Chapter 21: Metatheory of Recursive Types

Induction and Coinduction

Finite and Infinite Types/Subtyping

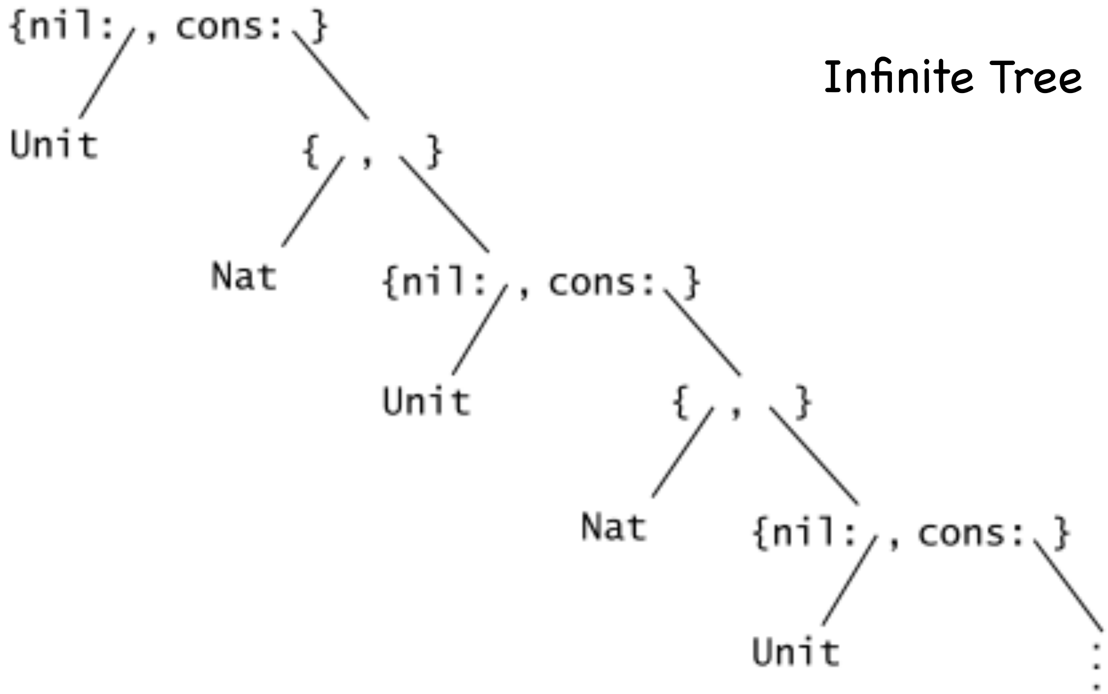
Membership Checking

Review of Chapter 20

Recursive Types

- **Lists**

  NatList = <nil:Unit, cons:{Nat, NatList}>



$$NatList = \mu X.\ \langle nil{:}Unit,\ cons{:}\{Nat, X\}\rangle$$

This means that let NatList be the infinite type satisfying the equation:

$$X = \langle nil{:}Unit,\ cons{:}\{Nat, X\}\rangle.$$

- **Hungry Functions**: accepting any number of numeric arguments and always return a new function that is hungry for more

$$\text{Hungry} = \mu A.\ \text{Nat} \rightarrow A$$

- **Streams**: consuming an arbitrary number of unit values, each time return- ing a pair of a number and a new stream

$$Stream = \mu A.\ Unit \rightarrow \{Nat, A\};$$

$$(Process = \mu A.\ Nat \rightarrow \{Nat, A\})$$

- **Objects**

Counter = μC. {get:Nat, inc:Unit$\to C$, dec:Unit$\to C$}

- **Recursive Values from Recursive Types**

$F = \mu A.A \to T$

$fixT = \lambda f{:}T\to T.\ (\lambda x{:}(\mu A.A\to T).\ f\ (x\ x))$
$\qquad\qquad (\lambda x{:}(\mu A.A\to T).\ f\ (x\ x))$

(Breaking the strong normalizing property:
diverge = λ_:Unit. fixT (λx:T. x) becomes typable)

- **Untyped Lambda-Calculus**: we can embed the whole untyped lambda-calculus—in a well-typed way—into a statically typed language with recursive types.

$D = \mu X. X \rightarrow X;$

We can extend it to include features like numbers.

$D = \mu X. \langle nat{:}Nat,\ fn{:}X \rightarrow X \rangle$

Relation between μX.T and its one-step unfolding: Two Approaches

- The equi-recursive approach
  - takes these two type expressions as definitionally equal—interchangeable in all contexts— since they stand for the same infinite tree.
  - more intuitive, but places stronger demands on the typechecker.

