CS 441 Programming Languages
Assignment #6

Due: Monday, 24 November 2008 (by midnight)

Preamble

In this assignment, there is a programming part and a theory part. You may do the programming part either individually or in pairs. If you work in pairs, each member of the pair will receive the same grade for the program. When you email your program files to Rob (rdockins@cs.princeton.edu), you should include in the email the names of all who contributed to the program.

Each person must write up and hand in the answers to the theory questions individually. You may toss around ideas with your classmates about the problems but you need to write up the end results yourself.

You have two weeks to complete the assignment. (Try not to leave it all to the very end!)

A Language with Pattern Matching

Programming languages like ML have pattern matching syntax that makes it easy to deconstruct and use complex data structures. We will take a look at pattern matching in a language with tuples and sum types. Here is the language:

Types	τ	::=	$\texttt{int} \mid \tau_1 \to \tau_2 \mid \texttt{unit} \mid \tau_1 * \tau_2 \mid \tau_1 + \tau_2$
Values	v	::=	$n \mid \lambda x{:}\tau.e \mid () \mid (v_1, v_2) \mid \texttt{in}_1[\tau]\ v \mid \texttt{in}_2[\tau]\ v$
Patterns	p	::=	$x \mid \_\_ \mid () \mid (p_1, p_2) \mid \texttt{in}_1\ p \mid \texttt{in}_2\ p$
Case Branches	bs	::=	$\_\_ \Rightarrow e \mid p \Rightarrow e \parallel bs$
Expressions	e	::=	$x \mid v \mid e_1 + e_2 \mid e_1\ e_2 \mid (e_1, e_2) \mid \texttt{in}_1[\tau]\ e \mid \texttt{in}_2[\tau]\ e \mid \texttt{case}\ e\ \texttt{of}\ (bs)$

(Comment: notice that $e_1 + e_2$ is syntax for the addition operation ($2 + 2 = 4$) whereas $\tau_1 + \tau_2$ is syntax for a sum type, something completely different.)

Here is a simple example program that builds a pair of pairs of integers and then passes that data structure to a function that uses pattern matching to deconstruct the term and add all the numbers up:

```
(λx:(int*int)*(int*int).
   case x of
      ((x1,x2),(x3,x4)) => x1 + x2 + x3 + x4
   || __ => 0
) ((6,18),(23,26))
```

Here is another example that ignores the values in the second pair using the "wildcard" (ie, underscore) pattern:

```
(λx:(int*int)*(int*int).
   case x of
      ((x1,x2),__) => x1 + x2
   || __ => 0
) ((6,18),(23,26))
```

Below is an example function that uses a sum pattern and the "unit" pattern. The type τ below is $(int * int) + (int * (unit + unit))$.

```
(λx:τ.
   case x of
      in1(x1,x2) => x1 + x2
   || in2(x1,in1()) => x1 + 10
   || in2(x1,in2()) => x1 + 100
   || __ => 0)
```

Assuming the function above is called f and `bool` is an abbreviation for the type `unit + unit` then we can use f as follows:

```
f (in1[τ] (1,2))               (* result is 3 *)
f (in2[τ] (1,in1[bool]()))     (* result is 11 *)
f (in2[τ] (1,in2[bool]()))     (* result is 101 *)
```

You will notice that each of the examples above has a "default" branch with the form $\_\_ \Rightarrow e$. That default branch will match any value that doesn't match one of the previous patterns. For example, here is another function g:

```
(λx:τ.
   case x of
      in2(x1,in1()) ==> x1 + 10
   || __ => 0)
```

Applying g to each of the arguments we used for f gives these results:

```
g (in1[τ] (1,2))               (* default branch matches; result = 0 *)
g (in2[τ] (1,in1[bool]()))     (* result is 11 *)
g (in2[τ] (1,in2[bool]()))     (* default branch matches; result = 0 *)
```

Operational Semantics

Appendix A contains the complete set of operational rules for the language (except for those you are asked to come up with yourselves). The special new rules and new judgement forms involved with pattern matching are presented below. The first new rule is a normal sort of "search rule:"

$$\frac{e \longrightarrow e'}{\texttt{case}\ e\ \texttt{of}\ (bs) \longrightarrow \texttt{case}\ e'\ \texttt{of}\ (bs)}\ \text{(Ecase1)}$$

