A Look At Checking Program Correctness

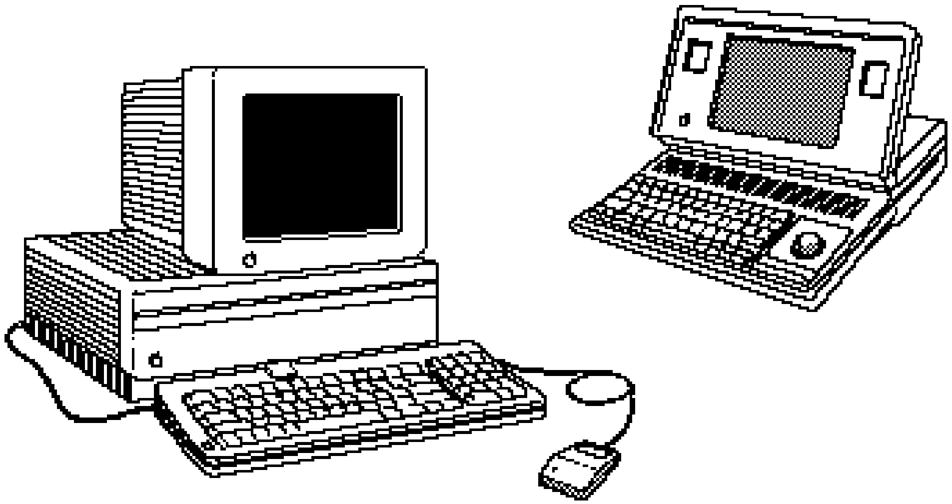
Marsha Chechik

University of Toronto

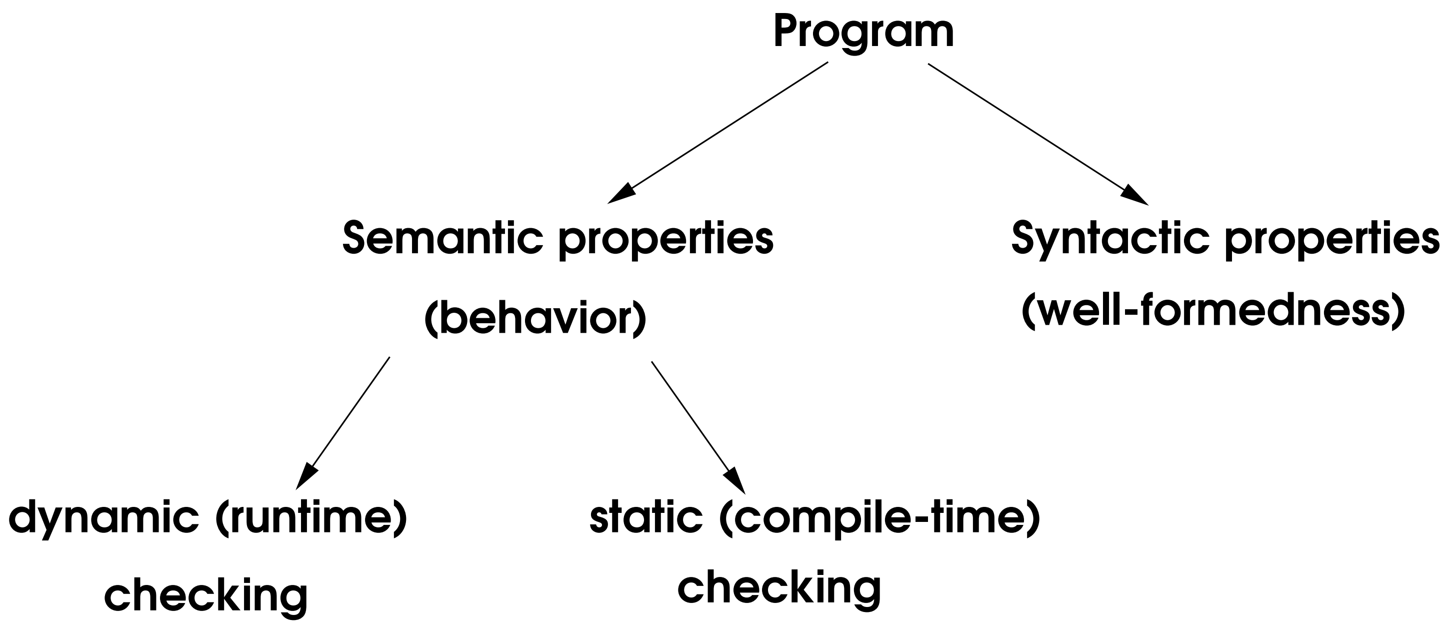
May 23, 1998

Goal: Automatic Checking

- Write a program that will check that another program is correct



Properties of Programs



Runtime (Dynamic) Analysis

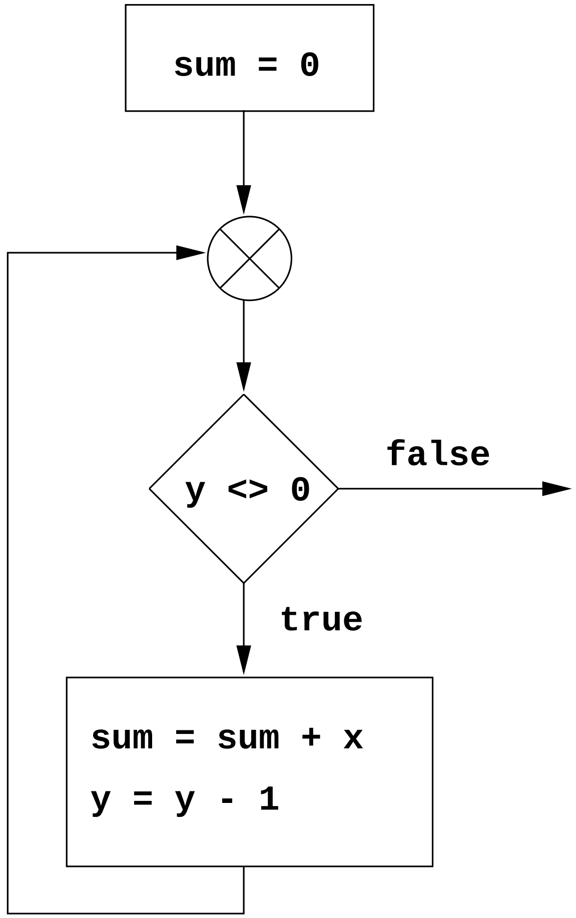
- execute the program one path at a time
- generate all possible inputs and check that outputs are correct

Example: "Does the following program implement $x \times y$?"

```
sum = 0
while (y <> 0) do
  sum = sum + x
  y   = y-1
od
```

Suggested input: x = 5, y = 2.

Example



Problems with Automated Testing

- For a program with infinite-valued variables (e.g. integers) we can have an infinite number of inputs
- For a given input, the program under test might not terminate (Halting Problem)

Maybe correctness is too hard and we can analyze other properties:

- reachability
- termination
- variable ranges

Rice's theorem: "Every non-trivial property of programs is undecidable".

What About Compilers?????

- They are programs
- They analyze other programs for properties like
  - dead code (reachability)
  - array bounds
  - uninitialized variables
  - unused variables

How and why does it work?

Solution

Analyze ("run") the program replacing the *actual* values of variables with easier, *abstract* values. This is called *abstract interpretation*.

How to choose a proper abstraction?

- The abstraction should be valid

  what you learn about the abstract program should tell you something about the actual one

- The abstraction should be "easy"

  guarantee that analysis always terminates

Goal 1: Valid Abstraction

Define the effect of every language construct on abstract values in a way that results of abstract analysis can be used to reason about the actual program.

Example: Instead of a value of a variable X, use *sign* of X, or zero:

-124 $\Rightarrow -$

124 $\Rightarrow +$

0 $\Rightarrow$ 0

Now, we can execute programs on abstract values:

-1515 $\times$ 17 = -(+) $\times$ (+) = (-) $\times$ (+) = (-)

-1515 + 17 = -(+) + (+) = (-) + (+) = $\left(\begin{smallmatrix}0\\-\\+\end{smallmatrix}\right)$

