Formal Methods & Functional Programming

Janis Hutz
https://janishutz.com

February 28, 2026

"A funny quote by a professor"
- Prof. Dr. Professor Name, YEAR

FS2026, ETHZ
Summary of the Lectures

Contents

1 Haskell

Haskell is a functional programming language. As such, its functions can be thought of as being similar to mathematical functions and as such are side-effect-free.

Haskell's type system is very robust and an interesting topic to learn about. The basic data types you already know from other programming languages are also present here. This includes all primitives like integers, floating point numbers, chars and booleans.

Strings are handled similarly to how `C` does it, in that strings are char arrays.

Arrays however are dynamic length in Haskell as opposed to many other statically typed programming languages.

Since Haskell is an imperative language (i.e. you describe *what* you want achieve) as opposed to a declarative language (i.e. you describe *how* you achieve what you want to achieve), there are no loops in Haskell, as loops don't appear in mathematical formulas and functions either. What we can do however is recursion and this is the main way of doing iterative work in Haskell.

Additionally, Haskell features *lazy evaluation* (i.e. statements are evaluated only as needed) as opposed to *eager evaluation* (i.e. statemetns are evaluated immediately).

In this course the `Glasgow Haskell Compiler`, short `ghc` is used. Installation is really easy (as long as you're on Linux)

1.1 The bad news: The syntax

In short: It's quite bad, but you will get used to it and some of the (arguably) poor looking syntax choices will start to make more sense.

You should use 2 space indents (yuck) and indents matter, just like in Python.

We can use binary functions in infix or prefix notation, i.e. `x ‘mod‘ z` and `mod x z` are equivalent.

For integers the following functions are available: Normal arithmetic operations `+, -, *, /, mod, abs`, as well as `^` which is used for exponentiation.

To use prefix notation on non-alphanumeric function names, wrap them in parenthesis like this: `(+) x z`. Using `+ x z` does not work.

We can use the normal comparison operators that return a boolean on evaluation. ***Booleans*** are `True` and `False`

2 Formal Reasoning

2.1 Formal proofs

Given a language like $\mathcal{L} = \{\oplus, \otimes, +, \times\}$, and derivation rules

- α: If $+$, then $\otimes$
- β: If $+$, then $\times$
- γ: If $\otimes$ and $\times$, then $\oplus$
- δ: $+$ holds

or displayed using graphical notation:

$$\frac{+}{\otimes}\,\alpha \qquad \frac{+}{\times}\,\beta$$

$$\frac{\otimes \quad \times}{\oplus}\,\gamma \qquad \frac{-}{+}\,\delta$$

Rules like δ above are also commonly referred to as an *axiom*.

To prove $\oplus$ in this language, we can either write the following or draw a derivation tree:

- $+$ holds by δ
- $\otimes$ holds by α with 1.
- $\times$ holds by β with 1.
- $\oplus$ holds by γ with 2 and 3

Or as derivation tree

$$\frac{\dfrac{-}{+}\,\delta}{\otimes}\,\alpha \qquad \frac{\dfrac{-}{+}\,\delta}{\times}\,\beta$$

$$\frac{\otimes \qquad \times}{\oplus}\,\gamma$$

2.2 Natural deduction

The rules from above here are used to construct derivations under assumptions, e.g. $A_1, \dots, A_n \vdash A$, which is read as "A follows from $A_1, \dots, A_n$".

The derivations are always represented as derivation trees and a ***proof*** is a derivation whose root has no assumptions.

Since we have to prove a statement, we have to draw the derivation trees from the bottom up, with the goal of reaching an axiom or a rule that is an assumption using the other rules of the rule set.

2.3 Propositional logic

2.3.1 Syntax

Definition 2.3.1 For a set of variables $\mathcal{V}$, the ***language of propositional logic*** $\mathcal{L}_P$ is the smallest set where

- $X \in \mathcal{L}_P$ if $X \in \mathcal{V}$
- $\bot \in \mathcal{L}_P$
- $A \wedge B \in \mathcal{L}_P$ if $A \in \mathcal{L}_P$ and $B \in \mathcal{L}_P$
- $A \vee B \in \mathcal{L}_P$ if $A \in \mathcal{L}_P$ and $B \in \mathcal{L}_P$
- $A \to B \in \mathcal{L}_P$ if $A \in \mathcal{L}_P$ and $B \in \mathcal{L}_P$

2.3.2 Semantics

Definition 2.3.2 *(Valuation)* $\sigma : \mathcal{V} \to \{\texttt{True}, \texttt{False}\}$ maps variables to truth values. They are the simple models (i.e. *interpretations*). ***Valuations*** is the set of valuations.

