**MCFromC Project Overview**

# Glossary

* AST – Abstract Syntax Tree – a tree data structure that contains the given source code as a syntax tree.
* CFG – Control Flow Graph
* MC – Monotonicity Constraints
* ACFG – Annotated Control Flow Graph is CFG that contains MCs on its edges.

# General Architecture Overview

The project architecture is based on stages where each stage takes an input, processes it, and outputs it as input to the next stage. **Figure 1** below shows a high level overview of the data flow between the system stages.

Figure 1 - System Stages

Output

CFG

CFG

AST

C Source

Syntax Parser

CFG Creation

CFG Simplifier

MC Creation

## Syntax Parser

This stage is responsible for converting a text file containing a subset of the  
**C programming language**[[1]](#footnote-1) into an AST that which will be processed into a CFG in the next stage. This stage is implemented using the following 3rd party tools:

* **Flex** – This tool uses **lex** file and produces source file that holds function that takes input stream and returns tokens according to the **lex** file.
* **Bison** – This tool is implementation for **YACC**. The tool receives a **YACC** input file (language grammar in a format similar to BNF, where each rule contains C code snippet) and generates a C source file which is the code of a parser for the given language.

Since the above tools produce a source files in the C language (and not C++), while the system main algorithms are coded in C++, this stage is implemented as a separate DLL with an interface which takes a file name and returns a pointer to the root node of the AST.

## CFG Creation

This stage is responsible for generating a CFG based on the AST input. The output is a CFG and all the program state variable names. The graph data structure implementation is based on a 3rd party library named **boost graph library** (**BGL**). The library is a C++ library, implemented in templates and contains only header files. This stage is implemented in a separate, C++ written, DLL named CFGGen. The algorithm for CFG generation based on the AST of the input program will be covered in a later section.

## CFG Simplifier

The CFG simplifier is a small stage, implemented in the same DLL as the previous stage (CFG Creation) and is responsible to removal of redundant flow points from the CFG. One example for such redundancy is appearance of consecutive expression flow points (basic block). Those flow points can be reduced into a single flow point since there are no branches within the block. The current stage aggregates those flow points into a single flow point that which represents and holds the information of all other expression flow points.

## MC Creation

This MC Creation layer is responsible for trying to evaluate changes in the values of the program state variables in each abstract transition of the program (represented as an edge in the CFG). The input for this stage is a CFG (simplified from the previous stage) and the names of all state variables (found by traversing the AST, during the CFG generation). Abstract Interpretation is used in order to track changes in the values of the program state variables. This stage will be covered in detail in **MC generation** section.

The following chart describes the normal flow of the application with regard to code projects that are involved.

Figure 2 - Code Project Flow

# User Manual

## Installation

## Usage

# Algorithms

## converting AST to CFG

## MC generation

### Description of how expressions are evaluated

### Algorithm

# Supported grammar (in BNF format)

# Benchmark programs and result graphs and MCs.

1. See **Supported grammar (in BNF format)** section for the exact language definition [↑](#footnote-ref-1)