61
62 declare break :: ('a -> Bool) -> ['a] -> (['a], ['a])
63 define break p := span (not . p)
64
65
66 declare lookup :: Eq 'k => 'k -> [( 'k, 'v)] -> Maybe 'v
67 define rec lookup _key [] := Nothing
68     | lookup key ((k, v):kvs) :=
69         if (key == k) then Just v else lookup key kvs
70
71
72 declare zipWith :: ('a -> 'b -> 'c) -> ['a] -> ['b] -> ['c]
73 define rec zipWith f (x:xs) (y:ys) := (f x y):(zipWith f xs ys)
74     | zipWith _ _ _ := undefined
75
76 declare zip :: ['a] -> ['b] -> [( 'a, 'b)]
77 define zip := zipWith (fn x y -> (x, y))

```

```

78
79 declare unzip :: [('a,'b)] -> [('a),('b)]
80 define unzip := foldr (fn (x,y) (xs,ys) -> (x:xs,y:ys)) ([],[])
81
82 end-module

```

## 1.8 Data/Maybe.ad

### Outline

- module Data.Maybe ..... lines 1 - 30
  - listToMaybe (declaration + definition) ..... lines 9 - 11
  - maybeToList (declaration + definition) .....lines 13 - 15
  - catMaybes (declaration + definition) .....lines 17 - 20
  - mapMaybe (declaration + definition) .....lines 22 - 28

### Content

```

1 module Data.Maybe (
2   module Prelude.Maybe,
3   listToMaybe,
4   maybeToList,
5   catMaybes,
6   mapMaybe
7 ) where
8
9 declare listToMaybe :: ['a] -> Maybe 'a
10 define listToMaybe [] := Nothing
11     | listToMaybe (x:_) := Just x
12
13 declare maybeToList :: Maybe 'a -> ['a]
14 define maybeToList Nothing := []
15     | maybeToList (Just v) := [v]
16
17 declare catMaybes :: [Maybe 'a] -> ['a]
18 define rec catMaybes [] := []
19     | catMaybes (Nothing:vs) := catMaybes vs
20     | catMaybes ((Just v):vs) := v : (catMaybes vs)
21
22 declare mapMaybe :: ('a -> Maybe 'b) -> ['a] -> ['b]
23 define rec mapMaybe _ [] := []
24     | mapMaybe mf (v : vs) :=
25         case (mf v) of
26             Nothing -> mapMaybe mf vs
27             | (Just x) -> x:(mapMaybe mf vs)
28         end
29
30 end-module

```

## 1.9 Prelude.ad

### Outline

- module Prelude ..... lines 1 - 31



## Content

```

1 module Prelude (
2   module Prelude.Unit,
3   module Prelude.Bool,
4   module Prelude.Literal,
5   module Prelude.Default,
6   module Prelude.Function,
7   module Prelude.Either,
8   module Prelude.Maybe hiding (map, bind),
9   module Prelude.Eq,
10  module Prelude.List hiding (foldr, foldl, partition, elem, elemP, notElemP, null, filter, all, any, f
11  module Prelude.Num,
12  module Prelude.Ord,
13  module Prelude.Tuple,
14  module Prelude.Char,
15  module Prelude.String,
16  module Prelude.Text,
17  module Prelude.BasicClasses,
18  module Prelude.Foldable,
19  module Prelude.Functor,
20  module Prelude.Monad,
21  module Prelude.Collection,
22  module Prelude.Template
23 ) where
24
25 {-# NoImplicitPrelude #-}
26
27 -- Just include instances, no need to explicitly export
28 import Prelude.ListCollection
29
30
31 end-module

```

## 1.10 Prelude/BasicClasses.ad

### Outline

- module Prelude.BasicClasses ..... lines 1 - 280
  - Enum (typeclass) .....lines 25 - 65
    - \* succ (declaration) .....line 26
    - \* pred (declaration) .....line 27
    - \* fromEnum (declaration) .....line 29
    - \* toEnum (declaration) .....line 30
    - \* succ (definition) .....line 32
    - \* pred (definition) .....line 33
    - \* enumFromTo (declaration + definition) .....lines 35 - 37
    - \* toFromEnumInv (test) ..... lines 39, 40
    - \* fromEnum11 (test) ..... lines 42, 43
    - \* toEnumBounds0 (test) .....line 45
    - \* toEnumBoundsSmaller (test) .....lines 46 - 49
    - \* compareOK (test) ..... lines 51, 52
    - \* succ\_greater\_ok (test) .....line 54
    - \* pred\_smaller\_ok (test) .....line 55

* succ_bound_ok (test) .....	lines 57, 58
* pred_bound_ok (test) .....	lines 59, 60
* pred_minbound (test) .....	line 61
* succ_maxbound (test) .....	line 62
– Enum Natural (typeclass-instance) .....	lines 67 - 72
* succ (definition) .....	line 68
* pred (definition) .....	line 69
* toEnum (definition) .....	line 70
* fromEnum (definition) .....	line 71
– Bounded (typeclass) .....	lines 76 - 86
* minBound (declaration) .....	line 77
* maxBound (declaration) .....	line 78
* minBound_ok (test) .....	line 80
* maxBound_ok (test) .....	line 81
* minBoundEnum (test) .....	line 83
* maxBoundEnum (test) .....	line 85
– Finite (typeclass) .....	lines 90 - 109
* universeList (declaration) .....	line 91
* universeFoldWithAbort (declaration) .....	line 93
* forallB_fun (declaration) .....	line 96
* existsB_fun (declaration) .....	line 97
* uexistsB_fun (declaration) .....	line 98
* universeFoldWithAbort (definition) .....	line 100
* universeList (definition) .....	line 101
* forallB_fun (definition) .....	line 103
* existsB_fun (definition) .....	line 104
* uexistsB_fun (definition) .....	lines 105, 106
– Semigroup (typeclass) .....	lines 112 - 114
* <> (declaration) .....	line 113
– Monoid (typeclass) .....	lines 117 - 122
* mempty (declaration) .....	line 118
* mempty_neutral_1 (property) .....	line 120
* mempty_neutral_2 (property) .....	line 121
– mappend (declaration + definition) .....	lines 124, 125
– Semigroup () (typeclass-instance) .....	lines 129 - 132
* <> (definition) .....	line 130
– Monoid () (typeclass-instance) .....	lines 134 - 137
* mempty (definition) .....	line 135
– Semigroup ['a] (typeclass-instance) .....	lines 139 - 142
* <> (definition) .....	line 140
– Monoid ['a] (typeclass-instance) .....	lines 144 - 147
* mempty (definition) .....	line 145
– Semigroup Ordering (typeclass-instance) .....	lines 149 - 153
* <> (definition) .....	lines 150 - 152

- Monoid Ordering (typeclass-instance) .....	lines 155 - 157
* mempty (definition) .....	line 156
- Finite ('a, 'b) (typeclass-instance) .....	lines 160 - 164
* universeFoldWithAbort (definition) .....	lines 161, 162
- BuildFiniteTupleInstance (template) .....	lines 166 - 184
- generate-code .....	line 186
- Finite () (typeclass-instance) .....	lines 193 - 197
* universeList (definition) .....	line 194
* universeFoldWithAbort (definition) .....	lines 195, 196
- Finite (Maybe 'a) (typeclass-instance) .....	lines 199 - 203
* universeFoldWithAbort (definition) .....	lines 200 - 202
- Finite (Either 'a 'b) (typeclass-instance) .....	lines 205 - 209
* universeFoldWithAbort (definition) .....	lines 206 - 208
- DeriveInstanceEnumSimpleEnumDatatypeBody (template) .....	lines 213 - 247
- DeriveInstanceFiniteSimpleEnumDatatypeBody (template) .....	lines 253 - 259
- DeriveInstanceBoundedSimpleEnumDatatypeBody (template) .....	lines 264 - 272
- Bounded Ordering (typeclass-instance) .....	line 277
- Finite Ordering (typeclass-instance) .....	line 277
- Enum Ordering (typeclass-instance) .....	line 277
- Bounded Bool (typeclass-instance) .....	line 278
- Finite Bool (typeclass-instance) .....	line 278
- Enum Bool (typeclass-instance) .....	line 278

## Content

```

1 module Prelude.BasicClasses (
2   Enum(..),
3   Bounded(..),
4   Finite(..),
5
6   Monoid(..),
7   mappend,
8   Semigroup(..)
9 ) where
10
11 {-# NoImplicitPrelude #-}
12
13 import Prelude.Bool
14 import Prelude.Function
15 import Prelude.Num
16 import Prelude.Eq
17 import Prelude.Ord
18 import Prelude.Maybe
19 import Prelude.Either
20 import Prelude.Template
21 import Prelude.Unit ()
22 import Prelude.List hiding (map)
23
24
25 class Ord 'a => Enum 'a where
26   declare succ :: 'a -> 'a
27   declare pred :: 'a -> 'a

```

```

28
29 declare fromEnum :: 'a -> Natural
30 declare toEnum  :: Natural -> Maybe 'a
31
32 define succ a := fromMaybe a (toEnum (fromEnum a + 1))
33 define pred a := fromMaybe a (toEnum (fromEnum a - 1))
34
35 declare enumFromTo :: 'a -> 'a -> ['a]
36 define rec enumFromTo n m :=
37   if (n > m) then [] else n : (enumFromTo (succ n) m)
38
39 test toFromEnumInv (a :: 'a) :=
40   toEnum (fromEnum a) == Just a
41
42 test fromEnum11 a1 a2 :=
43   (fromEnum a1 == fromEnum a2) <-> (a1 == a2)
44
45 test toEnumBounds0 := isJust ((toEnum 0) :: Maybe 'a)
46 test toEnumBoundsSmaller i j :=
47   (i > j) -->
48   isJust ((toEnum i) :: Maybe 'a) -->
49   isJust ((toEnum j) :: Maybe 'a)
50
51 test compareOK a1 a2 :=
52   compare a1 a2 == compare (fromEnum a1) (fromEnum a2)
53
54 test succ_greater_ok (x :: 'a) := (succ x >= x)
55 test pred_smaller_ok (x :: 'a) := (pred x <= x)
56
57 test Bounded 'a => succ_bound_ok (x :: 'a) :=
58   (x < maxBound) --> ((succ x > x) && (pred (succ x) == x))
59 test Bounded 'a => pred_bound_ok (x :: 'a) :=
60   (x > minBound) --> ((pred x < x) && (succ (pred x) == x))
61 test Bounded 'a => pred_minbound := pred (minBound :: 'a) == minBound
62 test Bounded 'a => succ_maxbound := succ (maxBound :: 'a) == maxBound
63
64 {-# minimal fromEnum, toEnum #-}
65 end-class
66
67 instance Enum Natural where
68   define succ := (+) 1
69   define pred := (-) 1
70   define toEnum n := Just n
71   define fromEnum n := n
72 end-instance
73
74 {-# allow-similar-names "minBound", "maxBound", "minBound_ok", "maxBound_ok", "minBoundEnum", "maxBoundEnum" #-}
75
76 class Ord 'a => Bounded 'a where
77   declare minBound :: 'a
78   declare maxBound :: 'a
79
80   test minBound_ok (x :: 'a) := (minBound :: 'a) <= x
81   test maxBound_ok (x :: 'a) := (maxBound :: 'a) >= x
82
83   test Enum 'a => minBoundEnum := fromEnum (minBound :: 'a) == 0
84
85   test Enum 'a => maxBoundEnum i := (i > fromEnum (maxBound :: 'a)) --> isNothing ((toEnum i) :: Maybe 'a)
86 end-class

```

```

87
88
89 -- finite, type
90 class Finite 'a where
91   declare universeList :: ['a]
92
93   declare universeFoldWithAbort :: ('acc -> Bool) -> ('acc -> 'a -> 'acc) -> 'acc -> 'acc
94   {-# static resolve universeFoldWithAbort #-}
95
96   declare forallB_fun :: ('a -> Bool) -> Bool
97   declare existsB_fun :: ('a -> Bool) -> Bool
98   declare uexistsB_fun :: ('a -> Bool) -> Bool
99
100  define universeFoldWithAbort ab f acc := foldlWithAbort ab f acc universeList
101  define universeList := universeFoldWithAbort (constant False) (fn l e -> e:l) []
102
103  define forallB_fun p := universeFoldWithAbort not (constant p) True
104  define existsB_fun p := universeFoldWithAbort id (constant p) False
105  define uexistsB_fun p :=
106    (universeFoldWithAbort (fn c -> (c >= 2)) (fn c e -> if p e then c + 1 else c) 0) == (1 :: Natural)
107
108    {-# minimal universeList | universeFoldWithAbort #-}
109  end-class
110
111
112 class Semigroup 'a where
113   declare (<>) :: 'a -> 'a -> 'a
114  end-class
115
116
117 class Semigroup 'a => Monoid 'a where
118   declare mempty :: 'a
119
120   property EqP 'a => mempty_neutral_1 (x :: 'a) := (mempty <> x) === x
121   property EqP 'a => mempty_neutral_2 (x :: 'a) := (x <> mempty) === x
122  end-class
123
124  declare mappend :: Monoid 'a => 'a -> 'a -> 'a
125  define mappend := (<>)
126  {-# inline mappend #-}
127
128
129  instance Semigroup () where
130    define (<>) := constant (constant ())
131    {-# inline (<>) #-}
132  end-instance
133
134  instance Monoid () where
135    define mempty := ()
136    {-# inline mempty #-}
137  end-instance
138
139  instance Semigroup ['a] where
140    define (<>) := (++)
141    {-# inline (<>) #-}
142  end-instance
143
144  instance Monoid ['a] where
145    define mempty := []

```

```

146   {-# inline mempty #-}
147   end-instance
148
149   instance Semigroup Ordering where
150     define (<<) LT _ := LT
151           | (<<) EQ o := o
152           | (<<) GT _ := GT
153   end-instance
154
155   instance Monoid Ordering where
156     define mempty := EQ
157   end-instance
158
159
160   instance (!Finite 'a, !Finite 'b) => Finite ('a, 'b) where
161     define universeFoldWithAbort ab f :=
162       universeFoldWithAbort ab (fn acc a -> universeFoldWithAbort ab (fn acc' b -> f acc' (a, b)) acc)
163     {-# inline universeFoldWithAbort #-}
164   end-instance
165
166   template BuildFiniteTupleInstance (n :: Nat) := '''
167   {{ name-scope {
168     var tyVars := !GetTypeVars(n);
169     var xs := generateNames("x", n);
170   }}
171   var super := [] :: [String];
172   foreach var tyV in tyVars {
173     super := insertAtEnd("!Finite " ++ tyV, super);
174   }
175
176   var inst := "Finite " ++ !BuildTuple(tyVars);
177   var m := n / 2;
178 }}
179 {{! BuildInstanceHeader(super, inst) !}}
180   define universeFoldWithAbort ab f :=
181     universeFoldWithAbort ab (fn acc ({{= !BuildTuple(take(m, xs)) =}}, {{= !BuildTuple(drop(m, xs)) =}})
182     \{-# INLINE universeFoldWithAbort #-\}
183   end-instance
184   '''
185
186   generate-code '''
187     {% for var i := 3 to 10 %}
188     {{! BuildFiniteTupleInstance(i) !}}
189
190     {% end-for %}'''
191
192
193   instance Finite () where
194     define universeList := [()]
195     define universeFoldWithAbort ab f a :=
196       if ab a then a else f a ()
197   end-instance
198
199   instance !Finite 'a => Finite (Maybe 'a) where
200     define universeFoldWithAbort ab f a :=
201       if ab a then a else
202         universeFoldWithAbort ab (fn acc v -> f acc (Just v)) (f a Nothing)
203   end-instance
204

```

```

205 instance (!Finite 'a, !Finite 'b) => Finite (Either 'a 'b) where
206   define universeFoldWithAbort ab f a :=
207     universeFoldWithAbort ab (fn acc v -> f acc (Left v)) $
208     universeFoldWithAbort ab (fn acc v -> f acc (Right v)) a
209 end-instance
210
211
212
213 template DeriveInstanceEnumSimpleEnumDatatypeBody (_ :: ClassID, ty :: TypeID, _ :: [String]) := '''
214 {{ if (not (!IsSimpleEnumDatatype(ty))) {
215     error("This template only works for simple enumeration types");
216   }
217   var constrs := ty.constructors;
218   var maxInd := constrs.length - 1;
219 }}
220 define fromEnum {{
221   for var pos := 0 to maxInd {
222     if (pos > 0) print("      | fromEnum ");
223     println(constrs[pos] ++ " := " ++ pos);
224   }
225 }}
226 define toEnum {{
227   for var pos := 0 to maxInd {
228     if (pos > 0) print("      | toEnum ");
229     println(pos ++ " := Just " ++ constrs[pos]);
230   }
231   println("      | toEnum _ := Nothing")
232 }}
233 define succ {{
234   for var pos := 0 to maxInd {
235     if (pos > 0) print("      | succ ");
236     var nextPos := pos+1;
237     if (nextPos > maxInd) nextPos := pos;
238     println(constrs[pos] ++ " := " ++ constrs[nextPos]);
239   }
240 }}
241 define pred {{
242   for var pos := 0 to maxInd {
243     if (pos > 0) print("      | pred ");
244     println(constrs[pos] ++ " := " ++ constrs[pos-1]);
245   }
246 }}
247 '''
248
249 register-derive-template Enum DeriveInstanceEnumSimpleEnumDatatypeBody
250
251 register-derive-template Enum DeriveInstanceEnumSimpleEnumDatatypeBody as "simpleEnum"
252
253 template DeriveInstanceFiniteSimpleEnumDatatypeBody (_ :: ClassID, ty :: TypeID, _ :: [String]) := '''
254 {{ if (not (!IsSimpleEnumDatatype(ty))) {
255     error("This template only works for simple enumeration types");
256   }
257 }}
258 define universeList := [{{= !Commafy(ty.constructors) =}}]
259 '''
260
261 register-derive-template Finite DeriveInstanceFiniteSimpleEnumDatatypeBody as "simpleEnum"
262
263

```

```

264 template DeriveInstanceBoundedSimpleEnumDatatypeBody (_ :: ClassID, ty :: TypeID, _ :: [String]) := '''
265 {{ if (not (!IsSimpleEnumDatatype(ty))) {
266     error("This template only works for simple enumeration types");
267 }
268     var constrs := ty.constructors;
269 }}
270 define minBound := {{= constrs[0] =}}
271 define maxBound := {{= constrs[constrs.length - 1] =}}
272 '''
273
274 register-derive-template Bounded DeriveInstanceBoundedSimpleEnumDatatypeBody as "simpleEnum"
275
276
277 derive Ordering instances (Bounded, Finite, Enum) via "simpleEnum"
278 derive Bool instances (Bounded, Finite, Enum) via "simpleEnum"
279
280 end-module

```

## 1.11 Prelude/Bool.ad

### Outline

- module Prelude.Bool ..... lines 1 - 74
  - Bool (datatype) .....line 22
  - not (declaration) .....line 23
  - Default Bool (typeclass-instance) ..... lines 25, 26
  - boolCase (declaration + definition) .....lines 28 - 30
  - not (definition) ..... lines 32, 33
  - && (declaration + definition) .....lines 35 - 37
  - || (declaration + definition) .....lines 39 - 41
  - --> (declaration + definition) ..... lines 43, 44
  - <-> (declaration + definition) .....lines 46 - 48
  - <-/-> (declaration + definition) ..... lines 50, 51
  - Prop (type) .....line 53
  - true (declaration) .....line 55
  - false (declaration) .....line 56
  - bool2prop (declaration + definition) ..... lines 58, 59
  - notP (declaration) .....line 62
  - /\ (declaration) .....line 63
  - \/ (declaration) .....line 64
  - ==> (declaration) .....line 65
  - <=> (declaration) .....line 66
  - <=/=> (declaration) .....line 67
  - forallP\_fun (declaration) .....line 70
  - existsP\_fun (declaration) .....line 71
  - uexistsP\_fun (declaration) .....line 72



## Content

```

1 module Prelude.Bool (
2   type Bool(..),
3   boolCase,
4   not, (&&), (||),
5   (-->), (<->), (<-/->),
6
7   type Prop,
8   true, false,
9   notP, (/\\), (\\/), (==>), (<=>), (<=/=>),
10  bool2prop,
11
12  forallP_fun,
13  existsP_fun,
14  uexistsP_fun
15 ) where
16
17 {-# NoImplicitPrelude #-}
18
19 import Prelude.Default
20 {-# allow-similar-names "False", "True", "true", "false" #-}
21
22 datatype Bool := False | True
23 declare not :: Bool -> Bool
24
25 instance Default Bool where
26 end-instance
27
28 declare boolCase :: Bool -> 'a -> 'a -> 'a
29 define boolCase True ct _ := ct
30       | boolCase False _ cf := cf
31
32 define not True := False
33       | not False := True
34
35 declare (&&) :: Bool -> Bool -> Bool
36 define (&&) True b := b
37       | (&&) False _ := False
38
39 declare (||) :: Bool -> Bool -> Bool
40 define (||) True _ := True
41       | (||) False b := b
42
43 declare (-->) :: Bool -> Bool -> Bool
44 define (-->) b1 b2 := not b1 || b2
45
46 declare (<->) :: Bool -> Bool -> Bool
47 define (<->) b1 b2 :=
48   (b1 --> b2) && (b2 --> b1)
49
50 declare (<-/->) :: Bool -> Bool -> Bool
51 define (<-/->) b1 b2 := not (b1 <-> b2)
52
53 type non-exec Prop
54
55 declare non-exec true :: Prop
56 declare non-exec false :: Prop
57

```

```

58 declare non-exec bool2prop :: Bool -> Prop
59 define non-exec bool2prop b := if b then true else false
60
61
62 declare non-exec notP    :: Prop -> Prop
63 declare non-exec (/\)    :: Prop -> Prop -> Prop
64 declare non-exec (\/)    :: Prop -> Prop -> Prop
65 declare non-exec (==>)  :: Prop -> Prop -> Prop
66 declare non-exec (<=>)  :: Prop -> Prop -> Prop
67 declare non-exec (<=/=>) :: Prop -> Prop -> Prop
68
69 {-# allow-similar-names "forallP_fun", "existsP_fun", "uexistsP_fun" #-}
70 declare non-exec forallP_fun :: ('a -> Prop) -> Prop
71 declare non-exec existsP_fun :: ('a -> Prop) -> Prop
72 declare non-exec uexistsP_fun :: ('a -> Prop) -> Prop
73
74 end-module

```

## 1.12 Prelude/Char.ad

### Outline

- module Prelude.Char ..... lines 1 - 50
  - Char (datatype) .....line 20
  - Eq Char (typeclass-instance) .....line 22
  - EqP Char (typeclass-instance) .....line 22
  - Ord Char (typeclass-instance) .....line 22
  - Default Char (typeclass-instance) .....lines 24 - 27
    - \* default (definition) .....line 25
  - chr (declaration + definition) ..... lines 29, 30
  - ord (declaration + definition) ..... lines 32, 33
  - mkChar (declaration) .....line 35
  - LiteralChar Char (typeclass-instance) .....lines 37 - 40
    - \* mkLiteralChar (definition) .....line 38
  - Enum Char (typeclass-instance) .....lines 43 - 48
    - \* succ (definition) .....line 44
    - \* pred (definition) .....line 45
    - \* fromEnum (definition) .....line 46
    - \* toEnum (definition) .....line 47

### Content

```

1 module Prelude.Char (
2   Char,
3   mkChar,
4   chr,
5   ord
6 ) where
7
8 {-# NoImplicitPrelude #-}
9
10 import Prelude.Eq
11 import Prelude.Bool

```

```

12 import Prelude.Ord
13 import Prelude.Num
14 import Prelude.Maybe
15 import Prelude.Literal
16 import Prelude.Default
17 import Prelude.BasicClasses
18
19 -- Characters are represented by their unicode-value
20 datatype Char := Char Natural
21
22 derive Char instances (Eq, EqP, Ord)
23
24 instance Default Char where
25   define default := '-'
26   {-# inline default #-}
27 end-instance
28
29 declare chr :: Natural -> Char
30 define chr := Char
31
32 declare ord :: Char -> Natural
33 define ord (Char c) := c
34
35 declare mkChar :: BuiltInChar -> Char
36
37 instance LiteralChar Char where
38   define mkLiteralChar := mkChar
39   {-# inline mkLiteralChar #-}
40 end-instance
41
42
43 instance Enum Char where
44   define succ (Char c) := Char (succ c)
45   define pred (Char c) := Char (pred c)
46   define fromEnum (Char c) := c
47   define toEnum c := Just (Char c)
48 end-instance
49
50 end-module

```

## 1.13 Prelude/Collection.ad

### Outline

- module Prelude.Collection ..... lines 1 - 211
  - Collection (typeclass) .....lines 45 - 63
    - \* insert (declaration + definition) ..... lines 46, 47
    - \* fromList (declaration + definition) ..... lines 49, 50
    - \* singleton (declaration + definition) ..... lines 52, 53
    - \* memberP (declaration) .....line 55
    - \* isEmptyP (declaration) .....line 57
    - \* isEmptyP\_ok (property) .....line 58
    - \* isEmptyP\_empty (property) .....line 59
    - \* memberP\_insert (property) .....line 61
    - \* memberP\_fromList (property) .....line 62

- empty (declaration + definition) .....	lines 65, 66
- union (declaration + definition) .....	lines 69, 70
- image (declaration + definition) .....	lines 74, 75
- CollectionWithEmptynessTest (typeclass) .....	lines 79 - 83
* isEmpty (declaration) .....	line 80
* isEmptyP_eq_isEmpty (property) .....	line 82
- CollectionWithMembershipTest (typeclass) .....	lines 88 - 103
* occur (declaration) .....	line 89
* member (declaration + definition) .....	lines 91, 92
* occur_insert_1 (test) .....	lines 94 - 96
* occur_insert_2 (test) .....	lines 97 - 99
* memberP_member (property) .....	lines 101, 102
- CollectionWithDelete (typeclass) .....	lines 107 - 140
* delete (declaration) .....	line 109
* filter (declaration + definition) .....	lines 112, 113
* difference (declaration) .....	line 115
* intersection (declaration) .....	line 116
* partition (declaration + definition) .....	lines 119, 120
* member_filter (test) .....	lines 123, 124
* occur_filter (test) .....	lines 126, 127
* occur_delete (test) .....	lines 129, 130
* member_delete (test) .....	lines 132, 133
* intersection_ok (test) .....	lines 135, 136
* difference_ok (test) .....	lines 137, 138
- FiniteCollection (typeclass) .....	lines 144 - 157
* empty_size (test) .....	line 145
* null_ok (test) .....	line 146
* size_fromToList (test) .....	line 148
* size_toFromList (test) .....	line 149
* elem_ok (test) .....	line 151
* memberP_toList (property) .....	lines 153, 154
* member_elem (test) .....	line 156
- CollectionWithSubsetTest (typeclass) .....	lines 162 - 177
* disjoint (declaration) .....	line 163
* isSubsetOf (declaration) .....	line 165
* isProperSubsetOf (declaration) .....	line 166
* isSetEquiv (declaration + definition) .....	lines 168, 169
* disjoint (definition) .....	line 171
* disjoint_ok (test) .....	line 173
* isSubsetOf_ok (test) .....	line 174
* isProperSubsetOf_ok (test) .....	line 175
- SetLike (typeclass) .....	lines 179 - 181
* noDups_occur (test) .....	line 180
- setEmpty (declaration + definition) .....	lines 183, 184

- setInsert (declaration + definition) .....	lines 187, 188
- setFromList (declaration + definition) .....	lines 191, 192
- BagLike (typeclass) .....	lines 195 - 197
* occur_insert_bag (test) .....	line 196
- Set (typeclass) .....	lines 199, 200
- FiniteSet (typeclass) .....	lines 202, 203
- Bag (typeclass) .....	lines 205, 206
- FiniteBag (typeclass) .....	lines 208, 209

## Content

```

1 module Prelude.Collection (
2   Collection(..),
3   empty,
4   union,
5   image,
6
7   setEmpty,
8   setInsert,
9   setFromList,
10
11  CollectionWithEmptynessTest(..),
12  CollectionWithMembershipTest(..),
13  CollectionWithDelete(..),
14  FiniteCollection(..),
15  CollectionWithSubsetTest(..),
16  SetLike(..),
17  BagLike(..),
18  Set(..),
19  FiniteSet(..),
20  Bag(..),
21  FiniteBag(..)
22 ) where
23
24 {-# NoImplicitPrelude #-}
25
26 import Prelude.Bool
27 import Prelude.BasicClasses
28 import Prelude.Eq
29 import Prelude.Ord
30 import Prelude.Function
31 import Prelude.Tuple
32 import Prelude.Functor
33 import Prelude.Foldable
34 import Prelude.Num
35 import qualified Prelude.List as List
36
37
38 -- {-# allow-similar-names "isEmptyP_ok", "isEmpty_ok" #-}
39
40 -- | A 'Collection' is a container for some elements. Not much is known
41 -- about this element or the container. There might be a finite or infinite number of elements, elements
42 -- might occur multiple times, their order might matter or not, there might not be a (decidable) equality
43 -- elements and membership might not be decidable.
44 -- Via the 'Monoid' type class an empty collection and combining of collections is available.
45 class Monoid 'c => Collection 'c 'e | 'c -> 'e where
46   declare insert :: 'e -> 'c -> 'c

```

```

47 define insert e c := (singleton e) <> c
48
49 declare fromList :: ['e] -> 'c
50 define fromList := foldl (fn c e -> insert e c) mempty
51
52 declare singleton :: 'e -> 'c
53 define singleton e := insert e mempty
54
55 declare non-exec memberP :: EqP 'e => 'e -> 'c -> Prop
56
57 declare non-exec isEmptyP :: 'c -> Prop
58 property EqP 'e => isEmptyP_ok (c :: 'c) := isEmptyP c <=> (forallP e. notP (memberP e c))
59 property isEmptyP_empty := isEmptyP (mempty :: 'c)
60
61 property EqP 'e => memberP_insert (e1 :: 'e) e2 (c :: 'c) := (memberP e1 (insert e2 c) <=> ((e1 === e2) <=> (memberP e1 c)))
62 property EqP 'e => memberP_fromList (e :: 'e) (l :: ['e]) := (memberP e ((fromList l)::'c) <=> List.elem e l)
63 end-class
64
65 declare empty :: Collection 'c 'e => 'c
66 define empty := mempty
67 {-# inline empty #-}
68
69 declare union :: Collection 'c 'e => 'c -> 'c -> 'c
70 define union := (<>)
71 {-# inline union #-}
72
73 {-# severity class-constr-unused ignore #-}
74 declare image :: (!Functor 'C, Collection ('C 'e1) 'e1, Collection ('C 'e2) 'e2) => ('e1 -> 'e2) -> 'C 'e
75 define image := fmap
76 {-# inline image #-}
77 {-# severity class-constr-unused reset #-}
78
79 class Collection 'c 'e => CollectionWithEmptynessTest 'c 'e | 'c -> 'e where
80   declare isEmpty :: 'c -> Bool
81
82   property isEmptyP_eq_isEmpty (c :: 'c) := (isEmptyP c <=> (isEmpty c === True))
83 end-class
84
85
86 -- | For most collections, we want a decidable membership test. Since elements might occur multiple times
87 -- there is also a function that returns that number.
88 class Collection 'c 'e => CollectionWithMembershipTest 'c 'e | 'c -> 'e where
89   declare occur :: Eq 'e => 'e -> 'c -> Natural
90
91   declare member :: Eq 'e => 'e -> 'c -> Bool
92   define member e c := (occur e c > 0)
93
94   test Eq 'e => occur_insert_1 (e :: 'e) (c :: 'c) :=
95     (occur e (insert e c) == occur e c + 1) ||
96     ((occur e (insert e c) == 1) && (occur e c == 1))
97   test Eq 'e => occur_insert_2 (e1 :: 'e) (e2 :: 'e) (c :: 'c) :=
98     (e1 /= e2) -->
99     ((occur e1 (insert e2 c) == occur e1 c))
100
101   property Eq 'e => memberP_member (e :: 'e) (c :: 'c) :=
102     (memberP e c <=> (member e c === True))
103 end-class
104
105 -- | Removing elements is also common. Remember that collections can contain elements multiple times and

```

