Live and Dead Code

what is dead code?

- Some instructions don't do anything (especially after other optimization has been done) and are **dead code**

```
1: A = B * C
2: A = C + X
```

First computation of A produces a value that won't be used

- Difficulty: not always obvious that an instruction is dead: property is transitive

```
1: A = B * C
2: B = C * A
3: D = A + B
4: E = D + A
5: E = 7
```

Instructions 1 through 4 are all dead, but it's hard to see that

what is dead code?

- Some instructions don't do anything (especially after other optimization has been done) and are **dead code**

```
1: A = B * C
2: A = C + X
```

First computation of A produces a value that won't be used

- Difficulty: not always obvious that an instruction is dead: property is transitive

```
1: A = B * C
2: B = C * A
3: D = A + B
4: E = D + A
5: E = 7
```

Instructions 1 through 4 are all dead, but it's hard to see that

turn it around: what is live?

- Easier to focus on the dual problem: what code is **live**
  - A variable is **live** if it has a value that *may* be used in the future
  - At any point in code, multiple variables can be live

- Question: how do you know what is going to happen in the future?
  - Answer: go backwards!

executing backwards

- A variable is live if its value may be used in the future
- At the end of a basic block, we can make a good guess about what is live
  - Temporaries are not live (they only get used during the execution of single statements, so are not used in the future)
  - Local variables and global variables *may* be used elsewhere, so they are live
  - If this block is the end of the whole program, nothing is live
- Can then propagate this information backwards

next: liveness analysis