EXERCISE 16

DEPENDENCE GRAPH REVIEW

Write your name and answer the following on a piece of paper

Draw the Control Dependence Graph for the following program

```c
int main(){
        i = getchar();
        if ( i == 1 ){
                printf("hi!");
        } else {
                i = 1;
        }
}
```

EXERCISE 16: SOLUTION

DEPENDENCE GRAPH REVIEW

EXERCISE 16: SOLUTION

DEPENDENCE GRAPH REVIEW

EXERCISE 16: SOLUTION

DEPENDENCE GRAPH REVIEW

EXERCISE 16: SOLUTION

DEPENDENCE GRAPH REVIEW

ADMINISTRIVIA AND ANNOUNCEMENTS

PROGRAM SLICING

EECS 677: Software Security Evaluation

Drew Davidson

LAST TIME: CONTROL DEPENDENCE

REVIEW: LAST LECTURE

FOCUS THE ANALYSIS ON WHAT WE CARE ABOUT

Control Dependence Graph (CDG)

- Shows what program statements most immediately decide which others execute

OVERVIEW

EXTENDING THE DEPENDENCE RELATION
AND SHOWING ITS USE

LECTURE OUTLINE

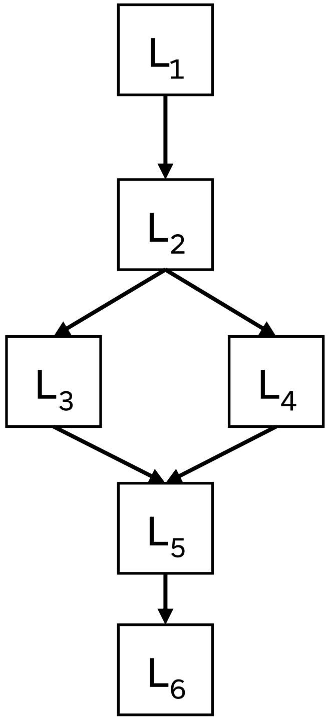
- Data Dependence
- PDGs
- Slicing

DATA DEPENDENCE

DEPENDENCE RELATIONS

Influence is more than control, it's also what values mattered to your behavior

```
1: READ i;
2: if ( i == 1)
3:    PRINT "hi!"
   else
4:    i = 1;
5: PRINT i;
6: end
```



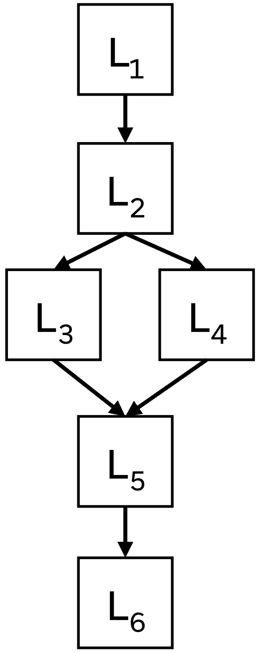
Note here: a value at L_1 might have set a value at L_5, but it's not control dependent!

THE DATA DEPENDENCE GRAPH

DEPENDENCE RELATIONS

Depiction of the *reaching definitions* of each statement

```
1: READ i;
2: if ( i == 1)
3:    PRINT "hi!"
   else
4:    i = 1;
5: PRINT i;
6: end
```

