Q&A Session for Programming Languages Lecture 14

Session Number: 1206874022
Date: 2020-10-23
Starting time: 14:26

ANON - 14:33
Q: why is foldr an essential function if it is not scheme built-in?
Priority: N/A
        Ana L. Milanova - 14:36
        A: fold is essential. And most implementations of fold do indeed do foldl. foldr is not so important, you are correct.

ANON - 14:35
Q: what does the function lambda do in the context of rev? would it be another predicate?
Priority: N/A
        Steven Haussmann - 14:37
        A: foldr accepts a function to apply at each step; we're using lambda to define a function

---

ANON - 14:35
Q: was having internet issues. what's the current "topic" it seems same as last lecture?
Priority: N/A
        Steven Haussmann - 14:37
        A: Yes, we're continuing the discussion of functional programming. The lecture outline is available in the lecture slides
        Ana L. Milanova - 14:38
        A: We'll talk about let bindings and scoping in Scheme.

ANON - 14:37
Q: would rev be shallow recursive or deep recusive?
Priority: N/A
        Steven Haussmann - 14:38
        A: It is shallow, as it doesn't care what the list elements are. It would have to check the type of each element and recurse on them as well if it was deep.

ANON - 14:38
Q: Why "list x" instead of just "x" in the lambda func?
Priority: N/A
        Steven Haussmann - 14:38
        A: Append operates on two lists.

```
----------------------------------------------------------------

ANON - 14:38
Q: Would it not be far more efficeint to use foldl to reverse a list?
Priority: N/A
        Ana L. Milanova - 14:39
        A: Yes, you are correct. It will be.
----------------------------------------------------------------

ANON - 14:40
Q: is "(x)" not a list?
Priority: N/A
        Ana L. Milanova - 14:41
        A: In Scheme (x) is an expression that will be evaluated.
        Steven Haussmann - 14:42
        A: (x) would attempt to evaluate a function named x with zero
arguments. '(x) is a list containing an atom named x.
        Steven Haussmann - 14:43
        A: *containing a symbol named x; that's the appropriate term
----------------------------------------------------------------

ANON - 14:41
Q: Hi, I noticed that the HW 5 is a team assignment? How many people
are allowed in a team?
Priority: N/A
        Ana L. Milanova - 14:43
        A: The maximal size in Submitty is set to 3. The ideal size of
the team will be 2.
----------------------------------------------------------------

ANON - 14:41
Q: why are we adding 1 to x exactly?
Priority: N/A
        Steven Haussmann - 14:43
        A: We were computing the length of a list, so for each element
of the list, we add 1.
----------------------------------------------------------------

ANON - 14:42
Q: What is the team size range for teams on homework 5?
Priority: N/A
        Ana L. Milanova - 14:44
        A: Maximal size is 3. The ideal (recommended) size is 2.
----------------------------------------------------------------

ANON - 14:48
Q: is there any benefit to "(list x)" over " '(x)" ?
Priority: N/A
        Ana L. Milanova - 14:49
        A: Yes, in (list x) x is a variable that is bound to a value
```

at runtime. In '(x) x is just the symbol 'x.
        Steven Haussmann - 14:53
        A: On the other hand, (list 'x) and '(x) should be identical (after evaluation, at least)

ANON - 14:54
Q: is flattening lists a really important concept?
Priority: N/A
        Steven Haussmann - 14:56
        A: It comes up pretty frequently, especially in conjunction with mapping -- for example, you might map user IDs to lists of their friends, then flatten the list to get a list of people.
        Ana L. Milanova - 14:57
        A: Yes, I second Steven's answer. In our class, I use flatten as a useful illustration of concepts --- deep recursion, fold, and map.

ANON - 14:58
Q: how are binding lists and s-expr different?
Priority: N/A
        Steven Haussmann - 14:59
        A: The binding list contains pairs of names and expressions; each expression is bound to its corresponding name. Afterward, the expression in the S-expr1 slot is evaluated.
        Ana L. Milanova - 14:59
        A: Syntactically, a binding list is a list S-expression, yes. It has the special semantics/interpretation that we discuss.

