EXERCISE #22

DEPENDENCE GRAPH REVIEW

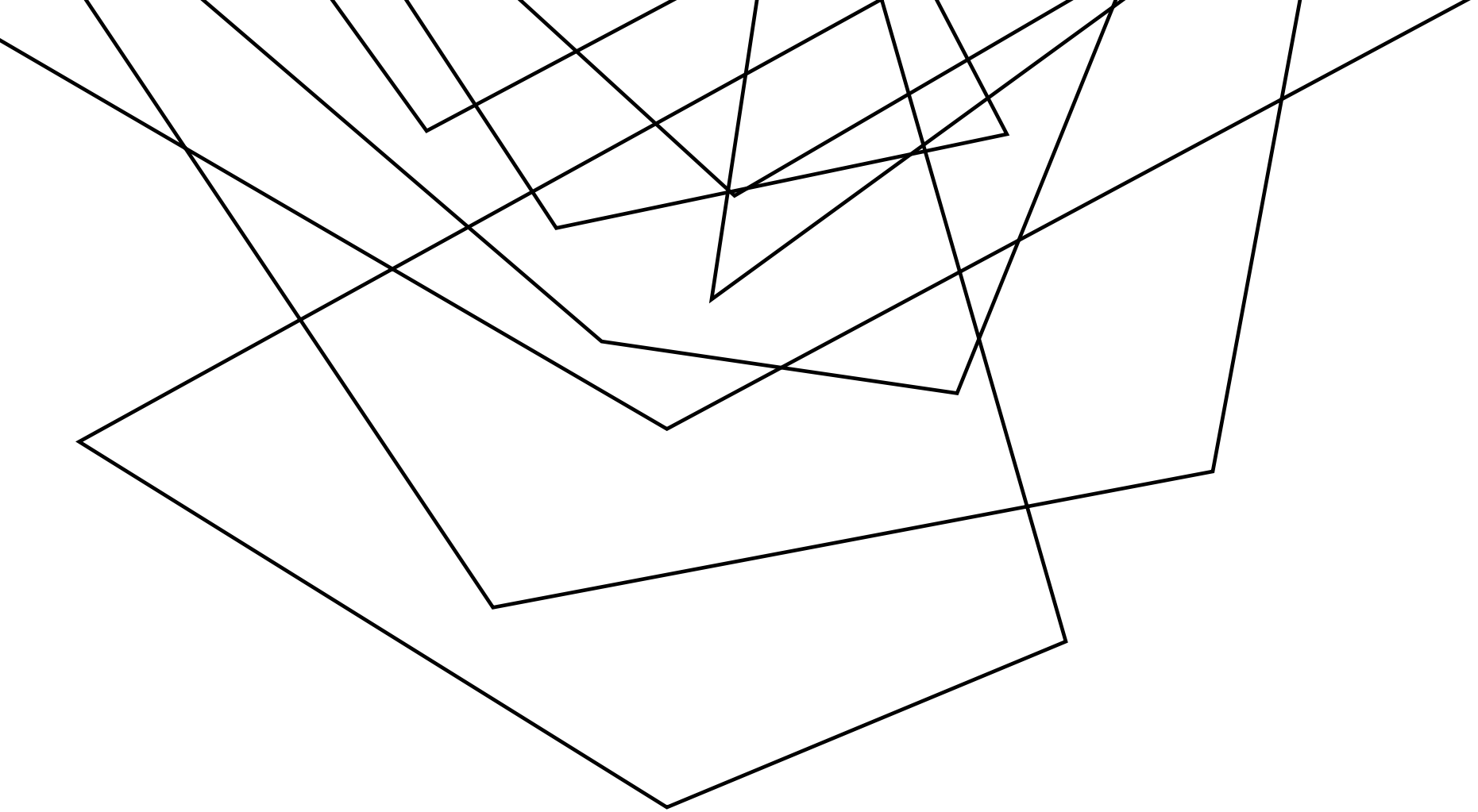
Write your name and answer the following on a piece of paper

