Q&A Session for Programming Languages Lecture 15

Session Number: 1201135161
Date: 2020-10-27
Starting time: 14:25

---

ANON - 14:34
Q: Are they speaking, I can't hear them
Priority: N/A
	Konstantin Kuzmin - 14:36
	A: Quiz 5 is now live on Submitty. Following the quiz, we will have our usual streaming of lecture videos, starting at 3:00 pm EDT.
---

ANON - 14:40
Q: Is there any specific way we should Q3 or will it be manually graded ?
Priority: N/A
	Ana L. Milanova - 14:43
	A: We have an autograder that can do fuzzy string comparison. You can phrase your answer as you would like.
---

ANON - 14:40
Q: *phrase
Priority: N/A
	Konstantin Kuzmin - 14:48
	A: Still, try to stick to regular English grammar and spelling though. ;)
---

ANON - 14:55
Q: What's the max amount of people we can have in a team for the next homework?
Priority: N/A
	Ana L. Milanova - 14:56
	A: min is 1 to max is 3.
---

ANON - 14:59
Q: So for Q3, is it expecting some keyword or we should go for full sentence?
Priority: N/A
	Konstantin Kuzmin - 15:02
	A: Just write your answer as you would for a human graded question.
---

ANON - 15:13
Q: Is a full stop necessary at the end of our answer for question 3? I assume not
Priority: N/A
	Konstantin Kuzmin - 15:15
	A: It's up to you, not really important.
---

ANON - 15:14
Q: how would you express multiple parameters?
Priority: N/A
	Ana L. Milanova - 15:15
	A: In the pure lambda calculus that we will be discussing, you cannot. But there is a way to express an n-ary function in terms of n applications of unary functions.
	Ana L. Milanova - 15:15
	A: We will discussing this in lectures.
---

ANON - 15:19
Q: does f x mean f applied on x?
Priority: N/A
	Ana L. Milanova - 15:19
	A: Correct.
---

ANON - 15:20
Q: would x ( y z ) be the same as y z x ?
Priority: N/A
	Ana L. Milanova - 15:22
	A: No, it wouldn't. x (y z) applies x on argument (y z). y z x is ((y z) x), it applies the result of (y z) on argument x. These are two different expressions.
---

ANON - 15:21
Q: how would we write lambda calculus as a production taking into account left-assocativity and application over abstraction?
Priority: N/A
	Ana L. Milanova - 15:25
	A: What do you mean by "a production"?
	Ana L. Milanova - 15:31
	A: Yes, this is a valid and interesting question. You can write a grammar that avoids redundant parentheses using the conventions we defined. You can do that using the techniques that we described earlier in class.
---

ANON - 15:22
Q: X Applied On Z means that Z Is An Argument To X?

Priority: N/A
Steven Haussmann - 15:22
A: Yes.

---

ANON - 15:25
Q: so Lx. x z means we apply x on z then apply the value to Lx.?
Priority: N/A
Steven Haussmann - 15:26
A: λx.x z is an abstraction that will apply its argument, x, on z. "Lx" is another way to write λx

---

ANON - 15:25
Q: In Lambda x. x z, is z the argument and x the parameter?
Priority: N/A
Steven Haussmann - 15:28
A: Since application has higher precedence, λx.x z is equivalent to λx.(x z). Here, x is the parameter of the abstraction. z will be the argument to whatever x winds up being.
Ana L. Milanova - 15:27
A: With our syntactic convention, no. Lambda x. x z = (Lambda x. (x z)). Function application takes precedence over abstraction. We would write (Lambda x. x) z for the case when z is the argument to the identity function.

---

ANON - 15:26
Q: By prodution, I meant grammar/list of rules
Priority: N/A

---

ANON - 15:32
Q: Can the one argument be a list?
Priority: N/A
Steven Haussmann - 15:33
A: The lambda calculus is defined entirely by variables, abstractions, and applications. The language has no other constructs. Of course, there's nothing stopping you from coming up with a representation of a list!

---

ANON - 15:33
Q: if there's no + or integers, then what would we use in pure lambda calculus?
Priority: N/A
Steven Haussmann - 15:34
A: There are ways to represent integers and arithmetic operations entirely in the λ-calculus. We'll likely be talking about

that in the coming lectures.

---

ANON - 15:35
Q: Is Abstraction also Left-Associative?
Priority: N/A
        Ana L. Milanova - 15:43
        A: We don't have associativity rules for \lambda x.E. The rule states that the scope of the dot extends as far to the right as possible. (I might be misinterpreting the question.)

