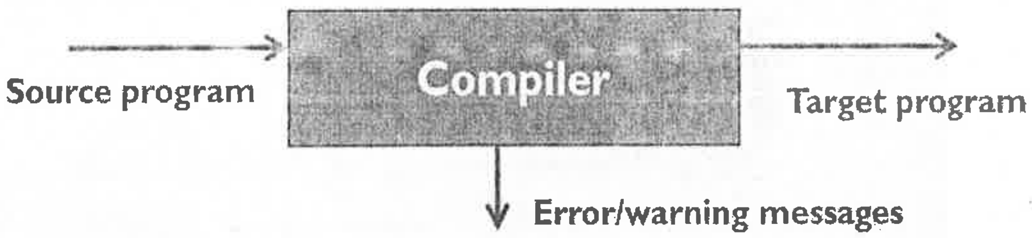
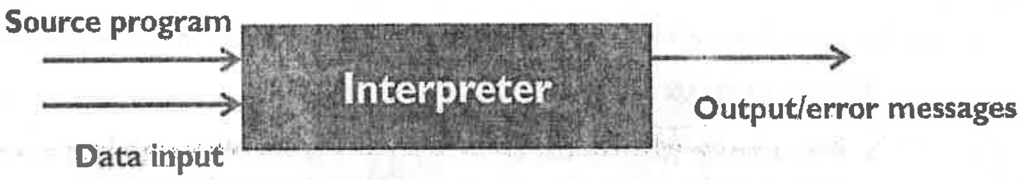
Solver: Liu Xue

Email Address: liux0054@e.ntu.edu.sg

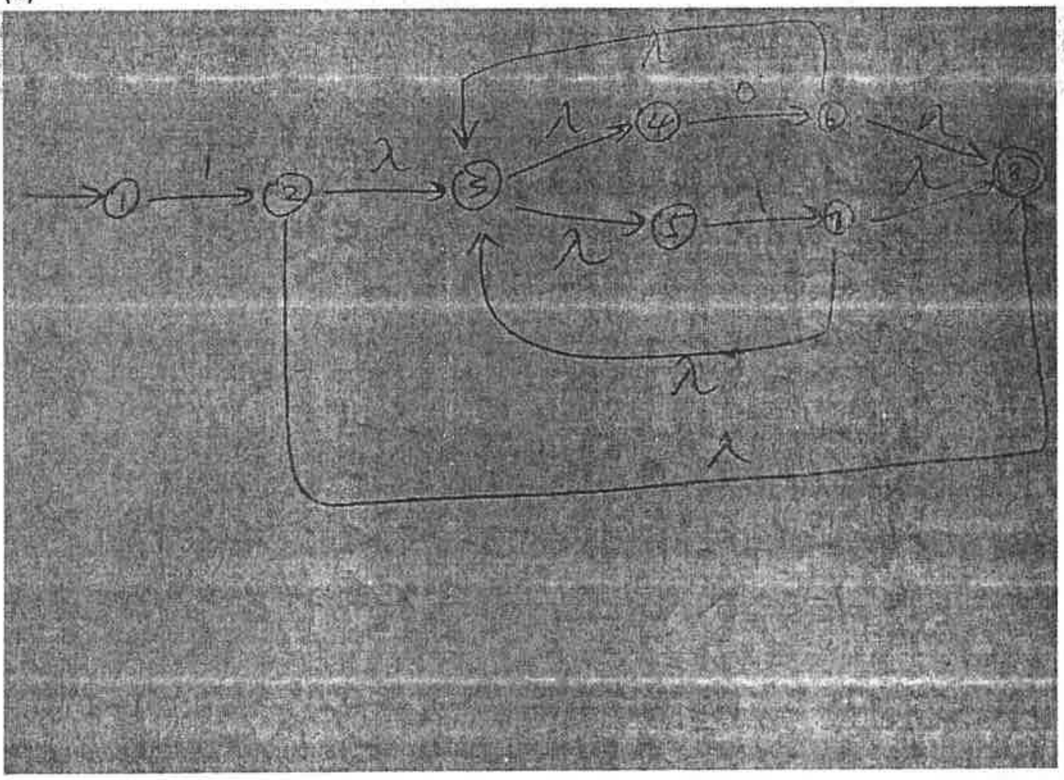
1. (a)
   (i) A compiler is a program that reads a program in one language and translate it into an equivalent program in another language – a translator.



