Week 1 Notes (Introduction to the Module)

February 20, 2024

1 Lecture Portion

1.1 Motivation

One of the main reasons why it's important to discuss logic in the context of computer science is that it is very much possible (and often necessary) to be 100% certain that a program will do what you want it to do.

There are some limitation to this, most notably the halting problem.

1.2 Notation

A statement over another statement means that the upper statement implies the lower statement. These may be nested.

eg.

$$\frac{\text{All men are mortal, Socrates is a man}}{\text{Socrates is mortal}}$$

1.3 Types of Logic

1.3.1 Aristotelian

Deals entirely with classes and belonging. Confusingly "X belongs to Y" means that the *classhood* of X belongs to Y, not that X belongs to the set Y. For example, the statement X belongs to all Y makes Y a subset of X.

There are four types of class relation:

- a: belongs to every
- e: belongs to no
- i: belongs to some
- o: does not belong to some

This results in a series of syllogisms:

$$\frac{AaB, BaC}{AaC} \quad (1) \qquad \frac{AeB, BaC}{AeC} \quad (2) \qquad \frac{AaB, BiC}{AiC} \quad (3)$$

$$\frac{AeB, BiC}{AoC} \quad (4) \qquad \frac{MaN, MeX}{NeX} \quad (5) \qquad \frac{MeN, MaX}{NeX} \quad (6)$$

$$\frac{MeN, MiX}{NoX} \quad (7) \qquad \frac{MaN, MoX}{NoX} \quad (8) \qquad \frac{PaS, RaS}{PiR} \quad (9)$$

$$\frac{PeS, RaS}{PoR} \quad (10) \qquad \frac{PiS, RaS}{PiR} \quad (11) \qquad \frac{PaS, RiS}{PiR} \quad (12)$$

$$\frac{PoS, RaS}{PoR} \quad (13) \qquad \frac{PeS, RiS}{PoR} \quad (14)$$

1.3.2 Boolean

Boolean logic uses not ($\neg$), and ($\wedge$) and or ($\vee$).

For example:

$$\frac{}{\neg(x \wedge \neg x)} \qquad \frac{x \vee y \qquad \neg x}{y}$$

This makes logic behave somewhat like arithmetic. Or ($\vee$) works a bit like addition ($+$), while and ($\wedge$) works something like muliplication ($\times$).

There is no sense of "any", "some" or "none". Everything is taken to be known as either true or false, so translating statements into their equivalent in boolean logic can be hard in some cases.

1.3.3 Frege's

Frege extended Boole's logic to include "all" and "some".

Frege's notation is unpopular and people tend to use a notation devised by Gentzen.

We use $\forall$ to mean for all, and $\exists$ to mean for some.

For example:

$$\frac{\forall x.\Phi}{\Phi[t/x]} \qquad \frac{\Phi[t/x]}{\exists x.\Phi}$$

The first means that if $x.\Phi$ holds for all x, replacing x with t will also hold. The second means that if replacing x with t holds, then $x.\Phi$ holds for at least some x.

The symbol Φ represents a string and x represents a substring location.

2 Tasks

2.1

2.1.1 Make up an example per syllogism

$\frac{AaB,BaC}{AaC}$ Mortality belongs to all men, manhood belongs to all Socrates: therefore mortality belongs to all Socrates.

$\frac{AeB,BaC}{AeC}$ No dog is green, all poodles are dogs: therefore no poodle is green.

$\frac{AaB,BiC}{AiC}$ All women are mortal, some people are women: therefore some people are mortal.

$\frac{AeB,BiC}{AoC}$ No apes have tails, some primates are apes: therefore not all primates have tails.

$\frac{MaN,MeX}{NeX}$ All buckets are concave, no inflated footballs are concave: therefore no inflated footballs are buckets.

$\frac{MeN,MaX}{NeX}$ No coconuts are purple, all ripe plums are purple: therefore no ripe plums are coconuts.

$\frac{MeN,MiX}{NoX}$ No diamonds are conductive, some forms of carbon are conductive: therefore not all forms of carbon are diamonds.

$\frac{MaN,MoX}{NoX}$ All cakes are sweet, not all foods are sweet: therefore not all foods are cake.

$\frac{PaS,RaS}{PiR}$ All cats are silly, all cats are carnivores: therefore some carnivores are silly.

$\frac{PeS,RaS}{PoR}$ No planets are undergoing fusion, all planets are celestial bodies: therefore not all celestial bodies are undergoing fusion.

$\frac{PiS,RaS}{PiR}$ Some dogs are loyal, all dogs have bones: therefore some things with bones are loyal.

$\frac{PaS,RiS}{PiR}$ All whales have fins, some whales are blue: therefore some blue things have fins.

