**CS164 Programming Languages and Compilers** **Spring 2024**

# Discussion Worksheet 11: Register Allocation

## 1 Liveness Analysis

Before we can compute the register interference graph for a program, we need to run liveness analysis.

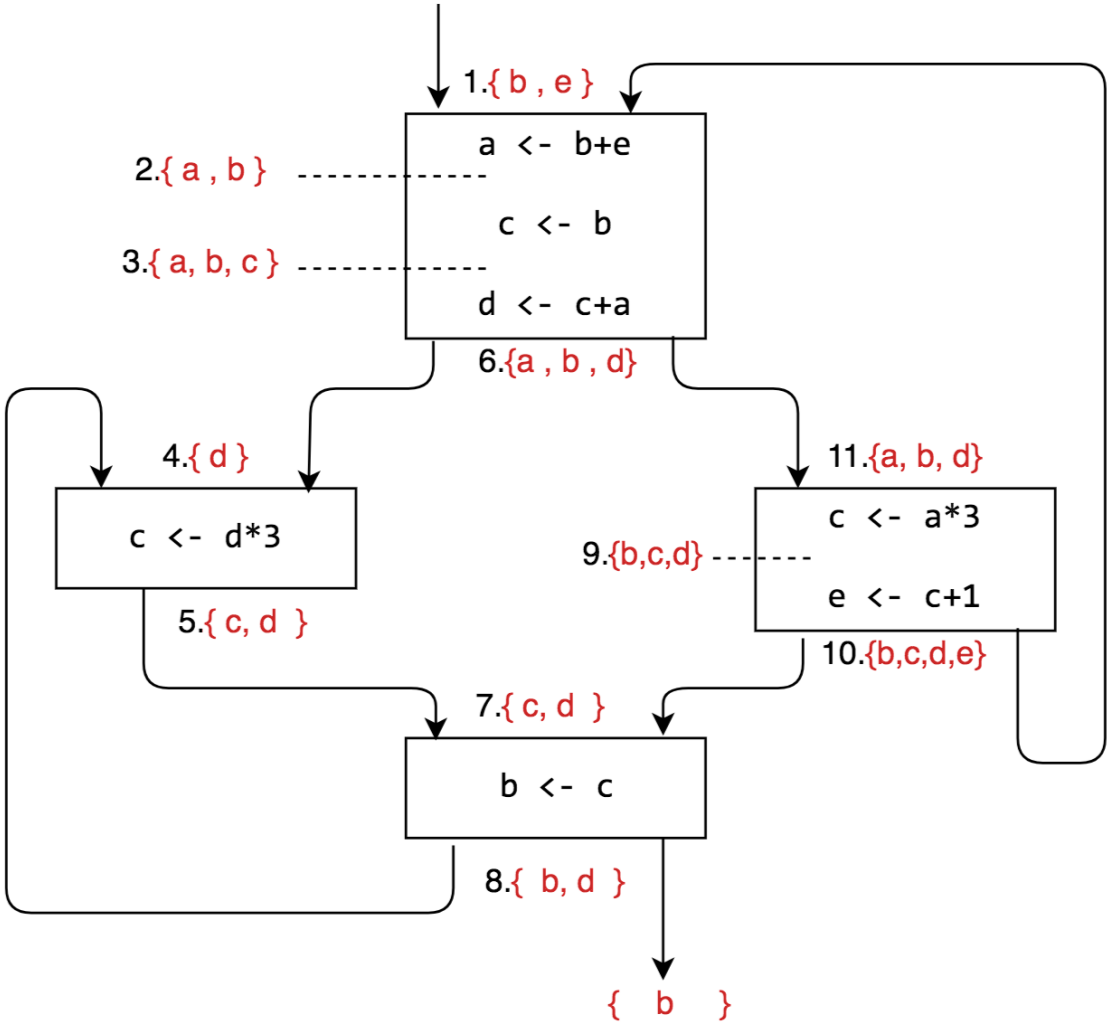
Recall the generalized dataflow transfer functions for liveness analysis. For a given statement $s$, the variables that are live before ($In(s)$) and after ($Out(s)$) the statement is executed are given by

$$Out(s) = \cup_{s' \in succ(s)} In(s')$$

$$In(s) = Gen(s) \cup (Out(s) - Kill(s))$$

Where $Gen(s)$ is the set of variables that are used before an assignment in $s$, and $Kill(s)$ is the set of variables that are assigned to in $s$.

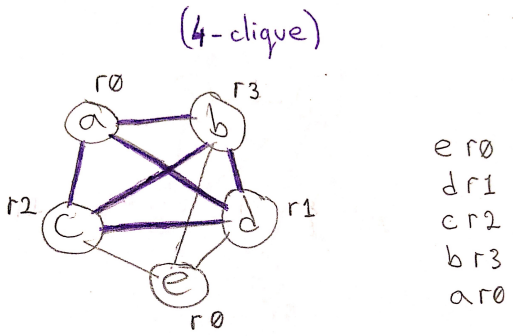
**Exercise 1.** Run liveness analysis on the following CFG. Assume `b` is live at exit. Here `<-` means `:=`.