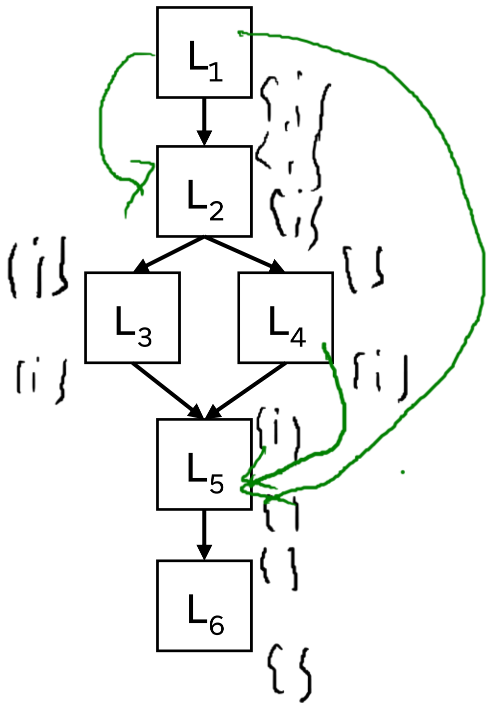


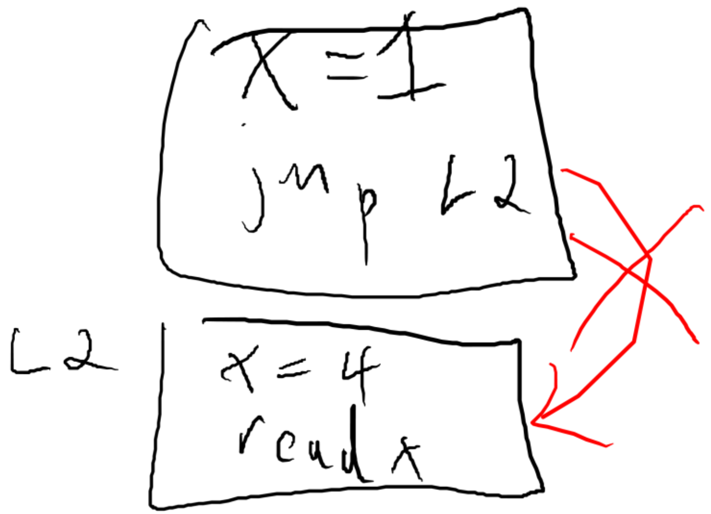
THE DATA DEPENDENCE GRAPH

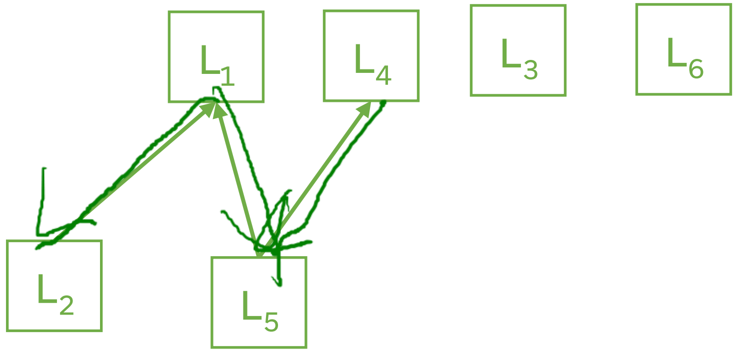
DEPENDENCE RELATIONS

Depiction of the *reaching definitions* of each statement

```
1: READ i;
2: if ( i == 1)
3:     PRINT "hi!"
   else
4:     i = 1;
5: PRINT i;
6: end
```







