Harvard School of Engineering and Applied Sciences — CS 152: Programming Languages

Sub-structural type systems; Algebraic structures
Section and Practice Problems

Apr 6–10, 2020

---

1 Sub-structural type systems

Recall the linear lambda calculus from Lecture 17.

(a) Suppose we extended the language with a negation operation $\mathsf{not}\ e$. Intuitively, e needs to have boolean type, and the expression evaluates e to a value v and then evaluates to the negation of the boolean represented by v. The result of the negation is unrestricted (i.e., it is not linear). Write a typing rule for the negation operator.

(b) Suppose we extended the language with a conjunction operator, $e_1\ \mathsf{and}\ e_2$. Intuitively, e_1 and e_2 both need to have boolean type, and the expression evaluates to the conjunction of the booleans. The boolean value that is the result of the conjunction is unrestricted (i.e., it is not linear). Write a typing rule for the conjunction operator.

(c) Let context $\Gamma = x : \mathsf{lin}\ (\mathsf{lin}\ \mathbf{bool} \times \mathsf{un}\ \mathbf{bool})$ and expression $e \equiv \mathsf{split}\ x\ \mathsf{as}\ y, z\ \mathsf{in}\ \mathsf{if}\ y\ \mathsf{then}\ z\ \mathsf{else}\ \mathsf{not}\ z$.

Is e well-typed for context Γ? That is, does $\Gamma \vdash e : \mathsf{un}\ \mathbf{bool}$ hold? If so, give the derivation.

(d) Consider the execution of the following (well-typed) expression, starting from the empty store. What does the final store look like? (In particular, recall that locations containing linear values are removed once the linear value is used.)

$$(\mathsf{lin}\ \lambda x : \mathsf{lin}\ (\mathsf{un}\ \mathbf{bool} \times \mathsf{un}\ \mathbf{bool}).\ \mathsf{split}\ x\ \mathsf{as}\ y, z\ \mathsf{in}\ y)\ (\mathsf{lin}\ (\mathsf{un}\ \mathsf{false}, \mathsf{un}\ \mathsf{true}))$$

2 Algebraic structures

(a) Show that the option type, with *map* defined as in the lecture notes (Lecture 18, Section 2.2) satisfy the functor laws.

(b) Consider the list type, $\tau\ \mathbf{list}$. Define functions *return* and *bind* for the list monad that satisfy the monad laws. Check that they satisfy the monad laws.

3 Haskell

(a) Install the Haskell Platform, via `https://www.haskell.org/platform/`.

(b) Get familiar with Haskell. Take a look at `http://www.seas.harvard.edu/courses/cs152/2020sp/resources.html` for some links to tutorials.

In particular, get comfortable doing functional programming in Haskell. Write the factorial function. Write the append function for lists.

(c) Get comfortable using monads, and the bind syntax. Try doing the exercises at `https://wiki.haskell.org/All_About_Monads#Exercises` (which will require you to read the previous sections to understand `do` notation, and their previous examples).

(d) Also, look at the Haskell code provided on the section page, which includes some example Haskell code (that will likely be covered in Section).