

10-1

## Free Types

10-2

### Data structures

We can model any data structure using sets, relations, or functions.

Where structure is important, and where different types are combined, a general mechanism is needed.

10-3

### Free types

The following definition introduces a new type  $T$  consisting of elements  $c_1, c_2, \dots, c_n$  and elements obtained by applying functions  $d_1, d_2, \dots, d_n$  to set expressions  $E_1, E_2, \dots, E_n$ :

$$T ::= c_1 \mid \dots \mid c_n \mid d_1 \langle E_1 \rangle \mid \dots \mid d_n \langle E_n \rangle$$

10-4

### Notes

- the elements  $c_1, c_2, \dots, c_n$  are called constants
- the functions  $d_1, d_2, \dots, d_n$  are called constructors
- the set expressions  $E_1, E_2, \dots, E_n$  may include instances of the type being defined

10-5

### Example

The following free type definition introduces a new type constructed using a single constant `zero` and a single constructor function `succ`:

```
nat ::= zero | succ(nat)
```

This type has a structure which is exactly that of the natural numbers (where zero corresponds to 0, and `succ` corresponds to the function  $+1$ ).

10-6

### Question

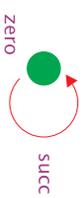
What does this mean? What would we have to do if we wanted to introduce the same set without a free type definition?

10-7

**Attempt 1**

zero : nat
succ : nat → nat
∀ n : nat • n = zero ∨ ∃ m : nat • n = succ m

10-8

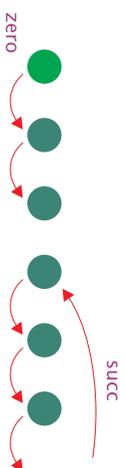


10-9

**Attempt 2**

zero : nat
succ : nat → nat
∀ n : nat • n = zero ∨ ∃ m : nat • n = succ m
{zero} ∩ ran succ = ∅

10-10

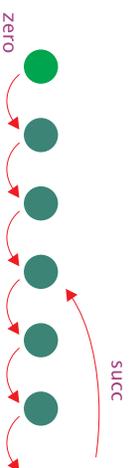


10-11

**Attempt 3**

zero : nat
succ : nat → nat
∀ n : nat • n = zero ∨ ∃ m : nat • n = succ m
{zero} ∩ ran succ = ∅

10-12

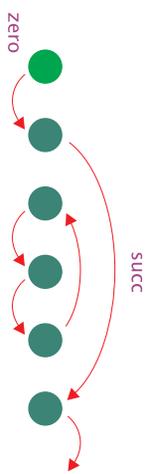


10-13

### Attempt 4

$zero : nat$ $succ : nat \rightarrow nat$
$\{zero\} \cap \text{ran } succ = \emptyset$ $\{zero\} \cup \text{ran } succ = nat$

10-14



10-15

### Conclusion

A free type definition involves:

- constants and constructed elements
- constructor functions
- closure

10-16

### Multiple constants

$Colours ::= red \mid orange \mid yellow \mid green \mid blue \mid$   
 $indigo \mid violet$

10-17

### Question

Is the free type definition on the previous slide equivalent to the following abbreviation?

$Colours ::=$   
 $\{red, orange, yellow, green, blue, indigo, violet\}$

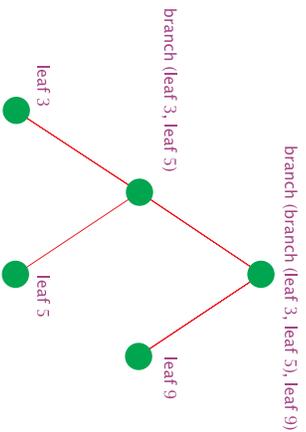
If not, why not?

10-18

### Multiple constructors

$Tree ::= leaf \langle \langle N \rangle \rangle \mid branch \langle \langle Tree \times Tree \rangle \rangle$

10-19



10-20

**Question**

What can we say about the functions leaf and branch?