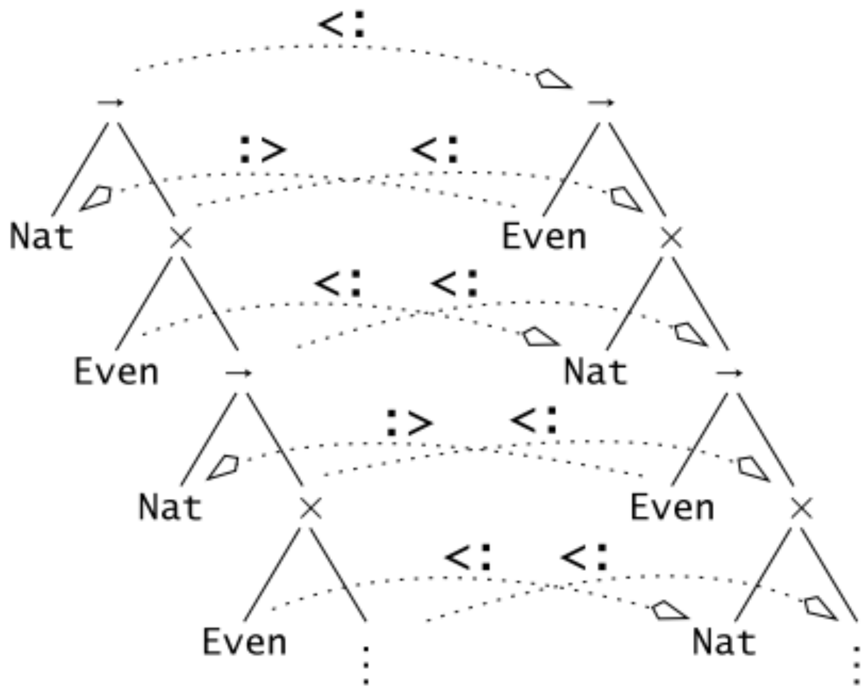
- 2. The iso-recursive approach
  - takes a recursive type and its unfolding as different, but isomorphic.
  - Notationally heavier, requiring programs to be decorated with fold and unfold instructions wherever recursive types are used.

Subtyping and Recursive Types

- Can we deduce

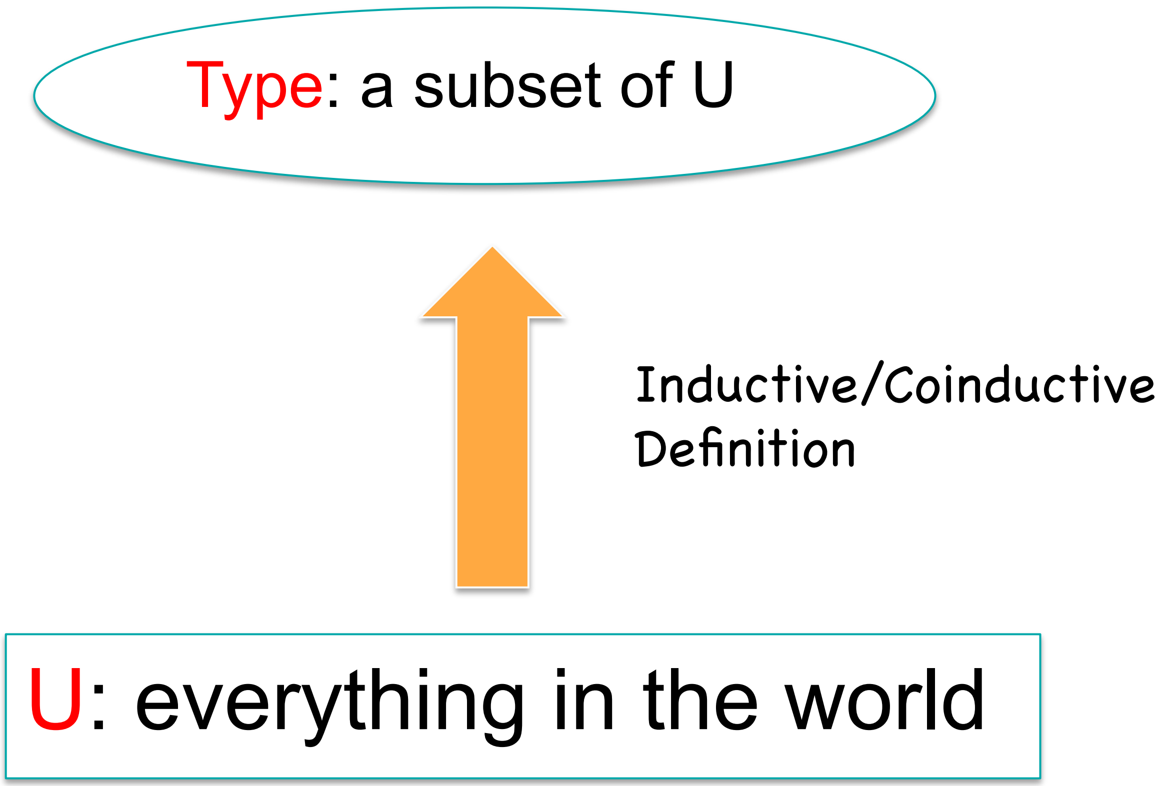
  $\mu X.\ Nat \to (Even \times X) <: \mu X.\ Even \to (Nat \times X)$

  from Even <: Nat?



