SAVITRIBAI PHULE PUNE UNIVERSITY

A PRELIMINARY PROJECT REPORT ON

PRECISION EVALUATION OF POINTER ANALYSIS VARIANTS

SUBMITTED TOWARDS THE
PARTIAL FULFILLMENT OF THE REQUIREMENTS OF

BACHELOR OF ENGINEERING (Computer Engineering)

BY

Ananya Roy	Exam No: B120204354
Jui Shinde	Exam No: B120204377
Mugdha Khedkar	Exam No: B120204315

Under The Guidance of

Prof. Chhaya Gosavi

Prof. Supratim Biswas

DEPARTMENT OF COMPUTER ENGINEERING
CUMMINS COLLEGE OF ENGINEERING FOR WOMEN
KARVENAGAR, PUNE - 411052

CUMMINS COLLEGE OF ENGINEERING FOR WOMEN
DEPARTMENT OF COMPUTER ENGINEERING

CERTIFICATE

This is to certify that the Project Entitled

PRECISION EVALUATION OF POINTER ANALYSIS VARIANTS

Submitted by

Ananya Roy	Exam No: B120204354
Jui Shinde	Exam No: B120204377
Mugdha Khedkar	Exam No: B120204315

is a bonafide work carried out by Students under the supervision of Prof. Chhaya Gosavi and it is submitted towards the partial fulfillment of the requirement of Bachelor of Engineering (Computer Engineering) Project.

Prof. Chhaya Gosavi
Internal Examiner
Dept. of Computer Engg.

External Examiner

Prof. Supriya Kelkar
H.O.D
Dept. of Computer Engg.

Abstract

Pointer analysis is a static code analysis technique that establishes which pointers can point to which variables, or storage locations [1]. It helps uncover indirect accesses thereby providing useful information about data manipulation by the programs. This improves the precision of program analyses and transformations that have to deal with programs containing pointers. It is not only useful for making the programs more efficient, but can also be used for increasing the reliability of IT based systems deployed in medical communities, banking system, defense industry etc. During the past thirty-five years, hundreds of papers and many Ph.D. theses have been published on pointer analysis. New pointer analysis algorithms are being developed and compared in order to analyze programs with optimized compiler features. The ongoing research in this field predicts the evolution of faster program analyses.

IT systems need to satisfy two basic requirements : Efficiency and Precision. It is important to strike a balance between the two. There are various pointer analysis algorithms but none of them provide satisfactory efficiency and precision together. The algorithms compromising on precision are quicker and easier to implement. Thus, they are the clear choice of programmers. But they lose out on precision. The primary motivation of the project is to compare all four pointer analysis variants and also to find out whether Context-Sensitive pointer analysis leads to some additional precision if combined with the already efficient (but comparatively less precise) Flow-Insensitive pointer analysis.

The input to these pointer analysis programs will be a set of pointer statements (of the form x = &y, x = y etc). The output will be in the form of a Points-to graph. The Points-to graphs generated by all the pointer analysis variants can be used in order to study their precision. Efficiency can be calculated by calculating the execution time of these programs. Measurements and testing of these programs is of prime importance since it can lead to many insightful results. If Context-Sensitivity when combined with Flow-Insensitivity does not give any additional precision, then it will

simplify things for researchers to a large extent.

The resulting implementation finds its application at Institutional Level by students and faculty to understand the optimization techniques easily by observing the intermediate steps and details displayed by it. It is also used by the data flow analyzers to experiment with new optimization techniques. As mentioned above, it can be used for deploying efficient and precise IT systems.

Acknowledgments

It gives us great pleasure in presenting the preliminary project report on ***'PRECISION EVALUATION OF POINTER ANALYSIS VARIANTS'****.*

We express our deepest gratitude towards our external guide ***Prof. Supratim Biswas*** *for giving us all the support and guidance we needed. His constant words of encouragement made this journey a very enriching experience.*

We would like to take this opportunity to thank our internal guide ***Prof. Chhaya Gosavi*** *for giving us all the help and guidance we needed. We are really grateful to her for her kind support. Her valuable suggestions were very helpful.*

This research project would not have been possible without the support of ***Miss. Pritam Gharat*** *and all the staff and students of GCC Resource Center, IIT Bombay who offered invaluable assistance, support and guidance.*

We are also grateful to ***Prof. Supriya Kelkar****, Head of Computer Engineering Department, Cummins College of Engineering for Women for her indispensable support and suggestions.*

In the end our special thanks to all the staff members of the Computer engineering Department for providing various resources such as laboratory with all needed software platforms, continuous Internet connection, for Our Project.

Ananya Roy
Jui Shinde
Mugdha Khedkar
(B.E. Computer Engg.)

INDEX

List of Figures

List of Tables

CHAPTER 1

SYNOPSIS

1.1 PROJECT TITLE

Precision Evaluation of Pointer Analysis Variants : A Practical Comparison of Flow and Context Sensitive as well as Insensitive pointer analysis methods.

1.2 PROJECT OPTION

Research based project.

1.3 INTERNAL GUIDE

Prof. Chhaya Gosavi

1.4 SPONSORSHIP AND EXTERNAL GUIDE

The project is sponsored by GCC Resource Center, Department of Computer Science and Engineering, IIT Bombay.

1.5 TECHNICAL KEYWORDS

1. PROGRAMMING LANGUAGES
    (a) Formal Definitions and Theory
        i. Semantics
        ii. Syntax
    (b) Language Classifications
        i. Applicative (functional) languages
        ii. Concurrent, distributed, and parallel languages
        iii. Data-flow languages
        iv. Design languages
        v. Extensible languages
        vi. Macro and assembly languages
    (c) Processors

i. Code generation

ii. Compilers

iii. Interpreters

iv. Memory management (garbage collection)

v. Optimization

vi. Parsing

vii. Preprocessors

viii. Retargetable compilers

ix. Run-time environments

2. LOGICS AND MEANINGS OF PROGRAMS

   (a) Semantics of Programming Languages

      i. Program analysis

1.6 PROBLEM STATEMENT

Compare the results of Flow and Context Sensitive as well as Insensitive pointer analysis methods and observe some useful insights.

1.7 ABSTRACT

Pointer analysis is a static code analysis technique that establishes which pointers can point to which variables, or storage locations. It helps uncover indirect accesses thereby providing useful information about data manipulation by the programs. This improves the precision of program analyses and transformations that have to deal with programs containing pointers. It is not only useful for making the programs more efficient, but can also be used for increasing the reliability of IT based systems deployed in medical communities, banking system, defence industry etc. During the past thirty-five years, hundreds of papers and many Ph.D. theses have been published on pointer analysis. New pointer analysis algorithms are being developed and compared in order to analyze programs with optimized compiler features. The ongoing research in this field predicts the evolution of faster program analyses.

IT systems need to satisfy two basic requirements : Efficiency and Precision. It is important to strike a balance between the two. There are various pointer analysis algorithms but none of them provide satisfactory efficiency and precision together. The algorithms compromising on precision are quicker and easier to implement. Thus, they are everyone's favorite. But they lose out on precision. The primary motivation of the project is to compare all four pointer analysis variants and also to find out whether Context-Sensitive pointer analysis leads to some additional precision if combined with the already efficient (but comparatively less precise) Flow-Insensitive pointer analysis.

1.8 GOALS AND OBJECTIVES

Compiler Optimization is a technique that transforms a program into a semantically equivalent program that uses lesser resources such as CPU, memory and thus, enables faster execution of the program thereby increasing the efficiency and utilization of the processor.

Pointer analysis or points-to analysis, a part of code optimisation techniques, is a static program analysis that determines information on the values of pointer variables or expressions. Such information offers a static model of a program's heap [3]. The information provided by pointer analysis can be further used for the optimisations in an optimising compiler.

The goal of the project is to compare the four variants of pointer analysis based on the parameters of efficiency and precision and gain useful insights from the results generated.

The implementation can be added as an optimising pass in the GCC compiler. The comparison information provided as an output will help the optimiser to select the appropriate variant of pointer analysis to analyse the source code accurately using minimum resources.

1.9 RELEVANT MATHEMATICS ASSOCIATED WITH THE PROJECT

System Description: S = {I, O, F_v, Sc, F}

where,

I = Input

O = Output

F_v = Functions

Sc = Success Conditions

F = Failure Conditions

Initialization :

- Ptr_Stmt = {x=&y, x=y, *x=y, x=*y, *x=&y, *x=*y | x,y $\in$ N}
- sec = {1,2...n} $\in$ N
- P = set of pointers $\in$ N
- P_Set(X) = pointee set of X, $\forall$ X $\in$ P

1. I = {x_1, x_2 ...x_n | x_1, x_2 ...x_n $\in$ Ptr_Stmt}

2. O = {Comparison of O_1, O_2, O_3 and O_4}
   O_1 = {x $\rightarrow$ P_Set(x) | $\forall$ x $\in$ P} (Module 1 ie. FICS)
   O_2 = {x $\rightarrow$ P_Set(x) | $\forall$ x $\in$ P} (Module 2 ie. FICI)
   O_3 = {x $\rightarrow$ P_Set(x) | $\forall$ x $\in$ P} (Module 3 ie. FSCI)
   O_4 = {x $\rightarrow$ P_Set(x) | $\forall$ x $\in$ P} (Module 4 ie. FSCS)

3. F_v = {F_{me}, F_{friend}}

   F_{me} = {F_1}
   F_1 = process input(ptr)
   $\forall$ ptr $\in$ Ptr_Stmt, recognise the pointer statement and decide actions to be taken.

$F_{friend} = \{F_2, F_3, F_4, F_5, F_6, F_7, F_8\}$

F_2 = add into pointee set(i, P_Set(x))

adds i into P_Set(X), where $\begin{cases} i \in N \\ x \in P \\ P\_Set(x) \in P\_Set(X) \end{cases}$

F_3 = display pointee set(x)

prints P_Set(x), where $\begin{cases} x \in P \\ P\_Set(x) \in P\_Set(X) \end{cases}$

F_4 = search into pointee set(i, P_Set(x))

= true if $i \in P\_Set(x)$, where $\begin{cases} i \in N \\ x \in P \\ P\_Set(x) \in P\_Set(X) \end{cases}$

F_5 = display points-to graph

$\{\text{prints } x \rightarrow P\_Set(x) \mid \forall\, x \in P\}$, where $\begin{cases} i \in N \\ x \in P \\ P\_Set(x) \in P\_Set(X) \end{cases}$

F_6 = union of pointee sets(P_Set(x), P_Set(y))

$\{\ P\_Set(x) \cup P\_Set(y)\ \forall\, x \in P\}$, where $\begin{cases} i \in N \\ x \in P \\ P\_Set(x) \in P\_Set(X) \end{cases}$

F_7 = calculate time taken

= t, where $\{\ t \in sec$

F_8 = print no. of failed constraints

= i, where $\{\ i \in N$

4. Sc = { f(x) }

   f(x)= function that compares the points-to graphs and statistics produced by modules 1, 2, 3 and 4.

5. $F = \{F_1, F_2\}$

   $F_1 = \{\text{if input} \notin \text{Ptr\_Stmt}\}$

   $F_2 = \{\text{points-to graph not generated}\}$

1.10 REVIEW OF CONFERENCE/JOURNAL PAPERS SUPPORTING PROJECT IDEA

1. R Padhye, Uday P Khedker : Interprocedural Data Flow Analysis in Soot using Value Contexts.

   A value context is defined by a particular input data flow value reaching a procedure . It is used to enumerate the summary flow functions in terms of (input $\rightarrow$ output) pairs . In order to compute these pairs, data flow analysis within a procedure is performed separately for each context (i.e. input data flow value) . When a new call to a procedure is encountered, the pairs are consulted do decide if the procedure needs to be analysed again .

   - If it was already analysed once for the input value, output can be directly processed.
   - Otherwise, a new context is created and the procedure is analysed for this new context.

2. Marc Shapiro and Susan Horwitz, Fast and accurate flow-insensitive points-to analysis. In Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL 97). ACM, New York, NY, USA, 1-14, 1997.