Draw the Control Dependence Graph for the following program

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

Quiz 2 → Friday next week

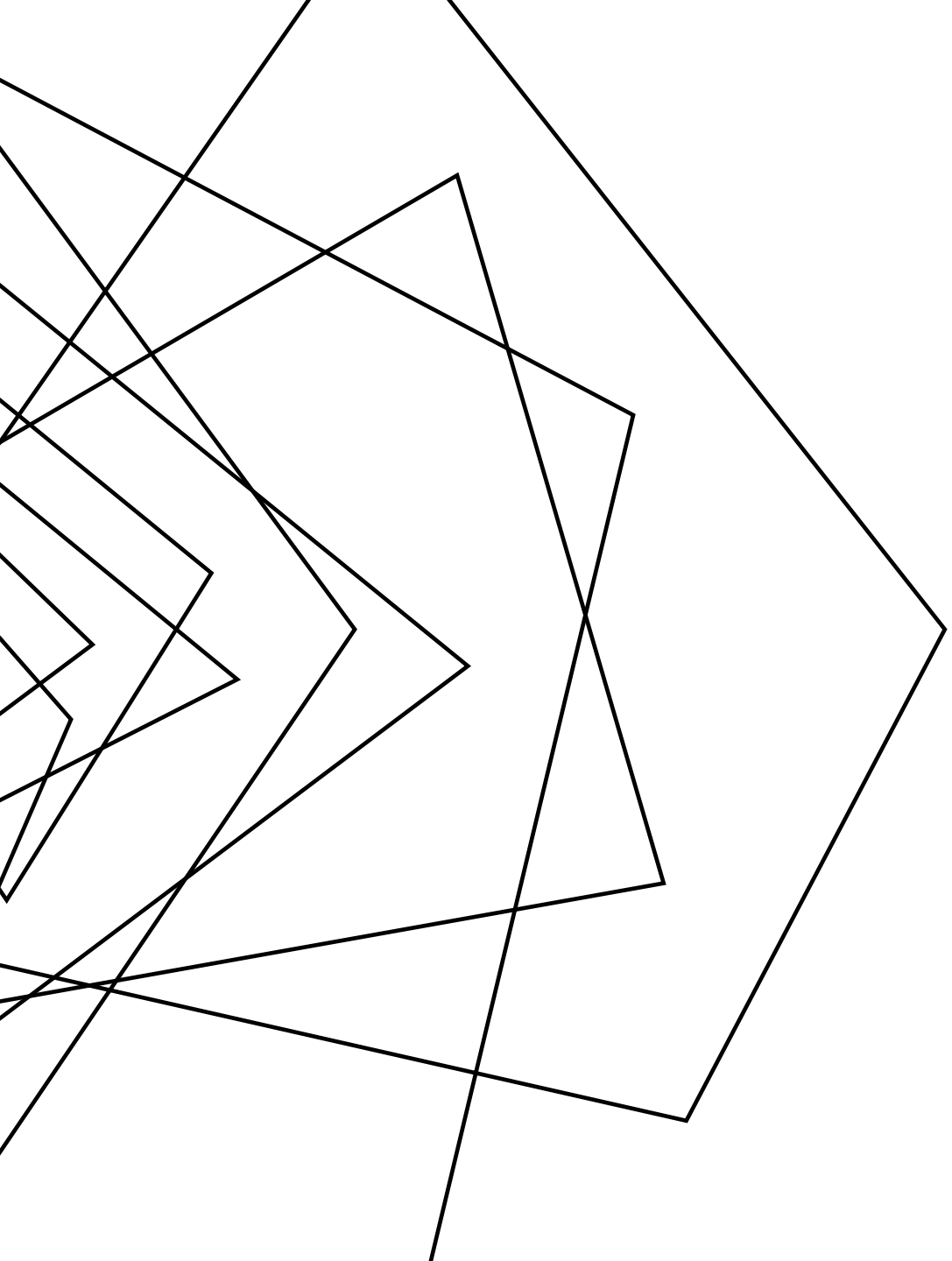
ADMINISTRIVIA AND ANNOUNCEMENTS



PROGRAM SLICING

EECS 677: Software Security Evaluation

Drew Davidson



CLASS PROGRESS

EXPLORING ANALYSES UNDERLYING OUR EVALUATION/ENFORCEMENT NEEDS

Evaluation / Enforcement: IRM, Data Leakage, CFI

Analysis tools: CFGs, Points-to graphs, Interprocedural techniques

Scaling: CDGs, PDGs

LAST TIME: DEPENDENCE GRAPHS

REVIEW: LAST LECTURE

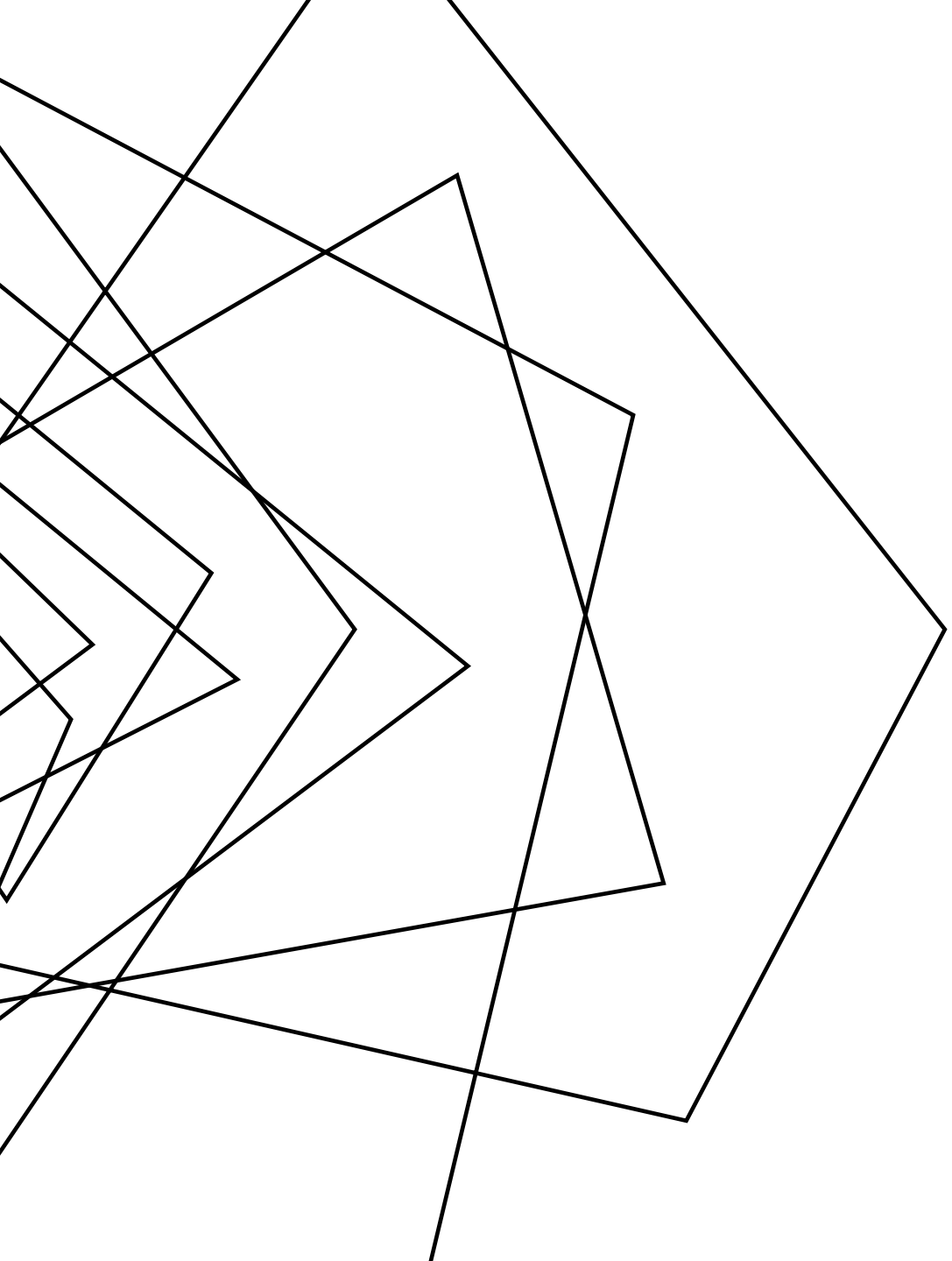
Focus the analysis on what we care about

Control Dependence Graph (CDG)

- Shows what program statements depend on each other

Program Dependence Graph (PDG)

- A CDG enriched with use/def information



OVERVIEW

WHAT WE CAN DO WITH THE PDG

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 reachability in the PDG

FORWARD SLICE

Everything **influenced by** a target statement

backward reachability in the PDG

BACKWARDS SLICE

Everything that **influences** a target statement

=

```
i = 7;
X = i;
k = 2;
y = X
```

variable defined

← data edge PDG def/use

variable used

def/use

```
x = rand();
X = NET_READ
if (y==1) {
    printf("hello");
}
array[x] = 4
```

