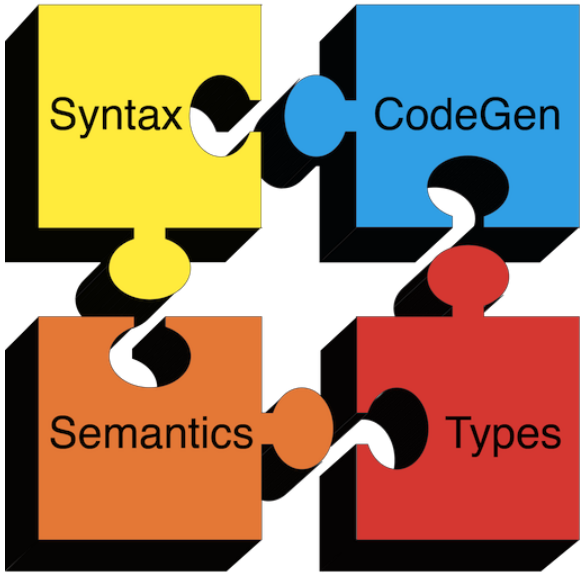


Compilers

Register Allocation

Register Allocation

- Intermediate code uses unlimited temporaries
  - Simplifies code generation and optimization
  - Complicates final translation to assembly

- Typical intermediate code uses too many temporaries

Register Allocation

- The problem:

  Rewrite the intermediate code to use no more temporaries than there are machine registers

- Method:
  - Assign multiple temporaries to each register
  - But without changing the program behavior

Register Allocation

- Consider the program

```
a := c + d
e := a + b
f := e - 1
```

- Assume a & e dead after use
  - A dead temporary can be "reused"

- Can allocate a, e, and f all to one register (r_1):

$$r_1 := r_2 + r_3$$
$$r_1 := r_1 + r_4$$
$$r_1 := r_1 - 1$$

Register Allocation

- Register allocation is as old as compilers
  - Register allocation was used in the original FORTRAN compiler in the ‘50s
  - Very crude algorithms

- A breakthrough came in 1980
  - Register allocation scheme based on graph coloring
  - Relatively simple, global and works well in practice

