CS520, Handout 20, 9 Nov 2006

Different Presentation of Let-Polymorphism

Reading: Not in [Pierce] but compare with [Pierce, Sect 22.7]

Syntax of Types

In addition to **types**, as in [Pierce, Ch 9 and 11] and [Handouts 07, 08, 09, 18], we have **type schemes** (a special case of **universal types** in [Pierce, Ch 23]):

X	$::=$	$\mathtt{A} \mid \mathtt{B} \mid \cdots \mid \mathtt{X} \mid \mathtt{Y} \mid \cdots$	*type variables*
T	$::=$		*types*
		$\mathtt{Nat} \mid \mathtt{Bool} \mid \cdots$	*base type*
		X	*type variable*
		$T \to T$	*function type*
S	$::=$		*type schemes*
		T	*type*
		$\forall X.S$	*type scheme*

Notation: Write $\bar{X}$ for a set of types variables $\{X_1, \ldots, X_n\}$ for some $n \geqslant 0$. Write $\forall \bar{X}.S$ as an abbreviation for $\forall X_1.\cdots.\forall X_n.S$.

Convention: $\forall X_1.\forall X_2.S = \forall X_2.\forall X_1.S$.

Instantiation of type schemes: Write $S \preccurlyeq T$ iff:

1. $S = \forall X_1.\cdots.\forall X_n.T'$ for some type T'.
2. $T = [X_1 \mapsto T_1, \ldots, X_n \mapsto T_n]T'$ for some types $T_1, \ldots, T_n$.

Instantiation of type schemes: Write $S \preccurlyeq T$ iff:

1. $S = \forall X_1.\cdots.\forall X_n.T'$ for some type T'.
2. $T = [X_1 \mapsto T_1, \ldots, X_n \mapsto T_n]T'$ for some types $T_1, \ldots, T_n$.

Typing Rules

(every variable binding is type-annotated)

$$\frac{(x : \forall\bar{X}.T) \in \Gamma \qquad \forall\bar{X}.T \preccurlyeq T'}{\Gamma \vdash x : T'} \quad \text{(T-Var)}$$

$$\frac{\Gamma, x : T \vdash t : T'}{\Gamma \vdash (\lambda x : T.\, t) : T \to T'} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash t : T \to T' \qquad \Gamma \vdash t' : T}{\Gamma \vdash t\, t' : T'} \quad \text{(T-App)}$$

$$\frac{\Gamma \vdash t : T \qquad \Gamma, x : \forall\bar{X}.T \vdash t' : T' \qquad \bar{X} = FTV(T) - FTV(\Gamma)}{\Gamma \vdash \mathtt{let}\ x : (\forall\bar{X}.T) = t\ \mathtt{in}\ t' : T'} \quad \text{(T-LetPoly)}$$

Typing Rules (once more, but omitting all type-annotations)

$$\frac{(x : \forall\bar{X}.T) \in \Gamma \qquad \forall\bar{X}.T \preccurlyeq T'}{\Gamma \vdash x : T'} \quad \text{(T-VAR)}$$

$$\frac{\Gamma, x : T \vdash t : T'}{\Gamma \vdash (\lambda x.\,t) : T \to T'} \quad \text{(T-ABS}'\text{)}$$

$$\frac{\Gamma \vdash t : T \to T' \qquad \Gamma \vdash t' : T}{\Gamma \vdash t\,t' : T'} \quad \text{(T-APP)}$$

$$\frac{\Gamma \vdash t : T \qquad \Gamma, x : \forall\bar{X}.T \vdash t' : T' \qquad \bar{X} = FTV(T) - FTV(\Gamma)}{\Gamma \vdash \texttt{let}\ x = t\ \texttt{in}\ t' : T'} \quad \text{(T-LETPOLY}'\text{)}$$

Exercise Show a complete typing derivation for the term

$$\texttt{let f} = \lambda\texttt{x}.\,\lambda\texttt{y}.\,\texttt{x y}\ \texttt{in}\ \texttt{f f succ}\ 3$$

according to the rules on page 4.

Exercise Give a precise reason why the following term

$$\lambda\texttt{x}.\,\texttt{let f} = \lambda\texttt{y}.\,\texttt{x y}\ \texttt{in}\ \texttt{f f succ}\ 3$$

has **no** typing derivation according to the rules on page 4.

Exercise Consider the rule T-LETPOLY′ on page 4, where the premise $\bar{X} = FTV(T) - FTV(\Gamma)$ is now omitted. Call the adjusted rule T-LETPOLY″. Show a complete typing derivation for the term

$$\lambda\texttt{x}.\,\texttt{let f} = \lambda\texttt{y}.\,\texttt{x y}\ \texttt{in}\ \texttt{f f succ}\ 3$$

in the presence of the new rule T-LETPOLY″.

Exercise Give a precise reason why the term in the preceding exercise is wrong and should not be allowed to be used (by applying it to some value).

In preparation for writing constraint typing rules we need to ...

