CSci 2041: Advanced Programming Principles
Overview of What We have Studied

Gopalan Nadathur

Department of Computer Science and Engineering
University of Minnesota

Lectures in Spring 2015

The Original Objectives for the Course

The goals we enunciated at the beginning of the term

- To understand techniques and principles for constructing complex programs that are *efficient* and *correct*
- To do this in a hands on way using a practical language that supports sophisticated programming techniques
- To learn the principles in a way that we could use them in the context of other languages

An auxiliary goal: to learn OCaml to a point where we might be inclined to use it in future programming tasks

In this review, we will look at what we learned in terms of principles and comment on their generality

A (Partial) Catalogue of Principles Studied

Some of the different kinds of ideas we studied during the term

- Programming as an activity that is not subservient to the structure of the machine used for computing
  - seeing computation as expression evaluation
  - thinking of data independently of how it is represented
  - Understanding the power and generality of recursion, observing iteration as only one special form
- Understanding the fundamental role of types in structuring programs
  - Types as a mechanism for conceptualizing and organizing data; sum and product types as universal mechanisms
  - Constructing programs based on the way types structure data
  - Flexibility in typing, polymorphism, type checking, type inference

A (Partial) Catalogue of Principles Studied (Contd)

- Treating functions as first-class objects
  - Understanding the continuum between objects like integers and functions
  - Issues of scope, the need for closures, related binding concepts
  - Usefulness in building higher-order functions, true polymorphism, treating control via continuations
  - Implicit parallelism, the map-reduce paradigm
- Reasoning about Program Behaviour
  - Understanding what it means for a program to be correct; preconditions, postconditions, invariants
  - Appreciating the close connection between recursion/iteration and inductive correctness arguments
  - Inductively defined types and associated induction principles

A (Partial) Catalogue of Principles Studied (Contd)

- Understanding how to assess the efficiency of programs
  - Appreciating what aspects to consider: execution time, stack and heap space
  - Learning how to assess such costs: constructing and solving recurrences, understanding structured data costs
  - Common programming techniques geared to efficiency: memoization, divide-and-conquer, accumulators
- Programming aspects beyond expression evaluation and their principled use
  - state as a means for communication, implicit treatment of state, references and assignment
  - Input/output as a means for communication with external processes
  - Mechanisms for dealing with exceptional situations in computing

A (Partial) Catalogue of Principles Studied (Contd)

- Search in Programming
  - Using search as a means for visualizing/structuring computation
  - Techniques to simplify search-based programming: exceptions, success and failure continuations
- Modularity in Programming

  The meaning of modularity, its use in design, its importance to various critical attributes of software
- Order in the Evaluation of Expressions
  - Common approaches like lazy and eager evaluation, differences at a behavioural level
  - delayed evaluation, streams as an underlying mechanism for a producer-consumer approach to programming

Aspects that We Did Not Manage to Discuss

- Concurrency as a Programming Paradigm
  - Refers to the idiom of realizing power from communication between processes running asynchronously
  - stream based computation provides part of the flavour
  - an idiom of growing importance because of the increasing number of mobile devices
- Implementation related issues such as garbage collection

Assessing What You Got Out of the Course

Here are some questions to ask yourself

- Did the course help you see programming as an analytical activity, rather than "just something to be done somehow?"
- Do you feel you have learnt some ways to structure your programs that you can use in other language settings?
- Did you learn about some new programming approaches, e.g. using search, streams, modularity, etc?
- Did you find OCaml interesting to the point where you would want to use it in your own pet project?

I would be happy if the answer to at least some of these questions is "Yes"