Homework 1: Substitution and Combinators

15-814: Types and Programming Languages
TA: Carlo Angiuli (cangiuli@cs.cmu.edu)

Out: 9/17/13
Due: 9/24/13 (end of day)

Welcome to 15-814's first homework assignment! The goal of this assignment is to develop and practice the basic techniques necessary for the study of programming languages, in particular, *rule induction*.

Please submit your work as a PDF to `cangiuli@cs.cmu.edu` and include the phrase "15-814 Homework 1" in the subject line of your email.

1 The substitution theorem

As a reminder, here is the presentation of the λ-calculus:

$$\begin{array}{lll} \Gamma & ::= & \cdot \mid \Gamma, x\ \mathsf{ok} \\ M, N & ::= & x \mid M\ N \mid \lambda x.M \end{array}$$

with these inductive rules for a well-defined term:

$$\frac{}{\Gamma, x\ \mathsf{ok} \vdash x\ \mathsf{ok}}\ (\mathsf{var}) \quad \frac{\Gamma \vdash M\ \mathsf{ok} \quad \Gamma \vdash N\ \mathsf{ok}}{\Gamma \vdash M\ N\ \mathsf{ok}}\ (\mathsf{ap}) \quad \frac{\Gamma, x\ \mathsf{ok} \vdash M\ \mathsf{ok}}{\Gamma \vdash \lambda x.M\ \mathsf{ok}}\ (\mathsf{abs})$$

Remark 1. *In the rules above, Γ is a (unordered) multiset; in particular, $(x\ \mathsf{ok}, y\ \mathsf{ok})$ is precisely the same context as $(y\ \mathsf{ok}, x\ \mathsf{ok})$. For example, one can conclude by the* var *rule that $x\ \mathsf{ok}, y\ \mathsf{ok} \vdash x\ \mathsf{ok}$.*

Remark 2. *In the rules above, and for this entire course, terms are only ever considered up to α-equivalence (as defined in class). So the judgment $\cdot \vdash \lambda x.x\ \mathsf{ok}$ is precisely the same as the judgment $\cdot \vdash \lambda y.y\ \mathsf{ok}$, and you may freely α-vary terms in this fashion whenever needed.*

Definition 1. *We say the variable x is in the context Γ, written $x \in \mathsf{dom}(\Gamma)$, when:*

$$\begin{array}{lll} x \in \mathsf{dom}(\cdot) & \iff & \textit{never} \\ x \in \mathsf{dom}(\Gamma, y\ \mathsf{ok}) & \iff & x = y \textit{ or } x \in \mathsf{dom}(\Gamma) \end{array}$$

Definition 2. *We say the variable x is* free *in the term M, written $x \in \mathsf{FV}(M)$, when:*

$$\begin{array}{lll} x \in \mathsf{FV}(y) & \iff & x = y \\ x \in \mathsf{FV}(M\ N) & \iff & x \in \mathsf{FV}(M) \textit{ or } x \in \mathsf{FV}(N) \\ x \in \mathsf{FV}(\lambda y.M) & \iff & x \neq y \textit{ and } x \in \mathsf{FV}(M) \end{array}$$

Definition 3. *The* substitution *relation $[N/x]M = M'$, denoting that M' is the result of substituting N for all occurrences of x in M, is defined by the following rules:*

$$\frac{}{[N/x]x = N}\ (\mathsf{var\ subst}_1) \quad \frac{x \neq y}{[N/x]y = y}\ (\mathsf{var\ subst}_2)$$

$$\frac{[N/x]M_1 = M_1' \quad [N/x]M_2 = M_2'}{[N/x]M_1\ M_2 = M_1'\ M_2'}\ (\mathsf{ap\ subst}) \quad \frac{y \neq x \quad y \notin \mathsf{FV}(N) \quad [N/x]M = M'}{[N/x]\lambda y.M = \lambda y.M'}\ (\lambda\ \mathsf{subst})$$

Note that the side-conditions on the free variables in the rule for the λ-abstraction make substitution a partial function on raw terms, but it is in fact a total function on α equivalence classes of terms.

