Design Principles of Programming Languages

Metatheory of Recursive Types

Zhenjiang Hu, Haiyan Zhao, Yingfei Xiong

Peking University, Spring Term, 2016

提醒：课程项目

- 6月8日课程项目报告
  - 每组报告20分钟，提问5分钟
  - 组队的同学请介绍每名成员的贡献
- 6月7日课程书面报告和代码提交
  - 邮件发给吴迪、杨至轩和我

Review: Iso-recursive Types

- What are the types of the following terms?
  - `Hungry =` μ`A. Nat→A;`
  - `h = fix (`λ`f: Nat→Hungry.` λ`n:Nat. f)`
    - `h ?`
    - `fold[Hungry] h ?`
    - `unfold[Hungry] (h 1) ?`

Review: Iso-recursive Types

$\rightarrow \mu$ *Extends $\lambda_{\rightarrow}$ (9-1)*

t	::=	...	*terms:*
		fold [T] t	*folding*
		unfold [T] t	*unfolding*
v	::=	...	*values:*
		fold [T] v	*folding*
T	::=	...	*types:*
		X	*type variable*
		μX.T	*recursive type*

New evaluation rules $\boxed{t \longrightarrow t'}$

$$\texttt{unfold [S] (fold [T] } v_1\texttt{)} \longrightarrow v_1 \qquad \text{(E-UnfldFld)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{fold [T] } t_1 \longrightarrow \texttt{fold [T] } t_1'} \qquad \text{(E-Fld)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{unfold [T] } t_1 \longrightarrow \texttt{unfold [T] } t_1'} \qquad \text{(E-Unfld)}$$

New typing rules $\boxed{\Gamma \vdash t : T}$

$$\frac{U = \mu X.T_1 \qquad \Gamma \vdash t_1 : [X \mapsto U]T_1}{\Gamma \vdash \texttt{fold [U] } t_1 : U} \qquad \text{(T-Fld)}$$

$$\frac{U = \mu X.T_1 \qquad \Gamma \vdash t_1 : U}{\Gamma \vdash \texttt{unfold [U] } t_1 : [X \mapsto U]T_1} \qquad \text{(T-Unfld)}$$

Figure 20-1: Iso-recursive types ($\lambda\mu$)

Equi-recursive approach

- Do not use explicit fold/unfold
- If type A can be constructed from type B by applying only fold and/or unfold, A and B are equal
- Example: the following three types are equal
  - Hungry
  - Nat→Hungry
  - Nat→Nat→Hungry

Solution

- Alternative 1: Deduce all equal types for a term
  - possibly infinite number of types

Solution

- Alternative 1: Deduce all equal types for a term
  - possibly infinite number of types
- Alternative 2: use algorithms to determine the subtyping relations
  - An algorithm to determine if type A is a subtype of type B
  - We do not need an algorithm to determine the equality of two types
    - It can be deduced from subtyping relations
      - $A <: B \land B <: A \rightarrow A = B$
    - It will never be used