```

106 -- order of elements might matter or not,
107 class (Eq 'e, CollectionWithMembershipTest 'c 'e) => CollectionWithDelete 'c 'e | 'c -> 'e where
108 -- remove the first occurrence of an element in a container
109 declare delete :: 'e -> 'c -> 'c
110
111 -- remove all elements not satisfying a predicate from a container, order is preserved
112 declare filter :: ('e -> Bool) -> 'c -> 'c
113 define filter p s := fst (partition p s)
114
115 declare difference :: 'c -> 'c -> 'c
116 declare intersection :: 'c -> 'c -> 'c
117
118 -- split a container in all elements not satisfying a predicate and the ones that don't, order is preserved
119 declare partition :: ('e -> Bool) -> 'c -> ('c, 'c)
120 define partition p s := (filter p s, filter (not . p) s)
121
122
123 test member_filter (e :: 'e) (c :: 'c) p :=
124   member e (filter p c) <-> (member e c && p e)
125
126 test occur_filter (e :: 'e) (c :: 'c) p :=
127   occur e (filter p c) == (if p e then occur e c else 0)
128
129 test occur_delete (e :: 'e) (c :: 'c) :=
130   occur e (delete e c) == max 0 (occur e c - 1)
131
132 test member_delete (e1 :: 'e) (e2 :: 'e) (c :: 'c) :=
133   (e1 /= e2) --> (occur e1 (delete e2 c) == occur e1 c)
134
135 test intersection_ok (e :: 'e) (c1 :: 'c) (c2 :: 'c) :=
136   occur e (intersection c1 c2) == min (occur e c1) (occur e c2)
137 test difference_ok (e :: 'e) (c1 :: 'c) (c2 :: 'c) :=
138   occur e (difference c1 c2) == max 0 (occur e c1 - occur e c2)
139
140 end-class
141
142 -- | A finite collection. One can fold over a finite collection. This means being able to get the size and
143 -- having decidable membership tests. Moreover, one can now also convert to a list.
144 class (CollectionWithMembershipTest 'c 'e, CollectionWithEmptynessTest 'c 'e, Foldable 'c 'e) => FiniteCollection 'c 'e
145 test empty_size := (size (empty :: 'c) == 0)
146 test null_ok (c :: 'c) := (isEmpty c <-> null c)
147
148 test size_fromToList (c :: 'c) := size ((fromList ((toList c) :: ['e])) :: 'c) == size c
149 test size_toFromList (l :: ['e]) := size ((fromList l) :: 'c) <= List.length l
150
151 test Eq 'e => elem_ok (e :: 'e) (c :: 'c) := member e c <-> elem e c
152
153 property EqP 'e => memberP_toList (e :: 'e) (c :: 'c) :=
154   (List.elemP e (toList (c :: 'c)) <=> memberP e c)
155
156 test Eq 'e => member_elem (e :: 'e) (c :: 'c) := (elem e c <-> member e c)
157 end-class
158
159
160 -- | collections might consider the order of elements. If we ignore the order,
161 -- we can define datatypes like sets and bags (multisets)
162 class (CollectionWithDelete 'c 'e, CollectionWithEmptynessTest 'c 'e) => CollectionWithSubsetTest 'c 'e
163 declare disjoint :: 'c -> 'c -> Bool
164

```

```

165 declare isSubsetOf :: 'c -> 'c -> Bool
166 declare isProperSubsetOf :: 'c -> 'c -> Bool
167
168 declare isSetEquiv :: 'c -> 'c -> Bool
169 define isSetEquiv xs ys := isSubsetOf xs ys && isSubsetOf ys xs
170
171 define disjoint xs ys := isEmpty (intersection xs ys)
172
173 test disjoint_ok (e :: 'e) (c1 :: 'c) (c2 :: 'c) := not (member e c1 && member e c2)
174 test isSubsetOf_ok (e :: 'e) (c1 :: 'c) (c2 :: 'c) := (occur e c1 <= occur e c2)
175 test isProperSubsetOf_ok (c1 :: 'c) (c2 :: 'c) := (isSubsetOf c1 c2) && not (isSubsetOf c2 c1)
176
177 end-class
178
179 class CollectionWithMembershipTest 'c 'e => SetLike 'c 'e | 'c -> 'e where
180   test Eq 'e => noDups_occur (e :: 'e) (c :: 'c) := occur e c <= 1
181 end-class
182
183 declare setEmpty :: SetLike 'c 'e => 'c
184 define setEmpty := mempty
185 {-# inline setEmpty #-}
186
187 declare setInsert :: SetLike 'c 'e => 'e -> 'c -> 'c
188 define setInsert := insert
189 {-# inline setInsert #-}
190
191 declare setFromList :: SetLike 'c 'e => ['e] -> 'c
192 define setFromList := fromList
193 {-# inline setFromList #-}
194
195 class CollectionWithMembershipTest 'c 'e => BagLike 'c 'e | 'c -> 'e where
196   test Eq 'e => occur_insert_bag (e :: 'e) (c :: 'c) := (occur e (insert e c) == occur e c + 1)
197 end-class
198
199 class (SetLike 'c 'e, CollectionWithDelete 'c 'e, CollectionWithEmptynessTest 'c 'e) => Set 'c 'e | 'c -> 'e
200 end-class
201
202 class (Set 'c 'e, FiniteCollection 'c 'e) => FiniteSet 'c 'e | 'c -> 'e where
203 end-class
204
205 class (BagLike 'c 'e, CollectionWithDelete 'c 'e, CollectionWithEmptynessTest 'c 'e) => Bag 'c 'e | 'c -> 'e
206 end-class
207
208 class (Bag 'c 'e, FiniteCollection 'c 'e) => FiniteBag 'c 'e | 'c -> 'e where
209 end-class
210
211 end-module

```

## 1.14 Prelude/Default.ad

### Outline

- module Prelude.Default ..... lines 1 - 29
  - Default (typeclass) ..... lines 8 - 16
    - \* default (declaration + definition) ..... lines 9, 10
    - \* undefined (declaration + definition) ..... lines 13, 14
  - unreachable (declaration + definition) ..... lines 18, 19



– DeriveInstanceDefaultBody (template) .....lines 22 - 25

## Content

```

1 module Prelude.Default (
2   Default(..),
3   unreachable
4 ) where
5
6 {-# NoImplicitPrelude #-}
7
8 class Default 'a where
9   declare default :: 'a
10  define default := ???
11  {-# inline default #-}
12
13  declare undefined :: 'a
14  define undefined := default
15  {-# inline undefined #-}
16 end-class
17
18 declare unreachable :: Default 'a => 'a
19 define unreachable := default
20 {-# inline unreachable #-}
21
22 template DeriveInstanceDefaultBody (_ :: ClassID, ty :: TypeID, _ :: [String]) := '''
23 define default := ???
24 \{-# INLINE default #-\}
25 '''
26
27 register-derive-template Default DeriveInstanceDefaultBody
28
29 end-module

```

## 1.15 Prelude/Either.ad

### Outline

- **module** Prelude.Either ..... lines 1 - 22
  - *Either* (datatype) .....lines 11 - 13
  - *destEither* (declaration + definition) .....lines 16 - 18
  - *EqP Either* (typeclass-instance) .....line 20
  - *Eq Either* (typeclass-instance) .....line 20
  - *Ord Either* (typeclass-instance) .....line 20

### Content

```

1 module Prelude.Either (
2   Either(..),
3   destEither
4 ) where
5
6 {-# NoImplicitPrelude #-}
7 import Prelude.Bool
8 import Prelude.Eq
9 import Prelude.Ord
10

```

```

11 datatype Either 'a 'b :=
12     Left 'a
13     | Right 'b
14
15
16 declare destEither :: ('a -> 'c) -> ('b -> 'c) -> Either 'a 'b -> 'c
17 define destEither lf _ (Left v) := lf v
18     | destEither _ rf (Right v) := rf v
19
20 derive Either instances (EqP, Eq, Ord)
21
22 end-module

```

## 1.16 Prelude/Eq.ad

### Outline

- module Prelude.Eq ..... lines 1 - 148
  - ===== (declaration) .....line 14
  - EqP (typeclass) .....lines 20 - 35
    - \* === (declaration + definition) ..... lines 21, 22
    - \* /= (declaration + definition) ..... lines 24, 25
    - \* negEqP (property) .....line 27
    - \* reflEqP (property) .....line 28
    - \* symEqP (property) .....line 29
    - \* transEqP (property) .....line 30
    - \* congEqP (property) .....line 32
  - DeriveInstanceEqPBodyFull (template) .....lines 37 - 64
  - DeriveInstanceEqPBodyEq (template) .....lines 68 - 70
  - DeriveInstanceEqPBody (template) .....lines 74 - 80
  - EqP ('a -> 'b) (typeclass-instance) .....lines 84 - 86
    - \* === (definition) .....line 85
  - Eq (typeclass) .....lines 90 - 104
    - \* == (declaration + definition) ..... lines 91, 92
    - \* /= (declaration + definition) ..... lines 94, 95
    - \* negEq (test) .....line 97
    - \* reflEq (test) .....line 98
    - \* symEq (test) .....line 99
    - \* transEq (test) .....line 100
  - DeriveInstanceEqBody (template) ..... lines 106 - 133
  - Eq () (typeclass-instance) ..... lines 138 - 140
    - \* == (definition) .....line 139
  - EqP () (typeclass-instance) ..... lines 142 - 144
    - \* === (definition) .....line 143
  - EqP Bool (typeclass-instance) .....line 146
  - Eq Bool (typeclass-instance) .....line 146

## Content

```

1 module Prelude.Eq (
2   (====),
3   EqP(..),
4   Eq(..)
5 ) where
6
7 {-# NoImplicitPrelude #-}
8
9 import Prelude.Bool
10 import Prelude.Unit ()
11 import Prelude.Template
12
13 -- | build-in equality, it checks whether there is exactly the same representation of both values
14 declare non-exec (====) :: 'a -> 'a -> Prop
15
16
17 -- | sometimes, we do not want absolute identity, but define a congruence relation instead. If we
18 -- for example define a set-datastructure via distinct lists, (====) will check whether the lists
19 -- are exactly the same, while (===) checks whether the lists are permutations of each other
20 class EqP 'a where
21   declare non-exec (===) :: 'a -> 'a -> Prop
22   define non-exec (===) x y := notP (x /= y)
23
24   declare non-exec (=/=) :: 'a -> 'a -> Prop
25   define non-exec (=/=) x y := notP (x === y)
26
27   property negEqP   (x :: 'a) y   := (x === y) <=/=> (x /= y)
28   property reflEqP (x :: 'a)     := (x === x)
29   property symEqP  (x :: 'a) y   := (x === y) <=> (y === x)
30   property transEqP (x :: 'a) y z := (x === y) ==> (y === z) ==> (x === z)
31
32   property congEqP f x y := (((x :: 'a) === y) ==> ((f x :: 'a) === f y))
33
34   {-# minimal (===) | (=/=) #-}
35 end-class
36
37 template DeriveInstanceEqPBodyFull (_ :: ClassID, ty :: TypeID, _ :: [String]) := '''
38   {{ var constrs := ty.constructors;
39     var clauses := [] :: [String];
40     foreach var i in constrs.indices {
41       name-scope {
42         var (args1, constr1) := !ConstWithArgs(True, "x", constrs[i]);
43         var (args2, constr2) := !ConstWithArgs(True, "y", constrs[i]);
44         var cl := "(===) " ++ constr1 ++ " " ++ constr2 ++ " := ";
45         if (args1.length > 0) {
46           var rhs := [] :: [String];
47           foreach var xy in zip(args1, args2) {
48             rhs := insertAtEnd("(" ++ #1(xy) ++ " === " ++ #2(xy) ++ ")", rhs);
49           }
50           cl := cl ++ !Intercalate(" /\\" , rhs);
51         } else {
52           cl := cl ++ "true"
53         }
54         clauses := insertAtEnd(cl, clauses);
55       }
56     }
57   if (constrs.length > 1) {

```

```

58     clauses := insertAtEnd("(===) _ _ := false", clauses);
59   }
60   var recDecl, indentRec := "";
61   if (ty.isRec) { recDecl := "rec "; indentRec := "    "; }
62   }}
63   {{= "define non-exec " ++ recDecl ++ !Intercalate("\n          " ++ indentRec ++ "| ", clauses) =}}
64   ""
65
66   register-derive-template EqP DeriveInstanceEqPBodyFull as "full"
67
68   template DeriveInstanceEqPBodyEq (_ :: ClassID, ty :: TypeID, _ :: [String]) := '''
69   define non-exec (===) x y := bool2prop (x == y)
70   '''
71
72   register-derive-template EqP DeriveInstanceEqPBodyEq - as "eq"
73
74   template DeriveInstanceEqPBody (cl :: ClassID, ty :: TypeID, args :: [String]) := '''
75   {{% if (args.length == 0) %}}
76   {{! DeriveInstanceEqPBodyEq(cl, ty, args) !}}
77   {{% else %}}
78   {{! DeriveInstanceEqPBodyFull(cl, ty, args) !}}
79   {{% end-if %}}
80   '''
81
82   register-derive-template EqP DeriveInstanceEqPBody
83
84   instance EqP 'b => EqP ('a -> 'b) where
85     define non-exec (===) f1 f2 := forallP a. f1 a === f2 a
86   end-instance
87
88   -- | (===) is not computable. An example is an equality check for functions, which might not be
89   -- computable. For computing computability (==) is provided.
90   class EqP 'a => Eq 'a where
91     declare (==) :: 'a -> 'a -> Bool
92     define (==) x y := not (x /= y)
93
94     declare (/=) :: 'a -> 'a -> Bool
95     define (/=) x y := not (x == y)
96
97     test negEq (x :: 'a) y := (x == y) <-> (not (x /= y))
98     test reflEq (x :: 'a) := (x == x)
99     test symEq (x :: 'a) y := (x == y) <-> (y == x)
100    test transEq (x :: 'a) y z := (x == y) --> (y == z) --> (x == z)
101
102    -- property eqProp x y := (((x :: 'a) == y) === True) <=> (x === y)
103    {-# minimal (==) | (/=) #-}
104  end-class
105
106  template DeriveInstanceEqBody (_ :: ClassID, ty :: TypeID, _ :: [String]) := '''
107  {{ var constrs := ty.constructors;
108  var clauses := [] :: [String];
109  foreach var i in constrs.indices {
110    name-scope {
111      var (args1, constr1) := !ConstWithArgs(True, "x", constrs[i]);
112      var (args2, constr2) := !ConstWithArgs(True, "y", constrs[i]);
113      var cl := "(==) " ++ constr1 ++ " " ++ constr2 ++ " := ";
114      if (args1.length > 0) {
115        var rhs := [] :: [String];
116        foreach var xy in zip(args1, args2) {

```

```

117         rhs := insertAtEnd("(" ++ #1(xy) ++ " == " ++ #2(xy) ++ ")", rhs);
118     }
119     cl := cl ++ !Intercalate(" && ", rhs);
120 } else {
121     cl := cl ++ "True"
122 }
123 clauses := insertAtEnd(cl, clauses);
124 }
125 }
126 if (constrs.length > 1) {
127     clauses := insertAtEnd("(==) _ _ := False", clauses);
128 }
129 var recDecl, indentRec := "";
130 if (ty.isRec) { recDecl := "rec "; indentRec := "    "; }
131 }}
132 {{= "define " ++ recDecl ++ !Intercalate("\n    "++indentRec++"| ", clauses) =}}
133 ""
134
135 register-derive-template Eq DeriveInstanceEqBody
136
137
138 instance Eq () where
139     define (==) _ _ := True
140 end-instance
141
142 instance EqP () where
143     define non-exec (===) _ _ := true
144 end-instance
145
146 derive Bool instances (EqP, Eq)
147
148 end-module

```

## 1.17 Prelude/Foldable.ad

### Outline

- module Prelude.Foldable ..... lines 1 - 172
  - Foldable (typeclass) .....lines 18 - 50
    - \* toList (declaration) .....line 20
    - \* foldr (declaration) .....line 22
    - \* foldl (declaration) .....line 25
    - \* foldlWithAbort (declaration) .....line 28
    - \* elem (declaration) .....line 31
    - \* all (declaration) .....line 32
    - \* any (declaration) .....line 33
    - \* size (declaration) .....line 35
    - \* null (declaration) .....line 36
    - \* foldl (definition) .....line 38
    - \* foldr (definition) .....line 39
    - \* foldlWithAbort (definition) .....line 40
    - \* toList (definition) .....line 41
    - \* size (definition) .....line 42

* null (definition)	line 43
* elem (definition)	line 44
* any (definition)	line 45
* all (definition)	line 46
– Foldable ['a] 'a (typeclass-instance)	lines 52 - 80
* toList (definition)	line 53
* foldl (definition)	line 56
* foldr (definition)	line 59
* foldlWithAbort (definition)	line 62
* size (definition)	line 65
* null (definition)	line 68
* any (definition)	line 71
* all (definition)	line 74
* elem (declaration + definition)	lines 77, 78
– Foldable (Maybe 'a) 'a (typeclass-instance)	lines 83 - 104
* foldr (definition)	lines 85, 86
* foldl (definition)	lines 88, 89
* toList (definition)	lines 91, 92
* any (definition)	lines 94, 95
* all (definition)	lines 97, 98
* foldlWithAbort (definition)	lines 100 - 102
– Foldable ('a, 'a) 'a (typeclass-instance)	lines 107 - 114
* toList (definition)	line 108
* null (definition)	line 109
* size (definition)	line 112
– Foldable ('a, 'a, 'a) 'a (typeclass-instance)	lines 116 - 123
* toList (definition)	line 117
* null (definition)	line 118
* size (definition)	line 121
– Foldable ('a, 'a, 'a, 'a) 'a (typeclass-instance)	lines 125 - 132
* toList (definition)	line 126
* null (definition)	line 127
* size (definition)	line 130
– Foldable ('a, 'a, 'a, 'a, 'a) 'a (typeclass-instance)	lines 134 - 141
* toList (definition)	line 135
* null (definition)	line 136
* size (definition)	line 139
– Foldable ('a, 'a, 'a, 'a, 'a, 'a) 'a (typeclass-instance)	lines 143 - 150
* toList (definition)	line 144
* null (definition)	line 145
* size (definition)	line 148
– Foldable ('a, 'a, 'a, 'a, 'a, 'a, 'a) 'a (typeclass-instance)	lines 152 - 160
* toList (definition)	line 153
* null (definition)	line 155

* size (definition) .....	line 158
- Foldable ('a, 'a, 'a, 'a, 'a, 'a, 'a, 'a) 'a (typeclass-instance) .....	lines 162 - 169
* toList (definition) .....	line 163
* null (definition) .....	line 164
* size (definition) .....	line 167

## Content

```

1 module Prelude.Foldable (
2   Foldable(..)
3 ) where
4
5 {-# NoImplicitPrelude #-}
6
7 import Prelude.Function
8 import Prelude.Bool
9 import Prelude.Num
10 import Prelude.Eq
11 import Prelude.BasicClasses
12 import Prelude.Unit ()
13 import Prelude.List hiding (foldl, foldr, foldlWithAbort, elem, length, null, all, any)
14 import qualified Prelude.List as List
15 import Prelude.Maybe hiding (map, bind)
16
17 {-# allow-similar-names "foldl", "foldr", "foldlWithAbort", "foldrWithAbort" #-}
18 class Foldable 't 'e | 't -> 'e where
19
20   declare toList :: 't -> ['e]
21
22   declare foldr :: ('e -> 'acc -> 'acc) -> 'acc -> 't -> 'acc
23   {-# static resolve foldr #-}
24
25   declare foldl :: ('acc -> 'e -> 'acc) -> 'acc -> 't -> 'acc
26   {-# static resolve foldl #-}
27
28   declare foldlWithAbort :: ('acc -> Bool) -> ('acc -> 'e -> 'acc) -> 'acc -> 't -> 'acc
29   {-# static resolve foldlWithAbort #-}
30
31   declare elem :: Eq 'e => 'e -> 't -> Bool
32   declare all :: ('e -> Bool) -> 't -> Bool
33   declare any :: ('e -> Bool) -> 't -> Bool
34
35   declare size :: 't -> Natural
36   declare null :: 't -> Bool
37
38   define foldl f a t := List.foldl f a (toList t)
39   define foldr f a t := List.foldr f a (toList t)
40   define foldlWithAbort abort f a t := List.foldlWithAbort abort f a (toList t)
41   define toList := foldr (:) []
42   define size := foldl (fn a (_::'e) -> succ a) 0
43   define null t := foldlWithAbort not (constant (constant False)) True t
44   define elem e := any ((==) e)
45   define any p t := foldlWithAbort id (constant p) False t
46   define all p t := foldlWithAbort not (constant p) True t
47
48   {-# minimal toList | foldr #-}
49
50 end-class

```

```

51
52 instance Foldable ['a] 'a where
53   define toList := id
54   {-# inline toList #-}
55
56   define foldl := List.foldl
57   {-# inline foldl #-}
58
59   define foldr := List.foldr
60   {-# inline foldr #-}
61
62   define foldlWithAbort := List.foldlWithAbort
63   {-# inline foldlWithAbort #-}
64
65   define size := List.length
66   {-# inline size #-}
67
68   define null := List.null
69   {-# inline null #-}
70
71   define any := List.any
72   {-# inline any #-}
73
74   define all := List.all
75   {-# inline all #-}
76
77   declare elem :: Eq 'a => 'a -> ['a] -> Bool
78   define elem := List.elem
79   {-# inline elem #-}
80 end-instance
81
82
83 instance Foldable (Maybe 'a) 'a where
84
85   define foldr _ z Nothing := z
86     | foldr f z (Just x) := f x z
87
88   define foldl _ z Nothing := z
89     | foldl f z (Just x) := f z x
90
91   define toList Nothing := []
92     | toList (Just x) := [x]
93
94   define any _ Nothing := False
95     | any p (Just x) := p x
96
97   define all _ Nothing := True
98     | all p (Just x) := p x
99
100  define foldlWithAbort _ _ z Nothing := z
101    | foldlWithAbort ab f z (Just x) :=
102      if (ab z) then z else f z x
103
104 end-instance
105
106
107 instance Foldable ('a, 'a) 'a where
108   define toList (x1, x2) := [x1,x2]
109   define null := constant False

```



```

110   {-# inline null #-}
111
112   define size := constant 2
113   {-# inline size #-}
114 end-instance
115
116 instance Foldable ('a, 'a, 'a) 'a where
117   define toList (x1, x2, x3) := [x1,x2,x3]
118   define null := constant False
119   {-# inline null #-}
120
121   define size := constant 3
122   {-# inline size #-}
123 end-instance
124
125 instance Foldable ('a, 'a, 'a, 'a) 'a where
126   define toList (x1, x2, x3, x4) := [x1,x2,x3,x4]
127   define null := constant False
128   {-# inline null #-}
129
130   define size := constant 4
131   {-# inline size #-}
132 end-instance
133
134 instance Foldable ('a, 'a, 'a, 'a, 'a) 'a where
135   define toList (x1, x2, x3, x4, x5) := [x1,x2, x3, x4, x5]
136   define null := constant False
137   {-# inline null #-}
138
139   define size := constant 5
140   {-# inline size #-}
141 end-instance
142
143 instance Foldable ('a, 'a, 'a, 'a, 'a, 'a) 'a where
144   define toList (x1, x2, x3, x4, x5, x6) := [x1,x2, x3, x4, x5, x6]
145   define null := constant False
146   {-# inline null #-}
147
148   define size := constant 6
149   {-# inline size #-}
150 end-instance
151
152 instance Foldable ('a, 'a, 'a, 'a, 'a, 'a, 'a) 'a where
153   define toList (x1, x2, x3, x4, x5, x6, x7) := [x1,x2, x3, x4, x5, x6, x7]
154
155   define null := constant False
156   {-# inline null #-}
157
158   define size := constant 7
159   {-# inline size #-}
160 end-instance
161
162 instance Foldable ('a, 'a, 'a, 'a, 'a, 'a, 'a, 'a) 'a where
163   define toList (x1, x2, x3, x4, x5, x6, x7, x8) := [x1,x2, x3, x4, x5, x6, x7, x8]
164   define null := constant False
165   {-# inline null #-}
166
167   define size := constant 8
168   {-# inline size #-}

```

169 **end-instance**  
 170  
 171  
 172 **end-module**

## 1.18 Prelude/Function.ad

### Outline

- **module Prelude.Function** ..... lines 1 - 35
  - **Dot (typeclass)** .....lines 11 - 13
    - \* . (declaration) .....line 12
  - **Dot ('b -> 'c) ('a -> 'b2) ('a -> 'c) (typeclass-instance)** .....lines 15 - 18
    - \* . (definition) .....line 16
  - **id (declaration + definition)** ..... lines 20, 21
  - **constant (declaration + definition)** ..... lines 24, 25
  - **flip (declaration + definition)** ..... lines 28, 29
  - **\$ (declaration + definition)** ..... lines 31, 32

### Content

```

1 module Prelude.Function (
2   Dot(..),
3   flip,
4   constant,
5   id,
6   ($)
7 ) where
8
9 {-# NoImplicitPrelude #-}
10
11 class Dot 'a 'b 'c | 'a 'b -> 'c where
12   declare (.) :: 'a -> 'b -> 'c
13 end-class
14
15 instance Dot ('b -> 'c) ('a -> 'b2) ('a -> 'c) | 'b2 -> 'b where
16   define (.) f1 f2 x := f1 (f2 x)
17   {-# inline (.) #-}
18 end-instance
19
20 declare id :: 'a -> 'a
21 define id x := x
22 {-# inline id #-}
23
24 declare constant :: 'a -> 'b -> 'a
25 define constant x := fn _ -> x
26 {-# inline constant #-}
27
28 declare flip :: ('a -> 'b -> 'c) -> ('b -> 'a -> 'c)
29 define flip f b a := f a b
30
31 declare ($) :: ('a -> 'b) -> 'a -> 'b
32 define ($) f x := f x
33 {-# inline ($) #-}
34
35 end-module
```

## 1.19 Prelude/Functor.ad

### Outline

- module Prelude.Functor ..... lines 1 - 21
  - Functor (typeclass) .....lines 12 - 17
    - \* fmap (declaration) .....line 13
    - \* fmap\_composition (property) .....line 15
    - \* fmap\_identity (property) .....line 16
  - <\$> (alias) .....line 19

### Content

```

1 module Prelude.Functor (
2   Functor(..),
3   (<$>)
4 ) where
5
6 {-# NoImplicitPrelude #-}
7
8 import Prelude.Eq
9 import Prelude.Function
10
11 {-# static resolve Functor #-}
12 class Functor ('F :: * -> *) where
13   declare fmap :: ('a -> 'b) -> 'F 'a -> 'F 'b
14
15   property EqP ('F 'c) => fmap_composition (f :: 'b -> 'c) (g :: 'a -> 'b) (x :: 'F 'a) := fmap (f . g)
16   property EqP ('F 'a) => fmap_identity (x :: 'F 'a) := fmap id x === x
17 end-class
18
19 alias (<$>) := fmap
20
21 end-module

```

## 1.20 Prelude/List.ad

### Outline

- module Prelude.List ..... lines 1 - 243
  - List (datatype) .....line 50
  - Cons (alias) .....line 53
  - Default ['a] (typeclass-instance) .....lines 55 - 57
    - \* default (definition) .....line 56
  - map (declaration + definition) .....lines 60 - 62
  - Functor List (typeclass-instance) .....lines 64 - 67
    - \* fmap (definition) .....line 65
  - ++ (declaration + definition) .....lines 70 - 72
  - snoc (declaration + definition) .....lines 74 - 76
  - last (declaration + definition) .....lines 78 - 81
  - destSnoc (declaration + definition) .....lines 83 - 89
  - caseListSnoc (declaration + definition) .....lines 91 - 96
  - constructor-family .....line 98

- filter (declaration + definition) .....	lines 100 - 103
- partition (declaration + definition) .....	lines 105 - 112
- takeFirst (declaration + definition) .....	lines 114, 115
- takeFirstAcc (declaration + definition) .....	lines 117 - 121
- removeFirst (declaration + definition) .....	lines 123 - 127
- concat (declaration + definition) .....	lines 129, 130
- concatMap (declaration + definition) .....	lines 132, 133
- null (declaration + definition) .....	lines 135 - 137
- foldl (declaration + definition) .....	lines 139 - 141
- foldlWithAbort (declaration + definition) .....	lines 143 - 146
- scanl (declaration + definition) .....	lines 148 - 150
- foldr (declaration + definition) .....	lines 153 - 155
- scanr (declaration + definition) .....	lines 157 - 163
- length (declaration + definition) .....	lines 165, 166
- count (declaration + definition) .....	lines 168, 169
- revAppend (declaration + definition) .....	lines 171 - 173
- reverse (declaration + definition) .....	lines 175, 176
- and (declaration + definition) .....	lines 179 - 182
- andP (declaration + definition) .....	lines 184 - 186
- or (declaration + definition) .....	lines 188 - 191
- orP (declaration + definition) .....	lines 193 - 195
- all (declaration + definition) .....	lines 197 - 199
- any (declaration + definition) .....	lines 201 - 203
- allP (declaration + definition) .....	lines 205 - 207
- anyP (declaration + definition) .....	lines 209 - 211
- elem (declaration + definition) .....	lines 214, 215
- notElem (declaration + definition) .....	lines 218, 219
- elemP (declaration + definition) .....	lines 223, 224
- notElemP (declaration + definition) .....	lines 227, 228
- Eq List (typeclass-instance) .....	line 232
- EqP List (typeclass-instance) .....	line 232
- Ord List (typeclass-instance) .....	line 232
- nubBy (declaration + definition) .....	lines 234 - 236
- nub (declaration + definition) .....	lines 238, 239

## Content

```

1 module Prelude.List (
2   List(..),
3   Cons,
4   snoc,
5   last,
6   map,
7   (++),
8   filter,
9   partition,
10  removeFirst,
11  takeFirst,

```

```

12  concat,
13  concatMap,
14  null,
15  length,
16  count,
17  foldl, foldr, foldlWithAbort,
18  scanl, scanr,
19  reverse,
20  revAppend,
21  and,
22  or,
23  all,
24  any,
25  elem,
26  notElem,
27  andP,
28  orP,
29  allP,
30  anyP,
31  elemP,
32  notElemP,
33  nub,
34  nubBy
35 ) where
36
37 {-# NoImplicitPrelude #-}
38 {-# allow-similar-names "foldl", "foldr", "scanl", "scanr" #-}
39
40 import Prelude.Bool
41 import Prelude.Num
42 import Prelude.Function
43 import Prelude.Default
44 import Prelude.Eq
45 import Prelude.Ord
46 import Prelude.Functor
47 import Prelude.Maybe hiding (map)
48
49 {-# naming-convention constructor - #-}
50 datatype rec List 'a := Nil | (:) 'a (List 'a)
51 {-# naming-convention constructor reset #-}
52
53 alias Cons := (:)
54
55 instance Default ['a] where
56   define default := []
57 end-instance
58
59
60 declare map :: ('a -> 'b) -> ['a] -> ['b]
61 define rec map _ [] := []
62         | map f (x : xs) := (f x) : (map f xs)
63
64 instance Functor List where
65   define fmap := map
66   {-# inline fmap #-}
67 end-instance
68
69
70 declare (++) :: ['a] -> ['a] -> ['a]

```

```

71 define rec (++) [] ys := ys
72     | (++) (x:xs) ys := x : (xs ++ ys)
73
74 declare snoc :: ['a] -> 'a -> ['a]
75 define rec snoc [] y := [y]
76     | snoc (x:xs) y := x : (snoc xs y)
77
78 declare last :: ['a] -> Maybe 'a
79 define rec last [] := Nothing
80     | last [x] := Just x
81     | last (_:x:xs) := last (x:xs)
82
83 declare destSnoc :: ['a] -> Maybe (['a], 'a)
84 define rec destSnoc [] := Nothing
85     | destSnoc (x:xs) :=
86         case destSnoc xs of
87             Nothing -> Just ([], x)
88             | Just (ys, y) -> Just (x:ys, y)
89         end
90
91 declare caseListSnoc :: 'a -> (['c] -> 'c -> 'a) -> ['c] -> 'a
92 define caseListSnoc nilV snocV l :=
93     case (destSnoc l) of
94         Nothing -> nilV
95         | Just (xs, x) -> snocV xs x
96     end
97
98 constructor-family caseListSnoc (Nil, snoc)
99
100 declare filter :: ('a -> Bool) -> List 'a -> List 'a
101 define rec filter _ [] := []
102     | filter p (x:xs) :=
103         if (p x) then (x : (filter p xs)) else filter p xs
104
105 declare partition :: ('a -> Bool) -> List 'a -> (List 'a, List 'a)
106 define rec partition _ [] := ([], [])
107     | partition p (x:xs) :=
108         let
109             val (lt, lf) = partition p xs
110         in
111             if p x then (x : lt, lf) else (lt, x : lf)
112         end
113
114 declare takeFirst :: ('a -> Bool) -> List 'a -> Maybe (List 'a, 'a)
115 define takeFirst p := takeFirstAcc p []
116
117 declare takeFirstAcc :: ('a -> Bool) -> List 'a -> List 'a -> Maybe (List 'a, 'a)
118 define rec takeFirstAcc _ _ [] := Nothing
119     | takeFirstAcc p acc (x:xs) :=
120         if p x then Just (reverse acc ++ xs, x) else
121             takeFirstAcc p (x:acc) xs
122
123 declare removeFirst :: ('a -> Bool) -> List 'a -> List 'a
124 define removeFirst p l := case takeFirst p l of
125     Nothing -> l
126     | Just (l', _) -> l'
127     end
128
129 declare concat :: [['a]] -> ['a]

```

