Homework-4

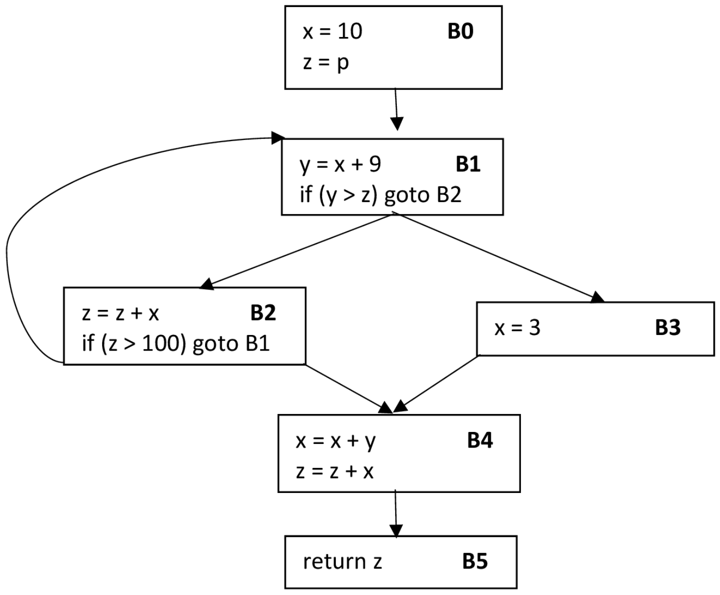

Figure 1: Control flow graph.

- Convert the CFG in Figure 1 into a minimal SSA form. Show all intermediate computations, i.e., dominator tree, dominance frontiers, liveness, placement of phi nodes, and reaching definitions. [15]
- Consider a routine `foo` below that takes the pointer argument `arr` as input.

  ```
  void foo(int *arr);
  ```

  Let's say `arr` is pointing to object `A`. There are no other references to `A` stored anywhere else in the program except the argument `arr`. You can assume that at the start of `foo`, the program can't compute the address of A from any other memory location or register except the value of `arr`. Write a static analysis for the LLVM IR to determine if object `A` can be updated before returning from `foo`. For example, the statement "arr[i] = 20" updates the object A, whereas the statement "t = arr[i]" doesn't update `A`. In LLVM IR, the store instruction is used to update the content of a memory location. You need to use the LLVM IR in the SSA form that uses virtual registers and phi-nodes whenever possible (instead of using stack locations for every variable). Your static analysis should work for any definition of `foo`. [20]

- Discuss why the following implementation of the write-barrier is incorrect. [10]

```
void write_barrier(void **pp, void *p) {
   if (concurrent mark is running) {
       if (pp reached-bit is 1 and p reached-bit is 0) {
            set the reached-bit of p to 1
            add p to WB_Unscanned list
       }
   }
   *pp = p;
}
```

- The problem with the above implementation can be fixed by inserting locks in the concurrent mark and write barrier algorithms. Insert locks in the concurrent mark algorithm discussed in the class and the `write_barrier` routine in the previous question to solve the concurrency issue. You are not supposed to change the other parts of the algorithm. Write the modified algorithm. [15]

1 How to submit

Submit your handwritten answers in class before the lecture.