Once we have reduced the primary argument of the case expression to a value, we try to match that value against the first pattern in the case expression. If it matches, we take the first branch. If it doesn't match, we throw the brach away. The last (default) branch will always match because it uses the wildcard pattern. To define "matching," we will use a new kind of judgement that has the form: $v \in p \rightsquigarrow S$. This judgement is valid when the value v matches the pattern p. S is a substitution of values (contained inside v) for corresponding variables from the pattern p. The substitution is applied to the expression in the case

statement branch that matches. (The notation $S(e)$ means that substitution S is applied to expression e.) A second new kind of judgement $v \notin p$ defines the conditions under which value v does not match pattern p. Here are the other operational rules for the case statement (note, names of rules, which you can refer to in your proofs, appear in parentheses beside each rule):

$$\frac{}{\texttt{case}\ v\ \texttt{of}\ (\_\_\ \Rightarrow\ e) \longrightarrow e}\ \text{(Ecase2)}$$

$$\frac{v \in p \rightsquigarrow S}{\texttt{case}\ v\ \texttt{of}\ (p\ \Rightarrow\ e\ ||\ bs) \longrightarrow S(e)}\ \text{(Ecase3)}$$

$$\frac{v \notin p}{\texttt{case}\ v\ \texttt{of}\ (p\ \Rightarrow\ e\ ||\ bs) \longrightarrow \texttt{case}\ v\ \texttt{of}\ (bs)}\ \text{(Ecase4)}$$

Here are the rules for matching patterns. As usual we write $[v/x]$ for the substitution that replaces x with v. We write "$\cdot$" for the empty substitution. We write $S_1 \circ S_2$ for the substitution composed of both S_1 and S_2 (S_1 and S_2 are required to substitute values for different sets of variables).

$$\frac{}{v \in x \rightsquigarrow [v/x]}\ \text{(EPvar)} \qquad \frac{}{v \in \_\_ \rightsquigarrow \cdot}\ \text{(EPwild)}$$

$$\frac{}{() \in () \rightsquigarrow \cdot}\ \text{(EPunit)} \qquad \frac{v_1 \in p_1 \rightsquigarrow S_1 \quad v_2 \in p_2 \rightsquigarrow S_2}{(v_1, v_2) \in (p_1, p_2) \rightsquigarrow S_1 \circ S_2}\ \text{(EPpair)}$$

$$\frac{v \in p \rightsquigarrow S}{\texttt{in}_1[\tau]\ v \in \texttt{in}_1\ p \rightsquigarrow S}\ \text{(EPin1)} \qquad \frac{v \in p \rightsquigarrow S}{\texttt{in}_2[\tau]\ v \in \texttt{in}_2\ p \rightsquigarrow S}\ \text{(EPin2)}$$

I am leaving out the definition of $v \notin p$. It will be your job to define it (see below for questions).

Typing Rules

The standard typing rules for addition, functions, pair and sum type introduction don't change when we add pattern matching. Naturally, however, there are new rules for the case statements and for type checking patterns. In addition to the standard type checking judgement with form $\Gamma \vdash e : \tau$, we define two other new judgement forms. The first has the form $\Gamma \vdash bs : \tau_1 \Longrightarrow \tau_2$. It checks that the case branches bs all have patterns p for values with type τ_1 and expression bodies e that return values with type τ_2. The second new judgement form is $\vdash p : \tau \Longrightarrow \Gamma$. This second judgement checks that p is a pattern for values with type τ and that it includes variables that match values with types given by Γ. Here are the new rules:

Judgement form: $\boxed{\Gamma \vdash e : \tau}$

$$\frac{\Gamma \vdash e : \tau_1 \qquad \Gamma \vdash bs : \tau_1 \Longrightarrow \tau_2}{\Gamma \vdash \texttt{case}\ e\ \texttt{of}\ (bs) : \tau_2}\ \text{(Tcase)}$$