Iso-recursive Subtyping

$$\frac{}{T <: Top} \qquad \frac{}{Even <: Nat}$$

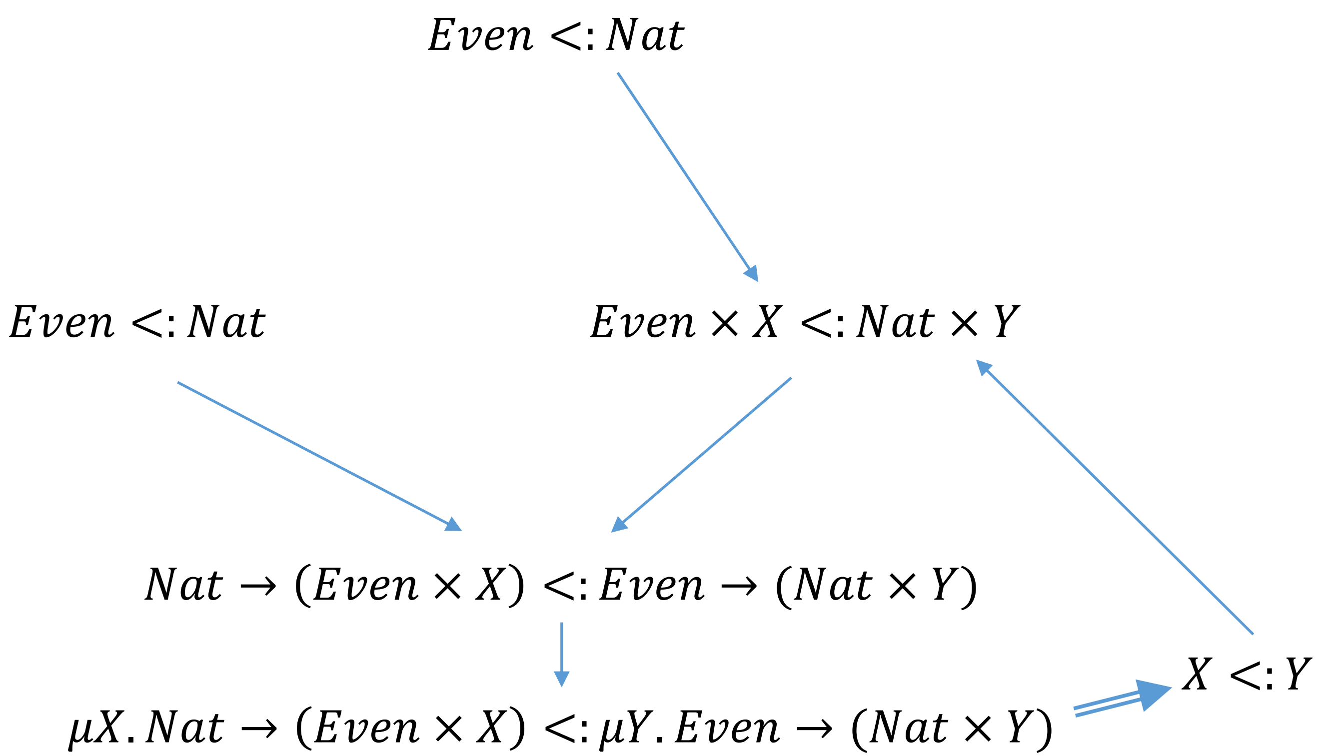
$$\frac{S_1 <: T_1 \qquad S_2 <: T_2}{S_1 \times S_2 <: T_1 \times T_2} \qquad \frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2}$$

$$\frac{\Sigma, X <: Y \vdash S <: T}{\Sigma \vdash \mu X.S <: \mu Y.T}$$

Without the last rule: A Derivation Tree

$$
\begin{tikzcd}
 & Even <: Nat \arrow[dr] & & Even <: Nat \arrow[dl] \\
Even <: Nat \arrow[dr] & & Even \times Even <: Nat \times Nat \arrow[dl] & \\
 & Nat \rightarrow (Even \times Even) <: Even \rightarrow (Nat \times Nat) & &
\end{tikzcd}
$$

With the last rule: A Derivation Graph



The premise function

- $premise(S <: T) =$
$$\begin{cases} \emptyset & if\ T = Top \lor (S = Even \land T = Nat) \\ \{S_1 <: T_1, S_2 <: T_2\} & if\ S = S_1 \times S_2 \land T = T_1 \times T_2 \\ \{T_1 <: S_1, S_2 <: T_2\} & if\ S = S_1 \to S_2 \land T = T_1 \to T_2 \\ \{S_1 <: T_1\} & if\ S = \mu X.S_1 \land T = \mu X.T_1 \\ \uparrow & otherwise \end{cases}$$

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

The derivation function

- $derivation(S <: T) = \begin{cases} \{S <: T, X <: Y\} & if\ S = \mu X.S_1 \wedge T = \mu Y.T_1 \\ \{S <: T\} & otherwise \end{cases}$

- $derivation(X) = \bigcup_{x \in X} derivation(x)$

The subtyping algorithm

- gfp(X)=if premise(X)↑ then false
  else premise(X)⊆derivation(X) then true
  else gfp(premise(X)∪X)
- isSubtype(S<:T)=gfp({S<:T})