   Flow-sensitive analysis takes into account the order in which the statements are executed while flow-insensitive analysis assumes that the statements can be executed in any order. Similarly, context-sensitive analysis takes into account the fact that the function must return to the site of the most recent call, while context-insensitive analysis propogates information from a call site, through the called function, and back to all call sites.

1.11 PLAN OF PROJECT EXECUTION

We have used Gantt chart to decide the plan of our project execution. We have illustrated the start and end dates of all our project phases shown in Table 5.1.

Online tool used : SmartSheet

The Gantt chart for our project is as follows:

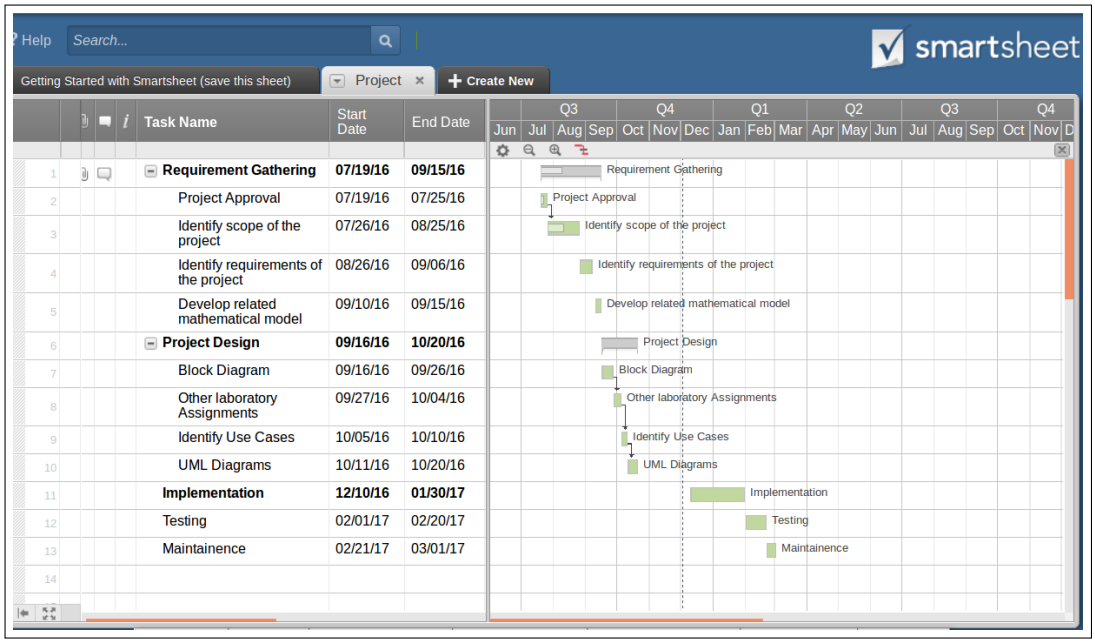


Figure 1.1: Gantt Chart

CHAPTER 2

TECHNICAL KEYWORDS

2.1 AREA OF PROJECT

Compiler Optimization

2.2 TECHNICAL KEYWORDS

Technical Key Words:

1. PROGRAMMING LANGUAGES
    (a) Formal Definitions and Theory
        i. Semantics
        ii. Syntax
    (b) Language Classifications
        i. Applicative (functional) languages
        ii. Concurrent, distributed, and parallel languages
        iii. Data-flow languages
        iv. Design languages
        v. Extensible languages
        vi. Macro and assembly languages
    (c) Processors
        i. Code generation
        ii. Compilers
        iii. Interpreters
        iv. Memory management (garbage collection)
        v. Optimization
        vi. Parsing
        vii. Preprocessors
        viii. Retargetable compilers
        ix. Run-time environments
2. LOGICS AND MEANINGS OF PROGRAMS

(a) Semantics of Programming Languages

    i. Program analysis

CHAPTER 3

INTRODUCTION

3.1 BACKGROUND

A compiler is a program that accepts a source program as input and converts it into a target program which is semantically equivalent [4]. The source program is written in any high level language like C,C++ etc. The target language of the compiler is most often the assembly language. This is further converted into the machine code by the assembler.

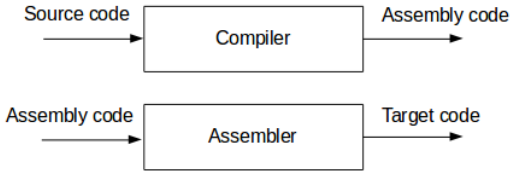


Figure 3.1: Compiler

Optimization is one of the most important phases of the compiler. Compiler optimization is a technique that converts a program into a semantically equivalent program that uses lesser resources than before by performing suitable analysis and transformations [4].

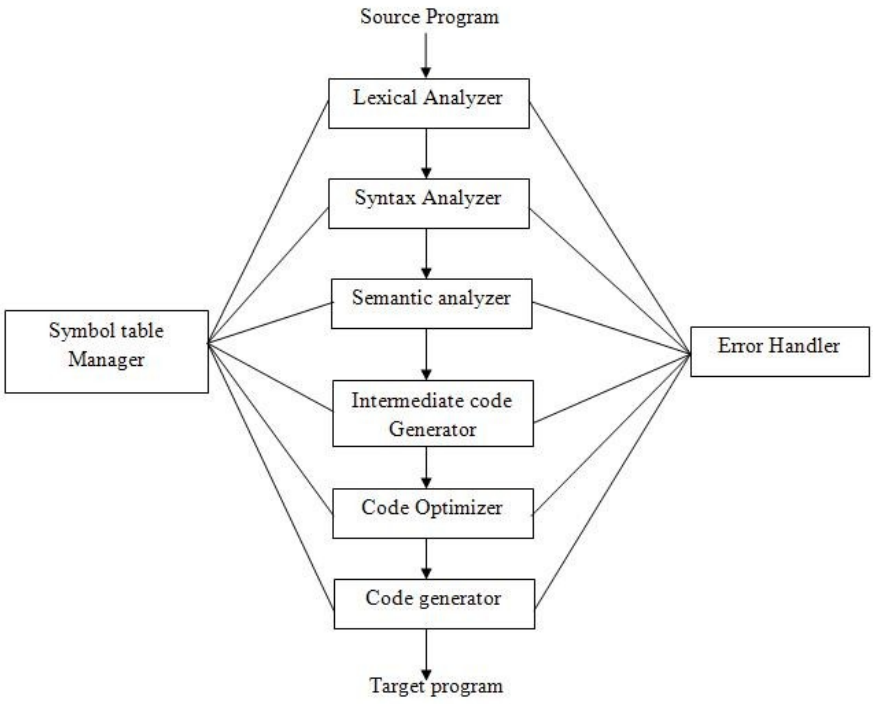


Figure 3.2: Phases of a compiler

3.2 PROJECT IDEA

Compare the results of Flow and Context Sensitive as well as Insensitive pointer analysis methods and observe some useful insights.

3.3 MOTIVATION OF THE PROJECT

Verification and validation of programs is of prime importance in today's world with increased dependence on IT based systems. Pointer analysis helps uncover indirect accesses thereby providing useful information about data manipulation by the programs. This improves the precision of program analyses and transformations that have to deal with programs containing pointers. IT systems need to satisfy two basic requirements : Efficiency and Precision. It is important to strike a balance between the two. The primary motivation of the project is to compare all four pointer analysis variants and also to find out whether context-sensitive pointer analysis leads to some additional precision if combined with the already efficient (but comparatively less precise) flow-insensitive pointer analysis.

3.4 LITERATURE SURVEY

Pointer analysis is a static code analysis technique that establishes which pointers can point to which variables, or storage locations [1]. On the intra procedural level, pointer analysis can be classified as Flow-Insensitive and Flow-Sensitive analysis. It can be classified into Context-Insensitive and Context-Sensitive analysis on the interprocedural level. Pointer analysis aims to determine what memory locations the code uses or modifies. It is useful in many program analyses.

Flow-insensitive pointer analysis does not depend on the control flow of the program. The points-to graphs produced after flow-insensitive analysis are often a superset of the graphs produced after flow-sensitive analysis. Flow-sensitive pointer analysis, which takes the control flow of the program into account, is a more precise analysis although it compromises on speed.

Flow-insensitive pointer analysis computes what memory locations pointers or pointer expressions may refer to at any time in program execution. On the other hand, flow-

sensitive analysis computes memory locations for every point in the program [5]. Flow-insensitive pointer analysis is generally used for whole program analysis because flow sensitive analysis is traditionally very expensive. Flow-insensitive analysis is faster than flow-sensitive analysis. They can be implemented in two ways -

- Inclusion based Using Andersens' algorithm
- Equality based Using Steensgaard's algorithm

On the interprocedural front, context-sensitive pointer analysis remembers the caller-callee relationship. A context-insensitive analysis does not distinguish between distinct calls to a procedure. This causes the propagation of data flow values across interprocedurally invalid paths (i.e. paths in which calls and returns may not match) resulting in a loss of precision. A context-sensitive analysis restricts the propagation to valid paths and hence is more precise [6].

3.4.1 Limitations of the existing system

1. Flow-insensitive pointer analysis (though being efficient) lacks precision as compared to flow-sensitive pointer analysis.
2. Flow-insensitive pointer analysis also produces a larger output than required.This is because the output generated by flow-insensitive analysis is always a superset of the the output generated by flow-sensitive analysis.

3.4.2 Pointer Analysis Variants

1. FLOW-INSENSITIVE CONTEXT-SENSITIVE POINTER ANALYSIS :
   This analysis does not take the control flow of the program into account. It computes what memory locations pointers or pointer expressions may refer to at any time in program execution. Context-sensitive analysis distinguishes between various calling contexts. Also, it restricts the propagation of data flow values to valid paths and hence is more precise than context-insensitive pointer analysis.

2. FLOW-INSENSITIVE CONTEXT-INSENSITIVE POINTER ANALYSIS :
   This analysis is performed using Andersen's algorithm. Andersen's algorithm is a subset-based,inclusion-based algorithm used for computing flow-insensitive points-to information. Andersen's algorithm supports context-insensitive analysis. This means that the analysis does not distinguish between different calling contexts. Also, the validity of paths is not considered. The basic idea of this algorithm is to view pointer assignments as constraints. Then these constraints are used to propagate points-to information.

3. FLOW-SENSITIVE CONTEXT-INSENSITIVE POINTER ANALYSIS :
   This analysis takes the flow of the program into consideration but the context is not considered. It does not distinguish between various calling contexts. It produces separate points-to information at every program statement.

4. FLOW-SENSITIVE CONTEXT-SENSITIVE POINTER ANALYSIS :
   This analysis considers both the flow control as well as the context of the program to be analyzed. It computes a separate solution for each program point. Iterative data flow analysis is required to be performed for Flow-Sensitive and Context-Sensitive Analysis. This analysis is considered to be the most precise and the most difficult to implement.

3.4.3 Data-Flow Analysis

Data flow analysis(DFA) is used for proving facts about programs. These are techniques that derive information about the flow of data along program execution paths [7]. Interprocedural DFA extends the scope of data flow analysis across procedure boundaries [8]. Incorporates the effects of procedure calls in the caller procedures and calling contexts in the callee procedures.

3.4.3.1 Interprocedural Data Flow Analysis Using Value Contexts

A value context is defined by a particular input data flow value reaching a procedure. It is used to enumerate the summary flow functions in terms of (input $\rightarrow$ output)

pairs. In order to compute these pairs, data flow analysis within a procedure is performed separately for each context (i.e. input data flow value). When a new call to a procedure is encounterd, the pairs are consulted to decide if the procedure needs to be analysed again.

- If it was already analysed once for the input value, output can be directly processed.
- Otherwise, a new context is created and the procedure is analysed for this new context.

CHAPTER 4

PROBLEM DEFINITION AND SCOPE

4.1 PROBLEM STATEMENT

Compare the results of Flow and Context Sensitive as well as Insensitive pointer analysis methods and observe some useful insights.

