Harvard School of Engineering and Applied Sciences — CS 153: Compilers

Self Assessment

Students taking CS 153 should be comfortable programming in OCaml and be familiar with topics covered in CSCI 61/CSCI E-61, including data representation, assembly and machine programming (particularly how the stack and heap work).

If you have taken CSCI 61 and CSCI 51, you are adequately prepared to take CSCI 153. If you have not taken these courses (or if you want to check how much you remember!) the questions below are designed to help you figure out if you have adequate preparation. Note that if you are fairly comfortable with programming you are likely to be able to self-study to gain sufficient knowledge of OCaml and of machine organization, but you should plan on spending significant time to do so.

1. Write an OCaml function `int2bits : int -> bool list` that takes as input an integer and returns a 2s complement representation of the integer as a list of booleans, with the most significant bit at the head of the list. The list should have length 32. Write test cases for your function.

2. Write an OCaml program that opens a file and reads integers from the file (one per line), calls your function `int2bits` on each integer and prints the result. Use the stdlib function `string_of_int` to convert lines of the file to integers. Your program should correctly handle any exceptions raised by `string_of_int` or the file IO functions, and continue as much as possible.

3. Implement in OCaml a binary tree of integers. Call the type of your binary tree `bintree`. Interior nodes and leaves should contain an integer. Interior nodes have a left child and a right child, each of which might optionally be `None`. The tree should be sorted (i.e., all integers in the left subtree of a node n should be less than or equal to the integer held in node n, and all integers in the right subtree of n should be greater than or equal to the integer in n). The integer of each node and the children of interior nodes should be imperatively updatable (i.e., use references). Your implementation should include the following functions. Write test cases for your functions to help ensure you have implemented them correctly.

   a. `insert : bintree option ref -> int -> unit` which inserts an integer into a tree. The insertion should be imperative.

   b. `check : bintree -> bool` which checks that the binary tree meets its invariants, including that interior nodes should have at least one child, and that the tree is sorted.

   c. `count : bintree -> int -> int` which counts how many times an integer occurs in a tree.

   d. `balance : bintree option ref -> unit` which imperatively balances the tree.

4. Write a recursive C function `int factorial(int n)` that computes factorials. Suppose you invoke your function `factorial(3)`. What's the maximum number of stack frames for the `factorial` function that will be on the stack at any time? What information is in each stack frame? (You can assume that arguments are passed on the stack. You don't need to know the exact details of the layout of a stack frame, but you should be comfortable with what kind of information is stored in a stack frame.)