Free Types

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.

Free types

The following definition introduces a new type T consisting of elements $\mathtt{c}_1, \mathtt{c}_2, .., \mathtt{c}_n$ and elements obtained by applying functions $\mathtt{d}_1, \mathtt{d}_2, .., \mathtt{d}_n$ to set expressions $E_1, E_2, .., E_n$:

$$T ::= \mathtt{c}_1 \mid \ldots \mid \mathtt{c}_m \mid \mathtt{d}_1 \langle\langle E_1 \rangle\rangle \mid \ldots \mid \mathtt{d}_\mathtt{n} \langle\langle E_n \rangle\rangle$$

Notes

- the elements $\mathtt{c}_1, \mathtt{c}_2, .., \mathtt{c}_n$ are called constants
- the functions $\mathtt{d}_1, \mathtt{d}_2, .., \mathtt{d}_n$ are called constructors
- the set expressions $E_1, E_2, .., E_n$ may include instances of the type being defined

Example

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

$$nat ::= \mathsf{zero} \mid \mathsf{succ}\langle\langle nat \rangle\rangle$$

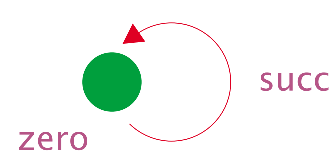
This type has a structure which is exactly that of the natural numbers (where zero corresponds to 0, and succ corresponds to the function $+1$).

Question

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

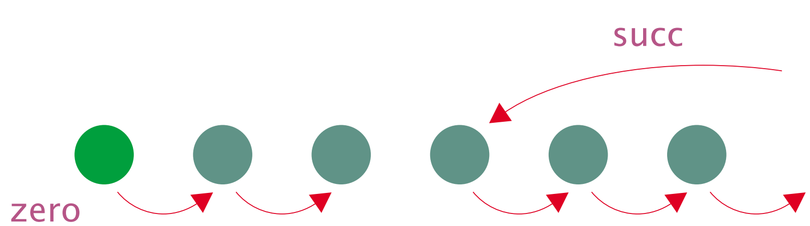
Attempt 1

$$\begin{array}{|l}
\mathsf{zero} : \mathit{nat} \\
\mathsf{succ} : \mathit{nat} \rightarrowtail \mathit{nat} \\
\hline
\forall\, n : \mathit{nat} \bullet n = \mathsf{zero} \lor \exists\, m : \mathit{nat} \bullet n = \mathsf{succ}\, m
\end{array}$$



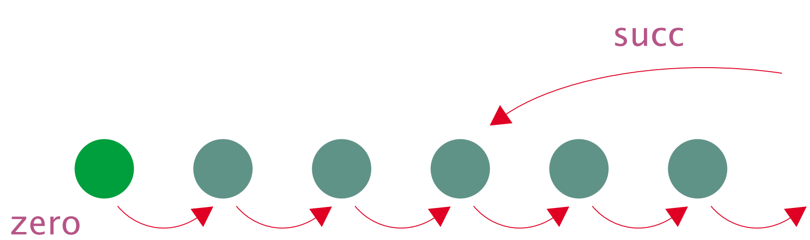
Attempt 2

$$
\begin{array}{|l}
\mathsf{zero} : \mathit{nat} \\
\mathsf{succ} : \mathit{nat} \rightarrowtail \mathit{nat} \\
\hline
\forall\, n : \mathit{nat} \bullet n = \mathsf{zero} \lor \exists\, m : \mathit{nat} \bullet n = \mathsf{succ}\, m \\
\{\mathsf{zero}\} \cap \operatorname{ran} \mathsf{succ} = \varnothing
\end{array}
$$



