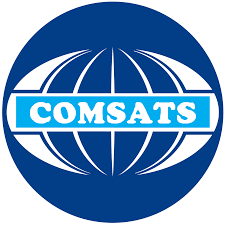
**Comsats University**



**Wah Campus**

**Software Testing**

Submitted by: Abdullah Sajid.

Reg#CIIT/FA20-BSE-030/WAH.

Subject: Software Testing.

Submitted to: Ma’am Mehwish Mukhtar.

Date: 29-10-2023

Two complementary methods for examining the behaviour of computer program are data and **control flow analysis**. While control flow analysis follows the flow of control through a program, data flow analysis follows the flow of data through a program. Compiler optimisations including register allocation, dead code removal, and loop optimisation use both approaches.

A **control flow graph** (CFG), a directed graph that shows the potential program execution paths, is usually used for data flow analysis. A program statement is represented by each node in the CFG, and a potential control flow from one statement to another is represented by each edge.

The compiler creates a data flow structure for each variable in the program before performing **data flow analysis**. A partially ordered set that illustrates the potential values a variable may have at a specific program point is called a data flow structure. For instance, the three elements true, false, and unknown could make up the data flow lattice for a Boolean variable.

On a CFG, **control flow analysis** is also commonly carried out. Control flow analysis aims to identify the range of potential program execution routes. Numerous uses for this data exist, including program verification and compiler optimisation.

Reaching Definitions Analysis is a popular algorithm for control flow analysis. The set of definitions for a variable that may be reached at a specific program point is ascertained through reaching definitions analysis. Code that is never executed, or dead code, can be removed with the help of this information.

**Here is an example:**

**def** Test(x):

**if** x > 0:

y = 2

**else**:

y = 3

**return** x + y

**print**(Test (4))

In order to conduct data flow analysis on this program, the compiler would create a **CFG** for the function **Test**() first. The **if** statement, the function's exit point, and the function's entry point would each have their own node in the CFG.

Every variable in the program would then have a data flow structure created by the compiler. For instance, the three elements **unknown**, **2**, and **3** would make up the data flow lattice for the variable **y**.

Subsequently, the compiler would calculate the data flow lattice values at every CFG node. For instance, since the **if** statement ensures that **y** will be assigned the value **2** before the function returns, the value of the data flow lattice for the variable y at the function's exit point would be **2**.

Lastly, the data flow information would be used by the compiler to optimise the program. For instance, the compiler might remove the dead code assignment to **y** from the **if** statement's else branch.