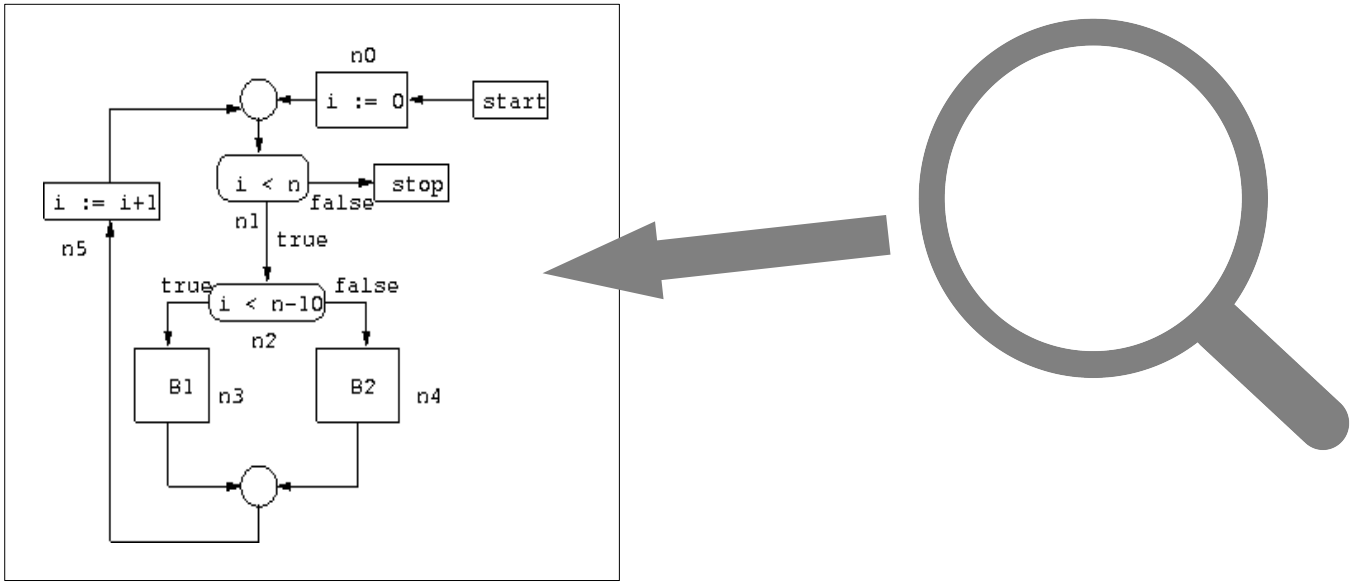
Static Program Analysis
Lecture 1: Introduction



Goals with This Module

Explain what static program analysis is, on very high level

Relation to testing

The major idea of *safety* for static analysis

Introduce the most important classes of static program analysis techniques

Give intuition for analysis methods through examples

Hands-on experience with a state-of-the-art tool

After taking this module you will not be an expert, but you should have some insights in the basic techniques and what they can be used for

Static Program Analysis

- What?
- How?
- Why?

What is Static Program Analysis?

To analyze a program without running it

Requires a mathematical model of the program

The model can then be analyzed by mathematical methods

Methods are algorithmic – automated tools are possible

The results can be trusted

Why Static Program Analysis?

Two major uses:

- Program verification (find possible bugs, or prove their absence)
- To find opportunities for compiler optimizations (the classical use)

Relation to Testing

Static analysis complements **testing**

Testing can in general not be 100% trusted (unless exhaustive)

Testing requires runnable code, thus only possible quite late in the development process

Program analysis need not run the code, can thus be performed earlier

Testing may take a long time

Program analysis potentially faster (may be compute intensive, though)

Program analysis can cut down on test cases by eliminating certain kinds of errors

Where to Apply

Static analysis is performed on code

Therefore best applied in the code development phase

Easily automated – works fine in agile development

(Some knowledge of the code may be needed to fine-tune the analysis)

Not so suitable for integration testing