

Metatheory of Recursive Types

Zhenjiang Hu, Haiyan Zhao, <u>Yingfei Xiong</u> Peking University, Spring Term, 2014



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
 - We do not need an algorithm to determine the equality of two types
 - It can be deduced from subtyping relations
 - $A <: B \land B <: A \to A = B$
 - It will never be used



Iso-recursive Subtyping

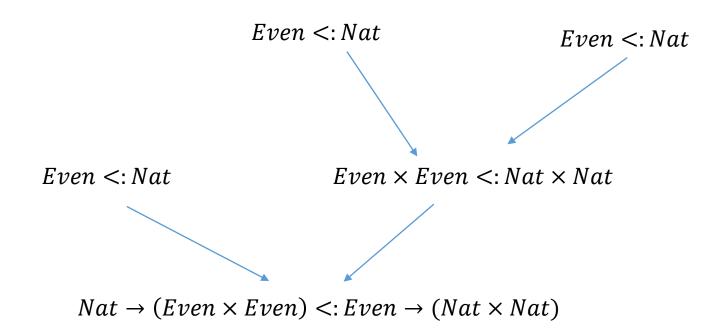


$$\frac{\Sigma, X<:Y \vdash S<:T}{\Sigma \vdash \mu X.S<:\mu Y.T}$$



Without the last rule: A Derivation Tree

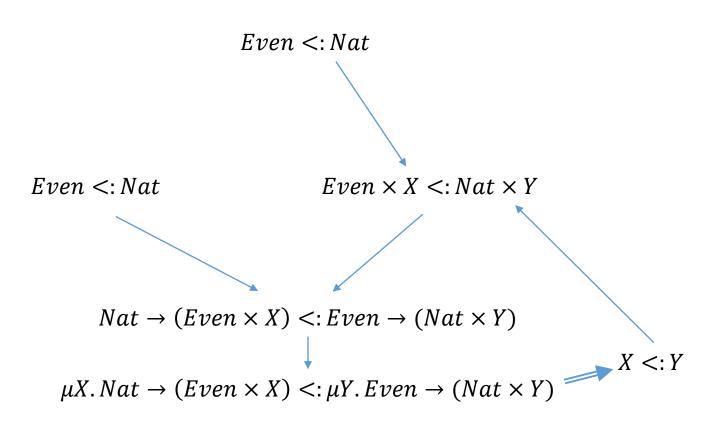






With the last rule: A Derivation Graph







The premise function



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



The derivation function



•
$$derivation(S <: T) =$$

$$\begin{cases} \{S <: T, X <: Y\} & if S = \mu X.S_1 \land T = \mu Y.T_1 \\ \{S <: T\} & otherwise \end{cases}$$

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



The subtyping algorithm



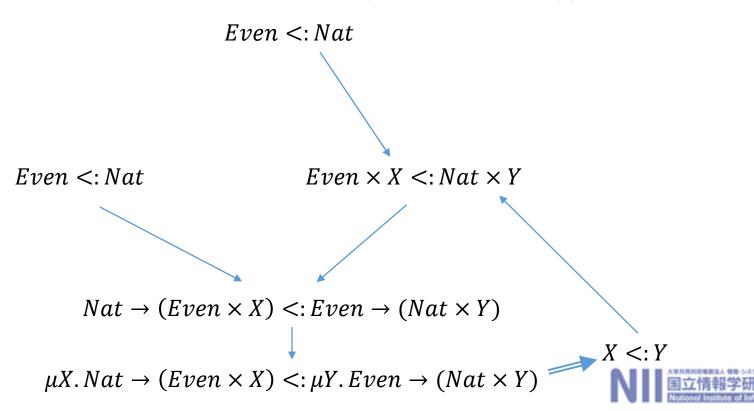
- gfp(X)=if premise(X)↑ then false
 else premise(X)⊆derivation(X) then true
 else gfp(premise(X)∪X)
- isSubtype(S<:T)=gfp({S<:T})



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



Problems of Equi-recursive types



- Let
 - $XX = \mu X. Nat \rightarrow (Even \times X)$
 - $YY = \mu Y \cdot Even \rightarrow (Nat \times Y)$
- What if isSubtype(XX $<: Even \rightarrow (Nat \times YY))$?



Changing the typing rule



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

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

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



Future simplify



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

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

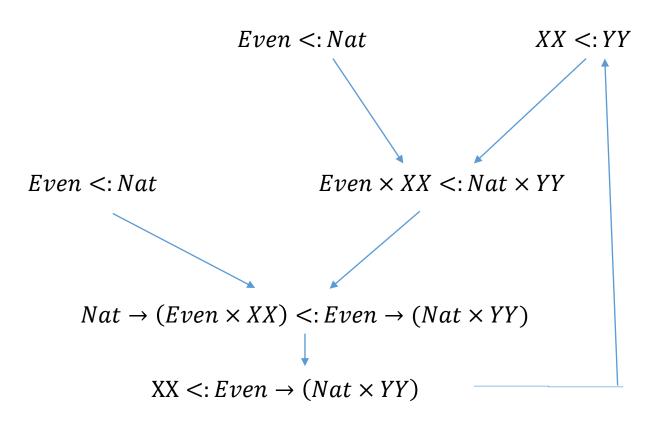
$$\frac{[X \to \mu X. S]S <: T}{\mu X. S <: T}$$

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



New Derivation Graph







Support Function



• $support_{S_m}(S <: T) =$

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

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



The algorithm



 $gfp(X) = \text{if } support(X) \uparrow, \text{ then } false$ else if $support(X) \subseteq X$, then trueelse $gfp(support(X) \cup X)$.