ANON - 14:59
Q: just to make sure i'm understanding, let will bind a variable to s-expr?
Priority: N/A
        Steven Haussmann - 14:59
        A: The let operator will evaluate an expression and bind the result to a variable for each entry in the Binding-list. It will then evaluate S-expr1.
        Steven Haussmann - 15:00
        A: For example, (let ((x 4) (y 5)) y) will return 5.

ANON - 15:03
Q: ohh so let will bind x to 2, and then evaluate it with the s-expr?
Priority: N/A
        Ana L. Milanova - 15:05
        A: Yes, that's correct. Let binds x to 2 and then evaluates the expression (S-expr1 in our notation) with that binding.

ANON - 15:05
Q: how are the third example and the fourth example different?
Priority: N/A
        Ana L. Milanova - 15:06
        A: let vs. let*.

ANON - 15:07
Q: does in-parallel mean the let won't evaluate anything else after?
Priority: N/A

ANON - 15:09
Q: So letrec allow the Variables in the Binding to be Undefined whereas let requires that they are Explicitly Defined, but both are evalulated in Parallel?
Priority: N/A
        Ana L. Milanova - 15:11
        A: The three kinds of let define different regions/scopes. I think this slide will illustrate better.

ANON - 15:10
Q: does mutually recursive mean they run at the same time or is there an order they follow?
Priority: N/A
        Ana L. Milanova - 15:14
        A: This is the standard meaning of mutual recursion: two or more functions that are defined in terms of one another. If we have two mutually recursive functions, f1 and f2, f1 calls f2, and f2 calls f1. The calls follow the order specified in code.

ANON - 15:17
Q: does the way you draw the boxes imply static scoping?
Priority: N/A
        Ana L. Milanova - 15:19
        A: Yes, the boxes are meant to show scopes and illustrate how Scheme uses static scoping.

ANON - 15:17
Q: For the example with the region of lecrec, the bindings of x and z can also be referenced from any of v1, v2, v3, and s-expr?
Priority: N/A
        Ana L. Milanova - 15:21
        A: Yes, that is correct.

```
ANON - 15:18
Q: In this Problem, would the Binding of x to 10 change if we had
let*? Would this allow us to see x = 2?
Priority: N/A
        Ana L. Milanova - 15:20
        A: No, it wouldn't. With static scoping, the x is bound at the
function definition to 10. Let* extends the region where the binding
is active, however, but it won't allow a nested let binding to be
"visible" in the enclosing let. Also the nested let shadows the
enclosing let, just as it did in static scoping we discussed a few
weeks ago.
```

```
ANON - 15:20
Q: is this lambda different than the lambda from the first slide?
Priority: N/A
        Ana L. Milanova - 15:23
        A: Can you specify the slide numbers?
```

```
ANON - 15:21
Q: A: No, it wouldn't. With static scoping, the x is bound at the
function definition to 10.  Would this also remain the same for
letrec? I thought that these expression would allow visibility of the
other values?
Priority: N/A
        Ana L. Milanova - 15:27
        A: letrect should not change the result here. The inner let
shadows the definition in the outer let. So a binding of x made in
outer scope is shadowed by the x in the inner scope; the inner x is
active in the scope of the inner let. Letter extends the region a
binding is active but does not change static  scoping rules.
```

```
ANON - 15:23
Q: why do most modern languages use static scoping again?
Priority: N/A
        Steven Haussmann - 15:28
        A: Dynamic scoping means that all of the locals in your
function could be changed by whatever you call in all situations, and
that the variables you can access depends entirely on who called you.
It's a lot harder to reason about.
```

```
ANON - 15:24
Q: this is a side question: but is the lecture 14 enough to start HW5
or should we wait?
Priority: N/A
        Ana L. Milanova - 15:29
        A: let and let bindings will come useful with HW5. And yes,
```

this lecture should be enough to complete the homework.

ANON - 15:32
Q: are all functions in scheme closers?
Priority: N/A
        Ana L. Milanova - 15:33
        A: Yes, you can think of it this way. That all functions are in fact closures. When the function has no free variables, then the ref. environment is just empty.

ANON - 15:34
Q: So closures are subjected to the same scoping rules?
Priority: N/A
        Ana L. Milanova - 15:36
        A: I would say yes. (If I am interpreting the question correctly.) You may have variables that are bound to closures.

