

Chapter 23: Universal Types

System F Power of System F Properties (Soundness, decidability, paramertricity, impredicativity)



System F



- First discovered by Jean-Yves Girard (1972)
- Independently developed by John Reynolds (1974) as polymorphic lambda-calculus
- A natural extension of $\lambda \rightarrow$ with a new form of abstract and application over types:

$$(\lambda X.t_{12}) [T_2] \rightarrow [X \mapsto T_2]t_{12}$$
$$(\lambda x:T_{11}.t_{12}) v_2 \rightarrow [x \mapsto v_2]t_{12}$$
$$\frac{\Gamma, X \vdash t_2 : T_2}{\Gamma \vdash \lambda X.t_2 : \forall X.T_2}$$
$$\frac{\Gamma \vdash t_1 : \forall X.T_{12}}{\Gamma \vdash t_1 [T_2] : [X \mapsto T_2]T_{12}}$$



Syntax and Evaluation

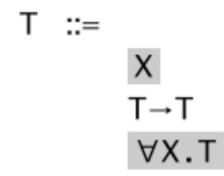


Syntax t ::=	x λx:T.t	terms: variable abstraction	$\begin{tabular}{lllllllllllllllllllllllllllllllllll$
	tt λX.t t[T]	application type abstraction type application	$\frac{t_2 \longrightarrow t_2'}{v_1 \ t_2 \longrightarrow v_1 \ t_2'} \tag{E-APP2}$ $(\lambda \mathbf{x}:T_{11}.t_{12}) \ v_2 \longrightarrow [x \mapsto v_2] t_{12} \ (\text{E-APPABS})$
v ::=	λx:T.t λX.t	values: abstraction value type abstraction value	$\frac{t_{1} \longrightarrow t_{1}'}{t_{1} \ [T_{2}] \longrightarrow t_{1}' \ [T_{2}]} \tag{E-TAPP}$ $(\lambda X. t_{12}) \ [T_{2}] \longrightarrow [X \mapsto T_{2}] t_{12} \ (E-TAPPTABS)$

