MAT605 Logic and Foundations with Haskell

Marius Furter

May 10, 2023

1 Resources

The video material for the course may be found either on SWITCHcast or on YouTube. In both places the videos contain chapter headings to help you navigate the material.

2 Learning goals

The following is a summary of the learning goals of the course by week. Each learning goal for the theory will be marked with one of the following symbols:

🙂 **(essentials)** These points are essential knowledge for passing the course. If you complete all of them, you should have no trouble passing the exam.

🤔 **(food for thought)** These points are more advanced and might require more work to understand. They are not required in order to get a passing grade, but some of these topics might appear on the exam.

🤯 **(exploding head)** These points are completely optional. They are either difficult, tedious, or things I included for your culture. They will not appear on the exam, but they might help you understand the other material better.

Aside from the learning goals, I will also provide some questions for you to contemplate.

Week 1

Haskell 1 :: Setup This video covers how to get Haskell installed on your computer. I recommend using GHC together with Visual Studio Code. In order to make the two work together, you need to let GHCup install all the stuff it asks you to, in particular you need the Haskell language server. In VSCode you will need to install the `Haskell` and `Haskell Syntax Highlighting` extensions.

Course Intro This video introduces the course. Watching it is optional.

Logic 1 :: Introduction This video introduces the logic portion of the course. It doesn't contain any essential material, but it sets up some ideas that will be helpful later. I would recommend watching it.

Question: Think about some mathematical statements of your favorite area. What does it mean for such statements to be 'true'? How would you translate this to formulas in a logical system?

Logic 2 :: Naive Propositional Logic This video covers the basics of propositional logic in an informal manner. We will see all of this in more detail in a few weeks. You probably have already encountered all of this material in your studies, but I included it so as to not assume any prior knowledge. If you feel like you satisfy the learning goals, feel free to skip this video. If you are unclear on some points, I would recommend skimming the video using the chapter feature.

- 🙂 Understand the *syntax* of propositional logic. Be able to recognize and write well-formed formulas.
- 🙂 Know the *truth table definitions* for the usual logical connectives ($\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$). Be able to calculate the truth value of propositional logc formulas.
- 🙂 Know the definition of *logical validity* and *logical equivalence* for propositional formulas. Be able to demonstrate these using truth tables.
- 🙂 Be able to prove new logical equivalences from old ones by *substitution*.

Week 2

Haskell 2 :: Basic Operations This video covers basic operations in Haskell such as arithmetic, comparison, lists, ranges, list comprehension, and simple functions. It follows chapter 2 of the "Learn you a Haskell for great good" tutorial.

- 🙂 Be able to do *basic arithmetic* in Haskell
- 🙂 Be able to *compare* objects
- 🙂 Be able to define *lists and ranges* and understand their limitations
- 🙂 Understand the syntax for *list comprehension*
- 🙂 Know how to write *basic functions*
- 🙂 Know the difference between *prefix and infix functions*. Be able to convert each into the other
- 🙂 Be able to define *tuples* and understand their limitations

Haskell 3 :: Types and Typeclasses This video covers the type system in Haskell. It follows chapter 3 of the "Learn you a Haskell for great good" tutorial.

- Know the *basic types* listed in the online tutorial. Be able to create objects of each of them.
- Know the syntax for declaring the *types of function* with one or more arguments
- Know how to use *type variables and typeclasses* in type declarations for functions
- Be familiar with the most common *typeclasses* (Eq, Ord, Num, Integral, Floating)

Logic 3 :: Naive First Order Logic This video covers first order logic informally based on examples. You probably already are familiar with much of the material in this video. I would recommend watching it if you haven't thought a lot about the structure of first order logic formulas before. You will then be better prepared for the formal definitions that will come later.

- Be able to parse well-formed first order logic formulas into a *parsing tree*
- Be able to *translate* between precise mathematical statements and first order logic formulas
- Be able to convert between *restricted quantifiers* and unrestricted ones

Week 3

Haskell 4 :: Functions This video covers how to define functions using pattern matching and guards. It also discusses `where`, `let`, and `case`.

- Know the *syntax for pattern matching* numbers, tuples, and lists.
- Know the *syntax* for defining functions using *guards*.
- Know the *syntax* for `where` and `let`.

Logic 4 :: Informal Proof Theory This video covers all the rules necessary to prove first order logic formulas. The main goal is for you to know and be able to apply these rules. Since most of you will already have quite some experience in proving things, this material should seem familiar. However, you might not have explicitly given the rules much thought. For additional examples and exercises, I recommend looking at Velleman's book "How to prove it".

- Know the proof rules for all first order logic connectives. I have summarized the rules here.

- 🤔 Be able to *write basic proofs* using the rules.
- 🙂 Be able to recognize the *logical equivalences* involving quantifiers presented at the end of the video.

Week 4

Haskell 5 :: Implementing Logical Functions In this video we implement our own version of the type Bool along with functions acting on Bool.

- 🙂 Be able to write functions on the level of the translation functions and the logical operators `not`, `&&`, ||.
- 🙂 Be able to understand the recursive definitions for `and`, `or`, `elem`, `all`, `any`, and `filter`.
- 🤔 Be able to implement functions on the level of `and`, `or` using recursion.

Logic 5 :: Natural Deduction This video covers the formal proof system called natural deduction. It roughly follows chapter 2 of Chiswell and Hodges' "Mathematical Logic". The rules are basically the same as those presented in last week's video on informal proof theory.

- 🙂 Know what a *derivation* is.
- 🙂 Know what a *sequent* is.
- 🙂 Be able to *recognize the natural deduction rules* for each connective and be able to identify if a derivation is correct.
- 🙂 Be able to *recognize whether a given sequent rule follows* from a given natural deduction rule.
- 🤔 Be able to *derive sequent rules* from natural deduction rules.
- 🤔 Be able to *write derivations* that prove simple sequents (ca. 2-4 rule applications).
- 🤯 Think about proof by contradiction and the law of the excluded middle. In what situations in life would you accept an answer based on 'proof by contradiction'. Can you find situations where a double negation $\neg\neg\varphi$ does not seem equivalent to φ. If you would create your own mathematical world, what level of proof would you require to call statements 'true'?

Week 5

Haskell 6 :: Implementing Sets In this video we implement a new datatype for sets and some basic functions associated to it. There are no exam-relevant learning goals from for this video.

Week 6

Haskell 7 :: Implementing Relations In this video we review the definition of relations and implement a type and functions for them.

- 🙂 Know the definition of a *relation* and *relation composition.*
- 🙂 Know the definitions for *reflexivity, symmetry,* and *transitivity.*

Logic 6 :: Language of Propositional Logic This video covers the formal syntax for propositional logic.

- 🙂 Understand the *inductive definition for LP formulas.*
- 🙂 Be able to *recognize whether an LP expression is a formula.*
- 🙂 Be able to determine *initial segments* of a formula and *calculate their depths.*
- 🙂 Understand the *statement* of the *unique parsing theorem.*
- 🤔 Understand the *proof* of the *unique parsing theorem* (including the preceding lemma).

Week 7

Haskell 7 :: Implementing Functions We implement functions as sets of pairs in Haskell.

- 🙂 Know the definition of a *function.*
- 🙂 Know the definitions for *injectivity, surjectivity* and *bijectivity.*

Week 8

Logic 7 :: Semantics for Propositional Logic We cover the semantic definitions for propositional logic which assign each LP-formula a truth value in a given σ-structure.

- 🙂 Know the definition of a *σ-structure.*
- 🙂 Understand how truth values are extended to all $LP(\sigma)$-formulas
- 🙂 Be able to determine if a given σ-structure is a model for an $LP(\sigma)$-formula.

Logic 8 :: Soundness of Natural Deduction for LP We prove soundness of the natural deduction proof calculus for propositional logic.

- 🙂 Understand the *statement of the soundness theorem.*

Haskell 9 :: Natural Numbers We implement the natural numbers, arithmetic and comparison.

- 🙂 Understand the *inductive definition of Nat*.

Week 9

Logic 9 :: Completeness of Natural Deduction for LP We prove completeness of the natural deduction proof calculus for propositional logic.

- 🙂 Understand the *statement of the lemmas* and how they fit together to prove the completeness theorem

Haskell 10 :: Folding over Lists We see how the folding pattern can be used to write compact implementations.

- 🤔 Understand the definitions of `foldr` and `foldl`.

Haskell 11 :: Partial and Multivalued Functions We discuss two methods for dealing with partial and multivalued functions.

- 🙂 Understand the definition of the `Maybe` datatype.
- 🙂 Be able to use `Maybe` and lists to implement partial and multivalued functions.

Week 10

Haskell 12 :: Typeclasses for Natural Numbers We implement typeclasses for the natural numbers.

