Lesson 9
Recursive Types

2/19, 21

Chapters 20, 21

Recursive type

- Recursive type terms are infinite, but regular
- For finite terms, use induction based on least fixed points
- For infinite terms/trees, we have to use coinduction, based on greatest fixed points.
- For least fixed points, recursively explore all subterms
- For greatest fixed points, recursively explore the support set (and hope that it is finite)

Greatest fixed point algorithm

How to check for membership in the gfp (of an invertible generating function F)?

gfp(X) = if support(X)$\uparrow$ then false
else if support(X) $\subseteq$ X then true
else gfp(support(X) $\cup$ X)

Correctness:

Thm: 1. If gfp(X) = true then X $\subseteq$ νF
2. If gfp(X) = false then X $\not\subseteq$ νF

Proof: induction on recursive computation of gfp(X)

Gfp algorithm: termination

pred(x) = $\emptyset$ if support(x)$\uparrow$
= support(X) if support(X)$\downarrow$

pred(X) = $\bigcup_{x \in X}$ pred(x)

reachable(X) = $\bigcup_{n \geq 0}$ $pred^n(x)$

reachable(x) = reachable({x})

F is *finite state* if reachable(x) is finite for each x

Thm: If reachable(X) is finite, then gfp(X) is defined.
If F is finite state, gfp(X) terminates for any finite X.

More efficient gfp: gfp^a

Avoid repeated addition of elements by keeping track of all elements examined before in a new argument A.

$$
\begin{aligned}
gfp^a(A,X) = \; & \text{if } support(X)\uparrow \text{ then false} \\
& \text{else if } X = \varnothing \text{ then true} \\
& \text{else } gfp^a(A \cup X, support(X) \setminus (A \cup X))
\end{aligned}
$$

This version considers only *new* elements added by support function.

$x \in \nu F$ iff $gfp^a(\varnothing,\{x\})$

More efficient gfp: gfp^t

Threaded version of gfp, adding one element at a time and producing visited set as the result (assume support(x) finite):

$$
\begin{aligned}
gfp^t(A,x) = \; & \text{if } x \in A \text{ then } A \\
& \text{else if } support(\{x\})\uparrow \text{ then fail} \\
& \text{else fold } gfp^t\,(A \cup \{x\})\ (support(x))
\end{aligned}
$$

where fold is a function like list fold but operating on sets:

$$
\begin{aligned}
& \text{fold } f\ X\ \varnothing = X \\
& \text{fold } f\ X\ \{y_1, y_2, \ldots, y_n\} = \text{fold } f\ (f(X,y_1))\ \{y_2, \ldots, y_n\}
\end{aligned}
$$

Correctness: $x \in \nu F$ iff $gfp^t(\varnothing,x)$

Regular Trees

Def: $S \in \mathcal{T}$ is a subtree of $T \in \mathcal{T}$ if $S = \lambda\sigma.\, T(\pi,\sigma)$ for some path $\pi \in dom(T)$. subtrees(T) is the set of subtrees of T.

Def: $T \in \mathcal{T}$ is regular if subtrees(T) is finite. $\mathcal{T}_r$ is the set of regular trees.

Subtype relation

The subtyping relation on $\mathcal{T}$ is defined as the greatest fixed point of the relation generator function

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

S_r is the restriction of S to $\mathcal{T}_r$ (regular trees).

Prop: S_r is finite state.

μ-Types

Def: $\mathcal{T}_{m\text{-raw}}$, the set of raw μ-types, is defined by the grammar:

$$T ::= X \mid Top \mid T \times T \mid T \to T \mid \mu X.T$$

This contains useless terms like $\mu X.X$ that we should exclude.

Def: $T \in \mathcal{T}_{m\text{-raw}}$ is contractive if for any subtree of Tof the form $\mu X.\mu X_1. \ldots \mu X_n. S$, S is not X. (I.e. there is always an occurrence of $\times$ or $\to$ between a binder μX and an applied occurrence of X. $\mathcal{T}_m$ denotes the set of contractive μ-types.

Can define a function *treeof* : $\mathcal{T}_m \to \mathcal{T}$ mapping μ-types to tree types.

Subtype relation for μ-Types

The generating function S_m for the subtyping relation on $\mathcal{T}_m$ is given by

$$
\begin{aligned}
S_m(R) = \; & \{(S, Top) \mid S \in \mathcal{T}_m\} \\
\cup \; & \{(S_1 \times S_2, T_1 \times T_2) \mid (S_1,T_1), (S_2,T_2) \in R\} \\
\cup \; & \{(S_1 \to S_2, T_1 \to T_2) \mid (T_1,S_1), (S_2,T_2) \in R\} \\
\cup \; & \{(S, \mu X.T) \mid (S, [X \text{->} \mu X.T]T) \in R\} \\
\cup \; & \{(\mu X.S, T) \mid ([X \text{->} \mu X.S]S, T) \in R, T \neq Top, \\
& \qquad T \neq \mu Y.T_1\}
\end{aligned}
$$

Subtype relation for μ-Types

S_m is invertible with support function

support(S,T)
- = $\emptyset$ if T = Top
- = $\{(S_1,T_1), (S_2,T_2)\}$ if $S = S_1 \times S_2$ and $T = T_1 \times T_2$
- = $\{(T_1,S_1), (S_2,T_2)\}$ if $S = S_1 \to S_2$ and $T = T_1 \to T_2$
- = $\{(S, [X \to \mu X.T_1]T_1)\}$ if $T = \mu X.T_1$
- = $\{([X \to \mu X.S_1]S_1, T)\}$ if $S = \mu X.S_1$, $T \neq Top$, $T \neq \mu Y.T_1$
- = $\uparrow$

Thm: $(S,T) \in \nu S_m$ iff treeof(S,T) $\in \nu S$

Subtyping algorithm

Specialize gfp^t to S_m.

subtype (S,T)
- = if $(S,T) \in A$ then A
  - else let $A_0 = A \cup \{(S,T)\}$ in
  - if T = Top then A_0
  - else if $S = S_1 \times S_2$ and $T = T_1 \times T_2$ then
    - let A_1 = subtype(A_0,S_1,T_1) in subtype(A_1,S_2,T_2)
  - else if $S = S_1 \to S_2$ and $T = T_1 \to T_2$ then
    - let A_1 = subtype(A_0,T_1,S_1) in subtype(A_1,S_2,T_2)
  - else if $T = \mu X.T_1$ then subtype(A_0,S,$[X \to \mu X.T_1]T_1$)
  - else if $S = \mu X.S_1$ then subtype(A_0,$[X \to \mu X.S_1]S_1$)
  - else *fail*

Subtyping Iso-recursive types

The Amber rule:

$$\frac{\Box, X <: Y \;|\text{-}\; S <: T}{\Box \;|\text{-}\; \Box X.S <: \Box Y.T} \qquad \text{(S-Amber)}$$

$$\frac{X <: Y \;\Box\; \Box}{\Box \;|\text{-}\; X <: Y} \qquad \text{(S-Assumption)}$$