Homework2

Hyunki Lee

Panther# 002-34-4677

1.

a) aBBcbbCa

S→AB→aBBB→ not possible. We should end with ‘a’ however form ‘B’ there are no ways to end with ‘a’

b) cacBbBcc

S→BC→cBbC→cCbbC→cAcbbC→caBBcbbC→cacBbBcbbC→ not possible. ‘cbbC’ cannot be ‘cc’

c)bbCbABBb

S->BC->CbC->bCAbC->bbCACAbC->bbCbCAbC-> not possible. ‘CAbC’ cannot be ‘ABBb’

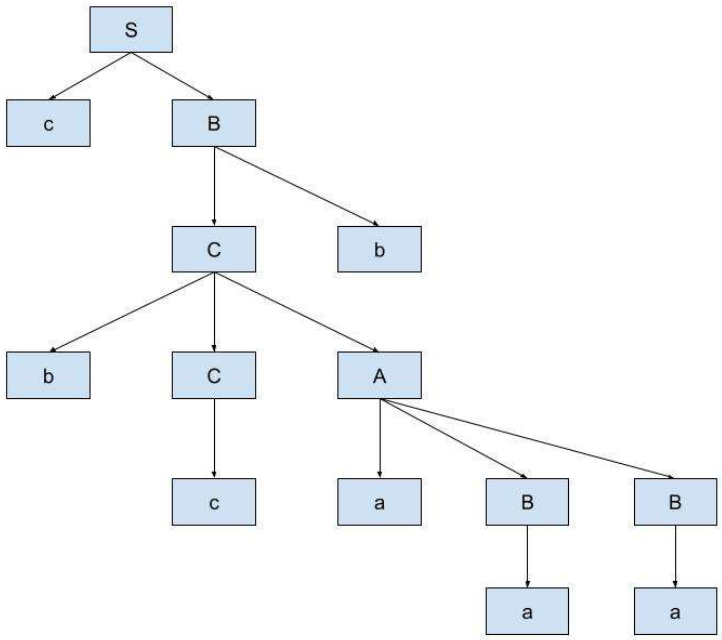
2.

a) cbcaaab

phrases: cbcaaab, bcaaab, bcaaa, caaa, aa

simple phrases: c, a, a

handle: c

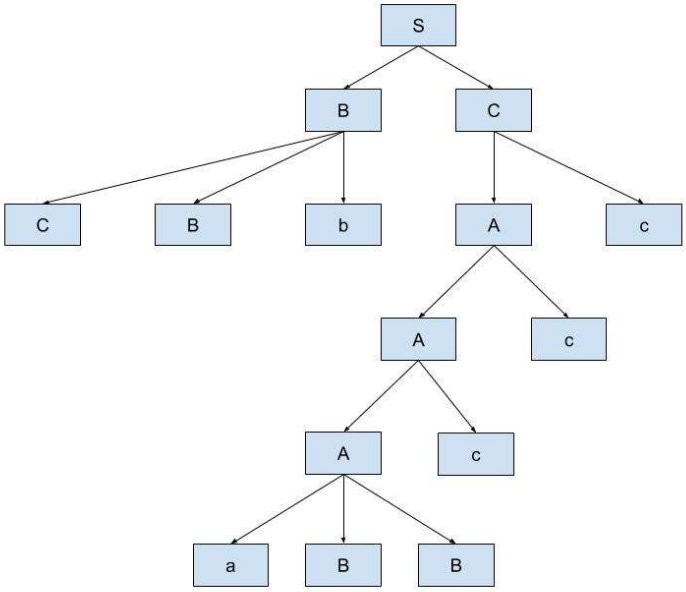


b) cBbaBBccc

phrases: cBbaBBccc, aBBcc, aBBc, aBB

simple phrases: CBb, aBB

handle: CBb

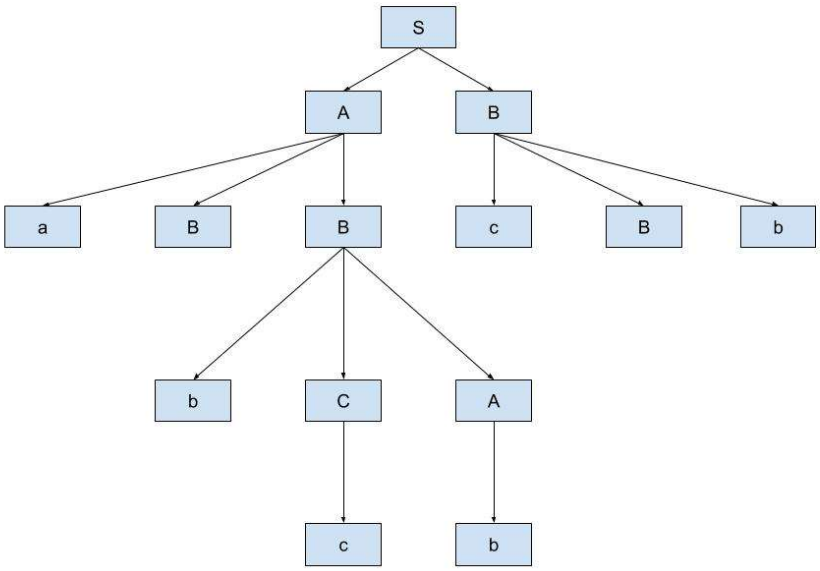


c)aBbcbcBb

phrases: aBbcbcBb, bcb, cb

simple phrases: c, b, cBb

handle: c



3.

