Beyond System T:

Adding other data types

There are two kinds of types

Inductive Types

- Well-founded structure
- Elements constructed from bottom up
- Examples: Nat, Binary Trees

CoInductive Types

- Non-well-founded structure
- Elements generated from top down

Example of CoInductive Type

Streams of bits

sequences of 0s and 1s that go on forever

Introduction Rules

It's not enough to give just the constructors, as for inductive types:

$$\frac{s : \mathrm{BitStream}}{0 \triangleleft s : \mathrm{BitStream}} \qquad \frac{s : \mathrm{BitStream}}{1 \triangleleft s : \mathrm{BitStream}}$$

There is no base case: how do we start building a stream?

We need a rule to generate streams dynamically

Introduction Rule expressing
CoRecursion

For any type X $(\mathrm{Bit} = \{0,1\})$

$$\frac{f : X \to \mathrm{Bit} \qquad t : X \to X}{\mathrm{corec}\ f\ t : X \to \mathrm{BitStream}}$$

Elimination Rules (Observation)

$$\frac{s : \mathrm{BitStream}}{\mathrm{bit}\ s : \mathrm{Bit}} \qquad \frac{s : \mathrm{BitStream}}{\mathrm{next}\ s : \mathrm{BitStream}}$$

Reduction Rules:

$\mathrm{bit}\ (\mathrm{corec}\ f\ t\ x) \rightsquigarrow f\ x$

$\mathrm{next}\ (\mathrm{corec}\ f\ t\ x)$
$\rightsquigarrow \mathrm{corec}\ f\ t\ (t\ x)$

Examples:

$\mathrm{alternate} : \mathrm{Bit} \to \mathrm{BitStream}$

$\mathrm{alternate} = \mathrm{corec}\ \mathrm{id}\ \mathit{flip}$

where
id is the identity function on Bit

$\mathit{flip} : \mathrm{Bit} \to \mathrm{Bit}$

$\mathit{flip}\ 0 = 1$

$\mathit{flip}\ 1 = 0$

intuitively

$\mathrm{alternate}\ 0 = 0 \triangleleft 1 \triangleleft 0 \triangleleft 1 \triangleleft \cdots$

$\mathrm{alternate}\ 1 = 1 \triangleleft 0 \triangleleft 1 \triangleleft 0 \triangleleft \cdots$

Imagine corecursion as a dynamic process:

$f : X \to \mathrm{Bit}$

X: a set of "process states"

f: function producing a bit in the current state

$t : X \longrightarrow X$

transition function: after producing a bit, the state of the process changes from x to $(t\,x)$

$(\mathrm{corec}\ f\ t\ x)$ is the stream of elements generated in output by the process

$$
\begin{tikzcd}
x \arrow[r, bend left, "t"] \arrow[d, "f"'] & x_1 \arrow[r, bend left, "t"] \arrow[d, "f"'] & x_2 \arrow[r, bend left, "t"] \arrow[d, "f"'] & x_3 \arrow[r, bend left, "t"] \arrow[d, "f"'] & x_4 & \cdots \\
b_0 & b_1 & b_2 & b_3 & & \cdots
\end{tikzcd}
$$

$$\underbrace{b_0 \quad b_1 \quad b_2 \quad b_3 \quad \cdots}_{(\mathrm{corec}\ f\ t\ x)}$$

$$(\mathrm{corec}\ f\ t\ x) = b_0 \triangleleft b_1 \triangleleft b_2 \triangleleft b_3 \triangleleft \cdots$$

The pair of functions $\langle f, t \rangle$ is called a coalgebra

We'll see a general theory of coalgeras in a later lecture

Example:

Stream of n-repetitions of 1 separated by 0:

$$0\,1^0\,0\,1^1\,0\,1^2\,0\,1^3\,0\,1^4\,0 \cdots$$

$$= 0010110111011110\cdots$$

We use as states pairs of numbers

Intuitively a state $\langle n, m\rangle$ generate the stream

$$1^m\,0\,1^n\,0\,1^{n+1}\,0\,1^{n+2} \cdots$$

The whole stream we want is generated by $\langle 0, 0\rangle$

$X = \mathrm{Nat} \times \mathrm{Nat}$

(We'll define product types next lecture)

$f : X \longrightarrow \mathrm{Bit}$

$f\,\langle n, 0\rangle = 0$

$f\,\langle n, Sm\rangle = 1$

$t : X \longrightarrow X$

$t\,\langle n, 0\rangle = \langle n+1, n\rangle$

$t\,\langle n, Sm\rangle = \langle n, m\rangle$

The stream we want is

$\mathrm{corec}\ f\ t\ \langle 0, 0\rangle$

If we see it as a generating process:

$$
\begin{tikzcd}
\langle 0,0\rangle \arrow[r, bend left, "t"] \arrow[d, "f"'] & \langle 1,0\rangle \arrow[r, bend left, "t"] \arrow[d, "f"'] & \langle 2,1\rangle \arrow[r, bend left, "t"] \arrow[d, "f"'] & \langle 2,0\rangle \arrow[r, bend left, "t"] \arrow[d, "f"'] & \langle 3,2\rangle \arrow[r, bend left, "t"] \arrow[d, "f"'] & \langle 3,1\rangle \arrow[r, bend left, "t"] \arrow[d, "f"'] & \langle 3,0\rangle \arrow[r, bend left, "t"] \arrow[d, "f"'] & {} \\
0 & 0 & 1 & 0 & 1 & 1 & 0 &
\end{tikzcd}
$$

$$
\begin{tikzcd}
{} \arrow[r, "t"] & \langle 4,3\rangle \arrow[r, bend left, "t"] \arrow[d, "f"'] & \langle 4,2\rangle \arrow[r, bend left, "t"] \arrow[d, "f"'] & \langle 4,1\rangle \arrow[r, bend left, "t"] \arrow[d, "f"'] & \langle 4,0\rangle \arrow[r, bend left, "t"] \arrow[d, "f"'] & \langle 5,4\rangle \arrow[r, bend left] \arrow[d, "f"'] & \cdots \\
& 1 & 1 & 1 & 0 & 1 &
\end{tikzcd}
$$

Very often, when we want to define a stream, we need to find a general type X of states that generates all the intermediate streams.