Operations on Abstract Values

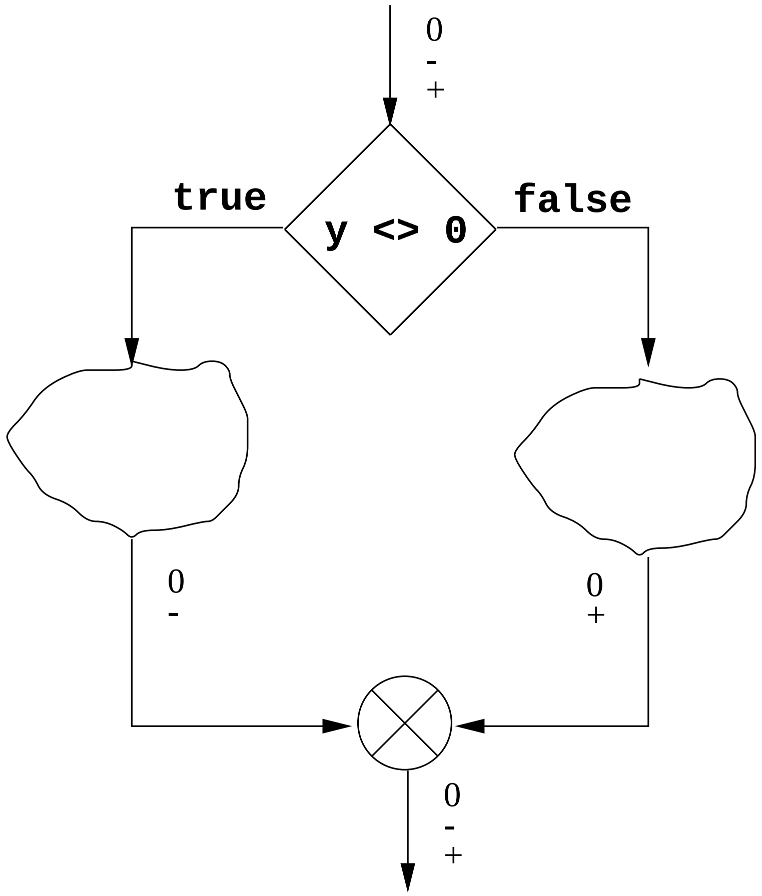
$a + b$

	0 − +	0 −	0 +	+ −	0	+	−
0 − +	0 − +	0 − +	0 − +	0 − +	0 − +	0 − +	0 − +
0 −	0 − +	0 −	0 − +	0 − +	0 −	0 − +	−
0 +	0 − +	0 − +	0 +	0 − +	0 +	+	0 − +
+ −	0 − +	0 − +	0 − +	0 − +	+ −	0 − +	0 − +
0	0 − +	0 −	0 +	+ −	0	+	−
+	0 − +	0 − +	+	0 − +	+	+	0 − +
−	0 − +	−	0 − +	0 − +	−	0 − +	−

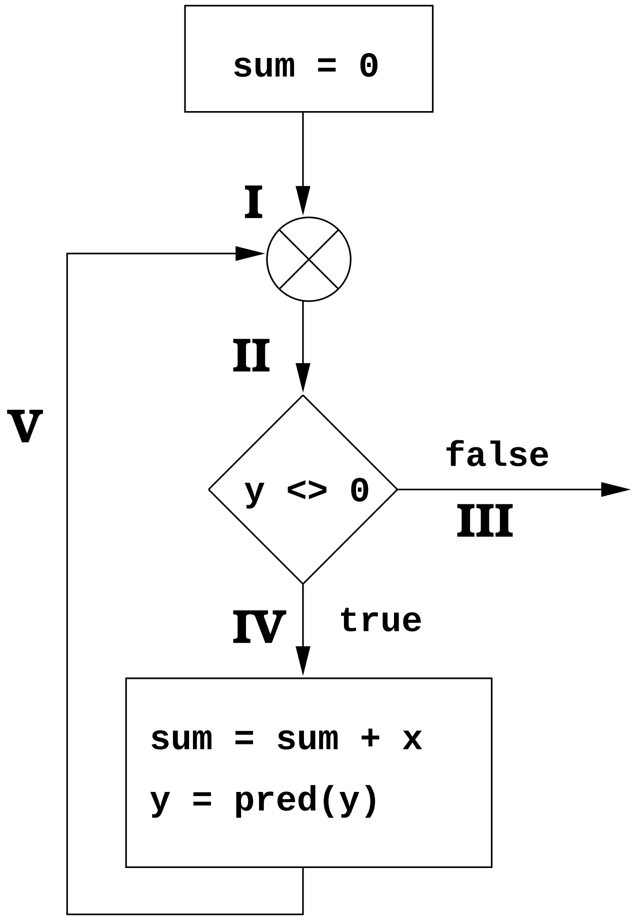
Pred(a)