```

130 define concat xss := foldr (++) [] xss
131
132 declare concatMap :: ('a -> ['b]) -> ['a] -> ['b]
133 define concatMap f := concat . (map f)
134
135 declare null :: ['a] -> Bool
136 define null [] := True
137     | null (_:_) := False
138
139 declare foldl :: ('a -> 'b -> 'a) -> 'a -> ['b] -> 'a
140 define rec foldl _ z [] := z
141     | foldl f z (x:xs) := foldl f (f z x) xs
142
143 declare foldlWithAbort :: ('a -> Bool) -> ('a -> 'b -> 'a) -> 'a -> ['b] -> 'a
144 define rec foldlWithAbort _ _ z [] := z
145     | foldlWithAbort ab f z (x:xs) :=
146     if (ab z) then z else foldlWithAbort ab f (f z x) xs
147
148 declare scanl :: ('a -> 'b -> 'a) -> 'a -> ['b] -> ['a]
149 define rec scanl _ q [] := [q]
150     | scanl f q (x:xs) := q : (scanl f (f q x) xs)
151
152
153 declare foldr :: ('a -> 'b -> 'b) -> 'b -> ['a] -> 'b
154 define rec foldr _ z [] := z
155     | foldr f z (x:xs) := f x (foldr f z xs)
156
157 declare scanr :: ('a -> 'b -> 'b) -> 'b -> ['a] -> ['b]
158 define rec scanr _ q0 [] := [q0]
159     | scanr f q0 (x:xs) :=
160     case (scanr f q0 xs) of
161     [] -> unreachable
162     | q : qs -> (f x q) : (q : qs)
163     end
164
165 declare length :: ['a] -> Natural
166 define length := foldl (fn c _ -> c + 1) 0
167
168 declare count :: Eq 'a => 'a -> ['a] -> Natural
169 define count e := foldl (fn c e' -> if (e == e') then c + 1 else c) 0
170
171 declare revAppend :: ['a] -> ['a] -> ['a]
172 define rec revAppend acc [] := acc
173     | revAppend acc (x:xs) := revAppend (x:acc) xs
174
175 declare reverse :: ['a] -> ['a]
176 define reverse := revAppend []
177
178
179 declare and :: [Bool] -> Bool
180 define rec and [] := True
181     | and (False:_) := False
182     | and (True:bs) := and bs
183
184 declare non-exec andP :: [Prop] -> Prop
185 define non-exec rec andP [] := true
186     | andP (p:ps) := p /\ andP ps
187
188 declare or :: [Bool] -> Bool

```

```

189 define rec or [] := False
190     | or (True:_) := True
191     | or (False:bs) := or bs
192
193 declare non-exec orP :: [Prop] -> Prop
194 define non-exec rec orP [] := false
195     | orP (p:ps) := p \\/ orP ps
196
197 declare all :: ('a -> Bool) -> ['a] -> Bool
198 define rec all _ [] := True
199     | all f (x:xs) := if (f x) then all f xs else False
200
201 declare any :: ('a -> Bool) -> ['a] -> Bool
202 define rec any _ [] := False
203     | any f (x:xs) := if (f x) then True else any f xs
204
205 declare non-exec allP :: ('a -> Prop) -> ['a] -> Prop
206 define non-exec rec allP _ [] := true
207     | allP f (x:xs) := (f x) /\ allP f xs
208
209 declare non-exec anyP :: ('a -> Prop) -> ['a] -> Prop
210 define non-exec rec anyP _ [] := false
211     | anyP f (x:xs) := (f x) \\/ anyP f xs
212
213
214 declare elem :: Eq 'a => 'a -> ['a] -> Bool
215 define elem x := any ((==) x)
216
217
218 declare notElem :: Eq 'a => 'a -> ['a] -> Bool
219 define notElem x xs := not (elem x xs)
220 {-# inline notElem #-}
221
222
223 declare non-exec elemP :: EqP 'a => 'a -> ['a] -> Prop
224 define non-exec elemP x := anyP ((===) x)
225
226
227 declare non-exec notElemP :: EqP 'a => 'a -> ['a] -> Prop
228 define non-exec notElemP x xs := notP (elemP x xs)
229 {-# inline notElemP #-}
230
231
232 derive List instances (Eq, EqP, Ord)
233
234 declare nubBy :: ('a -> 'a -> Bool) -> ['a] -> ['a]
235 define rec nubBy _ [] := []
236     | nubBy p (x:xs) := x : nubBy p (filter (fn y -> not (p x y)) xs)
237
238 declare nub :: Eq 'a => ['a] -> ['a]
239 define nub := nubBy (==)
240 {-# inline nub #-}
241
242
243 end-module

```



## 1.21 Prelude/ListCollection.ad

### Outline

- module Prelude.ListCollection ..... lines 1 - 70
  - Collection ['e] 'e (typeclass-instance) .....lines 13 - 24
    - \* insert (definition) .....line 14
    - \* fromList (definition) .....line 17
    - \* isEmptyP (definition) .....line 20
    - \* memberP (definition) .....line 21
  - CollectionWithMembershipTest ['e] 'e (typeclass-instance) .....lines 26 - 33
    - \* occur (definition) .....line 27
    - \* member (definition) .....line 30
  - CollectionWithEmptynessTest ['e] 'e (typeclass-instance) .....lines 35 - 38
    - \* isEmpty (definition) .....line 36
  - listIntersection (declaration + definition) .....lines 41 - 48
  - CollectionWithDelete ['e] 'e (typeclass-instance) .....lines 50 - 65
    - \* delete (definition) .....line 51
    - \* filter (definition) .....line 54
    - \* partition (definition) .....line 57
    - \* difference (definition) .....line 60
    - \* intersection (definition) .....line 63
  - FiniteCollection ['e] 'e (typeclass-instance) ..... lines 67, 68

### Content

```

1 module Prelude.ListCollection (
2 ) where
3
4 {-# NoImplicitPrelude #-}
5
6 import Prelude.Eq
7 import Prelude.Bool
8 import Prelude.Maybe hiding (map)
9 import Prelude.Collection
10 import Prelude.List hiding (filter, partition)
11 import qualified Prelude.List as List
12
13 instance Collection ['e] 'e where
14   define insert := (:)
15   {-# inline insert #-}
16
17   define fromList l := l
18   {-# inline fromList #-}
19
20   define non-exec isEmptyP c := bool2prop (null c)
21   define non-exec memberP := elemP
22   {-# inline memberP #-}
23
24 end-instance
25
26 instance CollectionWithMembershipTest ['e] 'e where
27   define occur := count

```

```

28   {-# inline occur #-}
29
30   define member := elem
31   {-# inline member #-}
32
33 end-instance
34
35 instance CollectionWithEmptynessTest ['e] 'e where
36   define isEmpty := null
37   {-# inline isEmpty #-}
38 end-instance
39
40
41 declare listIntersection :: Eq 'a => ['a] -> ['a] -> ['a]
42 define rec listIntersection [] _ := []
43   | listIntersection _ [] := []
44   | listIntersection (x:xs) l :=
45     case (takeFirst ((==) x) l) of
46       Nothing -> listIntersection xs l
47       | Just (l', _) -> x:(listIntersection xs l')
48     end
49
50 instance Eq 'e => CollectionWithDelete ['e] 'e where
51   define delete e := removeFirst ((==) e)
52   {-# inline delete #-}
53
54   define filter := List.filter
55   {-# inline filter #-}
56
57   define partition := List.partition
58   {-# inline partition #-}
59
60   define difference := foldl (fn l e -> List.removeFirst ((==) e) l)
61   {-# inline difference #-}
62
63   define intersection := listIntersection
64   {-# inline intersection #-}
65 end-instance
66
67 instance FiniteCollection ['e] 'e where
68 end-instance
69
70 end-module

```

## 1.22 Prelude/Literal.ad

### Outline

- module Prelude.Literal ..... lines 1 - 138
  - BuiltInNum (type) .....line 19
  - BuiltInNumRep (type) .....line 20
  - builtInNumEq (declaration) .....line 22
  - builtInNumEqP (declaration) .....line 23
  - EqP BuiltInNum (typeclass-instance) .....lines 25 - 28
    - \* === (definition) .....line 26
  - Eq BuiltInNum (typeclass-instance) .....lines 30 - 33

* == (definition) .....	line 31
- LiteralNum (typeclass) .....	lines 36 - 38
* mkLiteralNum (declaration) .....	line 37
- LiteralNum BuiltInNum (typeclass-instance) .....	lines 41 - 44
* mkLiteralNum (definition) .....	line 42
- BuiltInChar (type) .....	line 49
- builtInCharEq (declaration) .....	line 51
- builtInCharEqP (declaration) .....	line 52
- LiteralChar (typeclass) .....	lines 55 - 57
* mkLiteralChar (declaration) .....	line 56
- EqP BuiltInChar (typeclass-instance) .....	lines 60 - 63
* === (definition) .....	line 61
- Eq BuiltInChar (typeclass-instance) .....	lines 65 - 68
* == (definition) .....	line 66
- LiteralChar BuiltInChar (typeclass-instance) .....	lines 70 - 73
* mkLiteralChar (definition) .....	line 71
- BuiltInString (type) .....	line 77
- builtInStringEq (declaration) .....	line 79
- builtInStringEqP (declaration) .....	line 80
- EqP BuiltInString (typeclass-instance) .....	lines 82 - 85
* === (definition) .....	line 83
- Eq BuiltInString (typeclass-instance) .....	lines 87 - 90
* == (definition) .....	line 88
- LiteralString (typeclass) .....	lines 93 - 95
* mkLiteralString (declaration) .....	line 94
- LiteralString BuiltInString (typeclass-instance) .....	lines 98 - 101
* mkLiteralString (definition) .....	line 99
- BuiltInDecimal (type) .....	line 104
- builtInDecimalEq (declaration) .....	line 106
- builtInDecimalEqP (declaration) .....	line 107
- builtInNumToDecimal (declaration) .....	line 109
- EqP BuiltInDecimal (typeclass-instance) .....	lines 111 - 114
* === (definition) .....	line 112
- Eq BuiltInDecimal (typeclass-instance) .....	lines 116 - 119
* == (definition) .....	line 117
- LiteralDecimal (typeclass) .....	lines 122 - 124
* mkLiteralDecimal (declaration) .....	line 123
- LiteralNum BuiltInDecimal (typeclass-instance) .....	lines 127 - 130
* mkLiteralNum (definition) .....	line 128
- LiteralDecimal BuiltInDecimal (typeclass-instance) .....	lines 132 - 135
* mkLiteralDecimal (definition) .....	line 133

## Content

```

1 module Prelude.Literal (
2   BuiltInChar,
3   BuiltInNum,
4   BuiltInDecimal,
5   BuiltInString,
6   LiteralNum(..),
7   LiteralDecimal(..),
8   LiteralChar(..),
9   LiteralString(..)
10 ) where
11
12 {-# NoImplicitPrelude #-}
13
14 import Prelude.Bool
15 import Prelude.Eq
16
17 -- | internal, build-in number representation
18 {-# allow-similar-names "builtInNumEq", "builtInNumEqP", "BuiltInNumRep" #-}
19 type BuiltInNum
20 type BuiltInNumRep
21
22 declare builtInNumEq :: BuiltInNum -> BuiltInNum -> Bool
23 declare non-exec builtInNumEqP :: BuiltInNum -> BuiltInNum -> Prop
24
25 instance EqP BuiltInNum where
26   define non-exec (===) := builtInNumEqP
27   {-# inline (===) #-}
28 end-instance
29
30 instance Eq BuiltInNum where
31   define (==) := builtInNumEq
32   {-# inline (==) #-}
33 end-instance
34
35 -- | type class for making number literals
36 class Eq 'a => LiteralNum 'a where
37   declare mkLiteralNum :: (BuiltInNumRep, BuiltInNum) -> 'a
38 end-class
39 {-# static resolve LiteralNum #-}
40
41 instance LiteralNum BuiltInNum where
42   define mkLiteralNum (_, bi) := bi
43   {-# inline mkLiteralNum #-}
44 end-instance
45
46
47 -- | internal, build-in number representation
48 {-# allow-similar-names "builtInNumEq", "builtInNumEqP", "BuiltInNumRep" #-}
49 type BuiltInChar
50
51 declare builtInCharEq :: BuiltInChar -> BuiltInChar -> Bool
52 declare non-exec builtInCharEqP :: BuiltInChar -> BuiltInChar -> Prop
53
54 -- | type class for making number literals
55 class Eq 'a => LiteralChar 'a where
56   declare mkLiteralChar :: BuiltInChar -> 'a
57 end-class

```

```

58 {-# static resolve LiteralChar #-}
59
60 instance EqP BuiltInChar where
61   define non-exec (===) := builtInCharEqP
62   {-# inline (===) #-}
63 end-instance
64
65 instance Eq BuiltInChar where
66   define (==) := builtInCharEq
67   {-# inline (==) #-}
68 end-instance
69
70 instance LiteralChar BuiltInChar where
71   define mkLiteralChar c := c
72   {-# inline mkLiteralChar #-}
73 end-instance
74
75
76 -- | internal, build-in number representation
77 type BuiltInString
78
79 declare builtInStringEq :: BuiltInString -> BuiltInString -> Bool
80 declare non-exec builtInStringEqP :: BuiltInString -> BuiltInString -> Prop
81
82 instance EqP BuiltInString where
83   define non-exec (===) := builtInStringEqP
84   {-# inline (===) #-}
85 end-instance
86
87 instance Eq BuiltInString where
88   define (==) := builtInStringEq
89   {-# inline (==) #-}
90 end-instance
91
92 -- | type class for making number literals
93 class Eq 'a => LiteralString 'a where
94   declare mkLiteralString :: BuiltInString -> 'a
95 end-class
96 {-# static resolve LiteralString #-}
97
98 instance LiteralString BuiltInString where
99   define mkLiteralString s := s
100   {-# inline mkLiteralString #-}
101 end-instance
102
103 -- | internal, build-in number representation
104 type BuiltInDecimal
105
106 declare builtInDecimalEq :: BuiltInDecimal -> BuiltInDecimal -> Bool
107 declare non-exec builtInDecimalEqP :: BuiltInDecimal -> BuiltInDecimal -> Prop
108
109 declare builtInNumToDecimal :: BuiltInNum -> BuiltInDecimal
110
111 instance EqP BuiltInDecimal where
112   define non-exec (===) := builtInDecimalEqP
113   {-# inline (===) #-}
114 end-instance
115
116 instance Eq BuiltInDecimal where

```

```

117   define (==) := builtInDecimalEq
118   {-# inline (==) #-}
119 end-instance
120
121 -- | type class for making decimal literals
122 class LiteralNum 'a => LiteralDecimal 'a where
123   declare mkLiteralDecimal :: BuiltInDecimal -> 'a
124 end-class
125 {-# static resolve LiteralDecimal #-}
126
127 instance LiteralNum BuiltInDecimal where
128   define mkLiteralNum (_, i) := builtInNumToDecimal i
129   {-# inline mkLiteralNum #-}
130 end-instance
131
132 instance LiteralDecimal BuiltInDecimal where
133   define mkLiteralDecimal d := d
134   {-# inline mkLiteralDecimal #-}
135 end-instance
136
137
138 end-module

```

## 1.23 Prelude/Maybe.ad

### Outline

- module Prelude.Maybe ..... lines 1 - 52
  - Maybe (datatype) .....lines 20 - 22
  - Default (Maybe 'a) (typeclass-instance) .....lines 24 - 26
    - \* default (definition) .....line 25
  - map (declaration + definition) .....lines 28 - 30
  - Functor Maybe (typeclass-instance) .....lines 32 - 35
    - \* fmap (definition) .....line 33
  - bind (declaration + definition) .....lines 37 - 39
  - fromMaybe (declaration + definition) .....lines 41 - 43
  - fromMaybeMap (declaration + definition) ..... lines 45, 46
  - generate-code .....line 48
  - EqP Maybe (typeclass-instance) .....line 50
  - Eq Maybe (typeclass-instance) .....line 50
  - Ord Maybe (typeclass-instance) .....line 50

### Content

```

1 module Prelude.Maybe (
2   Maybe(..),
3   isJust,
4   isNothing,
5   fromMaybe,
6   map,
7   fromMaybeMap,
8   bind
9 ) where
10

```

```

11 {-# NoImplicitPrelude #-}
12
13 import Prelude.Bool
14 import Prelude.Eq
15 import Prelude.Ord
16 import Prelude.Default
17 import Prelude.Template
18 import Prelude.Functor
19
20 datatype Maybe 'a :=
21     Nothing
22     | Just 'a
23
24 instance Default (Maybe 'a) where
25     define default := Nothing
26 end-instance
27
28 declare map :: ('a -> 'b) -> Maybe 'a -> Maybe 'b
29 define map _ Nothing := Nothing
30     | map f (Just x) := Just (f x)
31
32 instance Functor Maybe where
33     define fmap := map
34     {-# inline fmap #-}
35 end-instance
36
37 declare bind :: Maybe 'a -> ('a -> Maybe 'b) -> Maybe 'b
38 define bind Nothing _ := Nothing
39     | bind (Just x) f := f x
40
41 declare fromMaybe :: 'a -> Maybe 'a -> 'a
42 define fromMaybe d Nothing := d
43     | fromMaybe _d (Just v) := v
44
45 declare fromMaybeMap :: 'b -> ('a -> 'b) -> Maybe 'a -> 'b
46 define fromMaybeMap d f m := fromMaybe d (map f m)
47
48 generate-code GenerateRecognisers(type Maybe)
49
50 derive Maybe instances (EqP, Eq, Ord)
51
52 end-module

```

## 1.24 Prelude/Monad.ad

### Outline

- module Prelude.Monad ..... lines 1 - 114
  - Monad (typeclass) .....lines 28 - 39
    - \* >>= (declaration) .....line 29
    - \* >> (declaration) .....line 30
    - \* return (declaration) .....line 31
    - \* >> (definition) .....line 33
    - \* monad\_left\_identity (property) .....line 36
    - \* monad\_right\_identity (property) .....line 37
    - \* monad\_assoc (property) .....line 38

- bind (alias) .....	.line 41
- pure (alias) .....	.line 42
- =<< (declaration + definition) .....	lines 44, 45
- guard (declaration + definition) .....	lines 48 - 50
- sequence' (declaration + definition) .....	lines 53 - 61
- sequence (declaration + definition) .....	lines 64, 65
- sequence_ (declaration + definition) .....	lines 68, 69
- mapM' (declaration + definition) .....	lines 72, 73
- mapM (declaration + definition) .....	lines 75, 76
- mapM_ (declaration + definition) .....	lines 79, 80
- Monad List (typeclass-instance) .....	lines 84 - 90
* return (definition) .....	.line 85
* >=> (definition) .....	.line 88
- Monad Maybe (typeclass-instance) .....	lines 92 - 98
* return (definition) .....	.line 93
* >=> (definition) .....	.line 96
- MonadFail (typeclass) .....	lines 103 - 107
* fail (declaration) .....	.line 104
* monad_fail (property) .....	.line 106
- MonadFail Maybe (typeclass-instance) .....	lines 109 - 112
* fail (definition) .....	.line 110

## Content

```

1 module Prelude.Monad (
2   Monad(..),
3   bind, (=<<),
4   pure,
5   sequence,
6   sequence_,
7   mapM,
8   mapM_,
9   guard,
10  MonadFail(..)
11 ) where
12
13 {-# NoImplicitPrelude #-}
14
15 import Prelude.Function
16 import Prelude.Bool
17 import Prelude.Eq
18 import Prelude.Unit ()
19 import Prelude.String
20 import Prelude.BasicClasses
21 import Prelude.List hiding (foldl, foldr, elem, length, null)
22 import qualified Prelude.List as List
23 import qualified Prelude.Maybe as Maybe
24 import Prelude.Maybe hiding (map, bind)
25
26
27 {-# static resolve Monad #-}
28 class Monad ('M :: * -> *) where

```



```

29 declare (>>=) :: 'M 'a -> ('a -> 'M 'b) -> 'M 'b
30 declare (>>)  :: 'M 'a -> 'M 'b -> 'M 'b
31 declare return :: 'a -> 'M 'a
32
33 define (>>) m k := m >>= (constant k)
34 {-# inline (>>) #-}
35
36 property EqP ('M 'b) => monad_left_identity (f :: 'a -> 'M 'b) a := ((return a) >>= f) === (f a)
37 property EqP ('M 'b) => monad_right_identity (a :: 'M 'b) := (a >>= return) === a
38 property EqP ('M 'b) => monad_assoc (m :: 'M 'a) (f :: 'a -> 'M 'c) (g :: 'c -> 'M 'b) := ((m >>= f) >> g) == (m >>= (f >> g))
39 end-class
40
41 alias bind := (>>=)
42 alias pure := return
43
44 declare (=⟨⟨) :: !Monad 'M => ('a -> 'M 'b) -> 'M 'a -> 'M 'b
45 define (=⟨⟨) f m := m >>= f
46 {-# inline (=⟨⟨) #-}
47
48 declare guard :: (!Monoid ('M ()), !Monad 'M) => Bool -> 'M ()
49 define guard True := return ()
50     | guard False := mempty
51 {-# inline guard #-}
52
53 declare sequence' :: (['a] -> 'mas)
54     -> ('ma -> ('a -> 'mas) -> 'mas)
55     -> ('mas -> (['a] -> 'mas) -> 'mas)
56     -> ['ma]
57     -> 'mas
58 define sequence' ret bindl bindl :=
59     List.foldr (fn p q ->
60         bindl p (fn x -> bindl q (fn xs -> ret (x:xs)))
61     ) (ret [])
62
63
64 declare sequence :: !Monad 'M => ['M 'a] -> 'M ['a]
65 define sequence := sequence' return bind bind
66 {-# inline sequence #-}
67
68 declare sequence_ :: !Monad 'M => ['M 'a] -> 'M ()
69 define sequence_ l := sequence l >> return ()
70 {-# inline sequence_ #-}
71
72 declare mapM' :: (['b] -> 'mbs) -> ('mb -> ('b -> 'mbs) -> 'mbs) -> ('mbs -> (['b] -> 'mbs) -> 'mbs) ->
73 define mapM' ret bl bl f xs := sequence' ret bl bl (map f xs)
74
75 declare mapM :: !Monad 'M => ('a -> 'M 'b) -> ['a] -> 'M ['b]
76 define mapM := mapM' return bind bind
77 {-# inline mapM #-}
78
79 declare mapM_ :: !Monad 'M => ('a -> 'M 'b) -> ['a] -> 'M ()
80 define mapM_ f xs := (mapM f xs) >> return ()
81 {-# inline mapM_ #-}
82
83
84 instance Monad List where
85     define return x := [x]
86     {-# inline return #-}
87

```

```

88   define (>>=) xs f := concatMap f xs
89   {-# inline (>>=) #-}
90 end-instance
91
92 instance Monad Maybe where
93   define return := Just
94   {-# inline return #-}
95
96   define (>>=) := Maybe.bind
97   {-# inline (>>=) #-}
98 end-instance
99
100
101
102 {-# static resolve MonadFail #-}
103 class !Monad 'M => MonadFail ('M :: * -> *) where
104   declare fail :: String -> 'M 'a
105
106   property EqP ('M 'b) => monad_fail (f :: 'a -> 'M 'b) s := ((fail s) >>= f) === fail s
107 end-class
108
109 instance MonadFail Maybe where
110   define fail := constant Nothing
111   {-# inline fail #-}
112 end-instance
113
114 end-module

```

## 1.25 Prelude/Num.ad

### Outline

- module Prelude.Num ..... lines 1 - 102
  - Natural (datatype) .....lines 19 - 21
  - Default Natural (typeclass-instance) .....lines 23 - 26
    - \* default (definition) .....line 24
  - EqP Natural (typeclass-instance) .....line 28
  - Eq Natural (typeclass-instance) .....line 28
  - Ord Natural (typeclass-instance) .....line 28
  - NumPlus (typeclass) .....lines 30 - 32
    - \* + (declaration) .....line 31
  - NumMinus (typeclass) .....lines 34 - 36
    - \* - (declaration) .....line 35
  - NumMult (typeclass) .....lines 38 - 40
    - \* \* (declaration) .....line 39
  - NumDiv (typeclass) .....lines 42 - 44
    - \* / (declaration) .....line 43
  - NumDivMod (typeclass) .....lines 46 - 54
    - \* divmod (declaration) .....line 47
    - \* div (declaration) .....line 48
    - \* mod (declaration) .....line 49
    - \* divmod (definition) .....line 51

* div (definition) .....	line 52
* mod (definition) .....	line 53
- LiteralNum Natural (typeclass-instance) .....	lines 57 - 59
* mkLiteralNum (definition) .....	line 58
- NumPlus Natural (typeclass-instance) .....	lines 61 - 64
* + (definition) .....	lines 62, 63
- NumMinus Natural (typeclass-instance) .....	lines 66 - 70
* - (definition) .....	lines 67 - 69
- NumMult Natural (typeclass-instance) .....	lines 72 - 75
* * (definition) .....	lines 73, 74
- NumDivMod Natural (typeclass-instance) .....	lines 77 - 86
* divmod (definition) .....	lines 78 - 85
- NumDiv Natural (typeclass-instance) .....	lines 88 - 91
* / (definition) .....	line 89
- LiteralNum Bool (typeclass-instance) .....	lines 94 - 99
* mkLiteralNum (definition) .....	lines 95, 96

## Content

```

1 module Prelude.Num (
2   NumPlus(..),
3   NumMinus(..),
4   NumMult(..),
5   NumDiv(..),
6   NumDivMod(..),
7   Natural(..)
8 ) where
9
10 {-# NoImplicitPrelude #-}
11
12 import Prelude.Bool
13 import Prelude.Default
14 import Prelude.Eq
15 import Prelude.Ord
16 import Prelude.Literal
17 import Prelude.Tuple
18
19 datatype rec Natural :=
20   Zero
21   | Suc Natural
22
23 instance Default Natural where
24   define default := Zero
25   {-# inline default #-}
26 end-instance
27
28 derive Natural instances (EqP, Eq, Ord)
29
30 class NumPlus 'a where
31   declare (+) :: 'a -> 'a -> 'a
32 end-class
33
34 class NumMinus 'a where
35   declare (-) :: 'a -> 'a -> 'a

```

```

36 end-class
37
38 class NumMult 'a where
39   declare (*) :: 'a -> 'a -> 'a
40 end-class
41
42 class NumDiv 'a where
43   declare (/) :: 'a -> 'a -> 'a
44 end-class
45
46 class NumDivMod 'a where
47   declare divmod :: 'a -> 'a -> ('a, 'a)
48   declare div :: 'a -> 'a -> 'a
49   declare mod :: 'a -> 'a -> 'a
50
51   define divmod (x :: 'a) y := (x 'div' y, x 'mod' y)
52   define div x y := fst (divmod x y)
53   define mod x y := snd (divmod x y)
54 end-class
55
56
57 instance LiteralNum Natural where
58   define mkLiteralNum _ := ???
59 end-instance
60
61 instance NumPlus Natural where
62   define rec (+) n Zero := n
63     | (+) n (Suc m) := (+) (Suc n) m
64 end-instance
65
66 instance NumMinus Natural where
67   define rec (-) n Zero := n
68     | (-) Zero _ := Zero
69     | (-) (Suc n) (Suc m) := (-) n m
70 end-instance
71
72 instance NumMult Natural where
73   define rec (*) _ Zero := Zero
74     | (*) n (Suc m) := (n + (n * m))
75 end-instance
76
77 instance NumDivMod Natural where
78   define rec divmod _ Zero := ???
79     | divmod x y :=
80       if x < y then
81         (0, x)
82       else
83         let
84           val (d, m) = divmod (x - y) y
85           in (d+1, m) end
86 end-instance
87
88 instance NumDiv Natural where
89   define (/) := div
90   {-# inline (/) #-}
91 end-instance
92
93
94 instance LiteralNum Bool where

```

```

95   define mkLiteralNum (_, 0) := False
96       | mkLiteralNum (_, _) := True
97
98   {-# set-num-literal-boundaries 0 1 #-}
99 end-instance
100
101
102 end-module

```

## 1.26 Prelude/Ord.ad

### Outline

- module Prelude.Ord ..... lines 1 - 120
  - Ordering (datatype) ..... lines 16 - 19
  - EqP Ordering (typeclass-instance) ..... line 21
  - Eq Ordering (typeclass-instance) ..... line 21
  - Default Ordering (typeclass-instance) ..... lines 23 - 26
    - \* default (definition) ..... line 24
  - Ord (typeclass) ..... lines 29 - 63
    - \* compare (declaration + definition) ..... lines 30 - 37
    - \* < (declaration) ..... line 39
    - \* <= (declaration) ..... line 40
    - \* > (declaration) ..... line 41
    - \* >= (declaration) ..... line 42
    - \* min (declaration) ..... line 43
    - \* max (declaration) ..... line 44
    - \* <= (definition) ..... line 46
    - \* < (definition) ..... line 47
    - \* >= (definition) ..... line 48
    - \* > (definition) ..... line 49
    - \* max (definition) ..... line 51
    - \* min (definition) ..... line 52
    - \* lt\_alt\_defs (test) ..... lines 54 - 57
    - \* lt\_irrefl (test) ..... line 59
    - \* lt\_trans (test) ..... line 60
  - Ord () (typeclass-instance) ..... lines 65 - 67
    - \* compare (definition) ..... line 66
  - comparing (declaration + definition) ..... lines 69, 70
  - lexCompare (declaration + definition) ..... lines 72 - 77
  - DeriveInstanceOrdBody (template) ..... lines 81 - 113
  - Ord Ordering (typeclass-instance) ..... line 117
  - Ord Bool (typeclass-instance) ..... line 118

## Content

```

1 module Prelude.Ord (
2   type Ordering(..),
3   class Ord(..),
4   comparing,
5   lexCompare
6 ) where
7
8 {-# NoImplicitPrelude #-}
9
10 import Prelude.Bool
11 import Prelude.Eq
12 import Prelude.Template
13 import Prelude.Default
14 import Prelude.Unit()
15
16 datatype Ordering :=
17   LT
18   | EQ
19   | GT
20
21 derive Ordering instances (EqP, Eq)
22
23 instance Default Ordering where
24   define default := EQ
25   {-# inline default #-}
26 end-instance
27
28
29 class Eq 'a => Ord 'a where
30   declare compare :: 'a -> 'a -> Ordering
31
32   define compare x y :=
33     cases
34       (x == y) -> EQ
35       | (x < y) -> LT
36       | otherwise -> GT
37   end
38
39 declare (<) :: 'a -> 'a -> Bool
40 declare (<=) :: 'a -> 'a -> Bool
41 declare (>) :: 'a -> 'a -> Bool
42 declare (>=) :: 'a -> 'a -> Bool
43 declare min :: 'a -> 'a -> 'a
44 declare max :: 'a -> 'a -> 'a
45
46 define (<=) x y := (compare x y /= GT)
47 define (<) x y := (compare x y == LT)
48 define (>=) x y := (compare x y /= LT)
49 define (>) x y := (compare x y == GT)
50
51 define max x y := if (x <= y) then y else x
52 define min x y := if (x <= y) then x else y
53
54 test lt_alt_defs (a :: 'a) b :=
55   ((a > b) <-> (b < a)) &&
56   ((a >= b) <-> (b <= a)) &&
57   ((a <= b) <-> ((a < b) || (a == b)))

```