Task 1. *What is the result of the substitution* $[(\lambda x.y)/z](\lambda y.(\lambda z.z)z)$*?*

We can think of contexts as lists of hypotheses: the judgment $x\ \mathsf{ok} \vdash M\ \mathsf{ok}$ means that, if we assume $x\ \mathsf{ok}$, then $M\ \mathsf{ok}$. In the derivation of this judgment, every occurrence of the var rule makes use of one of these assumptions—certainly, if we assume x is well-formed, then we should be able to conclude that x is well-formed.

The *substitution theorem* states that we can discharge the assumption that x is well-formed by replacing x with an actual well-formed term N. Then, each time we would make use of our assumption $x\ \mathsf{ok}$, we can instead defer to our proof that $N\ \mathsf{ok}$.

In order to prove it, we will need the following lemma, known as *weakening*.

Task 2 (Weakening theorem)**.** *Show that if* $\Gamma \vdash M\ \mathsf{ok}$*, then* $\Gamma, x\ \mathsf{ok} \vdash M\ \mathsf{ok}$ *for any* $x \notin \mathsf{dom}(\Gamma)$*. Then, show that the derivation of* $\Gamma, x\ \mathsf{ok} \vdash M\ \mathsf{ok}$ *which you constructed has the same shape*[1] *as the original derivation of* $\Gamma \vdash M\ \mathsf{ok}$*.*

Task 3 (Substitution theorem)**.** *Show that if* $\Gamma, x\ \mathsf{ok} \vdash M\ \mathsf{ok}$ *and* $\Gamma \vdash N\ \mathsf{ok}$*, then* $\Gamma \vdash [N/x]M\ \mathsf{ok}$*.*

2 Combinator calculi

We define β-reduction $M \to_\beta N$ on λ-terms as:

$$\frac{M \to_\beta M'}{M\,N \to_\beta M'\,N}\ (\beta\mathsf{ap}_1) \quad \frac{N \to_\beta N'}{M\,N \to_\beta M\,N'}\ (\beta\mathsf{ap}_2) \quad \frac{}{(\lambda x.M)\,N \to_\beta [N/x]M}\ (\beta)$$

The first two rules allow for reductions in an application. The last rule substitutes N for x in the body M of a λ-abstraction.

Surprisingly, the power of the λ-calculus does not depend on variable binding. It is possible to formulate calculi that, like λ-calculus, support Turing-complete computation with first-class, higher-order functions, but without any native support for variable binding. (Avoiding any notion of binding means that we need not implement capture-avoiding substitution, but as we will soon see, these calculi are less user-friendly than λ-calculus.)

In this exercise, we will investigate one such calculus, the SKI combinator calculus. The calculus includes three combinators: **I**, **K** and **S** (so named for historical reasons), and combinator application:

$$E \quad ::= \quad \mathbf{I} \mid \mathbf{K} \mid \mathbf{S} \mid E_1\,E_2$$

We define combinator reduction, written $E_1 \to_{\mathbf{SKI}} E_2$ as follows (where X, Y, Z stand for combinator terms):

$$\frac{}{\mathbf{I}\,X \to_{\mathbf{SKI}} X}\ (\mathbf{I}) \quad \frac{}{\mathbf{K}\,X\,Y \to_{\mathbf{SKI}} X}\ (\mathbf{K}) \quad \frac{}{\mathbf{S}\,X\,Y\,Z \to_{\mathbf{SKI}} (X\,Z)\,(Y\,Z)}\ (\mathbf{S})$$

$$\frac{E_1 \to_{\mathbf{SKI}} E_1'}{E_1\,E_2 \to_{\mathbf{SKI}} E_1'\,E_2}\ (\mathbf{SKI}\mathsf{ap}_1) \quad \frac{E_2 \to_{\mathbf{SKI}} E_2'}{E_1\,E_2 \to_{\mathbf{SKI}} E_1\,E_2'}\ (\mathbf{SKI}\mathsf{ap}_2)$$

We then define *iteration*, or *Kleene closure*, of combinator reduction in the standard way:

$$\frac{}{E \to^*_{\mathbf{SKI}} E} \qquad \frac{E \to_{\mathbf{SKI}} E' \quad E' \to^*_{\mathbf{SKI}} E''}{E \to^*_{\mathbf{SKI}} E''}$$

[1] By "same shape" we mean, for example, that the two derivations are isomorphic as trees. You do not need to make this notion precise, but it should be possible to argue convincingly.