Definition 2.3.3 *(Satisfiability)* smallest relation $\models \subseteq$ *Valuations* $\times \mathcal{L}_P$ such that

- $\sigma \models X$ if $\sigma(X) = \texttt{True}$
- $\sigma \models A \wedge B$ if $\sigma \models A$ and $\sigma \models B$
- $\sigma \models A \vee B$ if $\sigma \models A$ or $\sigma \models B$
- $\sigma \models A \to B$ if whenever $\sigma \models A$ then $\sigma \models B$

Definition 2.3.4 *(satisfiable formula)* A formula $A \in \mathcal{L}_P$ is ***satisfiable*** if $\sigma \models A$ for **some** valuation σ

Definition 2.3.5 *(tautology, valid formula)* A formula $A \in \mathcal{L}_P$ is ***valid*** (a ***tautology***) if $\sigma \models A$ for **all** valuations σ

Definition 2.3.6 *(Semantic entailment)* $A_1, \dots, A_n \models A$ if $\forall \sigma$ we have $\sigma \models A_1, \dots, \sigma \models A_n$, then $\sigma \models A$

2.3.3 Requirements for a deductive system

The derivation rules (syntactic entailment) and truth tables (semantic entailment) should agree. For that we have two requirements, for $\Gamma = A_1, \dots, A_n$ a collection of formulas:

- ***Soundness:*** If $\Gamma \vdash A$ can be derived, then $\Gamma \models A$
- ***Completeness:*** If $\Gamma \models A$, then $\Gamma \vdash A$ can be derived

Decidability is also desirable, i.e. having checks of the attributes be of low complexity.

2.3.4 Natural deduction for propositional formulas

Definition 2.3.7 *(Sequent)* Is an assertion of form $A_1, \dots, A_n \vdash A$, with $A, A_1, \dots, A_n$ being propositional formulas.

Intuition: A follows from the A_i and if the system is sound, the A_i semantically entail A.

Definition 2.3.8 *(Axiom)* is the starting point (usually the leaves) of the derivation trees and are usually of the form

$$\frac{}{\dots, A, \dots \vdash A} \text{ axiom}$$

i.e. when coming up with a derivation tree for a ***proof***, we want to reach a leaf where A is contained in Γ.

Definition 2.3.9 *(Proof)* of A is a derivation tree with $\texttt{root} \vdash A$. If a deductive system is *sound*, then A is a tautology.

There are two kinds of rules, ***introduce*** and ***eliminate*** connectives. If you are confused about the order when applying them when coming up with a deduction tree, they are oriented top-down, so e.g. the introduction rule is inverted when coming up with the deduction tree.

If all rules are sound (i.e. they preserve semantic entailment), then the logic is sound.

2.3.5 Derivation rules for propositional logic

Remember that E means *elimination* and I means *introduction*, with L and R being the side (so ER means elimination on the right)

2.3.5.1 Conjunction

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge\text{-I} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge\text{-EL} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge\text{-ER}$$

2.3.5.2 Disjunction

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee\text{-IL} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee\text{-IR} \qquad \frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C} \vee\text{-E}$$

2.3.5.3 Implication

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to\text{-I} \qquad \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B} \to\text{-E}$$

2.3.5.4 Others

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A} \bot\text{-E} \qquad \frac{\Gamma \vdash \neg A \qquad \Gamma \vdash A}{\Gamma \vdash B} \neg\text{-E} \qquad \frac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash A} \text{ RAA} \qquad \frac{}{\dots, A, \dots \vdash A} \text{ axiom}$$

2.4 First-Order Logic

2.4.1 Syntax

Definition 2.4.1 *(Signature)* consists of a set of function symbols $\mathcal{F}$ and a set of predicate symbols $\mathcal{P}$, as well as their arities.