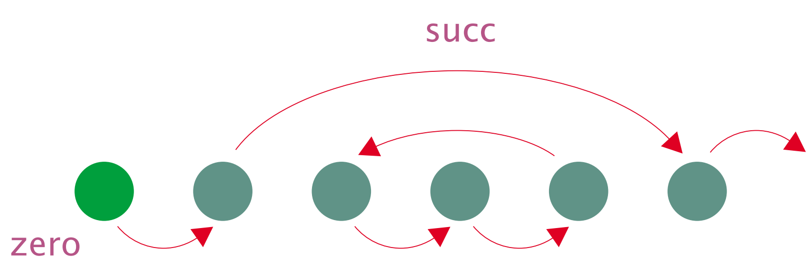
Attempt 3

$$
\begin{array}{|l}
\mathsf{zero} : \mathit{nat} \\
\mathsf{succ} : \mathit{nat} \rightarrow \mathit{nat} \\
\hline
\forall\, n : \mathit{nat} \bullet n = \mathsf{zero} \lor \exists\, m : \mathit{nat} \bullet n = \mathsf{succ}\, m \\
\{\mathsf{zero}\} \cap \operatorname{ran} \mathsf{succ} = \varnothing
\end{array}
$$



Attempt 4

$$
\begin{array}{|l}
\mathsf{zero} : \mathit{nat} \\
\mathsf{succ} : \mathit{nat} \rightarrowtail \mathit{nat} \\
\hline
\{\mathsf{zero}\} \cap \operatorname{ran} \mathsf{succ} = \varnothing \\
\{\mathsf{zero}\} \cup \operatorname{ran} \mathsf{succ} = \mathit{nat}
\end{array}
$$



Conclusion

A free type definition involves:

- constants and constructed elements
- constructor functions
- closure

Multiple constants

$Colours$ ::= red | orange | yellow | green | blue |
indigo | violet

Question

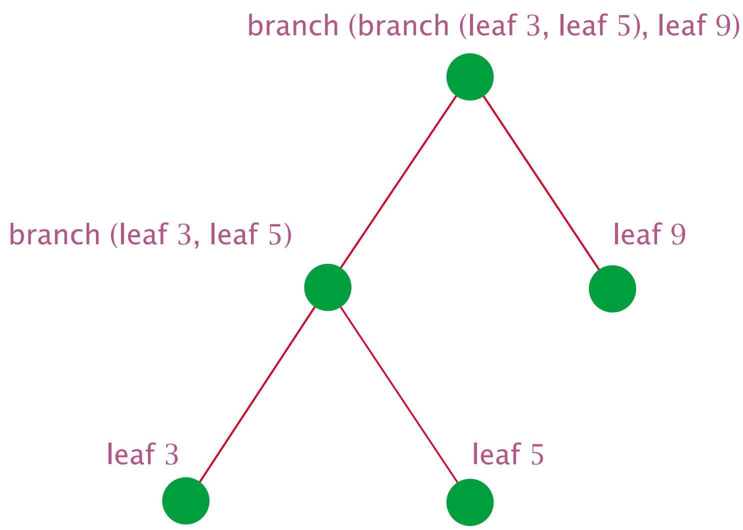
Is the free type definition on the previous slide equivalent to the following abbreviation?

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

If not, why not?

Multiple constructors

$$\mathit{Tree} ::= \mathsf{leaf}\, \langle\!\langle \mathbb{N} \rangle\!\rangle \mid \mathsf{branch}\, \langle\!\langle \mathit{Tree} \times \mathit{Tree} \rangle\!\rangle$$



Question

What can we say about the functions leaf and branch?

Example

$$Degree ::= \mathsf{status}\ \langle\!\langle 0 \mathinner{..} 3 \rangle\!\rangle$$

Useful names for elements of *Degree*:

$$\begin{array}{|l}
ba, msc, dphil, ma : Degree \\
\hline
ba \quad = \mathsf{status}\ 0 \\
msc \quad = \mathsf{status}\ 1 \\
dphil = \mathsf{status}\ 2 \\
ma \quad = \mathsf{status}\ 3
\end{array}$$