---

ANON - 15:36
Q: Is slide 10 just showing how any function with multiple parameters can be reexpressed as multiplt funtions with single parameter?
Priority: N/A
        Ana L. Milanova - 15:44
        A: Yes, currying illustrates how we can use n unary functions to represent one n-ary function.

---

ANON - 15:37
Q: how can anything be assigned to TRUE if lambda calculus doesn't have assignment?
Priority: N/A
        Steven Haussmann - 15:37
        A: Think of it as shorthand. When we write TRUE, we actually mean the lambda calculus expression we said it was equal to.

---

ANON - 15:37
Q: I missed what you said regarding precedence. What is the order of precedence for lambda calculus?
Priority: N/A
        Ana L. Milanova - 15:46
        A: The convention that we defined states (essentially) that 1) application is left-associative and 2) application takes precedence over abstraction. I said that we could define an expression grammar that uses this convention to avoid certain parentheses.

---

ANON - 15:43
Q: what scoping does this example use (slide 13)?
Priority: N/A
        Steven Haussmann - 15:44
        A: It's closest to static scoping, I suppose -- variables are bound by the closest enclosing abstraction.
        Steven Haussmann - 15:45
        A: But we don't have notions of things like "stack frames" here!

---

ANON - 15:46
Q: when is a variable bound versus when is a variable being applied?
Priority: N/A
        Steven Haussmann - 15:46
        A: A variable is bound when it is enclosed in an abstraction with the same name

---

ANON - 15:52
Q: although it is left-associative, should we evaluate from innermost to outermost?
Priority: N/A
        Ana L. Milanova - 15:58
        A: How we choose what to evaluate is determined by what reduction strategy we use (innermost=applicative, outermost=normal). We'll talk about reduction strategies next time. Right now, we are looking at just _one_ step of evaluation.

---

ANON - 15:54
Q: How would one actually call this lambda function?  I'm having a tough time visualizing what the arguments would be.
Priority: N/A
        Ana L. Milanova - 15:55
        A: We'll do a lot of examples with that particular term (or sligthly different ones).
        Ana L. Milanova - 15:56
        A: A useful argument is the identity function: \lambda x. x. That one comes up a lot.

---

ANON - 15:55
Q: Okay, thank you!
Priority: N/A

---

ANON - 15:56
Q: Why is it the empty set for Abs
Priority: N/A
        Ana L. Milanova - 16:08
        A: I'm assuming this is a question about free variables. It is not always the empty set for Abs. If we have \lambda x. x, then it is the empty set. But if we have \lambda x. x y, then it is {y}.

---

ANON - 15:57
Q: this is not a question, but i'm having a lot of trouble understanding these examples despite understanding part 1 of the

lecture
Priority: N/A

---

ANON - 15:58
Q: so all the x, y, z are the same thing no matter what order they are in? so \lambda x. \lambda y xy and  \lambda x. \lambda y yx are the same?
Priority: N/A
Steven Haussmann - 15:59
A: λx.λy.x y and λx.λy.y x are different. The former applies the first argument to the second argument; the latter applies the second argument to the first argument.

---

ANON - 16:08
Q: what is meant by if x = y? y is bound to x?
Priority: N/A
Ana L. Milanova - 16:10
A: This means y is x. When we have y[M/x] this means, we need to replace x with M in y. If y is x, i.e., we have x[M/x], then the result of the replacement is M.

---

ANON - 16:09
Q: does the substituion algorithm only follow one of var, app, abs? or all
Priority: N/A
Ana L. Milanova - 16:12
A: We have to be able to do substitution in Var, App, and Abs expressions. Substitution is recursively defined. Suppose we have to do E[M/x], i.e., substitute (i.e., replace) x with M in E. E can be arbitrarily complex, e.g., an Abs, an App, etc. and each Abs or App can be made
Ana L. Milanova - 16:12
A: up of subexpressions.

---

ANON - 16:11
Q: why was the [ ] notation chosen? I think that's what was confusing for me
Priority: N/A
Ana L. Milanova - 16:14
A: This is again a convention. This is the standard notation in several textbooks. (But I've seen different notation for the same thing in different textbook.) Try to get used to our notation, it will take some getting used to, one of the biggest hurdles is getting through the dense notation.

---

ANON - 16:12
Q: So the reason Abstraction Rule for Subtitution when x = y stays the same is because if x = y then we would be changing both all x in E1 and the formal parameter which is just equivalent to Lambda y . E1 that we startd with?
Priority: N/A

Ana L. Milanova - 16:18

A: In this case, we should not be changing x. We are moving into an inner scope. All references to x in E1 refer to that last binding, not our x that we are replacing. (We'll work through more examples next time to illustrate this better.)

