Suppose you want to write an interpreter for the simply typed lambda calculus.

How would you represent expressions using the tools we have studied so far?

We could use a variant type:

\[
\text{Expr} = \langle \text{var}: \text{string}, \\
\text{lam}: \text{string} \times \text{Expr}, \\
\text{app}: \text{Expr} \times \text{Expr} \rangle
\]

but note that the type we are defining, Expr, appears within the definition of Expr. It is circular!
Recursive Types

Syntax:

\[ T ::= \ldots \text{types} \]
\[ A \quad \text{type variables} \]
\[ \mu A. \ T \quad \text{recursive types} \]

Example:

\[ \text{Expr} = \mu A. \langle \text{var: string,} \]
\[ \quad \text{lam: string} \times A, \]
\[ \quad \text{app: A} \times A \rangle \]
Equi-recursive and iso-recursive types

A recursive type can be **unfolded** by replacing it with its body, substituting the bound variable for the type itself.

\[ \mu A. \ T \implies [A \mapsto \mu A. \ T]T \]

For example:

\[
\text{Expr} = \mu A. \langle \text{var: string, lam: string } \times A, \text{ app: A } \times A \rangle \\
\text{Expr} \implies \langle \text{var: string, lam: string } \times \text{Expr, app: Expr } \times \text{Expr} \rangle
\]

There are two main approaches to integrating recursive types into the type system of a language.

- **Equi-recursive types**: the type and its unfolding are considered equal.
- **Iso-recursive types**: there are unfold and fold operators that convert back and forth between the recursive type and its unfolding.
Iso-recursive Types

Syntax:
\[
e ::= \ldots | \text{fold}[T]e | \text{unfold}[T]e
\]
\[
\nu ::= \ldots | \text{fold}[T]\nu
\]

Evaluation contexts:
\[
E ::= \ldots | \text{fold}[T]E
\]

Reduction rules:
\[
\text{unfold}[S](\text{fold}[T]\nu) \rightarrow \nu
\]

Typing rules:
\[
\frac{U = \mu X. T \quad \Gamma \vdash e : [X \mapsto U]T}{\Gamma \vdash \text{fold}[U]e : U} \quad \text{(Fld)}
\]
\[
\frac{U = \mu X. T \quad \Gamma \vdash e : U}{\Gamma \vdash \text{unfold}[U]e : [X \mapsto U]T} \quad \text{(Unfld)}
\]
The type system for equi-recursive types is more complicated. Also, without too much extra complication, we’ll consider subtyping over equi-recursive types. We’ll need to learn some theory and techniques to handle equi-recursive types. In particular, we’ll need to learn about coinduction, which is like induction but works for recursive structures.
Induction and Coinduction

Suppose we have the set \{a, b, c\}. The power set of \{a, b, c\}, written \(\mathcal{P}(\{a, b, c\})\) is the set of all subsets of \{a, b, c\}:

\[
\mathcal{P}(\{a, b, c\}) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}
\]

We’ll be concerned with functions that map sets to sets, i.e., whose domain and codomain are power sets.

F is **monotone** if \(X \subseteq Y\) implies \(F(X) \subseteq F(Y)\). Example:

\[
\begin{align*}
F(\emptyset) & = F(\{c\}) & F(\{a, b\}) & = F(\{c\}) \\
F(\{a\}) & = F(\{c\}) & F(\{a, c\}) & = F(\{b, c\}) \\
F(\{b\}) & = F(\{c\}) & F(\{b, c\}) & = F(\{a, b, c\}) \\
F(\{c\}) & = F(\{b, c\}) & F(\{a, b, c\}) & = F(\{a, b, c\})
\end{align*}
\]

F can be compactly represented by inference rules:

\[
\begin{array}{c}
c \in F(X) \\
\hline
C \in X \\
\hline
b \in F(X) \\
\hline
b \in X \quad c \in X \\
\hline
a \in F(X)
\end{array}
\]
A set $X$ is closed with respect to function $F$, it is **F-closed**, if $F(X) \subseteq X$.

- A set $X$ is **F-consistent** if $X \subseteq F(X)$.
- A set $X$ is the **fixed point** of $F$ is $F(X) = X$.
- Find some sets in $\mathcal{P}(\{a, b, c\})$ that are F-closed, F-consistent, and are fixed points of $F$.

**Theorem (Knaster-Tarski)**

1. The intersection of all F-closed sets is the least fixed point of $F$.
2. The union of all F-consistent sets is the greatest fixed point of $F$.

We refer to $F$ as the **generating function** for the least and greatest fixed points.
Corollary

1. **Principle of induction**: If $X$ is $F$-closed, then the least fixed point of $F$ is a subset of $X$.

2. **Principle of coinduction**: If $X$ is $F$-consistent, then $X$ is a subset of the greatest fixed point of $F$.

What?!?
Principle of induction: If $X$ is $F$-closed, then the least fixed point of $F$ is a subset of $X$.