```

58
59  test lt_irrefl (a :: 'a) := not (a < a)
60  test lt_trans (a :: 'a) b c := ((a < b) && (b < c)) --> (a < c)
61
62  {-# minimal compare | (<) #-}
63  end-class
64
65  instance Ord () where
66    define compare () () := EQ
67  end-instance
68
69  declare comparing :: Ord 'a => ('b -> 'a) -> 'b -> 'b -> Ordering
70  define comparing p x y := compare (p x) (p y)
71
72  declare lexCompare :: Ord 'a => 'a -> 'a -> Ordering -> Ordering
73  define lexCompare x y next :=
74    case (compare x y) of
75      EQ -> next
76    | other -> other
77  end
78  {-# inline lexCompare #-}
79
80
81  template DeriveInstanceOrdBody (_ :: ClassID, ty :: TypeID, _ :: [String]) := '''
82    {{ var constrs := ty.constructors;
83      var clauses := [] :: [String];
84      foreach var i in constrs.indices {
85        name-scope {
86          var (args1, constr1) := !ConstWithArgs(True, "x", constrs[i]);
87          var (args2, constr2) := !ConstWithArgs(True, "y", constrs[i]);
88          var cl := "compare " ++ constr1 ++ " " ++ constr2 ++ " := ";
89          if (args1.length > 0) {
90            var rhs := [] :: [String];
91            for var j := 1 to (args1.length - 1) {
92              var el := "lexCompare " ++ args1[j-1] ++ " " ++ args2[j-1];
93              rhs := insertAtEnd(el, rhs);
94            }
95            rhs := insertAtEnd("compare " ++ args1[args1.length - 1] ++ " " ++ args2[args1.length - 1], rhs);
96            cl := cl ++ !Intercalate(" $ ", rhs);
97          } else {
98            cl := cl ++ "EQ"
99          }
100          clauses := insertAtEnd(cl, clauses);
101
102          var constrWc := !ConstWithWildcardArgs(True, constrs[i]);
103          if (i < constrs.length - 1) {
104            clauses := insertAtEnd("compare " ++ constrWc ++ " _ := LT", clauses);
105            clauses := insertAtEnd("compare _ " ++ constrWc ++ " := GT", clauses);
106          }
107        }
108      }
109      var recDecl, indentRec := "";
110      if (ty.isRec) { recDecl := "rec "; indentRec := "  "; }
111    }}
112    {{= "define " ++ recDecl ++ !Intercalate("\n      " ++ indentRec ++ "| ", clauses) =}}
113    '''
114
115  register-derive-template Ord DeriveInstanceOrdBody
116

```

```

117 derive Ordering instances Ord
118 derive Bool instances Ord
119
120 end-module

```

## 1.27 Prelude/String.ad

### Outline

- module Prelude.String ..... lines 1 - 21
  - String (type) .....line 12
  - mkString (declaration) .....line 14
  - LiteralString String (typeclass-instance) .....lines 16 - 19
    - \* mkLiteralString (definition) .....line 17

### Content

```

1 module Prelude.String (
2   String(..),
3   mkString
4 ) where
5
6 {-# NoImplicitPrelude #-}
7
8 import Prelude.List ()
9 import Prelude.Char
10 import Prelude.Literal
11
12 type String := [Char]
13
14 declare mkString :: BuiltInString -> String
15
16 instance LiteralString String where
17   define mkLiteralString := mkString
18   {-# inline mkLiteralString #-}
19 end-instance
20
21 end-module

```

## 1.28 Prelude/Template.ad

### Outline

- module Prelude.Template ..... lines 2 - 258
  - GetTypeVar (template) .....lines 30 - 39
  - GetTypeVars (template) .....lines 42 - 48
  - GetTypeVarsNumbered (template) .....lines 51 - 58
  - Intercalate (template) .....lines 61 - 71
  - Commafy (template) .....lines 74 - 76
  - BuildTuple (template) .....lines 79 - 88
  - BuildTupleLeft (template) .....lines 91 - 104
  - BuildTupleRight (template) .....lines 108 - 121
  - IsSimpleEnumDatatype (template) .....lines 124 - 133
  - AddArgs (template) .....lines 137 - 146



- ConstWithWildcardArgs (template) .....	lines 151 - 154
- ConstWithArgs (template) .....	lines 162 - 165
- TypeWithArgs (template) .....	lines 173 - 176
- GenerateRecognisers (template) .....	lines 179 - 189
- ClassWithArgs (template) .....	lines 197 - 200
- BuildInstanceHeader (template) .....	lines 205 - 212
- BuildInstance (template) .....	lines 215 - 219
- DeriveInstanceComputeSuperDefault (template) .....	lines 226 - 238
- DeriveInstance (template) .....	lines 243 - 248
- DerivingInstance (template) .....	lines 251 - 255

## Content

```

1 -- auxiliary template functions
2 module Prelude.Template (
3   GetTypeVar,
4   GetTypeVars,
5   GetTypeVarsNumbered,
6   Intercalate,
7   Commafy,
8   BuildTuple,
9   BuildTupleLeft,
10  BuildTupleRight,
11  BuildInstanceHeader,
12  BuildInstance,
13  GenerateRecognisers,
14
15  IsSimpleEnumDatatype,
16
17  ConstWithArgs,
18  TypeWithArgs,
19  ClassWithArgs,
20  ConstWithWildcardArgs,
21
22  DeriveInstanceComputeSuperDefault,
23  DeriveInstance,
24  DerivingInstance
25 ) where
26
27 {-# NoImplicitPrelude #-}
28
29 -- generate the type variable with given number 'a, 'b, 'c, ...
30 template GetTypeVar (n :: Nat) :: String := {{
31   var cn := n % 26;
32   n := n / 26;
33   var result := "\" ++ chr(cn + 97);
34   if (n == 0) {
35     return result;
36   } else {
37     return (result ++ n);
38   }
39 }}
40
41 -- generate a list of type variable of given length, 'a, 'b, ...
42 template GetTypeVars (n :: Nat) :: [String] := {{
43   var result := [] :: [String];

```

```

44   for var y := 1 to n {
45     result := insertAtEnd(!GetTypeVar(y-1), result);
46   };
47   return result;
48 }}
49
50 -- generate a list of type variable of given length, 'p1, 'p2, ...
51 template GetTypeVarsNumbered (p :: String, n :: Nat) :: [String] := {{
52   var result := [] :: [String];
53   for var y := 1 to n {
54     var vn := "" ++ p ++ y;
55     result := insertAtEnd(vn, result);
56   };
57   return result;
58 }}
59
60
61 template Intercalate (sep :: String, s :: [String]) :: String := {{
62   if (s.length == 0) {
63     return ""
64   } else {
65     var result := s[0];
66     for var i := 1 to (s.length - 1) {
67       result := result ++ sep ++ s[i];
68     };
69     return result;
70   }
71 }}
72
73 -- commafy an array of strings and return the resulting string
74 template Commafy (s :: [String]) :: String := {{
75   return !Intercalate(", ", s)
76 }}
77
78 -- commafy an array and also add parenthesis around it, i.e. it creates a tuple
79 template BuildTuple (s :: [String]) :: String := {{
80   if (s.length == 0) {
81     error("tuples should contain at least one element");
82     return "error";
83   } else-if (s.length == 1) {
84     return s[0];
85   } else {
86     return "(" ++ !Commafy(s) ++ ")";
87   }
88 }}
89
90 -- creates a left-leaning tuple, i.e something like ((a, b), c), d)
91 template BuildTupleLeft (s :: [String]) :: String := {{
92   if (s.length == 0) {
93     error("tuples should contain at least one element");
94     return "error";
95   } else-if (s.length == 1) {
96     return s[0];
97   } else {
98     var res := s[0];
99     for var i := 1 to (s.length - 1) {
100       res := "(" ++ res ++ ", " ++ s[i] ++ ")";
101     }
102     return res;

```

```

103   }
104 }}
105
106
107 -- creates a right-leaning tuple, i.e something like (a, (b, (c, d)))
108 template BuildTupleRight (s :: [String]) :: String := {{
109   if (s.length == 0) {
110     error("tuples should contain at least one element");
111     return "error";
112   } else-if (s.length == 1) {
113     return s[0];
114   } else {
115     var res := s[s.length - 1];
116     for var i := (s.length - 2) downto 0 {
117       res := "(" ++ s[i] ++ ", " ++ res ++"";
118     }
119     return res;
120   }
121 }}
122
123
124 template IsSimpleEnumDatatype (ty :: TypeID) :: Bool := {{
125   if (not (ty.typeVariety == "datatype")) return False;
126
127   foreach var c in ty.constructors {
128     if (c.argsNo > 0) return False;
129   }
130   if (ty.constructors.length == 0) return False;
131
132   return True;
133 }}
134
135
136 -- | add all the arguments separated by space to a base.
137 template AddArgs (forceParens :: Bool, base :: String, args :: [String]) :: String := {{
138   var res := base;
139   foreach var arg in args {
140     res := res ++ " " ++ arg;
141   }
142   if ((args.length > 0) && forceParens) {
143     res := "(" ++ res ++"";
144   }
145   return res
146 }}
147
148 -- | add the appropriate number of wildcard arguments to a constant and return
149 -- the resulting string. If at least one argument is used and if the parameter
150 -- 'forceParens' is set, parenthesis are added around the result
151 template ConstWithWildcardArgs (forceParens :: Bool, c :: ConstID) :: String := {{
152   var args := replicate(c.argsNo, "_");
153   return (!AddArgs(forceParens, c, args));
154 }}
155
156
157 -- | add the appropriate number of arguments to a constant and return
158 -- the added arguments as well as the resulting string. The names of the
159 -- arguments are generated using the given prefix.
160 -- If at least one argument is used and if the parameter
161 -- 'forceParens' is set, parenthesis are added around the result

```

```

162 template ConstWithArgs (forceParens :: Bool, prefix :: String, c :: ConstID) :: ([String], String) := {{
163   var args := generateNames(prefix, c.argsNo);
164   return (args, !AddArgs(forceParens, c, args));
165 }}
166
167
168 -- | add the appropriate number of arguments to a type and return
169 -- the added arguments as well as the resulting string. The names of the
170 -- arguments are take from the definition of the type.
171 -- If at least one argument is used and if the parameter
172 -- 'forceParens' is set, parenthesis are added around the result
173 template TypeWithArgs (forceParens :: Bool, ty :: TypeID) :: ([String], String) := {{
174   var args := ty.args;
175   return (args, !AddArgs(forceParens, ty, args));
176 }}
177
178 -- | build recognisers for all constructors of a data-type
179 template GenerateRecognisers (ty :: TypeID) := '''
180 {{ var (_, tyFull) := !TypeWithArgs(False, ty) }}
181 {{% foreach var constr in ty.constructors %}}
182 {{ var name := "is" ++ constr;
183   var constrWithWc := !ConstWithWildcardArgs(True, constr)}}
184 declare {{= name =}} :: {{=tyFull=}} -> Bool
185 define {{= name =}} {{= constrWithWc =}} := True
186   | {{= name =}} _ := False
187
188 {{% end-foreach %}}
189 '''
190
191
192 -- | add the appropriate number of arguments to a type and return
193 -- the added arguments as well as the resulting string. The names of the
194 -- arguments are take from the definition of the class.
195 -- If at least one argument is used and if the parameter
196 -- 'forceParens' is set, parenthesis are added around the result
197 template ClassWithArgs (forceParens :: Bool, cl :: ClassID) :: ([String], String) := {{
198   var args := cl.args;
199   return (args, !AddArgs(forceParens, cl, args));
200 }}
201
202
203
204 -- | given a list of super-constraints and the main instance, build the instance-header string
205 template BuildInstanceHeader (super :: [String], inst :: String) := '''
206 {{ var superS := "";
207   if (super.length > 0) {
208     superS := !BuildTuple(super) ++ " => ";
209   }
210 }}
211 instance {{= superS =}} {{= inst =}} where
212 '''
213
214 -- | call 'BuildInstanceHeader' followed by a body and the end-instance
215 template BuildInstance (super :: [String], inst :: String, body :: String) := '''
216 {{! BuildInstanceHeader(super, inst) !}}
217   {{= body =}}
218 end-instance
219 '''
220

```

```

221 -- | Auxiliary template for building instances of type-classes. Given a type-class with
222 -- exactly one argument and a type-ID to use for this argument, it looks up the arguments
223 -- of the type and uses each found argument with the main type-class as a class constraint of this instance.
224 -- This is useful for defining instances of classes like Eq automatically.
225 -- It returns the super-constraints, the instance as well as the full type and the used type args
226 template DeriveInstanceComputeSuperDefault (cl :: ClassID, ty :: TypeID) :: ([String], String, [String])
227   if (not (cl.argsNo == 1)) error("only type-classes with 1 argument supported");
228
229   var (tyArgs, tyFull) := !TypeWithArgs(True, ty);
230   var inst := cl ++ " " ++ tyFull;
231
232   var super := [] :: [String];
233   foreach var t in tyArgs {
234     super := insertAtEnd(cl ++ " " ++ t, super);
235   }
236
237   return (super, inst, tyArgs);
238 }}
239
240 -- | the build-in template used for "derive" statements. It gets a template to compute the class constraints,
241 -- the instance and a template to compute the body. If for a type-class no other template is registered,
242 -- 'DeriveInstanceComputeSuperDefault' is used automatically
243 template DeriveInstance (cl :: ClassID, ty :: TypeID,
244   superInst :: TemplateID(ClassID -> TypeID -> ([String], String, [String])),
245   body :: TemplateID(ClassID -> TypeID -> [String] -> Text)) := '''
246 {{ var (super, inst, tyArgs) := !superInst(cl, ty); }}
247 {{! DerivingInstance (cl, ty, tyArgs, super, inst, body) !}}
248 '''
249
250 -- | the build-in template used for "derivig" statements.
251 template DerivingInstance (cl :: ClassID, ty :: TypeID, tyArgs :: [String], super :: [String], inst :: String) := '''
252 {{! BuildInstanceHeader(super, inst) !}}
253 {{= !body(cl, ty, tyArgs) =}}
254 end-instance
255 '''
256
257
258 end-module

```

## 1.29 Prelude/Text.ad

### Outline

- module Prelude.Text ..... lines 1 - 43
  - Text (datatype) .....line 18
  - pack (declaration + definition) ..... lines 20, 21
  - unpack (declaration + definition) ..... lines 23, 24
  - LiteralString Text (typeclass-instance) .....lines 26 - 29
    - \* mkLiteralString (definition) .....line 27
  - EqP Text (typeclass-instance) .....line 31
  - Eq Text (typeclass-instance) .....line 31
  - Ord Text (typeclass-instance) .....line 31
  - Semigroup Text (typeclass-instance) .....lines 33 - 35
    - \* <> (definition) .....line 34
  - Monoid Text (typeclass-instance) .....lines 37 - 39

\* mempty (definition) .....line 38

## Content

```

1 module Prelude.Text (
2   Text,
3   pack,
4   unpack
5 ) where
6
7 {-# NoImplicitPrelude #-}
8
9 import Prelude.List ()
10 import Prelude.Eq
11 import Prelude.BasicClasses
12 import Prelude.Ord
13 import Prelude.String
14 import Prelude.Function
15 import Prelude.Literal
16 import Prelude.Bool
17
18 datatype Text := Text String
19
20 declare pack :: String -> Text
21 define pack := Text
22
23 declare unpack :: Text -> String
24 define unpack (Text s) := s
25
26 instance LiteralString Text where
27   define mkLiteralString := pack . mkLiteralString
28   {-# inline mkLiteralString #-}
29 end-instance
30
31 derive Text instances (EqP, Eq, Ord)
32
33 instance Semigroup Text where
34   define (<>) (Text s1) (Text s2) := Text (s1 <> s2)
35 end-instance
36
37 instance Monoid Text where
38   define mempty := pack ""
39 end-instance
40
41
42
43 end-module

```

## 1.30 Prelude/Tuple.ad

### Outline

- module Prelude.Tuple ..... lines 1 - 134
  - fst (declaration + definition) ..... lines 26, 27
  - snd (declaration + definition) ..... lines 29, 30
  - curry (declaration + definition) ..... lines 32, 33
  - uncurry (declaration + definition) ..... lines 35, 36

- BuildTupleAbbrev (template) .....	lines 38, 39
- BuildFunctorTupleInstance (template) .....	lines 42 - 54
- generate-code .....	line 56
- BuildTupleInstanceVars (template) .....	lines 65 - 79
- BuildEqPTupleInstance (template) .....	lines 81 - 93
- BuildEqTupleInstance (template) .....	lines 95 - 107
- BuildOrdTupleInstance (template) .....	lines 109 - 122
- generate-code .....	line 124

## Content

```

1 module Prelude.Tuple (
2   fst,
3   snd,
4   uncurry,
5   curry,
6
7   Tuple2(..),
8   Tuple3(..),
9   Tuple4(..),
10  Tuple5(..),
11  Tuple6(..),
12  Tuple7(..),
13  Tuple8(..),
14  Tuple9(..)
15 ) where
16
17 {-# NoImplicitPrelude #-}
18
19 import Prelude.Bool
20 import Prelude.Eq
21 import Prelude.Ord
22 import Prelude.Function
23 import Prelude.Template
24 import Prelude.Functor
25
26 declare fst :: ('a, 'b) -> 'a
27 define fst (x, _) := x
28
29 declare snd :: ('a, 'b) -> 'b
30 define snd (_, y) := y
31
32 declare curry :: (('a, 'b) -> 'c) -> ('a -> 'b -> 'c)
33 define curry f x y := f (x, y)
34
35 declare uncurry :: ('a -> 'b -> 'c) -> (('a, 'b) -> 'c)
36 define uncurry f (x, y) := f x y
37
38 template BuildTupleAbbrev (n :: Nat) :=
39   '''type Tuple{:= n =} \ 'a := {:=!BuildTuple(replicate(n, "'a"))=}''''
40
41
42 template BuildFunctorTupleInstance (n :: Nat) := '''
43 {{ name-scope {
44   var xs := generateNames("x", n)
45   }
46   var f_xs := [] :: [String];

```

```

47   foreach var x in xs {
48     f_xs := insertAtEnd("f " ++ x, f_xs);
49   }
50 }}
51 {{! BuildInstanceHeader([] :: [String], "Functor Tuple" ++ n) !}}
52   define fmap f {{= !BuildTuple(xs) =}} := {{= !BuildTuple(f_xs) =}}
53 end-instance
54 '''
55
56 generate-code '''
57   {{% for var i := 2 to 9 %}}
58   {{! BuildTupleAbbrev(i) !}}
59
60   {{! BuildFunctorTupleInstance(i) !}}
61
62   {{% end-for %}}
63 '''
64
65 template BuildTupleInstanceVars (className :: String, n :: Nat) :: ([String], [String], [String], String)
66 {{ name-scope {
67   var tyVars := !GetTypeVars(n);
68   var xs := generateNames("x", n);
69   var ys := generateNames("y", n);
70
71   var super := [] :: [String];
72   foreach var tyV in tyVars {
73     super := insertAtEnd(className ++ " " ++ tyV, super);
74   }
75
76   var inst := className ++ " " ++ !BuildTuple(tyVars);
77   }
78   return (xs, ys, super, inst);
79 }}
80
81 template BuildEqPTupleInstance (n :: Nat) := '''
82 {{ var (xs, ys, super, inst) := !BuildTupleInstanceVars("EqP", n);
83   var comps := [] :: [String];
84   foreach var xy in zip(xs, ys) {
85     var el := "(" ++ #1(xy) ++ " == " ++ #2(xy) ++ " ";
86     comps := insertAtEnd(el, comps);
87   }
88 }}
89 {{! BuildInstanceHeader(super, inst) !}}
90   define non-exec (==) {{= !BuildTuple(xs) =}} {{= !BuildTuple(ys) =}} :=
91     {{= !Intercalate(" /\n", comps) =}}
92 end-instance
93 '''
94
95 template BuildEqTupleInstance (n :: Nat) := '''
96 {{ var (xs, ys, super, inst) := !BuildTupleInstanceVars("Eq", n);
97   var comps := [] :: [String];
98   foreach var xy in zip(xs, ys) {
99     var el := "(" ++ #1(xy) ++ " == " ++ #2(xy) ++ " ";
100    comps := insertAtEnd(el, comps);
101  }
102 }}
103 {{! BuildInstanceHeader(super, inst) !}}
104   define (==) {{= !BuildTuple(xs) =}} {{= !BuildTuple(ys) =}} :=
105     {{= !Intercalate(" &&\n", comps) =}}

```



```

106 end-instance
107 '''
108
109 template BuildOrdTupleInstance (n :: Nat) := '''
110 {{ var (xs, ys, super, inst) := !BuildTupleInstanceVars("Ord", n);
111     var comps := [] :: [String];
112     for var i := 0 to (n - 2) {
113         var el := "lexCompare " ++ xs[i] ++ " " ++ ys[i];
114         comps := insertAtEnd(el, comps);
115     }
116 }}
117 {{! BuildInstanceHeader(super, inst) !}}
118 define compare {{= !BuildTuple(xs) =}} {{= !BuildTuple(ys) =}} :=
119     {{= !Intercalate(" $\n", comps) =}} $
120     compare {{=xs[n-1]=}} {{=ys[n-1]=}}
121 end-instance
122 '''
123
124 generate-code '''
125     {{% for var i := 2 to 9 %}}
126     {{! BuildEqPTupleInstance(i) !}}
127
128     {{! BuildEqTupleInstance(i) !}}
129
130     {{! BuildOrdTupleInstance(i) !}}
131
132     {{% end-for %}}
133 '''
134 end-module

```

## 1.31 Prelude/Unit.ad

### Outline

- module Prelude.Unit ..... lines 1 - 13
  - Unit (datatype) ..... line 8
  - Itself (datatype) .....line 11

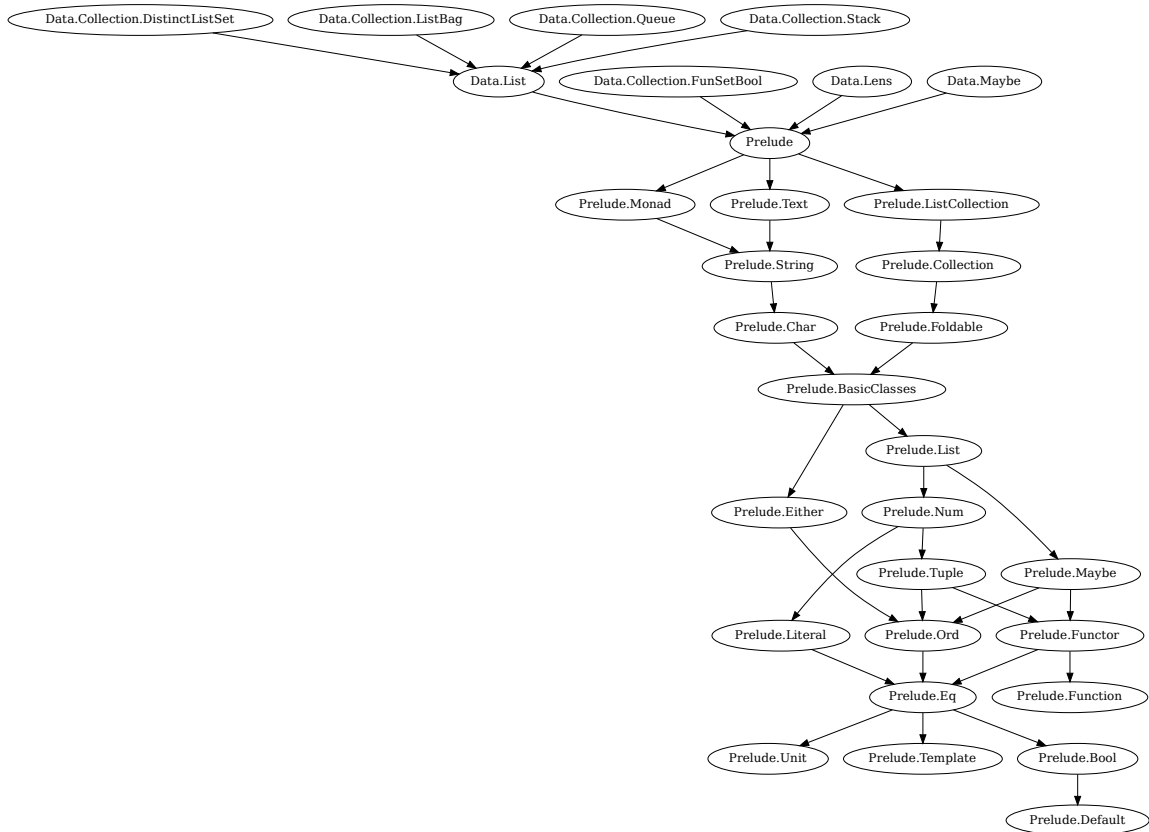
### Content

```

1 module Prelude.Unit (
2     Unit(..),
3     Itself(..)
4 ) where
5
6 {-# NoImplicitPrelude #-}
7
8 datatype Unit := Unit
9
10
11 datatype Itself 'a := Itself
12
13 end-module

```

## 2 Modules



- `Data.Collection.DistinctListSet`
  - `Data/Collection/DistinctListSet.ad` ..... lines 1 - 96
- `Data.Collection.FunSetBool`
  - `Data/Collection/FunSetBool.ad` ..... lines 1 - 52
- `Data.Collection.ListBag`
  - `Data/Collection/ListBag.ad` .....lines 1 - 155
- `Data.Collection.Queue`
  - `Data/Collection/Queue.ad` ..... lines 1 - 91
- `Data.Collection.Stack`
  - `Data/Collection/Stack.ad` ..... lines 1 - 82
- `Data.Lens`
  - `Data/Lens.ad` .....lines 2 - 259
- `Data.List`
  - `Data/List.ad` ..... lines 1 - 82
- `Data.Maybe`
  - `Data/Maybe.ad` ..... lines 1 - 30

- Prelude
  - Prelude.ad ..... lines 1 - 31
- Prelude.BasicClasses
  - Prelude/BasicClasses.ad .....lines 1 - 280
- Prelude.Bool
  - Prelude/Bool.ad ..... lines 1 - 74
- Prelude.Char
  - Prelude/Char.ad ..... lines 1 - 50
- Prelude.Collection
  - Prelude/Collection.ad .....lines 1 - 211
- Prelude.Default
  - Prelude/Default.ad ..... lines 1 - 29
- Prelude.Either
  - Prelude/Either.ad ..... lines 1 - 22
- Prelude.Eq
  - Prelude/Eq.ad .....lines 1 - 148
- Prelude.Foldable
  - Prelude/Foldable.ad .....lines 1 - 172
- Prelude.Function
  - Prelude/Function.ad ..... lines 1 - 35
- Prelude.Functor
  - Prelude/Functor.ad ..... lines 1 - 21
- Prelude.List
  - Prelude/List.ad .....lines 1 - 243
- Prelude.ListCollection
  - Prelude/ListCollection.ad ..... lines 1 - 70
- Prelude.Literal
  - Prelude/Literal.ad .....lines 1 - 138
- Prelude.Maybe
  - Prelude/Maybe.ad ..... lines 1 - 52
- Prelude.Monad
  - Prelude/Monad.ad .....lines 1 - 114
- Prelude.Num
  - Prelude/Num.ad .....lines 1 - 102
- Prelude.Ord
  - Prelude/Ord.ad .....lines 1 - 120
- Prelude.String
  - Prelude/String.ad ..... lines 1 - 21
- Prelude.Template
  - Prelude/Template.ad .....lines 2 - 258

- `Prelude.Text`
  - `Prelude/Text.ad` ..... lines 1 - 43
- `Prelude.Tuple`
  - `Prelude/Tuple.ad` .....lines 1 - 134
- `Prelude.Unit`
  - `Prelude/Unit.ad` ..... lines 1 - 13

# 3 Indices

## 3.1 Classes

### 3.1.1 B

- Bag

- `Prelude.Bag` (alias for `Prelude.Collection.Bag`)
- `Prelude.Collection.Bag` ..... `Prelude/Collection.ad`, lines 205, 206
  - \* arguments: 'c, 'e
  - \* super-classes:
    - `BagLike 'c 'e`
    - `CollectionWithDelete 'c 'e`
    - `CollectionWithEmptynessTest 'c 'e`
  - \* functional-dependencies:
    - 'c -> 'e
  - \* instances:
    - `Eq 'e => Bag (Data.Collection.ListBag.ListBag 'e)`
    - 'e ..... `Data/Collection/ListBag.ad`, lines 149, 150

- BagLike

- `Prelude.BagLike` (alias for `Prelude.Collection.BagLike`)
- `Prelude.Collection.BagLike` ..... `Prelude/Collection.ad`, lines 195 - 197
  - \* arguments: 'c, 'e
  - \* super-classes:
    - `CollectionWithMembershipTest 'c 'e`
  - \* functional-dependencies:
    - 'c -> 'e
  - \* instances:
    - `BagLike (Data.Collection.ListBag.ListBag 'e)`
    - 'e ..... `Data/Collection/ListBag.ad`, lines 146, 147

- Bounded

- `Prelude.Bounded` (alias for `Prelude.BasicClasses.Bounded`)
- `Prelude.BasicClasses.Bounded` ..... `Prelude/BasicClasses.ad`, lines 76 - 86
  - \* arguments: 'a
  - \* super-classes:
    - `Ord 'a`
  - \* functional-dependencies: -
  - \* instances:
    - `Bounded Bool` ..... `Prelude/BasicClasses.ad`, line 278
    - `Bounded Ordering` ..... `Prelude/BasicClasses.ad`, line 277

## 3.1.2 C

## • Collection

- `Prelude.Collection` (alias for `Prelude.Collection.Collection`)
- `Prelude.Collection.Collection` ..... `Prelude/Collection.ad`, lines 45 - 63
  - \* arguments: 'c, 'e
  - \* super-classes:
    - `Monoid 'c`
  - \* functional-dependencies:
    - 'c -> 'e
  - \* instances:
    - `Eq 'e => Collection (Data.Collection.DistinctListSet.DistinctListSet 'e)`  
 'e ..... `Data/Collection/DistinctListSet.ad`, lines 18 - 27
    - `Eq 'e => Collection (Data.Collection.FunSetBool.FunSetBool 'e)`  
 'e ..... `Data/Collection/FunSetBool.ad`, lines 17 - 27
    - `Collection (Data.Collection.ListBag.ListBag 'e)`  
 'e ..... `Data/Collection/ListBag.ad`, lines 30 - 41
    - `Collection (Data.Collection.Queue.Queue 'e)`  
 'e ..... `Data/Collection/Queue.ad`, lines 60 - 69
    - `Collection (Data.Collection.Stack.Stack 'e) 'e` `Data/Collection/Stack.ad`, lines 57 - 66
    - `Collection ['e] 'e` ..... `Prelude/ListCollection.ad`, lines 13 - 24

## • CollectionWithDelete

- `Prelude.CollectionWithDelete` (alias for `Prelude.Collection.CollectionWithDelete`)
- `Prelude.Collection.CollectionWithDelete` ..... `Prelude/Collection.ad`, lines 107 - 140
  - \* arguments: 'c, 'e
  - \* super-classes:
    - `Eq 'e`
    - `CollectionWithMembershipTest 'c 'e`
  - \* functional-dependencies:
    - 'c -> 'e
  - \* instances:
    - `Eq 'e => CollectionWithDelete (Data.Collection.DistinctListSet.DistinctListSet 'e) 'e` ..... `Data/Collection/DistinctListSet.ad`, lines 52 - 70
    - `Eq 'e => CollectionWithDelete (Data.Collection.FunSetBool.FunSetBool 'e)`  
 'e ..... `Data/Collection/FunSetBool.ad`, lines 34 - 47
    - `Eq 'e => CollectionWithDelete (Data.Collection.ListBag.ListBag 'e)`  
 'e ..... `Data/Collection/ListBag.ad`, lines 70 - 90
    - `Eq 'e => CollectionWithDelete ['e] 'e` ..... `Prelude/ListCollection.ad`, lines 50 - 65

## • CollectionWithEmptynessTest

- `Prelude.CollectionWithEmptynessTest` (alias for `Prelude.Collection.CollectionWithEmptynessTest`)
- `Prelude.Collection.CollectionWithEmptynessTest` ..... `Prelude/Collection.ad`, lines 79 - 83
  - \* arguments: 'c, 'e
  - \* super-classes:
    - `Collection 'c 'e`
  - \* functional-dependencies:
    - 'c -> 'e

- \* instances:
  - Eq 'e => CollectionWithEmptynessTest (Data.Collection.DistinctListSet.DistinctListSet 'e)  
'e .....Data/Collection/DistinctListSet.ad, lines 46 - 49
  - CollectionWithEmptynessTest (Data.Collection.ListBag.ListBag 'e)  
'e .....Data/Collection/ListBag.ad, lines 26 - 28
  - CollectionWithEmptynessTest (Data.Collection.Queue.Queue 'e)  
'e .....Data/Collection/Queue.ad, lines 56 - 58
  - CollectionWithEmptynessTest (Data.Collection.Stack.Stack 'e)  
'e .....Data/Collection/Stack.ad, lines 53 - 55
  - CollectionWithEmptynessTest ['e] 'e ..... Prelude/ListCollection.ad, lines 35 - 38
- CollectionWithMembershipTest
  - Prelude.CollectionWithMembershipTest (alias for Prelude.Collection.CollectionWithMembershipTest)
  - Prelude.Collection.CollectionWithMembershipTest ..... Prelude/Collection.ad, lines 88 - 103
    - \* arguments: 'c, 'e
    - \* super-classes:
      - Collection 'c 'e
    - \* functional-dependencies:
      - 'c -> 'e
    - \* instances:
      - Eq 'e => CollectionWithMembershipTest (Data.Collection.DistinctListSet.DistinctListSet 'e)  
'e .....Data/Collection/DistinctListSet.ad, lines 29 - 32
      - Eq 'e => CollectionWithMembershipTest (Data.Collection.FunSetBool.FunSetBool 'e) 'e .....Data/Collection/FunSetBool.ad, lines 29 - 32
      - CollectionWithMembershipTest (Data.Collection.ListBag.ListBag 'e)  
'e .....Data/Collection/ListBag.ad, lines 47 - 54
      - Eq 'e => CollectionWithMembershipTest (Data.Collection.Queue.Queue 'e)  
'e .....Data/Collection/Queue.ad, lines 82 - 86
      - Eq 'e => CollectionWithMembershipTest (Data.Collection.Stack.Stack 'e)  
'e .....Data/Collection/Stack.ad, lines 73 - 77
      - CollectionWithMembershipTest ['e] 'e ..... Prelude/ListCollection.ad, lines 26 - 33
- CollectionWithSubsetTest
  - Prelude.CollectionWithSubsetTest (alias for Prelude.Collection.CollectionWithSubsetTest)
  - Prelude.Collection.CollectionWithSubsetTest ..... Prelude/Collection.ad, lines 162 - 177
    - \* arguments: 'c, 'e
    - \* super-classes:
      - CollectionWithDelete 'c 'e
      - CollectionWithEmptynessTest 'c 'e
    - \* functional-dependencies:
      - 'c -> 'e
    - \* instances:
      - Eq 'e => CollectionWithSubsetTest (Data.Collection.DistinctListSet.DistinctListSet 'e)  
'e .....Data/Collection/DistinctListSet.ad, lines 72 - 82
      - Eq 'e => CollectionWithSubsetTest (Data.Collection.ListBag.ListBag 'e)  
'e .....Data/Collection/ListBag.ad, lines 123 - 135

## 3.1.3 D

## • Default

- `Prelude.Default` (alias for `Prelude.Default.Default`)
- `Prelude.Default.Default` ..... `Prelude/Default.ad`, lines 8 - 16
  - \* arguments: 'a
  - \* super-classes: -
  - \* functional-dependencies: -
  - \* instances:
    - `Default Bool` ..... `Prelude/Bool.ad`, lines 25, 26
    - `Default Char` ..... `Prelude/Char.ad`, lines 24 - 27
    - `Default (Maybe 'a)` ..... `Prelude/Maybe.ad`, lines 24 - 26
    - `Default Natural` ..... `Prelude/Num.ad`, lines 23 - 26
    - `Default Ordering` ..... `Prelude/Ord.ad`, lines 23 - 26
    - `Default ['a]` ..... `Prelude/List.ad`, lines 55 - 57

## • Dot

- `Prelude.Dot` (alias for `Prelude.Function.Dot`)
- `Prelude.Function.Dot` ..... `Prelude/Function.ad`, lines 11 - 13
  - \* arguments: 'a, 'b, 'c
  - \* super-classes: -
  - \* functional-dependencies:
    - 'a, 'b -> 'c
  - \* instances:
    - `Dot ('b -> 'c) ('a -> 'b2) ('a -> 'c) | 'b2 -> 'b` `Prelude/Function.ad`, lines 15 - 18
    - `Dot (Data.Lens.Getter 'a 'b) (Data.Lens.Getter 'b2 'c) (Data.Lens.Getter 'a 'c) | 'b2 -> 'b` ..... `Data/Lens.ad`, lines 120 - 123
    - `Dot (Data.Lens.Getter 'a 'b) (Data.Lens.Lens 'b2 'x 'c 'y) (Data.Lens.Getter 'a 'c) | 'b2 -> 'b` ..... `Data/Lens.ad`, lines 125 - 128
    - `Dot (Data.Lens.Lens 'a 'x 'b 'y) (Data.Lens.Getter 'b2 'c) (Data.Lens.Getter 'a 'c) | 'b2 -> 'b` ..... `Data/Lens.ad`, lines 130 - 133
    - `Dot (Data.Lens.Lens 's1 't1 'a1 'b1) (Data.Lens.Lens 's2 't2 'a2 'b2) (Data.Lens.Lens 's1 't1 'a2 'b2) | 'a1 -> 's2, 'b1 -> 't2` ..... `Data/Lens.ad`, lines 149 - 152
    - `Dot (Data.Lens.Lens 's1 't1 'a1 'b1) (Data.Lens.Setter 's2 't2 'b2) (Data.Lens.Setter 's1 't1 'b2) | 'a1 -> 's2, 'b1 -> 't2` ..... `Data/Lens.ad`, lines 144 - 147

## 3.1.4 E

## • Elem1

- `Data.Lens.Elem1` ..... `Data/Lens.ad`, line 191
  - \* arguments: 's, 't, 'a, 'b
  - \* super-classes: -
  - \* functional-dependencies:
    - 's -> 'a
    - 's, 'b -> 't
  - \* instances:



