Assignment 6 Extra Credit: Subtyping Intersection Types

15-312: Foundations of Programming Languages
Matthew Moore (mlmlm@cmu.edu)

Due: Thursday, November 4, 2004 (11:59 pm)

30 points total

1 Introduction

In class, you were briefly introduced to a new concept in type-theory, *intersection types*. One of the obvious benefits of intersection types is the ability to give a formal type to overloaded operators and overloaded user-defined functions. You may recall from class the notation: $e : \tau \wedge \sigma$ meaning that we can give e both type τ and type σ.

2 Typing Rules

Consider the following rules for extending our bi-directional typechecker to include intersection types:

$$\frac{\Gamma \vdash v \downarrow \tau \quad \Gamma \vdash v \downarrow \sigma \quad v \text{ value}}{\Gamma \vdash v \downarrow \tau \wedge \sigma} \wedge I \qquad \frac{\Gamma \vdash e \uparrow \tau \wedge \sigma}{\Gamma \vdash e \uparrow \tau} \wedge E_1 \qquad \frac{\Gamma \vdash e \uparrow \tau \wedge \sigma}{\Gamma \vdash e \uparrow \sigma} \wedge E_2$$

Under the coercion interpretation, an expression e of type $\tau_1 \wedge \tau_2$ would be compiled to a pair of expressions $\langle e_1, e_2 \rangle$, where e_1 has type τ_1 and e_2 has type τ_2. In order to avoid evaluating e twice (once as e_1 and once as e_2) we impose a value restriction on this rule.

If we have an overloaded symbol $+$ it is compiled into a pair of two functions: one operating only on integers and one operating only on floating point numbers, and similarly for $*$ and other arithmetic operators and constants. For example,

$$+ : (\texttt{int} * \texttt{int} \rightarrow \texttt{int}) \wedge (\texttt{float} * \texttt{float} \rightarrow \texttt{float}) \quad \text{elaborates to} \quad \langle +_{\mathsf{int}}, +_{\mathsf{float}} \rangle$$

User-defined functions can be similiary overloaded, for example,

$$\lambda x.\ x + 4 \downarrow (\texttt{int} \rightarrow \texttt{int}) \wedge (\texttt{float} \rightarrow \texttt{float})$$

Task: Subtyping (15 points)
Give subtyping rules (with coercions) for intersection types. This involves: determining where

the subtyping relation is appropriate, annotating the relation with a coercion which witnesses the subtype. You should use the notation given here (which gives the coercion for product types):

$$\frac{f : \tau_1 \sqsubseteq \sigma_1 \quad g : \tau_2 \sqsubseteq \sigma_2}{(\lambda x.\langle f(fst(x)), g(snd(x))\rangle) : \tau_1 * \tau_2 \sqsubseteq \sigma_1 * \sigma_2}$$

Task: Type Checking Algorithm (5 points)

Briefly explain how the new rules force our typechecking algorithm to allow for backtracking now. In other words, previously, when searching for a typing derivation, we always knew which rule to apply next, how do the new rules change this?

Task: Implementation (10 points)

Extend your implementation to allow for intersection types (and thereby overloading). If so, you have implemented a much more complicated type system than any currently existing language on the market, including universal, existential, recursive, and intersection types. Congratulations!

3 Hand-in Instructions

For the implementation portion of the extra credit, turn in **separate** extra credit files, put them in a subfolder named `EC`. You will have to modify other parts of the interpreter than the typechecker for this portion of the assignment, so turn in all of the assignment files. We should be able to simply type `CM.make()` into `sml` and have it load your solution properly.

Turn in written extra credit solutions as text, postscript or pdf files in the handin directory. Or, if you wish, you may turn in answers on paper, due in the instructor's office by 11:59 pm on the due date. **If you are using late days, paper handin is by arrangement only** (send mail and we'll figure something out).