0 − +	0 − +
0 −	−
0 +	0 − +
+ −	0 − +
0	−
+	0 +
−	−

Dealing with Program Control Structures



Example Revisited



Analyze with positive x and positive y

1st Pass:

I: sum = 0, x = +, y = +
II: sum = 0, x = +, y = +
III: <cannot happen>
IV: sum = 0, x = +, y = +
V: sum = +, x = +, y = $\substack{0 \\ +}$

2nd Pass

I: sum = 0, x = +, y = +
II: sum = +, x = +, y = $\substack{0 \\ +}$
III: sum = +, x = +, y = 0
IV: sum = +, x = +, y = +
V: sum = +, x = +, y = +

3rd Pass:

I: sum = 0, x = +, y = +
II: sum = +, x = +, y = $\substack{0 \\ +}$
III: sum = +, x = +, y = 0
IV: sum = +, x = +, y = +
V: sum = +, x = +, y = +

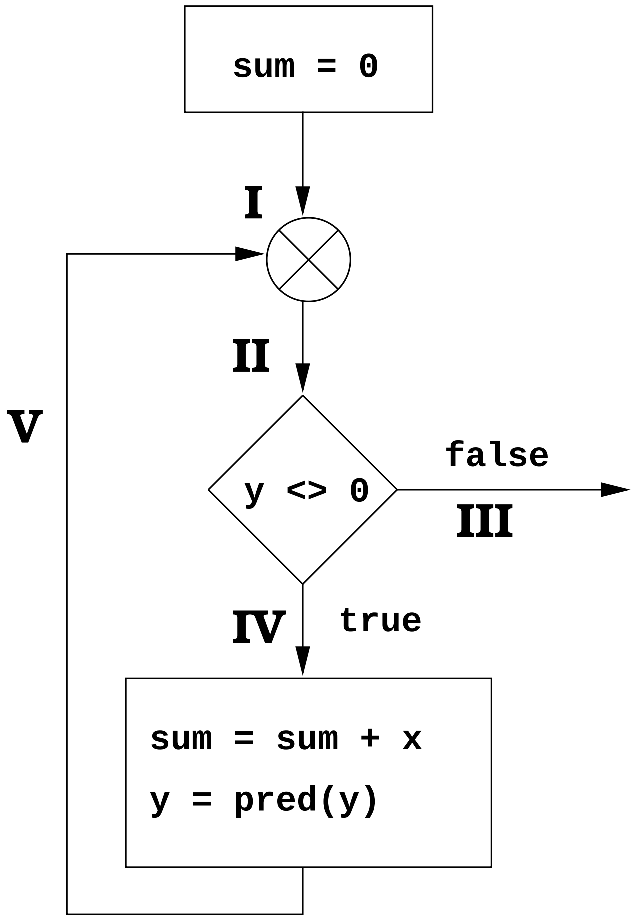
Example (Cont'd)

- Running the same analysis with $\texttt{x} = -$, $\texttt{y} = +$ yields $\texttt{sum} = -$.

What Does This Tell Us?

- As expected, sign($\texttt{sum}$) = sign($\texttt{x}$) $\times$ sign($\texttt{y}$)
- It is possible to take path `III` leading out of the loop.

Now try with negative `y`



1st Pass:

I: sum = 0, x = +, y = -
II: sum = 0, x = +, y = -
III: <cannot happen>
IV: sum = 0, x = +, y = -
V: sum = +, x = +, y = -

2nd Pass

I: sum = 0, x = +, y = -
II: sum = 0, x = +, y = -
III: <cannot happen>
IV: sum = 0, x = +, y = -
V: sum = +, x = +, y = -

Thus, path III out of the loop cannot be taken and

the loop is infinite!

Goal 2: Termination

How can we achieve termination?

Observation: if at every point every variable has finite number of values ("possibilities") associated with it, these possibilities can be tested exhaustively

Handling loops:

- "Normal" program may go through a loop infinite number of times
- Abstract program does the same thing but keeps track of which values it has seen already and would not repeat the computation when it is not necessary

Thus, abstract program has to terminate!

Conclusion

- Abstract interpretation is a general technique which substitutes execution on real values by an execution on abstract values
- The abstraction should guarantee validity and termination
- Most program analysis techniques can be formulated as abstract interpretations.

  For example:

  - various versions of data-flow analysis
    - definition-use
    - available expressions
    - dead code
  - equivalence partitioning
  - etc.