21.1 Induction and Coinduction

Universal Set U



Generating Function

- Definition: A function $F \in P(U) \rightarrow P(U)$ is monotone if $X \subseteq Y$ implies $F(X) \subseteq F(Y)$.

- Definition: Let X be a subset of U.
  - X is F-closed if $F(X) \subseteq X$.
  - X is F-consistent if $X \subseteq F(X)$.
  - X is a fixed point of F if $F(X) = X$.

Exercise: Consider the following generating function on the three-element universe U={a, b, c}:

E1(∅) = {c}

E1({a}) = {c}

E1({b}) = {c}

E1({c}) = {b, c}

E1({a,b}) = {c}

E1({a, c}) = {b, c}

E1({b, c}) = {a, b, c}

E1({a, b, c}) = {a, b, c}

$$\frac{}{c} \qquad \frac{c}{b} \qquad \frac{b \quad c}{a}$$

Q: Which subset is E1-closed, E1-consistent?

Knaster-Tarski Theorem (1955)

Theorem

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

Definition: The least fixed point of F is written μF.
The greatest fixed point of F is written νF.

Exercise: Consider the following generating function on the three-element universe U={a, b, c}:

E1(∅) = {c}

E1({a}) = {c}

E1({b}) = {c}

E1({c}) = {b, c}

E1({a,b}) = {c}

E1({a, c}) = {b, c}

E1({b, c}) = {a, b, c}

E1({a, b, c}) = {a, b, c}

$$\frac{}{c} \qquad \frac{c}{b} \qquad \frac{b \quad c}{a}$$

Q: What are μE1 and νE1?

Exercise: Suppose a generating function E2 on the universe {a, b, c} is defined by the following inference rules:

$$\frac{}{a} \qquad\qquad \frac{c}{b} \qquad \frac{a \qquad b}{c}$$

Q: Write out the set of pairs in the relation E2 explicitly, as we did for E1 above. List all the E2-closed and E2-consistent sets. What are μE2 and νE2?

Principles of Induction/Coinduction

Corollary:

- Principle of induction:

  If X is F-closed, then $\mu F \subseteq X$.

- Principle of coinduction:

  If X is F-consistent, then $X \subseteq \nu F$.

The induction principle says that any property whose characteristic set is closed under F is true of all the elements of the inductively defined set μF.

The coinduction principle, gives us a method for establishing that an element x is in the coinductively defined set νF.

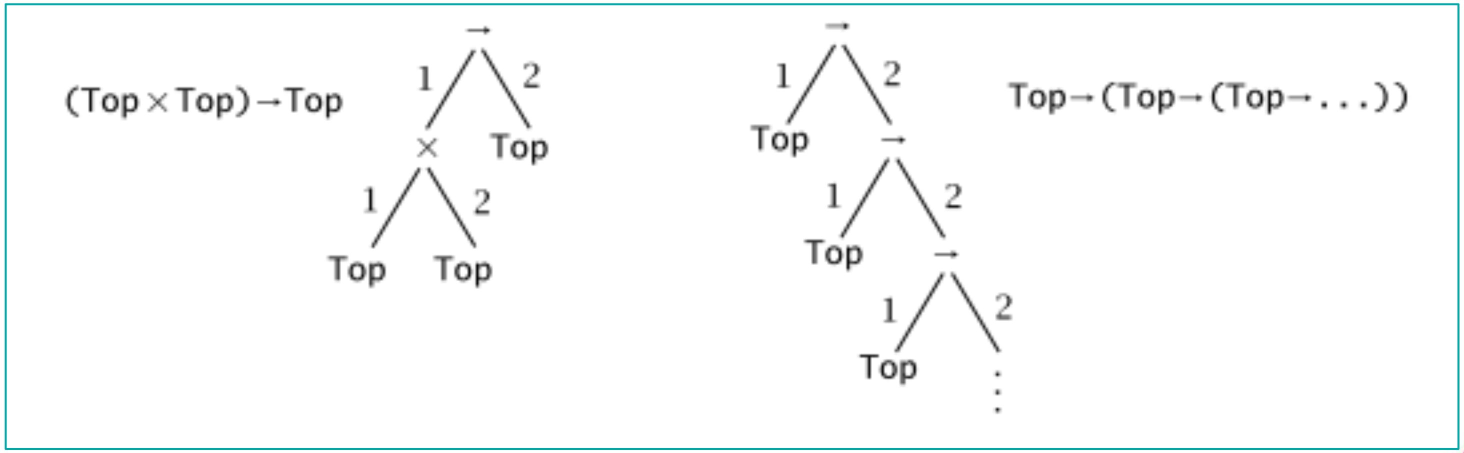
21.2 Finite and Infinite Types

To instantiate the general definitions of greatest fixed points and the coinductive proof method with the specifics of subtyping.

Tree Type

Definition: A tree type (or, simply, a tree) is a partial function $T \in \{1,2\}^* \rightharpoonup \{\rightarrow, \times, Top\}$ satisfying the following constraints:

- $T(\bullet)$ is defined;
- if $T(\pi,\sigma)$ is defined then $T(\pi)$ is defined;
- if $T(\pi) = \rightarrow$ or $T(\pi) = \times$ then $T(\pi,1)$ and $T(\pi,2)$ are defined;
- if $T(\pi) = Top$ then $T(\pi,1)$ and $T(\pi,2)$ are undefined.



Note: T(.) = Top

Definition: A tree type T is finite if dom(T) is finite. The set of all tree types is written $\mathcal{T}$; the subset of all finite tree types is written $\mathcal{T}_f$.

Exercise: Following the ideas in the previous paragraph, suggest a universe U and a generating function $F \in P(U) \to P(U)$ such that the set of finite tree types $\mathcal{T}_f$ is the least fixed point of F and the set of all tree types $\mathcal{T}$ is its greatest fixed point.

21.3 Subtyping

Finite Subtyping

Definition: Two finite tree types S and T are in the subtype relation ("S is a subtype of T") if $(S,T) \in \mu S_f$, where the monotone function

$$S_f \in P(\mathcal{T}_f \times \mathcal{T}_f) \to P(\mathcal{T}_f \times \mathcal{T}_f)$$

is defined by

$$
\begin{aligned}
S_f(R) = \{(T, Top) \mid T \in \mathcal{T}_f\} \\
\cup \{(S1 \times S2, T1 \times T2) \mid (S1,T1), (S2,T2) \in R\} \\
\cup \{(S1 \to S2, T1 \to T2) \mid (T1,S1), (S2,T2) \in R\}.
\end{aligned}
$$

Inference Rules

T <: Top

S1 <: T1   S2 <: T2

\-------------------

S1×S2 <: T1×T2

T1 <: S1 S2 <: T2

\--------------------

S1→S2 <: T1→T2

Infinite Subtyping

Definition: Two (finite or infinite) tree types S and T are in the subtype relation ("S is a subtype of T") if (S,T) ∈ νS, where the monotone function

$$S \in P(\mathcal{T} \times \mathcal{T}) \to P(\mathcal{T} \times \mathcal{T})$$

is defined by

$$S(R) = \{(T,Top) \mid T \in \mathcal{T}\}$$
$$\cup \{(S1\times S2, T1\times T2) \mid (S1,T1), (S2,T2) \in R\}$$
$$\cup \{(S1\to S2, T1\to T2) \mid (T1,S1), (S2,T2) \in R\}.$$

Transitivity

Definition: A relation $R \subseteq U \times U$ is transitive if R is closed under the monotone function

$$TR(R) = \{(x,y) \mid \exists z \in U.\ (x,z), (z,y) \in R\},$$

i.e., if $TR(R) \subseteq R$.

Lemma: Let $F \in P(U \times U) \rightarrow P(U \times U)$ be a monotone function. If $TR(F(R)) \subseteq F(TR(R))$ for any $R \subseteq U \times U$, then νF is transitive.

Theorem: νS is transitive.

A Digression on Transitivity

The possibility of giving a declarative presentation with the rule of transitivity turns out to be a consequence of a "trick" that can be played with inductive, but not coinductive, definitions.

- The union of two sets of rules, when applied inductively, generates the least relation that is closed under both sets of rules separately.
- Adding transitivity to the rules generating a coinductively defined relation always gives us a degenerate relation.

21.5 Membership Checking

Given a generating function F on some universe U and an element $x \in U$, check whether or not x falls in νF.

Invertible Generating Function

Definition: A generating function F is said to be **invertible** if, for all $x \in U$, the collection of sets

$$G_x = \{X \subseteq U \mid x \in F(X)\}$$

either is empty or contains a unique member that is a subset of all the others.

We will consider invertible generating function in the rest of this chapter.

F-Supported/F-Ground

When F is invertible, we define:

$$support_F(x) = \begin{cases} X & \text{if } X \in G_x \text{ and } \forall X' \in G_x.\ X \subseteq X' \\ \uparrow & \text{if } G_x = \emptyset \end{cases}$$

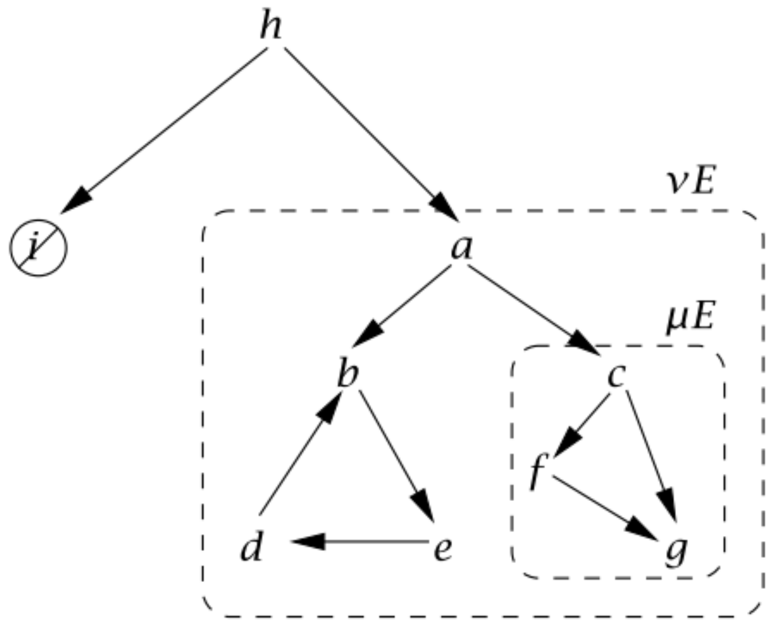
Definition: An element x is F-supported if $support_F(x)\downarrow$; otherwise, x is F- unsupported.

An F-supported element is called F-ground if $support_F(x) = \emptyset$.

$$support_F(X) = \begin{cases} \bigcup_{x \in X} support_F(x) & \text{if } \forall x \in X.\ support_F(x)\downarrow \\ \uparrow & \text{otherwise} \end{cases}$$

Support Graph

- An Example of the support graph of E function on {a,b,c,d,e,f,g,h,i}



x is in the greatest fixed point iff no unsupported element is reachable from x in the support graph.

Greatest Fixed Point

Definition: Suppose F is an invertible generating function. Define the boolean-valued function gfp_F (or just gfp) as follows:

$$gfp(X) = \begin{array}{l} \text{if } support(X)\uparrow, \text{ then } false \\ \text{else if } support(X) \subseteq X, \text{ then } true \\ \text{else } gfp(support(X) \cup X). \end{array}$$

Theorem (Sound):

1. If $gfp_F(X)$ = true, then $X \subseteq \nu F$.
2. 2. If $gfp_F(X)$ = false, then $X \not\subseteq \nu F$.

Theorem (Terminate): If $reachable_F(X)$ is finite, then $gfp_F(X)$ is defined. Consequently, if F is finite state, then $gfp_F(X)$ terminates for any finite $X \subseteq U$.

More Efficient Algorithms

Inefficiency

Recomputation of "support"

gfp({a})
= gfp({a, b, c})
= gfp({a, b, c, e, f ,g})
= gfp({a, b, c, e, f ,g,d})
= true

support(a) is recomputed four times!

A More Efficient Algorithm

Definition: Suppose F is an invertible generating function. Define the function gfp^a as follows

$$gfp^a(A, X) = \text{if } support(X) \uparrow, \text{ then } false$$
$$\text{else if } X = \emptyset, \text{ then } true$$
$$\text{else } gfp^a(A \cup X, support(X) \setminus (A \cup X)).$$

Tail-recursion

Example:

$$\begin{aligned} & gfp^a(\emptyset, \{a\}) \\ = \; & gfp^a(\{a\}, \{b, c\}) \\ = \; & gfp^a(\{a, b, c\}, \{e, f, g\}) \\ = \; & gfp^a(\{a, b, c, e, f, g\}, \{d\}) \\ = \; & gfp^a(\{a, b, c, e, f, g, d\}, \emptyset) \\ = \; & true. \end{aligned}$$

Variation 1

Definition: A small variation on gfp^s has the algorithm pick just one element at a time from X and expand its support. The new algorithm is called gfp^s

$$
\begin{aligned}
gfp^s(A, X) \quad = \quad & \text{if } X = \emptyset, \text{ then } \mathit{true} \\
& \text{else let } x \text{ be some element of } X \text{ in} \\
& \quad \text{if } x \in A \text{ then } gfp^s(A,\ X \setminus \{x\}) \\
& \quad \text{else if } \mathit{support}(x) \uparrow \text{ then } \mathit{false} \\
& \quad \text{else } gfp^s(A \cup \{x\},\ (X \cup \mathit{support}(x)) \setminus (A \cup \{x\})).
\end{aligned}
$$