```

· Data.Lens.Elem1 ('a1, 'a2) 'x 'z 'y | 'x -> ('y, 'a2), 'z ->
  'a1 ..... Data/Lens.ad, line 191
· Data.Lens.Elem1 ('a1, 'a2, 'a3) 'x 'z 'y | 'x -> ('y, 'a2, 'a3), 'z ->
  'a1 ..... Data/Lens.ad, line 191
· Data.Lens.Elem1 ('a1, 'a2, 'a3, 'a4) 'x 'z 'y | 'x -> ('y, 'a2, 'a3, 'a4), 'z
  -> 'a1 ..... Data/Lens.ad, line 191
· Data.Lens.Elem1 ('a1, 'a2, 'a3, 'a4, 'a5) 'x 'z 'y | 'x -> ('y, 'a2, 'a3, 'a4,
  'a5), 'z -> 'a1 ..... Data/Lens.ad, line 191
· Data.Lens.Elem1 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6) 'x 'z 'y | 'x -> ('y, 'a2, 'a3,
  'a4, 'a5, 'a6), 'z -> 'a1 ..... Data/Lens.ad, line 191
· Data.Lens.Elem1 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7) 'x 'z 'y | 'x -> ('y, 'a2,
  'a3, 'a4, 'a5, 'a6, 'a7), 'z -> 'a1 ..... Data/Lens.ad, line 191
· Data.Lens.Elem1 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8) 'x 'z 'y | 'x -> ('y,
  'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8), 'z -> 'a1 ..... Data/Lens.ad, line 191
· Data.Lens.Elem1 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8, 'a9) 'x 'z 'y | 'x ->
  ('y, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8, 'a9), 'z -> 'a1 ... Data/Lens.ad, line 191

```

- Elem2

```

- Data.Lens.Elem2 ..... Data/Lens.ad, line 191
  * arguments: 's, 't, 'a, 'b
  * super-classes: -
  * functional-dependencies:
    · 's -> 'a
    · 's, 'b -> 't
  * instances:
    · Data.Lens.Elem2 ('a1, 'a2) 'x 'z 'y | 'x -> ('a1, 'y), 'z ->
      'a2 ..... Data/Lens.ad, line 191
    · Data.Lens.Elem2 ('a1, 'a2, 'a3) 'x 'z 'y | 'x -> ('a1, 'y, 'a3), 'z ->
      'a2 ..... Data/Lens.ad, line 191
    · Data.Lens.Elem2 ('a1, 'a2, 'a3, 'a4) 'x 'z 'y | 'x -> ('a1, 'y, 'a3, 'a4), 'z
      -> 'a2 ..... Data/Lens.ad, line 191
    · Data.Lens.Elem2 ('a1, 'a2, 'a3, 'a4, 'a5) 'x 'z 'y | 'x -> ('a1, 'y, 'a3, 'a4,
      'a5), 'z -> 'a2 ..... Data/Lens.ad, line 191
    · Data.Lens.Elem2 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6) 'x 'z 'y | 'x -> ('a1, 'y, 'a3,
      'a4, 'a5, 'a6), 'z -> 'a2 ..... Data/Lens.ad, line 191
    · Data.Lens.Elem2 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7) 'x 'z 'y | 'x -> ('a1, 'y,
      'a3, 'a4, 'a5, 'a6, 'a7), 'z -> 'a2 ..... Data/Lens.ad, line 191
    · Data.Lens.Elem2 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8) 'x 'z 'y | 'x -> ('a1,
      'y, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8), 'z -> 'a2 ..... Data/Lens.ad, line 191
    · Data.Lens.Elem2 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8, 'a9) 'x 'z 'y | 'x ->
      ('a1, 'y, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8, 'a9), 'z -> 'a2 ... Data/Lens.ad, line 191

```

- Elem3

```

- Data.Lens.Elem3 ..... Data/Lens.ad, line 191
  * arguments: 's, 't, 'a, 'b
  * super-classes: -
  * functional-dependencies:
    · 's -> 'a
    · 's, 'b -> 't
  * instances:

```

```

· Data.Lens.Elem3 ('a1, 'a2, 'a3) 'x 'z 'y | 'x -> ('a1, 'a2, 'y), 'z ->
  'a3 ..... Data/Lens.ad, line 191
· Data.Lens.Elem3 ('a1, 'a2, 'a3, 'a4) 'x 'z 'y | 'x -> ('a1, 'a2, 'y, 'a4), 'z
  -> 'a3 ..... Data/Lens.ad, line 191
· Data.Lens.Elem3 ('a1, 'a2, 'a3, 'a4, 'a5) 'x 'z 'y | 'x -> ('a1, 'a2, 'y, 'a4,
  'a5), 'z -> 'a3 ..... Data/Lens.ad, line 191
· Data.Lens.Elem3 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6) 'x 'z 'y | 'x -> ('a1, 'a2, 'y,
  'a4, 'a5, 'a6), 'z -> 'a3 ..... Data/Lens.ad, line 191
· Data.Lens.Elem3 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7) 'x 'z 'y | 'x -> ('a1, 'a2,
  'y, 'a4, 'a5, 'a6, 'a7), 'z -> 'a3 ..... Data/Lens.ad, line 191
· Data.Lens.Elem3 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8) 'x 'z 'y | 'x -> ('a1,
  'a2, 'y, 'a4, 'a5, 'a6, 'a7, 'a8), 'z -> 'a3 ..... Data/Lens.ad, line 191
· Data.Lens.Elem3 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8, 'a9) 'x 'z 'y | 'x ->
  ('a1, 'a2, 'y, 'a4, 'a5, 'a6, 'a7, 'a8, 'a9), 'z -> 'a3 ... Data/Lens.ad, line 191

```

- Elem4

```

- Data.Lens.Elem4 ..... Data/Lens.ad, line 191
  * arguments: 's, 't, 'a, 'b
  * super-classes: -
  * functional-dependencies:
    · 's -> 'a
    · 's, 'b -> 't
  * instances:
    · Data.Lens.Elem4 ('a1, 'a2, 'a3, 'a4) 'x 'z 'y | 'x -> ('a1, 'a2, 'a3, 'y), 'z
      -> 'a4 ..... Data/Lens.ad, line 191
    · Data.Lens.Elem4 ('a1, 'a2, 'a3, 'a4, 'a5) 'x 'z 'y | 'x -> ('a1, 'a2, 'a3, 'y,
      'a5), 'z -> 'a4 ..... Data/Lens.ad, line 191
    · Data.Lens.Elem4 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6) 'x 'z 'y | 'x -> ('a1, 'a2, 'a3,
      'y, 'a5, 'a6), 'z -> 'a4 ..... Data/Lens.ad, line 191
    · Data.Lens.Elem4 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7) 'x 'z 'y | 'x -> ('a1, 'a2,
      'a3, 'y, 'a5, 'a6, 'a7), 'z -> 'a4 ..... Data/Lens.ad, line 191
    · Data.Lens.Elem4 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8) 'x 'z 'y | 'x -> ('a1,
      'a2, 'a3, 'y, 'a5, 'a6, 'a7, 'a8), 'z -> 'a4 ..... Data/Lens.ad, line 191
    · Data.Lens.Elem4 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8, 'a9) 'x 'z 'y | 'x ->
      ('a1, 'a2, 'a3, 'y, 'a5, 'a6, 'a7, 'a8, 'a9), 'z -> 'a4 ... Data/Lens.ad, line 191

```

- Elem5

```

- Data.Lens.Elem5 ..... Data/Lens.ad, line 191
  * arguments: 's, 't, 'a, 'b
  * super-classes: -
  * functional-dependencies:
    · 's -> 'a
    · 's, 'b -> 't
  * instances:
    · Data.Lens.Elem5 ('a1, 'a2, 'a3, 'a4, 'a5) 'x 'z 'y | 'x -> ('a1, 'a2, 'a3, 'a4,
      'y), 'z -> 'a5 ..... Data/Lens.ad, line 191
    · Data.Lens.Elem5 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6) 'x 'z 'y | 'x -> ('a1, 'a2, 'a3,
      'a4, 'y, 'a6), 'z -> 'a5 ..... Data/Lens.ad, line 191
    · Data.Lens.Elem5 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7) 'x 'z 'y | 'x -> ('a1, 'a2,
      'a3, 'a4, 'y, 'a6, 'a7), 'z -> 'a5 ..... Data/Lens.ad, line 191

```

```

· Data.Lens.Elem5 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8) 'x 'z 'y | 'x -> ('a1,
  'a2, 'a3, 'a4, 'y, 'a6, 'a7, 'a8), 'z -> 'a5 ..... Data/Lens.ad, line 191
· Data.Lens.Elem5 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8, 'a9) 'x 'z 'y | 'x ->
  ('a1, 'a2, 'a3, 'a4, 'y, 'a6, 'a7, 'a8, 'a9), 'z -> 'a5 ... Data/Lens.ad, line 191

```

- Elem6

```

- Data.Lens.Elem6 ..... Data/Lens.ad, line 191
* arguments: 's, 't, 'a, 'b
* super-classes: -
* functional-dependencies:
  · 's -> 'a
  · 's, 'b -> 't
* instances:
  · Data.Lens.Elem6 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6) 'x 'z 'y | 'x -> ('a1, 'a2, 'a3,
    'a4, 'a5, 'y), 'z -> 'a6 ..... Data/Lens.ad, line 191
  · Data.Lens.Elem6 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7) 'x 'z 'y | 'x -> ('a1, 'a2,
    'a3, 'a4, 'a5, 'y, 'a7), 'z -> 'a6 ..... Data/Lens.ad, line 191
  · Data.Lens.Elem6 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8) 'x 'z 'y | 'x -> ('a1,
    'a2, 'a3, 'a4, 'a5, 'y, 'a7, 'a8), 'z -> 'a6 ..... Data/Lens.ad, line 191
  · Data.Lens.Elem6 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8, 'a9) 'x 'z 'y | 'x ->
    ('a1, 'a2, 'a3, 'a4, 'a5, 'y, 'a7, 'a8, 'a9), 'z -> 'a6 ... Data/Lens.ad, line 191

```

- Elem7

```

- Data.Lens.Elem7 ..... Data/Lens.ad, line 191
* arguments: 's, 't, 'a, 'b
* super-classes: -
* functional-dependencies:
  · 's -> 'a
  · 's, 'b -> 't
* instances:
  · Data.Lens.Elem7 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7) 'x 'z 'y | 'x -> ('a1, 'a2,
    'a3, 'a4, 'a5, 'a6, 'y), 'z -> 'a7 ..... Data/Lens.ad, line 191
  · Data.Lens.Elem7 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8) 'x 'z 'y | 'x -> ('a1,
    'a2, 'a3, 'a4, 'a5, 'a6, 'y, 'a8), 'z -> 'a7 ..... Data/Lens.ad, line 191
  · Data.Lens.Elem7 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8, 'a9) 'x 'z 'y | 'x ->
    ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'y, 'a8, 'a9), 'z -> 'a7 ... Data/Lens.ad, line 191

```

- Elem8

```

- Data.Lens.Elem8 ..... Data/Lens.ad, line 191
* arguments: 's, 't, 'a, 'b
* super-classes: -
* functional-dependencies:
  · 's -> 'a
  · 's, 'b -> 't
* instances:
  · Data.Lens.Elem8 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8) 'x 'z 'y | 'x -> ('a1,
    'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'y), 'z -> 'a8 ..... Data/Lens.ad, line 191
  · Data.Lens.Elem8 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8, 'a9) 'x 'z 'y | 'x ->
    ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'y, 'a9), 'z -> 'a8 ... Data/Lens.ad, line 191

```

## • Elem9

- `Data.Lens.Elem9` ..... `Data/Lens.ad`, line 191
  - \* arguments: 's, 't, 'a, 'b
  - \* super-classes: -
  - \* functional-dependencies:
    - 's -> 'a
    - 's, 'b -> 't
  - \* instances:
    - `Data.Lens.Elem9 ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8, 'a9) 'x 'z 'y | 'x -> ('a1, 'a2, 'a3, 'a4, 'a5, 'a6, 'a7, 'a8, 'y), 'z -> 'a9` ... `Data/Lens.ad`, line 191

## • Enum

- `Prelude.Enum` (alias for `Prelude.BasicClasses.Enum`)
- `Prelude.BasicClasses.Enum` ..... `Prelude/BasicClasses.ad`, lines 25 - 65
  - \* arguments: 'a
  - \* super-classes:
    - `Ord 'a`
  - \* functional-dependencies: -
  - \* instances:
    - `Enum Bool` ..... `Prelude/BasicClasses.ad`, line 278
    - `Enum Char` ..... `Prelude/Char.ad`, lines 43 - 48
    - `Enum Natural` ..... `Prelude/BasicClasses.ad`, lines 67 - 72
    - `Enum Ordering` ..... `Prelude/BasicClasses.ad`, line 277

## • Eq

- `Prelude.Eq` (alias for `Prelude.Eq.Eq`)
- `Prelude.Eq.Eq` ..... `Prelude/Eq.ad`, lines 90 - 104
  - \* arguments: 'a
  - \* super-classes:
    - `EqP 'a`
  - \* functional-dependencies: -
  - \* instances:
    - `(Eq 'a, Eq 'b) => Eq ('a, 'b)` ..... `Prelude/Tuple.ad`, line 124
    - `(Eq 'a, Eq 'b, Eq 'c) => Eq ('a, 'b, 'c)` ..... `Prelude/Tuple.ad`, line 124
    - `(Eq 'a, Eq 'b, Eq 'c, Eq 'd) => Eq ('a, 'b, 'c, 'd)` .... `Prelude/Tuple.ad`, line 124
    - `(Eq 'a, Eq 'b, Eq 'c, Eq 'd, Eq 'e) => Eq ('a, 'b, 'c, 'd, 'e)` ..... `Prelude/Tuple.ad`, line 124
    - `(Eq 'a, Eq 'b, Eq 'c, Eq 'd, Eq 'e, Eq 'f) => Eq ('a, 'b, 'c, 'd, 'e, 'f)` ..... `Prelude/Tuple.ad`, line 124
    - `(Eq 'a, Eq 'b, Eq 'c, Eq 'd, Eq 'e, Eq 'f, Eq 'g) => Eq ('a, 'b, 'c, 'd, 'e, 'f, 'g)` ..... `Prelude/Tuple.ad`, line 124
    - `(Eq 'a, Eq 'b, Eq 'c, Eq 'd, Eq 'e, Eq 'f, Eq 'g, Eq 'h) => Eq ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h)` ..... `Prelude/Tuple.ad`, line 124
    - `(Eq 'a, Eq 'b, Eq 'c, Eq 'd, Eq 'e, Eq 'f, Eq 'g, Eq 'h, Eq 'i) => Eq ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i)` ..... `Prelude/Tuple.ad`, line 124
    - `Eq ()` ..... `Prelude/Eq.ad`, lines 138 - 140
    - `Eq Bool` ..... `Prelude/Eq.ad`, line 146

- Eq BuiltInChar .....Prelude/Literal.ad, lines 65 - 68
- Eq BuiltInDecimal .....Prelude/Literal.ad, lines 116 - 119
- Eq BuiltInNum .....Prelude/Literal.ad, lines 30 - 33
- Eq BuiltInString .....Prelude/Literal.ad, lines 87 - 90
- Eq Char .....Prelude/Char.ad, line 22
- Eq 'e => Eq (Data.Collection.ListBag.ListBag 'e) ..... Data/Collection/ListBag.ad, lines 141 - 144
- Eq 'a => Eq (Data.Collection.Queue.Queue 'a) .Data/Collection/Queue.ad, lines 39 - 41
- Eq 'a => Eq (Data.Collection.Stack.Stack 'a) . Data/Collection/Stack.ad, lines 36 - 38
- (Eq 'a, Eq 'b) => Eq (Either 'a 'b) .....Prelude/Either.ad, line 20
- Eq 'a => Eq (Maybe 'a) ..... Prelude/Maybe.ad, line 50
- Eq Natural .....Prelude/Num.ad, line 28
- Eq Ordering ..... Prelude/Ord.ad, line 21
- Eq Text ..... Prelude/Text.ad, line 31
- Eq 'a => Eq ['a] .....Prelude/List.ad, line 232

- EqP

- Prelude.EqP (alias for Prelude.Eq.EqP)
- Prelude.Eq.EqP ..... Prelude/Eq.ad, lines 20 - 35
  - \* arguments: 'a
  - \* super-classes: -
  - \* functional-dependencies: -
  - \* instances:
    - EqP 'b => EqP ('a -> 'b) ..... Prelude/Eq.ad, lines 84 - 86
    - (EqP 'a, EqP 'b) => EqP ('a, 'b) ..... Prelude/Tuple.ad, line 124
    - (EqP 'a, EqP 'b, EqP 'c) => EqP ('a, 'b, 'c) ..... Prelude/Tuple.ad, line 124
    - (EqP 'a, EqP 'b, EqP 'c, EqP 'd) => EqP ('a, 'b, 'c, 'd) Prelude/Tuple.ad, line 124
    - (EqP 'a, EqP 'b, EqP 'c, EqP 'd, EqP 'e) => EqP ('a, 'b, 'c, 'd, 'e) ..... Prelude/Tuple.ad, line 124
    - (EqP 'a, EqP 'b, EqP 'c, EqP 'd, EqP 'e, EqP 'f) => EqP ('a, 'b, 'c, 'd, 'e, 'f) ..... Prelude/Tuple.ad, line 124
    - (EqP 'a, EqP 'b, EqP 'c, EqP 'd, EqP 'e, EqP 'f, EqP 'g) => EqP ('a, 'b, 'c, 'd, 'e, 'f, 'g) ..... Prelude/Tuple.ad, line 124
    - (EqP 'a, EqP 'b, EqP 'c, EqP 'd, EqP 'e, EqP 'f, EqP 'g, EqP 'h) => EqP ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h) ..... Prelude/Tuple.ad, line 124
    - (EqP 'a, EqP 'b, EqP 'c, EqP 'd, EqP 'e, EqP 'f, EqP 'g, EqP 'h, EqP 'i) => EqP ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i) ..... Prelude/Tuple.ad, line 124
    - EqP () .....Prelude/Eq.ad, lines 142 - 144
    - EqP Bool ..... Prelude/Eq.ad, line 146
    - EqP BuiltInChar .....Prelude/Literal.ad, lines 60 - 63
    - EqP BuiltInDecimal .....Prelude/Literal.ad, lines 111 - 114
    - EqP BuiltInNum .....Prelude/Literal.ad, lines 25 - 28
    - EqP BuiltInString .....Prelude/Literal.ad, lines 82 - 85
    - EqP Char .....Prelude/Char.ad, line 22
    - Eq 'e => EqP (Data.Collection.ListBag.ListBag 'e) ..... Data/Collection/ListBag.ad, lines 137 - 139

- EqP 'a => EqP (Data.Collection.Queue.Queue 'a) .....Data/Collection/Queue.ad, lines 35 - 37
- EqP 'a => EqP (Data.Collection.Stack.Stack 'a) Data/Collection/Stack.ad, lines 32 - 34
- (EqP 'a, EqP 'b) => EqP (Either 'a 'b) .....Prelude/Either.ad, line 20
- EqP 'a => EqP (Maybe 'a) ..... Prelude/Maybe.ad, line 50
- EqP Natural .....Prelude/Num.ad, line 28
- EqP Ordering ..... Prelude/Ord.ad, line 21
- EqP Text ..... Prelude/Text.ad, line 31
- EqP 'a => EqP ['a] .....Prelude/List.ad, line 232

### 3.1.5 F

- Finite

- Prelude.Finite (alias for Prelude.BasicClasses.Finite)
- Prelude.BasicClasses.Finite ..... Prelude/BasicClasses.ad, lines 90 - 109
  - \* arguments: 'a
  - \* super-classes: -
  - \* functional-dependencies: -
  - \* instances:
    - (!Finite 'a, !Finite 'b) => Finite ('a, 'b) ..Prelude/BasicClasses.ad, lines 160 - 164
    - (!Finite 'a, !Finite 'b, !Finite 'c) => Finite ('a, 'b, 'c) ..... Prelude/BasicClasses.ad, line 186
    - (!Finite 'a, !Finite 'b, !Finite 'c, !Finite 'd) => Finite ('a, 'b, 'c, 'd) ..... Prelude/BasicClasses.ad, line 186
    - (!Finite 'a, !Finite 'b, !Finite 'c, !Finite 'd, !Finite 'e) => Finite ('a, 'b, 'c, 'd, 'e) ..... Prelude/BasicClasses.ad, line 186
    - (!Finite 'a, !Finite 'b, !Finite 'c, !Finite 'd, !Finite 'e, !Finite 'f) => Finite ('a, 'b, 'c, 'd, 'e, 'f) ..... Prelude/BasicClasses.ad, line 186
    - (!Finite 'a, !Finite 'b, !Finite 'c, !Finite 'd, !Finite 'e, !Finite 'f, !Finite 'g) => Finite ('a, 'b, 'c, 'd, 'e, 'f, 'g) Prelude/BasicClasses.ad, line 186
    - (!Finite 'a, !Finite 'b, !Finite 'c, !Finite 'd, !Finite 'e, !Finite 'f, !Finite 'g, !Finite 'h) => Finite ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h) ..... Prelude/BasicClasses.ad, line 186
    - (!Finite 'a, !Finite 'b, !Finite 'c, !Finite 'd, !Finite 'e, !Finite 'f, !Finite 'g, !Finite 'h, !Finite 'i) => Finite ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i) ..... Prelude/BasicClasses.ad, line 186
    - (!Finite 'a, !Finite 'b, !Finite 'c, !Finite 'd, !Finite 'e, !Finite 'f, !Finite 'g, !Finite 'h, !Finite 'i, !Finite 'j) => Finite ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j) ..... Prelude/BasicClasses.ad, line 186
    - Finite () .....Prelude/BasicClasses.ad, lines 193 - 197
    - Finite Bool ..... Prelude/BasicClasses.ad, line 278
    - (!Finite 'a, !Finite 'b) => Finite (Either 'a 'b) .....Prelude/BasicClasses.ad, lines 205 - 209
    - !Finite 'a => Finite (Maybe 'a) .....Prelude/BasicClasses.ad, lines 199 - 203
    - Finite Ordering ..... Prelude/BasicClasses.ad, line 277

- FiniteBag

- Prelude.FiniteBag (alias for Prelude.Collection.FiniteBag)
- Prelude.Collection.FiniteBag ..... Prelude/Collection.ad, lines 208, 209

- \* arguments: 'c, 'e
- \* super-classes:
  - Bag 'c 'e
  - FiniteCollection 'c 'e
- \* functional-dependencies:
  - 'c -> 'e
- \* instances:
  - Eq 'e => FiniteBag (Data.Collection.ListBag.ListBag 'e)
  - 'e .....Data/Collection/ListBag.ad, lines 152, 153
- FiniteCollection
  - Prelude.FiniteCollection (alias for Prelude.Collection.FiniteCollection)
  - Prelude.Collection.FiniteCollection ..... Prelude/Collection.ad, lines 144 - 157
  - \* arguments: 'c, 'e
  - \* super-classes:
    - CollectionWithMembershipTest 'c 'e
    - Foldable 'c 'e
    - CollectionWithEmptinessTest 'c 'e
  - \* functional-dependencies:
    - 'c -> 'e
  - \* instances:
    - Eq 'e => FiniteCollection (Data.Collection.DistinctListSet.DistinctListSet 'e)
    - 'e ..... Data/Collection/DistinctListSet.ad, lines 84, 85
    - Eq 'e => FiniteCollection (Data.Collection.ListBag.ListBag 'e)
    - 'e ..... Data/Collection/ListBag.ad, lines 56, 57
    - Eq 'e => FiniteCollection (Data.Collection.Queue.Queue 'e)
    - 'e ..... Data/Collection/Queue.ad, lines 88, 89
    - Eq 'e => FiniteCollection (Data.Collection.Stack.Stack 'e)
    - 'e ..... Data/Collection/Stack.ad, lines 79, 80
    - FiniteCollection ['e] 'e ..... Prelude/ListCollection.ad, lines 67, 68
- FiniteSet
  - Prelude.FiniteSet (alias for Prelude.Collection.FiniteSet)
  - Prelude.Collection.FiniteSet ..... Prelude/Collection.ad, lines 202, 203
  - \* arguments: 'c, 'e
  - \* super-classes:
    - Set 'c 'e
    - FiniteCollection 'c 'e
  - \* functional-dependencies:
    - 'c -> 'e
  - \* instances:
    - Eq 'e => FiniteSet (Data.Collection.DistinctListSet.DistinctListSet 'e)
    - 'e ..... Data/Collection/DistinctListSet.ad, lines 93, 94
- Foldable
  - Prelude.Foldable (alias for Prelude.Foldable.Foldable)
  - Prelude.Foldable.Foldable ..... Prelude/Foldable.ad, lines 18 - 50
  - \* arguments: 't, 'e

- \* super-classes: -
- \* functional-dependencies:
  - 't -> 'e
- \* instances:
  - Foldable ('a, 'a) 'a .....Prelude/Foldable.ad, lines 107 - 114
  - Foldable ('a, 'a, 'a) 'a .....Prelude/Foldable.ad, lines 116 - 123
  - Foldable ('a, 'a, 'a, 'a) 'a .....Prelude/Foldable.ad, lines 125 - 132
  - Foldable ('a, 'a, 'a, 'a, 'a) 'a .....Prelude/Foldable.ad, lines 134 - 141
  - Foldable ('a, 'a, 'a, 'a, 'a, 'a) 'a .....Prelude/Foldable.ad, lines 143 - 150
  - Foldable ('a, 'a, 'a, 'a, 'a, 'a, 'a) 'a .....Prelude/Foldable.ad, lines 152 - 160
  - Foldable ('a, 'a, 'a, 'a, 'a, 'a, 'a, 'a) 'a ....Prelude/Foldable.ad, lines 162 - 169
  - Foldable (Data.Collection.DistinctListSet.DistinctListSet 'e)  
'e .....Data/Collection/DistinctListSet.ad, lines 34 - 43
  - Foldable (Data.Collection.ListBag.ListBag 'e)  
'e .....Data/Collection/ListBag.ad, lines 43 - 45
  - Foldable (Data.Collection.Queue.Queue 'e) 'e .Data/Collection/Queue.ad, lines 71 - 80
  - Foldable (Data.Collection.Stack.Stack 'e) 'e . Data/Collection/Stack.ad, lines 68 - 71
  - Foldable (Maybe 'a) 'a ..... Prelude/Foldable.ad, lines 83 - 104
  - Foldable ['a] 'a ..... Prelude/Foldable.ad, lines 52 - 80
- Functor
  - Prelude.Functor (alias for Prelude.Functor.Functor)
  - Prelude.Functor.Functor .....Prelude/Functor.ad, lines 12 - 17
  - \* arguments: 'F
  - \* super-classes: -
  - \* functional-dependencies: -
  - \* instances:
    - Functor Data.Collection.ListBag.ListBag ....Data/Collection/ListBag.ad, lines 13 - 15
    - Functor Data.Collection.Queue.Queue .....Data/Collection/Queue.ad, lines 31 - 33
    - Functor Data.Collection.Stack.Stack ..... Data/Collection/Stack.ad, lines 28 - 30
    - Functor List .....Prelude/List.ad, lines 64 - 67
    - Functor Maybe .....Prelude/Maybe.ad, lines 32 - 35
    - Functor Tuple2 ..... Prelude/Tuple.ad, line 56
    - Functor Tuple3 ..... Prelude/Tuple.ad, line 56
    - Functor Tuple4 ..... Prelude/Tuple.ad, line 56
    - Functor Tuple5 ..... Prelude/Tuple.ad, line 56
    - Functor Tuple6 ..... Prelude/Tuple.ad, line 56
    - Functor Tuple7 ..... Prelude/Tuple.ad, line 56
    - Functor Tuple8 ..... Prelude/Tuple.ad, line 56
    - Functor Tuple9 ..... Prelude/Tuple.ad, line 56



**3.1.6 I**

- **IsGetter**
  - `Data.Lens.IsGetter` ..... `Data/Lens.ad`, lines 68 - 74
    - \* arguments: 'v, 'a, 'b
    - \* super-classes: -
    - \* functional-dependencies:
      - 'v -> 'a, 'b
    - \* instances:
      - `Data.Lens.IsGetter (Data.Lens.Getter 'a 'b) 'a 'b` ..... `Data/Lens.ad`, lines 87 - 90
      - `Data.Lens.IsGetter (Data.Lens.Lens 's 't 'a 'b) 's 'a` .. `Data/Lens.ad`, lines 92 - 95
- **IsSetter**
  - `Data.Lens.IsSetter` ..... `Data/Lens.ad`, lines 79 - 85
    - \* arguments: 'v, 's, 't, 'b
    - \* super-classes: -
    - \* functional-dependencies:
      - 'v -> 's, 't, 'b
    - \* instances:
      - `Data.Lens.IsSetter (Data.Lens.Lens 's 't 'a 'b) 's 't 'b` ..... `Data/Lens.ad`, lines 107 - 110
      - `Data.Lens.IsSetter (Data.Lens.Setter 's 't 'b) 's 't 'b` ..... `Data/Lens.ad`, lines 102 - 105

**3.1.7 L**

- **LiteralChar**
  - `Prelude.LiteralChar` (alias for `Prelude.Literal.LiteralChar`)
  - `Prelude.Literal.LiteralChar` ..... `Prelude/Literal.ad`, lines 55 - 57
    - \* arguments: 'a
    - \* super-classes:
      - `Eq 'a`
    - \* functional-dependencies: -
    - \* instances:
      - `LiteralChar BuiltInChar` ..... `Prelude/Literal.ad`, lines 70 - 73
      - `LiteralChar Char` ..... `Prelude/Char.ad`, lines 37 - 40
- **LiteralDecimal**
  - `Prelude.LiteralDecimal` (alias for `Prelude.Literal.LiteralDecimal`)
  - `Prelude.Literal.LiteralDecimal` ..... `Prelude/Literal.ad`, lines 122 - 124
    - \* arguments: 'a
    - \* super-classes:
      - `LiteralNum 'a`
    - \* functional-dependencies: -
    - \* instances:
      - `LiteralDecimal BuiltInDecimal` ..... `Prelude/Literal.ad`, lines 132 - 135
- **LiteralNum**
  - `Prelude.LiteralNum` (alias for `Prelude.Literal.LiteralNum`)

- `Prelude.Literal.LiteralNum` ..... Prelude/Literal.ad, lines 36 - 38
  - \* arguments: 'a
  - \* super-classes:
    - `Eq 'a`
  - \* functional-dependencies: -
  - \* instances:
    - `LiteralNum Bool` ..... Prelude/Num.ad, lines 94 - 99
    - `LiteralNum BuiltInDecimal` ..... Prelude/Literal.ad, lines 127 - 130
    - `LiteralNum BuiltInNum` ..... Prelude/Literal.ad, lines 41 - 44
    - `LiteralNum Natural` ..... Prelude/Num.ad, lines 57 - 59
- **LiteralString**
  - `Prelude.LiteralString` (alias for `Prelude.Literal.LiteralString`)
  - `Prelude.Literal.LiteralString` ..... Prelude/Literal.ad, lines 93 - 95
    - \* arguments: 'a
    - \* super-classes:
      - `Eq 'a`
    - \* functional-dependencies: -
    - \* instances:
      - `LiteralString BuiltInString` ..... Prelude/Literal.ad, lines 98 - 101
      - `LiteralString String` ..... Prelude/String.ad, lines 16 - 19
      - `LiteralString Text` ..... Prelude/Text.ad, lines 26 - 29

### 3.1.8 M

- **Monad**
  - `Prelude.Monad` (alias for `Prelude.Monad.Monad`)
  - `Prelude.Monad.Monad` ..... Prelude/Monad.ad, lines 28 - 39
    - \* arguments: 'M
    - \* super-classes: -
    - \* functional-dependencies: -
    - \* instances:
      - `Monad List` ..... Prelude/Monad.ad, lines 84 - 90
      - `Monad Maybe` ..... Prelude/Monad.ad, lines 92 - 98
- **MonadFail**
  - `Prelude.MonadFail` (alias for `Prelude.Monad.MonadFail`)
  - `Prelude.Monad.MonadFail` ..... Prelude/Monad.ad, lines 103 - 107
    - \* arguments: 'M
    - \* super-classes:
      - `!Monad 'M`
    - \* functional-dependencies: -
    - \* instances:
      - `MonadFail Maybe` ..... Prelude/Monad.ad, lines 109 - 112
- **Monoid**
  - `Prelude.Monoid` (alias for `Prelude.BasicClasses.Monoid`)
  - `Prelude.BasicClasses.Monoid` ..... Prelude/BasicClasses.ad, lines 117 - 122