Termination

- X grows larger in every iteration
- Function premise() only produce subexpressions
  - subexpression: a sub tree in the AST
- There are finite number of subexpressions for a type

$$Even <: Nat$$

$$Even <: Nat \qquad Even \times X <: Nat \times Y$$

$$Nat \to (Even \times X) <: Even \to (Nat \times Y)$$

$$X <: Y$$

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

Exercises

- Try to determine the following subtype relations using the algorithm
  - $\mu X.\mu Y.X \times Y <: \mu A.\mu B.A \times B$
  - $\mu X.X \rightarrow Nat <: \mu Y.Y \rightarrow Nat$
  - $\mu X.Nat \rightarrow (Even \times X) <:\ Even \rightarrow \big(Nat \times$

Exercises

- Try to determine the following subtype relations using the algorithm
  - $\mu X.\mu Y.X \times Y <: \mu A.\mu B.A \times B$
    - true
  - $\mu X.X \to Nat <: \mu Y.Y \to Nat$
    - false
    - the current algorithm allows only to unfold once
  - $\mu X.Nat \to (Even \times X) <:\ Even \to \big(Nat \times$

Changing the typing rule

$$\cancel{\dfrac{\Sigma, X <: Y \vdash S <: T}{\Sigma \vdash \mu X.S <: \mu Y.T}}$$

$$\dfrac{\Sigma, S <: \mu Y.T \vdash S <: [Y \to \mu Y.T]T}{\Sigma \vdash S <: \mu Y.T}$$

$$\dfrac{\Sigma, \mu X.S <: T \vdash [X \to \mu X.S]S <: T}{\Sigma \vdash \mu X.S <: T}$$

What is the derviation graph of XX $<: Even \to (Nat \times YY)$?

New Derivation Graph

$$
\begin{tikzcd}
 & Even <: Nat \arrow[d] & XX <: YY \arrow[dl] \\
Even <: Nat \arrow[dr] & Even \times XX <: Nat \times YY \arrow[d] & \\
 & Nat \to (Even \times XX) <: Even \to (Nat \times YY) \arrow[d] & \\
 & \mathrm{XX} <: Even \to (Nat \times YY) \arrow[uuur] &
\end{tikzcd}
$$

Support Function

- $support_{S_m}(S <: T) =$

$$\begin{cases} \emptyset & if\ T = Top \vee (S = Even \wedge T = Nat) \\ \{S_1 <: T_1, S_2 <: T_2\} & if\ S = S_1 \times S_2 \wedge T = T_1 \times T_2 \\ \{T_1 <: S_1, S_2 <: T_2\} & if\ S = S_1 \rightarrow S_2 \wedge T = T_1 \rightarrow T_2 \\ \{S <: [X \mapsto \mu X.T_1]T_1\} & if\ T = \mu X.T_1 \\ \{[X \mapsto \mu X.S_1]S_1 <: T\} & if\ S = \mu X.S_1 \wedge T \neq \mu X.T_1, T \neq Top \\ \uparrow & otherwise \end{cases}$$

- $support_{S_m}(X) =$

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

The algorithm

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

Termination

- X grows larger in every iteration
- S is a subexpression of T either
  - S forms a sub tree in the AST of T
  - S forms a sub tree in the AST of $[\mathrm{X} \rightarrow \mu X.T_1]T_1$ if T=$\mu X.T_1$
- All pairs produced by $support_{S_m}()$ are subexpressions of the original one
- There is only a finite number of subexpressions

Inversible Subtyping Rules

- Functions premise/support requires the subtyping rules are inversible:
  - There is only one set of premise for each conclusion
- The algorithm will be much more complex if the subtyping rules are not inversible
- Example: uninversible rules

$$\frac{S <: U \qquad U <: T}{S <: T}$$

$$S <: Top$$

$$\frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2}$$

$$\frac{S <: [Y \to \mu Y.T]T}{S <: \mu Y.T}$$

$$\frac{[X \to \mu X.S]S <: T}{\mu X.S <: T}$$

Inversible Subtyping Rules

- Functions premise/support requires the subtyping rules are inversible:
  - There is only one set of premise for each conclusion
- The algorithm will be much more complex if the subtyping rules are not inversible
- Example: uninversible rules

$$\cancel{\frac{S <: U \qquad U <: T}{S <: T}}$$

$$S <: Top$$

$$\frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1 \rightarrow S_2 <: T_1 \rightarrow T_2}$$

$$\frac{S <: [Y \rightarrow \mu Y.T]T}{S <: \mu Y.T}$$

$$\frac{[X \rightarrow \mu X.S]S <: T \wedge T \neq \mu Y.T_1 \wedge T \neq TOP}{\mu X.S <: T}$$

Exercise

- Find two types S<:T where S<:T does not hold in iso-recursive types (even with the help of fold/unfold) but holds in equi-recursive types.

