0.1 Digit: the basic building block

As the basic building block of numerals, we will demonstrate how to choose a suitable representation for digits in this section.

0.1.1 Fin

To represent a digit, we use a datatype that is conventionally called *Fin* which can be indexed to have some exact number of inhabitants.

```
data Fin : ℕ → Set where
    zero : {n : ℕ} → Fin (suc n)
    suc  : {n : ℕ} (i : Fin n) → Fin (suc n)
```

The definition of `Fin` looks the same as ℕ on the term level, but different on the type level. The index of a `Fin` increases with every `suc`, and there can only be at most `n` of them before reaching `Fin (suc n)`. In other words, `Fin n` would have exactly n inhabitants.

0.1.2 Definition

`Digit` is simply just a synonym for `Fin`, indexed by the number of digits `d` of a system. Since the same digit may represent different values in different numeral systems, it is essential to make the context clear.

```
Digit : ℕ → Set
Digit d = Fin d
```

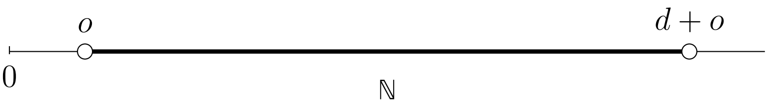
Ordinary binary digits for example can thus be represented as:

```
Binary : Set
Binary = Digit 2

零 : Binary
零 = zero

一 : Binary
一 = suc zero
```

0.1.3 Digit Assignment



Digits are assigned to ℕ together with the offset `o` of a system, ranging from o to $d + o$.

```
Digit-toℕ : ∀ {d} → Digit d → ℕ → ℕ
Digit-toℕ x o = toℕ x + o
```

[1]

However, not all natural numbers can be converted to digits. The value has to be in a certain range, between o and $d + o$. Values less than o are increased to o. Values greater than $d + o$ are prohibited by the supplied upper-bound.

```
Digit-fromℕ : ∀ {d}
    → (n o : ℕ)
    → (upper-bound : d + o ≥ n)
    → Digit (suc d)
Digit-fromℕ {d} n o upper-bound with n ∸ o ≤? d
Digit-fromℕ {d} n o upper-bound | yes p = fromℕ≤ (s≤s p)
Digit-fromℕ {d} n o upper-bound | no ¬p = contradiction p ¬p
    where   p : n ∸ o ≤ d
            p = start
                    n ∸ o
                ≤⟨ ∸n-mono o upper-bound ⟩
                    (d + o) ∸ o
                ≈⟨ m+n∸n≡m d o ⟩
                    d
                □
```

[2]

properties

$$\begin{tikzcd} \mathbb{N} \arrow[r, shift left, "\texttt{Digit-fromℕ}"] & \texttt{Digit d} \arrow[l, shift left, "\texttt{Digit-toℕ}"] \end{tikzcd}$$

---

[1] `toℕ : ∀ {n} → Fin n → ℕ`
converts from `Fin n` to ℕ.

[2] `fromℕ≤ : ∀ {m n} → m < n → Fin n`
converts from ℕ to `Fin n` given the number is small enough.

`Digit-fromℕ-toℕ` states that the value of a natural number should remain the same, after converting back and forth between `Digit` and ℕ.

```
Digit-fromℕ-toℕ : ∀ {d o}
    → (n : ℕ)
    → (lower-bound :     o ≤ n)
    → (upper-bound : d + o ≥ n)
    → Digit-toℕ (Digit-fromℕ {d} n o upper-bound) o ≡ n
Digit-fromℕ-toℕ {d} {o} n lb ub with n ∸ o ≤? d
Digit-fromℕ-toℕ {d} {o} n lb ub | yes q =
    begin
        toℕ (fromℕ≤ (s≤s q)) + o
    ≡⟨ cong (λ x → x + o) (toℕ-fromℕ≤ (s≤s q)) ⟩
        n ∸ o + o
    ≡⟨ m∸n+n≡m lb ⟩
        n
    ∎
Digit-fromℕ-toℕ {d} {o} n lb ub | no ¬q = contradiction q ¬q
    where   q : n ∸ o ≤ d
            q = +n-mono-inverse o (
                start
                    n ∸ o + o
                ≈⟨ m∸n+n≡m lb ⟩
                    n
                ≤⟨ ub ⟩
                    d + o
                □)
```