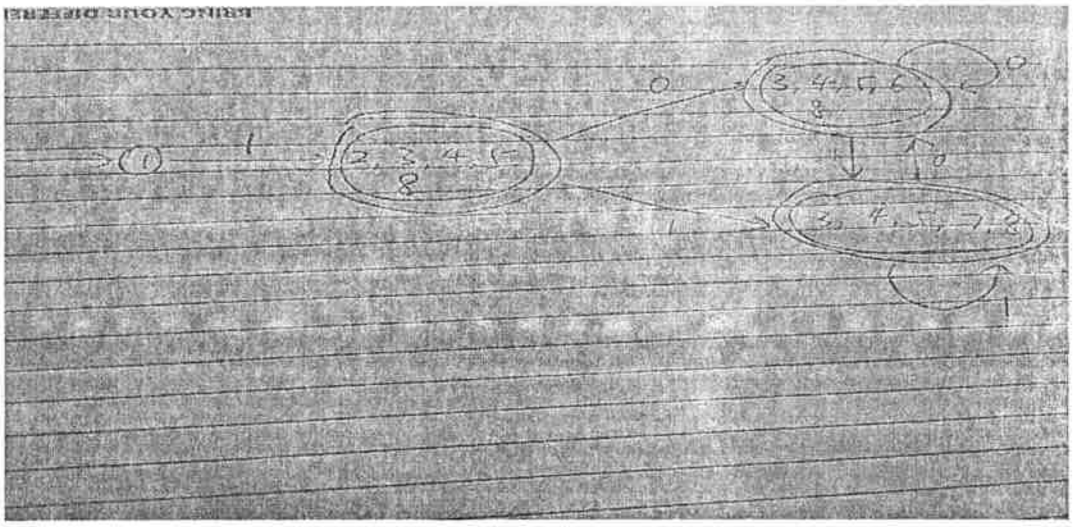
   (ii) An interpreter directly executes the operations specified in the source program on inputs supplied by the user.



(b)

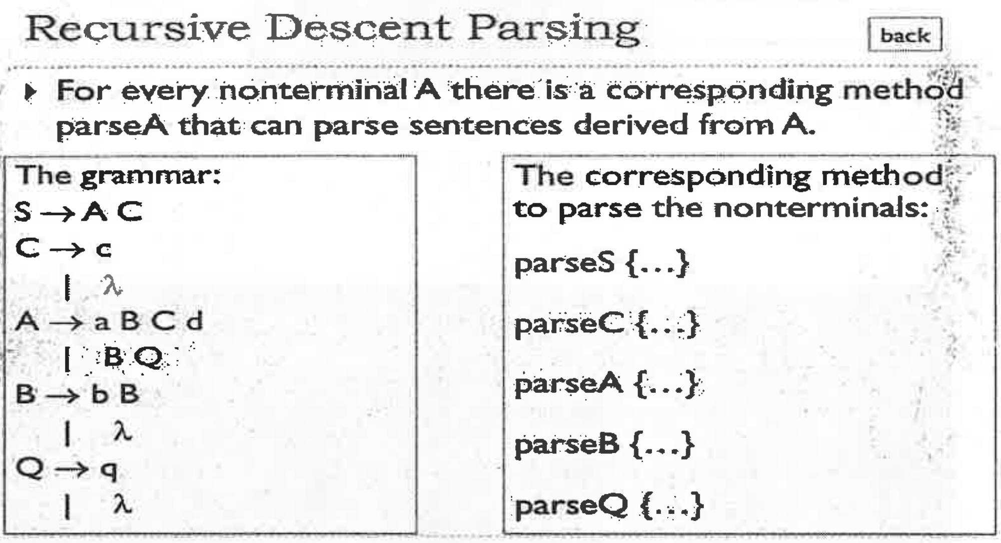


3



(c) (0[1-9] | [1-2][0-9] | 3[0-1]) / (0[1-9] | 1[0-2]) / ([0-9][0-9])

2. (a) Top-down Parsing. Implemented by a collection of mutually recursive procedures.