Judgement form: $\boxed{\vdash p : \tau \Longrightarrow \Gamma}$

$$\frac{}{\vdash x : \tau \Longrightarrow x{:}\tau}\ \text{(TPvar)} \qquad \frac{}{\vdash \_\_ : \tau \Longrightarrow \cdot}\ \text{(TPwild)}$$

$$\frac{}{\vdash () : \texttt{unit} \Longrightarrow \cdot}\ \text{(TPunit)} \qquad \frac{\vdash p_1 : \tau_1 \Longrightarrow \Gamma_1 \qquad \vdash p_2 : \tau_2 \Longrightarrow \Gamma_2 \qquad \mathrm{Dom}(\Gamma_1) \cap \mathrm{Dom}(\Gamma_2) = \emptyset}{\vdash (p_1, p_2) : \tau_1 * \tau_2 \Longrightarrow \Gamma_1, \Gamma_2}\ \text{(TPpair)}$$

$$\frac{\vdash p : \tau_1 \Longrightarrow \Gamma_1}{\vdash \texttt{in}_1\ p : \tau_1 + \tau_2 \Longrightarrow \Gamma_1}\ \text{(TPin1)} \qquad \frac{\vdash p : \tau_2 \Longrightarrow \Gamma_2}{\vdash \texttt{in}_2\ p : \tau_1 + \tau_2 \Longrightarrow \Gamma_2}\ \text{(TPin2)}$$

Judgement form: $\boxed{\Gamma \vdash bs : \tau_1 \Longrightarrow \tau_2}$

$$\frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \_ \ \Rightarrow\ e : \tau_1 \Longrightarrow \tau_2}\ \text{(Tbs1)}$$

$$\frac{\vdash p : \tau_1 \Longrightarrow \Gamma_1 \qquad \text{Dom}(\Gamma) \cap \text{Dom}(\Gamma_1) = \emptyset \qquad \Gamma, \Gamma_1 \vdash e : \tau_2 \qquad \Gamma \vdash bs : \tau_1 \Longrightarrow \tau_2}{\Gamma \vdash p\ \Rightarrow\ e\ ||\ bs : \tau_1 \Longrightarrow \tau_2}\ \text{(Tbs2)}$$

The complete set of typing rules for the language may be found in Appendix B.

Theory Questions [10 Points]

Q. 1 [2 Points] Write down a complete typing derivation showing that the following expression is well-typed (in the empty context) when the type τ is $(int * int) + (int * (unit + unit))$:

```
(λx:τ.
   case x of
      in2(x1,in1()) ==> x1 + 10
   || _ => 0)
```

Q. 2 [3 Points] Write down a collection of inference rules that define the "does not match" judgement $v \notin p$.

Q. 3 [3 Points] Given your definition of "does not match" from question Q. 2, prove that if $\vdash v : \tau$ and $\vdash p : \tau \Longrightarrow \Gamma$ then either $v \notin p$ or $v \in p \rightsquigarrow S$. (If this theorem is false because then change your definition in Q. 2 to make it true!!)

Q. 4 [2 Points] Suppose for a second that one of the other students in the class (not you, of course!) gave an improper definition of the judgement $v \notin p$ such that the lemma in question 3 was false. Which of the following other lemmas about the language are guaranteed to be false (more than one may be false – list all of the false ones in your answer): (a) substitution lemma (b) canonical forms lemma (c) exchange lemma (d) progress lemma (e) preservation lemma. Explain in a sentence or two why the lemma(s) in question is(are) false.

Implementation Question [10 Points]

Q. 5 [10 Points] Your job is to implement the operational semantics of the pattern matching language. T To do so, you will extend the DeBruijn-based implementation of assignment #4. In assignment #4, the language of terms was very simple. In particular, there was only one way to introduce a new variable and only one variable was introduced at a time (using the $\lambda x.e$ expression). In the pattern matching language however, *many* variables may be introduced at one time using a pattern. As an example, consider the following code (I'm omitting typing declarations from the code because they do not play a role in the operational semantics):

```
(λx.
   case x of
      ((x1,x2),_) => x1 + x2
   || _ => 0
) ((6,18),(23,26))
```

The equivalent DeBruijn expression will look like this:

```
(λ.
   case [1] of
      ((?,?),__) => [2] + [1]
   || __ => 0
) ((6,18),(23,26))
```

Notice that in the DeBruijn representation, the patterns have "?" in them in place of introducing a variable name. (The question marks differentiate variable introduction from wildcard patterns.)