4.1.1 Goals and objectives

Compiler Optimization is a technique that transforms a program into a semantically equivalent program that uses lesser resources such as CPU, memory and thus, enables faster execution of the program thereby increasing the efficiency and utilization of the processor. Pointer analysis or points-to analysis, a part of code optimisation techniques, is a static program analysis that determines information on the values of pointer variables or expressions. Such information offers a static model of a program's heap [3]. The information provided by pointer analysis can be further used for the optimisations in an optimising compiler.

The goal of the project is to compare the four variants of pointer analysis based on the parameters of efficiency and precision and gain useful insights from the results generated.

The implementation can be added as an optimising pass in the GCC compiler. The comparison information provided as an output will help the optimiser to select the appropriate variant of pointer analysis to analyse the source code accurately using minimum resources.

4.1.2 Statement of scope

During the past thirty-five years, hundreds of research papers have been published on pointer analysis. New pointer analysis algorithms are being developed and compared in order to analyze programs with optimized compiler features. The ongoing research in this field predicts the evolution of faster program analyses. Also, the implemented code can be used as a pass in GCC for compiler optimization i.e pointer analysis and can provide some useful results for the study that combines flow-insensitivity with context-sensitivity and context-insensitivity.

4.2 MAJOR CONSTRAINTS

- The implemented code can perform the static analysis of source codes written in only C and C++ programming languages.
- The pointer constraints which are not of the form x=&y, x=y, *x=y, x=*y, *x=&y, *x=*y will not be considered during analysis.
- Pointer arithmetic statements will be ignored during the analysis.

4.3 METHODOLOGIES OF PROBLEM SOLVING AND EFFICIENCY ISSUES

The substitute solutions available are implementations which can handle double and triple pointer statements. As we have observed, generally most of the programs to be analyzed have the pointer statements of the types x=&y, x=*y, x=y, *x=y, *x=&y and x=y which has been handled by our implementation. An alternative solution can be to handle constraints beyond the ones mentioned. We have used the vector data structure in order to extract the pointer statements. The implementation was already available with the GCC resource center and hence we have reused the same. An alternative solution may make use of other data structures such as forward lists, dequeues and lists in order to perform the same function.

ADVANTAGES:

1. More number of pointer statements can be handled.
2. Using the list data structure for extracting the pointer statements helps in better extraction, insertion and moving of the pointer statements withing the container for which the iterator has been obtained.

DISADVANTAGES:

1. Complexity of the code is increased.
2. Vectors are relatively more efficient in accessing elements as well as addition and removal of elements from the end. By using any other data structure, we will compromise on this efficiency.

4.4 OUTCOME

The implementation will be able to compare among the four types of pointer analysis variants and provide useful insights regarding precision and efficiency of each variant.

4.5 APPLICATIONS

- This implementation finds its application at Institutional Level by students and faculty to understand the pointer analysis techniques easily and their relative differences by observing the details displayed by it.
- Pointer analysis finds it application in the fields of several client analyses like typestate verification [9], security analysis [10] and bug detection [11]. All these fields can benefit from the information derived by the comparison of the pointer analysis variants as it helps them determine the suitable technique for their application specific analysis.

4.6 HARDWARE RESOURCES REQUIRED

64 bit / 32 bit Intel processor machine.

4.7 SOFTWARE RESOURCES REQUIRED

1. Operating System: Ubuntu14.04 LTS
2. IDE: None
3. Programming Language: C / C++
4. GCC Version: 4.7.2
5. Additional Libraries:
    - Libgmp-dev
    - Libmpfr-dev

- Libmpc-dev
- Libcloog-ppl-dev

CHAPTER 5

PROJECT PLAN

5.1 PROJECT ESTIMATES

5.1.1 Reconciled Estimates

5.1.1.1 Time Estimate

Activity	Start Date	End Date
Requirement Analysis	19/7/16	15/9/16
Project Design	16/9/16	20/10/16
Implementation	10/12/16	30/1/17
Testing	1/2/17	20/2/17
Maintainence	21/2/17	1/3/17

Table 5.1: Time Estimate

5.1.2 Project Resources

5.1.2.1 Hardware Resources

64 bit / 32 bit Intel processor machine.

5.1.2.2 Software Resources

Software Name	Version
Ubuntu	14.04
gcc	4.7.2
SPEC Benchmark	CPU 2006
make	4.1

Table 5.2: Software Resources

5.2 RISK MANAGEMENT

This section discusses Project risks and the approaches that can be followed to manage them. The risks for the Project can be analyzed within the constraints of time and quality.

5.2.1 Risk Identification and Analysis

Various possible risks are identified as follows :

ID	Risk Description	Probability	Impact		
			Schedule	Quality	Overall
1	Unexpected input	Medium	Medium	High	Medium
2	Corruption of files	Low	Low	Medium	Low
2	System crashes	Low	Low	High	High

Table 5.3: Risk Table

Probability	Value	Description
High	Probability of occurrence is	$> 75\%$
Medium	Probability of occurrence is	$26 - 75\%$
Low	Probability of occurrence is	$< 25\%$

Table 5.4: Risk Probability definitions [2]

Impact	Value	Description
Very high	$> 10\%$	Schedule impact or Unacceptable quality
High	$5 - 10\%$	Schedule impact or Some parts of the project have low quality
Medium	$< 5\%$	Schedule impact or Barely noticeable degradation in quality Low Impact on schedule or Quality can be incorporated

Table 5.5: Risk Impact definitions [2]

5.2.2 Overview of Risk Mitigation, Monitoring, Management

Following are the details for each risk.

Risk ID	1
Risk Description	Unexpected input
Category	Development Environment.
Source	Software requirement Specification document.
Probability	Medium
Impact	Medium
Response	Mitigate
Strategy	Ignore such inputs eg. pointer arithmetic.
Risk Status	Identified

Risk ID	2
Risk Description	Corruption of files
Category	Development Environment.
Source	Software Design Specification documentation review.
Probability	Low
Impact	High
Response	Mitigate
Strategy	Taking regular backups of files.
Risk Status	Identified

Risk ID	3
Risk Description	System crashes
Category	Development Environment.
Source	Implementation and testing phase.
Probability	Low
Impact	Very High
Response	Accept
Strategy	Keep backup of the entire system
Risk Status	Identified

5.3 PROJECT SCHEDULE

5.3.1 Project task set

The project is divided into 3 main tasks as follows:

- *Task 1*: The first task is to map the input file in C/C++ and extract the pointer statements. These pointer statements will be further processed using the pointer analysis pass. Pointer statements are stored in a vector recognised by our implementation. The LHS and RHS variables (eg. x,y etc) and indirection (*, & etc.) is processed to recognise various kinds of pointer statements.

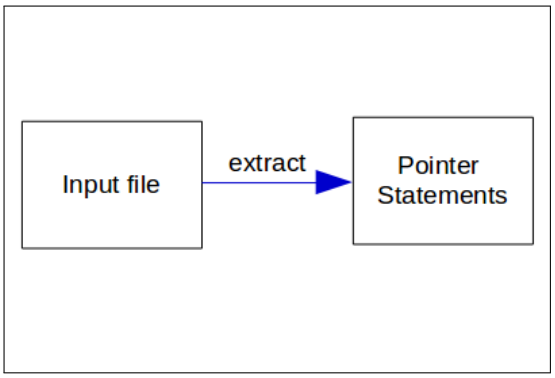


Figure 5.1: Task 1

- *Task 2*: The next task is to recognise these pointer statements and generate pointer analysis pass for all possible variants (flow-insensitive context-insensitive, flow-insensitive context-sensitive, flow-sensitive context-insensitive and flow-sensitive context-sensitive). This is the basic implementation phase of the project.

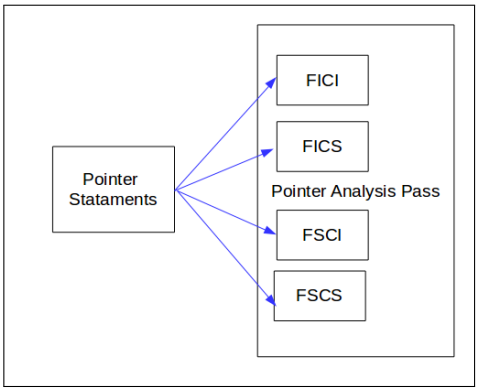


Figure 5.2: Task 2

- *Task 3*: The final task is to test the pointer analysis pass and compare the results of all pointer analysis variants.

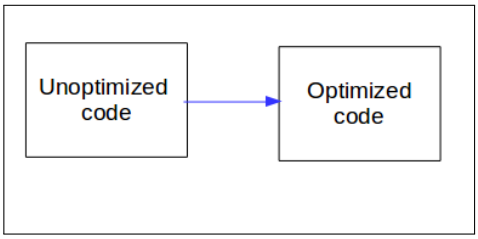


Figure 5.3: Task 3

5.3.2 Task network

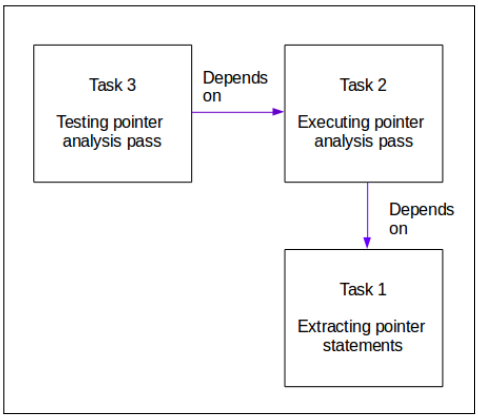


Figure 5.4: Task Network

5.3.3 Timeline Chart

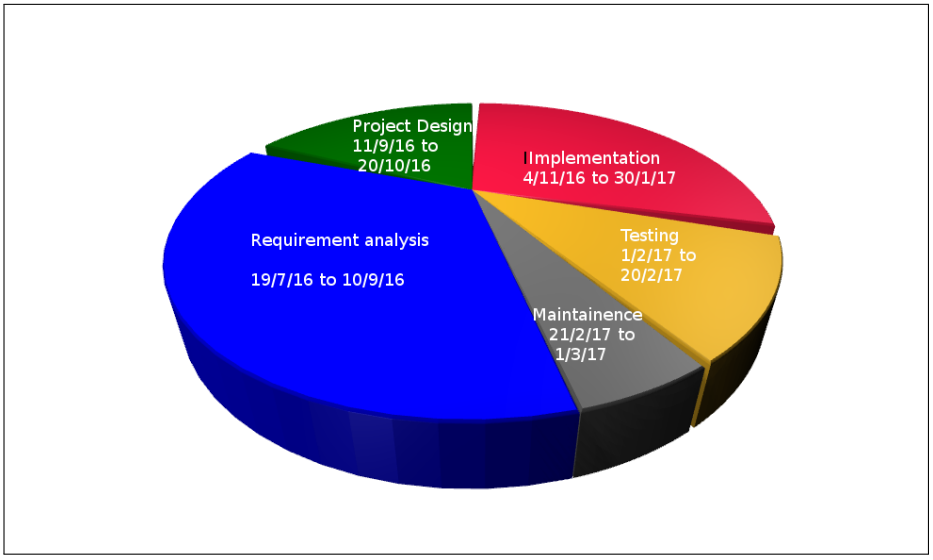


Figure 5.5: Timeline Chart for the project

5.4 TEAM ORGANIZATION

5.4.1 Team structure

The team consists of 3 people. The structure is as follows:

Team Member	Roles and Responsibilities
Team Leader	Assign tasks to everyone Requirement Analysis Documentation **Laboratory Assignments :** Mathematical Model Synopsis
Team Member 1	Literature Survey Documentation **Laboratory Assignments :** IDEA Matrix Divide and Conquer strategies UML diagrams
Team Member 2	Requirement Analysis Documentation **Laboratory Assignments :** Test plan Problem Classification UML Diagrams

Table 5.6: Team Structure

5.4.2 Management reporting and communication

The roles and responsibilites mentioned in Table 5.6 were communicated to all the members through regular meetings and discussions. The list of meetings is given below :

Meeting No.	Date	Coordinated by	Topic of Discussion
1	9/7/16	External Guide	Literature Survey Basics of Pointer Analysis
2	20/7/16	Team Leader	Requirement gathering
3	17/8/16	Team Leader	Literature Survey (Solving Pointer Analysis examples)
4	9/9/16	Team Leader	Literature Survey (Solving Pointer Analysis examples)
5	20/9/16	Internal Guide	Laboratory Assignments
6	26/9/16	Internal Guide	Laboratory Assignments
7	1/10/16	External Guide	Available expressions analysis using value Context method
8	4/10/16	External Guide Internal Guide	Installation steps Concepts required for the implementation
9	4/11/16	External Guide	Algorithm for FICS implementation Changes required to make it FICI

Table 5.7: Meetings Held

CHAPTER 6

SOFTWARE REQUIREMENT SPECIFICATION

6.1 INTRODUCTION

6.1.1 Purpose and Scope of Document

The purpose of this document is to provide a detailed analysis of different pointer analysis variants, namely Flow-Insensitive Context-Insensitive, Flow-Insensitive Context-Sensitive, Flow-Sensitive Context-Insensitive, Flow-Sensitive Context-Sensitive. It will explain the following:

1. How Context-Sensitivity will add more precision to already efficient Flow-Insensitive Context-Insensitive analysis.
2. The input required and output generated.
3. The constraints under which the code must execute i.e six constraints of the form: x=&y, x=y, x=*y, *x=y, *x=&y, x=y
4. Use case scenarios, mathematical model and test cases.
5. Software required for the project.