Variation 2

Definition: Given an invertible generating function F, define the function gfp^t as follows:

$$
\begin{aligned}
gfp^t(A, x) \quad = \quad & \text{if } x \in A, \text{ then } A \\
& \text{else if } support(x) \uparrow, \text{ then } fail \\
& \text{else} \\
& \quad \text{let } \{x_1, \ldots, x_n\} = support(x) \text{ in} \\
& \quad \text{let } A_0 = A \cup \{x\} \text{ in} \\
& \quad \text{let } A_1 = gfp^t(A_0, x_1) \text{ in} \\
& \quad \ldots \\
& \quad \text{let } A_n = gfp^t(A_{n-1}, x_n) \text{ in} \\
& \quad A_n.
\end{aligned}
$$

Regular Trees

If we restrict ourselves to regular types, then the sets of reachable states will be guaranteed to remain finite and the subtype checking algorithm will always terminate.

Regular Trees

Definition: A tree type S is a subtree of a tree type T if $S = \lambda \sigma.\ T(\pi, \sigma)$ for some π.

Definition: A tree type $T \in \mathcal{T}$ is regular if subtrees(T) is finite.

Examples:

- Every finite tree type is regular.
- T = Top x (Top x (Top x ⋯)) is regular.
- T = B x (A x (B x (A x (A x (B x (A x (A x (A x (B ⋯) is irregular.

Proposition: The restriction of the generating function S to regular tree types is finite state.

Proof: We need to show that for any pair (S,T) of regular tree types, the set reachable(S,T) is finite.

Since reachable (S,T) $\subseteq$ subtrees(S) $\times$subtrees(T); the latter is finite as S and T are regular.

μ-Types

Establishes the correspondence between subtyping on μ-expressions and the subtyping on tree types

μ-Types:

Definition: Let X range over a fixed countable set {X1,X2,...} of type variables. The set of raw μ-types is the set of expressions defined by the following grammar:

$$
\begin{array}{lll}
T & ::= & X \\
 & & Top \\
 & & T \times T \\
 & & T \to T \\
 & & \mu X.T
\end{array}
$$

$\mathcal{T}_m$

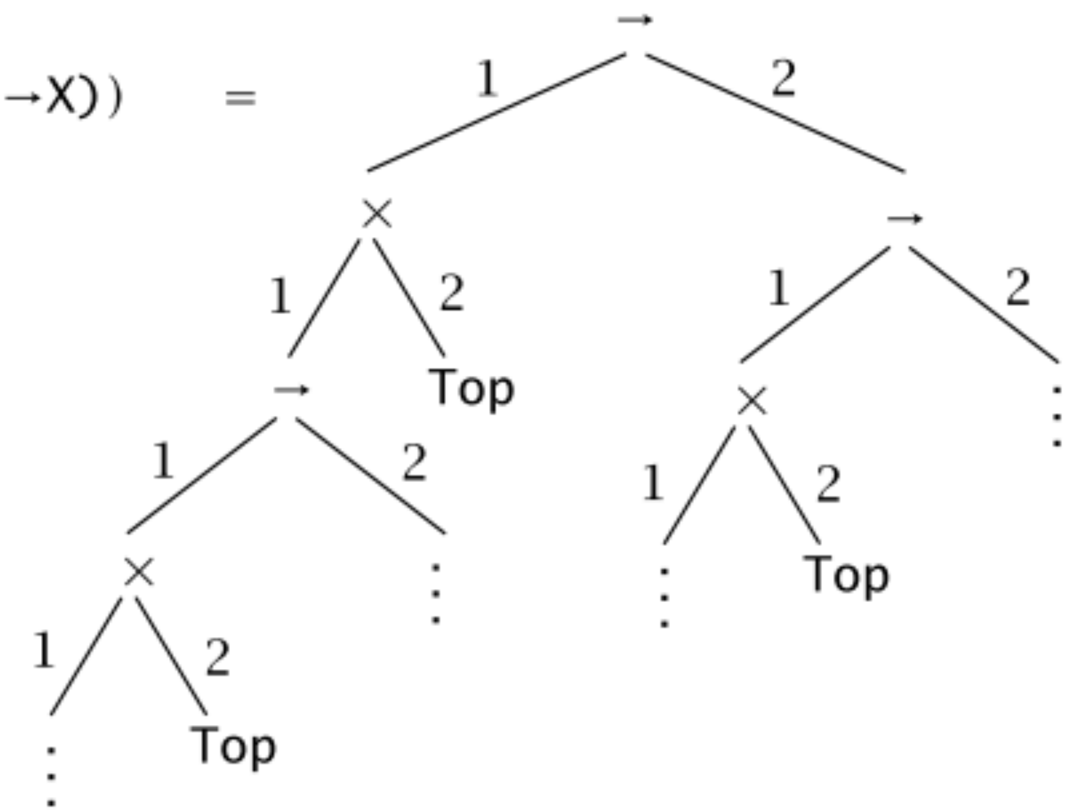
Definition: A raw μ-type T is contractive (and called μ-types) if, for any subexpression of T of the form μX.μX1...μXn.S, the body S is not X.