ANON - 15:37
Q: when do we know when there's a closure?
Priority: N/A

ANON - 15:37
Q: In the Last Example, it does not seem that the Let ever exited and the Bindings become Inactive, or I am confused on what the Let Expression Exits means? I was wondering what an Exit of this would be.
Priority: N/A
        Ana L. Milanova - 15:45
        A: In that example, slide 20, the blue let block exits and yields the closure: lambda () x, x->10. When a let block finishes evaluation and yields a result, then it exits. (Think of popping a frame.)

ANON - 15:38
Q: what are the team sizes for hw5, how should we indicate our team and should every team member submit the code?
Priority: N/A
        Ana L. Milanova - 15:47
        A: Size: max is 3, ideal is 2. You create the teams on Submitty. Submitty manages teams and only one member shall submit.

ANON - 15:39
Q: The reason I asked the above question is because I thought the Last Example was to illustrate the Immortality of the Closures but it never seemed like we left the Scope of the Let Expression to show this

Priority: N/A
        Ana L. Milanova - 15:48
        A: Slide 20. We left the scope of the blue let expression. But the x->10 binding stayed "immortal" as part of the closure.

ANON - 15:45
Q: So since we can work in groups for HW5, does one person from a team need to submit?
Priority: N/A
        Ana L. Milanova - 15:48
        A: Yes, that is correct.

ANON - 15:46
Q: The Last Example, f was defined in the Inner Let Expression and then used in the Outer S-expr1 = (lis x (f) x (f)), but this seems like it would be the same if we had (x 2) in the Inner Let and then that x Value could be used. So it does not show Immortali
Priority: N/A
        Ana L. Milanova - 15:50
        A: Slide 20. f was defined in the inner black let block. It was used in the S-expr1 of that inner black let. It was showing the immortality of x->10, as the blue let has exited.

ANON - 15:49
Q: Alright I understand now about the Last Example. Thank you Professor. X = 10 Lived On despite exiting the Inner Let Expression so that further calls to f would automatically know that binding.
Priority: N/A
        Ana L. Milanova - 15:51
        A: Yes, correct.

ANON - 15:50
Q: why does having a function be third-class mean it's available everywhere?
Priority: N/A
        Ana L. Milanova - 15:54
        A: With functions as third-class values, we are more limited in the way are visible and callable throughout the program. So this guarantees that when a function its called, its static reference environment is available on the stack.

ANON - 15:53
Q: Why do First-Class Values demand Immortality of Local Variables? I'm slightly confused. Is it just because those Local Variables may be needed for some reason?

```
Priority: N/A
        Ana L. Milanova - 15:55
        A: Because with static scoping, we bind the non-local
variables in the current static reference environment. But the
function value may outlive its static ref. environment (like with the
blue let). The binding will be needed when the function is called.
        Ana L. Milanova - 15:55
        A: And it can be called from practically anywhere.
```

```
ANON - 16:03
Q: So with Deep Binding, the V in Print Routine is still 10 even
though Other Routine has the Local Variable = 5 becuase of the
Reference Environment from Main?
Priority: N/A
        Ana L. Milanova - 16:03
        A: Yes, correct.
```

```
ANON - 16:03
Q: And even though this is Dynamic Scoping? ^
Priority: N/A
        Ana L. Milanova - 16:04
        A: Yes. Because we can pass the print routine as an argument
from another environment. Then Print Routine's closure will carry that
environment's value.
        Ana L. Milanova - 16:09
        A: Slide 30 is a good example, we may have yet another binding
in the let*, (E (lambda () (let (x 10) (C D)))), then D's closure
carries x->10.
```

```
ANON - 16:08
Q: Does C not pass D as a parameter as well? So why does it not print
4?
Priority: N/A
        Ana L. Milanova - 16:10
        A: C gets D as an argument. Dynamic scoping with shallow
binding should yield 4.
```

```
ANON - 16:08
Q: Are one person teams allowed for hw5?
Priority: N/A
        Ana L. Milanova - 16:09
        A: Yes.
```

```
ANON - 16:09
Q: in slide 30, when does the function print?
```

Priority: N/A
        Ana L. Milanova - 16:12
        A: This is in the sense of the REPL interpreter, when the function finishes evaluation it prints the result.
        Ana L. Milanova - 16:13
        A: Sorry, meant when *the interpreter* finishes evaluation, it prints the result.