10-21

**Example**

*Degree* ::= status ((0..3))

10-22

Useful names for elements of *Degree*:

<i>ba, msc, dphil, ma</i>	: <i>Degree</i>
<i>ba</i>	= status 0
<i>msc</i>	= status 1
<i>dphil</i>	= status 2
<i>ma</i>	= status 3

10-23

The structure is preserved:

$\leq_{status}$	: <i>Degree</i> ↔ <i>Degree</i>
$\forall d_1, d_2$	: <i>Degree</i> •
$d_1 \leq_{status} d_2$	↔ status~ $d_1 \leq$ status~ $d_2$

10-24

**Induction principle**

A recursive free type definition gives rise to a corresponding induction principle.

10-25

The free type definition

$$T ::= c_1 \mid \dots \mid c_m \mid d_1 \langle\langle E_1 \rangle\rangle \mid \dots \mid d_n \langle\langle E_n \rangle\rangle$$

has the same effect as a basic type definition

[T]

followed by...

10-26

$$\frac{\begin{array}{l} c_1 : T \\ \vdots \\ c_m : T \\ d_1 : E_1 \rightarrow T \\ \vdots \\ d_n : E_n \rightarrow T \\ \text{disjoint}(\{c_1, \dots, c_m\}, \text{ran } d_1, \dots, \text{ran } d_n) \\ \forall S : \mathbb{P} T \bullet \\ \{c_1, \dots, c_m\} \subseteq S \wedge \\ d_1 \langle\langle E_1[S/T] \rangle\rangle \cup \dots \cup d_n \langle\langle E_n[S/T] \rangle\rangle \subseteq S \Rightarrow \\ S = T \end{array}}{S = T}$$

10-27

### Closure rule

$$\frac{\begin{array}{l} S \subseteq T \\ \{c_1, \dots, c_m\} \subseteq S \\ (d_1 \langle\langle E_1[S/T] \rangle\rangle \cup \dots \cup d_n \langle\langle E_n[S/T] \rangle\rangle) \subseteq S \end{array}}{S = T}$$

10-28

### Inverse image

$$\begin{aligned} d_1 \langle\langle E_1[S/T] \rangle\rangle \subseteq S &\Leftrightarrow E_1[S/T] \subseteq d_1^{-1} \langle\langle S \rangle\rangle \\ &\Leftrightarrow \forall e : E_1[S/T] \bullet e \in d_1^{-1} \langle\langle S \rangle\rangle \\ &\Leftrightarrow \forall e : E_1[S/T] \bullet d_1 e \in S \end{aligned}$$

10-29

### Predicates

S may be the characteristic set of some property P:

$$S == \{t : T \mid P t\}$$

10-30

### Induction principle

$$\frac{\begin{array}{l} P c_1 \\ \vdots \\ P c_m \\ \forall e : E_1[S/T] \bullet P (d_1 e) \\ \vdots \\ \forall e : E_n[S/T] \bullet P (d_n e) \end{array}}{\forall t : T \bullet P t}$$

10-31

**Example**

$$S \subseteq \text{nat} \quad (\{\text{zero}\} \cup \text{succ}(\text{nat}[S / \text{nat}]\emptyset) \subseteq S$$

$$S = \text{nat}$$

10-32

**Alternative form**

$$S = \{n : \text{nat} \mid P n\}$$

$$P \text{ zero}$$

$$\forall m : \text{nat} \bullet P m \Rightarrow P (\text{succ } m)$$

$$\forall n : \text{nat} \bullet P n$$

10-33

**Question**

Can you suggest a suitable induction principle for *Tree*?

10-34

**Consistency**

- not all constructions make sense: some result in a free type with no elements
- Cartesian products, finite sequences, finite functions, and finite power sets are guaranteed to work

10-35

**Example**

The following type definition is inconsistent:

$$P ::= \text{power}(\langle P P \rangle)$$

10-36

**Summary**

- data structures
- free type definitions
- constants, constructors, and closure
- induction principle
- consistency