6.1.2 Overview of responsibilities of Developer

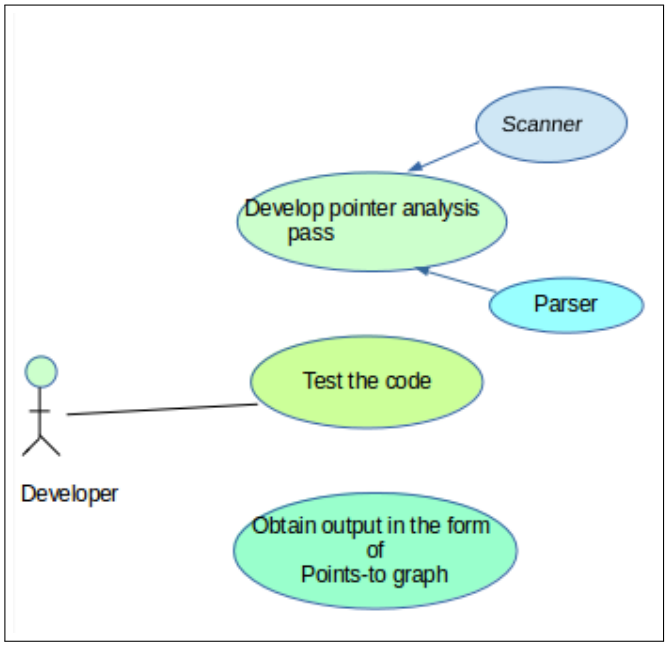


Figure 6.1: Responsibilites of the Developer

6.2 USAGE SCENARIO

This section provides various usage scenarios for the system to be developed.

6.2.1 User profiles

The users for this implementation would be :

1. Teachers who are competent in knowledge of compilers and pointer analysis.
2. Engineering students who have a basic knowledge of compilers and pointer analysis.
3. Researchers who would like to add their own study in the field of pointer analysis.

6.2.2 Use-cases

All use-cases for the software are presented. Description of all main Use cases using use case template is provided.

Sr No.	Use Case	Includes	Actors	Description
1	Give the input program	Constraints of the type x=&y, x=*y, x=y etc	User	Input C++ file given by the user
2	Select type of pointer analysis	Null	User	Choose type of pointer analysis
3	Pointer analysis pass	Scanning and parsing	Developer	The pointer analysis pass is scanned and parsed to generate the output

Table 6.1: Use Cases

6.2.3 Use Case View

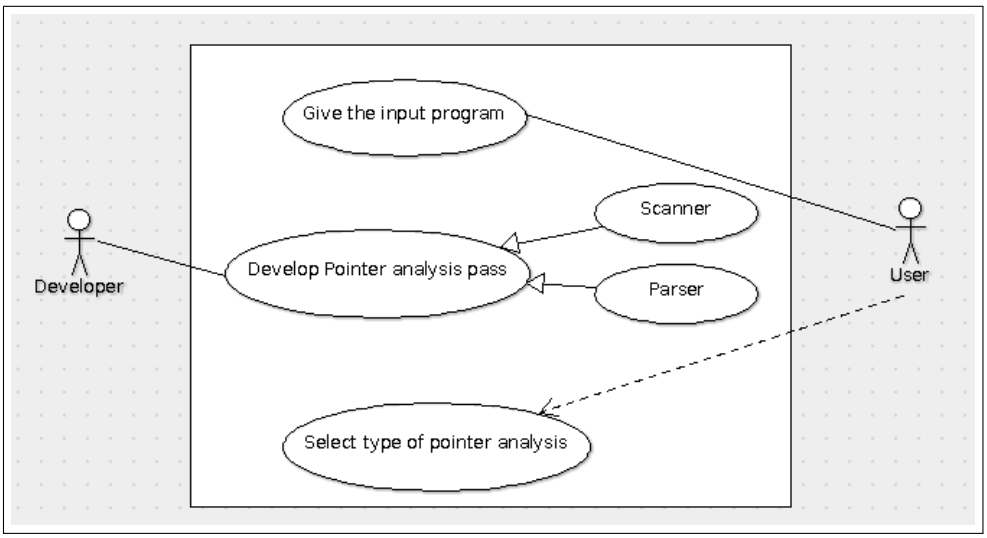


Figure 6.2: Use case diagram

6.3 DATA MODEL AND DESCRIPTION

6.3.1 Data Description

1. Input program: The input file is given by the user in C++. It will contain the following six constraints:

   - x = &y
   - x = *y
   - x = y
   - *x = y
   - *x = &y
   - *x = *y

2. Types of pointer analysis to be compared:

   - *Flow-Insensitive Context-Insensitive :* In flow-insensitive, order of the statements does not matter. This means that the analysis does not distinguish between different calling contexts. Also, the validity of paths is

not considered. The basic idea of this algorithm is to view pointer assignments as constraints. Then these constraints are used to propagate points-to information.

- *Flow-Insensitive Context-Sensitive :* It does not take the control flow of the program into account. It computes what memory locations pointers or pointer expressions may refer to at any time in program execution. Context-sensitive analysis distinguishes between various calling contexts. Also, it restricts the propagation of data flow values to valid paths and hence is more precise than context-insensitive pointer analysis
- *Flow-Sensitive Context-Insensitive :* This analysis takes the flow of the program into consideration but the context is not considered. It does not distinguish between various calling contexts. It produces separate points-to information at every program statement.
- *Flow-Sensitive Context-Sensitive :* This analysis considers both the flow control as well as the context of the program to be analyzed. It computes a separate solution for each program point. This analysis is considered to be the most precise and the most difficult to implement.

3. Interprocedural data flow analysis : Data flow analysis(DFA) is used for proving facts about programs. These are techniques that derive information about the flow of data along program execution paths [7]. Interprocedural DFA extends the scope of data flow analysis across procedure boundaries [8]. Incorporates the effects of procedure calls in the caller procedures and calling contexts in the callee procedures.

6.4 FUNCTIONAL MODEL AND DESCRIPTION

6.4.1 Data Flow Diagram

Data Flow Diagram Level 0

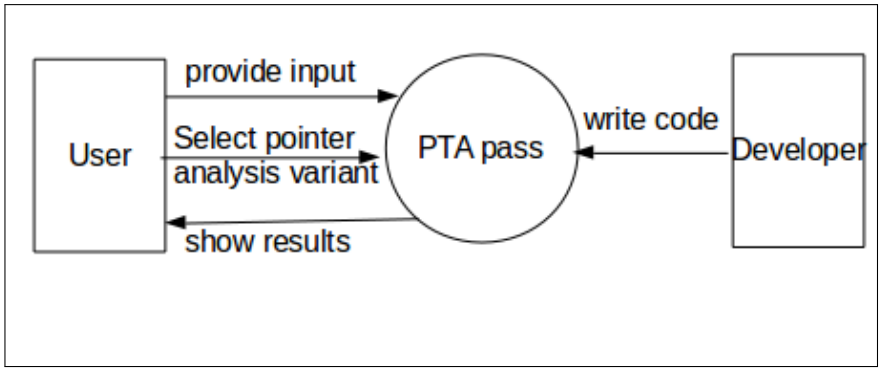


Figure 6.3: Data Flow Diagram Level 0

Data Flow Diagram Level 1

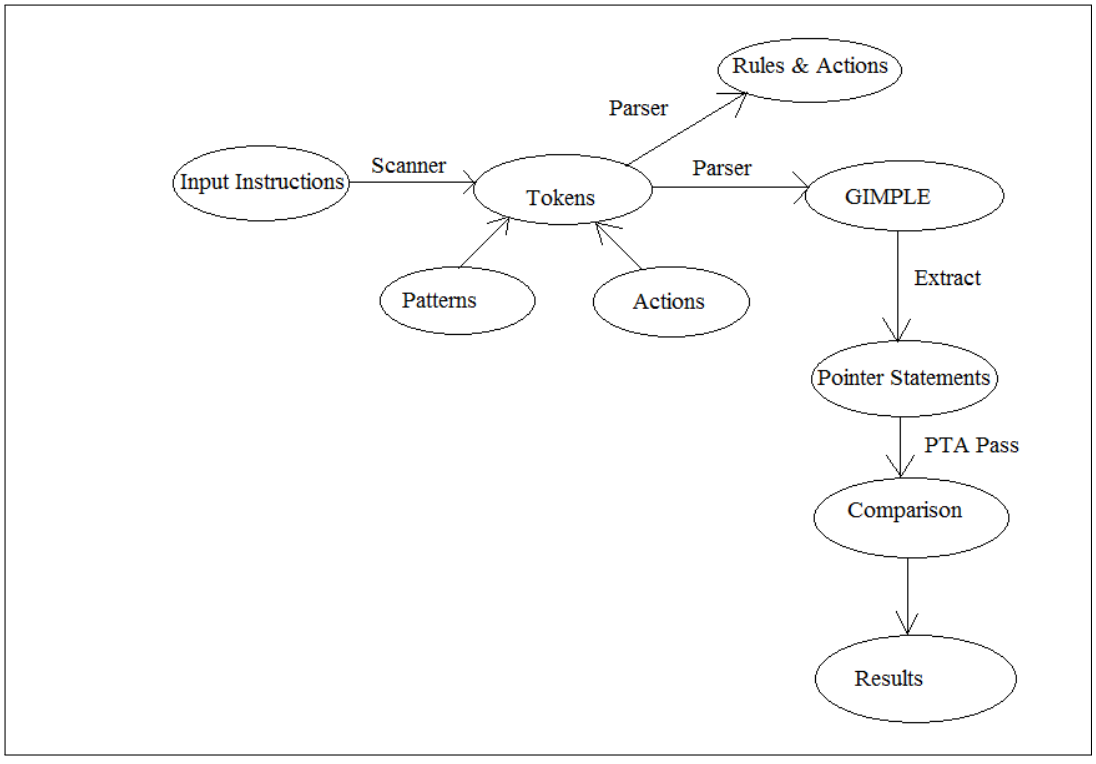


Figure 6.4: Data Flow Diagram Level 1

6.4.2 Description of functions

Description of functions in the implementation is as follows:

1. *Input file :* The language used for the input file is C++ and it will contain the pointer constraints such as x = &y, x = y, etc. The program can be tested by various input files using the SPEC banchmark.

2. *Perform analysis :* Flow-Insensitive Context-Insensitive, Flow-Insensitive Context-Sensitive, Flow-Sensitive Context-Insensitive and Flow-Sensitive Context-Sensitive pointer analysis will be performed.

3. *Generate pointee set :* For each variable of the pointer constraints, the pointee set will be generated which will further be used to display the output,i.e. the points-to graph.

4. *Generate points-to graph :* The variable and its pointee set will constitute the points-to graph. For each basic block, a points-to graph will be generated.

