FMFP - Lecture Notes Week 1

Ruben Schenk, ruben.schenk@inf.ethz.ch

March 9, 2022

1 Introduction & Basic Haskell Syntax

1.1 Example: GCD

The **GCD problem** is given as follows: Compute the greatest common divisor of two natural numbers. We have the following *specifications:* Let $x, y \in \mathcal{N}$ be given. The number z is the **greatest common divisor** of x and y iff. $z \mid x$ and $z \mid y$ and there is no z', with $z' > z$, such that $z' \mid x$ and $z' \mid y$. Here, $z \mid x \equiv \exists a \in \mathcal{N}. a \cdot z = x$.

The problem specification is not **constructive,** i.e. it does not describe how the GCD should be computed.

1.1.1 Imperative GCD

```
public static int gcd(int x, int y) {
    while(x != y) {
        if(x > y) {
            x = x - y;
        } else {
            y = y - x;
        }
    }
    return x;
}
```

The **imperative GCD**, as shown above, consists of control flow statements and assignments. Assignments change the computer's *state.* To understand `gcd`, one must understand how its state changes.

Poor man's reasoning would be to simulate and track the memory content during execution. A better way would be to use *Hoare logic* in the form of $\{P\}$ prog $\{Q\}$. Formal reasoning is possible, but not easy!

1.1.2 Functional GCD

```
gcd x y
    | x == y    = x
    | x > y     = gcd (x - y) y
    | otherwise = gcd x       (y - x)
```

The functional way formalizes *what* should be computed, rather than *how.* This is an algorithm, provided we have also specified how functions are executed.

1.2 Basic Concepts in Functional Programming

1.2.1 Referential Transparency

Functions compute values. But functions also *are* values: we can compute and return them. It is important to note that functions in functional programming have **no side effects:** `f(x)` always returns the same value. This in contrast to other programming languages we've known so far. Consider the following `Java` example:

```
class Test {
    static int y = 0;
    static int f(int x) {
        y = y + 1;
        return y;
    }
}
```

```
public static void main(String[] args) {
    System.out.println(f(0));
    System.out.println(f(0));
}
```

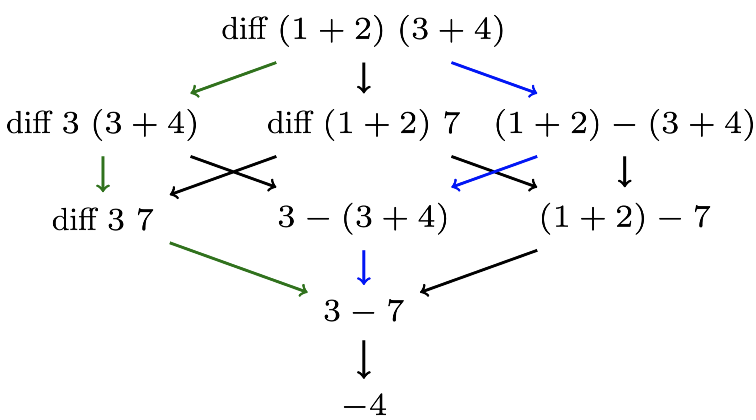
One will immediately see that this prints out `0` and then `1`, which means that `f(0)` returns different values with the same input.

Since functions have no side effects, we can reason with the more easily in mathematics. This property is also called **referential transparency:** an expression evaluates to the same value in every context.

1.2.2 Evaluation

An **evaluation strategy** defines how and when expressions are evaluated during the execution of a program. We differ between two strategies:

- *Eager evaluation:* evaluate arguments first. Also called "call-by-value", corresponds to the left (green) path in the figure below.
- *Lazy evaluation:* evaluate arguments only when needed (used by Haskell). Also called "call-by-need" (or "left-most/outermost"), corresponds to the right (blue) path in the figure below.



1.3 Basic Haskell Syntax

1.3.1 Syntax and Types

We present the basic syntax principles in the following code example:

```
gcd x y        -- functions and arguments start with lower-case letters
    | x == y    = x
    | x > y     = gcd (x - y) y          -- arguments are written in sequence and
    | otherwise = gcd x       (y - x)    -- separated by whitespace
```

Furthermore, functions consist of different cases and a program consists of several definitions:

```
myConstant = 5

afunction y1 y2 ... ym
    | guard1 = expr1
    | guard2 = expr2
    ...
    | guardm = exprm

anotherFucntion z1 z2 ... zk = ...
```

Indentation determines the separation of definitions. All function definitions must start at the same indentation level. If a definition requires $n > 1$ lines, we indent lines 2 to n further. This leads to the following *recommended layout:*

```
f1 x1 x2
    | a long guard which may go over
      a number of lines
        = a long expression that can also go over
          several lines
    | g2 = expr2

f2 x1 x2 x3 = ...
```

1.3.2 Functions

Functions live in a global scope. This means that a function can be called from any other. Example:

```
f x y = ...
g x = ... h ...
h z = ... f ... g ...
```

We can define functions and variables in local scope with `let` and `where`:

```
let x1 = e1
    ...
    xn = en
in e
```

2 Natural Deduction

2.1 Introduction to Natural Deduction

2.1.1 Abstract Example (without Assumptions)

Consider the following "meaningless" language:

$$\mathcal{L} = \{\oplus, \otimes, \times, +\}$$

We furthermore state the following *rules:*

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

Our goal is to prove $\oplus$. We might proceed as follows:

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

We might also present this proof as a **derivation tree:**

$$\dfrac{\dfrac{\dfrac{}{+}\,\delta}{\otimes}\,\alpha \qquad \dfrac{\dfrac{}{+}\,\delta}{\times}\,\beta}{\oplus}\,\gamma$$

2.1.2 Abstract Example (with Assumptions)

We revisit the previous example by slightly changing one of our rules:

- α: If $+$, then $\otimes$
- β: If $+$, then $\times$
- γ: If $\otimes$ and $\times$, then $\oplus$
- δ: We may assume $+$ when proving $\oplus$

We can build the following proof system. In this system, Γ is the set of assumptions we make during our proof:

$$\dfrac{}{\ldots, A, \ldots \vdash A}\ \textit{axiom}$$

$$\dfrac{\Gamma \vdash +}{\Gamma \vdash \otimes}\,\alpha \qquad \dfrac{\Gamma \vdash +}{\Gamma \vdash \times}\,\beta$$

$$\dfrac{\Gamma \vdash \otimes \quad \Gamma \vdash \times}{\Gamma \vdash \oplus}\,\gamma \qquad \dfrac{\Gamma, + \vdash \oplus}{\Gamma \vdash \oplus}\,\delta$$

Our derivation tree from previously changes slightly to the following:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{+\vdash +}\,\textit{axiom}}{+\vdash \otimes}\,\alpha \qquad \dfrac{\dfrac{}{+\vdash +}\,\textit{axiom}}{+\vdash \times}\,\beta}{+\vdash \oplus}\,\gamma}{\vdash \oplus}\,\delta$$

2.1.3 Summary

Rules are used to construct derivations under assumptions. $A_1, ..., A_n \vdash A$ reads as "A follows from $A_1, ..., A_n$".

Derivations are trees as shown in the examples above.

A **proof** is a derivation whose root has no assumptions.

2.2 Propositional Logic

2.2.1 Syntax

Propositions are built from a collection of variables and closed under disjunction, conjunction, implication, etc. More formally, let a set $\mathcal{V}$ of variables be given. $\mathcal{L}_P$, the **language of propositional logic,** is the smallest set where:

- $X \in \mathcal{L}_P$ if $X \in \mathcal{V}$
- $\perp \in \mathcal{L}_P$
- $A \wedge B \in \mathcal{L}_P$ 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 $B \in \mathcal{L}_P$
- $A \rightarrow B \in \mathcal{L}_P$ if $A \in \mathcal{L}_P$ and $B \in \mathcal{L}_P$

In the following: X ranges over variables, A and B over formulae.