We write f^k (or p^k, for predicates) to indicate that it has *arity* $k \in \mathbb{N}$. Constant functions have arity 0, linear functions have arity 1, thus, the arity of a given function (or predicate) is given by the number of parameters to uniquely describe it, minus one.

Definition 2.4.2 *(Term)* is the terms of first-order logic is smallest set, where (with $\mathcal{V}$ again a set of variables)

1. $x \in$ *Term* if $x \in \mathcal{V}$
2. $f^n(t_1, \dots, t_n) \in$ *Term* if $f^n \in \mathcal{F}$ and $t_i \in$ *Term*, $\forall 1 \leq i \leq n$ (description of form of formulas)

Definition 2.4.3 *(Form)* is the formulas of first-order logic, is the smallest set where

1. $\bot \in$ *Form*
2. $p^n(t_1, \dots, t_n) \in$ *Form* if $p^n \in \mathcal{P}$ and $t_j \in$ *Term*, $\forall 1 \leq j \leq n$ (description of form of predicates)
3. $A \circ B \in$ *Form* if $A \in$ *Form*, $B \in$ *Form* and $\circ \in \{\wedge, \vee, \to\}$ (i.e. formulas with logic symbols)
4. $Qx.A \in$ *Form* if $A \in$ *Form*, $x \in \mathcal{V}$ and $Q \in \{\forall, \exists\}$ (i.e. formulas with quantifiers)

2.4.1.1 Binding

Definition 2.4.4 *(Bound variable)* A variable that occurs in a quantifier in scope (blue in example below)

Definition 2.4.5 *(Free variable)* A variable that is not bound by a quantifier in scope (red in example below)

Example 2.4.6 $(q(x) \vee \exists x.\forall y.p(f(x), z) \wedge q(a)) \vee \forall x.r(x, z, g(x))$

Definition 2.4.7 *(α-conversion)* A ***bound*** variable can be renamed at any time, but must preserve binding structure.

2.4.1.2 Precedences

$\neg > \wedge > \vee > \to$, with $>$ a total order of precedences. Quantifiers extend as far right as possible, bounded by the end of line or a going out of scope by closing parenthesis.

2.4.2 Semantics

Definition 2.4.8 *(Structure)* is a pair $\mathcal{S} = \langle U_\mathcal{S}, I_\mathcal{S} \rangle$, where $U_\mathcal{S}$ is the universe and it is a non-empty set and $I_\mathcal{S}$ is a mapping with

1. $I_\mathcal{S}(p^n)$ is an n-ary relation on $U_\mathcal{S}$ for $p^n \in \mathcal{P}$ (short $p^\mathcal{S}$)
2. $I_\mathcal{S}(f^n)$ is an n-ary (total) function on $U_\mathcal{S}$ for $f^n \in \mathcal{F}$ short ($f^\mathcal{S}$)

Intuition: The $I_\mathcal{S}$ is essentially assigning to each predicate and formula its definition in the universe of the structure, noted as a relation.

Definition 2.4.9 *(Interpretation)* is a pair $\mathcal{I} = \langle \mathcal{S}, v \rangle$, with $v : \mathcal{V} \to U_\mathcal{S}$ a valuation

Intuition: it assigns definitions to the formulas and predicates (through the structure), as well as values to the variables (through the valuation).

Definition 2.4.10 *(Value)* of a term t under $\mathcal{I}$ is written as $\mathcal{I}(t)$ and defined by $\mathcal{I}(x) = v(x)$ for $x \in \mathcal{V}$ and $\mathcal{I}(f(t_1, \dots, t_n)) = f^\mathcal{S}(\mathcal{I}(t_1), \dots, \mathcal{I}(t_n))$

Definition 2.4.11 *(Satisfiability)* $\models \subseteq$ Interpretations $\times$ *Form* is the smallest relation satisfying

$$\begin{aligned}
&\langle \mathcal{S}, v \rangle \models p(t_1, \dots, t_n) && \text{if } (\mathcal{I}(t_1), \dots, \mathcal{I}(t_n)) \in p^\mathcal{S} \text{ where } \mathcal{I} = \langle \mathcal{S}, v \rangle \\
&\langle \mathcal{S}, v \rangle \models \forall x.A && \text{if } \langle \mathcal{S}, v[x \mapsto a] \rangle \models A, \text{ for all } a \in \mathbb{U}_\mathcal{S} \\
&\langle \mathcal{S}, v \rangle \models \exists x.A && \text{if } \langle \mathcal{S}, v[x \mapsto a] \rangle \models A, \text{ for some } a \in \mathbb{U}_\mathcal{S}
\end{aligned}$$