$\frac{PoS,RaS}{PoR}$ Not all cleaning products are bleach, all cleaning products are chemicals: therefore not all chemicals are bleach.

$\frac{PeS,RiS}{PoR}$ No monkeys can fly, some monkeys like bananas: therefore not everything that likes bananas can fly.

2.1.2 Explain the famous bug in Aristotle's logic, considering $\frac{PaS,RaS}{PiR}$, with S = unicorns, P = pink and R = fluffy

Translating this into standard English gives:

All unicorns are pink, all unicorns are fluffy: therefore some fluffy things are pink.

Of course, this doesn't actually hold because it assumes that at least some unicorns exists, which isn't necessarily the case.

2.1.3 Find a written political argument, map it to premises and conclusions, and then discuss validity and soundness

For this we'll be examining the statement "It's literally giving somebody money for nothing" from the Guardian headline: "'It's literally giving somebody money for nothing': the battle to reform property leaseholds" [1]

Premises:

1. Money is being recieved
2. The thing being given is nothing

2.2

2.2.1 Attempt to create some given words from the MU system, explaining why not if not

Thoughts while attempting these:

- You can make strings of I's which have length 2^n.
- You can remove any string of I's which have length $6n$.
- You can multiply any string of I's by 2^n.
- It feels impossible to get any odd number of I's.
- No power of 2 is a multiple of 6.
- Working backwards might help.
- I think I was over-abstracting in my head. We can make use of IaddU to get odd numbers. I was thinking of series of $6n$ I's as though they were already deleting themselves or as III as though it was already a U - but it isn't and the IaddU rules comes in very handy.
- This is best summarised as "You can always remove III at the end", or $\frac{\text{xIII}}{\text{x}}$.
- In order to get MU, we may need an odd number of U's. This would require an odd multiple of 3 of I's. So a number like 3, 9, 15 etc. This again feels impossible. To get an odd multiple of 3 we would need an even multiple of 3, which we can't do using powers of 2. However, we can also do numbers of the form $5 \cdot 2^n$. We just need any multiple of 3 of the form $5 \cdot 2^n$. That also can't exist - 2 and 5 would forever be the only prime factors. Another step must be required.

- I suspect we have to make smart use of doubling. It's worth remembering that lone U's can cause us to get stuck if we're not careful.
- Perhaps we don't need another insight after all, and we can reuse the way we managed to get 5. We can do any $2^n - 3$. That still can't be of the form $3n$, though.
- There seems to be something going on to do with prime numbers - does finding the first nonprime $2^n - 3$ help? This sequence is $1, 13, 29, 61, 125, 253$. The first nonprime is 125. This feels like a dead end because we're at a power of 5. I suspect that's all we'll get - non-3 primes and their powers.
- We can actually get any $2^n - 3m$. Does this help?
- Let's put that aside for a second, I want to try something.
  MIIIIIU MIIIIIUIIIIIU MIIUUIIUU MIIUUII MIIUUIIIIUUII
- Actually, let's create a set of the things that can happen numerically to the number of I's.
- This is $\{\times 2, -3\}$.
- This can't ever produce 0, I don't think.
- No, it can't. I'll put the proof of this in the section below for MU.

MI This is the axiomatic word - it can simply be asserted.

MIU MI (IaddU) MIU

MII MI (double) MII

MIIIII MI (double) MII (double) MIIII (double) MIIIIIIII (IaddU) MIIIIIIIIU (bang) MIIIIIUU (pop) MIIIII

ABC This one isn't possible - the only letters that can exist are M, I (both introduced in the axiom) and U (introduced via IaddU).

MIIUIIU (proved above) MIIIII (double) MIIIIIIIII (bang×2) MIIUIIU

II This one isn't possible - none of the rules allow the M to be removed.

MU This one isn't possible - the number of I's cannot be of the form $3n$, and therefore cannot be 0.

Proof: The rules have the following effects on the number of I's:

- IaddU: No effect.
- Double: $\times 2$.
- Bang: -3.
- Pop: No effect.

The initial number of I's is 1, a number of the form $3n+1$. The double rule will turn any number of the form $3n+1$ into a number of the form $3n-1$ and vice-versa. The bang rule will not change the form of a number of the form $3n+1$ or $3n-1$.

Therefore, the number of I's will always be of the form $3n+1$ or $3n-1$, and never $3n$.

2.2.2 Write a program to create all MU system theorems

Completed. See generate.rs for the code.

2.2.3 Write a program that can test if a string is a theorem in the MU system

This is as simple as counting I's and ensuring the count is not of the form $3n$. Any string with this property can be constructed.