Finite Notation for Infinite Tree Types

Definition: The function treeof , mapping closed μ-types to tree types, is defined inductively as follows:

$$
\begin{aligned}
treeof(\mathsf{Top})(\bullet) &= \mathsf{Top} \\
treeof(\mathsf{T}_1 \to \mathsf{T}_2)(\bullet) &= \to \\
treeof(\mathsf{T}_1 \to \mathsf{T}_2)(i,\pi) &= treeof(\mathsf{T}_i)(\pi) \\
treeof(\mathsf{T}_1 \times \mathsf{T}_2)(\bullet) &= \times \\
treeof(\mathsf{T}_1 \times \mathsf{T}_2)(i,\pi) &= treeof(\mathsf{T}_i)(\pi) \\
treeof(\mu \mathsf{X}.\mathsf{T})(\pi) &= treeof([\mathsf{X} \mapsto \mu \mathsf{X}.\mathsf{T}]\mathsf{T})(\pi)
\end{aligned}
$$

$treeof(\mu X.((X \times \mathsf{Top}) \to X)) \quad =$



Subtyping Correspondence: μ-Types and Tree Types

Definition: Two μ-types S and T are said to be in the subtype relation if $(S,T) \in \nu S_m$, where the monotone function $S_m \in P(\mathcal{T}_m \times \mathcal{T}_m) \rightarrow P(\mathcal{T}_m \times \mathcal{T}_m)$ is defined by:

$$
\begin{array}{rcl}
S_m(R) & = & \{(\mathsf{S}, \mathsf{Top}) \mid \mathsf{S} \in \mathcal{T}_m\} \\
& \cup & \{(\mathsf{S}_1 \times \mathsf{S}_2, \mathsf{T}_1 \times \mathsf{T}_2) \mid (\mathsf{S}_1, \mathsf{T}_1),\ (\mathsf{S}_2, \mathsf{T}_2) \in R\} \\
& \cup & \{(\mathsf{S}_1 \rightarrow \mathsf{S}_2, \mathsf{T}_1 \rightarrow \mathsf{T}_2) \mid (\mathsf{T}_1, \mathsf{S}_1),\ (\mathsf{S}_2, \mathsf{T}_2) \in R\} \\
& \cup & \{(\mathsf{S}, \mu \mathsf{X}.\mathsf{T}) \mid (\mathsf{S}, [\mathsf{X} \mapsto \mu \mathsf{X}.\mathsf{T}]\mathsf{T}) \in R\} \\
& \cup & \{(\mu \mathsf{X}.\mathsf{S}, \mathsf{T}) \mid ([\mathsf{X} \mapsto \mu \mathsf{X}.\mathsf{S}]\mathsf{S}, \mathsf{T}) \in R,\ \mathsf{T} \neq \mathsf{Top},\ \text{and}\ \mathsf{T} \neq \mu \mathsf{Y}.\mathsf{T}_1\}.
\end{array}
$$

Theorem: Let $(S,T) \in \mathcal{T}_m \times \mathcal{T}_m$. Then $(S,T) \in \nu S_m$ iff (treeof S, treesof T) $\in \nu S$.

Exercise: What is the support for S_m?

$$
support_{S_m}(S,T) = \begin{cases}
\emptyset & \text{if } T = Top \\
\{(S_1,T_1),\ (S_2,T_2)\} & \text{if } S = S_1 \times S_2 \text{ and} \\
 & \quad T = T_1 \times T_2 \\
\{(T_1,S_1),\ (S_2,T_2)\} & \text{if } S = S_1 \rightarrow S_2 \text{ and} \\
 & \quad T = T_1 \rightarrow T_2 \\
\{(S, [X \mapsto \mu X.T_1]T_1)\} & \text{if } T = \mu X.T_1 \\
\{([X \mapsto \mu X.S_1]S_1, T)\} & \text{if } S = \mu X.S_1 \text{ and} \\
 & \quad T \neq \mu X.T_1,\ T \neq Top \\
\uparrow & \text{otherwise.}
\end{cases}
$$

Subtyping Algorithm for μ-Types

Instantiating gfp[†] for subtyping relation on μ-Types.

