COMP0174 Practical Program Analysis
Data Flow Analysis

Sergey Mechtaev
s.mechtaev@ucl.ac.uk

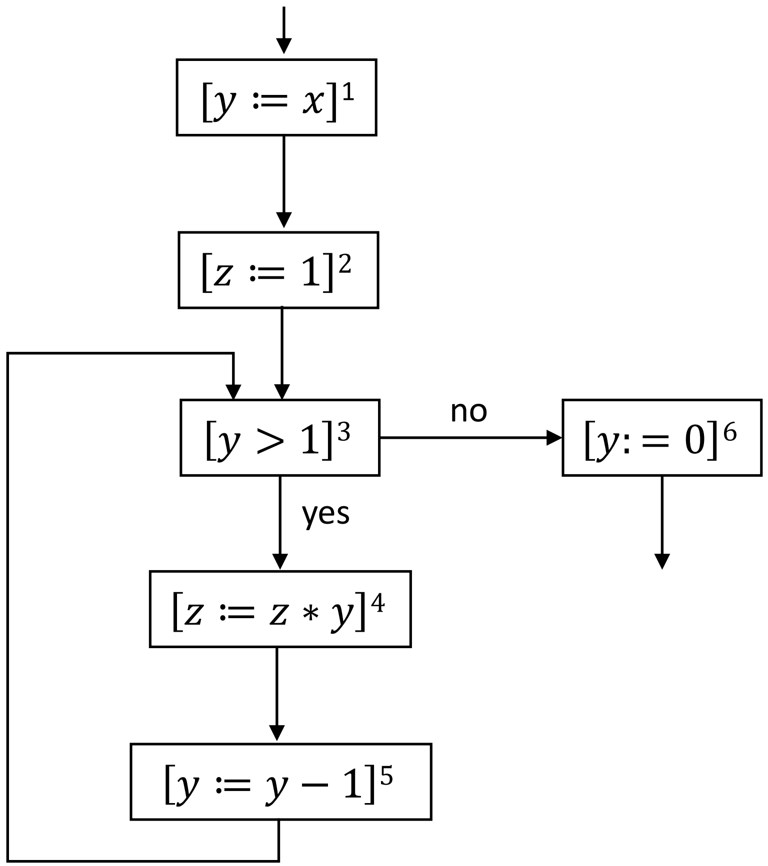
UCL Computer Science

Data Flow Analysis

- **Data flow analysis** is a technique for gathering information about the possible set of values calculated at various points in a computer program.
- The program represented using **control-flow graph** (CFG)
- The information inferred from each node of CFG is described using **lattice**.

Control Flow Graph

$[y := x]^1;$
$[z\ := 1]^2;$
$while\ [y > 1]^3\ do$
$\quad([z\ := z * y]^4;$
$\quad\ [y\ := y - 1]^5);$
$[y := 0]^6$



NODES = elementary blocks

EDGES = describe how control might pass from one elementary block to another

Initial Labels

The *init* function returns the initial label of a statement:

$$init\left([x := a]^l\right) = l$$
$$init\left([skip]^l\right) = l$$
$$init(S_1; S_2) = init(S_1)$$
$$init\left(if\ [b]^l\ then\ S_1\ else\ S_2\right) = l$$
$$init\left(while\ [b]^l\ do\ S\right) = l$$

Final Labels

The $final$ function returns the set of final labels of a statement:

$$final([x := a]^l) = \{l\}$$
$$final([skip]^l) = \{l\}$$
$$final(S_1; S_2) = final(S_2)$$
$$final(if\ [b]^l\ then\ S_1\ else\ S_2) = final(S_1) \cup final(S_2)$$
$$final(while\ [b]^l\ do\ S) = \{l\}$$

Blocks

The $blocks$ function collects the elementary blocks associated with a statements:

$$blocks\left([x \coloneqq a]^l\right) = \{[x \coloneqq a]^l\}$$
$$blocks\left([skip]^l\right) = \{[skip]^l\}$$
$$blocks(S_1; S_2) = blocks(S_1) \cup blocks(S_2)$$
$$blocks\left(if\ [b]^l\ then\ S_1\ else\ S_2\right) = \{[b]^l\} \cup blocks(S_1) \cup blocks(S_2)$$
$$blocks\left(while\ [b]^l\ do\ S\right) = \{[b]^l\} \cup blocks(S)$$

Flow

The $flow$ function extracts edges of the flow graph as pairs (l, l') :