To prove $P(x)$ for all $x \in \mathbb{N}$, show:

1. $P(0)$
2. $P(n) \implies P(n + 1)$

Correspondence: $X$ is $P$, $F$ is represented by the inference rules:

\[
\begin{align*}
0 & \in F(X) \\
n & \in X \\
n + 1 & \in F(X)
\end{align*}
\]

and the least fixed point of $F$ is $\mathbb{N}$. Saying that $\mathbb{N} \subseteq P$ is the same as saying that $x \in \mathbb{N}$ implies $x \in P$.

Recall that $F$-closed means $F(P) \subseteq P$. So the principle of induction says that you need to show that applying $F$ to stuff in $P$ results in stuff that is also in $P$. 

Recall that an inductively defined set is the least fixed point of some inference rules. A **coinductively defined set** is the greatest fixed point of some rules.

**Principle of coinduction:** If $X$ is $F$-consistent, then $X$ is a subset of the greatest fixed point of $F$ ($\nu F$).

In other words: if you want to show that some element $x$ is in a coinductively defined set ($\nu F$), then find some set $X$ such that $x \in X$ and $X$ is $F$-consistent (i.e. $X \subseteq F(X)$, or, everything in the input shows up in the output).
Subtyping rules (defining the generating function S):

\[
\begin{align*}
(T, \text{Top}) & \in S(X) \\
(T_1 \times T_2, T_3 \times T_4) & \in S(X) \\
(T_1, T_3) & \in X \\
(T_2, T_4) & \in X \\
(T_1 \rightarrow T_2, T_3 \rightarrow T_4) & \in S(X) \\
((T_1, [A \mapsto \mu A. T_2] T_2) & \in X \\
(T_1, \mu A. T_2) & \in S(X) \\
([A \mapsto \mu A. T_1] T_1, T_2) & \in X \\
(\mu A. T_1, T_2) & \in S(X)
\end{align*}
\]

The **subtyping relation** (\(<:\)) is the greatest fixed point of S.

For some given \(T_1\) and \(T_2\), how do we know if \(T_1 <: T_2\)?

The coinduction principle says: find some relation \(R\) on types such that \((T_1, T_2) \in R\) and \(R\) is S-consistent (i.e., \(R \subseteq S(R)\)).
The main idea behind finding this set $R$ is that we start off with $R_0 = \{(T_1, T_2)\}$ and then run $S$ backwards so that $R_{i+1} = R_i \cup S^{-1}(R_i)$ until $S^{-1}(R_j) \subseteq R_j$ for some $j$. We then let $R = R_j$, and we know that $R$ is $S$-consistent by the following reasoning.

\[
R^{-1}(R) \subseteq R \\
S(R^{-1}(R)) \subseteq S(R) \quad \text{by monotonicity} \\
R \subseteq S(R) \quad \text{by definition of inverse}
\]

Suppose $T_1 = \text{int} \to \text{int}$ and $T_2 = \text{int} \to \text{Top}$.

\[
R_0 = \{(\text{int} \to \text{int}, \text{int} \to \text{Top})\} \\
R_1 = \{(\text{int} \to \text{int}, \text{int} \to \text{Top}), (\text{int}, \text{int}), (\text{int}, \text{Top})\} \\
R_2 = R_1
\]
Subtyping Via Coinduction

Suppose $T_1 = \mu X. \text{int} \to X \times \text{int}$ and $T_2 = \mu Y. \text{int} \to Y \times \text{Top}.$

\[ R_0 = \{ (\mu X. \text{int} \to X \times \text{int}, \mu Y. \text{int} \to Y \times \text{Top}) \} \]
\[ R_1 = \{ (\mu X. \text{int} \to X \times \text{int}, \mu Y. \text{int} \to Y \times \text{Top}), \]
\[ (\text{int} \to (\mu X. \ldots) \times \text{int}, \mu Y. \text{int} \to Y \times \text{Top}) \} \]
\[ R_2 = \{ (\mu X. \text{int} \to X \times \text{int}, \mu Y. \text{int} \to Y \times \text{Top}), \]
\[ (\text{int} \to (\mu X. \ldots) \times \text{int}, \mu Y. \text{int} \to Y \times \text{Top}), \]
\[ (\text{int} \to (\mu X. \ldots) \times \text{int}, \text{int} \to (\mu Y. \ldots) \times \text{Top}) \} \]
\[ R_3 = \{ (\mu X. \text{int} \to X \times \text{int}, \mu Y. \text{int} \to Y \times \text{Top}), \]
\[ (\text{int} \to (\mu X. \ldots) \times \text{int}, \mu Y. \text{int} \to Y \times \text{Top}), \]
\[ (\text{int} \to (\mu X. \ldots) \times \text{int}, \text{int} \to (\mu Y. \ldots) \times \text{Top}), \]
\[ (\text{int}, \text{int}), (\text{int}, \text{Top}) \} \]
\[ R_4 = R_3 \]