$$
\begin{array}{lll}
\textit{subtype}(A, \mathsf{S}, \mathsf{T}) & = & \text{if } (\mathsf{S},\mathsf{T}) \in A, \text{ then} \\
& & \quad A \\
& & \text{else let } A_0 = A \cup \{(\mathsf{S},\mathsf{T})\} \text{ in} \\
& & \quad \text{if } \mathsf{T} = \mathsf{Top}, \text{ then} \\
& & \qquad A_0 \\
& & \quad \text{else if } \mathsf{S} = \mathsf{S}_1 \times \mathsf{S}_2 \text{ and } \mathsf{T} = \mathsf{T}_1 \times \mathsf{T}_2, \text{ then} \\
& & \qquad \text{let } A_1 = \textit{subtype}(A_0, \mathsf{S}_1, \mathsf{T}_1) \text{ in} \\
& & \qquad \textit{subtype}(A_1, \mathsf{S}_2, \mathsf{T}_2) \\
& & \quad \text{else if } \mathsf{S} = \mathsf{S}_1 \to \mathsf{S}_2 \text{ and } \mathsf{T} = \mathsf{T}_1 \to \mathsf{T}_2, \text{ then} \\
& & \qquad \text{let } A_1 = \textit{subtype}(A_0, \mathsf{T}_1, \mathsf{S}_1) \text{ in} \\
& & \qquad \textit{subtype}(A_1, \mathsf{S}_2, \mathsf{T}_2) \\
& & \quad \text{else if } \mathsf{T} = \mu \mathsf{X}.\mathsf{T}_1, \text{ then} \\
& & \qquad \textit{subtype}(A_0, \mathsf{S}, [\mathsf{X} \mapsto \mu \mathsf{X}.\mathsf{T}_1]\mathsf{T}_1) \\
& & \quad \text{else if } \mathsf{S} = \mu \mathsf{X}.\mathsf{S}_1, \text{ then} \\
& & \qquad \textit{subtype}(A_0, [\mathsf{X} \mapsto \mu \mathsf{X}.\mathsf{S}_1]\mathsf{S}_1, \mathsf{T}) \\
& & \quad \text{else} \\
& & \qquad \textit{fail}
\end{array}
$$

Terminate?

An Digressed (Exponential) Subtyping Algorithm

$$
\begin{aligned}
\mathit{subtype}^{ac}(A, \mathsf{S}, \mathsf{T}) \quad = \quad & \text{if } (\mathsf{S}, \mathsf{T}) \in A, \text{ then } \mathit{true} \\
& \text{else let } A_0 = A \cup (\mathsf{S}, \mathsf{T}) \text{ in} \\
& \quad \text{if } \mathsf{T} = \mathsf{Top}, \text{ then } \mathit{true} \\
& \quad \text{else if } \mathsf{S} = \mathsf{S}_1 \times \mathsf{S}_2 \text{ and } \mathsf{T} = \mathsf{T}_1 \times \mathsf{T}_2, \text{ then} \\
& \qquad \mathit{subtype}^{ac}(A_0, \mathsf{S}_1, \mathsf{T}_1) \text{ and} \\
& \qquad \mathit{subtype}^{ac}(A_0, \mathsf{S}_2, \mathsf{T}_2) \\
& \quad \text{else if } \mathsf{S} = \mathsf{S}_1 \to \mathsf{S}_2 \text{ and } \mathsf{T} = \mathsf{T}_1 \to \mathsf{T}_2, \text{ then} \\
& \qquad \mathit{subtype}^{ac}(A_0, \mathsf{T}_1, \mathsf{S}_1) \text{ and} \\
& \qquad \mathit{subtype}^{ac}(A_0, \mathsf{S}_2, \mathsf{T}_2) \\
& \quad \text{else if } \mathsf{S} = \mu \mathsf{X}.\mathsf{S}_1, \text{ then} \\
& \qquad \mathit{subtype}^{ac}(A_0, [\mathsf{X} \mapsto \mu \mathsf{X}.\mathsf{S}_1]\mathsf{S}_1, \mathsf{T}) \\
& \quad \text{else if } \mathsf{T} = \mu \mathsf{X}.\mathsf{T}_1, \text{ then} \\
& \qquad \mathit{subtype}^{ac}(A_0, \mathsf{S}, [\mathsf{X} \mapsto \mu \mathsf{X}.\mathsf{T}_1]\mathsf{T}_1) \\
& \quad \text{else } \mathit{false}.
\end{aligned}
$$

Summary

- We study the theoretical foundation of type checkers (subtyping) for equi-recursive types.
  - Induction/coinduction & proof principles
  - Finite and Infinite Types/Subtyping
  - Membership checking algorithm

Homework

21.5.2 EXERCISE [★★]: Verify that S_f and S, the generating functions for the subtyping relations from Definitions 21.3.1 and 21.3.2, are invertible, and give their support functions. □