The structure is preserved:

$$\begin{array}{|l} \leq_{status} : Degree \leftrightarrow Degree \\ \hline \forall\, d_1, d_2 : Degree \bullet \\ \qquad d_1 \leq_{status} d_2 \Leftrightarrow \mathsf{status}^{\sim}\, d_1 \leq \mathsf{status}^{\sim}\, d_2 \end{array}$$

Induction principle

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

The free type definition

$$T ::= \mathsf{c}_1 \mid \ldots \mid \mathsf{c}_m \mid \mathsf{d}_1 \langle\langle E_1 \rangle\rangle \mid \ldots \mid \mathsf{d}_n \langle\langle E_n \rangle\rangle$$

has the same effect as a basic type definition

$$[T]$$

followed by…

$$
\begin{array}{|l}
\mathsf{c}_1 : T \\
\vdots \\
\mathsf{c}_m : T \\
\mathsf{d}_1 : E_1 \rightarrowtail T \\
\vdots \\
\mathsf{d}_n : E_n \rightarrowtail T \\
\hline
\mathsf{disjoint}\ \langle \{\mathsf{c}_1\}, \ldots, \{\mathsf{c}_m\}, \operatorname{ran} \mathsf{d}_1, \ldots \operatorname{ran} \mathsf{d}_n \rangle \\
\forall\, S : \mathbb{P}\, T \bullet \\
\quad \{\mathsf{c}_1, \ldots, \mathsf{c}_m\} \subseteq S \land \\
\qquad \mathsf{d}_1 (\!|\, E_1[S / T] \,|\!) \cup \ldots \cup \mathsf{d}_n (\!|\, E_n[S / T] \,|\!) \subseteq S \Rightarrow \\
\qquad\quad S = T
\end{array}
$$

Closure rule

$$\frac{S \subseteq T \qquad \{\mathsf{c}_1, \ldots, \mathsf{c}_m\} \subseteq S \qquad (\mathsf{d}_1 \llparenthesis\, E_1[S\,/\,T] \,\rrparenthesis \cup \ldots \cup \mathsf{d}_n \llparenthesis\, E_n[S\,/\,T] \,\rrparenthesis) \subseteq S}{S = T}$$

Inverse image

$$\begin{aligned}
&\mathsf{d_i} \llparenthesis\, E_i[S\,/\,T] \,\rrparenthesis \subseteq S \\
&\quad \Leftrightarrow E_i[S\,/\,T] \subseteq \mathsf{d_i}^{\sim} \llparenthesis\, S \,\rrparenthesis \\
&\quad \Leftrightarrow \forall\, e : E_i[S\,/\,T] \bullet e \in \mathsf{d_i}^{\sim} \llparenthesis\, S \,\rrparenthesis \\
&\quad \Leftrightarrow \forall\, e : E_i[S\,/\,T] \bullet \mathsf{d_i}\, e \in S
\end{aligned}$$

Predicates

S may be the characteristic set of some property P:

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

Induction principle

$$\frac{\begin{array}{l} P\,\mathsf{c}_1 \\ \vdots \\ P\,\mathsf{c}_m \\ \forall\, e : E_1[S\,/\,T] \bullet P\,(\mathsf{d}_1\, e) \\ \vdots \\ \forall\, e : E_n[S\,/\,T] \bullet P\,(\mathsf{d}_n\, e) \end{array}}{\forall\, t : T \bullet P\, t}$$

Example

$$\frac{S \subseteq nat \quad (\{\mathsf{zero}\} \cup \mathsf{succ}(\!|\, nat[S / nat] \,|\!)) \subseteq S}{S = nat}$$

Alternative form

$$S \;==\; \{n : nat \mid P\,n\}$$

$$\frac{\begin{array}{l} P\,\mathsf{zero} \\ \forall\, m : nat \bullet P\,m \Rightarrow P\,(\mathsf{succ}\,m) \end{array}}{\forall\, n : nat \bullet P\,n}$$

Question

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

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

Example

The following type definition is inconsistent:

$$P ::= \mathsf{power}\langle\!\langle \mathbb{P}\, P \rangle\!\rangle$$

Summary

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