Proof:

- Arbirarily large numbers of consecutive I's of the form 2^n can be created of both the forms $3n+1$ and $3n-1$, because powers of 2 always alternate between these two forms.
- Arbirarily large numbers of consecutive I's can be created of both the forms $3n+1$ and $3n-1$, because you can simply create a larger power of 2 of the same form and remove any amount of I's of the form $3n$ using the sequence (IaddU) (bang) (pop) for each 3 items.
- Arbitraryly placed U's also pose no problem. Simply construct a string with $3\text{U}_{count} + \text{I}_{count}$ I's (this will still be of a correct form mod 3 since we are adding a multiple of 3). Then it's trivial to do the desired (bang) steps.

See validate.rs for the code.

References

[1] J. Kollewe, "'it's literally giving somebody money for nothing': the battle to reform property leaseholds," *The Guardian*.

Appendix

main.rs

```rust
use std::io;

pub mod generate;
pub mod validate;

enum Menu {
    Generate,
    Validate,
}

fn main() {
    let choice: Menu;
    loop {
        println!("Please choose an option:");
        println!("1. Generate MU system theorems");
        println!("2. Test if a string is part of the MU system");
        let mut input = String::new();
        io::stdin().read_line(&mut input).expect("Failed to read line");
        choice = match input.trim().parse() {
            Ok(1) => Menu::Generate,
            Ok(2) => Menu::Validate,
            _ => continue,
        };
        break;
    }
    match choice {
        Menu::Generate => generate::generate(),
        Menu::Validate => validate::validate(),
    }
}
```

generate.rs

```rust
use std::io;
use std::str;

pub fn generate() {
    let start_string: [u8;2] = *b"MI";
    let amount: usize;
    loop {
        println!("How many MU system theorems would you like to generate?");
        let mut input = String::new();
        io::stdin().read_line(&mut input).expect("Failed to read line");
        amount = match input.trim().parse() {
            Ok(n) => n,
            Err(_) => continue,
        };
```

```
        break;
    }
    let mut strings: Vec<Vec<u8>> = vec![start_string.to_vec()];
    for counter in 0..amount {
        println!("{}", str::from_utf8(&strings[counter]).unwrap());
        strings.append(&mut get_IaddUs(&strings[counter]));
        strings.append(&mut get_doubles(&strings[counter]));
        strings.append(&mut get_bangs(&strings[counter]));
        strings.append(&mut get_pops(&strings[counter]));
    }
}

// Add a U at the end if there's an I at the end
#[allow(non_snake_case)]
fn get_IaddUs(premise: &[u8]) -> Vec<Vec<u8>> {
    if premise[premise.len()-1] != b'I' { return vec![]; }
    let mut conclusion = premise.to_vec();
    conclusion.push(b'U');
    return vec![conclusion];
}

// Double everything after an M
fn get_doubles(premise: &[u8]) -> Vec<Vec<u8>> {
    if premise[0] != b'M' { return vec![]; }
    let mut conclusion = premise.to_vec();
    conclusion.extend_from_slice(&premise[1..]);
    return vec![conclusion];
}

// Replace any III with U
fn get_bangs(premise: &[u8]) -> Vec<Vec<u8>> {
    return get_replaced_slice_results(premise, b"III", b"U");
}

// Replace any UU with nothing
fn get_pops(premise: &[u8]) -> Vec<Vec<u8>> {
    return get_replaced_slice_results(premise, b"UU", b"");
}

fn get_replaced_slice_results(
    premise: &[u8],
    replacee: &[u8],
    replacer: &[u8]
) -> Vec<Vec<u8>> {
    if premise.len() < replacee.len() { return vec![]; }
    let mut conclusions: Vec<Vec<u8>> = vec![];
    for i in 0..=premise.len() - replacee.len() {
        if premise[i..i+replacee.len()].to_vec() == replacee.to_vec() {
            let mut conclusion = premise.to_vec();
            conclusion.splice(i..i+replacee.len(), replacer.iter().cloned());
            conclusions.push(conclusion);
        }
    }
    return conclusions;
}
```

validate.rs

```rust
use std::io;

pub fn validate() {
    println!("Please enter a string:");
    let mut input = String::new();
    io::stdin().read_line(&mut input).expect("Failed to read line");
    let input = input.trim();
    let input_bytes = input.as_bytes();
    if validate_theorem(input_bytes) {
        println!("Yes, {} is a valid MU system theorem", input);
    } else {
        println!("No, {} is not a valid MU system theorem", input);
    }
}

fn validate_theorem(theorem: &[u8]) -> bool {
    if theorem[0] != b'M' { return false; }
    #[allow(non_snake_case)]
    let mut I_counter: usize = 0;
    for character in &theorem[1..] {
        match character {
            b'I' => I_counter += 1,
            b'U' => (),
            _ => return false,
        }
    }
    return I_counter % 3 != 0;
}
```