Purely Functional Algorithm Specification
Course Overview

Jan van Eijck
CWI & ILLC, Amsterdam

ESSLLI, Opole, August 6–10, 2012

homepages.cwi.nl/~jve/courses/12/esslli12/

Algorithms

An algorithm is an effective method expressed as a list of instructions describing a computation for calculating a result. Algorithms have to be written in human readable form, either using pseudocode (natural language looking like executable code), a high level specification language like Dijkstra's Guarded Command Language, or an executable formal specification formalism such as Z.

Algorithms are usually specified in imperative style.

Functional View on Imperative Algorithms

The course will adopt a purely functional view on the key ingredients of imperative programming: while loops, repeat loops, and for loops, and demonstrate how this can be used for specifying (executable) algorithms, and for automated testing of Hoare correctness statements about these algorithms.

Inspiration for this was the talk by Leslie Lamport at CWI, Amsterdam, on the executable algorithm specification language PlusCal [3], plus Edsger W. Dijkstra, "EWD472: Guarded commands, nondeterminacy and formal derivation of programs" [1]. Instead of formal program derivation, we demonstrate test automation of Hoare style assertions.

Prerequisites

This is course has no formal prerequisites at all. The introduction will be at a serious speed, while remaining accessible to students with a wide variety of backgrounds. Anyone with a willingness to learn formal methods should be able to follow. Students with some previous experience with (functional) programming will be at an advantage, though.

To benefit the most, install the Haskell platform on your laptop, and attack some of the exercises.

Please don't worry about the fact that this course was classified by the ESSLLI program commitee as advanced.

Haskell as a Tool for Functional Algorithm Specification

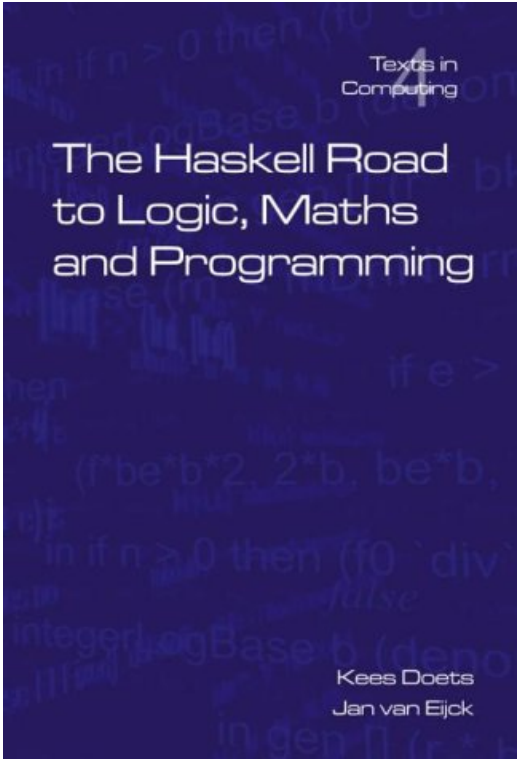
Our tool for functional algorithm specification will be the language Haskell.

`www.haskell.org`

`http://hackage.haskell.org/platform/`

The first lecture will be devoted to a lightning introduction to Haskell.

The Haskell Road [2]



Functional Imperative Style

This lecture explains how imperative algorithms can be written directly in functional style.

We show how to define while loops as functional programs, and how this trick can be used to write what we call functional imperative code.

Reasoning About Functions

This lecture explains reasoning about functions using assertions taken from Hoare logic and dynamic logic.

$$\{P\}\ C\ \{Q\}.$$

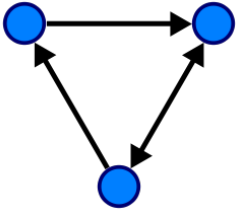
Algorithm Specification for Assertive Coding

We show how to specify preconditions, postconditions, assertions and invariants, and how to wrap these around functional code or functional imperative code. We illustrate the use of this for writing programs for automated testing of code that is wrapped in appropriate assertions. We call this assertive coding.

An assertive version of a function f is a function f' that behaves exactly like f as long as f complies with its specification, and aborts with error otherwise.

Graph Algorithms

This lecture discusses a number of well-known graph algorithms, develops testable specifications for them, and uses the specifications to write assertive (self-testing) versions of the algorithms.

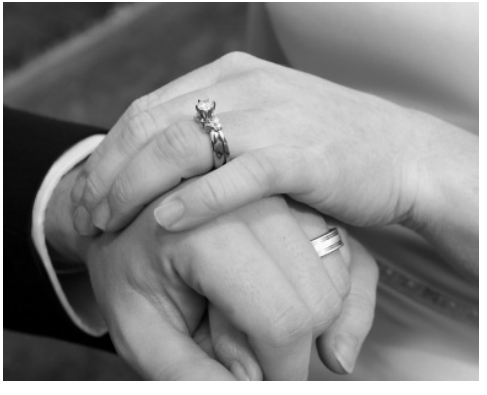


Algorithms for Matching and Task Assignment

This lecture deals with algorithms for matching and assignment.

The matching problem is the problem of connecting nodes in a bipartite graph, while making sure that certain requirements are met.

Exemplar for this is the stable marriage algorithm.



Fair Division Algorithms

This lecture discusses fair division algorithms, develops testable specifications for them, and uses the specifications to write self-testing versions of the algorithms.

The problem of cutting a cake among N participant in such a way that every participant is satisfied with her share serves as an exemplar.

References

[1] E.W. Dijkstra. Guarded commands, nondeterminacy and the formal derivation of programs. Communications of the ACM, 18:453–457, 1975.

[2] K. Doets and J. van Eijck. The Haskell Road to Logic, Maths and Programming, volume 4 of Texts in Computing. College Publications, London, 2004.

[3] Leslie Lamport. The PlusCal algorithm language. In Martin Leucker and Carroll Morgan, editors, Theoretical Aspects of Computing – ICTAC 2009, number 5684 in Lecture Notes in Computer Science, pages 36–60. Springer, 2009.