Here is a slightly more complex example:

```
(λx.
   case x of
      ((x1,x2),(x3,x4)) => x1 + x2 + x3 + x4
   || __ => 0
) ((6,18),(23,26))
```

It will be translated into:

```
(λ.
   case [1] of
      ((?,?),(?,?)) => [4] + [3] + [2] + [1]
   || __ => 0
) ((6,18),(23,26))
```

In the file `lam.sml`, there are 2 structures, `DB` and `Lam`. `DB` contains datatypes that defines the syntax of expressions (`exp`), branches (`bs`) and patterns (`pat`) in the DeBruijn representation. `Lam` contains contains similar datatypes for the variable representation. Your job is to implement substitution and single step functions in `Lam` and to implement conversion functions in `DB`. For the single step functions, you should follow the operational definitions exactly. In particular, you should implement one function for each different judgement form.

Each structure contains printing functions to help you with debugging. In addition, the `test.sml` file implements a random expression generator that you can use to test your implementation. Use comments to explain your implementation and any interesting design decisions.

Appendix A: Operational Semantics

Pattern Matching Judgement Form: $\boxed{v \in p \rightsquigarrow S}$

$$\frac{}{v \in x \rightsquigarrow [v/x]}\ \text{(EPvar)} \qquad \frac{}{v \in \_\_ \rightsquigarrow .}\ \text{(EPwild)}$$

$$\frac{}{() \in () \rightsquigarrow .}\ \text{(EPunit)} \qquad \frac{v_1 \in p_1 \rightsquigarrow S_1 \quad v_2 \in p_2 \rightsquigarrow S_2}{(v_1, v_2) \in (p_1, p_2) \rightsquigarrow S_1 \circ S_2}\ \text{(EPpair)}$$

$$\frac{v \in p \rightsquigarrow S}{\mathtt{in}_1[\tau]\ v \in \mathtt{in}_1\ p \rightsquigarrow S}\ \text{(EPin1)} \qquad \frac{v \in p \rightsquigarrow S}{\mathtt{in}_2[\tau]\ v \in \mathtt{in}_2\ p \rightsquigarrow S}\ \text{(EPin2)}$$

Pattern Non-Matching Judgement Form: $\boxed{v \notin p}$

(Your job)

Single Step Judgement Form: $\boxed{e \longrightarrow e'}$

$$\frac{e_1 \longrightarrow e_1'}{e_1 + e_2 \longrightarrow e_1' + e_2}\ \text{(Eadd1)} \qquad \frac{e_2 \longrightarrow e_2'}{v_1 + e_2 \longrightarrow v_1 + e_2'}\ \text{(Eadd2)} \qquad \frac{(n_1 + n_2 = n_3)}{n_1 + n_2 \longrightarrow n_3}\ \text{(Eadd3)}$$

$$\frac{e_1 \longrightarrow e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2}\ \text{(Eapp1)} \qquad \frac{e_2 \longrightarrow e_2'}{v_1\ e_2 \longrightarrow v_1\ e_2'}\ \text{(Eapp2)} \qquad \frac{}{(\lambda x{:}\tau.e)\ v \longrightarrow e[v/x]}\ \text{(Eapp3)}$$

$$\frac{e_1 \longrightarrow e_1'}{(e_1, e_2) \longrightarrow (e_1', e_2)}\ \text{(Epair1)} \qquad \frac{e_2 \longrightarrow e_2'}{(v_1, e_2) \longrightarrow (v_1, e_2')}\ \text{(Epair2)}$$

$$\frac{e \longrightarrow e'}{\texttt{in}_1[\tau]\ e \longrightarrow \texttt{in}_1[\tau]\ e'}\ \text{(Ein1)} \qquad \frac{e \longrightarrow e'}{\texttt{in}_2[\tau]\ e \longrightarrow \texttt{in}_2[\tau]\ e'}\ \text{(Ein2)}$$

$$\frac{e \longrightarrow e'}{\texttt{case}\ e\ \texttt{of}\ (bs) \longrightarrow \texttt{case}\ e'\ \texttt{of}\ (bs)}\ \text{(Ecase1)}$$

$$\frac{}{\texttt{case}\ v\ \texttt{of}\ (\_ \Rightarrow e) \longrightarrow e}\ \text{(Ecase2)}$$

$$\frac{v \in p \rightsquigarrow S}{\texttt{case}\ v\ \texttt{of}\ (p \Rightarrow e \mid\mid bs) \longrightarrow S(e)}\ \text{(Ecase3)}$$

$$\frac{v \notin p}{\texttt{case}\ v\ \texttt{of}\ (p \Rightarrow e \mid\mid bs) \longrightarrow \texttt{case}\ v\ \texttt{of}\ (bs)}\ \text{(Ecase4)}$$