```

* arguments: 'a
* super-classes:
  · Semigroup 'a
* functional-dependencies: -
* instances:
  · Monoid () .....Prelude/BasicClasses.ad, lines 134 - 137
  · Eq 'e => Monoid (Data.Collection.DistinctListSet.DistinctListSet
    'e) .....Data/Collection/DistinctListSet.ad, lines 14 - 16
  · Eq 'e => Monoid (Data.Collection.FunSetBool.FunSetBool
    'e) .....Data/Collection/FunSetBool.ad, lines 13 - 15
  · Monoid (Data.Collection.ListBag.ListBag 'a) .Data/Collection/ListBag.ad, lines 21 - 24
  · Monoid (Data.Collection.Queue.Queue 'a) .....Data/Collection/Queue.ad, lines 51 - 54
  · Monoid (Data.Collection.Stack.Stack 'a) .....Data/Collection/Stack.ad, lines 48 - 51
  · Monoid Ordering .....Prelude/BasicClasses.ad, lines 155 - 157
  · Monoid Text .....Prelude/Text.ad, lines 37 - 39
  · Monoid ['a] .....Prelude/BasicClasses.ad, lines 144 - 147

```

### 3.1.9 N

- NumDiv

```

- Prelude.NumDiv (alias for Prelude.Num.NumDiv)
- Prelude.Num.NumDiv .....Prelude/Num.ad, lines 42 - 44
  * arguments: 'a
  * super-classes: -
  * functional-dependencies: -
  * instances:
    · NumDiv Natural .....Prelude/Num.ad, lines 88 - 91

```

- NumDivMod

```

- Prelude.NumDivMod (alias for Prelude.Num.NumDivMod)
- Prelude.Num.NumDivMod .....Prelude/Num.ad, lines 46 - 54
  * arguments: 'a
  * super-classes: -
  * functional-dependencies: -
  * instances:
    · NumDivMod Natural .....Prelude/Num.ad, lines 77 - 86

```

- NumMinus

```

- Prelude.NumMinus (alias for Prelude.Num.NumMinus)
- Prelude.Num.NumMinus .....Prelude/Num.ad, lines 34 - 36
  * arguments: 'a
  * super-classes: -
  * functional-dependencies: -
  * instances:
    · NumMinus Natural .....Prelude/Num.ad, lines 66 - 70

```

- NumMult

```

- Prelude.NumMult (alias for Prelude.Num.NumMult)

```

- `Prelude.Num.NumMult` .....Prelude/Num.ad, lines 38 - 40
  - \* arguments: 'a
  - \* super-classes: -
  - \* functional-dependencies: -
  - \* instances:
    - `NumMult Natural` .....Prelude/Num.ad, lines 72 - 75
- **NumPlus**
  - `Prelude.NumPlus` (alias for `Prelude.Num.NumPlus`)
  - `Prelude.Num.NumPlus` .....Prelude/Num.ad, lines 30 - 32
    - \* arguments: 'a
    - \* super-classes: -
    - \* functional-dependencies: -
    - \* instances:
      - `NumPlus Natural` .....Prelude/Num.ad, lines 61 - 64

### 3.1.10 O

- **Ord**
  - `Prelude.Ord` (alias for `Prelude.Ord.Ord`)
  - `Prelude.Ord.Ord` ..... Prelude/Ord.ad, lines 29 - 63
    - \* arguments: 'a
    - \* super-classes:
      - `Eq 'a`
    - \* functional-dependencies: -
    - \* instances:
      - `(Ord 'a, Ord 'b) => Ord ('a, 'b)` ..... Prelude/Tuple.ad, line 124
      - `(Ord 'a, Ord 'b, Ord 'c) => Ord ('a, 'b, 'c)` ..... Prelude/Tuple.ad, line 124
      - `(Ord 'a, Ord 'b, Ord 'c, Ord 'd) => Ord ('a, 'b, 'c, 'd)` Prelude/Tuple.ad, line 124
      - `(Ord 'a, Ord 'b, Ord 'c, Ord 'd, Ord 'e) => Ord ('a, 'b, 'c, 'd, 'e)` ..... Prelude/Tuple.ad, line 124
      - `(Ord 'a, Ord 'b, Ord 'c, Ord 'd, Ord 'e, Ord 'f) => Ord ('a, 'b, 'c, 'd, 'e, 'f)` ..... Prelude/Tuple.ad, line 124
      - `(Ord 'a, Ord 'b, Ord 'c, Ord 'd, Ord 'e, Ord 'f, Ord 'g) => Ord ('a, 'b, 'c, 'd, 'e, 'f, 'g)` ..... Prelude/Tuple.ad, line 124
      - `(Ord 'a, Ord 'b, Ord 'c, Ord 'd, Ord 'e, Ord 'f, Ord 'g, Ord 'h) => Ord ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h)` ..... Prelude/Tuple.ad, line 124
      - `(Ord 'a, Ord 'b, Ord 'c, Ord 'd, Ord 'e, Ord 'f, Ord 'g, Ord 'h, Ord 'i) => Ord ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i)` ..... Prelude/Tuple.ad, line 124
      - `Ord ()` ..... Prelude/Ord.ad, lines 65 - 67
      - `Ord Bool` .....Prelude/Ord.ad, line 118
      - `Ord Char` .....Prelude/Char.ad, line 22
      - `Ord 'a => Ord (Data.Collection.Queue.Queue 'a)` .....Data/Collection/Queue.ad, lines 43 - 45
      - `Ord 'a => Ord (Data.Collection.Stack.Stack 'a)` Data/Collection/Stack.ad, lines 40 - 42
      - `(Ord 'a, Ord 'b) => Ord (Either 'a 'b)` .....Prelude/Either.ad, line 20
      - `Ord 'a => Ord (Maybe 'a)` ..... Prelude/Maybe.ad, line 50
      - `Ord Natural` .....Prelude/Num.ad, line 28

- Ord Ordering .....Prelude/Ord.ad, line 117
- Ord Text ..... Prelude/Text.ad, line 31
- Ord 'a => Ord ['a] .....Prelude/List.ad, line 232

### 3.1.11 S

- Semigroup

- Prelude.Semigroup (alias for Prelude.BasicClasses.Semigroup)
- Prelude.BasicClasses.Semigroup .....Prelude/BasicClasses.ad, lines 112 - 114
  - \* arguments: 'a
  - \* super-classes: -
  - \* functional-dependencies: -
  - \* instances:
    - Semigroup () .....Prelude/BasicClasses.ad, lines 129 - 132
    - Eq 'e => Semigroup (Data.Collection.DistinctListSet.DistinctListSet 'e) ..... Data/Collection/DistinctListSet.ad, lines 9 - 12
    - Eq 'e => Semigroup (Data.Collection.FunSetBool.FunSetBool 'e) .....Data/Collection/FunSetBool.ad, lines 8 - 11
    - Semigroup (Data.Collection.ListBag.ListBag 'a) .....Data/Collection/ListBag.ad, lines 17 - 19
    - Semigroup (Data.Collection.Queue.Queue 'a) ..Data/Collection/Queue.ad, lines 47 - 49
    - Semigroup (Data.Collection.Stack.Stack 'a) .. Data/Collection/Stack.ad, lines 44 - 46
    - Semigroup Ordering .....Prelude/BasicClasses.ad, lines 149 - 153
    - Semigroup Text ..... Prelude/Text.ad, lines 33 - 35
    - Semigroup ['a] .....Prelude/BasicClasses.ad, lines 139 - 142

- Set

- Prelude.Set (alias for Prelude.Collection.Set)
- Prelude.Collection.Set ..... Prelude/Collection.ad, lines 199, 200
  - \* arguments: 'c, 'e
  - \* super-classes:
    - SetLike 'c 'e
    - CollectionWithDelete 'c 'e
    - CollectionWithEmptynessTest 'c 'e
  - \* functional-dependencies:
    - 'c -> 'e
  - \* instances:
    - Eq 'e => Set (Data.Collection.DistinctListSet.DistinctListSet 'e) 'e ..... Data/Collection/DistinctListSet.ad, lines 90, 91

- SetLike

- Prelude.SetLike (alias for Prelude.Collection.SetLike)
- Prelude.Collection.SetLike ..... Prelude/Collection.ad, lines 179 - 181
  - \* arguments: 'c, 'e
  - \* super-classes:
    - CollectionWithMembershipTest 'c 'e
  - \* functional-dependencies:
    - 'c -> 'e

```

* instances:
  · Eq 'e => SetLike (Data.Collection.DistinctListSet.DistinctListSet 'e)
    'e ..... Data/Collection/DistinctListSet.ad, lines 87, 88
  · Eq 'e => SetLike (Data.Collection.FunSetBool.FunSetBool 'e)
    'e ..... Data/Collection/FunSetBool.ad, lines 49, 50

```

## 3.2 Constants

### 3.2.1 (

- (\$)
  - Prelude.\$ (alias for Prelude.Function.\$)
  - Prelude.Function.\$ ..... Prelude/Function.ad, line 31
    - \* constant
    - \* type ('a -> 'b) -> 'a -> 'b
    - \* executable
- (&&)
  - Prelude.&& (alias for Prelude.Bool.&&)
  - Prelude.Bool.&& ..... Prelude/Bool.ad, line 35
    - \* constant
    - \* type Bool -> Bool -> Bool
    - \* executable
- (\*)
  - Prelude.\* (alias for Prelude.Num.\*)
  - Prelude.Num.\* ..... Prelude/Num.ad, line 39
    - \* constant of type-class NumMult
    - \* type 'a -> 'a -> 'a
    - \* executable
- (+)
  - Prelude.+ (alias for Prelude.Num.+)
  - Prelude.Num.+ ..... Prelude/Num.ad, line 31
    - \* constant of type-class NumPlus
    - \* type 'a -> 'a -> 'a
    - \* executable
- (++)
  - Data.List.++ (alias for Prelude.List.++)
  - Prelude.++ (alias for Prelude.List.++)
  - Prelude.List.++ ..... Prelude/List.ad, line 70
    - \* constant
    - \* type ['a] -> ['a] -> ['a]
    - \* executable
- (-)
  - Prelude.- (alias for Prelude.Num.-)
  - Prelude.Num.- ..... Prelude/Num.ad, line 35
    - \* constant of type-class NumMinus

- \* type 'a -> 'a -> 'a
- \* executable
- (`-->`)
  - `Prelude.-->` (alias for `Prelude.Bool.-->`)
  - `Prelude.Bool.-->` ..... `Prelude/Bool.ad`, line 43
    - \* constant
    - \* type `Bool -> Bool -> Bool`
    - \* executable
- (`.`)
  - `Prelude.(.)` (alias for `Prelude.Function.(.)`)
  - `Prelude.Function.(.)` ..... `Prelude/Function.ad`, line 12
    - \* constant of type-class `Dot`
    - \* type 'a -> 'b -> 'c
    - \* executable
- (`/`)
  - `Prelude./` (alias for `Prelude.Num./`)
  - `Prelude.Num./` ..... `Prelude/Num.ad`, line 43
    - \* constant of type-class `NumDiv`
    - \* type 'a -> 'a -> 'a
    - \* executable
- (`/=`)
  - `Prelude./=` (alias for `Prelude.Eq./=`)
  - `Prelude.Eq./=` ..... `Prelude/Eq.ad`, line 94
    - \* constant of type-class `Eq`
    - \* type 'a -> 'a -> `Bool`
    - \* executable
- (`/\`)
  - `Prelude./\` (alias for `Prelude.Bool./\`)
  - `Prelude.Bool./\` ..... `Prelude/Bool.ad`, line 63
    - \* constant
    - \* type `Prop -> Prop -> Prop`
    - \* non executable
- (`:`)
  - `Data.List.(:)` (alias for `Prelude.List.(:)`)
  - `Prelude.(:)` (alias for `Prelude.List.(:)`)
  - `Prelude.List.(:)` ..... `Prelude/List.ad`, line 50
    - \* constructor of datatype `List`
    - \* type 'a -> ['a] -> ['a]
    - \* executable
- (`<$>`)
  - `Prelude.<$>` (alias for `Prelude.Functor.fmap`)
  - `Prelude.Functor.<$>` (alias for `Prelude.Functor.fmap`)
- (`<`)

- `Prelude.<` (alias for `Prelude.Ord.<`)
- `Prelude.Ord.<` ..... `Prelude/Ord.ad`, line 39
  - \* constant of type-class `Ord`
  - \* type `'a -> 'a -> Bool`
  - \* executable
- `<-/->`
  - `Prelude.<-/->` (alias for `Prelude.Bool.<-/->`)
  - `Prelude.Bool.<-/->` ..... `Prelude/Bool.ad`, line 50
    - \* constant
    - \* type `Bool -> Bool -> Bool`
    - \* executable
- `<->`
  - `Prelude.<->` (alias for `Prelude.Bool.<->`)
  - `Prelude.Bool.<->` ..... `Prelude/Bool.ad`, line 46
    - \* constant
    - \* type `Bool -> Bool -> Bool`
    - \* executable
- `<=`
  - `Prelude.<=` (alias for `Prelude.Ord.<=`)
  - `Prelude.Ord.<=` ..... `Prelude/Ord.ad`, line 40
    - \* constant of type-class `Ord`
    - \* type `'a -> 'a -> Bool`
    - \* executable
- `<=/=`
  - `Prelude.<=/=` (alias for `Prelude.Bool.<=/=`)
  - `Prelude.Bool.<=/=` ..... `Prelude/Bool.ad`, line 67
    - \* constant
    - \* type `Prop -> Prop -> Prop`
    - \* non executable
- `<=>`
  - `Prelude.<=>` (alias for `Prelude.Bool.<=>`)
  - `Prelude.Bool.<=>` ..... `Prelude/Bool.ad`, line 66
    - \* constant
    - \* type `Prop -> Prop -> Prop`
    - \* non executable
- `<>`
  - `Prelude.<>` (alias for `Prelude.BasicClasses.<>`)
  - `Prelude.BasicClasses.<>` ..... `Prelude/BasicClasses.ad`, line 113
    - \* constant of type-class `Semigroup`
    - \* type `'a -> 'a -> 'a`
    - \* executable
- `(=)`
  - `Prelude.(=)` (alias for `Prelude.Eq.(=)`)



- **Prelude.Eq.(=/=)** ..... Prelude/Eq.ad, line 24
  - \* constant of type-class **EqP**
  - \* type **'a -> 'a -> Prop**
  - \* non executable
- **(=<<)**
  - **Prelude.(=<<)** (alias for **Prelude.Monad.(=<<)**)
  - **Prelude.Monad.(=<<)** ..... Prelude/Monad.ad, line 44
    - \* constant
    - \* type **('a -> 'M 'b) -> 'M 'a -> 'M 'b**
    - \* executable
- **(==)**
  - **Prelude.(==)** (alias for **Prelude.Eq.(==)**)
  - **Prelude.Eq.(==)** ..... Prelude/Eq.ad, line 91
    - \* constant of type-class **Eq**
    - \* type **'a -> 'a -> Bool**
    - \* executable
- **(===)**
  - **Prelude.(===)** (alias for **Prelude.Eq.(===)**)
  - **Prelude.Eq.(===)** ..... Prelude/Eq.ad, line 21
    - \* constant of type-class **EqP**
    - \* type **'a -> 'a -> Prop**
    - \* non executable
- **(====)**
  - **Prelude.(====)** (alias for **Prelude.Eq.(====)**)
  - **Prelude.Eq.(====)** ..... Prelude/Eq.ad, line 14
    - \* constant
    - \* type **'a -> 'a -> Prop**
    - \* non executable
- **(==>)**
  - **Prelude.(==>)** (alias for **Prelude.Bool.(==>)**)
  - **Prelude.Bool.(==>)** ..... Prelude/Bool.ad, line 65
    - \* constant
    - \* type **Prop -> Prop -> Prop**
    - \* non executable
- **(>)**
  - **Prelude.>)** (alias for **Prelude.Ord.>)**)
  - **Prelude.Ord.>)** ..... Prelude/Ord.ad, line 41
    - \* constant of type-class **Ord**
    - \* type **'a -> 'a -> Bool**
    - \* executable
- **(>=)**
  - **Prelude.>=)** (alias for **Prelude.Ord.>=)**)
  - **Prelude.Ord.>=)** ..... Prelude/Ord.ad, line 42

- \* constant of type-class `Ord`
- \* type `'a -> 'a -> Bool`
- \* executable
- `(>>)`
  - `Prelude.>>` (alias for `Prelude.Monad.>>`)
  - `Prelude.Monad.>>` ..... `Prelude/Monad.ad`, line 30
    - \* statically resolved constant of type-class `Monad`
    - \* type `'M 'a -> 'M 'b -> 'M 'b`
    - \* executable
- `(>>=)`
  - `Prelude.>>=` (alias for `Prelude.Monad.>>=`)
  - `Prelude.Monad.>>=` ..... `Prelude/Monad.ad`, line 29
    - \* statically resolved constant of type-class `Monad`
    - \* type `'M 'a -> ('a -> 'M 'b) -> 'M 'b`
    - \* executable
- `(\)`
  - `Prelude.\` (alias for `Prelude.Bool.\`)
  - `Prelude.Bool.\` ..... `Prelude/Bool.ad`, line 64
    - \* constant
    - \* type `Prop -> Prop -> Prop`
    - \* non executable
- `(^.)`
  - `Data.Lens.^.` (alias for `Data.Lens.view`)
- `(||)`
  - `Prelude.||` (alias for `Prelude.Bool.||`)
  - `Prelude.Bool.||` ..... `Prelude/Bool.ad`, line 39
    - \* constant
    - \* type `Bool -> Bool -> Bool`
    - \* executable

### 3.2.2 A

- `all`
  - `Data.List.all` (alias for `Prelude.List.all`)
  - `Prelude.all` (alias for `Prelude.Foldable.all`)
  - `Prelude.Foldable.all` ..... `Prelude/Foldable.ad`, line 32
    - \* constant of type-class `Foldable`
    - \* type `('e -> Bool) -> 't -> Bool`
    - \* executable
  - `Prelude.List.all` ..... `Prelude/List.ad`, line 197
    - \* constant
    - \* type `('a -> Bool) -> ['a] -> Bool`
    - \* executable
- `allP`

- `Data.List.allP` (alias for `Prelude.List.allP`)
- `Prelude.allP` (alias for `Prelude.List.allP`)
- `Prelude.List.allP` .....Prelude/List.ad, line 205
  - \* constant
  - \* type (`'a -> Prop`) -> [`'a`] -> `Prop`
  - \* non executable
- **and**
  - `Data.List.and` (alias for `Prelude.List.and`)
  - `Prelude.and` (alias for `Prelude.List.and`)
  - `Prelude.List.and` .....Prelude/List.ad, line 179
    - \* constant
    - \* type [`Bool`] -> `Bool`
    - \* executable
- **andP**
  - `Data.List.andP` (alias for `Prelude.List.andP`)
  - `Prelude.andP` (alias for `Prelude.List.andP`)
  - `Prelude.List.andP` .....Prelude/List.ad, line 184
    - \* constant
    - \* type [`Prop`] -> `Prop`
    - \* non executable
- **any**
  - `Data.List.any` (alias for `Prelude.List.any`)
  - `Prelude.any` (alias for `Prelude.Foldable.any`)
  - `Prelude.Foldable.any` .....Prelude/Foldable.ad, line 33
    - \* constant of type-class `Foldable`
    - \* type (`'e -> Bool`) -> `'t -> Bool`
    - \* executable
  - `Prelude.List.any` .....Prelude/List.ad, line 201
    - \* constant
    - \* type (`'a -> Bool`) -> [`'a`] -> `Bool`
    - \* executable
- **anyP**
  - `Data.List.anyP` (alias for `Prelude.List.anyP`)
  - `Prelude.anyP` (alias for `Prelude.List.anyP`)
  - `Prelude.List.anyP` .....Prelude/List.ad, line 209
    - \* constant
    - \* type (`'a -> Prop`) -> [`'a`] -> `Prop`
    - \* non executable

## 3.2.3 B

- **bind**
  - `Data.Maybe.bind` (alias for `Prelude.Maybe.bind`)
  - `Prelude.bind` (alias for `Prelude.Monad.>=>`)
  - `Prelude.Maybe.bind` ..... `Prelude/Maybe.ad`, line 37
    - \* constant
    - \* type `Maybe 'a -> ('a -> Maybe 'b) -> Maybe 'b`
    - \* executable
  - `Prelude.Monad.bind` (alias for `Prelude.Monad.>=>`)
- **bool2prop**
  - `Prelude.bool2prop` (alias for `Prelude.Bool.bool2prop`)
  - `Prelude.Bool.bool2prop` ..... `Prelude/Bool.ad`, line 58
    - \* constant
    - \* type `Bool -> Prop`
    - \* non executable
- **boolCase**
  - `Prelude.boolCase` (alias for `Prelude.Bool.boolCase`)
  - `Prelude.Bool.boolCase` ..... `Prelude/Bool.ad`, line 28
    - \* constant
    - \* type `Bool -> 'a -> 'a -> 'a`
    - \* executable
- **break**
  - `Data.List.break` ..... `Data/List.ad`, line 62
    - \* constant
    - \* type `('a -> Bool) -> ['a] -> (['a], ['a])`
    - \* executable

## 3.2.4 C

- **caseBool**
  - `Prelude.caseBool` (alias for `Prelude.Bool.caseBool`)
  - `Prelude.Bool.caseBool` ..... `Prelude/Bool.ad`, line 22
    - \* datatype case-constant
    - \* type `'a -> 'a -> Bool -> 'a`
    - \* executable
- **caseDistinctListSet**
  - - `Data.Collection.DistinctListSet.caseDistinctListSet` `Data/Collection/DistinctListSet.ad`, line 7
    - \* datatype case-constant
    - \* type `(['a] -> 'b) -> Data.Collection.DistinctListSet.DistinctListSet 'a -> 'b`
    - \* executable
- **caseEither**
  - `Prelude.caseEither` (alias for `Prelude.Either.caseEither`)
  - `Prelude.Either.caseEither` ..... `Prelude/Either.ad`, line 11

- \* datatype case-constant
- \* type ('a -> 'c) -> ('b -> 'c) -> Either 'a 'b -> 'c
- \* executable
- **caseFunSetBool**
  - `Data.Collection.FunSetBool.caseFunSetBool` .....`Data/Collection/FunSetBool.ad`, line 6
  - \* datatype case-constant
  - \* type (('a -> Bool) -> 'b) -> `Data.Collection.FunSetBool.FunSetBool` 'a -> 'b
  - \* executable
- **caseItself**
  - `Prelude.caseItself` (alias for `Prelude.Unit.caseItself`)
  - `Prelude.Unit.caseItself` .....`Prelude/Unit.ad`, line 11
  - \* datatype case-constant
  - \* type 'b -> (::'a) -> 'b
  - \* executable
- **caseList**
  - `Data.List.caseList` (alias for `Prelude.List.caseList`)
  - `Prelude.caseList` (alias for `Prelude.List.caseList`)
  - `Prelude.List.caseList` .....`Prelude/List.ad`, line 50
  - \* datatype case-constant
  - \* type 'b -> ('a -> ['a] -> 'b) -> ['a] -> 'b
  - \* executable
- **caseMaybe**
  - `Data.Maybe.caseMaybe` (alias for `Prelude.Maybe.caseMaybe`)
  - `Prelude.caseMaybe` (alias for `Prelude.Maybe.caseMaybe`)
  - `Prelude.Maybe.caseMaybe` .....`Prelude/Maybe.ad`, line 20
  - \* datatype case-constant
  - \* type 'b -> ('a -> 'b) -> `Maybe` 'a -> 'b
  - \* executable
- **caseNatural**
  - `Prelude.caseNatural` (alias for `Prelude.Num.caseNatural`)
  - `Prelude.Num.caseNatural` .....`Prelude/Num.ad`, line 19
  - \* datatype case-constant
  - \* type 'a -> (`Natural` -> 'a) -> `Natural` -> 'a
  - \* executable
- **caseOrdering**
  - `Prelude.caseOrdering` (alias for `Prelude.Ord.caseOrdering`)
  - `Prelude.Ord.caseOrdering` .....`Prelude/Ord.ad`, line 16
  - \* datatype case-constant
  - \* type 'a -> 'a -> 'a -> `Ordering` -> 'a
  - \* executable
- **caseUnit**
  - `Prelude.caseUnit` (alias for `Prelude.Unit.caseUnit`)
  - `Prelude.Unit.caseUnit` .....`Prelude/Unit.ad`, line 8

- \* datatype case-constant
- \* type 'a -> () -> 'a
- \* executable
- **catMaybes**
  - `Data.Maybe.catMaybes` ..... `Data/Maybe.ad`, line 17
  - \* constant
  - \* type `[(Maybe 'a)] -> ['a]`
  - \* executable
- **chr**
  - `Prelude.chr` (alias for `Prelude.Char.chr`)
  - `Prelude.Char.chr` ..... `Prelude/Char.ad`, line 29
  - \* constant
  - \* type `Natural -> Char`
  - \* executable
- **compare**
  - `Prelude.compare` (alias for `Prelude.Ord.compare`)
  - `Prelude.Ord.compare` ..... `Prelude/Ord.ad`, line 30
  - \* constant of type-class `Ord`
  - \* type `'a -> 'a -> Ordering`
  - \* executable
- **comparing**
  - `Prelude.comparing` (alias for `Prelude.Ord.comparing`)
  - `Prelude.Ord.comparing` ..... `Prelude/Ord.ad`, line 69
  - \* constant
  - \* type `('b -> 'a) -> 'b -> 'b -> Ordering`
  - \* executable
- **concat**
  - `Data.List.concat` (alias for `Prelude.List.concat`)
  - `Prelude.concat` (alias for `Prelude.List.concat`)
  - `Prelude.List.concat` ..... `Prelude/List.ad`, line 129
  - \* constant
  - \* type `[['a]] -> ['a]`
  - \* executable
- **concatMap**
  - `Data.List.concatMap` (alias for `Prelude.List.concatMap`)
  - `Prelude.concatMap` (alias for `Prelude.List.concatMap`)
  - `Prelude.List.concatMap` ..... `Prelude/List.ad`, line 132
  - \* constant
  - \* type `('a -> ['b]) -> ['a] -> ['b]`
  - \* executable
- **Cons**
  - `Data.List.Cons` (alias for `Prelude.List.(:)`)
  - `Prelude.Cons` (alias for `Prelude.List.(:)`)

- `Prelude.List.Cons` (alias for `Prelude.List.(:)`)
- **constant**
  - `Prelude.constant` (alias for `Prelude.Function.constant`)
  - `Prelude.Function.constant` ..... `Prelude/Function.ad`, line 24
    - \* constant
    - \* type 'a -> 'b -> 'a
    - \* executable
- **count**
  - `Data.List.count` (alias for `Prelude.List.count`)
  - `Prelude.count` (alias for `Prelude.List.count`)
  - `Prelude.List.count` ..... `Prelude/List.ad`, line 168
    - \* constant
    - \* type 'a -> ['a] -> Natural
    - \* executable
- **curry**
  - `Prelude.curry` (alias for `Prelude.Tuple.curry`)
  - `Prelude.Tuple.curry` ..... `Prelude/Tuple.ad`, line 32
    - \* constant
    - \* type (('a, 'b) -> 'c) -> 'a -> 'b -> 'c
    - \* executable

### 3.2.5 D

- **default**
  - `Prelude.default` (alias for `Prelude.Default.default`)
  - `Prelude.Default.default` ..... `Prelude/Default.ad`, line 9
    - \* constant of type-class `Default`
    - \* type 'a
    - \* executable
- **delete**
  - `Prelude.delete` (alias for `Prelude.Collection.delete`)
  - `Prelude.Collection.delete` ..... `Prelude/Collection.ad`, line 109
    - \* constant of type-class `CollectionWithDelete`
    - \* type 'e -> 'c -> 'c
    - \* executable
- **dequeue**
  - `Data.Collection.Queue.dequeue` ..... `Data/Collection/Queue.ad`, line 20
    - \* constant
    - \* type `Data.Collection.Queue.Queue 'a -> Maybe ('a, Data.Collection.Queue.Queue 'a)`
    - \* executable
- **destEither**
  - `Prelude.destEither` (alias for `Prelude.Either.destEither`)
  - `Prelude.Either.destEither` ..... `Prelude/Either.ad`, line 16
    - \* constant

- \* type ('a -> 'c) -> ('b -> 'c) -> Either 'a 'b -> 'c
- \* executable
- **difference**
  - `Prelude.difference` (alias for `Prelude.Collection.difference`)
  - `Prelude.Collection.difference` ..... `Prelude/Collection.ad`, line 115
    - \* constant of type-class `CollectionWithDelete`
    - \* type 'c -> 'c -> 'c
    - \* executable
- **disjoint**
  - `Prelude.disjoint` (alias for `Prelude.Collection.disjoint`)
  - `Prelude.Collection.disjoint` ..... `Prelude/Collection.ad`, line 163
    - \* constant of type-class `CollectionWithSubsetTest`
    - \* type 'c -> 'c -> Bool
    - \* executable
- **DistinctListSet**
  - `Data.Collection.DistinctListSet.DistinctListSet` .. `Data/Collection/DistinctListSet.ad`, line 7
    - \* constructor of datatype `Data.Collection.DistinctListSet.DistinctListSet`
    - \* type ['a] -> `Data.Collection.DistinctListSet.DistinctListSet` 'a
    - \* executable
- **div**
  - `Prelude.div` (alias for `Prelude.Num.div`)
  - `Prelude.Num.div` ..... `Prelude/Num.ad`, line 48
    - \* constant of type-class `NumDivMod`
    - \* type 'a -> 'a -> 'a
    - \* executable
- **divmod**
  - `Prelude.divmod` (alias for `Prelude.Num.divmod`)
  - `Prelude.Num.divmod` ..... `Prelude/Num.ad`, line 47
    - \* constant of type-class `NumDivMod`
    - \* type 'a -> 'a -> ('a, 'a)
    - \* executable
- **drop**
  - `Data.List.drop` ..... `Data/List.ad`, line 25
    - \* constant
    - \* type `Natural` -> ['a] -> ['a]
    - \* executable
- **dropWhile**
  - `Data.List.dropWhile` ..... `Data/List.ad`, line 48
    - \* constant
    - \* type ('a -> Bool) -> ['a] -> ['a]
    - \* executable



## 3.2.6 E

- `elem`
  - `Data.List.elem` (alias for `Prelude.List.elem`)
  - `Prelude.elem` (alias for `Prelude.Foldable.elem`)
  - `Prelude.Foldable.elem` ..... `Prelude/Foldable.ad`, line 31
    - \* constant of type-class `Foldable`
    - \* type `'e -> 't -> Bool`
    - \* executable
  - `Prelude.List.elem` ..... `Prelude/List.ad`, line 214
    - \* constant
    - \* type `'a -> ['a] -> Bool`
    - \* executable
- `elem1`
  - `Data.Lens.elem1` ..... `Data/Lens.ad`, line 191
    - \* constant of type-class `Data.Lens.Elem1`
    - \* type `Data.Lens.Lens 's 't 'a 'b`
    - \* executable
- `elem2`
  - `Data.Lens.elem2` ..... `Data/Lens.ad`, line 191
    - \* constant of type-class `Data.Lens.Elem2`
    - \* type `Data.Lens.Lens 's 't 'a 'b`
    - \* executable
- `elem3`
  - `Data.Lens.elem3` ..... `Data/Lens.ad`, line 191
    - \* constant of type-class `Data.Lens.Elem3`
    - \* type `Data.Lens.Lens 's 't 'a 'b`
    - \* executable
- `elem4`
  - `Data.Lens.elem4` ..... `Data/Lens.ad`, line 191
    - \* constant of type-class `Data.Lens.Elem4`
    - \* type `Data.Lens.Lens 's 't 'a 'b`
    - \* executable
- `elem5`
  - `Data.Lens.elem5` ..... `Data/Lens.ad`, line 191
    - \* constant of type-class `Data.Lens.Elem5`
    - \* type `Data.Lens.Lens 's 't 'a 'b`
    - \* executable
- `elem6`
  - `Data.Lens.elem6` ..... `Data/Lens.ad`, line 191
    - \* constant of type-class `Data.Lens.Elem6`
    - \* type `Data.Lens.Lens 's 't 'a 'b`
    - \* executable
- `elem7`

- `Data.Lens.elem7` ..... `Data/Lens.ad`, line 191
  - \* constant of type-class `Data.Lens.Elem7`
  - \* type `Data.Lens.Lens 's 't 'a 'b`
  - \* executable
- `elem8`
  - `Data.Lens.elem8` ..... `Data/Lens.ad`, line 191
    - \* constant of type-class `Data.Lens.Elem8`
    - \* type `Data.Lens.Lens 's 't 'a 'b`
    - \* executable
- `elem9`
  - `Data.Lens.elem9` ..... `Data/Lens.ad`, line 191
    - \* constant of type-class `Data.Lens.Elem9`
    - \* type `Data.Lens.Lens 's 't 'a 'b`
    - \* executable
- `elemP`
  - `Data.List.elemP` (alias for `Prelude.List.elemP`)
  - `Prelude.List.elemP` ..... `Prelude/List.ad`, line 223
    - \* constant
    - \* type `'a -> ['a] -> Prop`
    - \* non executable
- `empty`
  - `Prelude.empty` (alias for `Prelude.Collection.empty`)
  - `Prelude.Collection.empty` ..... `Prelude/Collection.ad`, line 65
    - \* constant
    - \* type `'c`
    - \* executable
- `emptyQueue`
  - `Data.Collection.Queue.emptyQueue` ..... `Data/Collection/Queue.ad`, line 12
    - \* constant
    - \* type `Data.Collection.Queue.Queue 'a`
    - \* executable
- `emptyStack`
  - `Data.Collection.Stack.emptyStack` ..... `Data/Collection/Stack.ad`, line 12
    - \* constant
    - \* type `Data.Collection.Stack.Stack 'a`
    - \* executable
- `enqueue`
  - `Data.Collection.Queue.enqueue` ..... `Data/Collection/Queue.ad`, line 17
    - \* constant
    - \* type `'a -> Data.Collection.Queue.Queue 'a -> Data.Collection.Queue.Queue 'a`
    - \* executable
- `enumFromTo`
  - `Prelude.enumFromTo` (alias for `Prelude.BasicClasses.enumFromTo`)

- `Prelude.BasicClasses.enumFromTo` ..... Prelude/BasicClasses.ad, line 35
  - \* constant of type-class `Enum`
  - \* type `'a -> 'a -> ['a]`
  - \* executable
- **EQ**
  - `Prelude.EQ` (alias for `Prelude.Ord.EQ`)
  - `Prelude.Ord.EQ` ..... Prelude/Ord.ad, line 18
    - \* constructor of datatype `Ordering`
    - \* type `Ordering`
    - \* executable
- **existsB\_fun**
  - `Prelude.existsB_fun` (alias for `Prelude.BasicClasses.existsB_fun`)
  - `Prelude.BasicClasses.existsB_fun` ..... Prelude/BasicClasses.ad, line 97
    - \* constant of type-class `Finite`
    - \* type `('a -> Bool) -> Bool`
    - \* executable
- **existsP\_fun**
  - `Prelude.existsP_fun` (alias for `Prelude.Bool.existsP_fun`)
  - `Prelude.Bool.existsP_fun` ..... Prelude/Bool.ad, line 71
    - \* constant
    - \* type `('a -> Prop) -> Prop`
    - \* non executable

### 3.2.7 F

- **fail**
  - `Prelude.fail` (alias for `Prelude.Monad.fail`)
  - `Prelude.Monad.fail` ..... Prelude/Monad.ad, line 104
    - \* statically resolved constant of type-class `MonadFail`
    - \* type `String -> 'M 'a`
    - \* executable
- **false**
  - `Prelude.false` (alias for `Prelude.Bool.false`)
  - `Prelude.Bool.false` ..... Prelude/Bool.ad, line 56
    - \* constant
    - \* type `Prop`
    - \* non executable
- **False**
  - `Prelude.False` (alias for `Prelude.Bool.False`)
  - `Prelude.Bool.False` ..... Prelude/Bool.ad, line 22
    - \* constructor of datatype `Bool`
    - \* type `Bool`
    - \* executable
- **filter**

- `Data.List.filter` (alias for `Prelude.List.filter`)
- `Prelude.filter` (alias for `Prelude.Collection.filter`)
- `Prelude.Collection.filter` ..... `Prelude/Collection.ad`, line 112
  - \* constant of type-class `CollectionWithDelete`
  - \* type `('e -> Bool) -> 'c -> 'c`
  - \* executable
- `Prelude.List.filter` ..... `Prelude/List.ad`, line 100
  - \* constant
  - \* type `('a -> Bool) -> ['a] -> ['a]`
  - \* executable
- **flip**
  - `Prelude.flip` (alias for `Prelude.Function.flip`)
  - `Prelude.Function.flip` ..... `Prelude/Function.ad`, line 28
    - \* constant
    - \* type `('a -> 'b -> 'c) -> 'b -> 'a -> 'c`
    - \* executable
- **fmap**
  - `Prelude.fmap` (alias for `Prelude.Functor.fmap`)
  - `Prelude.Functor.fmap` ..... `Prelude/Functor.ad`, line 13
    - \* statically resolved constant of type-class `Functor`
    - \* type `('a -> 'b) -> 'F 'a -> 'F 'b`
    - \* executable
- **foldl**
  - `Data.List.foldl` (alias for `Prelude.List.foldl`)
  - `Prelude.foldl` (alias for `Prelude.Foldable.foldl`)
  - `Prelude.Foldable.foldl` ..... `Prelude/Foldable.ad`, line 25
    - \* statically resolved constant of type-class `Foldable`
    - \* type `('acc -> 'e -> 'acc) -> 'acc -> 't -> 'acc`
    - \* executable
  - `Prelude.List.foldl` ..... `Prelude/List.ad`, line 139
    - \* constant
    - \* type `('a -> 'b -> 'a) -> 'a -> ['b] -> 'a`
    - \* executable
- **foldlWithAbort**
  - `Data.List.foldlWithAbort` (alias for `Prelude.List.foldlWithAbort`)
  - `Prelude.foldlWithAbort` (alias for `Prelude.Foldable.foldlWithAbort`)
  - `Prelude.Foldable.foldlWithAbort` ..... `Prelude/Foldable.ad`, line 28
    - \* statically resolved constant of type-class `Foldable`
    - \* type `('acc -> Bool) -> ('acc -> 'e -> 'acc) -> 'acc -> 't -> 'acc`
    - \* executable
  - `Prelude.List.foldlWithAbort` ..... `Prelude/List.ad`, line 143
    - \* constant
    - \* type `('a -> Bool) -> ('a -> 'b -> 'a) -> 'a -> ['b] -> 'a`
    - \* executable

- **foldr**
  - `Data.List.foldr` (alias for `Prelude.List.foldr`)
  - `Prelude.foldr` (alias for `Prelude.Foldable.foldr`)
  - `Prelude.Foldable.foldr` ..... `Prelude/Foldable.ad`, line 22
    - \* statically resolved constant of type-class `Foldable`
    - \* type `('e -> 'acc -> 'acc) -> 'acc -> 't -> 'acc`
    - \* executable
  - `Prelude.List.foldr` ..... `Prelude/List.ad`, line 153
    - \* constant
    - \* type `('a -> 'b -> 'b) -> 'b -> ['a] -> 'b`
    - \* executable
- **forallB\_fun**
  - `Prelude.forallB_fun` (alias for `Prelude.BasicClasses.forallB_fun`)
  - `Prelude.BasicClasses.forallB_fun` ..... `Prelude/BasicClasses.ad`, line 96
    - \* constant of type-class `Finite`
    - \* type `('a -> Bool) -> Bool`
    - \* executable
- **forallP\_fun**
  - `Prelude.forallP_fun` (alias for `Prelude.Bool.forallP_fun`)
  - `Prelude.Bool.forallP_fun` ..... `Prelude/Bool.ad`, line 70
    - \* constant
    - \* type `('a -> Prop) -> Prop`
    - \* non executable
- **fromEnum**
  - `Prelude.fromEnum` (alias for `Prelude.BasicClasses.fromEnum`)
  - `Prelude.BasicClasses.fromEnum` ..... `Prelude/BasicClasses.ad`, line 29
    - \* constant of type-class `Enum`
    - \* type `'a -> Natural`
    - \* executable
- **fromList**
  - `Prelude.fromList` (alias for `Prelude.Collection.fromList`)
  - `Prelude.Collection.fromList` ..... `Prelude/Collection.ad`, line 49
    - \* constant of type-class `Collection`
    - \* type `['e] -> 'c`
    - \* executable
- **fromMaybe**
  - `Data.Maybe.fromMaybe` (alias for `Prelude.Maybe.fromMaybe`)
  - `Prelude.fromMaybe` (alias for `Prelude.Maybe.fromMaybe`)
  - `Prelude.Maybe.fromMaybe` ..... `Prelude/Maybe.ad`, line 41
    - \* constant
    - \* type `'a -> Maybe 'a -> 'a`
    - \* executable
- **fromMaybeMap**

- `Data.Maybe.fromMaybeMap` (alias for `Prelude.Maybe.fromMaybeMap`)
- `Prelude.fromMaybeMap` (alias for `Prelude.Maybe.fromMaybeMap`)
- `Prelude.Maybe.fromMaybeMap` ..... `Prelude/Maybe.ad`, line 45
  - \* constant
  - \* type `'b -> ('a -> 'b) -> Maybe 'a -> 'b`
  - \* executable
- **fst**
  - `Prelude.fst` (alias for `Prelude.Tuple.fst`)
  - `Prelude.Tuple.fst` ..... `Prelude/Tuple.ad`, line 26
    - \* constant
    - \* type `('a, 'b) -> 'a`
    - \* executable
- **FunSetBool**
  - `Data.Collection.FunSetBool.FunSetBool` ..... `Data/Collection/FunSetBool.ad`, line 6
    - \* constructor of datatype `Data.Collection.FunSetBool.FunSetBool`
    - \* type `('a -> Bool) -> Data.Collection.FunSetBool.FunSetBool 'a`
    - \* executable

### 3.2.8 G

- **Getter**
  - `Data.Lens.Getter` ..... `Data/Lens.ad`, line 50
    - \* constructor of record-type `Data.Lens.Getter`
    - \* type `('a -> 'b) -> Data.Lens.Getter 'a 'b`
    - \* executable
- **getterGet**
  - `Data.Lens.getterGet` ..... `Data/Lens.ad`, line 51
    - \* field of record-type `Data.Lens.Getter`
    - \* type `Data.Lens.Getter 'a 'b -> 'a -> 'b`
    - \* executable
- **GT**
  - `Prelude.GT` (alias for `Prelude.Ord.GT`)
  - `Prelude.Ord.GT` ..... `Prelude/Ord.ad`, line 19
    - \* constructor of datatype `Ordering`
    - \* type `Ordering`
    - \* executable
- **guard**
  - `Prelude.guard` (alias for `Prelude.Monad.guard`)
  - `Prelude.Monad.guard` ..... `Prelude/Monad.ad`, line 48
    - \* constant
    - \* type `Bool -> 'M ()`
    - \* executable

## 3.2.9 I

- **id**
  - `Prelude.id` (alias for `Prelude.Function.id`)
  - `Prelude.Function.id` ..... `Prelude/Function.ad`, line 20
    - \* constant
    - \* type 'a -> 'a
    - \* executable
- **image**
  - `Prelude.image` (alias for `Prelude.Collection.image`)
  - `Prelude.Collection.image` ..... `Prelude/Collection.ad`, line 74
    - \* constant
    - \* type ('e1 -> 'e2) -> 'C 'e1 -> 'C 'e2
    - \* executable
- **insert**
  - `Prelude.insert` (alias for `Prelude.Collection.insert`)
  - `Prelude.Collection.insert` ..... `Prelude/Collection.ad`, line 46
    - \* constant of type-class `Collection`
    - \* type 'e -> 'c -> 'c
    - \* executable
- **intersection**
  - `Prelude.intersection` (alias for `Prelude.Collection.intersection`)
  - `Prelude.Collection.intersection` ..... `Prelude/Collection.ad`, line 116
    - \* constant of type-class `CollectionWithDelete`
    - \* type 'c -> 'c -> 'c
    - \* executable
- **isEmpty**
  - `Prelude.isEmpty` (alias for `Prelude.Collection.isEmpty`)
  - `Prelude.Collection.isEmpty` ..... `Prelude/Collection.ad`, line 80
    - \* constant of type-class `CollectionWithEmptynessTest`
    - \* type 'c -> `Bool`
    - \* executable
- **isEmptyP**
  - `Prelude.isEmptyP` (alias for `Prelude.Collection.isEmptyP`)
  - `Prelude.Collection.isEmptyP` ..... `Prelude/Collection.ad`, line 57
    - \* constant of type-class `Collection`
    - \* type 'c -> `Prop`
    - \* non executable
- **isJust**
  - `Data.Maybe.isJust` (alias for `Prelude.Maybe.isJust`)
  - `Prelude.isJust` (alias for `Prelude.Maybe.isJust`)
  - `Prelude.Maybe.isJust` ..... `Prelude/Maybe.ad`, line 48
    - \* constant
    - \* type `Maybe 'a` -> `Bool`

- \* executable
- **isNothing**
  - `Data.Maybe.isNothing` (alias for `Prelude.Maybe.isNothing`)
  - `Prelude.isNothing` (alias for `Prelude.Maybe.isNothing`)
  - `Prelude.Maybe.isNothing` ..... `Prelude/Maybe.ad`, line 48
    - \* constant
    - \* type `Maybe 'a -> Bool`
    - \* executable
- **isProperSubsetOf**
  - `Prelude.isProperSubsetOf` (alias for `Prelude.Collection.isProperSubsetOf`)
  - `Prelude.Collection.isProperSubsetOf` ..... `Prelude/Collection.ad`, line 166
    - \* constant of type-class `CollectionWithSubsetTest`
    - \* type `'c -> 'c -> Bool`
    - \* executable
- **isSetEquiv**
  - `Prelude.isSetEquiv` (alias for `Prelude.Collection.isSetEquiv`)
  - `Prelude.Collection.isSetEquiv` ..... `Prelude/Collection.ad`, line 168
    - \* constant of type-class `CollectionWithSubsetTest`
    - \* type `'c -> 'c -> Bool`
    - \* executable
- **isSubsetOf**
  - `Prelude.isSubsetOf` (alias for `Prelude.Collection.isSubsetOf`)
  - `Prelude.Collection.isSubsetOf` ..... `Prelude/Collection.ad`, line 165
    - \* constant of type-class `CollectionWithSubsetTest`
    - \* type `'c -> 'c -> Bool`
    - \* executable
- **Itself**
  - `Prelude.Itself` (alias for `Prelude.Unit.Itself`)
  - `Prelude.Unit.Itself` ..... `Prelude/Unit.ad`, line 11
    - \* constructor of datatype `Itself`
    - \* type `(:: 'a)`
    - \* executable

### 3.2.10 J

- **Just**
  - `Data.Maybe.Just` (alias for `Prelude.Maybe.Just`)
  - `Prelude.Just` (alias for `Prelude.Maybe.Just`)
  - `Prelude.Maybe.Just` ..... `Prelude/Maybe.ad`, line 22
    - \* constructor of datatype `Maybe`
    - \* type `'a -> Maybe 'a`
    - \* executable



## 3.2.11 L

- **last**
  - `Data.List.last` (alias for `Prelude.List.last`)
  - `Prelude.last` (alias for `Prelude.List.last`)
  - `Prelude.List.last` ..... `Prelude/List.ad`, line 78
    - \* constant
    - \* type `'a` `-> Maybe 'a`
    - \* executable
- **Left**
  - `Prelude.Left` (alias for `Prelude.Either.Left`)
  - `Prelude.Either.Left` ..... `Prelude/Either.ad`, line 12
    - \* constructor of datatype `Either`
    - \* type `'a -> Either 'a 'b`
    - \* executable
- **length**
  - `Data.List.length` (alias for `Prelude.List.length`)
  - `Prelude.length` (alias for `Prelude.List.length`)
  - `Prelude.List.length` ..... `Prelude/List.ad`, line 165
    - \* constant
    - \* type `'a` `-> Natural`
    - \* executable
- **Lens**
  - `Data.Lens.Lens` ..... `Data/Lens.ad`, line 41
    - \* constructor of record-type `Data.Lens.Lens`
    - \* type `('s -> 'a) -> ('s -> 'b -> 't) -> Data.Lens.Lens 's 't 'a 'b`
    - \* executable
- **lensGet**
  - `Data.Lens.lensGet` ..... `Data/Lens.ad`, line 42
    - \* field of record-type `Data.Lens.Lens`
    - \* type `Data.Lens.Lens 's 't 'a 'b -> 's -> 'a`
    - \* executable
- **lensSet**
  - `Data.Lens.lensSet` ..... `Data/Lens.ad`, line 43
    - \* field of record-type `Data.Lens.Lens`
    - \* type `Data.Lens.Lens 's 't 'a 'b -> 's -> 'b -> 't`
    - \* executable
- **lexCompare**
  - `Prelude.lexCompare` (alias for `Prelude.Ord.lexCompare`)
  - `Prelude.Ord.lexCompare` ..... `Prelude/Ord.ad`, line 72
    - \* constant
    - \* type `'a -> 'a -> Ordering -> Ordering`
    - \* executable
- **listToMaybe**

- `Data.Maybe.listToMaybe` ..... `Data/Maybe.ad`, line 9
  - \* constant
  - \* type `[ 'a ] -> Maybe 'a`
  - \* executable
- **lookup**
  - `Data.List.lookup` ..... `Data/List.ad`, line 66
    - \* constant
    - \* type `'k -> [( 'k, 'v )] -> Maybe 'v`
    - \* executable
- **LT**
  - `Prelude.LT` (alias for `Prelude.Ord.LT`)