## Recursive Descent Parsing

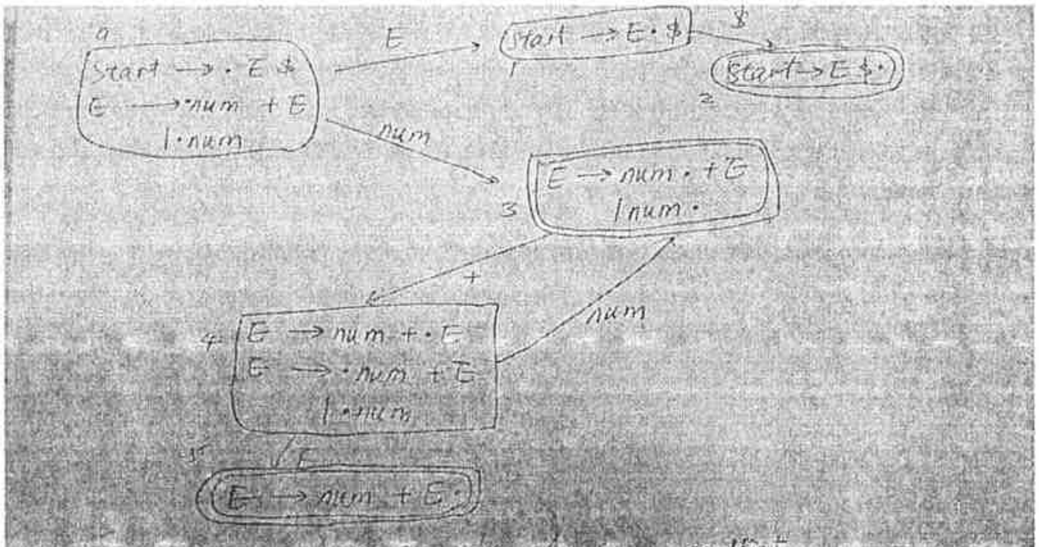
back

- For every nonterminal A there is a corresponding method parseA that can parse sentences derived from A.

| The grammar: | The corresponding method to parse the nonterminals: |
|---|---|
| S → A C | parseS {...} |
| C → c<br>\| λ | parseC {...} |
| A → a B C d<br>\| B Q | parseA {...} |
| B → b B<br>\| λ | parseB {...} |
| Q → q<br>\| λ | parseQ {...} |

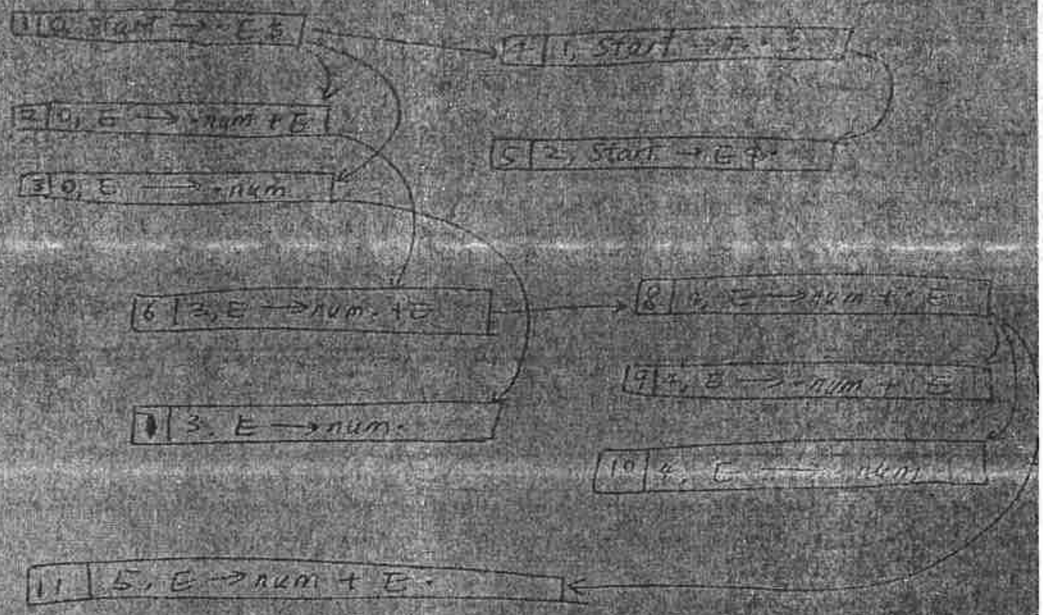
Predict() compute set of tokens for rule p such that:
includes
1. The set of possible tokens that are first produced in some derivation from p: $X \rightarrow X_1X_2...X_m$
2. The set of first tokens in $X_1$
3. If $X_1$may be empty, the set of first tokens n $X_2$ and so on.
4. Those tokens that can follow X in some derivation from $X_1X_2...X_m$
5. If $X_1X_2...X_m$ may be empty, the first tokens that may follow X

(b) (i)



In state 3, there is a shift/reduce conflict.
Shift among E → num · + E or reduce by E → num·

**(ii)**



| State | num | + | $ | Start | E |
|---|---|---|---|---|---|
| 0 | shift 3 | error | error | error | shift 1 |
| 1 | error | error | shift 2 | error | error |
| 2 | Reduce (Start → E $) | | | | |
| 3 | conflict | shift 4 | error | error | error |
| 4 | shift 3 | error | error | error | shift 5 |
| 5 | Reduce (E → num + E ·) | | | | |

3. (a) The goal of name analysis is to determine, for every identifier appearing in the program, which declaration it refers to. Type analysis computes the types of composite expressions from the types of their components. Based on name and type analysis, we can perform semantic error checking. For example, for Java, no two fields in the same class should have the same name.

   It is necessary to detect these errors in semantic analysis. Semantic analysis is the last step before code generation. To prevent errors reported during code generation phase that is too late and too expensive to rectify, it is necessary to detect errors in semantic analysis phase.

   (b) (i)

| |
|---|
| Abstract BExpr; |
| Abstract BTerm : BExpr; |
| Abstract BFactor : BTerm; |
| ANDBTerm : BTerm ::= Left:BTerm  Right:BFactor |
| ORBExpr : BExpr ::= Left: BExpr Right: BTerm; |
| XORBExpr : BExpr ::=  Left: BExpr Right: BTerm; |
| Zero : BFactor ::= <Value: Boolean> |
| One : BFactor ::= <Value: Boolean> |
| COMBFactor ::= child:BFactor |

   (ii) Synthesized attribute. The value of attribute bitValue depends on other attribute of its own and bitValue attribute of its children.

4. (a)

| | |
|---|---|
| 1. | Only need to generate code for one platform: the virtual machine. |
| 2. | Native instruction sets emphasise efficient execution over ease of compilation, so code generation is difficult |
| 3. | Virtual machine instruction sets and runtime environments are often more high-level than their native counterparts: e.g. both the JVM and the CLR offer automated garbage collection |
| 4. | Virtual machine optimisations benefit every compiler targeting it |
| 5. | Bytecode is often more compact than native code: particularly suitable for resource-constrained platforms (e.g. mobile apps on Android phones) |

   (b) (i) $s1 = x * y;
   $s2 = x +2;
   $s3 = $s2*z;
   s= $s1-$s3;

   (ii)

| | net stack height variation (nshv) | minimum stack height variation (mshv) |
|---|---|---|
| 1: load x; | +1 | +1 |

| | | |
|---|---|---|
| 2: load y; | +2 | +1 |
| 3: mul; | +1 | +1 |
| ~~4: store $s1;~~ | 0 | 0 |
| 5: load x; | +1 | 0 |
| 6: load 2; | +2 | 0 |
| 7: add; | +1 | 0 |
| ~~8: store $s2;~~ | | |
| ~~9: load $s2;~~ | | |
| 10: load z; | +2 | 0 |
| 11: mul; | +1 | 0 |
| ~~12: store $s3;~~ | 0 | 0 |
| ~~13: load $s1;~~ | | |
| ~~14: load $s3;~~ | | |
| 15: sub; | | |
| 16: store s; | | |

First eliminate instruction 8 and 9, then according to nshv and mshv, elimate instruction 4 and 13. Then instruction 12 and 14 become consecutive load and store instruction for $s3. Hence instruction 12 and 14 are eliminated. Hence after eliminate all redundant load and store instruction. The instruction set will be:

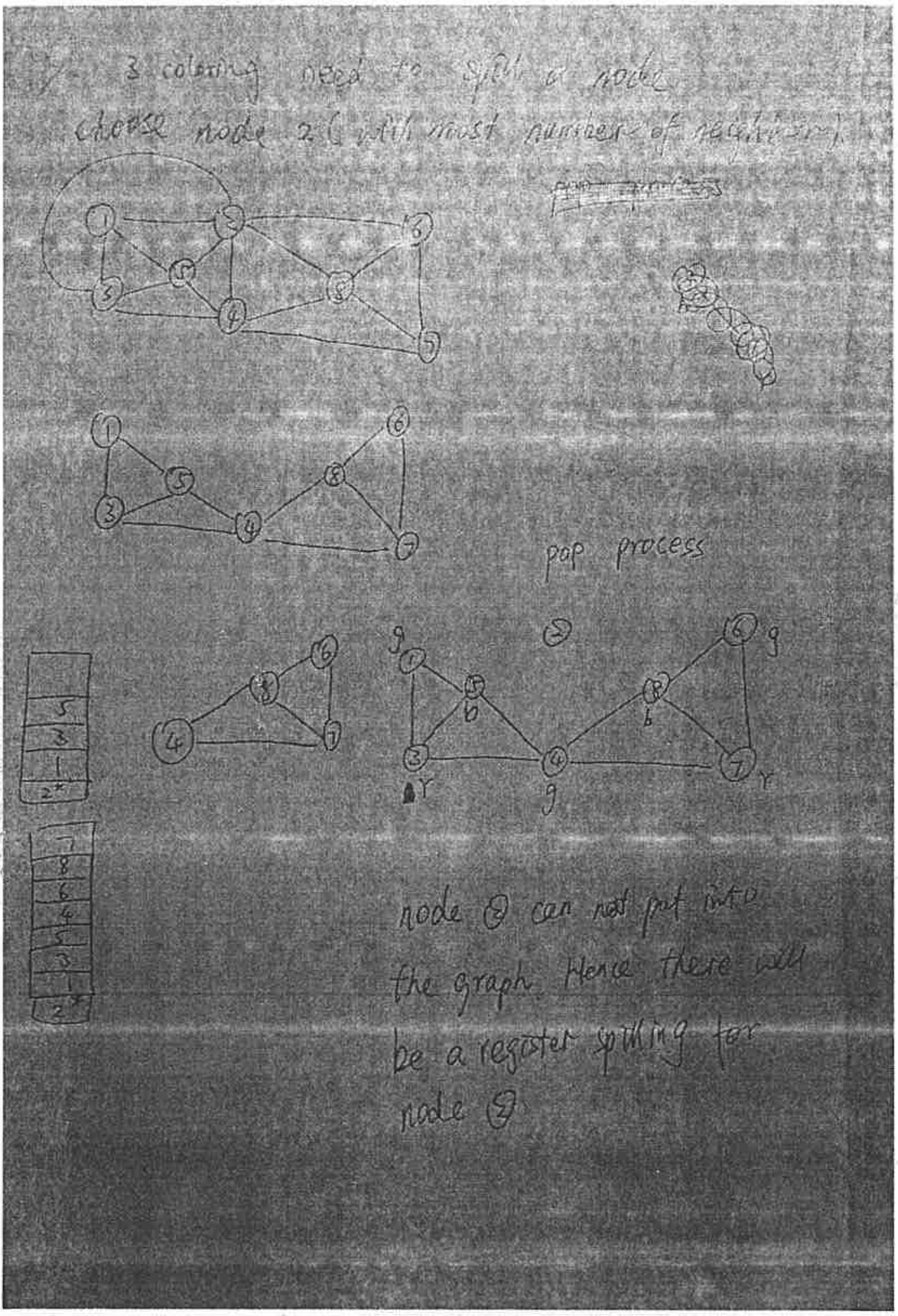
| |
|---|
| 1: load x; |
| 2: load y; |
| 3: mul; |
| 4: load x; |
| 5: load 2; |
| 6: add; |
| 7: load z; |
| 8: mul; |
| 9: sub; |
| 10: store s; |

(c) When generating code for this an if statement, we generate the code for the conditional expression and assign the result to an object c of typeValue If the condition is false, we should skip the then part, so we need to generate the target statement for the branching. However, when generating the code for the jump, we do not know which statement should be our target statement (i.e. which unit is the statement after the then part).

