CPSC-354 Report

Your Name
Chapman University

November 3, 2024

Abstract

...

Contents

1 Introduction

...

2 Week by Week

2.1 Week 1

Notes

In the reading about the mu puzzle, I learned about formal systems. The idea is that within the rules, you can reach the same desired outcome through different paths (at a given point you may have the option to apply several rules but can only chose one at any given step). In this context, a string which is created by applying one of the rules to the previous string is reffered to as a theorem.

I also learned about discrete mathematics proofs and look forward to learning about how they will be applied to the developement of a programing language.

Homework

NNG Tutorial World Level 5:

a+(b+0)+(c+0)=a+b+c.

Solution:

First we use the Lean add zero proof to remove the 0 in b+0.

Then we use the Lean add zero proof to remove the 0 in c+0.

Finally we are left with a+b+c = a+b+c and we can use rfl to confirm our proof with reflexive property.

...

NNG Tutorial World Level 6:

a+(b+0)+(c+0)=a+b+c.

Solution:

First we use the Lean precision add zero proof (tarteting c) to remove the 0 in c+0.

Then we use the Lean add zero proof to remove the remaining 0 in b+0.

Finally we are left with a+b+c = a+b+c and we can use rfl to confirm our proof with reflexive property.

...

NNG Tutorial World Level 7:

For all natural numbers a, we have succ(a)=a+1.

Solution:

First we unwravel the one with the Lean rw proof to eliminate the one with a succ0.

Then we use the Lean rw proof with add succ to change n + succ0 into succ(n+0).

Then we use the rw proof to rewrite succ(n+0) into succ(n).

Finally we are left with succ(n) = succ(n) and we can use rfl to confirm our proof with reflexive property.

...

NNG Tutorial World Level 8:

For all natural numbers a, we have succ(a)=a+1.

Solution:

For this problem I simplified both sides of the equation using the rewrite proof with succesors such as 3 = succ 2 and more.

Eventually I got to this point: succ (succ 0) + succ (succ 0) = succ (succ (succ (succ 0)))

At this point I began using rw add succ to simplify the left side of the equation

Then I used rw add zero to remove a remaining zero and I was left with this: succ (succ (succ (succ 0))) = succ (succ (succ (succ 0)))

At this point I used the rfl to confirm my proof reflexively.

...

In all of the above examples I used the Lean rfl proof which directly corresponds to the mathematical reflexive property which states that: any number a is equal to itself. In other words a = a or b+c = b+c, etc.

Comments and Questions

This section was a good refresh on some of the discrete mathematics concepts that I had forgotten over break. My question for this week is the following:

How are these mathematical concepts applied to the developement of programming languages?

2.2 Week 2

Notes

In the readings and class lectures about recursion and the towers of hanoi, I learned about what recursion is and how it can be used to optimize certain sysetems. This is evident in the towers of hanoi where we use a recursive formula or pattern to complete the challenge in the minimum amount of steps.

This was a good refresh on discrete mathematics and I look forward to learning how to apply the use of recursion in this course and in other challenges that can be solved mathematically.

Homework

NNG Addition World Level 1:

For all natural numbers n, we have 0 + n = n.

Solution:

1. induction n with d hd

2. rw add zero

3. rfl

4. rw add succ

5. rw hd

6. rfl

...

NNG Addition World Level 2:

For all natural numbers a,b, we have succ(a)+b=succ(a+b)

Solution:

1. induction b

2. rw add zero

3. rfl

4. rw add succ

5. rw add succ

6. rw n ih

7. rfl

...

NNG Addition World Level 3:

On the set of natural numbers, addition is commutative. In other words, if a and b are arbitrary natural numbers, then a+b=b+a

Solution:

1. induction b
2. rw add zero
3. rw zero add
4. rfl
5. rw add succ
6. rw succ add
7. rw n ih
8. rfl

...

NNG Addition World Level 4:

On the set of natural numbers, addition is associative. In other words, if a,b and c are arbitrary natural numbers, we have (a+b)+c=a+(b+c).

Solution:

1. induction a
2. rw zero add
3. rw zero add
4. rfl
5. rw succ add
6. rw succ add
7. rw succ add
8. rw n ih
9. rfl

...

NNG Addition World Level 5:

If a,b and c are arbitrary natural numbers, we have (a+b)+c=(a+c)+b.

Solution:

1. induction a
2. rw zero add
3. rw zero add
4. rw add comm
5. rfl

6. rw succ add
7. rw succ add
8. rw succ add
9. rw succ add
10. rw n ih
11. rfl

This lean proof is similar to the corresponding mathematical proof in the way that it uses induction to prove the theory on a single smaller equation and applies the inductive step and uses said proof to prove a property across all equations!

Comments and Questions

This section was yet another good refresh on discrete and reintroduced me to the concept of induction in proofs. I also loved the towers of hanoi activity and its modeling of recursive processes! My question is how similar puzzles can appear in programming languages and specifically the creation of them?

2.3 Week 3

Notes

This week we began developing our calculator in python which has proven to be more difficult than expected. I believe that the idea is to use a recursive method to scan an input for the subsections of a more complicated long input.

Homework

https://github.com/YoniKazovsky/reportrepo/blob/LLM-Assignment/LLM

2.4 Week 4

Notes

This week we continued to work on the python project which proved to become much more difficult now that I was trying to implement methods that allowed me to breakdown an expression and parse it into multiple subsections with operators and operands. In addition we learned about parsing and parsing trees which proved to be a very useful concept in implementing my python calculator.

Homework

1. 2 + 1

Exp→Exp+Exp1

Exp→Exp1

Exp1→Exp2

Exp2→Integer

Exp1→Exp2

Exp2→Integer

—

—

2. 1 + 2 * 3

Exp→Exp+Exp1

Exp→Exp1

Exp1→Exp2

Exp2→Integer

Exp1→Exp1*Exp2

Exp1→Exp2

Exp2→Integer

Exp2→Integer

—

3. 1 + (2 * 3)

Exp→Exp+Exp1

Exp→Exp1

Exp1→Exp2

Exp2→Integer

Exp1→Exp2

Exp2→(Exp)

Exp→Exp1

Exp1→Exp1*Exp2

Exp1→Exp2

Exp2→Integer

Exp2→Integer

—

4. (1 + 2) * 3

Exp→Exp1

Exp1→Exp1*Exp2

Exp1→Exp2

Exp2→(Exp)

Exp→Exp+Exp1

Exp→Exp1

Exp1→Exp2

Exp2→Integer

Exp1→Exp2

Exp2→Integer

Exp2→Integer

—

5. 1 + 2 * 3 + 4 * 5 + 6

Exp→Exp+Exp1

Exp→Exp+Exp1

Exp→Exp1

Exp1→Exp2

Exp2→Integer

Exp1→Exp1*Exp2

Exp1→Exp2

Exp2→Integer

Exp2→Integer

Exp1→Exp1*Exp2

Exp2→Integer

Exp2→Integer

Exp2→Integer

Comments and Questions

My question for this week is the following: How can this concept of parcing be applied to the developement of a programming language, specifically in the ways in which the backend handles order of operations regarding actual code as opposed to just calculations like we did in our calculators?

2.5 Week 5

Notes

This week we learned about lean logic proofs and how to convert mathematical proofs into lean logic proofs.

Homework

Lean Logic Game Level 1:

1. exact todolist

...

Lean Logic Game Level 2:

1. exact andintro p s

...

Lean Logic Game Level 3:

1. exact andintro (andintro a i) (andintro o u)

...

Lean Logic Game Level 4:

1. have p := vm.left
2. exact p

...

Lean Logic Game Level 5:

1. exact h.right

...

Lean Logic Game Level 6:

1. have a := h1.left
2. have u := h2.right
3. exact andintro a u

...

Lean Logic Game Level 7:

1. exact h.left.right.left.left.right

...

Lean Logic Game Level 8:

1. have h1 := h.left.left.left
2. have h2 := h.left.left.right
3. have h3 := h.left.right
4. have h4 := h.right.right.left.left
5. have acps := andintro h3 (andintro h4 (andintro h1 h2))
6. exact acps

Mathematical Proof:

1. ((P AND S) AND A) AND NOT I AND (C AND NOT O) AND NOT U
2. P
3. S
4. A
5. C
6. A AND C AND P AND S

...

Comments and Questions

How can lean logic proofs help us construct programming languages? What other mathematical concepts can be applied in the developement of a programming language?

2.6 Week 6

Notes

This week we learned about lamda calculus and different types of propositions. Proofs using these are slightly different that other proofs we have done.

Homework

Tutorial World: Party Invites

Level 1:

1. exact todo list

...

Level 2:

1. exact and intro p s

...

Level 3:

1. exact and intro (and intro a i) (and intro o u)

...

Level 4:

1. have p := and left vm
2. exact p

...

Level 5:

1. have p := and right h
2. exact p

...

Level 6:

1. have a := and left h1
2. have u := and right h2
3. exact and intro a u

...