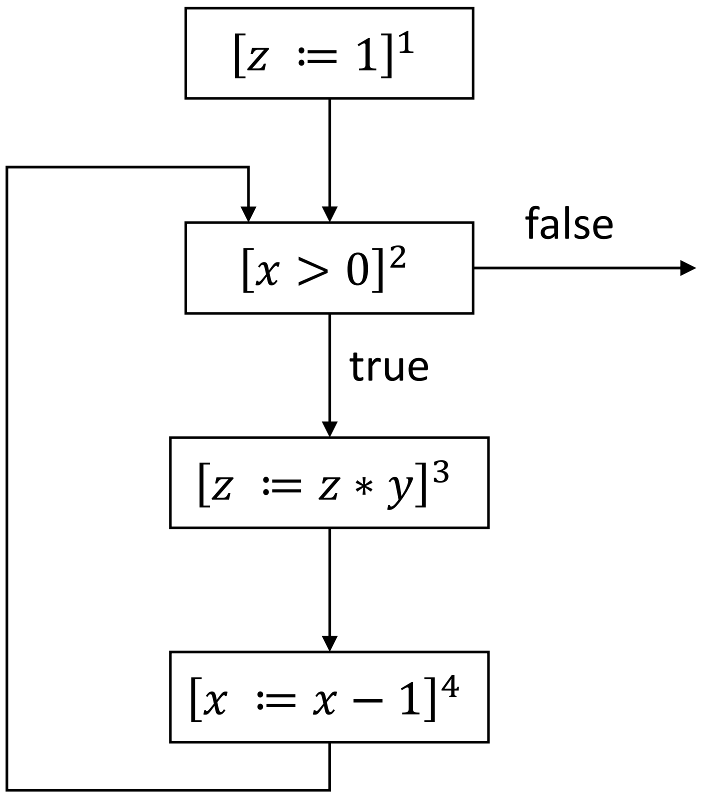
$$flow([x := a]^l) = \emptyset$$
$$flow([skip]^l) = \emptyset$$
$$flow(S_1; S_2) = flow(S_1) \cup flow(S_2) \cup \{(l, init(S_2)) \mid l \in final(S_1)\}$$
$$init(if\ [b]^l\ then\ S_1\ else\ S_2) = flow(S_1) \cup flow(S_2)$$
$$\cup \{(l, init(S_1)), (l, init(S_2))\}$$
$$init(while\ [b]^l\ do\ S) = flow(S) \cup \{(l, init(S))$$
$$\cup \{(l', l) \mid l' \in final(S)\}$$

Reverse Flow

Edges of the flow graph for backward analysis:

$$flow^R(S) = \{(l', l) \mid (l, l') \in flow(S)\}$$

Example (The Power Program)



$$init(Power) = 1$$
$$final(Power) = \{2\}$$
$$labels(Power) = \{1,2,3,4\}$$
$$flow(Power)$$
$$= \{(1,2), (2,3), (3,4), (4,2)\}$$

Classification of Analyses

- Intraprocedural vs interprocedural
- Flow-sensitive vs flow-insensitive
- Context-insensitive vs context-sensitive
- Must vs may
- Forward vs backward

Intraprocedural vs Interprocedural

- **Intraprocedural analysis** is a mechanism for performing analysis for each function, using only the information available for that.
- **Interprocedural analysis** is a mechanism for performing analysis across function boundaries.

Flow-sensitive vs flow-insensitive

- **Flow-sensitive** analyses is an analysis whose results depends on the order of statements (requires a model of program state at each program point).
- **Flow-insensitive** is an analysis whose result is the same regardless of the statement order (requires only a single global state).

Context-insensitive vs context-sensitive

- A **context-insensitive** analysis is an interprocedural analysis that cannot distinguish between different calls of a procedure (the analysis information is combined for all call sites)
- A **context-sensitive** analysis is an interprocedural analysis that takes the context of procedure calls into account (more precise, but also more costly).

Must vs may

- **Must analysis** detect properties that are satisfied by all paths of execution.
- **May analysis** detect properties that are satisfied by at least one execution path.

Forward vs Backward

- **Forward analysis** propagates information from the beginning to the end of the program
- **Backward analysis** propagates information from the end to the beginning of the program.

Four Classic Analyses

	Forward	**Backward**
Must	Available Expressions	Very Busy Expressions
May	Reaching Definitions	Live Variables