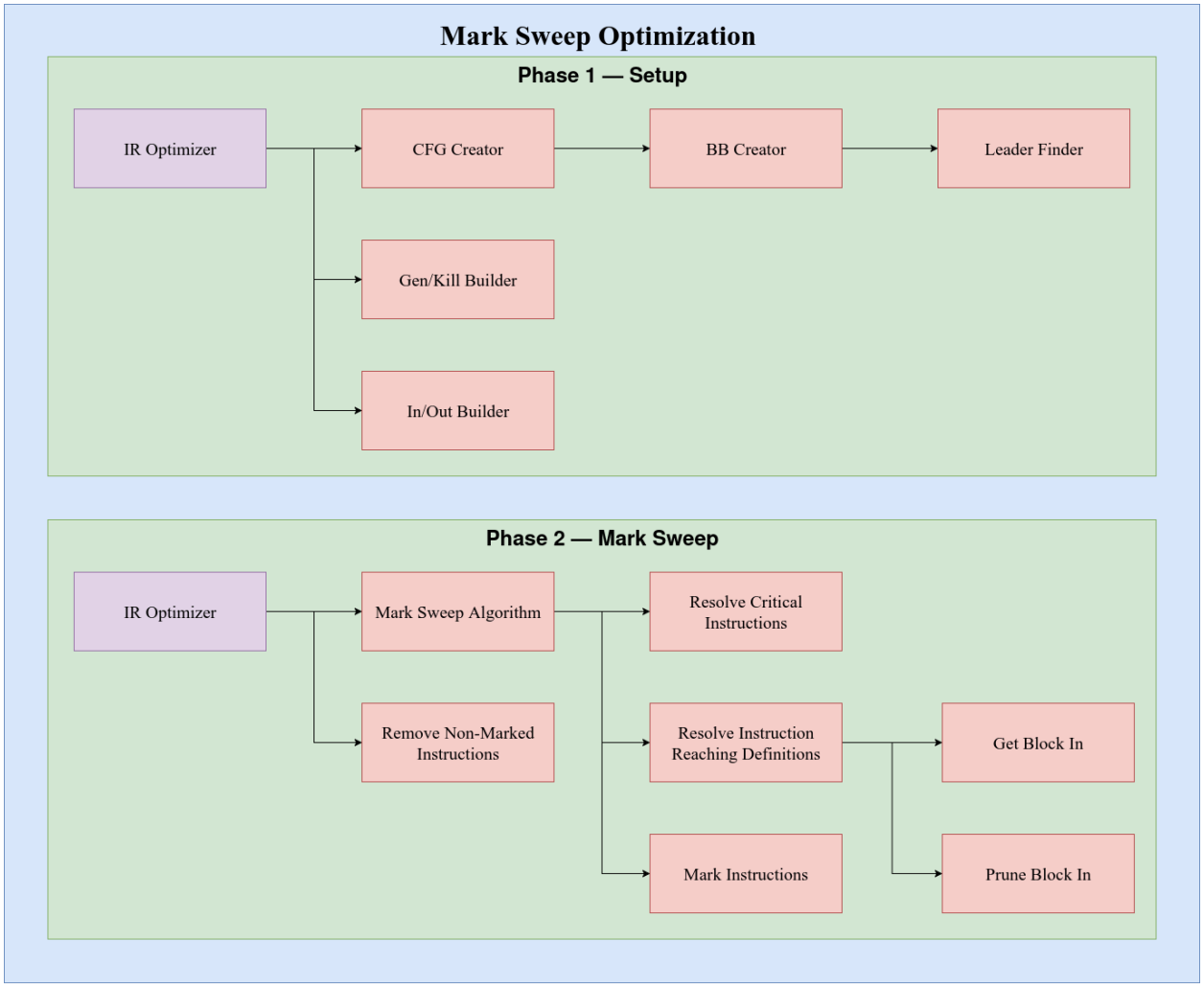
Vishwanath & Uz

CS 4240 Compilers and Interpreters Project 1 Design Doc

High-Level Architecture



- IR Optimizer is marked purple to highlight that it is repeated and that it is the orchestrator of the entire optimization algorithm.
- Implemented algorithm: mark sweep with reaching definitions.
- Reasons for choosing the approach in the diagram: elegant and modular design.
    - Each arrow indicates what calls what (e.g. CFG Creator calls BB Creator, which calls Leader Marker).
    - Each block only does one thing and relies on function calls to accomplish more low level tasks for it. For instance, the Mark Sweep Algorithm box does not care about the nitty gritty details of the reaching definition resolver. It only cares that the call to the reaching definition resolver provides the reaching definitions for a particular instruction.

Low-Level Design Decisions

- Optimizations are done on a function-by-function basis.
    - Since each call and callr is a critical instruction, there is no need to perform interfunction optimizations. Even if the parameter *y* in function call ‘call x, y’ reaches some critical use in function x, we do not need to take this into account as call itself is a critical instruction so any reaching definition to this call will be marked anyway.
- Labels are critical instructions.
    - Since the provided input to the middle-end comes from the front-end, we made the assumption that the front end would produce a reasonable output. This means that the IR produced by the front end would not have unreachable labels.
    - If not a single instruction in the label is marked, we still keep the label as it could be the target of the branch instruction and we are not performing branch optimization.
- We used Java as the implementation language
    - Thanks to its object-oriented nature and the provided helper files, it made sense to use Java as the language of implementation.
- Each instruction points to the block it is in
    - Allows for easy retrieval of the IN set to the BB for any given instruction which is later used to compute the reaching definitions to that instruction.