6.4.3 Non Functional Requirements

- Interface requirements : Hardware interface : No hardware interface required. Software interface for GCC 4.7.2 :
  - libgmp-dev package
  - libmpfr-dev package
  - libmpc-dev package
  - libcloog-ppl-dev package
- Performance Requirements : The output (i.e. the points-to graph) should be generated within just a fraction of seconds (some milliseconds). It should handle one input file at a time consisting of the pointer constraints. Any constraint other than the specified six constraints should not be analysed. The points-to graph should be generated for each basic block.
- Project Quality Attributes :

- *Maintainability :* For proper input file, expected output is generated. It displays error conditions if any and notifies appropriate steps to be taken.
- *Reliability :* The performance of the system is fast and it generates the correct points-to graph.
- *Efficiency :* According to the given input file, the system performs Flow-Insensitive Context-Insensitive, Flow-Insensitive Context-Sensitive, Flow-Sensitive Context-Sensitive and Flow-Sensitive Context-Insensitive analysis in optimum time.
- *Reusability :* The implemented code can be used as a pass in GCC for compiler optimization i.e pointer analysis and can provide some useful results for the study that combines flow-insensitivity with context-sensitivity and context-insensitivity.
- *Usability :* The system is user friendly since the people working on this software need to have very little knowledge about the internal working of the tool.
- *Testabilty :* The testability of the code can be checked using the SPEC benchmarks.

6.4.4 Design Constraints

- ASSUMPTIONS

  It is assumed that input to the system should be given in C++ or C language and the input file should contain the specified six pointer constraints. Other type of constraints should be ignored.

- LIMITATIONS

  1. The input file will not work if it is in any language other than C or C++.
  2. It will ignore live variable information.

6.4.5 Activity Diagram

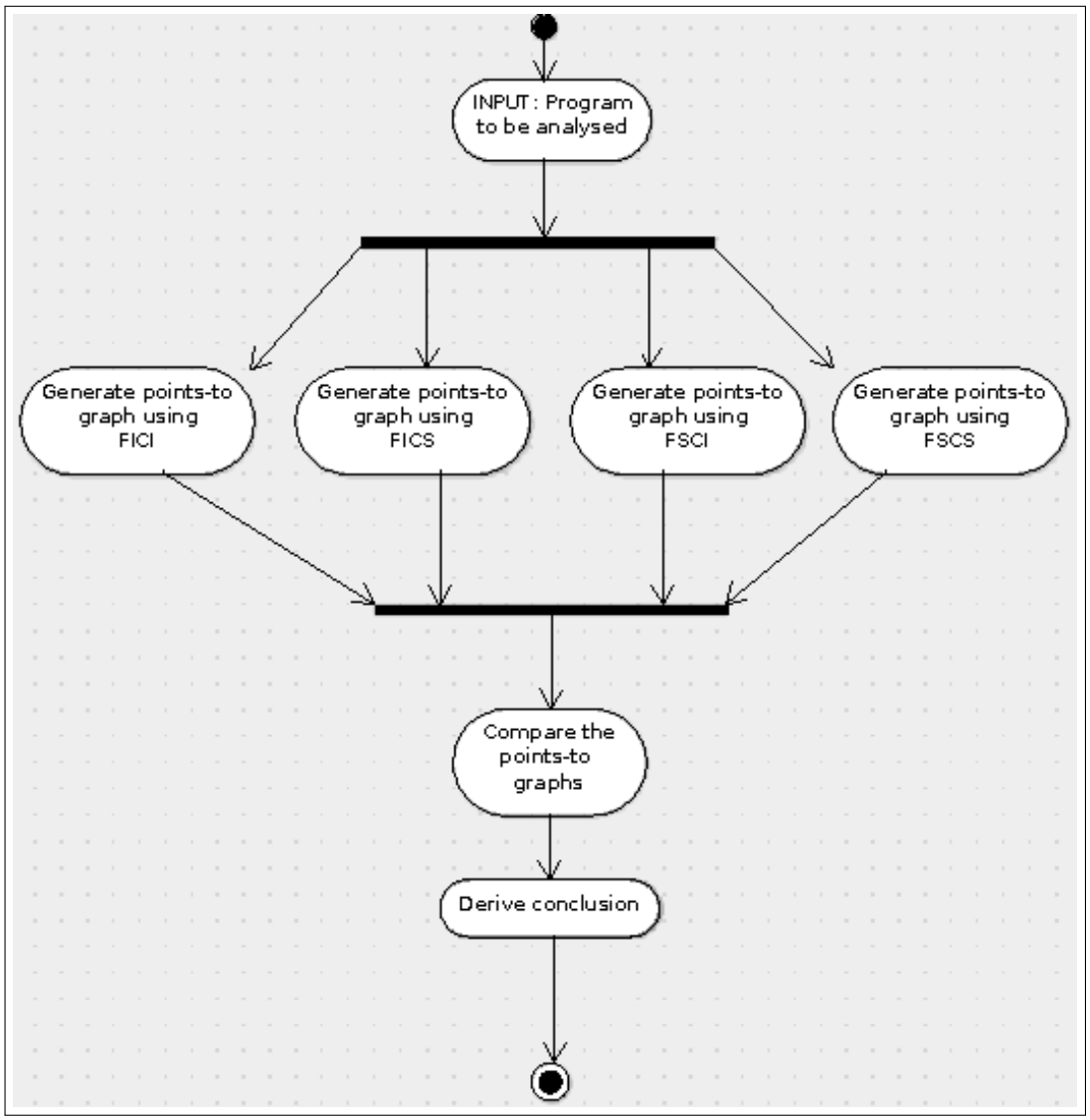


Figure 6.5: Activity diagram

6.4.6 Software Interface Description

The software interface(s) to the outside world are described as follows:

1. libgmp-dev package
2. libmpfr-dev
3. libmpc-dev

4. libcloog-ppl-dev

5. make utility

6.4.7 State Diagram

Fig.6.6 shows the state transition diagram of the implementation.

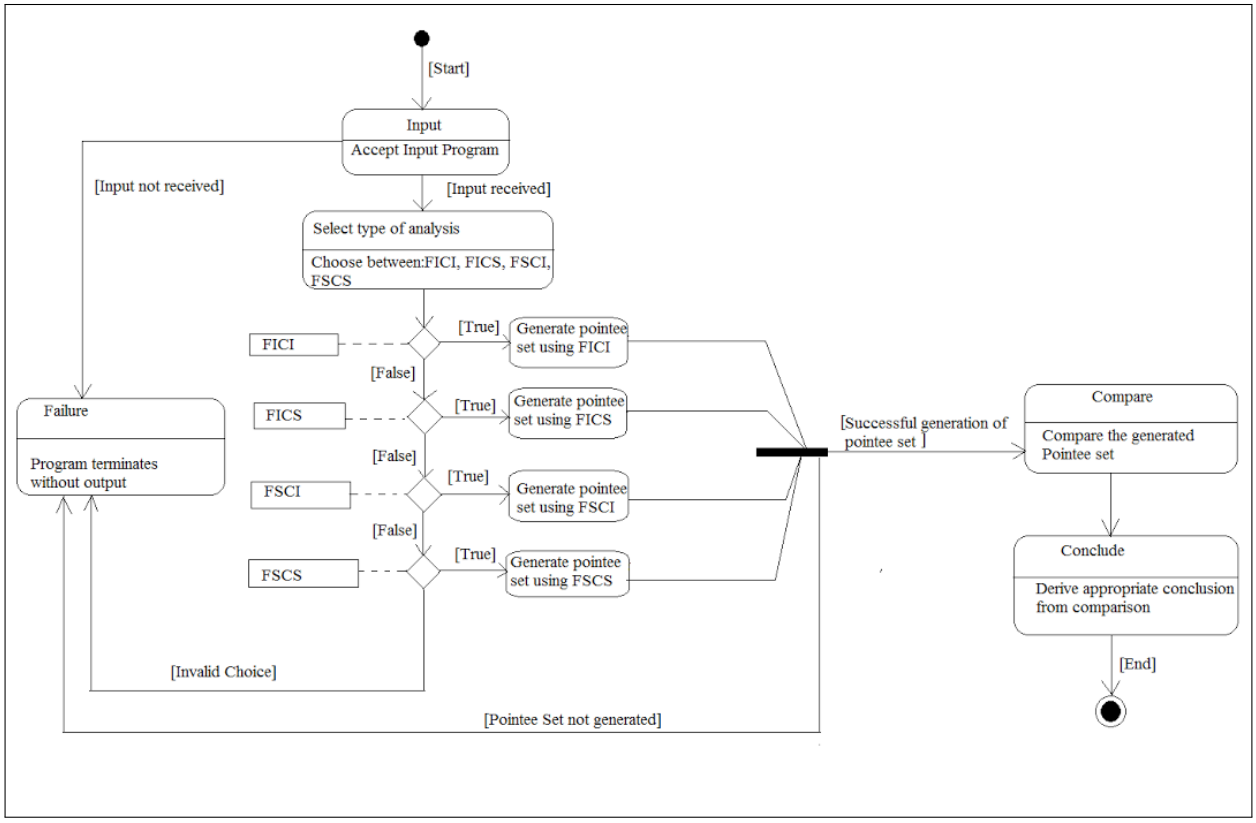


Figure 6.6: State transition diagram

CHAPTER 7

DETAILED DESIGN DOCUMENT USING APPENDIX A AND B

7.1 INTRODUCTION

This document specifies the design that is used to solve the problem.

7.2 ARCHITECTURAL DESIGN

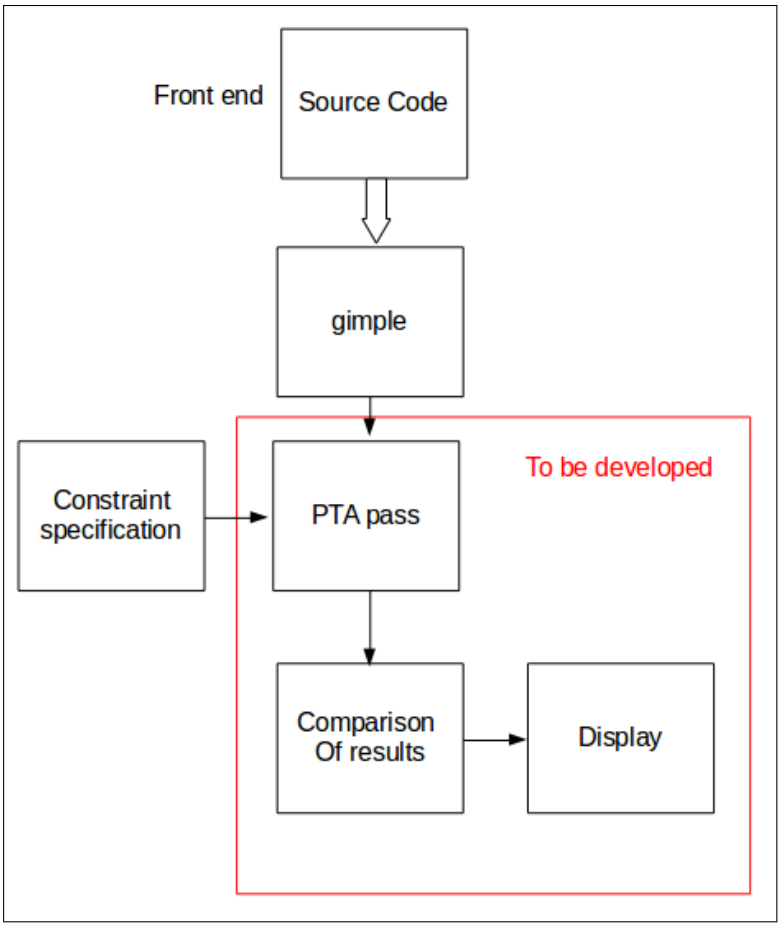


Figure 7.1: Architecture Diagram

7.3 DATA DESIGN (USING APPENDICES A AND B)

7.3.1 Internal software data structure

Tokens are generated by the Lexical Analyser while scanning the input specification file. A symbol table is generated which stores the tokens, their data type, value and other related information. This symbol table is used by the Parser for Syntactic Analysis.

