CS 5615 - Introduction to Program Analysis
Spring 2018-2019
Lahore University of Managaement Sciences

Dataflow Analysis

17th March 2019

Mohammad Maaz 20100068

Ali Ahad 20100284

OVERVIEW

Dataflow analyses have very interesting commonalities such as transfer functions, meet operators and very similar iterative algorithms. This project serves to highlight that it is possible to write a generic framework for these analyses after parametrizing the aforementioned commonalities and defining the specific program components and analysis directions.

GOALS

1. Implement an iterative dataflow analysis framework in LLVM for solving bit-vector dataflow problems.
2. Apply this framework for the following analyses:
    a. Forward Analysis of Available Expressions.
    b. Backward Analysis of Live Variables.

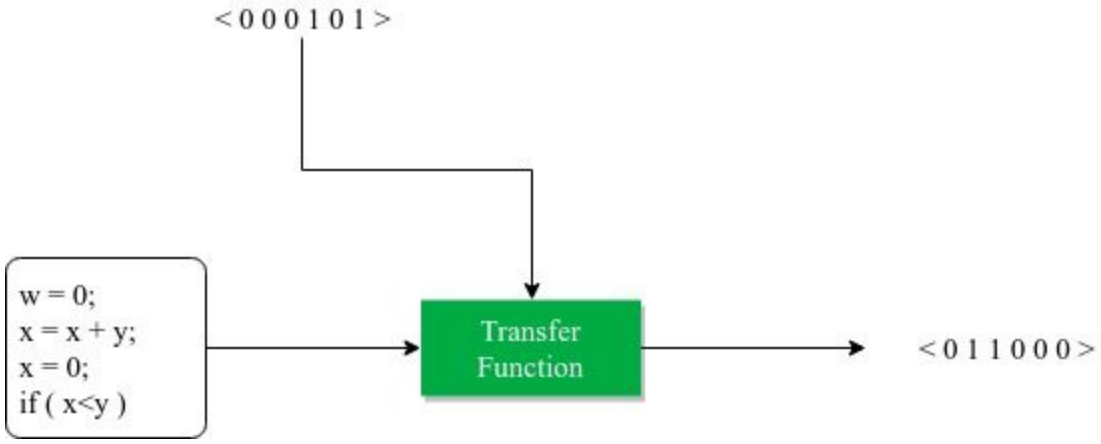
IMPLEMENTATION

Framework

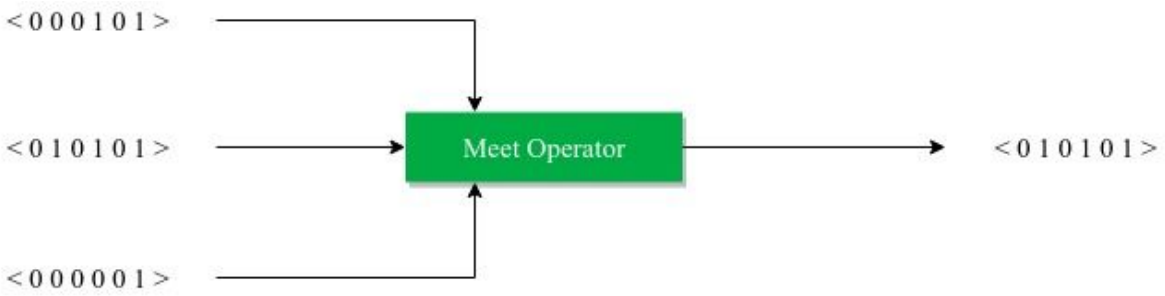
The framework was initialized using the following parameters.

1. Function to run analysis on
2. Type domain to analyze
3. Transfer function
4. Meet operator

A transfer function is a function that takes an input domain set and basic block and generates an output domain set based on the instructions in the basic block.

