Daniel Cao
Andy Cao

Project 1 Pseudocode

Algorithm Calculate
Given a simple instruction and an instruction from the block of n instructions

Pass the simple instruction and one instruction from the block of n instructions into the algorithm.

```
Bool algorithm_calculate(simple, block) {
        // Keep track of empty sets, each 0 represents an empty set
        Initialize an array of 3 elements called result, where all elements are initialized to 0

        // We begin checking for intersections between the two instructions
        // OUT( I 1) ∩ IN(Block instruction)
        For each char in the block instruction:
                // We check to make sure both chars are alphabets and not operators or whitespace
                If block char is an alphabet char and output of i1 is an alphabet char:
                        // we have an intersection
                        If the output of the simple instruction == an alphabet character in the block instruction:
                                // change the first element in result to 1 to indicate no empty set
                                Result[0] = 1;
                                Break;

        // IN( I 1 ) ∩ OUT(Block instruction)
        For each char in the simple instruction:
                If simple char is an alphabet char and output out block instruction is an alphabet char:
                        If the output of the simple instruction == an alphabet character in the block instruction:
                                // change the second element in result to 1 to indicate no empty set
                                Result[1] = 1;
                                Break;

        // OUT( I 1 ) ∩ OUT ( Block instruction )
        If the output of I 1 == output of the block instruction:
                If they are both alphabet chars:
                        // change the third element in result to 1 to indicate no empty set
                        Result[2] = 1;

        For each element in result:
                // if result contains all empty sets, we return true
                // which means we have two instructions that can run in parallel
                // otherwise, we return false
                If an element in result == 1:
                        Return false;

        Return true;
}
```

```
// Pass in two different instructions from the block of n instructions

Bool algorithm_verify(block1, block2) {
        // Keep track of empty sets, each 0 represents an empty set
        Initialize an array of 3 elements called result, where all elements are initialized to 0

        // We begin checking for intersections between the two instructions
        // OUT( block1 ) ∩ IN(Block2)
        For each char in the block2 instruction:
                If Block2 char is an alphabet char and output of block1 is an alphabet char:
                        // we have an intersection
                        If the output of the block1 instruction == a character in the block2 instruction:
                                // change the first element in result to 1 to indicate no empty set
                                Result[0] = 1;
                                Break;

        // IN( block1 ) ∩ OUT(Block2)
        For each char in the block1 instruction:
                If block1 char is an alphabet char and output of Block2 instruction is an alphabet char:
                        If the output of the Block2 instruction == an character in the block1 instruction:
                                // change the second element in result to 1 to indicate no empty set
                                Result[1] = 1;
                                Break;

        // OUT( block1) ∩ OUT ( Block2)
        If the output of block1 instruction == output of the Block2 instruction:
                If they are both alphabet chars:
                        // change the third element in result to 1 to indicate no empty set
                        Result[2] = 1;

        For each element in result:
                // if result contains all empty sets, we return true
                // which means we have two instructions that can run in parallel
                // otherwise, we return false
                If an element in result == 1:
                        Return false;

        Return true;
}
```

Summary:

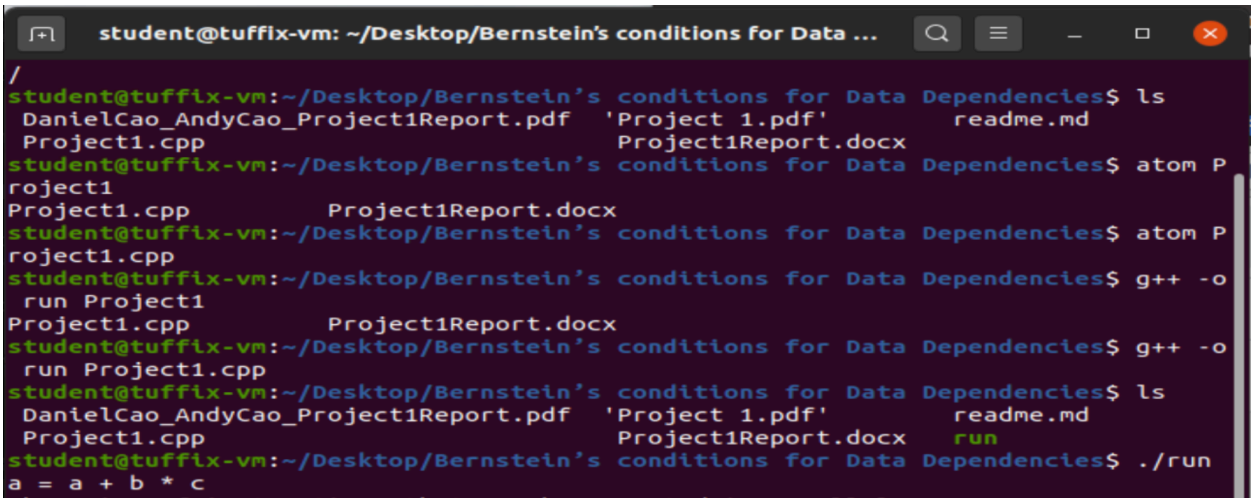
These algorithm are implemented using C++ programming language and mainly focus on checking whether the instructions of a sequential program can be parallelized or not. The first algorithm is given one instruction and determine whether if there is an instruction parallel with the given instruction. The second algorithm checks all the pairs of instructions can be executed in parallel. So in the algorithm, we would iterate all the instructions and check in three cases: OUT(block1) ∩ IN(Block2), IN(block1) ∩ OUT(Block2) and OUT(block1) ∩ OUT (Block2).

Group Members:

Team Members:

- Andy Cao: dongyicao123@csu.fullerton.edu
- Daniel Cao: dcao182@csu.fullerton.edu

ScreenShots for Algorithm Calculate



ScreenShots for Algorithm Verify: