Contents

Comparisons and Ordering

```
(==)   :  {a}   (Cmp a) => a -> a -> Bit
(!=)   :  {a}   (Cmp a) => a -> a -> Bit
(===)  :  {a,b} (Cmp b) => (a -> b) -> (a -> b) -> a -> Bit
(!==)  :  {a,b} (Cmp b) => (a -> b) -> (a -> b) -> a -> Bit

(<)    :  {a} (Cmp a) => a -> a -> Bit
(>)    :  {a} (Cmp a) => a -> a -> Bit
(<=)   :  {a} (Cmp a) => a -> a -> Bit
(>=)   :  {a} (Cmp a) => a -> a -> Bit

min    :  {a} (Cmp a) => a -> a -> a
max    :  {a} (Cmp a) => a -> a -> a

instance Cmp Bit
// No instance for functions.
instance (Cmp a, fin n) => Cmp [n]a
instance (Cmp a, Cmp b) => Cmp (a, b)
instance (Cmp a, Cmp b) => Cmp { x : a, y : b }
instance                   Cmp Integer
```

Signed Comparisons

```
(<$) : {a} (SignedCmp a) => a -> a -> Bit
(>$) : {a} (SignedCmp a) => a -> a -> Bit
(<=$) : {a} (SignedCmp a) => a -> a -> Bit
(>=$) : {a} (SignedCmp a) => a -> a -> Bit

// No instance for Bit
// No instance for functions.
instance (fin n, n >= 1)            => SignedCmp [n]
instance (SignedCmp a, fin n)       => SignedCmp [n]a
```

```
        // (for [n]a, where a is other than Bit)
instance (SignedCmp a, SignedCmp b) => SignedCmp (a, b)
instance (SignedCmp a, SignedCmp b) => SignedCmp { x : a, y : b }
```

Arithmetic

```
(+)    : {a} (Arith a) => a -> a -> a
(-)    : {a} (Arith a) => a -> a -> a
(*)    : {a} (Arith a) => a -> a -> a
(/)    : {a} (Arith a) => a -> a -> a
(%)    : {a} (Arith a) => a -> a -> a
(^^)   : {a} (Arith a) => a -> a -> a
(/$)   : {a} (Arith a) => a -> a -> a
(%$)   : {a} (Arith a) => a -> a -> a
lg2    : {a} (Arith a) => a -> a
negate : {a} (Arith a) => a -> a
```

The prefix notation `- x` is syntactic sugar for `negate x`.

```
// No instance for `Bit`.
instance (fin n)            => Arith ([n]Bit)
instance (Arith a)          => Arith ([n]a)
instance (Arith b)          => Arith (a -> b)
instance (Arith a, Arith b) => Arith (a, b)
instance (Arith a, Arith b) => Arith { x : a, y : b }
instance                       Arith Integer
```

Note that because there is no instance for `Arith Bit` the top two instances do not actually overlap.

Boolean

```
False      : Bit
True       : Bit

zero       : {a} (Zero a) => a
(&&)       : {a} (Logic a) => a -> a -> a
(||)       : {a} (Logic a) => a -> a -> a
(^)        : {a} (Logic a) => a -> a -> a
complement : {a} (Logic a) => a -> a

(==>)      : Bit -> Bit -> Bit
(/\)       : Bit -> Bit -> Bit
(\/)       : Bit -> Bit -> Bit
```

```
instance                        Logic Bit
instance (Logic a)           => Logic ([n]a)
instance (Logic b)           => Logic (a -> b)
instance (Logic a, Logic b) => Logic (a, b)
instance (Logic a, Logic b) => Logic { x : a, y : b }
// No instance for `Logic Integer`.
```

Sequences

```
join      : {parts,each,a} (fin each) => [parts][each]a -> [parts * each]a
split     : {parts,each,a} (fin each) => [parts * each]a -> [parts][each]a

(#)       : {front,back,a} (fin front) => [front]a -> [back]a -> [front + back]a
splitAt   : {front,back,a} (fin front) => [from + back] a -> ([front] a, [back] a)

reverse   : {n,a} (fin n) => [n]a -> [n]a
transpose : {n,m,a} [n][m]a -> [m][n]a

(@)        : {n,a,m}              [n]a -> [m]    -> a
(@@)       : {n,a,m,i}            [n]a -> [m][i] -> [m]a
(!)        : {n,a,m}   (fin n) => [n]a -> [m]    -> a
(!!)       : {n,a,m,i} (fin n) => [n]a -> [m][i] -> [m]a
update     : {n,a,m} (fin m) => [n]a -> [m] -> a -> [n]a
updateEnd  : {n,a,m} (fin n, fin m) => [n]a -> [m] -> a -> [n]a
updates    : {n,a,m,d} (fin m, fin d) => [n]a -> [d][m] -> [d]a -> [n]a
updatesEnd : {n,a,m,d} (fin n, fin m, fin d) => [n]a -> [d][m] -> [d]a -> [n]a

take       : {front,back,elem} (fin front) => [front + back]elem -> [front]elem
drop       : {front,back,elem} (fin front) => [front + back]elem -> [back]elem
head       : {a, b} [1 + a]b -> b
tail       : {a, b} [1 + a]b -> [a]b
last       : {a, b} [1 + a]b -> b

groupBy    : {each,parts,elem} (fin each) => [parts * each]elem -> [parts][each]elem
```

Function `groupBy` is the same as `split` but with its type arguments in a different order.

Shift And Rotate

```
(<<)   : {n,a,m} (fin n, Zero a) => [n]a -> [m] -> [n]a
(>>)   : {n,a,m} (fin n, Zero a) => [n]a -> [m] -> [n]a
(<<<)  : {n,a,m} (fin n) => [n]a -> [m] -> [n]a
(>>>)  : {n,a,m} (fin n) => [n]a -> [m] -> [n]a
```

```
// Arithmetic shift only for bitvectors
(>>$)  : {n, k} (fin n, n >= 1, fin k) => [n] -> [k] -> [n]
```

Random Values

```
random    : {a} => [256] -> a
```

Debugging

```
undefined   : {a} a
error       : {n a} [n][8] -> a
trace       : {n, a, b} (fin n) => [n][8] -> a -> b -> b
traceVal    : {n, a} (fin n) => [n][8] -> a -> a
```