LECTURE OUTLINE

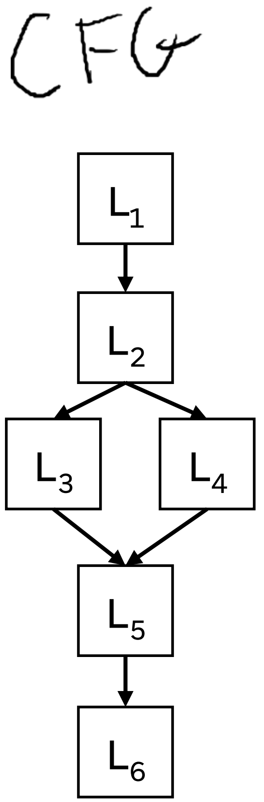
- Data Dependence
- PDGs
- Slicing

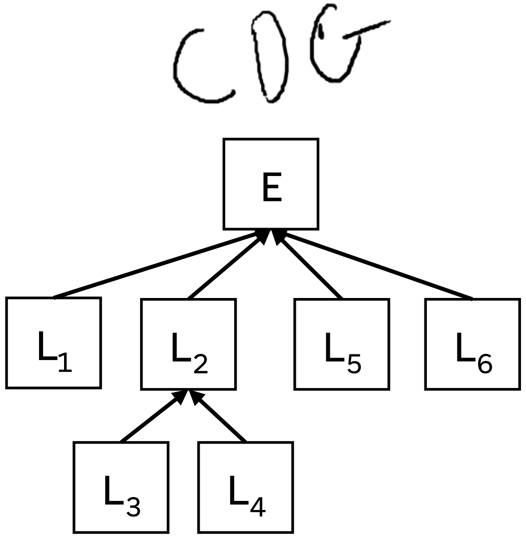
THE PROGRAM DEPENDENCE GRAPH

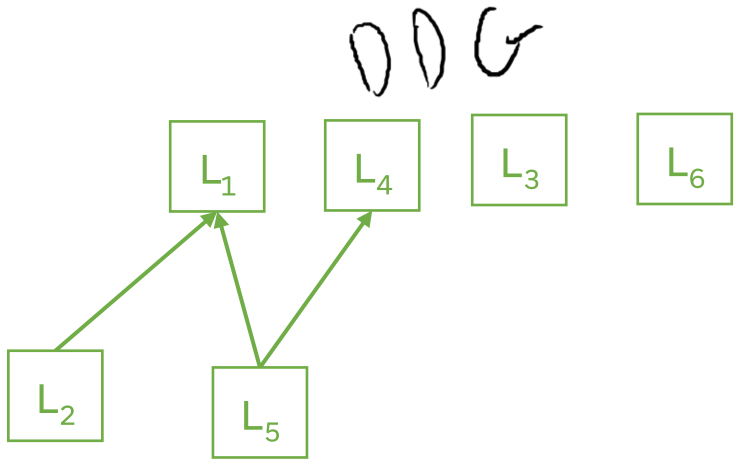
DEPENDENCE RELATIONS

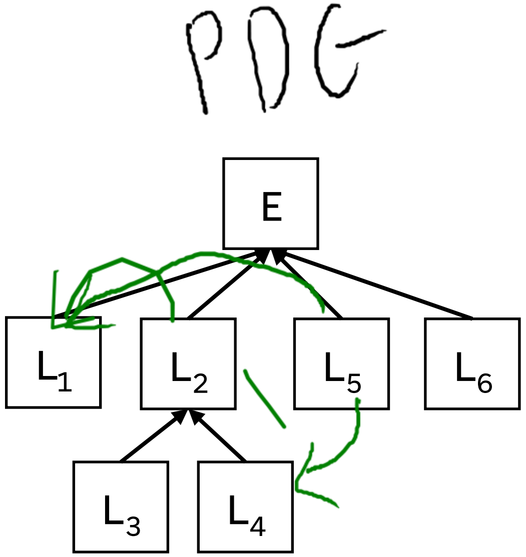
An overlay of the CDG + DDG = PDG

```
1: READ i;
2: if ( i == 1)
3:     PRINT "hi!"
   else
4:     i = 1;
5: PRINT i;
6: end
```









LECTURE OUTLINE

- Data Dependence
- PDGs
- Slicing

THE "SUB-PROGRAM" CONCEPT

PROGRAM SLICING

BIG IDEA: IGNORE "IRRELEVANT" FUNCTIONALITY FOR A PARTICULAR CASE

Control Dependence Graph (CDG)

- Shows what program statements depend on each other

Program Dependence Graph (PDG)

- At minimum: A CDG enriched with data dependence information

THE SLICE OF THE PROGRAM

PROGRAM SLICING

Forward Slice

Everything **influenced by** statement K

Forward reachability in the PDG

Program

```
x = rand()
y = rand()
x = net_read()
if (y == 1){
  printf("hello");
}
if (x > 2){
  x = 2;
}
array[x] = 4;
```

Forward

forward slice

```
x = net_read()
if (x > 2){
  x = 2;
}
array[x] = 4;
```

Backwards Slice

Everything that **influences** statement K

Backward reachability in the PDG

Backward slice

```
y = rand()
if (y == 1){
  printf("hello");
}
```

SLICE EXECUTION

PROGRAM SLICING

Do we need our sliced subprogram to be Executable?

If so, we may need to include additional instructions

OUTPUT DEPENDENCE

PROGRAM SLICING

DO WE NEED OUR SLICED SUBPROGRAM TO PERFORM IDENTICALLY TO THE ORIGINAL?

If so, we'll need additional output dependence edges

SLICING SUMMARY

PROGRAM SLICING

Static Slicing has some promising applications

It's not a one-size-fits-all scalability panacea

Any (sound) slicing is likely a benefit!

Some applications Beyond analysis

Automatic parallelization

Software metrics (how big of a change is this refactor?)

ANALYSIS TOOLS

SWITCHING GEARS

We've Covered several popular analysis techniques for imperative programming

Let's talk a bit about their tooling

LLVM: STATIC SLICING

ANALYSIS TOOLS

https://github.com/mchalupa/dg

README.md

DG

Linux CI passing macOS CI passing

DG is a library containing various bits for program analysis. However, the main motivation of this library is program slicing. The library contains implementation of a pointer analysis, data dependence analysis, control dependence analysis, and an analysis of relations between values in LLVM bitcode. All of the analyses target LLVM bitcode, but most of them are written in a generic way, so they are not dependent on LLVM in particular.

Further, DG contains an implementation of dependence graphs and a static program slicer for LLVM bitcode. Some documentation can be found in the doc/ directory.

- Downloading DG
- Compiling DG
- Using llvm-slicer
- Other tools

NEXT TIME

DEALING WITH “REAL” PROGRAMS

- POINTERS
- (AFTER THAT) CLASSES
- (AFTER THAT) INTERPROCEDURAL ANALYSIS