Software Engineering Challenges

- Kotlin + Java integration.
    - We tried writing the project partially in Kotlin and partially in Java but found out this would be harder to build than we thought so we reverted to only Java.
- Method organization
    - It was challenging to decide which class to put each method under. For instance, buildGen() is under the class Block whereas the method buildKill() is under the class Definitions, even though intuitively one would think they should be under the same class.
        - This separation is because gen can be done on a block by block basis, but kill needs the entire function.
    - We ended up creating two new classes:
        - Block — a class to represent blocks
        - Definition — a helper class to assist in reaching definition computations

Building and Usage

- Building: run `$ ./build.sh` in the root project directory.
- Usage: run `$ ./run.sh <Input IR file> <Output IR file>` in the root project directory.

Test Result Summary

Input	Output

`#start_function`
`int fib(int n):`
`int-list: t1, t2, x, r`
`float-list:`
`  assign, r, 1`
`  brgt, if_label0, n, 1`
`  assign, r, n`
`  goto, end`
`if_label0:`
`  sub, n, n, 1`
`  callr, t1, fib, n`
`  sub, x, n, 1`
`  sub, n, n, 1`
`  callr, t2, fib, x`
`  add, r, t1, t2`
`end:`
`  return, r`
`#end_function`
 	
`#start_function`
`void main():`
`int-list: x, y, z`
`float-list:`
`  callr, x, geti`
`  callr, z, fib, x`
`  assign, x, z`
`  call, puti, x`
`  call, putc, 10`
`#end_function`
	`#start_function`
	`int fib(int n):`
	`int-list: t1, t2, x, r`
	`float-list:`
	`  ~~assign, r, 1~~`
	`  brgt, if_label0, n, 1`
	`  assign, r, n`
	`  goto, end`
	`if_label0:`
	`  sub, n, n, 1`
	`  callr, t1, fib, n`
	`  sub, x, n, 1`
	`  ~~sub, n, n, 1~~`
	`  callr, t2, fib, x`
	`  add, r, t1, t2`
	`end:`
	`  return, r`
	`#end_function`
	
	`#start_function`
	`void main():`
	`int-list: x, y, z`
	`float-list:`
	`  callr, x, geti`
	`  callr, z, fib, x`
	`  assign, x, z`
	`  call, puti, x`
	`  call, putc, 10`
	`#end_function`