Project #1: Data Dependencies

Jason Duong

October 14, 2022

1 Summary

The report is organized into three sections, Summary, Pseudocode, and Screenshots. The summary section provides the specifications and overview of this document. The pseudocode section provides a high-level description of two algorithms, namely, $CALCULATE$ and $VALIDATE$. The inputs for both algorithms are a list of instructions encoded as a string. For each algorithm, we use the following notational conventions. Boldface text represents the data type for a variable following the keyword, such as "**Set** *result*". Whenever an undefined variable is declared without a specified data type, its data type is inferred by the right-hand side expression of the assignment. Assignment expressions, $\langle token \rangle = \langle expression \rangle$ are denoted using left-arrow symbols, $\leftarrow$. Element index operations on some iterable container are preceded by subscripts. All algorithms were implemented in the Python programming language, and some pseudocode instructions were abstracted to suit the "pythonic" coding style. Lastly, the screenshot section shows three snapshots of group members and the result of two executions with $N = 3$ and $N = 5$.

2 Pseudocode

Algorithm 1 CALCULATE

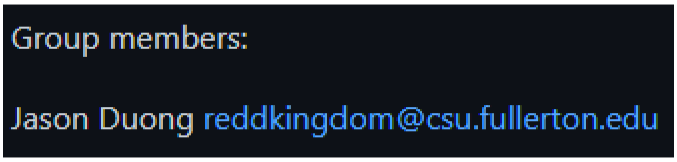
```
    Input: instr, block
    Output: result
Set $result \leftarrow \emptyset$
procedure INPUT($x$)
    List$\langle char \rangle$ $rhs \leftarrow$ extract all characters after the $\langle = \rangle$ token from input instruction $x$
    $vars \leftarrow \{rhs_i \mid rhs_i \in \{a, \dots, z\},\ i \in \{0, \dots, |rhs| - 1\}\}$
    return $vars$
end procedure
procedure OUTPUT($y$)
    List$\langle char \rangle$ $lhs \leftarrow$ extract all characters before the $\langle = \rangle$ token from input instruction $y$
    $vars \leftarrow \{lhs_i \mid lhs_i \in \{a, \dots, z\},\ i \in \{0, \dots, |lhs| - 1\}\}$
    return $vars$
end procedure
for $i \leftarrow 1$ to $|block|$ do
    if
        $input(instr) \cap\ output(block_i)\ = \emptyset \wedge$
        $output(block_i) \cap\ input(instr)\ = \emptyset \wedge$
        $output(instr) \cap\ output(block_i) = \emptyset$ then
        $result \leftarrow result \cup \{block_i\}$
    end if
end for
if $result = \emptyset$ then
    $return$ "$None$"
else
    $return\ result$
end if
```

Algorithm 2 VALIDATE

```
    Input: block
    Output: result
Set $result \leftarrow \emptyset$
procedure INPUT($x$)
    List$\langle char \rangle$ $rhs \leftarrow$ extract all characters after the $\langle = \rangle$ token from input instruction $x$
    $vars \leftarrow \{rhs_i \mid rhs_i \in \{a, \ldots, z\},\ i \in \{0, \ldots, |rhs| - 1\}\}$
    return $vars$
end procedure
procedure OUTPUT($y$)
    List$\langle char \rangle$ $lhs \leftarrow$ extract all characters before the $\langle = \rangle$ token from input instruction $y$
    $vars \leftarrow \{lhs_i \mid lhs_i \in \{a, \ldots, z\},\ i \in \{0, \ldots, |lhs| - 1\}\}$
    return $vars$
end procedure
for each $\{i, j\} \in \binom{|block|}{2}$ do
    if
        $input(block_i) \cap\ output(block_j)\ = \emptyset\ \wedge$
        $output(block_i) \cap\ input(block_j)\ = \emptyset\ \wedge$
        $output(block_i) \cap\ output(block_j) = \emptyset$  then
        $result \leftarrow result \cup \{(block_i, block_j)\}$
    end if
end for
if $result = \emptyset$ then
    return "None"
else
    return $result$
end if
```

3 Screenshots

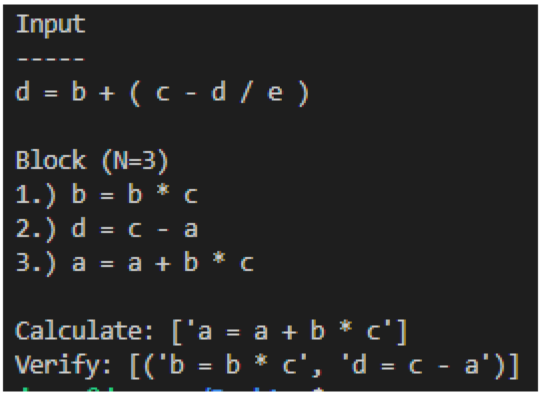
3.1 Group Members



Group members:

Jason Duong reddkingdom@csu.fullerton.edu

3.2 Execution with $N = 3$

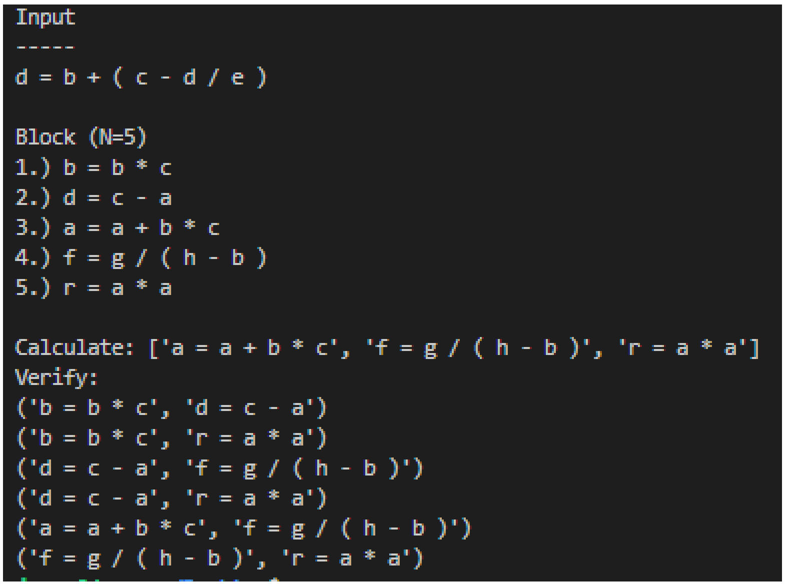


```
Input
-----
d = b + ( c - d / e )

Block (N=3)
1.) b = b * c
2.) d = c - a
3.) a = a + b * c

Calculate: ['a = a + b * c']
Verify: [('b = b * c', 'd = c - a')]
```

3.3 Execution with $N = 5$



```
Input
-----
d = b + ( c - d / e )

Block (N=5)
1.) b = b * c
2.) d = c - a
3.) a = a + b * c
4.) f = g / ( h - b )
5.) r = a * a

Calculate: ['a = a + b * c', 'f = g / ( h - b )', 'r = a * a']
Verify:
('b = b * c', 'd = c - a')
('b = b * c', 'r = a * a')
('d = c - a', 'f = g / ( h - b )')
('d = c - a', 'r = a * a')
('a = a + b * c', 'f = g / ( h - b )')
('f = g / ( h - b )', 'r = a * a')
```