COMP 523: Language-based security
Assignment 4 (100 points total)

Prof. B. Pientka
McGill University

Due: Thursday, 16 Nov 2017 at 2:35pm

1 Hereditary substitutions (60 points)

In class, we discussed the hereditary substitution operation, a substitution operation which will preserve normal forms. This provides an elegant way of designing a core dependently calculus syntactically which precisely defines well-typed normal terms. A substantial benefit of such a design is that we do not need to prove normalization via logical relations – instead, by design, we only characterize normal terms.

In this question, you are asked to extend the definition of hereditary substitution to Σ-types.

Let us review definition of a dependently typed lambda-calculus where we only admit normal forms (i.e. terms not containing any redex). While Π-types describe the dependent function space, Σ-types describe dependent (cross) products. An example of a dependent (cross) products is $\Sigma x{:}\mathsf{term}.\mathsf{oft}\ x\ \mathsf{nat}$.

Types	A, B	$:=$	$P \mid \Pi x{:}A.B \mid \Sigma x{:}A.B$
Atomic type	P	$:=$	$a\ M_1 \ldots M_n$
Normal Terms	M	$:=$	$\lambda x.M \mid (M_1, M_2) \mid R$
Neutral Terms	R	$:=$	$x \mid R\ N \mid \mathsf{fst}\ R \mid \mathsf{snd}\ R$

We review the typing rules for this language below using two judgments:

$\Psi \vdash M \Leftarrow A$	Check normal term M against type A
$\Psi \vdash R \Rightarrow A$	Synthesize for neutral term R type A

Data-level normal terms

$$\frac{\Psi, x{:}A \vdash M \Leftarrow B}{\Psi \vdash \lambda x.\, M \Leftarrow \Pi x{:}A.B} \qquad \frac{\Psi \vdash M_1 \Leftarrow A \qquad \Psi \vdash M_2 \Leftarrow [M_1/x]^a_A B}{\Psi \vdash (M_1, M_2) \Leftarrow \Sigma x{:}A.B} \qquad \frac{\Psi \vdash R \Rightarrow P' \qquad P' = P}{\Psi \vdash R \Leftarrow P}$$

Object-level neutral terms

$$\frac{x{:}A \in \Psi}{\Psi \vdash x \Rightarrow A} \qquad \frac{\Psi \vdash R \Rightarrow \Sigma x{:}A.B}{\Psi \vdash \mathsf{fst}\ R \Rightarrow A} \qquad \frac{\Psi \vdash R \Rightarrow \Sigma x{:}A.B}{\Psi \vdash \mathsf{snd}\ R \Rightarrow [\mathsf{fst}\ R/x]^a_A(B)} \qquad \frac{\Psi \vdash R \Rightarrow \Pi x{:}A.B \quad \Psi \vdash N \Leftarrow A}{\Psi \vdash R\ N \Rightarrow [N/x]^a_A(B)}$$

While our grammar does only enforce that our terms are in β-normal form, i.e. they contain no redex of the form $(\lambda x.\, M)\ N$ our typing rules also enforce that terms are η-long, i.e. whenever we have a term x of function type $\Pi x{:}A.B$, then we must have expanded it to $\lambda y.\, .x\ y$. This is enforced in the typing rules by allowing the coercion between normal and neutral terms to only happen when R is of atomic type P.

Data-level normal terms

$[M/x]^n_\alpha(\lambda y.\, N) = \lambda y.\, N'$ where $N' = [M/x]^n_\alpha(N)$ choosing $y \notin FV(M)$, and $y \neq x$

$$
\begin{array}{lll}
[M/x]^n_\alpha(R) & = M' & \text{if } [M/x]^r_\alpha(R) = M' : \alpha' \\
[M/x]^n_\alpha(R) & = R' & \text{if } [M/x]^r_\alpha(R) = R' \\
[M/x]^n_\alpha(N) & \text{fails} & \text{otherwise}
\end{array}
$$

Data-level neutral terms

$$
\begin{array}{lll}
[M/x]^r_\alpha(x) & = M : \alpha & \\
[M/x]^r_\alpha(y) & = y & \text{if } y \neq x \\
[M/x]^r_\alpha(R\ N) & = R'\ N' & \text{where } R' = [M/x]^r_\alpha(R) \text{ and } N' = [M/x]^n_\alpha(N) \\
[M/x]^r_\alpha(R\ N) & = M'' : \beta & \text{if } [M/x]^r_\alpha(R) = \lambda y.\, M' : \alpha_1 \to \beta \text{ where} \\
 & & \alpha_1 \to \beta \leq \alpha \text{ and } N' = [M/x]^n_\alpha(N) \text{ and } M'' = [N'/y]^n_{\alpha_1}(M') \\
 & & \\
[M/x]^r_\alpha(R) & \text{fails} & \text{otherwise}
\end{array}
$$

Figure 1: Hereditary substitution

In the formal development, it is simpler if we can stick to non-dependent types. We therefore first define type approximations α and an erasure operation $()^-$ that removes dependencies. Before applying any hereditary substitution $[M/x]^a_A(B)$ we first erase dependencies to obtain $\alpha = A^-$ and then carry out the hereditary substitution formally as $[M/x]^a_\alpha(B)$. A similar convention applies to the other forms of hereditary substitutions. Types relate to type approximations via an erasure operation $()^-$ which we overload to work on types.

