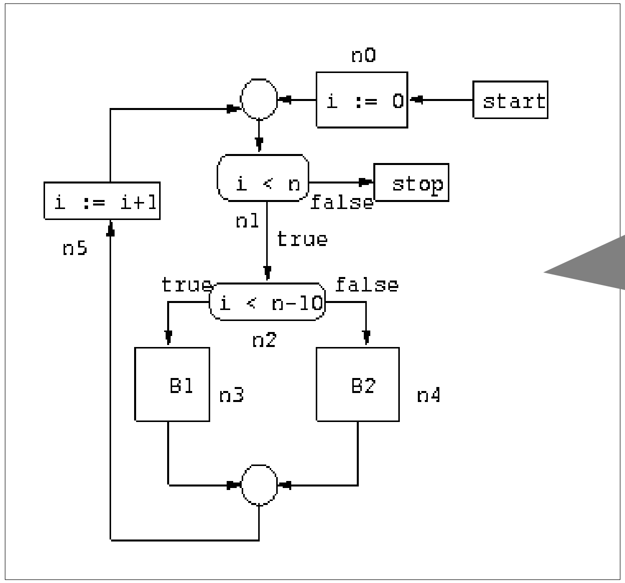
Static Program Analysis
Lecture 5: Basics of Value Analysis



Value Analysis

To find out what values that different program variables may hold in different program points

Example:

```
/* x can be anything here */
x = 0;
/* x is in the interval [0 .. 0] here */
for i = 1 to 10 do
  /* x is in the interval [0 .. 18] here */
  x = x + 2;
  /* x is in the interval [2 .. 20] here */
```

Usages of Value Analysis

Knowledge of program variable values can be used to find a number of potential bugs:

- Division by zero
- Under/overflows
- Array accesses out of bounds
- Etc . . .

A tool that performs value analysis can **warn** for these kinds of bugs

A **safe** value analysis can sometimes **prove** that such bugs never can occur!

Very interesting for high-integrity systems, since hard to verify through testing alone

Concrete States

A program that is executing will transit between *concrete states*

These states consist of a *current program location* and a *memory contents*

The memory contents is a *table*, which tells for each memory location what the contents is

In a high-level language (like C), memory locations are program variables

Some concrete states for a program with (integer) variables `x` and `y`:

x	17
y	3

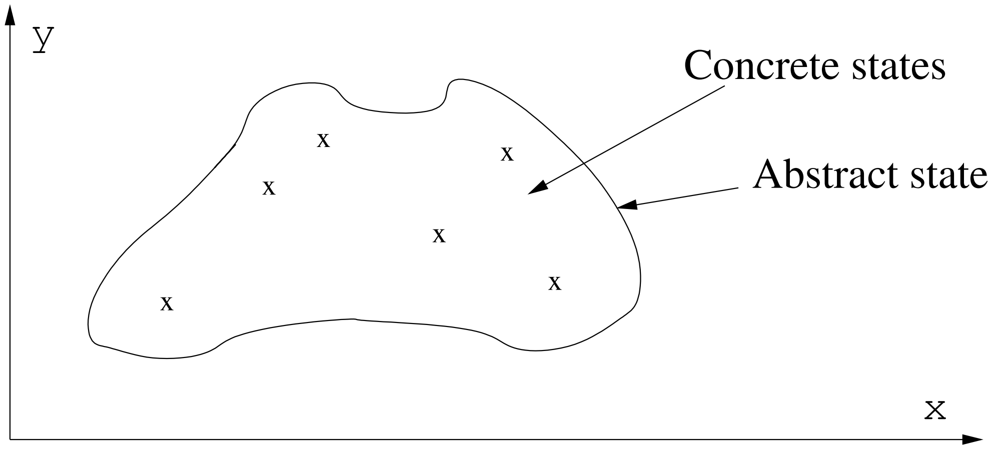
x	0
y	-7

x	4711
y	1

x	32767
y	-32768

Abstract States

Abstract states represent *sets of concrete program states*:



A value analysis computes abstract states for the different program points in a program. This tells which values program variables may hold there

A **safe** value analysis computes abstract states that **always contain** all the possible concrete states in a given program point

Interval Analysis

An *interval* $[a, b]$ is the set of all numbers between a and b:

Intervals are efficiently represented by two numbers, regardless of size

Therefore *interval analysis*, with abstract states mapping variables to intervals, is common. This is a fast but somewhat imprecise analysis

Some abstract states with intervals for a program with variables `x` and `y`:

x	$[0, 1]$
y	$[-5, 5]$

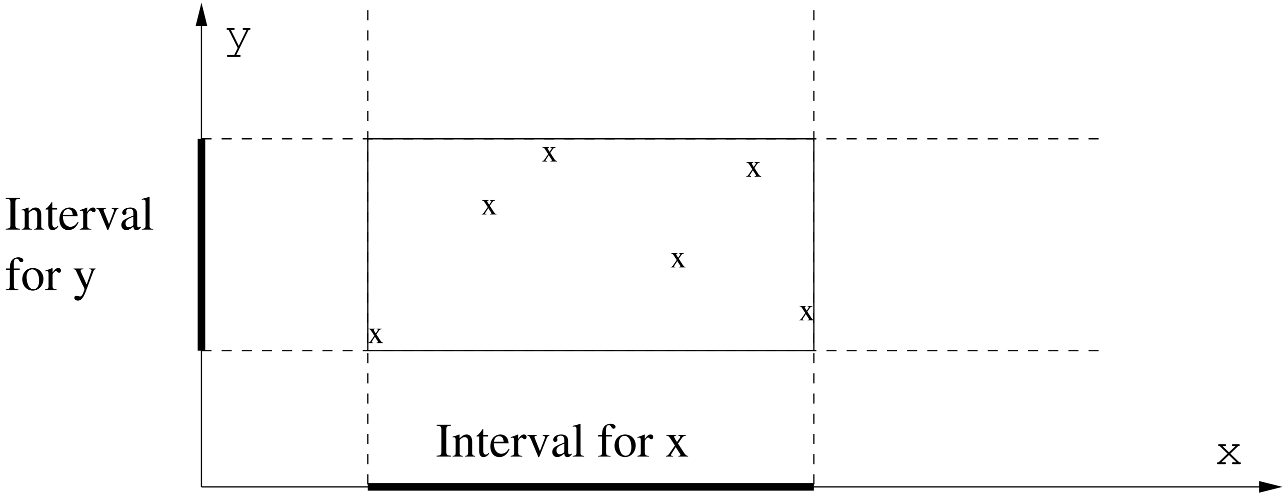
x	$[1, 1]$
y	$[0, 32767]$

x	$[-\infty, \infty]$
y	$[5, \infty]$

x	$\emptyset$
y	$\emptyset$

Abstract States with Intervals

Intervals yield abstract states that are “bounding boxes”:



Abstract states with intervals form the “Interval Domain”

How to Perform Value Analysis

Value analysis is done by solving equations

Very similar to data flow analysis

The equations relate abstract states rather than sets: one abstract state per program point (edge in the CFG)

The equations are formed from transfer functions for CFG nodes

Same method for solving the system of equations – fixed-point iteration starting with the “least” abstract state

We will show how it works for the Interval Domain in the next lecture

Safety of Value Analysis

Underlying mathematical framework of *Abstract Interpretation*

Relates sets of concrete states with abstract states

If certain conditions are fulfilled, then:

- Fixed-point iteration from the “least” abstract state is **safe**, and
- it will yield the **best** (smallest) solution to the system of equations

Having a formal mathematical framework increases the confidence in the analysis a lot. An analysis tool basically performs a mathematical proof when analysing code