7.3.2 Global data structure

7.3.2.1 Lattice

Lattice is a data structure used to store information about the data flow analysis.

7.3.2.2 Data Flow Variables

The variables are stored in two sets, namely, In and Out, during the data flow analysis of the input IR. For populating these sets, the global data flow equations are used. These can be indirectly used to check the generated points-to graph for every context.

CHAPTER 8

SUMMARY AND CONCLUSION

8.1 SUMMARY

This report explains the variants of pointer analysis and the need for their comparison. It presents information regarding data flow analysis, pointer analysis techniques and the parameters to be used for comparison.The implementation aims at successfully comparing the pointer analysis variants by observing the control flow graph generated as output. The observations made can further help in determining the relative importance of the analysis techniques and whether the addition of flow-sensitivity or context-sensitivity adds any precision when combined with their insensitive variants (flow-insensitive or context-insensitive). The precision, efficiency as well as the trade-off between the two is also recorded.

The suitable analysis technique can then be selected based on the observations made, thereby increasing the precision as well as efficiency of analysis. The observations are to be made programmatically as the design aims at analysing source codes containing thousands of lines of code. The amount of resources required can be decreased and the available resources can be used wisely by having the knowledge about the results of comparison of the pointer analysis variants.

8.2 CONCLUSION

The implementation can be added as a PTA pass in the GCC compiler. An additional PTA pass can increase the optimising capability of an optimising compiler such as GCC. The result of the analysis can be used by researchers to aid their further research in this area and used by students, faculty and at an Institutional level to understand the pointer analysis techniques.

REFERENCES

[1] “Pointer analysis:https://en.wikipedia.org.”

[2] R. S. Pressman, *Software Engineering (3rd Ed.): A Practitioner’s Approach*. New York, NY, USA: McGraw-Hill, Inc., 1992.

[3] G. B. Yannis Smaragdakis, “Pointer analysis, foundations and trends in programming languages vol. 2,” tech. rep., University of Athens, 2013.

[4] J. U. M. S. L. Alfred Aho and R. Sethi, *COMPILERS : Principles,Techniques and Tool.* Pearson, 2014.

[5] “Ben hardekopf university of california, santa barbara benh@cs.ucsb.edu.”

[6] U. P. K. R Padhye, “Interprocedural data flow analysis in soot using value contexts,” tech. rep., Indian Institute of Technology,Bombay, 2013.

[7] Y. Srikant, “Nptel course slides: Data-flow analysis.”

[8] U. Khedker, “Lecture slides: Interprocedural data flow analysis,” 2015.

[9] S. J. F. E. Y. N. D. G. Ramalingam and E. Geay, “Effective typestate verication in the presence of aliasing,” 2008.

[10] B. S. W. Chang and C. Lin, “Efficient and extensible security enforcement using dynamic dataflow analysis,” 2008.

[11] C. L. S. Z. Guyer, “Error checking with client-driven pointer analysis,” 2008.

ANNEXURE A

LABORATORY ASSIGNMENTS ON PROJECT ANALYSIS OF ALGORITHMIC DESIGN

Assignment 1

Develop the Problem under consideration and justify feasibility using concepts of knowledge canvas and IDEA Matrix.

The IDEA Matrix describes how to design a system efficiently by defining the following factors with respect to a system. The IDEA Matrix for this system is as follows :

I	D	E	A
Increase Precision, Efficiency	Drive Improve trade-off between precision and efficiency	Educate Pointer Analysis, Data Flow Analysis	Accelerate Compiler Optimisation
Improve Balance between precision and efficiency	Deliver Conclusion	Evaluate Improvement in precision	Associate Code Optimisation with improved IT systems
Ignore More than 50,000 lines of code*	Decrease Time of execution	Eliminate Ambiguity, Errors in code	Avoid Unprocessed constraints

Note : * indicates that this value is subject to change based on testing phase.

INCREASE:

Increasing the overall precision as well as efficiency of the pointer analysis.

IMPROVE:

Improving the balance between the precision and efficiency by finding whether context sensitive pointer analysis increases precision when combined with the efficient but comparatively less precise flow insensitive pointer analysis.

IGNORE:

Ignore programs exceeding 50,000 lines of code.

DRIVE:

Drive Compiler Optimisation by improving the trade-off between efficiency and precision.

DELIVER:

Deliver a conclusion whether the precision of the efficient flow insensitive analysis is improved with the addition of context sensitive analysis.

DECREASE:

Decrease the time taken to process the pointer statements.

EDUCATE:

Educate the team members about data flow analysis, pointer analysis and the various techniques to achieve the same.

EVALUATE:

Evaluate improvement in precision by the addition of context sensitivity to flow insensitive analysis.

ELIMINATE:

Eliminate ambiguity as well as errors in code.

ACCELERATE:

Accelerate compiler optimization through efficient pointer analysis.

ASSOCIATE:

Associate machine independent code optimisation with improved, efficient and safe IT systems.

AVOID:

Avoid the unprocessed constraints, that is, the constraints which are not of the form x=&y, x=y, *x=y, x=*y, *x=&y, *x=*y

Assignment 2

Project Problem statement feasibility assessment using NP-Hard, NP-Complete or satisfiability issues using modern algebra and relevant mathematical models.

All modules in the project can be classified into P, NP-Complete or NP-Hard classes as shown below:

Module	Class
Flow-Insensitive Context-Insensitive	P
Flow-Insensitive Context-Sensitive	P
Flow-Sensitive Context-Insensitive	P*
Flow-Sensitive Context-Sensitive	P*
Measurement and Comparison of Data	P

Note : * indicates that this value is subject to change based on the input program. If the input program has multiple level pointers or structures, the problem can become NP-Hard. Such input programs are beyond the scope of this project.

ANNEXURE B

LABORATORY ASSIGNMENTS ON PROJECT QUALITY AND RELIABILITY TESTING OF PROJECT DESIGN

Assignment 3

Use of Divide and conquer strategies to exploit distributed or parallel or concurrent processing of the above to identify objects, morphisms, overloading in functions (if any) and functional relations and any other dependencies.

The modules of our project are as follows:

1. FLOW-INSENSITIVE CONTEXT-SENSITIVE : This analysis does not take the control flow of the program into account.It computes what memory locations pointers or pointer expressions may refer to at any time in program execution. Context-sensitive analysis distinguishes between various calling contexts. Also, it restricts the propogation of data flow values to valid paths and hence is more precise than context-insensitive analysis.

2. FLOW-INSENSITIVE CONTEXT-INSENSITIVE : This analysis is performed using Andersens algorithm. Andersens algorithm is a subset-based, inclusion-based algorithm used for computing flow-insensitive points-to information. Andersens algorithm supports context-insensitive analysis. This means that the analysis does not distinguish between different calling contexts. Also, the validity of paths is not considered. The basic idea of this algorithm is to view pointer assignments as constraints. Then these constraints are used to propagate points-to information. The time complexity is $O(n^3)$.

3. FLOW-SENSITIVE CONTEXT-INSENSITIVE POINTER ANALYSIS : This analysis takes the flow of the program into consideration but the context is not considered. It does not distinguish between various calling contexts. It produces separate points-to information at every program statement.

4. FLOW-SENSITIVE CONTEXT-SENSITIVE POINTER ANALYSIS : This analysis considers both the flow control as well as the context of the program

to be analyzed. It computes a separate solution for each program point. Iterative data flow analysis is required to be performed for Flow-Sensitive and Context-Sensitive Analysis. This analysis is considered to be the most precise and the most difficult to implement.

5. MEASUREMENT AND COMPARISON OF DATA : This module deals with the measurements (this includes time taken, number of pointer statements, number of failed constraints, time taken to process each constraint etc.) and comparison of data recorded from both previous modules. Dierent statistics can provide valuable insights and the output generated can be used to nd out if context-sensitivity leads to a signicant increase in precision when combined with ow-insensitive pointer analysis.

Assignment 4

Use of above to draw functional dependency graphs and relevant Software modeling methods, techniques including UML diagrams or other necessities using appropriate tools.