- 🙂 Understand how to *implement instances of Ord, Enum and Num* for your own datatypes.

Haskell 13 :: Integers from Natural Numbers We build the integers from the Natural Numbers in two ways.

- 🙂 Understand how to *implement instances of Eq* for your own datatypes.

Naive Set Theory We cover the basic definitions of set theory without axiomatic foundations.

- 🙂 Understand all of the *definitions* presented in the video.
- 🙂 Be able to *recognize the set theoretic identities* in the proposition.

Week 11

Axiomatic Set Theory We build the axiomatic foundations of set theory using the Zermelo-Fraenkel axioms with choice (ZFC).

- 🙂 Understand and recognize all *ZFC axioms*. (You do not need to be able to recite them from memory, but you should be able to tell whether a given formula is one of the axioms or not.)

- 🤔 Be able to argue that a given class is a set.

3 Example exam questions

The written exam will consist of two types of questions that will cover both theory and the basic syntax of Haskell. The first 2/3 of the exam will test your knowledge of the essential learning goals. These questions will not require much thought. If you get about 80% of these correct, you will pass the exam. If you get all of them right, that would be about a 4.7. The remaining 1/3 will consist of questions that either require more thinking or come from the food for thought learning goals. To get a 6 you need to score around 88% of points overall. I will adjust the grading scale if it turns out to be too strict. You will also be allowed to write down reasons for your answers to the multiple choice question if you are unsure about a question so that I can assign partial points if your reasoning is partially correct.

3.1 Example essential questions

The learning goal: 'Be able to recognize well-formed formulas of propositional logic' could translate into the exam question:

Which of the following formulas are well-formed formulas of propositional logic?

- ☐ $\forall x \exists y((p_1(x) \land p_2(x)) \to p_3(y))$
- ☐ $((p_1 \lor p_2) \land p_3)$
- ☐ $(p_1 \to p_2 \to p_3)$
- ☐ $((p_1 \neg p_2) \lor p_3)$

The learning goals: 'Know how to write basic functions' and 'Know the syntax for pattern matching tuples' might translate to the question:

Which of the following pieces of code define valid functions in Haskell?

- ☐ `fun x y = x+y`
- ☐ `Fun x = x+x`
- ☐ `fun(x,y) = x+y`

☐ `fun (x,y) = x+y`

The learning goal 'Know what a sequent is' might translate to the question *What is the correct definition of a sequent?*

☐ A sequent is an expression $\Gamma \vdash \varphi$. It is correct, if whenever all the assumptions in Γ are true, then so is φ.

☐ A sequent is a type of formal proof of the form $\Gamma \vdash \varphi$ that uses only the sequent rules of natural deduction.

☐ A sequent is an expression $\Gamma \vdash \varphi$. It is correct, if there exists a derivation with conclusion φ using only assumptions occurring in Γ.

☐ A sequent is a rule of the form $\Gamma \vdash \varphi$ that may be used in natural deduction derivation to prove φ from Γ.

3.2 Example thinking questions

The learning goals: 'Be able to translate between precise mathematical statements and first order logic formulas' and 'Be able to translate between restricted and unrestricted quantification' might translate to the question:

Which of the following first order logic formulas capture the statement: 'For all x in A there exists y in A satisfying $x < y$' ?

☐ $\forall x \exists y (x \in A \wedge (y \in A \wedge \phi < \psi))$

☐ $\exists y \in A \forall x \in A (x < y)$

☐ $\forall x \in A \exists y \in A (x < y)$

☐ $\forall x \exists y (x \in A \to (y \in A \wedge x < y))$

The learning goal: 'Be able to derive new logical equivalence from old ones by substitution' might translate into the following question:

Consider the following logical equivalences:

$$\Phi \to \Psi \cong (\neg\Phi) \vee \Psi, \qquad \neg(\neg\Phi) \cong \Phi,$$

$$\neg(\Phi \vee \Psi) \cong (\neg\Phi) \wedge (\neg\Psi), \qquad \neg(\Phi \wedge \Psi) \cong (\neg\Phi) \vee (\neg\Psi).$$

Use these to show that

$$(\neg(\Phi \to \Psi)) \to (\neg\Psi) \cong ((\neg\Phi) \vee \Psi) \vee (\neg\Psi)$$

What does this tell you about this formula?

The learning goal: 'Be able to write derivations that prove simple sequents' might translate to:

Provide a derivation that proves $\{\phi, \psi\} \vdash (\phi \vee \tau) \wedge \psi$