# Modelling Decision Trees with Functional Programming

## Summary

I will investigate how models in context of functional programming can lead to an improvement in programming accessibility and correctness. The specific model which I will formalise is the notion of how decision trees learn models.

## Aims and objectives:

1. **(Week 1)** Perform literary review of previous work on DSLs and decision trees. Simultaneously research advanced methods in Haskell.
2. **(Week 2)** Produce a model which describes decision trees in functional notation. Given time, I will implement and execute the model in Haskell.
3. **(Week 3)** Discuss ways in which composition could be used to create a meaningful notion of a language.
4. **(Week 4)** I will conclude by discussing potential improvements with respect to compile time analysis, efficiency, usability, security and extensibility, if DSLs can be adopted.

## Deliverables:

1. Construction of a model that describes how decision trees classify.
2. A representation of the model in functional notation and a possible implementation of this model in a functional programming language, specifically Haskell.

## Introduction

Individuals without direct knowledge of the specifics of software engineering face challenges when verifying the correctness of code with respect to a given model. This is especially challenging in languages such as C. I contend that languages which are “close to the hardware” force the implementor to think as a machine, rather than a human. This restricts the ability of people outside of software engineering to program computers. I advocate that the description of algorithms that exist within our heads should match the implementation of the algorithms as closely as possible. I feel that the functional programming paradigm allows one, versed in small amounts of mathematical literature, to do precisely this.

A concise model for data structures allows one to quickly verify and understand it with respect to a specification, without dealing with hardware related bureaucracies. In order to come up with models in languages such as C can be incredibly time consuming, using methods such as axiomatic semantics with Hoare axioms. Models can be further analysed and decomposed into primitive, reusable operations and a DSL can be constructed. I also point out the difference between an Application Programming Interface (API) and a DSL. I see an API as a construct that operates on a known data type, whereas a DSL operates on an abstract data type.

Oliver Goldstein:
og14775