Level 7:

1. have h := h.left
2. have x := h.left
3. have z := x.right
4. have q := z.left
5. have r := q.left

6. have o := r.right

7. exact o

...

Level 8:

1. have h1 := and left h

2. have h2 := and right h

3. have h3 := and left h1

4. have a := and right h1

5. have ps := and left h3

6. have p := and right h2

7. have r := and left p

8. have rs := and left r

9. have ss := and right h3

10. exact and intro a (and intro rs (and intro ps ss))

Comments and Questions

My question for this week is how can some of the concepts we learned with different types of proofs be applied to lamda calculus proofs. And how can these lamda calculus proofs then be used in the developement of a programming language?

2.7 Week 7

Notes

In this week we dove deeper into lambda calculus

Homework

Comments and Questions

2.8 Week 8 and 9

Notes

In weeks 8 and 9 we worked on developing a lambda calculus interpreter. This taught me a lot about lambda calculus and how it is implemented in progrmaming languages.

Homework

Excercises 2-8:

2. Explain why a b c d reduces to (((a b) c) d) and why (a) reduces to a:

In lambda calc. expressions such as a b c d are evaluated as nested applications because the function application is inheretly left-associative. In tha case of (a), parenthesis in lambda calculus are used for grouping 2 or more variables so since a is just one variable the parenthesis can be dropped leaving us with just a.

3. How does capture avoiding substitution work? Investigate both by making relevant test cases and by looking at the source code. How is it implemented?

Capture-avoiding substitution prevents accidental overlaps of variable names when replacing variables in an expression by renaming certain variables to keep them distinct. This process ensures that each variable keeps its intended meaning, even if it needs a new name to avoid conflicts during substitution. It is implemented by checking each variable before the substitution occurs

4. Do you always get the expected result? Do all computations reduce to normal form?

I got the expected result sometimes. Not all computations reduce to normal form because some lambda expressions can cause infinite loops.

5. What is the smallest lambda expression you can find (minimal working example, MWE) that does not reduce to normal form?

The smallest expresion that does not reduce to normal form is (lambdax.x x) (lambdax.x x)

6. no question asked here

7. How does the interpreter evaluate ((lambdam.lambdan. m n) (lambdaf.lambdax. f (f x))) (lambdaf.lambdax. f (f (f x)))? Do a calculation similarly to when you evaluated ((lambdam.lambdan. m n) (lambdaf.lambdax. f (f x))) (lambdaf.lambdax. f (f (f x))) for the homework, but now follow precisely the steps taken by interpreter.py

Using the input:"((lambdam.lambdan. m n) (lambdaf.lambdax. f (f x))) (lambdaf.lambdax. f (f (f x)))"

Substitute called: replacing m with (lambdaf.(lambdax.(f (f x)))) in (lambdan.(m n))

Proceeding with substitution in lambda body for n

Substitute called: replacing m with (lambdaf.(lambdax.(f (f x)))) in (m n)

Substitute called: replacing m with (lambdaf.(lambdax.(f (f x)))) in m

Substituting variable m with (lambdaf.(lambdax.(f (f x))))

Substitute called: replacing m with (lambdaf.(lambdax.(f (f x)))) in n

Substitute called: replacing n with (lambdaf.(lambdax.(f (f (f x))))) in ((lambdaf.(lambdax.(f (f x)))) n)

Substitute called: replacing n with (lambdaf.(lambdax.(f (f (f x))))) in (lambdaf.(lambdax.(f (f x))))

Proceeding with substitution in lambda body for f

Substitute called: replacing n with (lambdaf.(lambdax.(f (f (f x))))) in (lambdax.(f (f x)))

Proceeding with substitution in lambda body for x

Substitute called: replacing n with (lambdaf.(lambdax.(f (f (f x))))) in (f (f x))

Substitute called: replacing n with (lambdaf.(lambdax.(f (f (f x))))) in f

Substitute called: replacing n with (lambdaf.(lambdax.(f (f (f x))))) in (f x)

Substitute called: replacing n with (lambdaf.(lambdax.(f (f (f x))))) in f

Substitute called: replacing n with (lambdaf.(lambdax.(f (f (f x))))) in x

Substitute called: replacing n with (lambdaf.(lambdax.(f (f (f x))))) in n

Substituting variable n with (lambdaf.(lambdax.(f (f (f x)))))

Substitute called: replacing f with (lambdaf.(lambdax.(f (f (f x))))) in (lambdax.(f (f x)))

Proceeding with substitution in lambda body for x

Substitute called: replacing f with (lambdaf.(lambdax.(f (f (f x))))) in (f (f x))