With backpatching, if we have many statements in the then part, we would need to overwrite the null reference in a jump that was generated a long time ago.

NOP padding generate two NOP instructions as jump targets, and insert them in the right positions. No need to go back and it is more clear which statement will be the target jumping position in nested if or while statements.

(d)



3 coloring need to split a node
choose node 2 (with most number of neighbors).

node 8 can not put into the graph. Hence there will be a register spilling for node 8

5. (a) Available Expressions is a forward-must analysis: whether an expression is available after a node depends on whether it is available before the node (but not vice versa), and an expression is available before a node if it is available after every predecessor node.

(b)

| Intra-node transfer equation | |
|---|---|
| Origin | Simplified |
| $in_L(1) = out_L(1) \backslash \emptyset \cup \emptyset$ | $in_L(1) = out_L(1)$ |
| $in_L(2) = out_L(2) \backslash \{t\} \cup \{x\}$ | $in_L(2) = out_L(2) \backslash \{t\} \cup \{x\}$ |
| $in_L(3) = out_L(3) \backslash \{z\} \cup \{y\}$ | $in_L(3) = out_L(3) \backslash \{z\} \cup \{y\}$ |
| $in_L(4) = out_L(4) \backslash \emptyset \cup \{x, z\}$ | $in_L(4) = out_L(4) \cup \{x, z\}$ |
| $in_L(5) = out_L(5) \backslash \{t\} \cup \{t\}$ | $in_L(5) = out_L(5) \cup \{t\}$ |
| $in_L(6) = out_L(6) \backslash \{z\} \cup \{z\}$ | $in_L(6) = out_L(6) \cup \{z\}$ |
| $in_L(7) = out_L(7) \backslash \emptyset \cup \{t\}$ | $in_L(7) = out_L(7) \cup \{t\}$ |
| $in_L(8) = out_L(8) \backslash \emptyset \cup \emptyset$ | $in_L(8) = out_L(8) \backslash \emptyset \cup \emptyset$ |

| Inter-Node Transfer equation |
|---|
| $out_L(1) = in_L(2)$ |
| $out_L(2) = in_L(3)$ |
| $out_L(3) = in_L(4)$ |
| $out_L(4) = in_L(5) \cup in_L(7)$ |
| $out_L(5) = in_L(6)$ |
| $out_L(6) = in_L(4)$ |
| $out_L(7) = in_L(8)$ |
| $out_L(8) = \emptyset$ |

(c)

After substitute away $out_L$. Following equation will be shown.

| Simplified |
|---|
| $in_L(1) = in_L(2)$ |
| $in_L(2) = in_L(3) \backslash \{t\} \cup \{x\}$ |
| $in_L(3) = in_L(4) \backslash \{z\} \cup \{y\}$ |
| $in_L(4) = in_L(5) \cup in_L(7) \cup \{x, z\}$ |
| $in_L(5) = in_L(6) \cup \{t\}$ |
| $in_L(6) = in_L(4) \cup \{z\}$ |
| $in_L(7) = in_L(8) \cup \{t\}$ |
| $in_L(8) = \emptyset \backslash \emptyset \cup \emptyset = \emptyset$ |

After 5 rounds of iteration, the final results will converge.

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $in_L(1)$ | ∅ | ∅ | x | x, y | x, y | x, y |
| $in_L(2)$ | ∅ | x | x, y | x, y | x, y | x, y |
| $in_L(3)$ | ∅ | y | x, y | x, t, y | x, t, y | x, t, y |
| $in_L(4)$ | ∅ | x, z | x, t, z | x, t, z | x, t, z | x, t, z |
| $in_L(5)$ | ∅ | t | t, z | x, t, z | x, t, z | x, t, z |
| $in_L(6)$ | ∅ | z | x, z | x, z | x, t, z | x, t, z |
| $in_L(7)$ | ∅ | t | t | t | t | t |
| $in_L(8)$ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |

For reporting of errors and errata, please visit pvpdiscuss.appspot.com
Thank you and all the best for your exams!