KU | Fall 2018 | Drew Davidson

EECS 665 COMPILER CONSTRUCTION

24 – Control Flow Graphs

Administrivia

- Homework 7 out now
  - Question 1: Justify various IRs

    "ASTs are good because they're a natural target for SDT", etc.
  - Question 2: Given some source code, write out an AST, 3AC, and Control-Flow Graph representation
- Project 4 out later today
  - I'll send an email
  - Name analysis
- Project 3 grades *almost* done
  - What kind of joker wrote this lexer?

Warmup Exercise: AST to 3AC

Generate AST

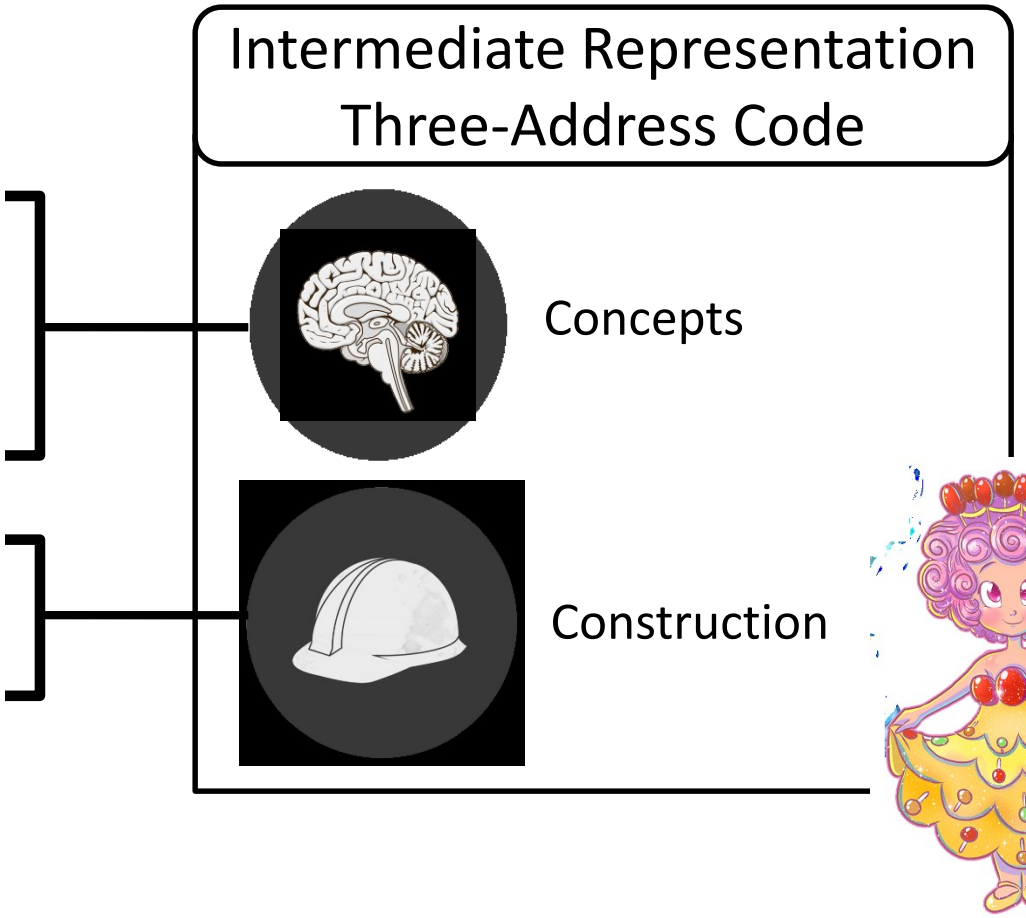
Generate 3AC

```
int main(){
    while (a < 4){
      a++;
    }
    return a;
}
```