[3]

Digits have a upper-bound and a lower-bound after evaluation.

```
Digit-upper-bound : ∀ {d} → (o : ℕ) → (x : Digit d) → Digit-toℕ x o < d + o
Digit-upper-bound {d} o x = +n-mono o (bounded x)

Digit-lower-bound : ∀ {d} → (o : ℕ) → (x : Digit d) → Digit-toℕ x o ≥ o
Digit-lower-bound {d} o x = m≤n+m o (toℕ x)
```

[4]

---

[3] `toℕ-fromℕ≤ : ∀ {m n} (m<n : m < n) → toℕ (fromℕ≤ m<n) ≡ m`
states that a number should remain the same after converting back and forth.

[4] `bounded : ∀ {n} (i : Fin n) → toℕ i < n`
a property about the upper-bound of a `Fin n`.

0.1.4 Operations

Increment

To increment a digit, the digit must not be *the greatest*.

```
digit+1 : ∀ {d}
    → (x : Digit d)
    → (¬greatest : ¬ (Greatest x))
    → Fin d
digit+1 x ¬greatest =
    fromℕ≤ {suc (toℕ x)} (≤∧≢⇒< (bounded x) ¬greatest)
```

Where `≤∧≢⇒< (bounded x) ¬greatest : suc (toℕ x) < d`.

properties

$$
\begin{tikzcd}
\texttt{Digit d} \arrow[r, "\texttt{digit+1}"] \arrow[d, "\texttt{Digit-to}\mathbb{N}"'] & \texttt{Digit d} \arrow[d, "\texttt{Digit-to}\mathbb{N}"] \\
\mathbb{N} \arrow[r, "\texttt{suc}"'] & \mathbb{N}
\end{tikzcd}
$$

A digit taking these two routes should result in the same ℕ.

```
digit+1-toℕ : ∀ {d o}
    → (x : Digit d)
    → (¬greatest : ¬ (Greatest x))
    → Digit-toℕ (digit+1 x ¬greatest) o ≡ suc (Digit-toℕ x o)
digit+1-toℕ {d} {o} x ¬greatest =
    begin
        Digit-toℕ (digit+1 x ¬greatest) o
    ≡⟨ cong (λ w → w + o) (toℕ-fromℕ≤ (≤∧≢⇒< (bounded x) ¬greatest)) ⟩
        suc (Digit-toℕ x o)
    ∎
```

Increase then Subtract

Increases a digit and then subtract it by n. This function is useful for implementing *carrying*. When the digit to increase is already the greatest, we have to subtract it by an amount (usually the base) after the increment.

```
digit+1-n : ∀ {d}
    → (x : Digit d)
    → Greatest x
    → (n : ℕ)
    → n > 0
    → Digit d
digit+1-n x greatest n n>0 =
    fromℕ≤ (digit+1-n-lemma x greatest n n>0)
```

properties

$$
\begin{tikzcd}
\mathtt{Digit\ d} \arrow[rr, "\mathtt{digit{+}1{-}n}"] \arrow[d, "\mathtt{Digit{-}to}\mathbb{N}"'] & & \mathtt{Digit\ d} \arrow[d, "\mathtt{Digit{-}to}\mathbb{N}"] \\
\mathbb{N} \arrow[r, "\mathtt{suc}"'] & \mathbb{N} \arrow[r, "\dot{-}\mathtt{n}"'] & \mathbb{N}
\end{tikzcd}
$$

A digit taking these two routes should result in the same ℕ.

```
digit+1-n-toℕ : ∀ {d o}
    → (x : Digit d)
    → (greatest : Greatest x)
    → (n : ℕ)
    → (n>0 : n > 0)
    → n ≤ d
    → Digit-toℕ (digit+1-n x greatest n n>0) o ≡ suc (Digit-toℕ x o) ∸ n
digit+1-n-toℕ {zero}  {o} () greatest n n>0 n≤d
digit+1-n-toℕ {suc d} {o} x greatest n n>0  n≤d =
    begin
        toℕ (digit+1-n x greatest n n>0) + o
    ≡⟨ cong (λ w → w + o) (toℕ-fromℕ≤ (digit+1-n-lemma x greatest n n>0)) ⟩
        suc (toℕ x) ∸ n + o
    ≡⟨ +-comm (suc (toℕ x) ∸ n) o ⟩
        o + (suc (toℕ x) ∸ n)
    ≡⟨ sym (+-∸-assoc o {suc (toℕ x)} {n} (
        start
            n
        ≤⟨ n≤d ⟩
            suc d
        ≈⟨ sym greatest ⟩
            suc (toℕ x)
        □)
    ⟩
        (o + suc (toℕ x)) ∸ n
    ≡⟨ cong (λ w → w ∸ n) (+-comm o (suc (toℕ x))) ⟩
        suc (toℕ x) + o ∸ n
    ∎)
```