Definition 2.4.12 *(Model)* When $\langle \mathcal{S}, v \rangle \models A$, then $\langle \mathcal{S}, v \rangle$ is a ***model*** for A. If A does not have free variables, the satisfaction does not depend on the valuation v and we write $\mathcal{S} \models A$

Definition 2.4.13 *(Validity)* When every interpretation is a model, we write $\models A$, and we say that A is ***valid***

Definition 2.4.14 *(Satisfiability)* A is ***satisfiable***, if there exists at least one model for A.

Example 2.4.15 Given $\forall x.p(x, s(x))$, we a model would be

$$\begin{aligned} U_{\mathcal{S}} &= \mathbb{N} \\ p^{\mathcal{S}} &= \{(m,n) \mid m,n \in U_{\mathcal{S}} \text{ and } m < n\} \\ s^{\mathcal{S}} &= \text{ successor function on } U_{\mathcal{S}}, \text{ i.e. } s^{\mathcal{S}}(x) = x+1 \end{aligned}$$

2.4.2.1 Substitution

Definition 2.4.16 *(Substitution)* Replace all occurrences of a free variable x with some term t in A. To denote a substitution, we write $A[x \mapsto t]$. Important All free variables in t must still be free in $A[x \mapsto t]$. If that would not be true anymore, do a α-conversion first.

2.4.3 Quantifiers

2.4.3.1 Universal quantification

Additional rules are needed for the universal quantifier. * side condition is that x is not free in any assumption of Γ

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x.A}\ \forall\text{-I*} \qquad \frac{\Gamma \vdash \forall x.A}{\Gamma \vdash A[x \to t]}\ \forall\text{-E}$$

Again here be mindful not to capture free variables.

2.4.3.2 Existential quantification

Additional rules are needed for the existential quantifier. ** side condition is that x is neither free in B nor free in Γ

$$\frac{\Gamma \vdash A[x \mapsto t]}{\Gamma \vdash \exists x.A}\ \exists\text{-I} \qquad \frac{\Gamma \vdash \exists x.A \qquad \Gamma, A \vdash B}{\Gamma \vdash A[x \to t]}\ \exists\text{-E**}$$

Again here be mindful not to capture free variables.

2.5 Equality

Since equality is such an important concept, it isn't just a predicate, but a separate First-Order Logic (FOL), called ***FOL with equality***.

The language is extended by $t_1 = t_2 \in \mathit{Form}$ if $t_1, t_2 \in \mathit{Term}$, the semantic entailment $\models$ is also extended by "$\mathcal{I} \models t_1 = t_2$ if $\mathcal{I}(t_1) = \mathcal{I}(t_2)$". This definition is the exact intuition of equality of two terms, in that they are equal if their value under the interpretation is equal.

Equality is an equivalence relation, so the following rules apply (ref = reflexivity, sym = symmetry, trans = transitivity):

$$\frac{}{\Gamma \vdash t = t}\ \text{ref} \qquad \frac{\Gamma \vdash t = s}{\Gamma \vdash s = t}\ \text{sym} \qquad \frac{\Gamma \vdash t = s \qquad \Gamma \vdash s = r}{\Gamma \vdash s = r}\ \text{sym}$$

Equality is also a congruence on terms and (definable) relations

$$\frac{\Gamma \vdash t_1 = s_1 \qquad \dots \qquad \Gamma \vdash t_n = s_n}{\Gamma f(t_1, \dots, t_n) = f(s_1, \dots, s_n)}\ cong_1$$

$$\frac{\Gamma \vdash t_1 = s_1 \qquad \dots \qquad \Gamma \vdash t_n = s_n \qquad \Gamma \vdash p(t_1, \dots, t_n)}{\Gamma p(s_1, \dots, s_n)}\ cong_2$$