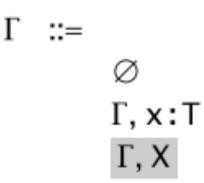


Types and Type Context



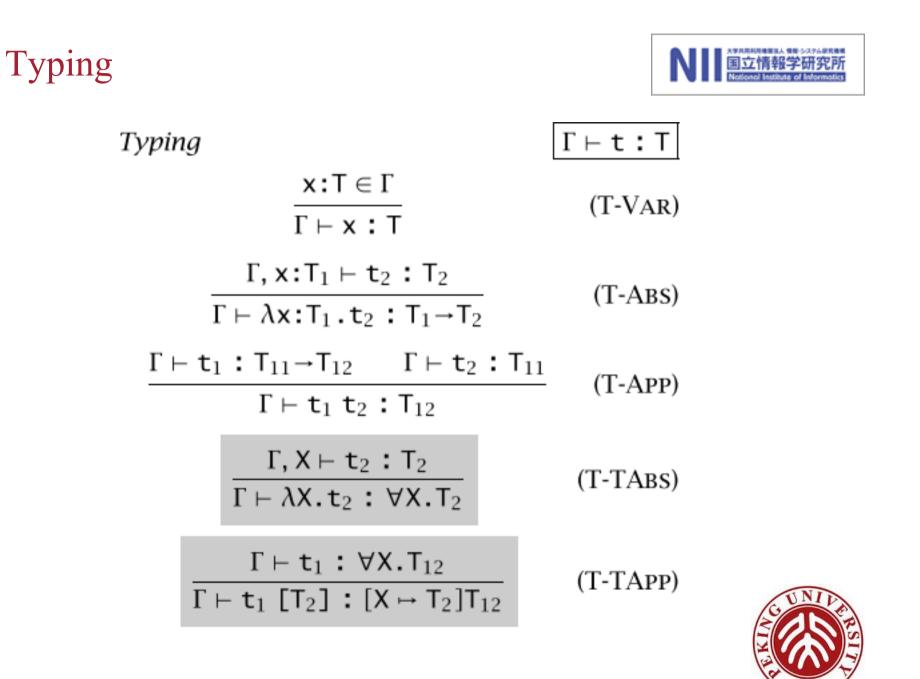


types: type variable type of functions universal type



contexts: empty context term variable binding type variable binding





Ex.: Defining Polymorphic Functions



- id = $\lambda \times \lambda \times \times \times$
 - id : $\forall X. \ X \rightarrow X$
 - id [Nat] 0 → 0
- double = $\lambda \times \lambda f: X \rightarrow X$. $\lambda a: X$. f (f a)
 - double : $\forall X. (X \rightarrow X) \rightarrow X \rightarrow X$
 - double [Nat] (λ x:Nat. succ(succ(x))) 3 → 7
 - quadruple = λX . double [X \rightarrow X] (double [X])
- selfApp = $\lambda x: \forall X.X \rightarrow X. x [\forall X.X \rightarrow X] x$ - selfApp : ($\forall X. X \rightarrow X$) $\rightarrow (\forall X. X \rightarrow X)$



Ex.: Polymorphic Lists



- nil : $\forall X$. List X
- cons : $\forall X. X \rightarrow List X \rightarrow List X$
- isnil : $\forall X$. List $X \rightarrow Bool$
- head : $\forall X$. List $X \to X$
- tail : $\forall X$. List $X \rightarrow List X$
- map: ∀X. ∀Y. (X→Y) → List X → List Y map = λX. λY. λf: X→Y. (fix (λm: (List X) → (List Y). λl: List X. if isnil [X] I then nil [Y] else cons [Y] (f (head [X] I)) (m (tail [X] I))))

IS98.

Exercise: Can you write reverse?

Ex.: Church Encoding



- Church encodings can be carried out in System F.
- CBool = $\forall X.X \rightarrow X \rightarrow X;$
 - tru = λX . λt :X. λf :X. t;
 - fls = λX . λt :X. λf :X. f;
 - and = ?
- CNat = $\forall X. (X \rightarrow X) \rightarrow X \rightarrow X$
 - c0 = λX . $\lambda s: X \rightarrow X$. $\lambda z: X$. Z
 - cl= λX . $\lambda s: X \rightarrow X$. $\lambda z: X$. s z;
 - csucc = λ n:CNat. λ X. λ s:X→X. λ z:X. s (n [X] s z)
 - cplus = λ m:CNat. λ n:CNat. λ X. λ s:X \rightarrow X. λ z:X.

m [X] s (n [X] s z)



Ex.: Encoding Lists



- List X = $\forall R. (X \rightarrow R \rightarrow R) \rightarrow R \rightarrow R$
 - nil = λX . (λR . $\lambda c: X \rightarrow R \rightarrow R$. $\lambda n: R$. n) as List X
 - cons = λX . λhd :X. λtl :List X.
 - $(\lambda R. \lambda c: X \rightarrow R \rightarrow R. \lambda n: R. c hd (tl [R] c n))$ as List X;
 - isnil = λX . λI :List X.
 - l [Bool] (λ hd:X. λ tl:Bool. false) true
 - head = λX . λI :List X.
 - $I[X] (\lambda hd:X. \lambda tl:X. hd) (diverge [X] unit)$
 - sum : List Nat \rightarrow Nat

sum = ··· definition without using fix ···?



Basic Properties of System F



Very similar to those of the simply typed λ -calculus.

Theorem [Preservation]: If $\Gamma \vdash t : T$ and $t \rightarrow t'$, then $\Gamma \vdash t' : T$.

Theorem [Progress]: If t is a closed, well-typed term, then either t is a value or there is some t' with $t \rightarrow t'$.

