CS4450 Lecture 21 Notes

Monday, October 8, 2018 3:01 PM

First class after the midterm!

We'll be most likely getting our midterms back on Wednesday.

Today, we'll be talking about program variables.

First, some vocabulary:

So far, we've been working with two languages, object language and metalangauge.

The object language is the language being studied (e.g. arithmetic expressions in the Op and Exp examples), and the metalanguage is the language we're using (Haskell)

Binding is the process of associating a variable with a value.

"Mathematical variables" vary by context. E.g., x and x vary because they are in different contexts:

x^2 + 2x + 9 = 0

x^2 - 5x -7 = 0

"Program Variables" mutate - They are really **containers**

For example, x contains a number of values in the same context:

-- During the loop, x holds between 0 and 10, and is a different value depending on where you are in the loop.

x = 0; while (x++ < 10) { … }

In Haskell, variables are more like mathematical variables, because you can't assign anything to them (without **let**, I think). Something like Java allows you to assign variables to values, however.

Substitution:

Substitution **replaces** a variable with a value in an expression.

Substitution: x -> 5
Expression: x + 2
Produces: 5 + 2

If you have two distinct variables but only one has a substitution rule, then you only change the first variable and leave the second.

Substitutions can be represented as pairs, with the first item being the variable and the second item being the associated value. For example, ("x", 5) would be a representation of substitution, and x would be substituted with 5.

Environment: Collection of multiple bindings. We've seen this concept before with propositional calculus.

type Env = [(String,Int)]

-- (String,Int) is a binding. It's saying that String is a variable and Int is a value.

Haskell has **let**, which allows a variable to be defined in a local context.

int x = 3 in C is basically the same thing as **let x = 3** in Haskell.

The scope of a variable declaration is the portion of the code text where that declaration holds. To put another way, it's the section of code where the variable is what it was declared to be.

New kind of Haskell error we may run into: Attempting to evaluate an expression containing a variable that does not have a value.

I think Harrison might ask some scope questions on the final, seems like they could be easily testable.

Statically typed language (static types) means that every expresssion has to have a type that can be returned at compile time. The fact that it can be returned at compile time makes it static.

A static property of a program can be determined by examining the text of the program but without executing or evaluating it.

A dynamic property of a program can only be determined by evaluating the program.

Examples of this include the random() function, or potential division by zero (may not always see it at compile time).

Variables being defined is a static property. Whether it is defined depends only on program text, not upon the particular data that the program is manipulating.