Substitute called: replacing f with (lambdaf.(lambdax.(f (f (f x))))) in f

Substituting variable f with (lambdaf.(lambdax.(f (f (f x)))))

Substitute called: replacing f with (lambdaf.(lambdax.(f (f (f x))))) in (f x)

Substitute called: replacing f with (lambdaf.(lambdax.(f (f (f x))))) in f

Substituting variable f with (lambdaf.(lambdax.(f (f (f x)))))

Substitute called: replacing f with (lambdaf.(lambdax.(f (f (f x))))) in x (lambdax.((lambdaf.(lambdax.(f (f (f x))))) ((lambdaf.(lambdax.(f (f (f x))))) x)))

8. Write out the trace of the interpreter in the format we used to picture the recursive trace of hanoi. Only write lines that contain calls to evaluate() or calls tosubstitute()[4]. Add the line numbers

12: eval(('app', ('app', ('lam', 'm', ('lam', 'n', ('app', ('var', 'm'), ('var', 'n')))), ('lam', 'f', ('lam', 'x', ('app', ('var', 'f'), ('app', ('var', 'f'), ('var', 'x')))))), ('lam', 'f', ('lam', 'x', ('app', ('var', 'f'), ('app', ('var', 'f'), ('app', ('var', 'f'), ('var', 'x'))))))))

12: eval(('app', ('lam', 'm', ('lam', 'n', ('app', ('var', 'm'), ('var', 'n')))), ('lam', 'f', ('lam', 'x', ('app', ('var', 'f'), ('app', ('var', 'f'), ('var', 'x')))))))

12: eval(('lam', 'm', ('lam', 'n', ('app', ('var', 'm'), ('var', 'n')))))

51: Returning ('lam', 'm', ('lam', 'n', ('app', ('var', 'm'), ('var', 'n'))))

39: substitute(('lam', 'n', ('app', ('var', 'm'), ('var', 'n'))), m, ('lam', 'f', ('lam', 'x', ('app', ('var', 'f'), ('app', ('var', 'f'), ('var', 'x'))))))

60: substitute(('lam', 'n', ('app', ('var', 'm'), ('var', 'n'))), m, ('lam', 'f', ('lam', 'x', ('app', ('var', 'f'), ('app', ('var', 'f'), ('var', 'x'))))))

60: substitute(('app', ('var', 'm'), ('var', 'n')), m, ('lam', 'f', ('lam', 'x', ('app', ('var', 'f'), ('app', ('var', 'f'), ('var', 'x'))))))

60: substitute(('var', 'm'), m, ('lam', 'f', ('lam', 'x', ('app', ('var', 'f'), ('app', ('var', 'f'), ('var', 'x'))))))

64: Returning ('lam', 'f', ('lam', 'x', ('app', ('var', 'f'), ('app', ('var', 'f'), ('var', 'x')))))

60: substitute(('var', 'n'), m, ('lam', 'f', ('lam', 'x', ('app', ('var', 'f'), ('app', ('var', 'f'), ('var', 'x'))))))

64: Returning ('var', 'n')

Comments and Questions

Week 8: How can we use lambda calculus to perform tasks in different orders when managing a tasks in a programming language?

Week 9: What implications does lambda calculus have on the functionality of programming languages? How is it implemented?

2.9 Week 10

Notes

Week 8 and 9 as well as assignment 3 where challening but very useful!

Homework

Questions of Reflection from Assignment 3

1) What did you find most challenging when working through Homework 8/9 and Assignment 3?

The most challenging part was implementing capture-avoiding substitution in the interpreter, especially ensuring that variable renaming didn't accidentally change the meaning of expressions. Tracking variable scopes and managing recursive evaluations required careful debugging and a solid understanding of lambda calculus mechanics, which was conceptually and technically demanding.

2) How did you come up with the key insight for Assignment 3?

The key insight came from breaking down each component of the interpreter into smaller, testable parts. By isolating the substitution mechanism and testing it with minimal expressions, I could see how substitutions worked in different cases. This modular approach helped me catch issues early on and better understand the flow of lambda expressions through the interpreter.

3) What is your most interesting takeaway from Homework 8/9 and Assignment 3?

The most interesting takeaway was realizing how lambda calculus can be used to simulate computation purely through function applications, without any inherent notion of numbers, operators, or other primitives. Building an interpreter from scratch gave me a deeper appreciation of how abstract computation principles can be directly implemented in code, connecting theory with practice in a very hands-on way

Comments and Questions

What is the application of interpreters like this in real-world systems?

3 Lessons from the Assignments

4 Conclusion

References

[BLA] Author, Title, Publisher, Year.