Extend Substitutions

Compare with the definition in [Handout 18, page 5]:

σ ::=		*substitutions*
	$\varnothing$	*empty substitution*
	$\sigma, X \mapsto T$	*type variable binding*
	$\sigma, \alpha \mapsto S$	*scheme variable binding*

Implicit in the preceding definition is the introduction of scheme variables. Scheme variables are ranged over by the metavariable α.

In preparation for writing constraint typing rules we need to ...

Extend Substitutions

Compare with the definition in [Handout 18, page 5]:

σ ::=		*substitutions*
	$\varnothing$	*empty substitution*
	$\sigma, X \mapsto T$	*type variable binding*
	$\sigma, \alpha \mapsto S$	*scheme variable binding*

Implicit in the preceding definition is the introduction of scheme variables. Scheme variables are ranged over by the metavariable α.

Note

Types do not contain scheme variables, only type variables, just as in [Handouts 18 and 19]. So, lifting the application of substitutions as extended above to types, contexts, typings, etc., is the same as in [Handout 18, page 6].

Constraint Typing Rules

An initial context $\tilde{\Gamma}$ is of the form:

$$x_1 : \alpha_1, \ldots, x_m : \alpha_m,\ x_{m+1} : X_1, \ldots, x_{m+n} : X_n$$

Constraint Typing Rules

An initial context $\tilde{\Gamma}$ is of the form:

$$x_1 : \alpha_1, \ldots, x_m : \alpha_m,\ x_{m+1} : X_1, \ldots, x_{m+n} : X_n$$

Compare with the rules in [Handout 18, p. 7] and [Handout 19, p. 7]:

$$\frac{(x : \alpha) \in \tilde{\Gamma} \qquad X \text{ fresh}}{\tilde{\Gamma} \vdash x : X \mid \{\alpha \mathrel{\dot{\preccurlyeq}} X\}} \qquad \text{(CT-VAR1)}$$

$$\frac{(x : X) \in \tilde{\Gamma}}{\tilde{\Gamma} \vdash x : X \mid \{\}} \qquad \text{(CT-VAR2)}$$

$$\frac{\tilde{\Gamma}, x : X \vdash t : T \mid C}{\tilde{\Gamma} \vdash (\lambda x : X.\, t) : X \to T \mid C} \qquad \text{(CT-ABS)}$$

$$\frac{\tilde{\Gamma} \vdash t_1 : T_1 \mid C_1 \qquad \tilde{\Gamma} \vdash t_2 : T_2 \mid C_2 \qquad X \text{ fresh}}{\tilde{\Gamma} \vdash t_1\, t_2 : X \mid C_1 \cup C_2 \cup \{T_1 \doteq T_2 \to X\}} \qquad \text{(CT-APP)}$$

$$\frac{\tilde{\Gamma} \vdash t_1 : T_1 \mid C_1 \qquad \tilde{\Gamma}, x : \alpha \vdash t_2 : T_2 \mid C_2}{\tilde{\Gamma} \vdash \mathtt{let}\ x : \alpha = t_1\ \mathtt{in}\ t_2 : T_2 \mid (C_1 \cup C_2 \cup \{\alpha \doteq_{\tilde{\Gamma}} T_1\})} \qquad \text{(CT-LETPOLY)}$$

We can also use constraint typing rules that omit type annotations of variable bindings. For this, we replace rules CT-ABS and CT-LETPOLY by the rules:

$$\frac{\tilde{\Gamma}, x : X \vdash t : T \mid C}{\tilde{\Gamma} \vdash (\lambda x.\, t) : X \to T \mid C} \qquad \text{(CT-ABS}'\text{)}$$

$$\frac{\tilde{\Gamma} \vdash t_1 : T_1 \mid C_1 \qquad \tilde{\Gamma}, x : \alpha \vdash t_2 : T_2 \mid C_2}{\tilde{\Gamma} \vdash \mathtt{let}\ x = t_1\ \mathtt{in}\ t_2 : T_2 \mid (C_1 \cup C_2 \cup \{\alpha \doteq_{\tilde{\Gamma}} T_1\})} \qquad \text{(CT-LETPOLY}'\text{)}$$

We can also use constraint typing rules that omit type annotations of variable bindings. For this, we replace rules CT-ABS and CT-LETPOLY by the rules:

$$\frac{\tilde{\Gamma}, x : X \vdash t : T \mid C}{\tilde{\Gamma} \vdash (\lambda x.\, t) : X \to T \mid C} \qquad \text{(CT-ABS}'\text{)}$$

$$\frac{\tilde{\Gamma} \vdash t_1 : T_1 \mid C_1 \qquad \tilde{\Gamma}, x : \alpha \vdash t_2 : T_2 \mid C_2}{\tilde{\Gamma} \vdash \mathtt{let}\ x = t_1\ \mathtt{in}\ t_2 : T_2 \mid (C_1 \cup C_2 \cup \{\alpha \doteq_{\tilde{\Gamma}} T_1\})} \qquad \text{(CT-LETPOLY}'\text{)}$$

3 Kinds of Constraints

1. σ satisfies the constraint $\boxed{\alpha \mathrel{\dot{\preccurlyeq}} X}$ iff $\sigma(\alpha) \preccurlyeq \sigma(X)$.
2. σ satisfies the constraint $\boxed{T \doteq T'}$ iff $\sigma(T) = \sigma(T')$.
3. σ satisfies the constraint $\boxed{\alpha \doteq_{\Gamma} T}$ iff $\sigma(\alpha) = \forall \bar{X}.\sigma(T)$ where $\bar{X} = FTV(\sigma(T)) - FTV(\sigma(\Gamma))$.