## 2 Register Allocation

**Exercise 2** Recall that a Register Interference Graph has as nodes each register in the program, and edges between registers that are live at the same program point.

**2.1** Construct a Register Interference Graph (RIG) for the program from Exercise 1.




**2.2** We can claim that if the RIG is $k$-colorable then we only need $k$ registers to store all the variables. Why is this true?

Solution: Register allocation can be reduced to the problem of K-coloring the resulting graph, where K is the number of registers available on the target architecture. No two vertices sharing an interference edge may be assigned the same color, and vertices sharing a preference edge should be assigned the same color if possible. You can create an RIG to model register allocation by creating an edge between two register nodes if they are ever alive at the same point in a program.

**2.3** Color the graph using optimistic coloring with 4 colors. Is this the minimum number of colors we need such that no two nodes of the same color have a connecting edge? Solution: Yes, the the **a-b-c-d** clique prevents us from using less than 4 colors.

**2.4** Suppose we only have 3 registers to allocate, thus we must spill a temporary. Which temporary would you suggest to spill? What are some heuristics you can think of to help decide which temporaries to spill? Solution: The heuristics presented in lecture are:

(a) Spill temporaries with most conflicts

(b) Spill temporaries with few definitions and uses

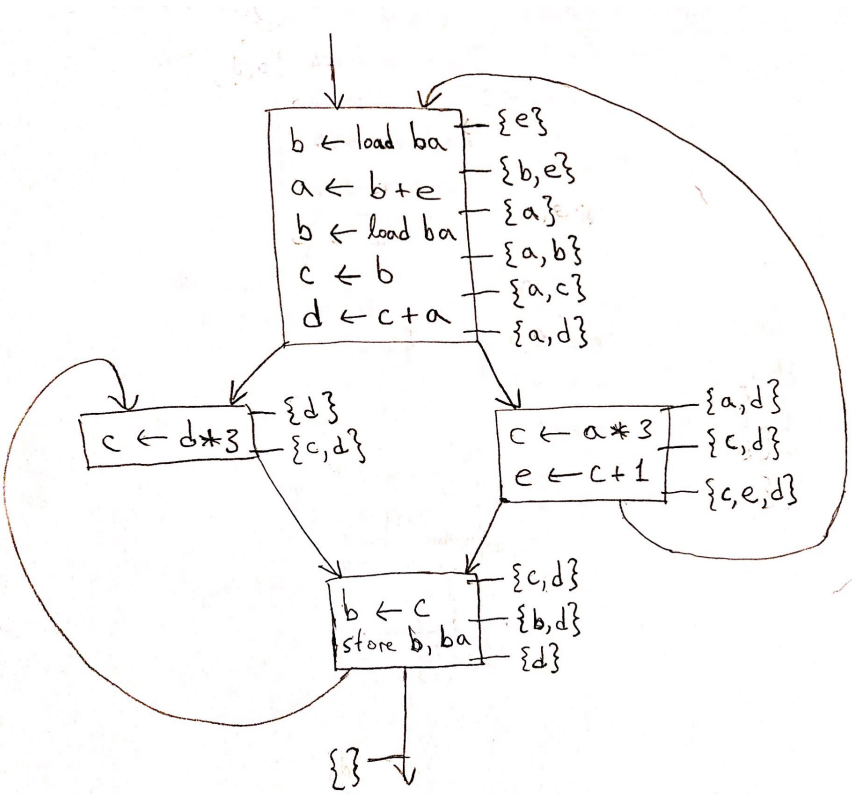
(c) Avoid spilling in inner loops

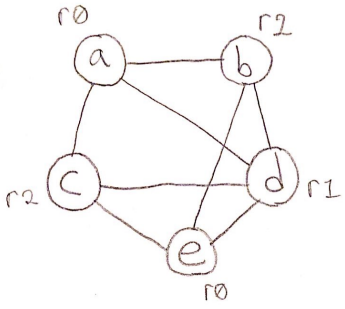
Modern Register Allocation Heuristics can be very complicated.
We must spill one of **a, b, c,** or **d** otherwise we will still need 4 registers. We might spill **b** because it has many conflicts, or **d** because it has only one assignment and one use.

**2.5** Suppose we chose to spill **b**. Annotate the CFG so that we can store **b** on the stack, and re-compute register allocation.

Solution: Pick an address to store **b**: ba.

1. Before **a** ← **b** + **e** insert **b** ← **load ba**
2. Before **c** ← **b** insert **b** ← **load ba**
3. After **b** ← **c** insert **store b, ba**




Still none have < 3 neighbors.

Optimistic coloring:

e r0
d r1
c r2
a r0
b r2