Intuitively, **I** is the *identity* function; **K** generates *constant* functions: given a term, **K** returns a function that returns that term for any argument; **S** is "augmented" application: **S** propagates its third argument to its first and second arguments (via application), before applying the first to the second.

Our goal is to define a way to compile a λ-term to a term in the SKI calculus. The obvious difficulty is that we must somehow compile away all variables and variable bindings. We do so in two steps. First, we extend the SKI calculus to include variables ($E ::= \ldots \mid x$) and devise a way of simulating binding in the SKI calculus. Then, we write a translation from λ terms to SKI-terms-with-simulated-binding.

This simulation of binding is achieved by *bracket abstraction*, written $[x]E$. E stands for a SKI term and x is a variable that may appear in E. Here, $[x]E$ yields a SKI term that does not contain x and, when applied to a term E_2, essentially substitutes E_2 for all occurrences of x in E.

You will define bracket abstraction in Task 5, and in Task 6 prove that it indeed works this way. Notice that E cannot itself contain bracket abstractions (because they are not part of the syntax of E), so there are no concerns with capture-avoiding substitution—all variables in E are free.

Given bracket abstraction, the compilation function is straightforward: $(\cdot)^*$ takes a λ-term and compiles it to a SKI term (making use of bracket abstraction):

$$\begin{array}{lcl} x^* & = & x \\ (M\,N)^* & = & M^*\,N^* \\ (\lambda x.M)^* & = & [x]M^* \end{array}$$

Task 4. *Define the substitution operation $[E/x]E'$ for terms from the SKI calculus extended with variables. Be sure to distinguish the case when the variable in question matches that being abstracted and the case when it differs.*

Task 5. *Now, define bracket abstraction inductively over the SKI term E. The difficult case is:*

$$[x]E \quad = \quad \mathbf{K}\,E \quad \textit{for } x \notin \mathsf{FV}_{\mathbf{SKI}}(E)$$

Complete the definition of bracket abstraction by providing the cases for all other E.
(Hint) *Your definition of the bracket abstraction should closely resemble your definition of substitution.*

We must now check that this translation is "correct," in the sense that the combinator term E that results from the translation of a λ-term M can simulate the reduction steps that can be performed by M.[2]

You may freely use the following lemmas. Make sure you understand why they are true, but you need not prove them.

Lemma 2.1. *If $E \to^*_{\mathbf{SKI}} E'$ and $E' \to^*_{\mathbf{SKI}} E''$ then $E \to^*_{\mathbf{SKI}} E''$.*

Lemma 2.2. *If $E \to^*_{\mathbf{SKI}} E'$ then for any E'' we have $E\,E'' \to^*_{\mathbf{SKI}} E'\,E''$.*

Lemma 2.3. *If $E \to^*_{\mathbf{SKI}} E'$ then for any E'' we have $E''\,E \to^*_{\mathbf{SKI}} E''\,E'$.*

We begin by first proving the correctness of bracket abstraction, that is, the application of a bracket abstraction reduces to a substitution.

Task 6. *Show that, for any E and E', $([x]E)\,E' \to^*_{\mathbf{SKI}} [E'/x]E$.*
(Hint) *Proceed by induction on the structure of E.*

[2]Technically, you will only be proving that the compilation is *complete*, which is generally only considered one part of correctness.

We can now prove our main result. You may assume that $([N/x]M)^* = [N^*/x]M^*$.

Task 7. *Show that if* $M \to_\beta N$*, then* $M^* \to^*_{\mathbf{SKI}} N^*$*.*

Task 8. *It turns out that the* **I** *combinator is actually redundant, in that its reduction behavior can be mimicked by application,* **S** *and* **K***. Produce a combinator term* **T** *that only makes use of* **S**, **K** *and application that can simulate the behavior of* **I***, in the following way:*

$$\mathbf{T}\, X \to^*_{\mathbf{SKI}} X$$