State Transition Diagram

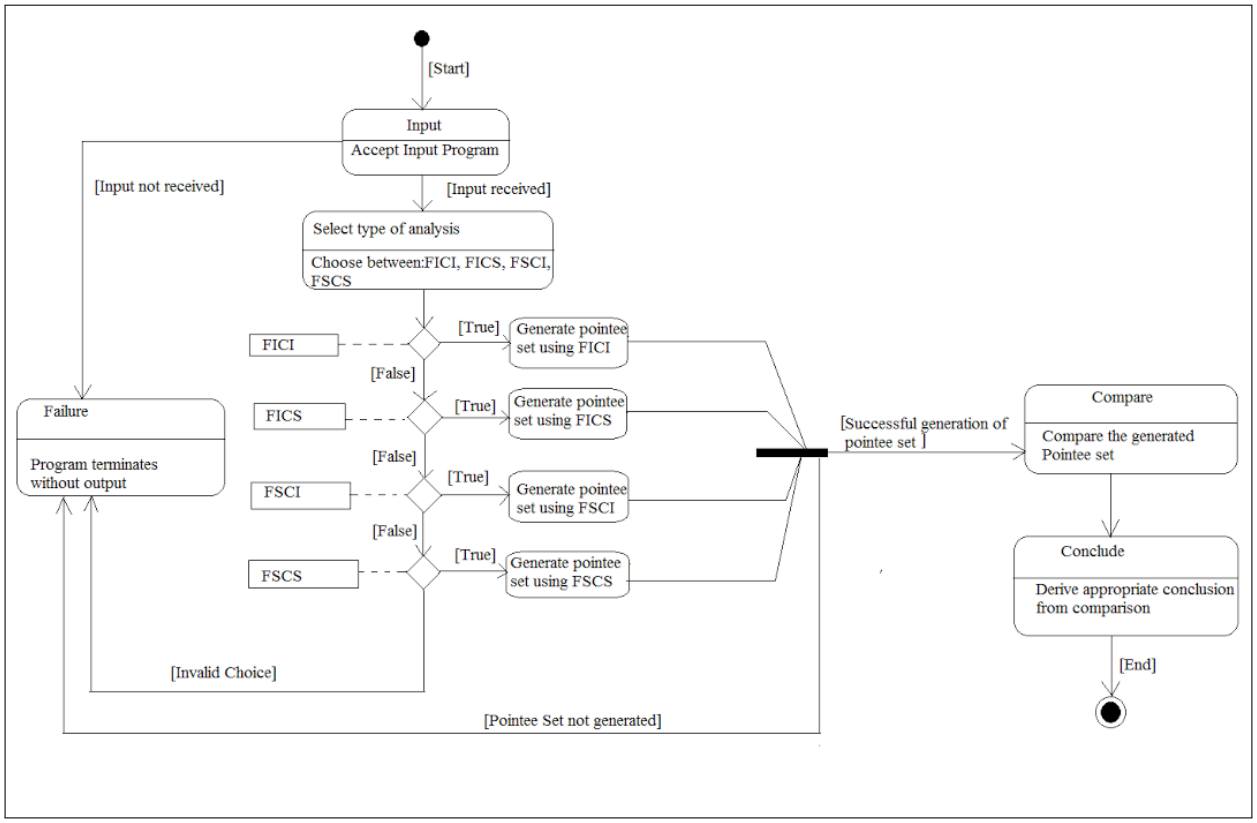


Figure B.1: State diagram

Use Case Diagram

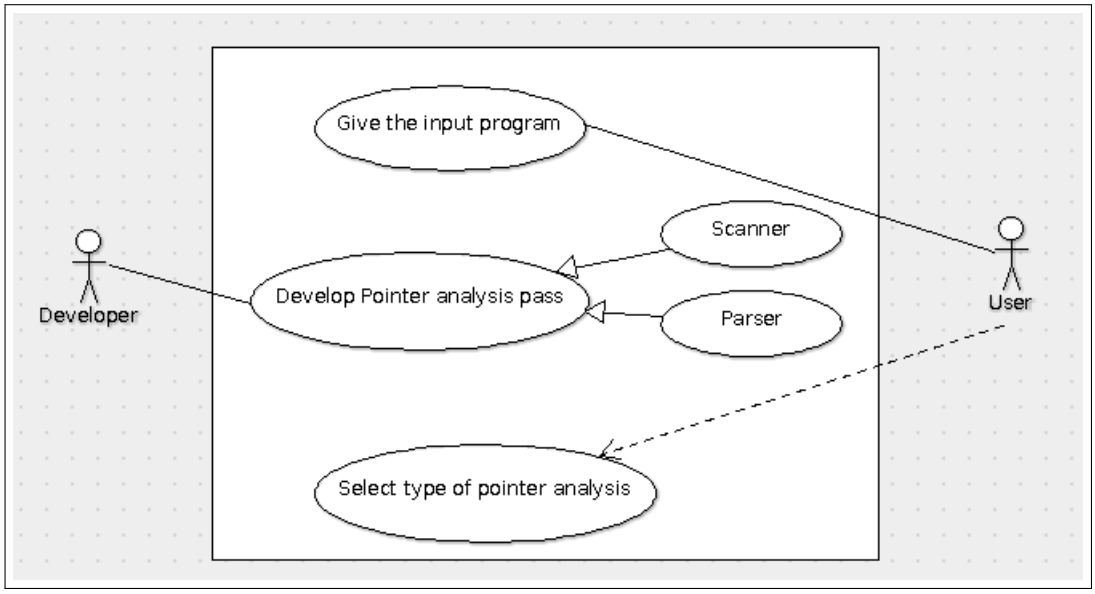


Figure B.2: Use case diagram

Activity Diagram

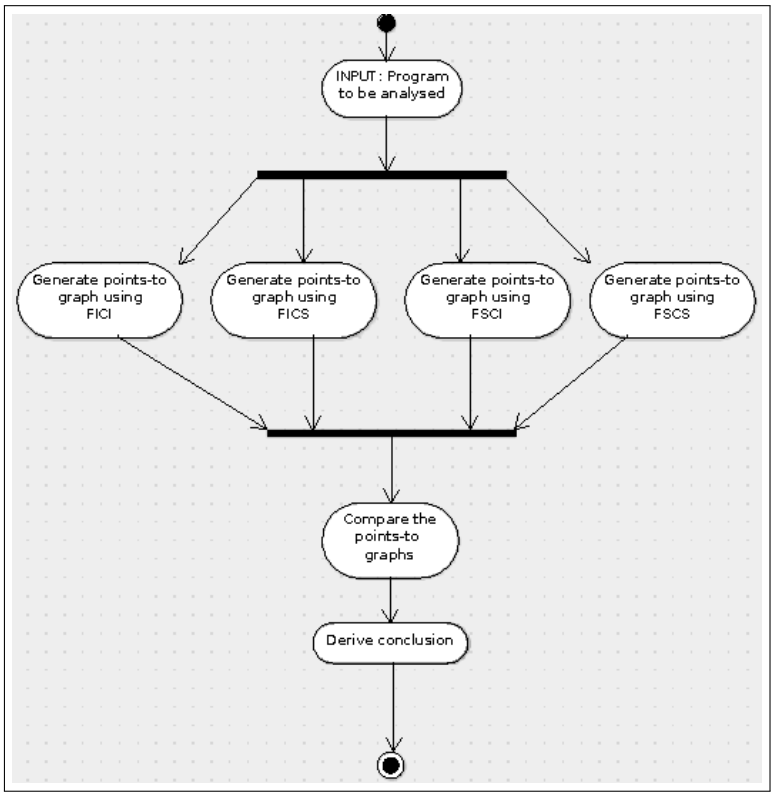


Figure B.3: Activity diagram

Block Diagram

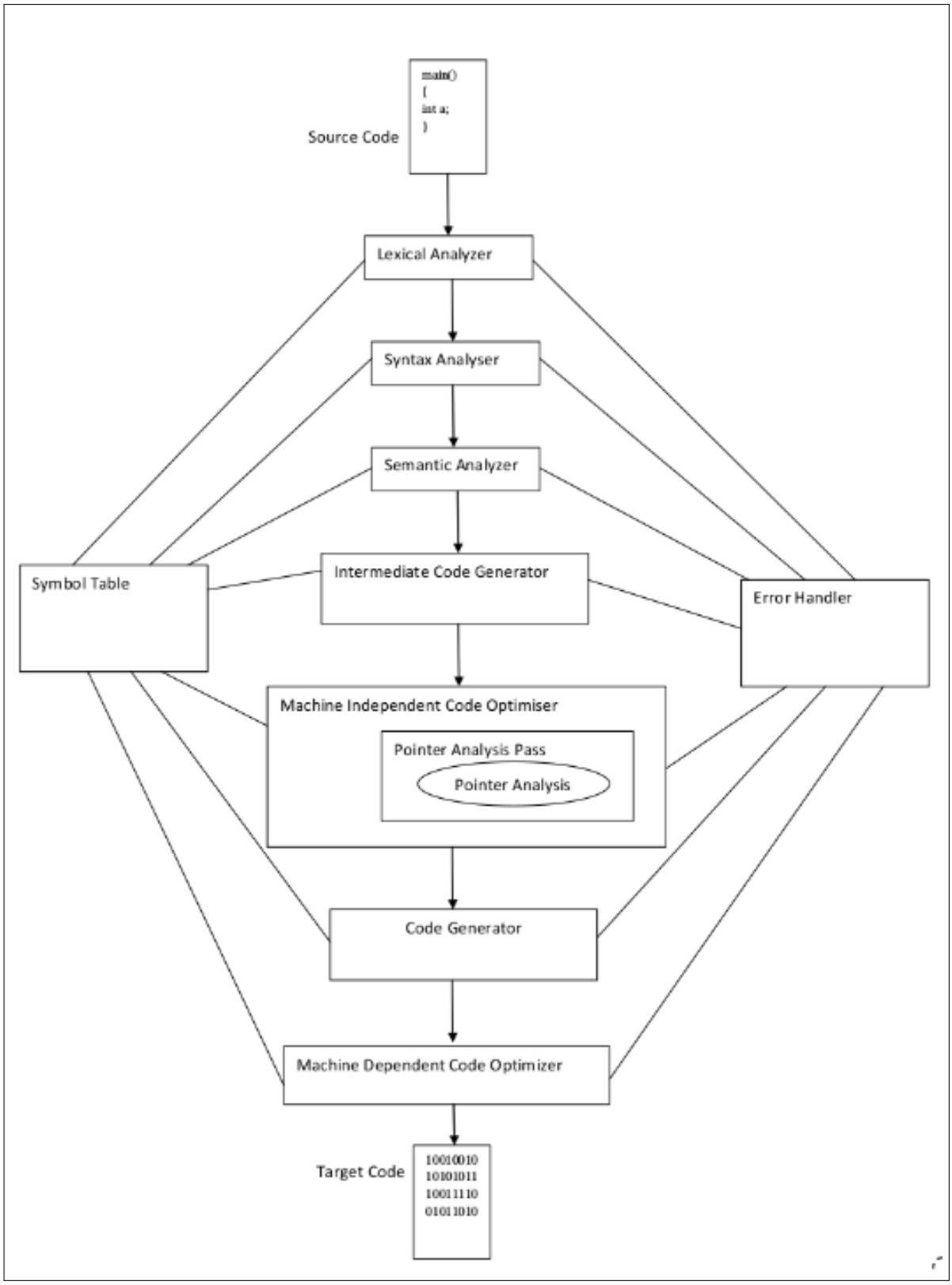


Figure B.4: Block diagram

Assignment 5

Testing of Project problem statement using generated test data (using Mathematical Models, GUI, Function testing Principles if any) selection and appropriate use of testing tools.

Test ID	Test Case	Expected Results	Observed Results	Pass/Fail
1	The path of the input file containing the program to be analysed is specified	The file exists at the mentioned path destination and the input program to be analysed is read successfully	-	-
2	Pointee-set formed as a result of FICI analysis is given as input	Points-to graph generated correctly	-	-
3	Pointee-set formed as a result of FICS analysis is given as input	Points-to graph generated correctly	-	-
4	Pointee-set formed as a result of FSCS analysis is given as input	Points-to graph generated correctly	-	-
5	Pointee-set formed as a result of FSCI analysis is given as input	Points-to graph generated correctly	-	-
6	Pointer Statements other than x=*y, x=&y, x=y, *x=y, *x=&y and *x=*y are given	Analysis program ignores them and increments the failed constraints counter	-	-
7	Programs with pointer arithmetic statements are given as input	Analysis Program ignores them	-	-

ANNEXURE C

PROJECT PLANNER

We have used Gantt chart to illustrate the start and end dates of all our project phases shown in Table 5.1.

Online tool used : SmartSheet

The Gantt chart for our project is as follows:

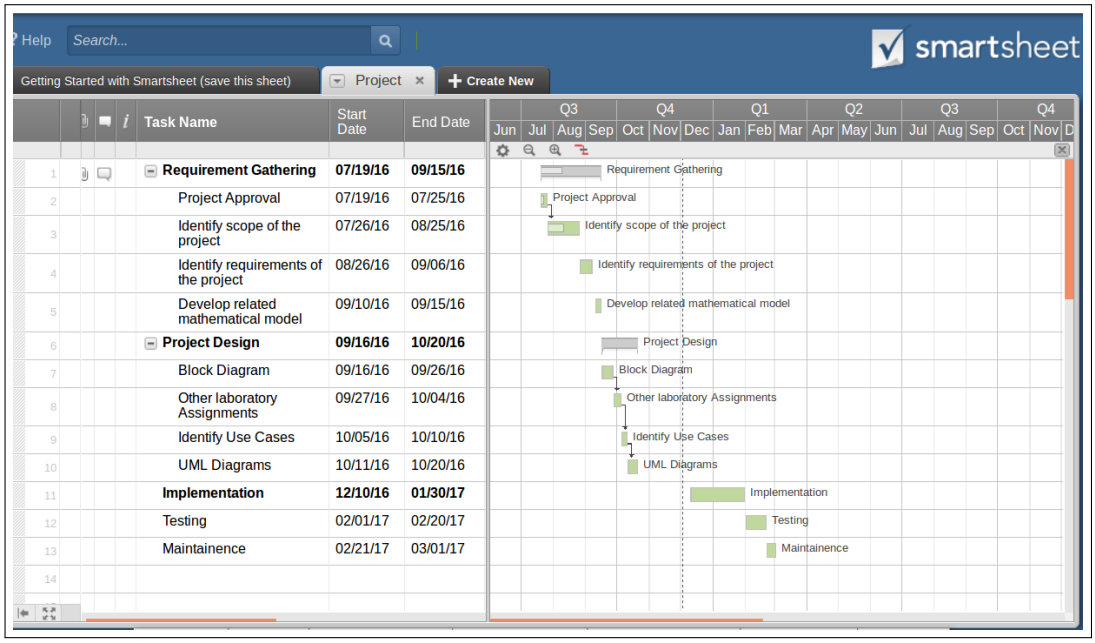


Figure C.1: Gantt Chart

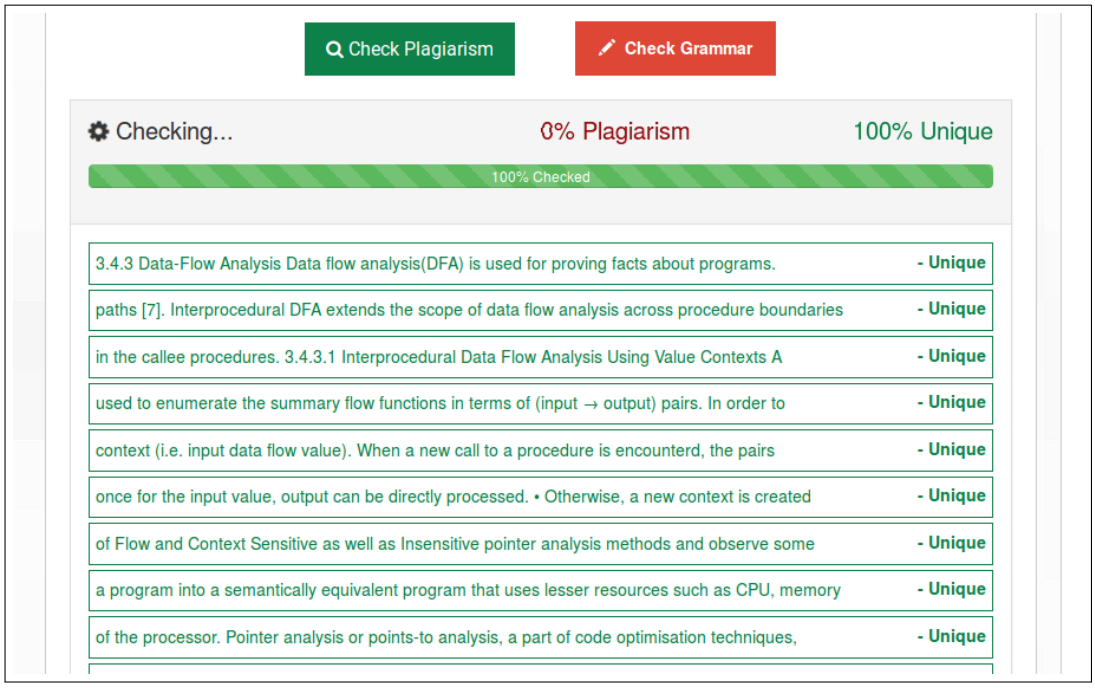
ANNEXURE D

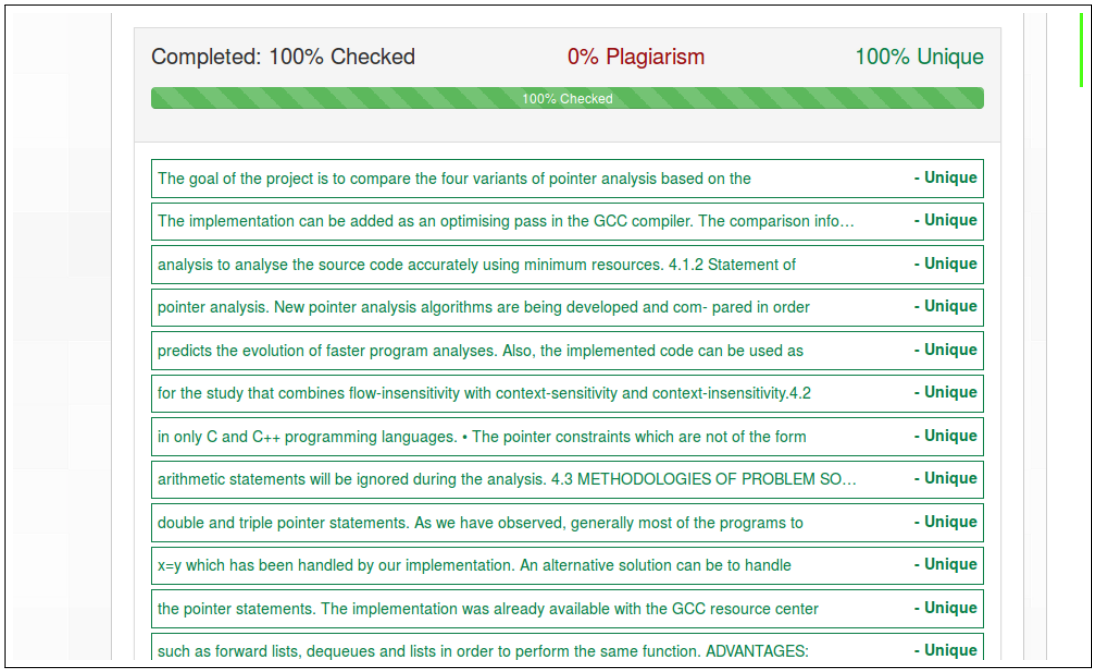
PLAGIARISM REPORT

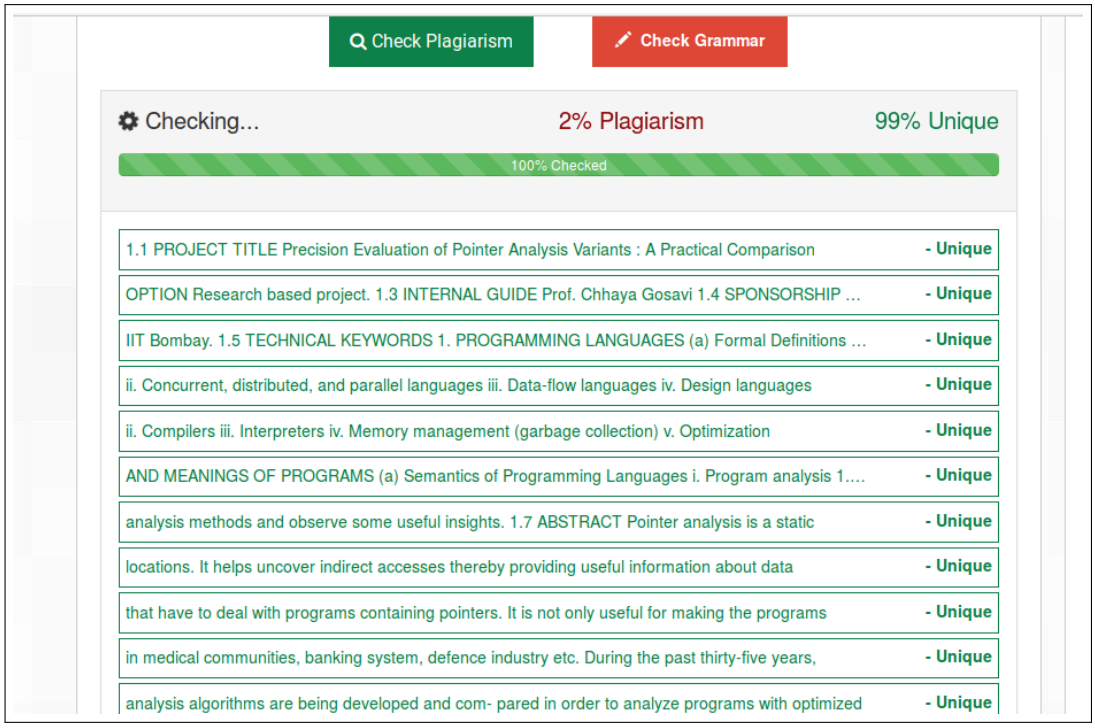
This Project Group has successfully completed their Report on

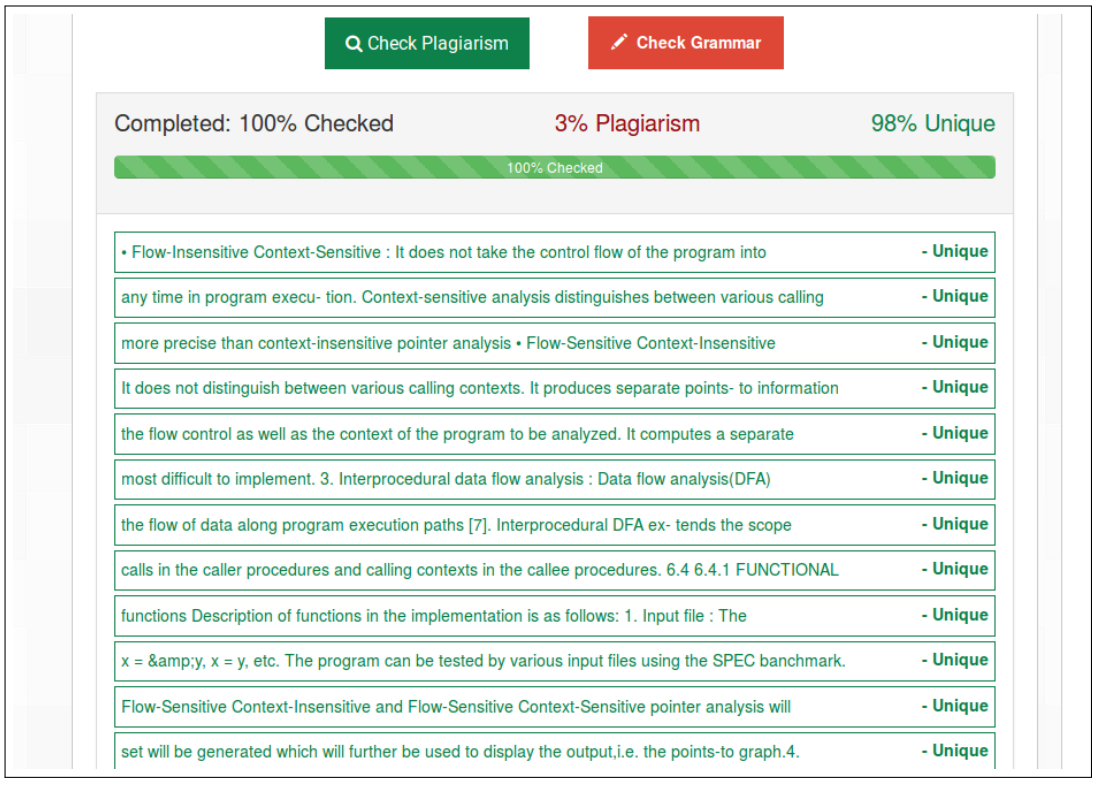
"PRECISION EVALUATION OF POINTER ANALYSIS VARIANTS"

and the report is found to be **98 %** plagiarism free as per smallseotools.com.






Check Plagiarism
Check Grammar
Checking...
2% Plagiarism
99% Unique
100% Checked
1.1 PROJECT TITLE Precision Evaluation of Pointer Analysis Variants : A Practical Comparison - Unique
OPTION Research based project. 1.3 INTERNAL GUIDE Prof. Chhaya Gosavi 1.4 SPONSORSHIP ... - Unique
IIT Bombay. 1.5 TECHNICAL KEYWORDS 1. PROGRAMMING LANGUAGES (a) Formal Definitions ... - Unique
ii. Concurrent, distributed, and parallel languages iii. Data-flow languages iv. Design languages - Unique
ii. Compilers iii. Interpreters iv. Memory management (garbage collection) v. Optimization - Unique
AND MEANINGS OF PROGRAMS (a) Semantics of Programming Languages i. Program analysis 1.... - Unique
analysis methods and observe some useful insights. 1.7 ABSTRACT Pointer analysis is a static - Unique
locations. It helps uncover indirect accesses thereby providing useful information about data - Unique
that have to deal with programs containing pointers. It is not only useful for making the programs - Unique
in medical communities, banking system, defence industry etc. During the past thirty-five years, - Unique
analysis algorithms are being developed and com- pared in order to analyze programs with optimized - Unique


Check Plagiarism
Check Grammar
Completed: 100% Checked
3% Plagiarism
98% Unique
100% Checked
• Flow-Insensitive Context-Sensitive : It does not take the control flow of the program into - Unique
any time in program execu- tion. Context-sensitive analysis distinguishes between various calling - Unique
more precise than context-insensitive pointer analysis • Flow-Sensitive Context-Insensitive - Unique
It does not distinguish between various calling contexts. It produces separate points- to information - Unique
the flow control as well as the context of the program to be analyzed. It computes a separate - Unique
most difficult to implement. 3. Interprocedural data flow analysis : Data flow analysis(DFA) - Unique
the flow of data along program execution paths [7]. Interprocedural DFA ex- tends the scope - Unique
calls in the caller procedures and calling contexts in the callee procedures. 6.4 6.4.1 FUNCTIONAL - Unique
functions Description of functions in the implementation is as follows: 1. Input file : The - Unique
x = &y, x = y, etc. The program can be tested by various input files using the SPEC banchmark. - Unique
Flow-Sensitive Context-Insensitive and Flow-Sensitive Context-Sensitive pointer analysis will - Unique
set will be generated which will further be used to display the output,i.e. the points-to graph.4. - Unique