Theorem [Normalization]: Well-typed System F terms are normalizing.





$$erase(\lambda x:T_1.t_2) = \lambda x. erase(t_2)$$

$$erase(t_1 t_2) = erase(t_1) erase(t_2)$$

$$erase(\lambda X.t_2) = erase(t_2)$$

Theorem [Wells, 1994]: It is undecidable whether, given a closed term m of the untyped lambda-calculus, there is some well-typed term t in System F such that erase(t) = m.





$$erase_{p}(x) = x$$

$$erase_{p}(\lambda x:T_{1}.t_{2}) = \lambda x:T_{1}.erase_{p}(t_{2})$$

$$erase_{p}(t_{1}t_{2}) = erase_{p}(t_{1})erase_{p}(t_{2})$$

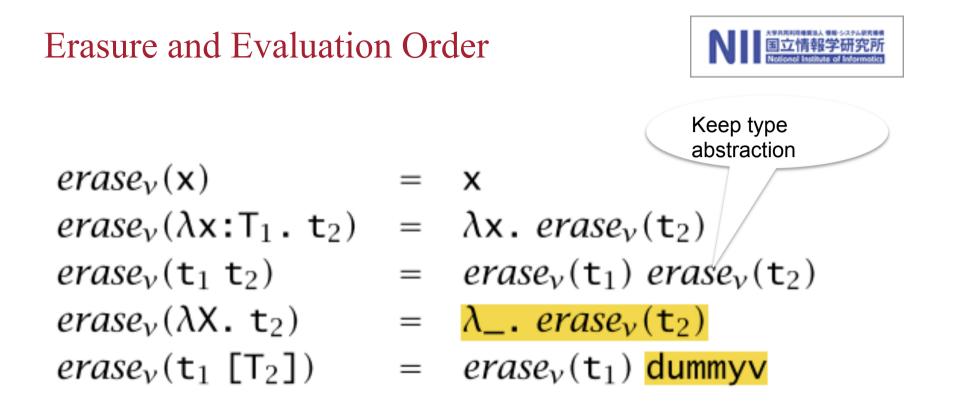
$$erase_{p}(\lambda X.t_{2}) = \lambda X.erase_{p}(t_{2})$$

$$erase_{p}(t_{1}[T_{2}]) = erase_{p}(t_{1})[]$$

Theorem [Boehm 1985, 1989]: It is undecidable whether, given a closed term s in which type applications are marked but the arguments are omitted, there is some well-typed System F term t such that $erase_{p}(t) = s$.

Type reconstruction is as hard as higher-order unification. (But many practical algorithms have been developed)





Theorem: If $erase_v(t) = u$, then either (1) both t and u are normal forms according to their respective evaluation relations, or (2) $t \rightarrow t'$ and $u \rightarrow u'$, with $erase_v(t') = u'$.



Fragments of System F



- Rank-1 (prenex) polymorphism
 - type variables may not be instantiated with polymorphic types
- Rank-2 polymorphism
 - A type is said to be of rank 2 if no path from its root to
 a ∀ quantifier passes to the left of 2 or more arrows.

Type reconstruction for ranks 2 and lower is decidable, and that for rank 3 and higher of System F is undecidable.



Parametricity



• Uniform behavior of polymorphic programs

CBool = $\forall X.X \rightarrow X \rightarrow X;$ tru = $\lambda X. \lambda t:X. \lambda f:X. t;$ fls = $\lambda X. \lambda t:X. \lambda f:X. f;$

(1) Tru and fls are the only two basic inhabitants of Cbool.

(2) Free Theorem:
 e.g., for reverse: ∀X. List X -> List X, we have
 map f . reverse = reverse . map f



Impredicativity



Definition. A definition (of a set, a type, etc.) is called "impredicative" if it involves a quan-tifier whose domain includes the very thing being defined

System F is impredicative, because the type variable X in the type

 $\mathsf{T} = \forall \mathsf{X}.\mathsf{X} {\rightarrow} \mathsf{X}$

ranges over all types, including T itself.

Russell's paradox: let $A = \{ x \mid x \text{ is not in } x \}$, then is "A in A"?



Homework



- 23.5.1 THEOREM [PRESERVATION]: If $\Gamma \vdash t : T$ and $t \rightarrow t'$, then $\Gamma \vdash t' : T$. *Proof:* Exercise [Recommended, ***].
- 23.5.2 THEOREM [PROGRESS]: If t is a closed, well-typed term, then either t is a value or else there is some t' with $t \rightarrow t'$.

Proof: Exercise [Recommended, $\star \star \star$].