Last Lecture

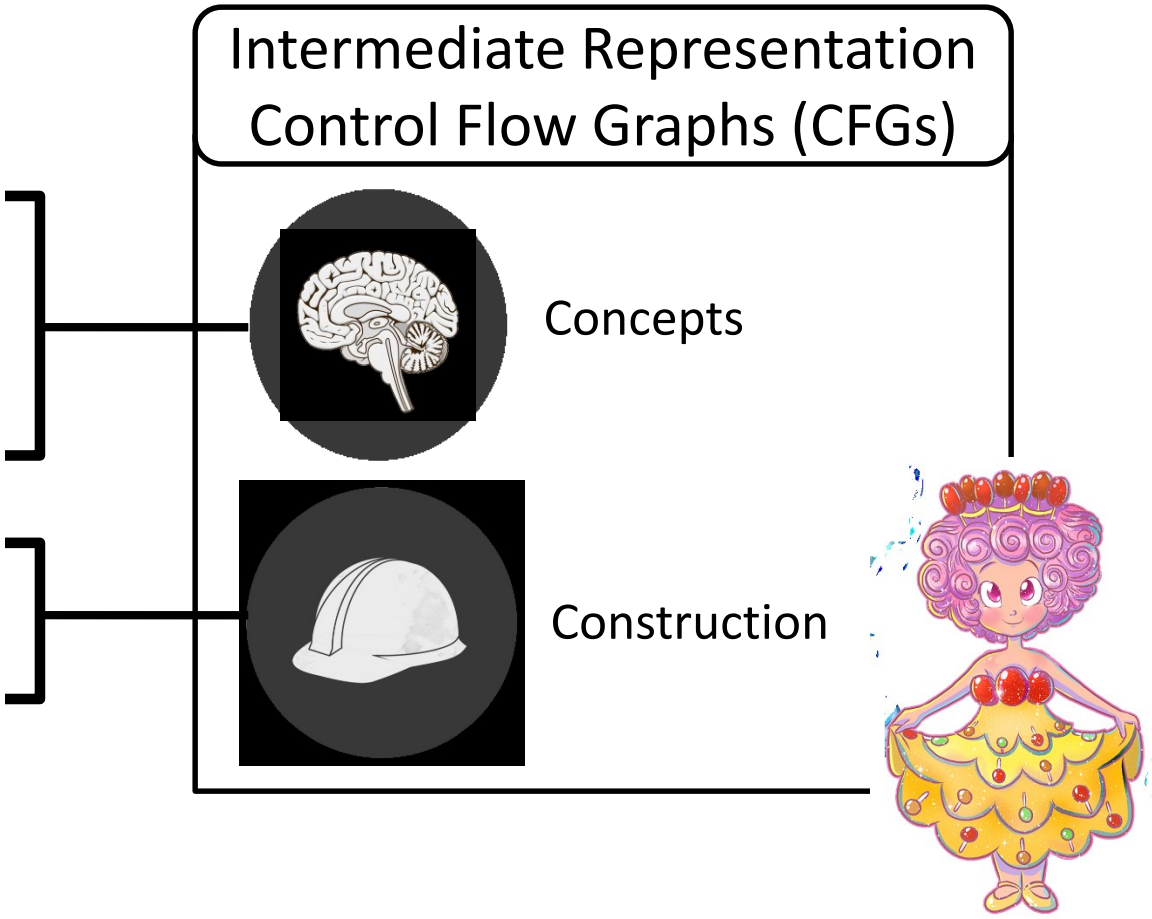
- 3AC: Definition
- 3AC: Intuition
- 3AC: Examples

