

Contents

Comparisons and Ordering	1
Signed Comparisons	1
Arithmetic	2
Boolean	2
Sequences	2
Shift And Rotate	3
Random Values	3
Debugging	4

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 }
```

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
```

// 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 }
```

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

Boolean

```
False : Bit
True  : Bit
```

```
zero : a
(&&) : a -> a -> a
(||) : a -> a -> a
(^)  : a -> a -> a
(~)  : a -> a
```

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

Sequences

```
length : {n,a,m} (m >= width n) => [n]a -> [m]
```

```

join      : {parts,ench,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

// Abbreviations
groupBy n = split`{each = n}
tail n    = splitAt`{front = 1}.1
take n    = splitAt`{front = n}.0
drop n    = splitAt`{front = n}.1

/* Also, `length` is not really needed:
   length : {n,a,m} (m >= width n) => [n]a -> [m]
   length _ = `n
*/

```

Shift And Rotate

New types:

```

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