Recursive Types

- 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:

$$\begin{aligned}\text{Expr} = <&\text{var: string},\\ &\text{lam: string} \times \text{Expr},\\ &\text{app: Expr} \times \text{Expr}>\end{aligned}$$

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

Recursive Types

Syntax:

$$
\begin{array}{rcll}
T & ::= & \ldots & \text{types} \\
 & & A & \text{type variables} \\
 & & \mu A.\ T & \text{recursive types}
\end{array}
$$

Example:

$$
\begin{array}{l}
\text{Expr} = \mu\ \text{A.}\ \langle \text{var: string}, \\
\qquad\qquad\quad \text{lam: string} \times \text{A}, \\
\qquad\qquad\quad \text{app: A} \times \text{A} \rangle
\end{array}
$$

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.

$$\mu A.\ T \Longrightarrow [A \mapsto \mu A.\ T]\,T$$

For example:

$$\text{Expr} = \mu\ \text{A.}\ \langle \text{var: string,lam: string} \times \text{A, app: A} \times \text{A} \rangle$$
$$\text{Expr} \Longrightarrow \langle \text{var: string,lam: string} \times \text{Expr, app: Expr} \times \text{Expr} \rangle$$

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

$$\begin{array}{lcl} e & ::= & \ldots \mid \mathit{fold}[T]e \mid \mathit{unfold}[T]e \\ v & ::= & \ldots \mid \mathit{fold}[T]v \end{array}$$

Evaluation contexts:

$$E ::= \ldots \mid \mathit{fold}[T]E$$

Reduction rules:

$$\mathit{unfold}[S](\mathit{fold}[T]v) \longrightarrow v$$

Typing rules:

$$(\text{Fld})\frac{U = \mu X.\ T \qquad \Gamma \vdash e : [X \mapsto U]T}{\Gamma \vdash \mathit{fold}[U]e : U}$$

$$(\text{Unfld})\frac{U = \mu X.\ T \qquad \Gamma \vdash e : U}{\Gamma \vdash \mathit{unfold}[U]e : [X \mapsto U]T}$$

Equi-recursive Types

- The type system for equi-recursive types is more complicated.
- Also, without to 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:

$$\begin{array}{lcl|lcl} F(\emptyset) & = & F(\{c\}) & F(\{a, b\}) & = & F(\{c\}) \\ F(\{a\}) & = & F(\{c\}) & F(\{a, c\}) & = & F(\{b, c\}) \\ F(\{b\}) & = & F(\{c\}) & F(\{b, c\}) & = & F(\{a, b, c\}) \\ F(\{c\}) & = & F(\{b, c\}) & F(\{a, b, c\}) & = & F(\{a, b, c\}) \end{array}$$

- F can be compactly represented by inference rules:

$$\frac{}{c \in F(X)} \qquad \frac{c \in X}{b \in F(X)} \qquad \frac{b \in X \quad c \in X}{a \in F(X)}$$

Induction and Coinduction

- 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.

Induction and Coinduction

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?!?

Induction

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. $P(n) \Longrightarrow P(n+1)$

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

$$\frac{}{0 \in F(X)} \qquad \frac{n \in X}{n+1 \in F(X)}$$

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.

Coinduction

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 (νF).

In other words: if you want to show that some element x is in a coinductively defined set (νF), 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

- Subtyping rules (defining the generating function S):

$$\frac{}{(T, Top) \in S(X)} \qquad \frac{(T_1, T_3) \in X \quad (T_2, T_4) \in X}{(T_1 \times T_2, T_3 \times T_4) \in S(X)}$$

$$\frac{(T_3, T_1) \in X \quad (T_2, T_4) \in X}{(T_1 \to T_2, T_3 \to T_4) \in S(X)}$$

$$\frac{((T_1, [A \mapsto \mu A.\ T_2]T_2) \in X}{(T_1, \mu A.\ T_2) \in S(X)} \qquad \frac{([A \mapsto \mu A.\ T_1]T_1, T_2) \in X}{(\mu A.\ T_1, T_2) \in S(X)}$$

- 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., $R \subseteq S(R)$).

Subtyping Via Coinduction

- 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.

$$
\begin{aligned}
S^{-1}(R) &\subseteq R \\
S(S^{-1}(R)) &\subseteq S(R) && \text{by monotonicity} \\
R &\subseteq S(R) && \text{by definition of inverse}
\end{aligned}
$$

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

$$
\begin{aligned}
R_0 &= \{(\mathit{int} \to \mathit{int}, \mathit{int} \to \mathit{Top})\} \\
R_1 &= \{(\mathit{int} \to \mathit{int}, \mathit{int} \to \mathit{Top}), (\mathit{int}, \mathit{int}), (\mathit{int}, \mathit{Top})\} \\
R_2 &= R_1
\end{aligned}
$$

Subtyping Via Coinduction

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

$$
\begin{aligned}
R_0 &= \{(\mu X.\ int \to X \times int, \mu Y.\ int \to Y \times Top)\} \\
R_1 &= \{(\mu X.\ int \to X \times int, \mu Y.\ int \to Y \times Top), \\
&\qquad (int \to (\mu X.\ \ldots) \times int, \mu Y.\ int \to Y \times Top)\} \\
R_2 &= \{(\mu X.\ int \to X \times int, \mu Y.\ int \to Y \times Top), \\
&\qquad (int \to (\mu X.\ \ldots) \times int, \mu Y.\ int \to Y \times Top), \\
&\qquad (int \to (\mu X.\ \ldots) \times int, int \to (\mu Y.\ \ldots) \times Top)\} \\
R_3 &= \{(\mu X.\ int \to X \times int, \mu Y.\ int \to Y \times Top), \\
&\qquad (int \to (\mu X.\ \ldots) \times int, \mu Y.\ int \to Y \times Top), \\
&\qquad (int \to (\mu X.\ \ldots) \times int, int \to (\mu Y.\ \ldots) \times Top), \\
&\qquad (int, int), (int, Top)\} \\
R_4 &= R_3
\end{aligned}
$$