- From AST to 3AC



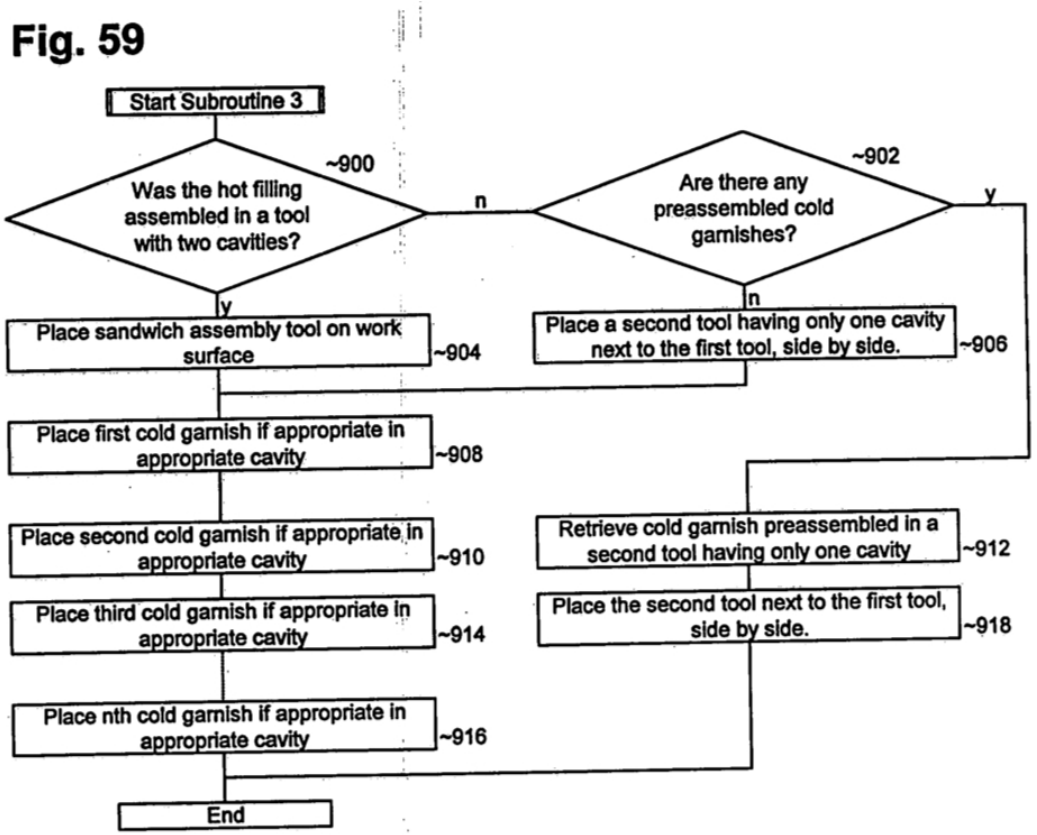
This Lecture

- Control Flow Graphs
  - Intuition
  - Examples
- From 3AC to CFG



CFG precursor: Flow Chart

- Nodes are instructions
- Edges go to successor nodes
- Operation:
  - Execute the instruction
  - Decide on the next instruction



Flow chart for building a sandwich, appearing in a McDonald's patent

Flow Graphs for Code

Like how you lay out imperative code on a whiteboard

```
void funk(int a, int b){
    print(1);
    if (a < b){
        if (b < a){
            print(3);
        }
    }
    if (b < 3){
       print(4);
    }
}
```

CFGs: A Flow Chart Refinement

Control Flow Graphs

- "Try to get by with as few nodes as possible"
- Rather than nodes being single instructions, they will be runs of instructions called *Basic Blocks* (BBLs).

```
void funk(int a){
    if (a < b){
        b = 1;
        a += b;
    } else {
        a = 7;
    }
}
```

Summary

CFGs add a “graph overlay” to the linear code

Basic Blocks: Definition

Largest run of instructions guaranteed to run in sequence

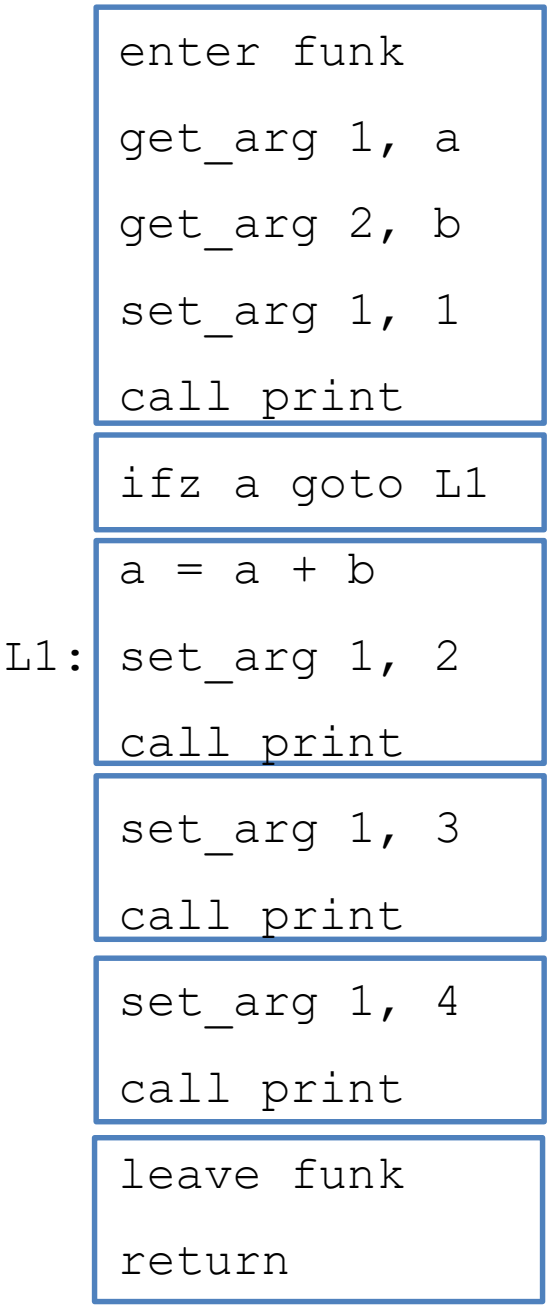
```
void funk(int a, int b){
    print(1);
    if (a != 0){
       a += b;
       print(2);
       print(3);
    }
    print(4);
}
```

```
    enter funk
    get_arg 1, a
    get_arg 2, b
    set_arg 1, 1
    call print
    ifz a goto L1
    a = a + b
    set_arg 1, 2
    call print
    set_arg 1, 3
    call print
L1: set_arg 1, 4
    call print
    leave funk
    return
```