Solving Constraints: Soundness and Completeness

Assume throughout that $\tilde{\Gamma}$ is an initial context and t a term of the pure λ-calculus augmented with `let-in` such that $dom(\tilde{\Gamma}) \supseteq FV(t)$. For definiteness, take t to be without any type annotations.

Lemma (compare with [Handout 18, p. 10])
There is a unique type T and a unique constraint set C such that $\tilde{\Gamma} \vdash t : T \mid C$ is derivable according to the constraint typing rules (pages 6 and 7).

Solving Constraints: Soundness and Completeness

Assume throughout that $\tilde{\Gamma}$ is an initial context and t a term of the pure λ-calculus augmented with `let-in` such that $dom(\tilde{\Gamma}) \supseteq FV(t)$. For definiteness, take t to be without any type annotations.

Lemma (compare with [Handout 18, p. 10])
There is a unique type T and a unique constraint set C such that $\tilde{\Gamma} \vdash t : T \mid C$ is derivable according to the constraint typing rules (pages 6 and 7).

Proof of the lemma is based on the implementation (left to you, but see Handout 18, p. 8, for a similar implementation) of a function $\texttt{constraints}(\tilde{\Gamma}, t)$ defined as:

$$\texttt{constraints}(\tilde{\Gamma}, t) = \begin{cases} (T, C) & \text{if } dom(\tilde{\Gamma}) \supseteq FV(t)\text{, in which case} \\ & T \text{ is unique type and } C \text{ is unique constraint set} \\ & \text{set such that } \tilde{\Gamma} \vdash t : T \mid C, \\ \mathrm{error} & \text{if } dom(\tilde{\Gamma}) \not\supseteq FV(t). \end{cases}$$

Solving Constraints: Soundness and Completeness

Definition (compare with [Handout 18, p. 11])
A substitution σ is a solution for $(\tilde{\Gamma}, t)$ if there is a type T such that $\sigma\tilde{\Gamma} \vdash t : T$ is derivable. In words, "there is a solution for $(\tilde{\Gamma}, t)$" means "t is typable".

Note

We write $\sigma\tilde{\Gamma} \vdash t : T$ instead of $\sigma\tilde{\Gamma} \vdash \sigma t : T$ because t is not type-annotated.

Solving Constraints: Soundness and Completeness

Definition (compare with [Handout 18, p. 11])
A substitution σ is a solution for $(\tilde{\Gamma}, t)$ if there is a type T such that $\sigma\tilde{\Gamma} \vdash t : T$ is derivable. In words, "there is a solution for $(\tilde{\Gamma}, t)$" means "t is typable".

Note

We write $\sigma\tilde{\Gamma} \vdash t : T$ instead of $\sigma\tilde{\Gamma} \vdash \sigma t : T$ because t is not type-annotated.

Soundness Theorem (compare with soundness in [Handout 18, p. 12])
If substitution σ is a solution of $\texttt{constraints}(\tilde{\Gamma}, t)$ then σ is a solution of $(\tilde{\Gamma}, t)$.

Solving Constraints: Soundness and Completeness

Definition (compare with [Handout 18, p. 11])
A substitution σ is a solution for $(\tilde{\Gamma}, t)$ if there is a type T such that $\sigma\tilde{\Gamma} \vdash t : T$ is derivable. In words, "there is a solution for $(\tilde{\Gamma}, t)$" means "t is typable".

Note

We write $\sigma\tilde{\Gamma} \vdash t : T$ instead of $\sigma\tilde{\Gamma} \vdash \sigma t : T$ because t is not type-annotated.

Soundness Theorem (compare with soundness in [Handout 18, p. 12])
If substitution σ is a solution of $\mathtt{constraints}(\tilde{\Gamma}, t)$ then σ is a solution of $(\tilde{\Gamma}, t)$.

Completeness Theorem (compare with completeness in [Handout 18, p. 12])
If substitution σ is a solution for $(\tilde{\Gamma}, t)$ and $dom(\sigma) \subseteq FTV(\tilde{\Gamma})$ then there is a substitution σ' which is a solution of $\mathtt{constraints}(\tilde{\Gamma}, t)$ with $\sigma' \supseteq \sigma$.

Finding Solutions For Constraint Sets

The unification theory of [Handout 18, pp 13 and on] is no longer sufficient. We need a new way of unifying constraints . . .