---

ANON - 16:15
Q: With these substitutions, we are essentially illiminating the higher order abstraction?
Priority: N/A

Ana L. Milanova - 16:20

A: We need the substitutions when we reduce expressions: (\lambda x. E) M, this is an application expression.

Ana L. Milanova - 16:24

A: The evaluation of this expression requires that we substitute M into E for every occurrence of parameter x in E. (Just as in a normal function.) The substitution is an essential part of each step of evaluation.

Ana L. Milanova - 16:26

A: And yes, I think you are right, the evaluation (and substitution) does illustrate higher-order functions. It illustrates how we pass function arguments and then apply those functions inside the body of the original function that we are passing them to.

---

ANON - 16:18
Q: i cant really ready the handwriting in the example
Priority: N/A

Ana L. Milanova - 16:27

A: I will type that one then, like the previous slide.

---

ANON - 16:18
Q: oh its a 2 nvm
Priority: N/A

---

ANON - 16:18
Q: how do we know what to write next or evaluate next?
Priority: N/A

Ana L. Milanova - 16:29

A: For the substitution algorithm in Slide 20, the order of evaluation is well-defined by the algorithm.

---

ANON - 16:20
Q: Is it okay if we're still a bit confused with the details of this lecture?  It seems like we'll be covering this for a bit
Priority: N/A
        Ana L. Milanova - 16:30
        A: Yes! This is dense notation and it takes some time to get used to. We'll be covering this for sometime and I will adjust material/lecture/examples based on feedback.

---

ANON - 16:21
Q: Yes, more examples would be helpful. Could you also go through them slightly slower?
Priority: N/A
        Ana L. Milanova - 16:31
        A: Yes, will do and will adjust.

---

ANON - 16:23
Q: "A: Yes,  we should not be changing x. We are moving into an inner scope. All references to x in E1 refer to that last binding, not our x that we are replacing " So since x = y, all x in E1 are bound to the formal parameter y = x so stays the same?
Priority: N/A
        Ana L. Milanova - 16:31
        A: Yes, that is correct.

---

ANON - 16:25
Q: I also don't quite understand how free and bound variables impact lambda calculus
Priority: N/A
        Ana L. Milanova - 16:32
        A: The key issue is that you should never make a free variable bound, because you are changing the meaning of the expression.
        Ana L. Milanova - 16:33
        A: In our example, (\lambda x. \lambda y. x y) (y w), here the second "y" is a free variable, it refers to something that comes from outer scope.
        Ana L. Milanova - 16:34
        A: If we replace the formal parameter with the argument blindly, we will get \lambda y. (y w) y, which now binds the "free" y to the \lambda y. But this is a different expression than the intended one, \lambda z. (y w) z.

---

ANON - 16:26
Q: Additionally, should we start out with the inner-most term for

subsitution and work our way out since that's how application precedence works?
Priority: N/A

Ana L. Milanova - 16:36

A: Try to just work through the algorithm in Slide 20. It recursively applies substitution and you don't have to worry about evaluation order. The substitution is carried out as a result of _one_ step of evaluation. We shouldn't worry (quite yet) about multiple steps of evaluation.

Ana L. Milanova - 16:38

A: E.g., suppose we have (\x. x) ((\x.x) z)[M/z], i.e., we are replacing M for z in (\x. x) ((\x.x) z). Since this is an application expression, we first do (\x. x)[M/z] which yields (\x.x) (essentially, I am skipping the step in the algorithm that always replaces the formal parameter with a fresh variable).

Ana L. Milanova - 16:41

A: Then we do ((\x.x) z)[M/z], which again recurses into application and yields ((\x.x) M). So the final result of the substitution will be (\x.x) ((\x.x) M). We don't have to worry about evaluation order. The substitution step is carried out for just _ONE_ step of evaluation.

---

ANON - 16:29
Q: I noticed that after the substitution, we have one less lambda abstraction. Is this always the case?
Priority: N/A

Ana L. Milanova - 16:44

A: Yes. Substitution is the essential component of _one_ step of evaluation. When we evaluate (\x. E) M we have to substitute M for every occurrence of x in E. (Like with a normal function.) We end up with one less \lambda after that one step of evaluation.

---

Lorson Blair(blairl@rpi.edu) - 16:30
Q: ok thank you. your second response answered my question.
Priority: N/A

Ana L. Milanova - 16:44

A: Thanks!