CMPT 384: Assignment #5

Anders Miltner
miltner@cs.sfu.ca

Due Mar 25

Introduction

In this assignment, we will be coding up a realistic programming language interpreter. This language will support a variety of features like higher-order functions, sum types, and subtyping.

This language has types of the form:

$$
\begin{array}{rcll}
t & ::= & top & \text{Top} \\
& | & t_1 * t_2 & \text{Pair} \\
& | & t_1 \to t_2 & \text{Arrow} \\
& | & \text{`}C_1\ t_1 | \ldots | \text{`}C_n\ t_n & \text{Sum}
\end{array}
$$

The Sum with 0 elements (represented as Sum []) is also referred to as "bot" and can be written as such.

There is subtyping in these types. Namely, subtyping is described as follows:

- t is a subtype of top, for all t
- bot is a subtype of t, for all t
- if t'_1 is a subtype of t_1 and t_2 is a subtype of t'_2, then $t_1 \to t_2$ is a subtype of $t'_1 \to t'_2$ (careful about the primes, this one is a little tricky)
- $\text{`}C_1\ t_1 | \ldots | \text{`}C_n\ t_n$ is a subtype of $\text{`}C'_1\ t'_1 | \ldots | \text{`}C'_n\ t'_m$ if, for all $\text{`}C_i$, there exists some $\text{`}C'_j$ such that $\text{`}C_i = \text{`}C'_j$ and t_i is a subtype of t'_j

This language has expressions of the form:

$$
\begin{array}{rcl}
e & ::= & () \\
& | & (e_1, e_2) \\
& | & fst\ e \\
& | & snd\ e \\
& | & \lambda(x : t).e \\
& | & e_1\ e_2 \\
& | & x \\
& | & \text{`}Ce \\
& | & match\ e\ with\ | \text{`}C_1\ (x_1 : t_1) \to e_1\ |\ \ldots\ | \text{`}C_n\ (x_n : t_n) \to e_n
\end{array}
$$

Programming Your Interpreter

The semantics of code in this language is `semantics : Expression.t -> semantics_response`, where `semantics_response` can either be a stuck expression or a value expression.

A program is a value if:

- The expression is $()$
- The expression is (e_1, e_2) and both e_1 and e_2 are values
- The expression is $\lambda(x : t).e$

- The expression is $`C\ e$ and e is a value

A program is stuck if there is nothing it can small step to (defined in a second), and it is not a value.

The semantics of an expression e is an expression e' where small steps to e' in some number of steps, and such e' is either a value or stuck.

The small-step semantics ($\rightarrow$) of our language is provided below:

- if $e_1 \rightarrow e_1'$ then $(e_1, e_2) \rightarrow (e_1', e_2)$
- if e_1 is a value and $e_2 \rightarrow e_2'$ then $(e_1, e_2) \rightarrow (e_1, e_2')$
- if $e \rightarrow e'$ then $\mathit{fst}\ e \rightarrow \mathit{fst}\ e'$
- if $e \rightarrow e'$ then $\mathit{snd}\ e \rightarrow \mathit{snd}\ e'$
- if (e_1, e_2) is a value then $\mathit{fst}\ (e1, e2) \rightarrow e_1$
- if (e_1, e_2) is a value then $\mathit{snd}\ (e1, e2) \rightarrow e_2$
- if $e_1 \rightarrow e_1'$ then $e_1 e_2 \rightarrow e_1' e_2$
- if e_1 is a value and $e_2 \rightarrow e_2'$ then $e_1 e_2 \rightarrow e_1 e_2'$
- if e_2 is a value then $(\lambda(x : t).e)e_2 \rightarrow e[e_2/x]$ where $e[e_2/x]$ is e with every instance of x replaced by e_2
- if $e \rightarrow e'$ then $`C\ e \rightarrow `C\ e'$
- if e is a value, then $(\mathit{match}\ `C\ e\ \mathit{with}\ \mid `C_1\ (x_1 : t_1) \rightarrow e_1 \mid \ \ldots\ \mid `C_n\ (x_n : t_n) \rightarrow e_n) \rightarrow e_i[e/x_i]$, where $`C = `C_i$ and $e_i[e/x_i]$ is e_i with every instance of x_i replaced by e

Type-Checking Your Interpreter

Now you will type check your code. The type of type checking is `typecheck : Expression.t -> Type.t option`.

I will use Γ to denote a mapping from variables to types of variables ($\Gamma : \mathit{string} \rightarrow \mathit{type\ option}$). I will use $\Gamma \vdash e : t$ to denote that typechecking e under context Γ results in some output type, t. (In other words, say you have a helper function,
`typecheck_helper : (string -> Type.t option) -> Expression.t -> Type.t option`
$\Gamma \vdash e : t$ would mean that `typecheck_helper gamma e` returns `Some t`).

- $\Gamma \vdash x : \Gamma(x)$
- $\Gamma \vdash () : \mathit{top}$
- If $\Gamma \vdash e : t_1 * t_2$ then $\Gamma \vdash \mathit{fst}\ e : t_1$
- If $\Gamma \vdash e : t_1 * t_2$ then $\Gamma \vdash \mathit{snd}\ e : t_2$
- If $(\Gamma \cup (x \mapsto t_1)) \vdash e : t_2$ then $\Gamma \vdash \lambda(x : t_1).e : t_1 \rightarrow t_2$
- $\Gamma \vdash e_2 : t_1 \rightarrow t_2$ and $\Gamma \vdash e_1 : t_1'$ and t_1' is a subtype of t_1 then $\Gamma \vdash e_2\ e_1 : t_2$
- If $\Gamma \vdash e : t$ then $\Gamma \vdash `C\ e : `C\ t$
- If $\Gamma \vdash e : \mathit{bot}$ then $\Gamma \vdash \mathit{match}\ e\ \mathit{with}\ : \mathit{bot}$
- If $\Gamma \vdash e : t$ and t is a subtype of $`C_1\ t1| \ldots |`C_n\ t_n$ and $\forall i$ between 1 and n, $(\Gamma \cup (x_i \mapsto t_i)) \vdash e_i : t_i'$ then $\Gamma \vdash \mathit{match}\ e\ \mathit{with} \mid `C_1\ (x_1 : t_1) \rightarrow e_1 \mid \ \ldots\ \mid `C_n\ (x_n : t_n) \rightarrow e_n : t'$ where t' is the unique type such that t_i' is a subtype of t' for all i, and for any other type t'' such that t_i' is a subtype of t'', then t' is a subtype of t''. (In other words, t' the smallest supertype of all t_i')