Exercise

- Find two types S<:T where S<:T does not hold in iso-recursive types (even with the help of fold/unfold) but holds in equi-recursive types.

- $S = \mu X. Nat \times X$
- $T = \mu X. Nat \times (Nat \times X)$

Fixpoints, Induction, and Coinduction

Fixed points

- The fixed point of a function $f:T \to T$, is a value $(\text{fix } f) \in T$ satisfying the following condition:
  - fix f = f (fix f)
- When T is a function type
  - fix f is a recursive function
  - Y and fix combinators produce such fixed point
- When T is not a function
  - Y and fix combinators no longer work

Review: Terms, by Inference Rules

The set of terms is defined by the following rules:

$$\texttt{true} \in \mathcal{T} \qquad \texttt{false} \in \mathcal{T} \qquad \texttt{0} \in \mathcal{T}$$

$$\frac{\texttt{t}_1 \in \mathcal{T}}{\texttt{succ}\ \texttt{t}_1 \in \mathcal{T}} \qquad \frac{\texttt{t}_1 \in \mathcal{T}}{\texttt{pred}\ \texttt{t}_1 \in \mathcal{T}} \qquad \frac{\texttt{t}_1 \in \mathcal{T}}{\texttt{iszero}\ \texttt{t}_1 \in \mathcal{T}}$$

$$\frac{\texttt{t}_1 \in \mathcal{T} \qquad \texttt{t}_2 \in \mathcal{T} \qquad \texttt{t}_3 \in \mathcal{T}}{\texttt{if}\ \texttt{t}_1\ \texttt{then}\ \texttt{t}_2\ \texttt{else}\ \texttt{t}_3 \in \mathcal{T}}$$

Inference rules = Axioms + Proper rules

Review: Terms, Concretely

For each natural number i, define a set S_i as follows:

$$
\begin{aligned}
S_0 &= \varnothing \\
S_{i+1} &= \quad \{\mathtt{true}, \mathtt{false}, 0\} \\
&\quad \cup \{\mathtt{succ}\ \mathtt{t}_1, \mathtt{pred}\ \mathtt{t}_1, \mathtt{iszero}\ \mathtt{t}_1 \mid \mathtt{t}_1 \in S_i\} \\
&\quad \cup \{\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \mid \mathtt{t}_1, \mathtt{t}_2, \mathtt{t}_3 \in S_i\}.
\end{aligned}
$$

Finally, let

$$S = \bigcup_i S_i.$$

Generating Function

- f(X) = {true, false, 0}
  ∪ {succ t_1, pred t_1, iszero t_1 | $t_1 \in X$}
  ∪ {if t_1 then t_2 else t_3 | $t_1, t_2, t_3 \in X$}

- S = $\bigcup f^n(\emptyset)$
- We will show that S is the least fixed point of f

Monotone function and closed sets

- Monotone function: f : P(U)→P(U) is monotone iff
  - $\forall X, Y: X \subseteq Y = f(X) \subseteq f(Y)$

- Let $f: P(U) \to P(U)$, X is f-closed if $f(X) \subseteq X$.

Knaster-Tarski Theorem

- Knaster-Tarski Theorem
  - The intersection of all f-closed sets is the least fixed point of monotone function f, denoted $lfp(f)$.
  - Proof:
    - Let K be the intersection of all f-closed sets
    - Let A be an arbitrary f-closed set
    - $K \subseteq A \rightarrow f(K) \subseteq f(A) \rightarrow f(K) \subseteq A$
    - Since A can be any f-closed set, $\mathrm{f}(K) \subseteq K$
    - $f(K) \subseteq K \rightarrow f\big(f(K)\big) \subseteq f(K) \rightarrow f(K)$ is f-closed $\rightarrow K \subseteq f(K)$
    - Therefore $f(K) = K$
    - K is the least because any fixed point is f-closed

Principle of Induction

- If X is f-closed, then $lfp(f) \subseteq X$.