  - `Prelude.Ord.LT` ..... `Prelude/Ord.ad`, line 17
    - \* constructor of datatype `Ordering`
    - \* type `Ordering`
    - \* executable

### 3.2.12 M

- **map**
  - `Data.List.map` (alias for `Prelude.List.map`)
  - `Data.Maybe.map` (alias for `Prelude.Maybe.map`)
  - `Prelude.map` (alias for `Prelude.List.map`)
  - `Prelude.List.map` ..... `Prelude/List.ad`, line 60
    - \* constant
    - \* type `('a -> 'b) -> [ 'a ] -> [ 'b ]`
    - \* executable
  - `Prelude.Maybe.map` ..... `Prelude/Maybe.ad`, line 28
    - \* constant
    - \* type `('a -> 'b) -> Maybe 'a -> Maybe 'b`
    - \* executable
- **mapM**
  - `Prelude.mapM` (alias for `Prelude.Monad.mapM`)
  - `Prelude.Monad.mapM` ..... `Prelude/Monad.ad`, line 75
    - \* constant
    - \* type `('a -> 'M 'b) -> [ 'a ] -> 'M [ 'b ]`
    - \* executable
- **mapMaybe**
  - `Data.Maybe.mapMaybe` ..... `Data/Maybe.ad`, line 22
    - \* constant
    - \* type `('a -> Maybe 'b) -> [ 'a ] -> [ 'b ]`
    - \* executable
- **mapM\_**
  - `Prelude.mapM_` (alias for `Prelude.Monad.mapM_`)
  - `Prelude.Monad.mapM_` ..... `Prelude/Monad.ad`, line 79

- \* constant
- \* type ('a -> 'M 'b) -> ['a] -> 'M ()
- \* executable
- **mappend**
  - `Prelude.mappend` (alias for `Prelude.BasicClasses.mappend`)
  - `Prelude.BasicClasses.mappend` ..... `Prelude/BasicClasses.ad`, line 124
    - \* constant
    - \* type 'a -> 'a -> 'a
    - \* executable
- **max**
  - `Prelude.max` (alias for `Prelude.Ord.max`)
  - `Prelude.Ord.max` ..... `Prelude/Ord.ad`, line 44
    - \* constant of type-class `Ord`
    - \* type 'a -> 'a -> 'a
    - \* executable
- **maxBound**
  - `Prelude.maxBound` (alias for `Prelude.BasicClasses.maxBound`)
  - `Prelude.BasicClasses.maxBound` ..... `Prelude/BasicClasses.ad`, line 78
    - \* constant of type-class `Bounded`
    - \* type 'a
    - \* executable
- **maybeToList**
  - `Data.Maybe.maybeToList` ..... `Data/Maybe.ad`, line 13
    - \* constant
    - \* type `Maybe 'a -> ['a]`
    - \* executable
- **member**
  - `Prelude.member` (alias for `Prelude.Collection.member`)
  - `Prelude.Collection.member` ..... `Prelude/Collection.ad`, line 91
    - \* constant of type-class `CollectionWithMembershipTest`
    - \* type 'e -> 'c -> `Bool`
    - \* executable
- **memberP**
  - `Prelude.memberP` (alias for `Prelude.Collection.memberP`)
  - `Prelude.Collection.memberP` ..... `Prelude/Collection.ad`, line 55
    - \* constant of type-class `Collection`
    - \* type 'e -> 'c -> `Prop`
    - \* non executable
- **mempty**
  - `Prelude.mempty` (alias for `Prelude.BasicClasses.mempty`)
  - `Prelude.BasicClasses.mempty` ..... `Prelude/BasicClasses.ad`, line 118
    - \* constant of type-class `Monoid`
    - \* type 'a

- \* executable
- **min**
  - `Prelude.min` (alias for `Prelude.Ord.min`)
  - `Prelude.Ord.min` ..... `Prelude/Ord.ad`, line 43
    - \* constant of type-class `Ord`
    - \* type `'a -> 'a -> 'a`
    - \* executable
- **minBound**
  - `Prelude.minBound` (alias for `Prelude.BasicClasses.minBound`)
  - `Prelude.BasicClasses.minBound` ..... `Prelude/BasicClasses.ad`, line 77
    - \* constant of type-class `Bounded`
    - \* type `'a`
    - \* executable
- **mkChar**
  - `Prelude.mkChar` (alias for `Prelude.Char.mkChar`)
  - `Prelude.Char.mkChar` ..... `Prelude/Char.ad`, line 35
    - \* constant
    - \* type `BuiltInChar -> Char`
    - \* executable
- **mkLiteralChar**
  - `Prelude.mkLiteralChar` (alias for `Prelude.Literal.mkLiteralChar`)
  - `Prelude.Literal.mkLiteralChar` ..... `Prelude/Literal.ad`, line 56
    - \* statically resolved constant of type-class `LiteralChar`
    - \* type `BuiltInChar -> 'a`
    - \* executable
- **mkLiteralDecimal**
  - `Prelude.mkLiteralDecimal` (alias for `Prelude.Literal.mkLiteralDecimal`)
  - `Prelude.Literal.mkLiteralDecimal` ..... `Prelude/Literal.ad`, line 123
    - \* statically resolved constant of type-class `LiteralDecimal`
    - \* type `BuiltInDecimal -> 'a`
    - \* executable
- **mkLiteralNum**
  - `Prelude.mkLiteralNum` (alias for `Prelude.Literal.mkLiteralNum`)
  - `Prelude.Literal.mkLiteralNum` ..... `Prelude/Literal.ad`, line 37
    - \* statically resolved constant of type-class `LiteralNum`
    - \* type `(Prelude.Literal.BuiltInNumRep, BuiltInNum) -> 'a`
    - \* executable
- **mkLiteralString**
  - `Prelude.mkLiteralString` (alias for `Prelude.Literal.mkLiteralString`)
  - `Prelude.Literal.mkLiteralString` ..... `Prelude/Literal.ad`, line 94
    - \* statically resolved constant of type-class `LiteralString`
    - \* type `BuiltInString -> 'a`
    - \* executable

- **mkString**
  - `Prelude.mkString` (alias for `Prelude.String.mkString`)
  - `Prelude.String.mkString` ..... `Prelude/String.ad`, line 14
    - \* constant
    - \* type `BuiltInString -> String`
    - \* executable
- **mod**
  - `Prelude.mod` (alias for `Prelude.Num.mod`)
  - `Prelude.Num.mod` ..... `Prelude/Num.ad`, line 49
    - \* constant of type-class `NumDivMod`
    - \* type `'a -> 'a -> 'a`
    - \* executable

### 3.2.13 N

- **Nil**
  - `Data.List.Nil` (alias for `Prelude.List.Nil`)
  - `Prelude.Nil` (alias for `Prelude.List.Nil`)
  - `Prelude.List.Nil` ..... `Prelude/List.ad`, line 50
    - \* constructor of datatype `List`
    - \* type `['a]`
    - \* executable
- **not**
  - `Prelude.not` (alias for `Prelude.Bool.not`)
  - `Prelude.Bool.not` ..... `Prelude/Bool.ad`, line 23
    - \* constant
    - \* type `Bool -> Bool`
    - \* executable
- **notElem**
  - `Data.List.notElem` (alias for `Prelude.List.notElem`)
  - `Prelude.notElem` (alias for `Prelude.List.notElem`)
  - `Prelude.List.notElem` ..... `Prelude/List.ad`, line 218
    - \* constant
    - \* type `'a -> ['a] -> Bool`
    - \* executable
- **notElemP**
  - `Data.List.notElemP` (alias for `Prelude.List.notElemP`)
  - `Prelude.List.notElemP` ..... `Prelude/List.ad`, line 227
    - \* constant
    - \* type `'a -> ['a] -> Prop`
    - \* non executable
- **Nothing**
  - `Data.Maybe.Nothing` (alias for `Prelude.Maybe.Nothing`)
  - `Prelude.Nothing` (alias for `Prelude.Maybe.Nothing`)

- `Prelude.Maybe.Nothing` ..... `Prelude/Maybe.ad`, line 21
  - \* constructor of datatype `Maybe`
  - \* type `Maybe 'a`
  - \* executable
- `notP`
  - `Prelude.notP` (alias for `Prelude.Bool.notP`)
  - `Prelude.Bool.notP` ..... `Prelude/Bool.ad`, line 62
    - \* constant
    - \* type `Prop -> Prop`
    - \* non executable
- `nub`
  - `Data.List.nub` (alias for `Prelude.List.nub`)
  - `Prelude.nub` (alias for `Prelude.List.nub`)
  - `Prelude.List.nub` ..... `Prelude/List.ad`, line 238
    - \* constant
    - \* type `['a] -> ['a]`
    - \* executable
- `nubBy`
  - `Data.List.nubBy` (alias for `Prelude.List.nubBy`)
  - `Prelude.nubBy` (alias for `Prelude.List.nubBy`)
  - `Prelude.List.nubBy` ..... `Prelude/List.ad`, line 234
    - \* constant
    - \* type `('a -> 'a -> Bool) -> ['a] -> ['a]`
    - \* executable
- `null`
  - `Data.List.null` (alias for `Prelude.List.null`)
  - `Prelude.null` (alias for `Prelude.Foldable.null`)
  - `Prelude.Foldable.null` ..... `Prelude/Foldable.ad`, line 36
    - \* constant of type-class `Foldable`
    - \* type `'t -> Bool`
    - \* executable
  - `Prelude.List.null` ..... `Prelude/List.ad`, line 135
    - \* constant
    - \* type `['a] -> Bool`
    - \* executable

### 3.2.14 O

- `occur`
  - `Prelude.occur` (alias for `Prelude.Collection.occur`)
  - `Prelude.Collection.occur` ..... `Prelude/Collection.ad`, line 89
    - \* constant of type-class `CollectionWithMembershipTest`
    - \* type `'e -> 'c -> Natural`
    - \* executable

- **or**
  - `Data.List.or` (alias for `Prelude.List.or`)
  - `Prelude.or` (alias for `Prelude.List.or`)
  - `Prelude.List.or` ..... `Prelude/List.ad`, line 188
    - \* constant
    - \* type `[Bool] -> Bool`
    - \* executable
- **ord**
  - `Prelude.ord` (alias for `Prelude.Char.ord`)
  - `Prelude.Char.ord` ..... `Prelude/Char.ad`, line 32
    - \* constant
    - \* type `Char -> Natural`
    - \* executable
- **orP**
  - `Data.List.orP` (alias for `Prelude.List.orP`)
  - `Prelude.orP` (alias for `Prelude.List.orP`)
  - `Prelude.List.orP` ..... `Prelude/List.ad`, line 193
    - \* constant
    - \* type `[Prop] -> Prop`
    - \* non executable

### 3.2.15 P

- **pack**
  - `Prelude.pack` (alias for `Prelude.Text.pack`)
  - `Prelude.Text.pack` ..... `Prelude/Text.ad`, line 20
    - \* constant
    - \* type `String -> Text`
    - \* executable
- **partition**
  - `Data.List.partition` (alias for `Prelude.List.partition`)
  - `Prelude.partition` (alias for `Prelude.Collection.partition`)
  - `Prelude.Collection.partition` ..... `Prelude/Collection.ad`, line 119
    - \* constant of type-class `CollectionWithDelete`
    - \* type `('e -> Bool) -> 'c -> ('c, 'c)`
    - \* executable
  - `Prelude.List.partition` ..... `Prelude/List.ad`, line 105
    - \* constant
    - \* type `('a -> Bool) -> ['a] -> (['a], ['a])`
    - \* executable
- **pop**
  - `Data.Collection.Stack.pop` ..... `Data/Collection/Stack.ad`, line 18
    - \* constant
    - \* type `Data.Collection.Stack.Stack 'a -> Maybe ('a, Data.Collection.Stack.Stack 'a)`
    - \* executable

- **pred**
  - `Prelude.pred` (alias for `Prelude.BasicClasses.pred`)
  - `Prelude.BasicClasses.pred` ..... `Prelude/BasicClasses.ad`, line 27
    - \* constant of type-class `Enum`
    - \* type `'a -> 'a`
    - \* executable
- **pure**
  - `Prelude.pure` (alias for `Prelude.Monad.return`)
  - `Prelude.Monad.pure` (alias for `Prelude.Monad.return`)
- **push**
  - `Data.Collection.Stack.push` ..... `Data/Collection/Stack.ad`, line 15
    - \* constant
    - \* type `'a -> Data.Collection.Stack.Stack 'a -> Data.Collection.Stack.Stack 'a`
    - \* executable

### 3.2.16 R

- **removeFirst**
  - `Data.List.removeFirst` (alias for `Prelude.List.removeFirst`)
  - `Prelude.removeFirst` (alias for `Prelude.List.removeFirst`)
  - `Prelude.List.removeFirst` ..... `Prelude/List.ad`, line 123
    - \* constant
    - \* type `('a -> Bool) -> ['a] -> ['a]`
    - \* executable
- **replicate**
  - `Data.List.replicate` ..... `Data/List.ad`, line 16
    - \* constant
    - \* type `Natural -> 'a -> ['a]`
    - \* executable
- **return**
  - `Prelude.return` (alias for `Prelude.Monad.return`)
  - `Prelude.Monad.return` ..... `Prelude/Monad.ad`, line 31
    - \* statically resolved constant of type-class `Monad`
    - \* type `'a -> 'M 'a`
    - \* executable
- **revAppend**
  - `Data.List.revAppend` (alias for `Prelude.List.revAppend`)
  - `Prelude.revAppend` (alias for `Prelude.List.revAppend`)
  - `Prelude.List.revAppend` ..... `Prelude/List.ad`, line 171
    - \* constant
    - \* type `['a] -> ['a] -> ['a]`
    - \* executable
- **reverse**
  - `Data.List.reverse` (alias for `Prelude.List.reverse`)



- `Prelude.reverse` (alias for `Prelude.List.reverse`)
- `Prelude.List.reverse` .....`Prelude/List.ad`, line 175
  - \* constant
  - \* type `['a] -> ['a]`
  - \* executable
- **Right**
  - `Prelude.Right` (alias for `Prelude.Either.Right`)
  - `Prelude.Either.Right` .....`Prelude/Either.ad`, line 13
    - \* constructor of datatype `Either`
    - \* type `'b -> Either 'a 'b`
    - \* executable

### 3.2.17 S

- **scanl**
  - `Data.List.scanl` (alias for `Prelude.List.scanl`)
  - `Prelude.scanl` (alias for `Prelude.List.scanl`)
  - `Prelude.List.scanl` .....`Prelude/List.ad`, line 148
    - \* constant
    - \* type `('a -> 'b -> 'a) -> 'a -> ['b] -> ['a]`
    - \* executable
- **scanr**
  - `Data.List.scanr` (alias for `Prelude.List.scanr`)
  - `Prelude.scanr` (alias for `Prelude.List.scanr`)
  - `Prelude.List.scanr` .....`Prelude/List.ad`, line 157
    - \* constant
    - \* type `('a -> 'b -> 'b) -> 'b -> ['a] -> ['b]`
    - \* executable
- **sequence**
  - `Prelude.sequence` (alias for `Prelude.Monad.sequence`)
  - `Prelude.Monad.sequence` .....`Prelude/Monad.ad`, line 64
    - \* constant
    - \* type `[( 'M 'a)] -> 'M ['a]`
    - \* executable
- **sequence\_**
  - `Prelude.sequence_` (alias for `Prelude.Monad.sequence_`)
  - `Prelude.Monad.sequence_` .....`Prelude/Monad.ad`, line 68
    - \* constant
    - \* type `[( 'M 'a)] -> 'M ()`
    - \* executable
- **set**
  - `Data.Lens.set` .....`Data/Lens.ad`, line 80
    - \* constant of type-class `Data.Lens.IsSetter`
    - \* type `'v -> 's -> 'b -> 't`

- \* executable
- **setEmpty**
  - `Prelude.setEmpty` (alias for `Prelude.Collection.setEmpty`)
  - `Prelude.Collection.setEmpty` ..... `Prelude/Collection.ad`, line 183
    - \* constant
    - \* type 'c
    - \* executable
- **setFromList**
  - `Prelude.setFromList` (alias for `Prelude.Collection.setFromList`)
  - `Prelude.Collection.setFromList` ..... `Prelude/Collection.ad`, line 191
    - \* constant
    - \* type ['e] -> 'c
    - \* executable
- **setInsert**
  - `Prelude.setInsert` (alias for `Prelude.Collection.setInsert`)
  - `Prelude.Collection.setInsert` ..... `Prelude/Collection.ad`, line 187
    - \* constant
    - \* type 'e -> 'c -> 'c
    - \* executable
- **Setter**
  - `Data.Lens.Setter` ..... `Data/Lens.ad`, line 60
    - \* constructor of record-type `Data.Lens.Setter`
    - \* type ('s -> 'b -> 't) -> `Data.Lens.Setter` 's 't 'b
    - \* executable
- **setterSet**
  - `Data.Lens.setterSet` ..... `Data/Lens.ad`, line 61
    - \* field of record-type `Data.Lens.Setter`
    - \* type `Data.Lens.Setter` 's 't 'b -> 's -> 'b -> 't
    - \* executable
- **simpleLensToGetter**
  - `Data.Lens.simpleLensToGetter` ..... `Data/Lens.ad`, line 98
    - \* constant
    - \* type `Data.Lens.SimpleLens` 'a 'b -> `Data.Lens.Getter` 'a 'b
    - \* executable
- **singleton**
  - `Prelude.singleton` (alias for `Prelude.Collection.singleton`)
  - `Prelude.Collection.singleton` ..... `Prelude/Collection.ad`, line 52
    - \* constant of type-class `Collection`
    - \* type 'e -> 'c
    - \* executable
- **size**
  - `Prelude.size` (alias for `Prelude.Foldable.size`)
  - `Prelude.Foldable.size` ..... `Prelude/Foldable.ad`, line 35

- \* constant of type-class **Foldable**
- \* type 't -> **Natural**
- \* executable
- **snd**
  - **Prelude.snd** (alias for **Prelude.Tuple.snd**)
  - **Prelude.Tuple.snd** ..... **Prelude/Tuple.ad**, line 29
    - \* constant
    - \* type ('a, 'b) -> 'b
    - \* executable
- **snoc**
  - **Data.List.snoc** (alias for **Prelude.List.snoc**)
  - **Prelude.snoc** (alias for **Prelude.List.snoc**)
  - **Prelude.List.snoc** ..... **Prelude/List.ad**, line 74
    - \* constant
    - \* type ['a] -> 'a -> ['a]
    - \* executable
- **span**
  - **Data.List.span** ..... **Data/List.ad**, line 53
    - \* constant
    - \* type ('a -> Bool) -> ['a] -> (['a], ['a])
    - \* executable
- **splitAt**
  - **Data.List.splitAt** ..... **Data/List.ad**, line 30
    - \* constant
    - \* type **Natural** -> ['a] -> (['a], ['a])
    - \* executable
- **Suc**
  - **Prelude.Suc** (alias for **Prelude.Num.Suc**)
  - **Prelude.Num.Suc** ..... **Prelude/Num.ad**, line 21
    - \* constructor of datatype **Natural**
    - \* type **Natural** -> **Natural**
    - \* executable
- **succ**
  - **Prelude.succ** (alias for **Prelude.BasicClasses.succ**)
  - **Prelude.BasicClasses.succ** ..... **Prelude/BasicClasses.ad**, line 26
    - \* constant of type-class **Enum**
    - \* type 'a -> 'a
    - \* executable

## 3.2.18 T

- **take**
  - `Data.List.take` ..... `Data/List.ad`, line 20
    - \* constant
    - \* type `Natural -> ['a] -> ['a]`
    - \* executable
- **takeFirst**
  - `Data.List.takeFirst` (alias for `Prelude.List.takeFirst`)
  - `Prelude.takeFirst` (alias for `Prelude.List.takeFirst`)
  - `Prelude.List.takeFirst` ..... `Prelude/List.ad`, line 114
    - \* constant
    - \* type `('a -> Bool) -> ['a] -> Maybe (['a], 'a)`
    - \* executable
- **takeWhile**
  - `Data.List.takeWhile` ..... `Data/List.ad`, line 43
    - \* constant
    - \* type `('a -> Bool) -> ['a] -> ['a]`
    - \* executable
- **to**
  - `Data.Lens.to` ..... `Data/Lens.ad`, line 55
    - \* constant
    - \* type `('a -> 'b) -> Data.Lens.Getter 'a 'b`
    - \* executable
- **toEnum**
  - `Prelude.toEnum` (alias for `Prelude.BasicClasses.toEnum`)
  - `Prelude.BasicClasses.toEnum` ..... `Prelude/BasicClasses.ad`, line 30
    - \* constant of type-class `Enum`
    - \* type `Natural -> Maybe 'a`
    - \* executable
- **toGetter**
  - `Data.Lens.toGetter` ..... `Data/Lens.ad`, line 71
    - \* constant of type-class `Data.Lens.IsGetter`
    - \* type `'v -> Data.Lens.Getter 'a 'b`
    - \* executable
- **toList**
  - `Prelude.toList` (alias for `Prelude.Foldable.toList`)
  - `Prelude.Foldable.toList` ..... `Prelude/Foldable.ad`, line 20
    - \* constant of type-class `Foldable`
    - \* type `'t -> ['e]`
    - \* executable
- **toSetter**
  - `Data.Lens.toSetter` ..... `Data/Lens.ad`, line 82
    - \* constant of type-class `Data.Lens.IsSetter`

\* type 'v -> Data.Lens.Setter 's 't 'b  
 \* executable

- **true**

- `Prelude.true` (alias for `Prelude.Bool.true`)
- `Prelude.Bool.true` ..... `Prelude/Bool.ad`, line 55
  - \* constant
  - \* type `Prop`
  - \* non executable

- **True**

- `Prelude.True` (alias for `Prelude.Bool.True`)
- `Prelude.Bool.True` ..... `Prelude/Bool.ad`, line 22
  - \* constructor of datatype `Bool`
  - \* type `Bool`
  - \* executable

### 3.2.19 U

- **uexistsB\_fun**

- `Prelude.uexistsB_fun` (alias for `Prelude.BasicClasses.uexistsB_fun`)
- `Prelude.BasicClasses.uexistsB_fun` ..... `Prelude/BasicClasses.ad`, line 98
  - \* constant of type-class `Finite`
  - \* type (`'a -> Bool`) -> `Bool`
  - \* executable

- **uexistsP\_fun**

- `Prelude.uexistsP_fun` (alias for `Prelude.Bool.uexistsP_fun`)
- `Prelude.Bool.uexistsP_fun` ..... `Prelude/Bool.ad`, line 72
  - \* constant
  - \* type (`'a -> Prop`) -> `Prop`
  - \* non executable

- **uncurry**

- `Prelude.uncurry` (alias for `Prelude.Tuple.uncurry`)
- `Prelude.Tuple.uncurry` ..... `Prelude/Tuple.ad`, line 35
  - \* constant
  - \* type (`'a -> 'b -> 'c`) -> (`'a, 'b`) -> `'c`
  - \* executable

- **undefined**

- `Prelude.undefined` (alias for `Prelude.Default.undefined`)
- `Prelude.Default.undefined` ..... `Prelude/Default.ad`, line 13
  - \* constant of type-class `Default`
  - \* type `'a`
  - \* executable

- **union**

- `Prelude.union` (alias for `Prelude.Collection.union`)
- `Prelude.Collection.union` ..... `Prelude/Collection.ad`, line 69

- \* constant
- \* type 'c -> 'c -> 'c
- \* executable
- **Unit**
  - **Prelude.Unit** (alias for **Prelude.Unit.Unit**)
  - **Prelude.Unit.Unit** .....Prelude/Unit.ad, line 8
    - \* constructor of datatype ()
    - \* type ()
    - \* executable
- **universeFoldWithAbort**
  - **Prelude.universeFoldWithAbort** (alias for **Prelude.BasicClasses.universeFoldWithAbort**)
  - **Prelude.BasicClasses.universeFoldWithAbort** .....Prelude/BasicClasses.ad, line 93
    - \* statically resolved constant of type-class **Finite**
    - \* type ('acc -> Bool) -> ('acc -> 'a -> 'acc) -> 'acc -> 'acc
    - \* executable
- **universeList**
  - **Prelude.universeList** (alias for **Prelude.BasicClasses.universeList**)
  - **Prelude.BasicClasses.universeList** .....Prelude/BasicClasses.ad, line 91
    - \* constant of type-class **Finite**
    - \* type ['a]
    - \* executable
- **unpack**
  - **Prelude.unpack** (alias for **Prelude.Text.unpack**)
  - **Prelude.Text.unpack** .....Prelude/Text.ad, line 23
    - \* constant
    - \* type Text -> String
    - \* executable
- **unreachable**
  - **Prelude.unreachable** (alias for **Prelude.Default.unreachable**)
  - **Prelude.Default.unreachable** .....Prelude/Default.ad, line 18
    - \* constant
    - \* type 'a
    - \* executable
- **unzip**
  - **Data.List.unzip** .....Data/List.ad, line 79
    - \* constant
    - \* type [('a, 'b)] -> (['a], ['b])
    - \* executable

### 3.2.20 V

- **view**
  - **Data.Lens.view** .....Data/Lens.ad, line 69
    - \* constant of type-class **Data.Lens.IsGetter**
    - \* type 'v -> 'a -> 'b
    - \* executable

### 3.2.21 Z

- Zero
  - `Prelude.Zero` (alias for `Prelude.Num.Zero`)
  - `Prelude.Num.Zero` ..... `Prelude/Num.ad`, line 20
    - \* constructor of datatype `Natural`
    - \* type `Natural`
    - \* executable
- zip
  - `Data.List.zip` ..... `Data/List.ad`, line 76
    - \* constant
    - \* type `['a] -> ['b] -> [( 'a, 'b)]`
    - \* executable

## 3.3 Templates

### 3.3.1 B

- `BuildInstance`
  - `Prelude.BuildInstance` (alias for `Prelude.Template.BuildInstance`)
  - `Prelude.Template.BuildInstance` ..... `Prelude/Template.ad`, lines 215 - 219
- `BuildInstanceHeader`
  - `Prelude.BuildInstanceHeader` (alias for `Prelude.Template.BuildInstanceHeader`)
  - `Prelude.Template.BuildInstanceHeader` ..... `Prelude/Template.ad`, lines 205 - 212
- `BuildRecordFieldLens`
  - `Data.Lens.BuildRecordFieldLens` ..... `Data/Lens.ad`, lines 224 - 238
- `BuildRecordFieldLenses`
  - `Data.Lens.BuildRecordFieldLenses` ..... `Data/Lens.ad`, line 257
- `BuildRecordFieldLensesPrefix`
  - `Data.Lens.BuildRecordFieldLensesPrefix` ..... `Data/Lens.ad`, lines 249 - 255
- `BuildRecordFieldLensesSuffix`
  - `Data.Lens.BuildRecordFieldLensesSuffix` ..... `Data/Lens.ad`, lines 241 - 247
- `BuildTuple`
  - `Prelude.BuildTuple` (alias for `Prelude.Template.BuildTuple`)
  - `Prelude.Template.BuildTuple` ..... `Prelude/Template.ad`, lines 79 - 88
- `BuildTupleLeft`
  - `Prelude.BuildTupleLeft` (alias for `Prelude.Template.BuildTupleLeft`)
  - `Prelude.Template.BuildTupleLeft` ..... `Prelude/Template.ad`, lines 91 - 104
- `BuildTupleRight`
  - `Prelude.BuildTupleRight` (alias for `Prelude.Template.BuildTupleRight`)
  - `Prelude.Template.BuildTupleRight` ..... `Prelude/Template.ad`, lines 108 - 121

**3.3.2 C**

- **ClassWithArgs**
  - `Prelude.ClassWithArgs` (alias for `Prelude.Template.ClassWithArgs`)
  - `Prelude.Template.ClassWithArgs` ..... `Prelude/Template.ad`, lines 197 - 200
- **Commafy**
  - `Prelude.Commafy` (alias for `Prelude.Template.Commafy`)
  - `Prelude.Template.Commafy` ..... `Prelude/Template.ad`, lines 74 - 76
- **ConstWithArgs**
  - `Prelude.ConstWithArgs` (alias for `Prelude.Template.ConstWithArgs`)
  - `Prelude.Template.ConstWithArgs` ..... `Prelude/Template.ad`, lines 162 - 165
- **ConstWithWildcardArgs**
  - `Prelude.ConstWithWildcardArgs` (alias for `Prelude.Template.ConstWithWildcardArgs`)
  - `Prelude.Template.ConstWithWildcardArgs` ..... `Prelude/Template.ad`, lines 151 - 154

**3.3.3 D**

- **DeriveInstance**
  - `Prelude.DeriveInstance` (alias for `Prelude.Template.DeriveInstance`)
  - `Prelude.Template.DeriveInstance` ..... `Prelude/Template.ad`, lines 243 - 248
- **DeriveInstanceComputeSuperDefault**
  - `Prelude.DeriveInstanceComputeSuperDefault` (alias for `Prelude.Template.DeriveInstanceComputeSuperDefault`)
  - `Prelude.Template.DeriveInstanceComputeSuperDefault` . . . `Prelude/Template.ad`, lines 226 - 238
- **DerivingInstance**
  - `Prelude.DerivingInstance` (alias for `Prelude.Template.DerivingInstance`)
  - `Prelude.Template.DerivingInstance` ..... `Prelude/Template.ad`, lines 251 - 255

**3.3.4 G**

- **GenerateRecognisers**
  - `Prelude.GenerateRecognisers` (alias for `Prelude.Template.GenerateRecognisers`)
  - `Prelude.Template.GenerateRecognisers` ..... `Prelude/Template.ad`, lines 179 - 189
- **GetTypeVar**
  - `Prelude.GetTypeVar` (alias for `Prelude.Template.GetTypeVar`)
  - `Prelude.Template.GetTypeVar` ..... `Prelude/Template.ad`, lines 30 - 39
- **GetTypeVars**
  - `Prelude.GetTypeVars` (alias for `Prelude.Template.GetTypeVars`)
  - `Prelude.Template.GetTypeVars` ..... `Prelude/Template.ad`, lines 42 - 48
- **GetTypeVarsNumbered**
  - `Prelude.GetTypeVarsNumbered` (alias for `Prelude.Template.GetTypeVarsNumbered`)
  - `Prelude.Template.GetTypeVarsNumbered` ..... `Prelude/Template.ad`, lines 51 - 58



### 3.3.5 I

- Intercalate
  - `Prelude.Intercalate` (alias for `Prelude.Template.Intercalate`)
  - `Prelude.Template.Intercalate` ..... `Prelude/Template.ad`, lines 61 - 71
- `IsSimpleEnumDatatype`
  - `Prelude.IsSimpleEnumDatatype` (alias for `Prelude.Template.IsSimpleEnumDatatype`)
  - `Prelude.Template.IsSimpleEnumDatatype` ..... `Prelude/Template.ad`, lines 124 - 133

### 3.3.6 T

- `TypeWithArgs`
  - `Prelude.TypeWithArgs` (alias for `Prelude.Template.TypeWithArgs`)
  - `Prelude.Template.TypeWithArgs` ..... `Prelude/Template.ad`, lines 173 - 176

## 3.4 Types

### 3.4.1 B

- `Bool`
  - `Prelude.Bool` (alias for `Prelude.Bool.Bool`)
  - `Prelude.Bool.Bool` ..... `Prelude/Bool.ad`, line 22
    - \* datatype `Bool`
    - \* constructors
      - `False :: Bool`
      - `True :: Bool`
- `BuiltInChar`
  - `Prelude.BuiltInChar` (alias for `Prelude.Literal.BuiltInChar`)
  - `Prelude.Literal.BuiltInChar` ..... `Prelude/Literal.ad`, line 49
    - \* type `BuiltInChar`
- `BuiltInDecimal`
  - `Prelude.BuiltInDecimal` (alias for `Prelude.Literal.BuiltInDecimal`)
  - `Prelude.Literal.BuiltInDecimal` ..... `Prelude/Literal.ad`, line 104
    - \* type `BuiltInDecimal`
- `BuiltInNum`
  - `Prelude.BuiltInNum` (alias for `Prelude.Literal.BuiltInNum`)
  - `Prelude.Literal.BuiltInNum` ..... `Prelude/Literal.ad`, line 19
    - \* type `BuiltInNum`
- `BuiltInString`
  - `Prelude.BuiltInString` (alias for `Prelude.Literal.BuiltInString`)
  - `Prelude.Literal.BuiltInString` ..... `Prelude/Literal.ad`, line 77
    - \* type `BuiltInString`

### 3.4.2 C

- Char
  - `Prelude.Char` (alias for `Prelude.Char.Char`)
  - `Prelude.Char.Char` ..... `Prelude/Char.ad`, line 20
    - \* datatype `Char`
    - \* constructors
      - `Char :: Natural -> Char`

### 3.4.3 D

- `DistinctListSet`
  - `Data.Collection.DistinctListSet.DistinctListSet` .. `Data/Collection/DistinctListSet.ad`, line 7
    - \* datatype `DistinctListSet 'a`
    - \* constructors
      - `DistinctListSet :: ['a] -> Data.Collection.DistinctListSet.DistinctListSet 'a`

### 3.4.4 E

- `Either`
  - `Prelude.Either` (alias for `Prelude.Either.Either`)
  - `Prelude.Either.Either` ..... `Prelude/Either.ad`, lines 11 - 13
    - \* datatype `Either 'a 'b`
    - \* constructors
      - `Left :: 'a -> Either 'a 'b`
      - `Right :: 'b -> Either 'a 'b`

### 3.4.5 F

- `FunSetBool`
  - `Data.Collection.FunSetBool.FunSetBool` ..... `Data/Collection/FunSetBool.ad`, line 6
    - \* datatype `FunSetBool 'a`
    - \* constructors
      - `FunSetBool :: ('a -> Bool) -> Data.Collection.FunSetBool.FunSetBool 'a`

### 3.4.6 G

- `Getter`
  - `Data.Lens.Getter` ..... `Data/Lens.ad`, lines 50 - 52
    - \* record-type `Getter 'a 'b`
    - \* fields
      - `getterGet :: Data.Lens.Getter 'a 'b -> 'a -> 'b`

### 3.4.7 I

- `Itself`
  - `Prelude.Itself` (alias for `Prelude.Unit.Itself`)
  - `Prelude.Unit.Itself` ..... `Prelude/Unit.ad`, line 11
    - \* datatype `Itself 'a`
    - \* constructors
      - `Itself :: (::'a)`

**3.4.8 L**

- **Lens**
  - `Data.Lens.Lens` ..... `Data/Lens.ad`, lines 41 - 44
    - \* record-type `Lens 's 't 'a 'b`
    - \* fields
      - `lensGet :: Data.Lens.Lens 's 't 'a 'b -> 's -> 'a`
      - `lensSet :: Data.Lens.Lens 's 't 'a 'b -> 's -> 'b -> 't`
- **List**
  - `Data.List.List` (alias for `Prelude.List.List`)
  - `Prelude.List` (alias for `Prelude.List.List`)
  - `Prelude.List.List` ..... `Prelude/List.ad`, line 50
    - \* datatype `List 'a`
    - \* constructors
      - `Nil :: ['a]`
      - `(:) :: 'a -> ['a] -> ['a]`
- **ListBag**
  - `Data.Collection.ListBag.ListBag` ..... `Data/Collection/ListBag.ad`, line 8
    - \* datatype `ListBag 'a`
    - \* constructors
      - `ListBag :: ['a] -> Data.Collection.ListBag.ListBag 'a`

**3.4.9 M**

- **Maybe**
  - `Data.Maybe.Maybe` (alias for `Prelude.Maybe.Maybe`)
  - `Prelude.Maybe` (alias for `Prelude.Maybe.Maybe`)
  - `Prelude.Maybe.Maybe` ..... `Prelude/Maybe.ad`, lines 20 - 22
    - \* datatype `Maybe 'a`
    - \* constructors
      - `Nothing :: Maybe 'a`
      - `Just :: 'a -> Maybe 'a`

**3.4.10 N**

- **Natural**
  - `Prelude.Natural` (alias for `Prelude.Num.Natural`)
  - `Prelude.Num.Natural` ..... `Prelude/Num.ad`, lines 19 - 21
    - \* datatype `Natural`
    - \* constructors
      - `Zero :: Natural`
      - `Suc :: Natural -> Natural`

### 3.4.11 O

- Ordering
  - `Prelude.Ordering` (alias for `Prelude.Ord.Ordering`)
  - `Prelude.Ord.Ordering` ..... `Prelude/Ord.ad`, lines 16 - 19
    - \* datatype `Ordering`
    - \* constructors
      - `LT :: Ordering`
      - `EQ :: Ordering`
      - `GT :: Ordering`

### 3.4.12 P

- Prop
  - `Prelude.Prop` (alias for `Prelude.Bool.Prop`)
  - `Prelude.Bool.Prop` ..... `Prelude/Bool.ad`, line 53
    - \* type `Prop`
    - \* non executable

### 3.4.13 Q

- Queue
  - `Data.Collection.Queue.Queue` ..... `Data/Collection/Queue.ad`, line 10
    - \* datatype `Queue 'a`
    - \* constructors
      - `Queue :: ['a] -> ['a] -> Data.Collection.Queue.Queue 'a`

### 3.4.14 S

- Setter
  - `Data.Lens.Setter` ..... `Data/Lens.ad`, lines 60 - 62
    - \* record-type `Setter 's 't 'b`
    - \* fields
      - `setterSet :: Data.Lens.Setter 's 't 'b -> 's -> 'b -> 't`
- SimpleLens
  - `Data.Lens.SimpleLens` ..... `Data/Lens.ad`, line 47
    - \* abbreviation-type `SimpleLens 's 'a`
    - \* abbreviation for `Data.Lens.Lens 's 's 'a 'a`
- SimpleSetter
  - `Data.Lens.SimpleSetter` ..... `Data/Lens.ad`, line 64
    - \* abbreviation-type `SimpleSetter 's 'b`
    - \* abbreviation for `Data.Lens.Setter 's 's 'b`
- Stack
  - `Data.Collection.Stack.Stack` ..... `Data/Collection/Stack.ad`, line 10
    - \* datatype `Stack 'a`
    - \* constructors
      - `Stack :: ['a] -> Data.Collection.Stack.Stack 'a`
- String

- `Prelude.String` (alias for `Prelude.String.String`)
- `Prelude.String.String` ..... `Prelude/String.ad`, line 12
  - \* abbreviation-type `String`
  - \* abbreviation for `[Char]`

### 3.4.15 T

- **Text**
  - `Prelude.Text` (alias for `Prelude.Text.Text`)
  - `Prelude.Text.Text` ..... `Prelude/Text.ad`, line 18
    - \* datatype `Text`
    - \* constructors
      - `Text :: String -> Text`
- **Tuple2**
  - `Prelude.Tuple2` (alias for `Prelude.Tuple.Tuple2`)
  - `Prelude.Tuple.Tuple2` ..... `Prelude/Tuple.ad`, line 56
    - \* abbreviation-type `Tuple2 'a`
    - \* abbreviation for `('a, 'a)`
- **Tuple3**
  - `Prelude.Tuple3` (alias for `Prelude.Tuple.Tuple3`)
  - `Prelude.Tuple.Tuple3` ..... `Prelude/Tuple.ad`, line 56
    - \* abbreviation-type `Tuple3 'a`
    - \* abbreviation for `('a, 'a, 'a)`
- **Tuple4**
  - `Prelude.Tuple4` (alias for `Prelude.Tuple.Tuple4`)
  - `Prelude.Tuple.Tuple4` ..... `Prelude/Tuple.ad`, line 56
    - \* abbreviation-type `Tuple4 'a`
    - \* abbreviation for `('a, 'a, 'a, 'a)`
- **Tuple5**
  - `Prelude.Tuple5` (alias for `Prelude.Tuple.Tuple5`)
  - `Prelude.Tuple.Tuple5` ..... `Prelude/Tuple.ad`, line 56
    - \* abbreviation-type `Tuple5 'a`
    - \* abbreviation for `('a, 'a, 'a, 'a, 'a)`
- **Tuple6**
  - `Prelude.Tuple6` (alias for `Prelude.Tuple.Tuple6`)
  - `Prelude.Tuple.Tuple6` ..... `Prelude/Tuple.ad`, line 56
    - \* abbreviation-type `Tuple6 'a`
    - \* abbreviation for `('a, 'a, 'a, 'a, 'a, 'a)`
- **Tuple7**
  - `Prelude.Tuple7` (alias for `Prelude.Tuple.Tuple7`)
  - `Prelude.Tuple.Tuple7` ..... `Prelude/Tuple.ad`, line 56
    - \* abbreviation-type `Tuple7 'a`
    - \* abbreviation for `('a, 'a, 'a, 'a, 'a, 'a, 'a)`
- **Tuple8**

- `Prelude.Tuple8` (alias for `Prelude.Tuple.Tuple8`)
- `Prelude.Tuple.Tuple8` ..... `Prelude/Tuple.ad`, line 56
  - \* abbreviation-type `Tuple8 'a`
  - \* abbreviation for `('a, 'a, 'a, 'a, 'a, 'a, 'a, 'a)`
- **Tuple9**
  - `Prelude.Tuple9` (alias for `Prelude.Tuple.Tuple9`)
  - `Prelude.Tuple.Tuple9` ..... `Prelude/Tuple.ad`, line 56
    - \* abbreviation-type `Tuple9 'a`
    - \* abbreviation for `('a, 'a, 'a, 'a, 'a, 'a, 'a, 'a, 'a)`

### 3.4.16 U

- **Unit**
  - `Prelude.Unit` (alias for `Prelude.Unit.Unit`)
  - `Prelude.Unit.Unit` ..... `Prelude/Unit.ad`, line 8
    - \* datatype `Unit`
    - \* constructors
      - `Unit :: ()`