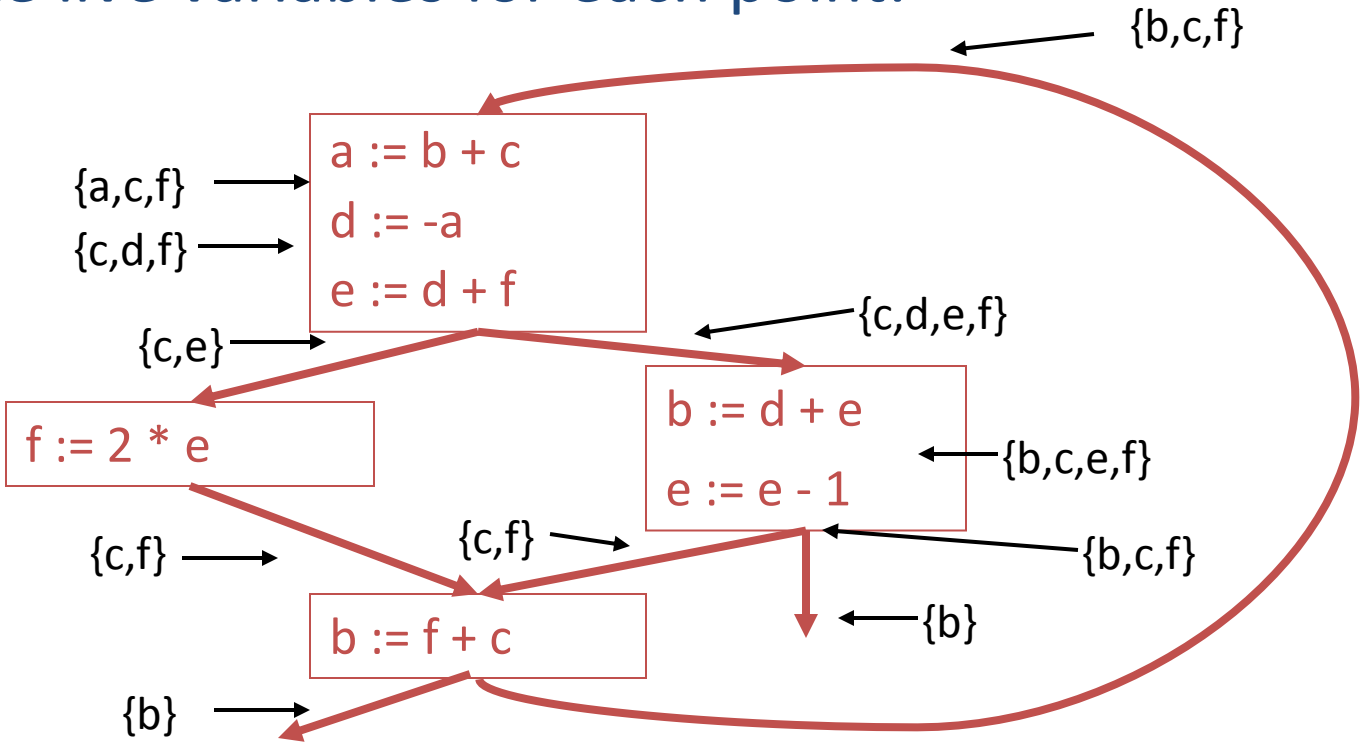
Register Allocation

Temporaries t_1 and t_2 can share the same register if <u>at any point in the program at most one</u> of t_1 or t_2 is live.

Or

If t_1 and t_2 are live at the same time, they cannot share a register

- Compute live variables for each point:



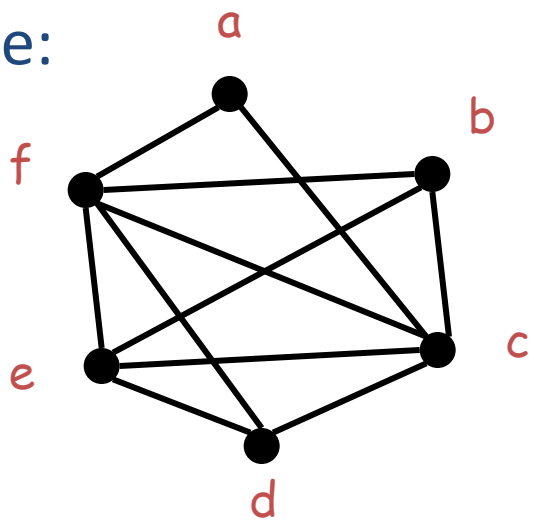
Register Allocation

- Construct an undirected graph
  - A node for each temporary
  - An edge between t_1 and t_2 if they are live simultaneously at some point in the program

- This is the *register interference graph* (RIG)
  - Two temporaries can be allocated to the same register if there is no edge connecting them

Register Allocation

- For our example:

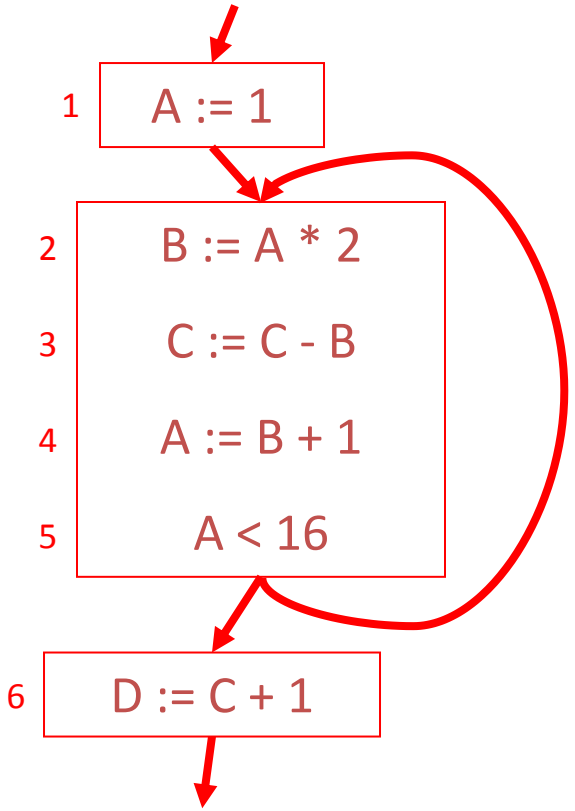


- E.g., b and c cannot be in the same register
- E.g., b and d could be in the same register

Register Allocation

Which of the following pairs of temporaries interfere in the code fragment given at right?

- ☐ A and B
- ☐ A and C
- ☐ B and C
- ☐ C and D



Register Allocation

- Extracts exactly the information needed to characterize legal register assignments
- Gives a global (i.e., over the entire flow graph) picture of the register requirements
- After RIG construction the register allocation algorithm is architecture independent