Add nodes

Conditional Blocks

Branch instructions cause a node to have multiple out-edges

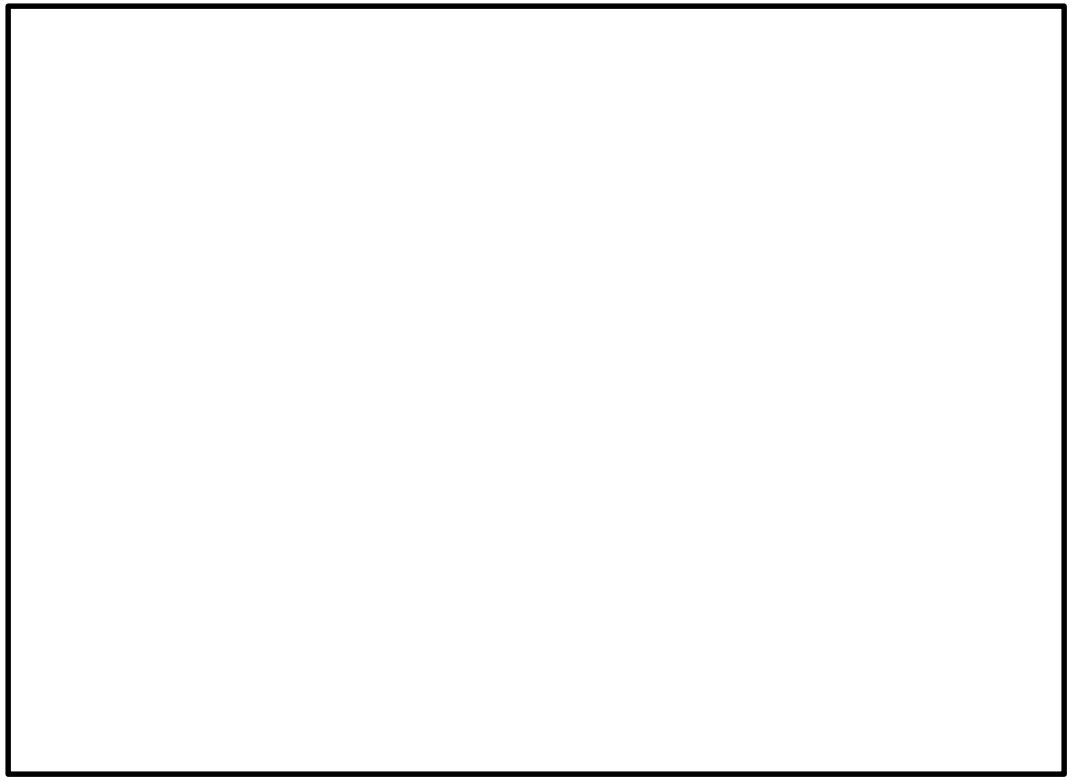


Add edges

CFG Visualization

CFGs can maintain a linear order

- Visually nice to spread it out



Block layout

BBL1
```
enter funk
get_arg 1, a
get_arg 2, b
set_arg 1, 1
call print
```

BBL2
```
ifz a goto L1
```

BBL3
```
a = a + b
set_arg 1, 2
call print
```

BBL4
```
set_arg 1, 3
call print
```

BBL5
```
set_arg 1, 4
call print
```

BBL6
```
leave funk
return
```

CFG Visualization: Patterns

Loops Ifs If/else Nested if

Let’s Talk about CFGs!

Why we (I) love ‘em, what they don’t do

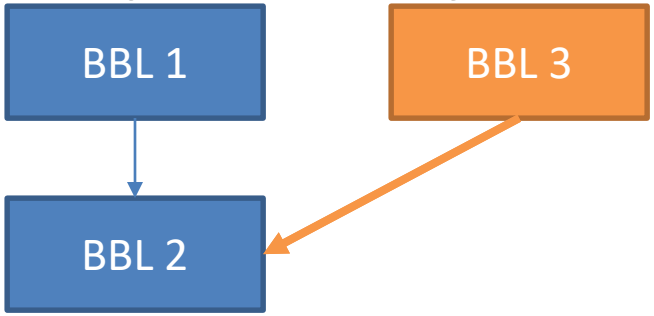
Summary

CFGs have many benefits, primary for intraprocedural control-flow

Intraprocedural Analysis

We'd really like to know if certain properties hold in code *across* blocks

- Is a used variable previously defined?



- There is a property that one block must be executed before another, but it goes by a pretty boring and forgettable name

Just kidding, it's called…


Domination

Dominator Trees

Next Lecture

- Computing *data* flow on the Control Flow Graph