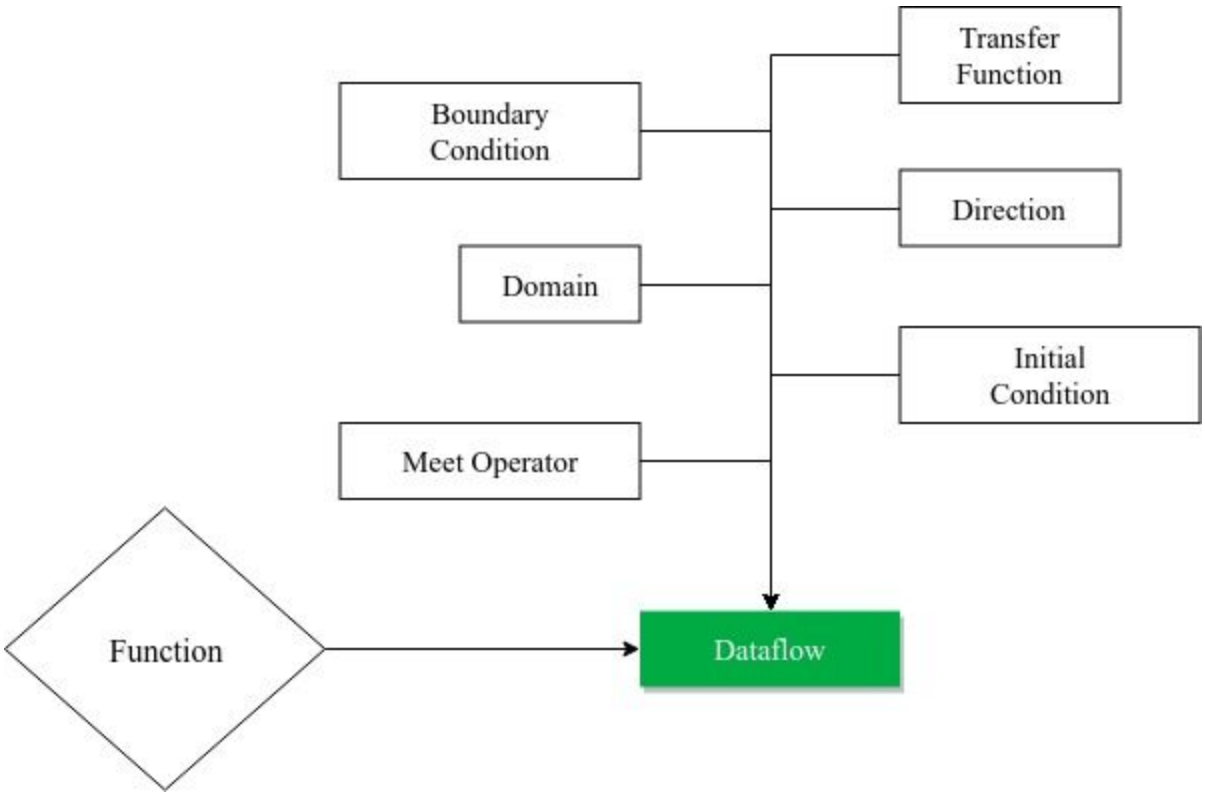


The meet operator takes a set of domain sets and returns a single domain set after applying a set operator.




```
//Constructor
Framework(
    Function &F,
    BitVector init,
    bool dir,
    BitVector(*meet_function)(std::vector<BitVector> v),
    BitVector (*transfer_function)(
        BitVector input,
        std::vector<void*> domain,
        BasicBlock *ptr,
        std::map<BasicBlock*, block> &state));
```

The core of this framework was the iterative algorithm that takes the input function and iterates over its basic blocks. It then applies the corresponding meet operators and transfer functions during the iterative process.

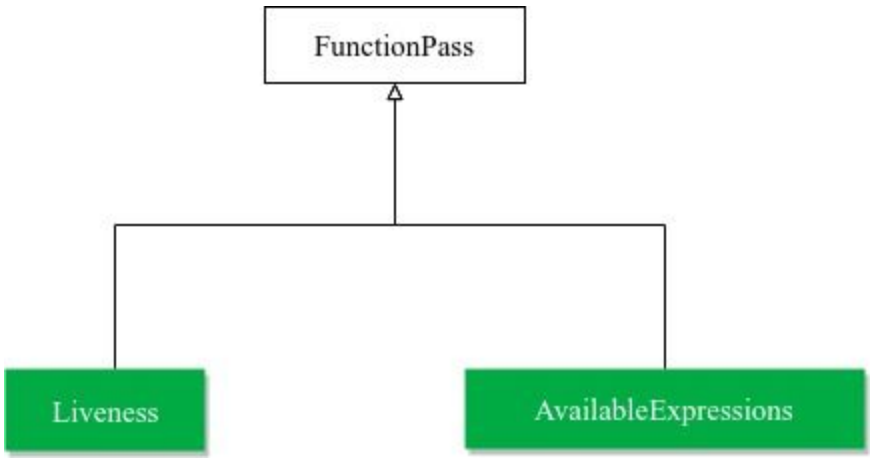


The state of every block during the iterations was maintained using a map that maps every basic block to an input and output.

```
struct blockState
{
    BitVector IN;
    BitVector OUT;
};
```

```
std::map<BasicBlock*, blockState> state;
```

Classes for Available Expressions and Liveness inherits from the LLVM FunctionPass superclass.



Available Expressions

Domain: Set of expressions

Direction: Forward

Meet Operator: Intersection

Transfer function: OUT(b) = Gen[b] U (IN[b] -Kill[b])

Liveness

Domain: Set of variables

Direction: Backward

Meet Operator: Union

Transfer function: IN[b] = Use[b] U (OUT[b]-Def[[b])

TESTING

Commands

make

opt-6.0 -load ./available.so -available available-test-m2r.bc -o /dev/null

opt-6.0 -load ./liveness.so -liveness liveness-test-m2r.bc -o /dev/null

Specifications

The project was tested in an environment containing LLVM 6.0 and Clang 6.0.