Uncontractive Types



• Type $\mu X. \mu X_1. \mu X_2 ... \mu X_n. X$

- Meaningless type
- All uncontractive types are equal



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 $[X \to \mu X. T_1]T_1$ 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 is the subtyping rules are not inversible
- Example: uninversible rules

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

$$S <: Top$$

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

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

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



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 is the subtyping rules are not inversible
- Example: uninversible rules

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

$$S <: Top$$

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

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

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



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→T, is a value (fix f)∈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:

$$\begin{array}{ll} \mathsf{true} \in \mathcal{T} & \mathsf{false} \in \mathcal{T} & \mathsf{0} \in \mathcal{T} \\ \\ \frac{\mathsf{t}_1 \in \mathcal{T}}{\mathsf{succ} \; \mathsf{t}_1 \in \mathcal{T}} & \frac{\mathsf{t}_1 \in \mathcal{T}}{\mathsf{pred} \; \mathsf{t}_1 \in \mathcal{T}} & \frac{\mathsf{t}_1 \in \mathcal{T}}{\mathsf{iszero} \; \mathsf{t}_1 \in \mathcal{T}} \\ \\ \frac{\mathsf{t}_1 \in \mathcal{T} \quad \mathsf{t}_2 \in \mathcal{T} \quad \mathsf{t}_3 \in \mathcal{T}}{\mathsf{if} \; \mathsf{t}_1 \; \mathsf{then} \; \mathsf{t}_2 \; \mathsf{else} \; \mathsf{t}_3 \in \mathcal{T}} \end{array}$$

Inference rules = Axioms + Proper rules



Review: Terms, Concretely



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

$$egin{array}{lll} S_0 &=& \varnothing \ S_{i+1} &=& \{ exttt{true}, exttt{false}, 0 \} \ & \cup & \{ exttt{succ} \ exttt{t}_1, exttt{pred} \ exttt{t}_1, exttt{iszero} \ exttt{t}_1 \ | \ exttt{t}_1 \in S_i \} \ & \cup & \{ exttt{if} \ exttt{t}_1 \ exttt{then} \ exttt{t}_2 \ exttt{else} \ exttt{t}_3 \ | \ exttt{t}_1, exttt{t}_2, exttt{t}_3 \in S_i \}. \end{array}$$

Finally, let
$$S = \bigcup_{i} S_i$$
.



Generating Function



•
$$f(X) = \{true, false, 0\}$$

 $\cup \{succ\ t_1, pred\ t_1, iszero\ t_1 \mid t_1 \in X\}$
 $\cup \{if\ t_1\ then\ t_2\ else\ t_3 \mid t_1, t_2, t_3 \in X\}$

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



Monotone function and closed sets



- Monotone function: $f : P(U) \rightarrow P(U)$ 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
 - $K \subseteq A \to f(K) \subseteq f(A) \to f(K) \subseteq A$
 - Since A can be any f-closed set, $f(K) \subseteq K$
 - $f(K) \subseteq K \to f(f(K)) \subseteq f(K) \to f(K)$ is f-closed $\to K \subseteq f(K)$
 - 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 = $\bigcup f^n(\emptyset) = lfp(f)$
 - $\emptyset \subseteq lfp(f) \rightarrow f^n(\emptyset) \subseteq lfp(f)$ 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
 - $f(S) = f(\bigcup f^n(\emptyset)) = \bigcup f^{n+1}(\emptyset) = S$, 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
 - 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
 - $0 \in PP \land i \in PP \rightarrow i + 1 \in PP$
 - 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 Ifp(X)?



Principle of Coinduction



• Let $f: P(U) \to P(U)$, 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 gfp(f).
 - Proof: By duality

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



Infinite Members and Greatest Fixed Point



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

 Let f(X)={nil}∪{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

$$\frac{\mathsf{S}_1 \mathrel{<:} \mathsf{T}_1 \qquad \mathsf{S}_2 \mathrel{<:} \mathsf{T}_2}{\mathsf{S}_1 \times \mathsf{S}_2 \mathrel{<:} \mathsf{T}_1 \times \mathsf{T}_2}$$

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

$$\frac{\mathsf{T}_1 <: \mathsf{S}_1}{\mathsf{S}_1 \to \mathsf{S}_2 <: \mathsf{T}_1 \to \mathsf{T}_2}$$

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



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

```
s(R) = { S <: Top | 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 [X \mapsto \mu X.T]S <: T \in R\}
```



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.