- Proving S = $\cup f^n(\emptyset) = lfp(f)$
  - $\emptyset \subseteq lfp(f) \rightarrow f^n(\emptyset) \subseteq lfp(f)$ for any n
  - Thus, $S \subseteq lfp(f)$
  - Let $A \subseteq B$, we have $f(A \cup B) = f(A) \cup f(B)$
  - From $\emptyset \subseteq f(\emptyset)$, we have $f^n(\emptyset) \subseteq f^{n+1}(\emptyset)$ for any n
  - $f(S) = f(\cup f^n(\emptyset)) = \cup f^{n+1}(\emptyset) = S$, e.g., S is f-closed
  - $lfp(f) \subseteq S$

Proving Mathematical Induction

- Mathematical induction
  1. Show P holds for case n=0
  2. When P holds for case n=k, show P holds for case n=k+1
  3. P holds for any natural number

- Let $f(X) = \{0\} \cup \{\, i+1 \mid i \in X \,\}$. We have lfp(f) is the whole set of natural numbers
- Let PP be the set of natural number where P holds. We have
  - $0 \in PP \wedge i \in PP \rightarrow i+1 \in PP$
  - PP is f-closed
  - $lfp(f) \subseteq PP$

Infinite Values

- Let $f(X)=\{nil\} \cup \{cons\ i\ t \mid i \in Nat,\ t \in X\}$

- What is in lfp(X)?

Principle of Coinduction

- Let $f: P(U) \rightarrow P(U)$, X is f-consistent if $X \subseteq f(X)$.

- The dual of Knaster-Tarski Theorem
  - The union of all f-consistent sets is the greatest fixed point of monotone function f, denoted $\mathrm{g}fp(f)$.
    - Proof: By duality

- Principle of Coinduction
  - If X is f-consistent, then $\mathrm{X} \subseteq gfp(f)$.
    - Proof: By duality

Infinite Members and Greatest Fixed Point

- $gfp(f) = \cap f^n(U), n$ is any natural number is the greatest fixed point of the monotone function f and the universal set U

- Let f(X)={nil}U{cons i t | i∈Nat, t∈X}, gfp(f) contains all finite and infinite lists

Summary

- Rules can be represented as generating functions
- The least fixed point is the set of finite terms
- The greatest fixed point is the set of finite and infinite terms
- Principles of Induction and Coinduction are useful in proving theorems
  - See book for examples of using principles of coinduction

Exercise

- Defining a generating function s for the subtyping relation, where gfp(s) is the set of all pairs of (A, B) where A<:B

$$\frac{}{T <: Top}$$

$$\frac{}{Even <: Nat}$$

$$\frac{S_1 <: T_1 \qquad S_2 <: T_2}{S_1 \times S_2 <: T_1 \times T_2}$$

$$\frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1 \rightarrow S_2 <: T_1 \rightarrow T_2}$$

$$\frac{S <: [Y \rightarrow \mu Y.T]T}{S <: \mu Y.T}$$

$$\frac{[X \rightarrow \mu X.S]S <: T}{\mu X.S <: T}$$

Exercise

- Defining a generating function s for the subtyping relation, where gfp(s) is the set of all pairs of (A, B) where A<:B

$$
\begin{aligned}
s(R) = &\{ S <: Top \mid \text{for any type } S \} \\
&\cup \{S_1 \times S_2 <: T_1 \times T_2 \mid S_1 <: T_1, S_2 <: T_2 \in R\} \\
&\cup \{S_1 \to S_2 <: T_1 \to T_2 \mid T_1 <: S_1, S_2 <: T_2 \in R\} \\
&\cup \{S <: \mu X.T \mid S <: [X \mapsto \mu X.T]T \in R\} \\
&\cup \{\mu X.S <: T \mid [\mathrm{X} \mapsto \mu X.T]S <: T \in R\}
\end{aligned}
$$

Homework

- Choose a language with high-order function support, and investigate
  - (1) Whether and how this language supports recursive types,
  - (2) How this support differs from what we learned in the course, and
  - (3) Why this design is adopted for the language.
- Summarize the findings as a report.