Programming Languages — Homework 7
Types

Due: Wednesday, 8 May 2013, 23:55

Recall the syntax and semantics of the call-by-value simply typed lambda calculus $\lambda^{\to}$:

Contexts		
	$\Gamma ::= \emptyset$	empty context
	$\mid \Gamma, x : \tau$	context extension
Expressions		
	$e ::= x$	variable
	$\mid e_1\ e_2$	function application
	$\mid v$	values
Values		
	$v ::= \lambda x : \tau.\ e$	function abstraction
Types		
	$\tau ::= \tau_1 \to \tau_2$	function type

Operational semantics:

$$\frac{}{(\lambda x : \tau.\ e)\ v \longrightarrow e[x \mapsto v]}\ \text{(E-Beta)}$$

$$\frac{e_1 \longrightarrow e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2}\ \text{(EC-Left)} \qquad \frac{e_2 \longrightarrow e_2'}{v_1\ e_2 \longrightarrow v_1\ e_2'}\ \text{(EC-Right)}$$

Static semantics:

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ \text{(T-Var)}$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1.\ e : \tau_1 \to \tau_2}\ \text{(T-Abs)} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}\ \text{(T-App)}$$

1. [1 pt] Extend the operational semantics of $\lambda^{\to}$ with `int` and `real` literals, and the binary operations + and & (bitwise and).

	$e ::= \ldots$	
	$\mid e_1 + e_2$	addition
	$\mid e_1 \,\&\, e_2$	bitwise and
	$v ::= \ldots$	
	$\mid n$	integer literals
	$\mid r$	real literals
	$\tau ::= \ldots$	
	$\mid$ `int`	integer type
	$\mid$ `real`	real type

   The + operator is *overloaded*: it can be used with either `int` or `real` operands. If either operand is `real`, the result of the operation should be `real`. If both operands are `int`, the result should be `int`. The & operator is not overloaded: both of its operands must be `int`s. We'll call the resulting

language $\lambda^{\rightarrow,\mathsf{int},\mathsf{real}}$. You can assume there are "mathematical" operations $+$ and $\&$ to implement the semantics.

2. [1 pt] Write the static semantics of $\lambda^{\rightarrow,\mathsf{int},\mathsf{real}}$, extending the semantics of $\lambda^{\rightarrow}$.

3. [1 pt] Show by drawing the derivation trees that the following $\lambda^{\rightarrow,\mathsf{int},\mathsf{real}}$ terms have the given types.

   (a) $f : \mathtt{int} \rightarrow \mathtt{int} \vdash f\ (3\ \&\ 4) : \mathtt{int}$

   (b) $f : \mathtt{int} \rightarrow \mathtt{int} \vdash \lambda x : \mathtt{int}.\ f\ (3 + x) : \mathtt{int} \rightarrow \mathtt{int}$

4. [0.5 pts] Find all contexts Γ (containing only the three variables `f`, `x`, and `y`) under which the term `f x` $(5 + \mathtt{y})$ has type `real`.

5. [0.5 pts] Is there a context Γ and type τ such that $\Gamma \vdash z\ z : \tau$? If so, give a typing derivation. If not, prove that no such derivation exists.

6. [2 pts] Show that $\lambda^{\rightarrow,\mathsf{int},\mathsf{real}}$ satisfies the *progress* property: if e is a well-typed, *closed* term, i.e., $\vdash e : \tau$, then either e is a value v or there is an e' such that and $e \longrightarrow e'$.

7. [1 pt] Suppose we add a "cast" operation with the following typing rules:

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash (\mathtt{real})\ e : \mathtt{real}} \qquad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash (\mathtt{int})\ e : \mathtt{int}}$$

   and the following evaluation rules:

$$(\mathtt{real})\ r \longrightarrow r \qquad (\mathtt{int})\ n \longrightarrow n$$

   Show that progress *does not hold* in this extended language.

8. [1 pt] A *record* is a a value that maps names to values. They are similar to tuples except that the fields are named. Let's extend the above language with records, producing the language $\lambda^{\rightarrow,\mathsf{int},\mathsf{real},\mathsf{record}}$.

$e ::= \ldots$		
	$\mid\ \{x_1 = e_1, \ldots, x_n = e_n\}$	record creation
	$\mid\ e.x$	field selection
	$\mid\ e_1[x := e_2]$	record update
	$\mid\ \mathtt{delete}\ e.x$	field deletion
$v ::= \ldots$		
	$\mid\ \{x_1 = v_1, \ldots, x_n = v_n\}$	record value
$\tau ::= \ldots$		
	$\mid\ \{x_1 : \tau_1, \ldots, x_n : \tau_n\}$	record type

   A record creation expression evaluates the fields left to right until the result is a record value. A field selection expression $e.x$ selects the value associated with the field x from a record value e. The record update expression $e_1[x := e_2]$ evaluates its subexpressions left to right and then returns a copy of record e_1 where the x field is changed to the value of e_2. The record delete operation $\mathtt{delete}\ e.x$ returns a copy of record e where field x has been removed.

   Write the operational semantics for $\lambda^{\rightarrow,\mathsf{int},\mathsf{real},\mathsf{record}}$.

9. [1 pt] Write the static semantics for $\lambda^{\rightarrow,\mathsf{int},\mathsf{real},\mathsf{record}}$. There are (at least) two possible typing rules for record updates: one where the type of the record cannot change and one where it can. Record updates should *not* change the type of the record.

10. [1 pt] Extend the proof of progress from part 6 to prove progress for $\lambda^{\rightarrow,\mathsf{int},\mathsf{real},\mathsf{record}}$.

Submission

1. Complete the survey linked from the course web page after completing this assignment.

2. Submit your code and solutions on Moodle by 23:55 on Wednesday, 8 May 2013. Include your name in each file you submit.