ANON - 16:10
Q: Is there a scenario where applicative-order and normal-order produces different results?
Priority: N/A
        Steven Haussmann - 16:11
        A: It's possible for applicative order to fail to terminate. However, I believe that they produce identical results if both ways terminate.
        Steven Haussmann - 16:11
        A: Consider a function that takes an argument and ignores it -- if that argument recurses infinitely, applicative order would cause it to get stuck, whilst normal order would never had to evaluate the argument at all.

ANON - 16:11
Q: do normal and applicative order always result in the same answer?
Priority: N/A
        Ana L. Milanova - 16:13
        A: We'll discuss this next week!

ANON - 16:12
Q: what does reduction semantics mean?
Priority: N/A
        Steven Haussmann - 16:13
        A: The reduction semantics are the rules used when performing reductions of an expression.
        Steven Haussmann - 16:13
        A: In general, "semantics" are properties or attributes of something that give it meaning.

ANON - 16:13
Q: Slide 30: Since B has C D as arguments, the Reference Environments for them is fixed as {x -> 2} and so when C is called, despite D being an argument and Local x=4, it ignores this and passes its {x -> 10} obtained from B to D? Right?
Priority: N/A
        Ana L. Milanova - 16:32
        A: I am not sure what you are referring to. There is no x->10

on the slide, but I used this as an example in an earlier answer.

ANON - 16:14
Q: what exacrly does higher order functions like map and reduce mean? they just seem like normal functions
Priority: N/A
        Steven Haussmann - 16:14
        A: A higher-order function accepts one or more functions as part of its arguments.
        Steven Haussmann - 16:15
        A: + is not higher-order because it only accepts numbers. map is higher-order because it takes a function that it applies to each element of a list.

ANON - 16:18
Q: Does the recommended team of two imply double the typical workload expected for this homework?
Priority: N/A
        Ana L. Milanova - 16:33
        A: No it shouldn't be double. I expect it will mean 1/2 of the typical workload assuming you distribute the tasks equally.

ANON - 16:19
Q: I was told a question on the test was being regraded because of a key issue is this true and when should we expect this to be done?
Priority: N/A
        Ana L. Milanova - 16:38
        A: Yes, my mistake for giving an incorrect key. (Mixed V1 and V2.) I apologize. Correct answers to that question that were marked down will go up, typically by 2 points. No grade will go down. (Key was so off, that no one got full points.)

ANON - 16:19
Q: I assume that the 2 person is just cuz there are two functions to do so it is easier to split work?
Priority: N/A
        Ana L. Milanova - 16:39
        A: Yes, you can do that. But we recommend to do "pair programming" as much as possible under the circumstances.

ANON - 16:19
Q: do closures just happen in the background?
Priority: N/A
        Ana L. Milanova - 16:39
        A: Yes, essentially.

ANON - 16:21
Q: Can we get a hint how many parse trees are there for question 1 in the exam?
Priority: N/A

ANON - 16:21
Q: Where can I see the rubric for the exam?
Priority: N/A

ANON - 16:23
Q: Nevermind. found it
Priority: N/A

ANON - 16:29
Q: For Immortiality, I understand that we need th Bindings of the Non-Local Static Reference Variables, but why do we need the Local Variables to be unlimited? Am I misunderstanding what is meant by Local?
Priority: N/A

Ana L. Milanova - 16:41

A: Local variables of the reference environment, i.e., enclosing function. They may need to become "immortal" if referenced from a nested function value that we return. Non-local variables are in the function value that we return.

ANON - 16:34
Q: I am not sure what you are referring to. There is no x->10 on the slide, but I used this as an example in an earlier answer. I meant {x -> 2}. Sorry for the confusion Professor
Priority: N/A

Ana L. Milanova - 16:44

A: Ok! With static scoping, yes, D's x gets bound to x->2. So when D gets called eventually from C, D carries the closure binding x->2. (I hope I interpreted the question right. I have to go to Office hours now. But will "see" you next week!)