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

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

```
X = NET-READ
if (x > 2) { x = 2; }
array[x] = 4;
```

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

More requirements of your slice, more statements / overapproximate you'll see

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: OPT

SWITCHING GEARS

The LLVM Optimization module

Can formulate analysis and program transformation tasks as “optimization passes”

```
#include "stdio.h"

int fact(int num){
  if (num <= 0){ return 1; }
  else { return fact(num-1) * num; }
}

int main(){
  int res = getchar();
  return fact(res);
}
```

clang -c -S -emit-llvm factorial.c -o factorial.ll

LLVM: OPT

SWITCHING GEARS

The LLVM Optimization module

Can formulate analysis and program transformation tasks as “optimization passes”

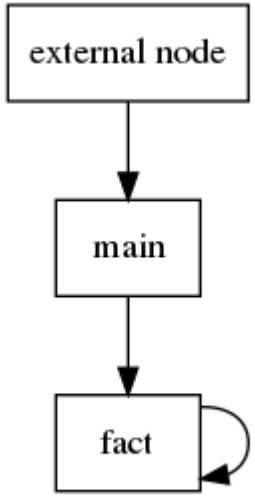
```c
#include "stdio.h"

int fact(int num){
  if (num <= 0){ return 1; }
  else { return fact(num-1) * num; }
}

int main(){
  int res = getchar();
  return fact(res);
}
```

clang -c -S -emit-llvm factorial.c -o fact.ll

OPT: CALLGRAPH BUILDING

ANALYSIS TOOLS

opt fact.ll –dot-callgraph

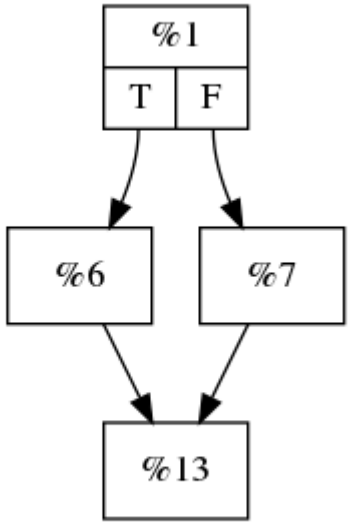


Call graph

OPT: CFG BUILDING

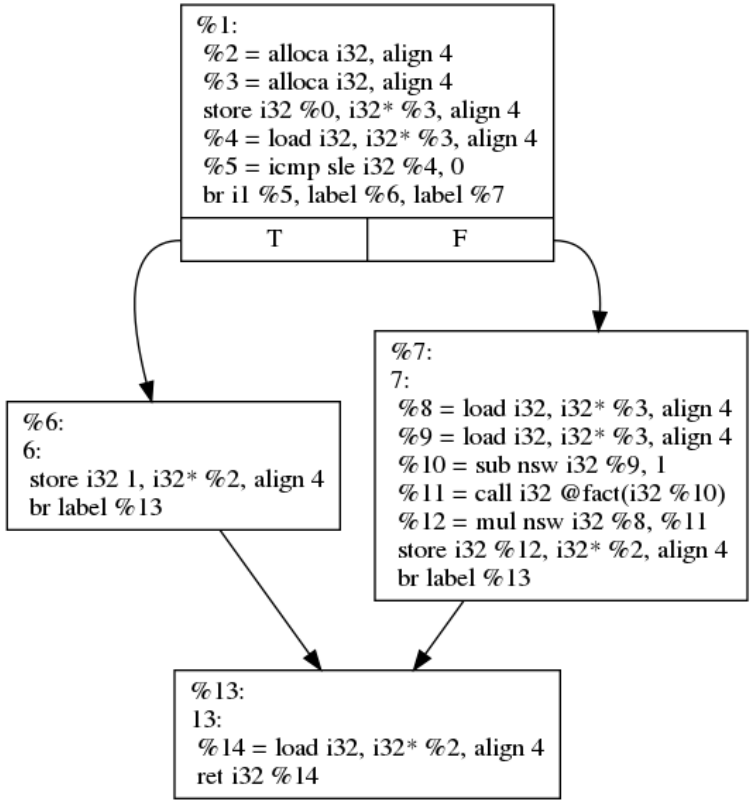
ANALYSIS TOOLS

opt fact.ll –dot-cfg-only



CFG for 'fact' function

opt fact.ll –dot-cfg



CFG for 'fact' function

OPT: 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

MODULE COMPLETE

SWITCHING GEARS