0.1.5 Special Digits

The Greatest Digit

constructions The greatest digit of a system is constructed by converting the index `d` to `Fin`.

```
greatest-digit : ∀ d → Digit (suc d)
greatest-digit d = fromℕ d
```

[5]

predicates Judging whether a digit is the greatest by converting it to ℕ. This predicate also comes with a decidable version.

```
Greatest : ∀ {d} (x : Digit d) → Set
Greatest {d} x = suc (toℕ x) ≡ d

Greatest? : ∀ {d} (x : Digit d) → Dec (Greatest x)
Greatest? {d} x = suc (toℕ x) ≟ d
```

properties Converting from the greatest digit to ℕ should result in `d + o`.

```
greatest-digit-toℕ : ∀ {d o}
    → (x : Digit (suc d))
    → Greatest x
    → Digit-toℕ x o ≡ d + o
greatest-digit-toℕ {d} {o} x greatest = cancel-suc (
    begin
        suc (Digit-toℕ x o)
    ≡⟨ refl ⟩
        suc (toℕ x) + o
    ≡⟨ cong (λ w → w + o) greatest ⟩
        suc d + o
    ∎)
```

A digit is the greatest if and only if it is greater than or equal to all other digits. This proposition is proven by induction on both of the compared digits.

```
greatest-of-all : ∀ {d} (o : ℕ) → (x y : Digit d)
    → Greatest x
```

[5] `fromℕ : ∀ {n} → Fin n → ℕ`
construct the greatest possible `Fin n` when given an index `n`.

```
    → Digit-toℕ x o ≥ Digit-toℕ y o
greatest-of-all o zero    zero     refl     = ≤-refl
greatest-of-all o zero    (suc ()) refl
greatest-of-all o (suc x) zero     greatest
    = +n-mono o {zero} {suc (toℕ x)} z≤n
greatest-of-all o (suc x) (suc y)  greatest
    = s≤s (greatest-of-all o x y (cancel-suc greatest))
```

The Carry

A carry is a digit that is transferred to a more significant digit to compensate the "loss" of the original digit.

constructions The carry is defined as the greater of these two values:

- the least digit of a system
- the digit that is assigned to 1

In case that that the least digit is assigned to 0, rendering the carry useless. Since the least digit is determined by the offset o, the value of the carry is defined as follows.

```
carry : ℕ → ℕ
carry o = 1 ⊔ o
```

And then we construct the carry by converting `carry o` to `Digit`:

```
carry-digit : ∀ d o → 2 ≤ suc d + o → Digit (suc d)
carry-digit d o proper =
    Digit-fromℕ
        (carry o)
        o
        (carry-upper-bound {d} proper)
```

properties The value of the carry should remain the same after converting back and forth.

```
carry-digit-toℕ : ∀ d o
    → (proper : 2 ≤ suc (d + o))
    → Digit-toℕ (carry-digit d o proper) o ≡ carry o
carry-digit-toℕ d o proper
    = Digit-fromℕ-toℕ
```

```
        (carry o)
        (m≤n⊔m o 1)
        (carry-upper-bound {d} proper)
```

The carry also have an upper-bound and a lower-bound, similar to that of `Digit`.

```
carry-lower-bound : ∀ {o} → carry o ≥ o
carry-lower-bound {o} = m≤n⊔m o 1

carry-upper-bound : ∀ {d o} → 2 ≤ suc d + o → carry o ≤ d + o
carry-upper-bound {d} {zero}  proper = ≤-pred proper
carry-upper-bound {d} {suc o} proper = n≤m+n d (suc o)
```