Type approximations $\alpha, \beta \quad ::= \quad a \mid \alpha \to \beta \mid \alpha \times \beta$

$$
\begin{array}{lcl}
(a\ N_1 \ldots N_n)^- & = & a \\
(\Pi x{:}A \ldots B)^- & = & A^- \to B^-
\end{array}
$$

We can define $[M/x]^n_\alpha(N)$, $[M/x]^r_\alpha(R)$, and $[M/x]^s_\alpha(\sigma)$ by nested induction, first on the structure of the type approximation α and second on the structure of the objects N, R and σ. In other words, we either go to a smaller type approximation (in which case the objects can become larger), or the type approximation remains the same and the objects become smaller. The following hereditary substitution operations are defined in Figure 1.

$$
\begin{array}{lcll}
[M/x]^n_\alpha(N) & = & N' & \text{Normal terms } N \\
[M/x]^r_\alpha(R) & = & R' \text{ or } M' : \alpha' & \text{Neutral terms } R
\end{array}
$$

We write $\alpha \leq \beta$ and $\alpha < \beta$ if α occurs in β (as a proper subexpression in the latter case). If the original term is not well-typed, a hereditary substitution, though terminating, cannot always return a meaningful term. We formalize this as failure to return a result. However, on well-typed terms, hereditary substitution will always return well-typed terms. This substitution operation can be extended to types for which we write $[M/x]^a_\alpha(A)$.

10 points Extend the type erasure and approximate types for Σ-types.

15 points Extend the hereditary substitution operation for Σ-types, i.e. define $[M/x]^r_\alpha(N_1, N_2)$, $[M/x]^r_\alpha(\mathsf{fst}\, N)$, and $[M/x]^r_\alpha(\mathsf{snd}\, N)$.

35 points Prove termination of hereditary substitution:

1. If $[M/x]^r_\alpha(R) = M' : \alpha'$ then $\alpha' \leq \alpha$
2. $[M/x]^*_\alpha(\_)$ terminates, either by returning a result or failing after a finite number of steps.

The first part is proven by induction on the definition of all operations. The second part follows by a nested induction, first on the structure of α and second on the structure of the term we apply hereditary substitution to.

2 Bi-directional typing(40 points)

Bidirectional typing rests on the idea of that we can classify terms into "intro" forms and "elim" forms. "intro" forms are those terms which introduce a type, while "elim" forms are those which eliminate a type. Bidirectional typing allows us to minimize type annotations in lambda-abstraction and other constructs; but we pay a minimal price; on the top-level the programmer needs to specify the type of functions, since we only check functions against their type. In class and in the first question, we used bidirectional typing to characterize normal forms using the two judgments:

$\Psi \vdash M \Leftarrow A$ Check normal term M against type A
$\Psi \vdash R \Rightarrow A$ Synthesize for neutral term R type A

The question, we investigate now is the soundness and completeness of bidirectional typing with respect to the (undirected) typing rules we have seen earlier in class. To simplify the matter, we consider only the simply typed case.

Data-level normal terms

$$\frac{\Psi, x{:}A \vdash M \Leftarrow B}{\Psi \vdash \lambda x.\, M \Leftarrow A \to B} \qquad \frac{\Psi \vdash M_1 \Leftarrow A \quad \Psi \vdash M_2 \Leftarrow B}{\Psi \vdash (M_1, M_2) \Leftarrow A \times B} \qquad \frac{\Psi \vdash R \Rightarrow P' \quad P' = P}{\Psi \vdash R \Leftarrow P}$$

Object-level neutral terms

$$\frac{x{:}A \in \Psi}{\Psi \vdash x \Rightarrow A} \quad \frac{\Psi \vdash R \Rightarrow A \times B}{\Psi \vdash \mathsf{fst}\, R \Rightarrow A} \quad \frac{\Psi \vdash R \Rightarrow A \times B}{\Psi \vdash \mathsf{snd}\, R \Rightarrow B} \qquad \frac{\Psi \vdash R \Rightarrow A \to B \quad \Psi \vdash N \Leftarrow A}{\Psi \vdash R\, N \Rightarrow B}$$

Since we do not only want to characterize normal forms, we also add a rule for type annotations to allow normal terms to occur inside neutral terms, provided they are annotated with their type.

$$\frac{\Psi \vdash M \Leftarrow A}{\Psi \vdash (M : A) \Rightarrow A}$$

Next we recall the undirected typing rules given as a type assignment system.

Typing rules for dependently typed lambda-terms (Type assignment - undirected)

$$\frac{\Psi, x{:}A \vdash M : B}{\Psi \vdash \lambda x.\, M : A \to B} \qquad \frac{\Psi \vdash M_1 : A \quad \Psi \vdash M_2 : B}{\Psi \vdash (M_1, M_2) : A \times B}$$

$$\frac{x{:}A \in \Psi}{\Psi \vdash x : A} \quad \frac{\Psi \vdash M : A \times B}{\Psi \vdash \mathsf{fst}\, M : A} \quad \frac{\Psi \vdash M : A \times B}{\Psi \vdash \mathsf{snd}\, M : B} \qquad \frac{\Psi \vdash M : A \to B \quad \Psi \vdash N : A}{\Psi \vdash M\ N : B}$$

Our goal is to show that bi-directional typing is sound and complete.

20 pt Prove soundness. If a term type-checks in the bi-directional setting, there exists a derivation in the undirected type assignment system. Show a few key cases.

20 pt Prove completeness. If a term type-checks in the undirected type assignment system, then there exists a typing derivation in the bi-directional system.