Top-down parsers: build a tree from the root downward to the leaves. Top-down parser is easy to be formed. Top-down parsers is used for left recursion. Left recursion could affect that top-down parsers can enter infinite loop. Usually, LL algorithms.

Ex) xAα is a part of a leftmost derivation. If A-rules are A→bB, A→cBb A→a, then the next

sentential form could be xbBα, xcBbα, or xaα.

Bottom-up parsers: build a tree from the leaves upward to the root. Bottom-up parsers is stronger than top-down parsers. Bottom-up parser is structed ambiguous grammar. Usually, LR algorithms.

Ex) S→aAc A→aA|b then

aabc→aaAc→ aAc→ S

Parsers are in Syntax analyzers stage. Tokens are checked to see if they match the syntax of the programming language.

4.

- Attribute grammars: It can be used to describe more of the structure of a programming language than is possible with a context-free grammar. Attribute grammars help specify the syntax and semantics of a programming language.
- Operational Semantics: It attempts to describe the meaning of a statement or program by specifying the effects of running it on a machine. It builds a translator and a simulator for the idealized computer.
- Axiomatic semantics: It is based on mathematical logic, and it is the most abstract technique for specifying semantics.
- Denotational semantics: It is based on recursive function theory, and it is the most

rigorous and most widely known formal method for describing the meaning of programs.

They are in the semantic analysis stage. Semantics help interpret symbols, types, and relations with each other.

5.

<switch_stmt>→ switch “(“ (<expr>|<identifier>) “)” “{“ <body> “}”

<body>→

Case<literal>:

“{“<stmt>;”}”{<stmt>;}{case<literal>:“{“<stmt>;”}”{<stmt>;}}

[default: “{“<stmt>;”}”{<stmt>;}]

Convert EBNF into a BNF

<switch_stmt>→ switch (<expr> | <identifier>) {<body>}

<expr>→<expr> | <expr><expr>

<identifier>→<identifier> | <identifier><identifier>

<body>→case<literal> : <stmt>;

<body>→ case<literal> : <stmt>; default : <stmt>;

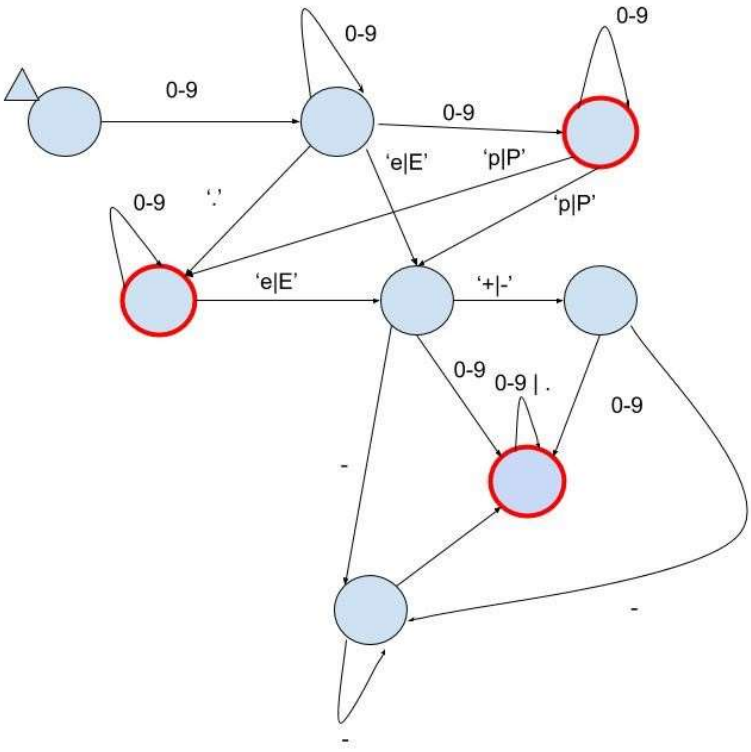
<body>→ case<literal> : <stmt>; | case<literal> : <stmt>; case<literal> : <stmt>;

<body>→ case<literal> : <stmt>; default : <stmt>; | case<literal> : <stmt>;

case<literal> : <stmt>;

<stmt>→<stmt> | <stmt><stmt>

6.



7.