Appendix B: Typing Rules

Judgement form: $\boxed{\vdash p : \tau \Longrightarrow \Gamma}$

$$\frac{}{\vdash x : \tau \Longrightarrow x{:}\tau}\ \text{(TPvar)} \qquad \frac{}{\vdash \_ : \tau \Longrightarrow \cdot}\ \text{(TPwild)}$$

$$\frac{}{\vdash () : \texttt{unit} \Longrightarrow \cdot}\ \text{(TPunit)} \qquad \frac{\vdash p_1 : \tau_1 \Longrightarrow \Gamma_1 \qquad \vdash p_2 : \tau_2 \Longrightarrow \Gamma_2 \qquad \mathrm{Dom}(\Gamma_1) \cap \mathrm{Dom}(\Gamma_2) = \emptyset}{\vdash (p_1, p_2) : \tau_1 * \tau_2 \Longrightarrow \Gamma_1, \Gamma_2}\ \text{(TPpair)}$$

$$\frac{\vdash p : \tau_1 \Longrightarrow \Gamma_1}{\vdash \texttt{in}_1\ p : \tau_1 + \tau_2 \Longrightarrow \Gamma_1}\ \text{(TPin1)} \qquad \frac{\vdash p : \tau_2 \Longrightarrow \Gamma_2}{\vdash \texttt{in}_2\ p : \tau_1 + \tau_2 \Longrightarrow \Gamma_2}\ \text{(TPin2)}$$

Judgement form: $\boxed{\Gamma \vdash bs : \tau_1 \Longrightarrow \tau_2}$

$$\frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \_ \Rightarrow e : \tau_1 \Longrightarrow \tau_2}\ \text{(Tbs1)}$$

$$\frac{\vdash p : \tau_1 \Longrightarrow \Gamma_1 \qquad \mathrm{Dom}(\Gamma) \cap \mathrm{Dom}(\Gamma_1) = \emptyset \qquad \Gamma, \Gamma_1 \vdash e : \tau_2 \qquad \Gamma \vdash bs : \tau_1 \Longrightarrow \tau_2}{\Gamma \vdash p \;\Rightarrow\; e \;||\; bs : \tau_1 \Longrightarrow \tau_2} \text{ (Tbs2)}$$

Judgement form: $\boxed{\Gamma \vdash e : \tau}$

$$\frac{x{:}\tau \in \Gamma}{\Gamma \vdash x : \tau} \text{ (Tvar)} \qquad \frac{}{\Gamma \vdash n : \texttt{int}} \text{ (Tint)}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \tau_2} \text{ (Tadd)}$$

$$\frac{\Gamma, x{:}\tau_1 \vdash e : \tau_2 \qquad x \notin \mathrm{Dom}(\Gamma)}{\Gamma \vdash \lambda x{:}\tau_1.e : \tau_1 \to \tau_2} \text{ (Tfun)} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2} \text{ (Tapp)}$$

$$\frac{}{\Gamma \vdash () : \texttt{unit}} \text{ (Tunit)} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 * \tau_2} \text{ (Tpair)}$$

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \texttt{in}_1[\tau_1 + \tau_2]\ e : \tau_1 + \tau_2} \text{ (Tin1)} \qquad \frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \texttt{in}_2[\tau_1 + \tau_2]\ e : \tau_1 + \tau_2} \text{ (Tin2)}$$

$$\frac{\Gamma \vdash e : \tau_1 \qquad \Gamma \vdash bs : \tau_1 \Longrightarrow \tau_2}{\Gamma \vdash \texttt{case}\ e\ \texttt{of}\ (bs) : \tau_2} \text{ (Tcase)}$$