2.2.2 Semantics

A **valuation** $\sigma : \mathcal{V} \rightarrow \{\text{True, False}\}$ is a function mapping variables to truth values. Valuations are simple kinds of models (or interpretations). We denote the set of valuations as Valuations.

Satisfiability is the smallest relation $\vDash \subseteq \text{Valuations} \times \mathcal{L}_P$ such that:

- $\sigma \vDash X$ if $\sigma(X) = \text{True}$
- $\sigma \vDash A \wedge B$ if $\sigma \vDash A$ and $\sigma \vDash B$
- $\sigma \vDash A \vee B$ if $\sigma \vDash A$ or $\sigma \vDash B$
- $\sigma \vDash A \rightarrow B$ if whenever $\sigma \vDash A$ then $\sigma \vDash B$

Note that $\sigma \nvDash \perp$ for every $\sigma \in \text{Valuations}$.

We furthermore introduce the following characteristics about propositional logic:

- A formula $A \in \mathcal{L}_P$ is **satisfiable** if $\sigma \vDash A$, for some valuation σ
- A formula $A \in \mathcal{L}_P$ is **valid** (a **tautology**) if $\sigma \vDash A$, for all valuations σ
- **Semantic entailment:** $A_1, ..., A_n \vDash A$ if for all σ, if $\sigma \vDash A_1, ..., \sigma \vDash A_n$ then $\sigma \vDash A$

Examples:

- $X \wedge Y$ is satisfiable as $\sigma \vDash X \wedge Y$ for $\sigma(X) = \sigma(Y) = \text{True}$
- $X \rightarrow X$ is valid
- $\neg X,\ X \vee Y \vDash Y$ holds as $\sigma \vDash \neg X$ and $\sigma \vDash X \vee Y$ constraint σ to $\sigma(X) = \text{False}$ and $\sigma(Y) = \text{True}$, so $\sigma \vDash Y$

2.2.3 Requirements

We need some **requirements** for *deductive systems.* The main requirement is that syntactic entailment $\vdash$ (derivation rules) and semantic entailment *vDash* (truth tables) should agree. This requirement has two parts:

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

Here, $\Gamma \equiv A_1, ..., A_n$ is some collection of formulae.

2.2.4 Natural Deduction for Propositional Logic

A **sequent** is an assertion (judgement) of the form $A_1, ..., A_n \vdash A$, where all $A, A_1, ..., A_n$ are propositional formulae. A **proof** of A is a derivation tree with root $\vdash A$. If the deductive system is sound, then A is a tautology.

Conjunction **Conjunction** proposes rules of two kinds: *introduce* and *eliminate* connectives. The rules are given as follows:

$$\dfrac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \land B}\land\text{-}I \qquad \dfrac{\Gamma \vdash A \land B}{\Gamma \vdash A}\land\text{-}EL \qquad \dfrac{\Gamma \vdash A \land B}{\Gamma \vdash B}\land\text{-}ER$$

Example: The following figure shows an example derivation using conjunction rules.

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash X \land (Y \land Z)}\text{axiom}}{\Gamma \vdash X}\land\text{-}EL \qquad \dfrac{\dfrac{\dfrac{}{\Gamma \vdash X \land (Y \land Z)}\text{axiom}}{\Gamma \vdash Y \land Z}\land\text{-}ER}{\Gamma \vdash Z}\land\text{-}ER}{\underbrace{X \land (Y \land Z)}_{\equiv \Gamma} \vdash X \land Z}\land\text{-}I$$

Implication The rules for **implication** are given as follows:

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

Disjunction The rules for **disjunction** are given as follows:

$$\dfrac{\Gamma \vdash A}{\Gamma \vdash A \lor B}\lor\text{-}IL \qquad \dfrac{\Gamma \vdash B}{\Gamma \vdash A \lor B}\lor\text{-}IR$$

$$\dfrac{\Gamma \vdash A \lor B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C}\lor\text{-}E$$