Assignment 3 – COMP 523
Language-based security

Brigitte Pientka

Winter 2008
Due Feb 21st 2008

In many functional programming languages, we find support for defining functions via pattern matching. In this exercise, we explore the use of pattern matching. For example, we can write a simple function which accepts as input a tuple and adds up both parts as follows in SML: `fn (x,y) => x + y`.

Unfortunately, we cannot write functions via pattern matching in the language we have defined so far. To allow for pattern matching we change the rule for λ-abstraction and application.

$$\begin{array}{ll} \text{Terms} & t ::= \ldots \mid t_1\ t_2 \mid\ (\mathsf{fn}\ p_1 \Rightarrow t_1 \mid \ldots \mid p_n \Rightarrow t_n) \\ \text{Patterns} & p ::= x \mid \mathtt{true} \mid \mathtt{false} \mid \mathsf{z} \mid \mathsf{succ}\ (p) \mid (p_1\ ,\ p_2) \end{array}$$

The goal is to replace the case-expression for natural numbers and the first and second projections for tuples by this general pattern matching facility.

To illustrate, we define a function and and a function pred as follows:

$$\begin{array}{ll} \mathsf{and} & = \mathsf{fn}\ (\mathtt{true}, x) \Rightarrow x \mid (\mathtt{false}, x) \Rightarrow \mathtt{false} \\ \mathsf{pred} & = \mathsf{fn}\ \mathsf{z} \Rightarrow \mathsf{z} \mid \mathsf{succ}\ (x) \Rightarrow x \end{array}$$

When we apply for example $(\mathtt{true}, \mathtt{false})$ to the function and we will match $(\mathtt{true}, \mathtt{false})$ against the pattern $(\mathtt{true}, x)$. This will yield an instantiation for x s.t. these two terms are equal. In this case x will be instantiated with `false`.

15 pts First we consider valid patterns. A valid pattern must be linear, i.e. every variable occurring in a pattern must occur uniquely and only once. For example, z is clearly linear. So is succ (x), or $(x, \mathsf{succ}\ (y))$. However, $(x, \mathsf{succ}\ (x))$ is not linear, since x occurs twice.

Using the judgement $\Gamma \vdash p$ linear, define via axioms and inference rules whether a given pattern is linear. You can think of Γ as a list of variables occurring in p, i.e. $\Gamma = x_1, \ldots, x_n$.

Hint: Note that you can simply split a context by writing Γ_1, Γ_2.

15 pts Define the typing rule for $(\mathsf{fn}\, p_1 \Rightarrow t_1 \mid \ldots \mid p_n \Rightarrow t_n)$. Since we are defining a function, the type of this term should be $T_1 \to T_2$. In fact for every branch $p_i \Rightarrow t_i$, we must have that p_i has type T_1 and the body of the function t_i has type T_2. But wait, patterns may contain variables and simply checking that p_i has type T_1 is not sufficient – while verifying that p_i has type T_1 we must also extract the type of the variables occurring in p_i. This is achieved by the judgement $p_i : T_1 / \Gamma'$.

So the typing rule for our new functions which support pattern matching looks as follows:

$$\frac{\text{for every } i \quad p_i : T_1 \;/\; \Gamma_i \qquad ?}{\Gamma \vdash (\mathsf{fn}\, p_1 \Rightarrow t_1 \mid \ldots \mid p_n \Rightarrow t_n) : T_1 \to T_2}$$

Your task is to first define $p : T \;/\; \Gamma$, s.t. indeed the pattern p is linear and has type T in the typing context Γ. We write Γ to the right, to emphasize this Γ must be constructed and is not given as an input. This will follow closely what you have done already in the first question.

Second, complete the rule above by filling in the correct second premise.

30 pts Matching is obviously an important operation which we will use in the operational semantics. We say a term t matches a pattern p if

- t is ground, i.e. t does not contain any free variables, and
- there exists an instantiation θ for the free variables in the pattern p s.t. $[\theta]p = t$.

Some examples:

$$\begin{array}{lcl}
\mathsf{match}(\mathsf{z}, x) & = & [\mathsf{z}/x] \\
\mathsf{match}((\mathsf{succ}\ (\mathsf{z}), (\mathtt{true}, \mathsf{z})), (\mathsf{succ}\ (x), y)) & = & [\mathsf{z}/x, (\mathtt{true}, \mathsf{z})/y] \\
\mathsf{match}((\mathsf{succ}\ (\mathsf{z}), \mathtt{true}), (\mathsf{succ}\ (x), \mathtt{false})) & = & \text{no match}
\end{array}$$

We only allow matching between a value and a pattern if both have the same type. Moreover, you can assume that the pattern p is linear, i.e. every variable in p occurs only once.

15pts Define $\mathsf{match}(t, p) = \theta$ using axioms and inference rules. Remember that p is linear.

15pts Prove the soundness of your algorithm by showing the following:

If $\cdot \vdash t : T$ and $p : T \,/\, x_1{:}T_1 \ldots, x_n{:}T_n$ and
$\mathsf{match}(t, p) = \theta$ s.t. $\theta = [v_1/x_1, \ldots, v_n/x_n]$
then for every v_i we have $\cdot \vdash v_i : T_i$ and $t = [\theta]p$.

20 pts Define small-step evaluation rules for evaluating a function application. – Do we need any special assumptions about the different patterns to ensure our language is deterministic? What about if we want to ensure progress? State clearly the assumptions you are making, and devise a check which will ensure that these assumptions are satisfied.

20 pts Prove type preservation for this extension.