2.2.1

$S \rightarrow SS+ \mid SS* \mid a$

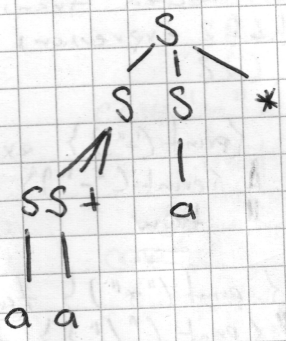
a) String "aa+a*" can be generated

"aa+" → SS+ → S ; SS* → S

"a"

Valid String

b) Construct parse tree



c) What language it generates?

↳ postfix with terminals "a" (digits), plus and multiplication signs.

2.2.2

What language is generated?

a) $S \rightarrow 0S1 \mid 01$

↳ zeros seguidos de unos en igual cantidad $\{0^n 1^n \mid n>=1\}$

b) $S \rightarrow +SS \mid -SS \mid a$

↳ prefix language/expression with plus and minus signs

d) $S \rightarrow aSbS \mid bSaS \mid \varepsilon$

↳ same amount of "b" than "a" including ε

2.2.3

Which from 2.2.2 are ambiguous?

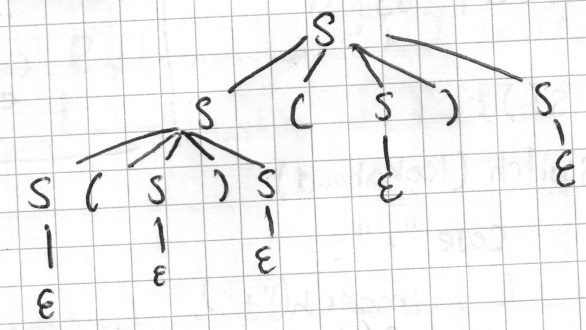
[redacted]

c, d and e are ambiguous as different trees can express the same expression, one from the Right an other from the left

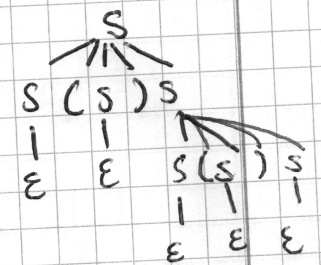
a) Not ambigue : $S \to 0S1 \mid 01$

b) Not ambigue : $S \to +SS \mid -SS \mid a$

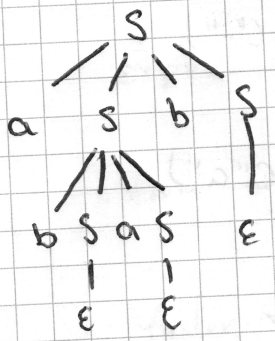
c) Ambigue : $S \to S(S)S \mid \varepsilon$



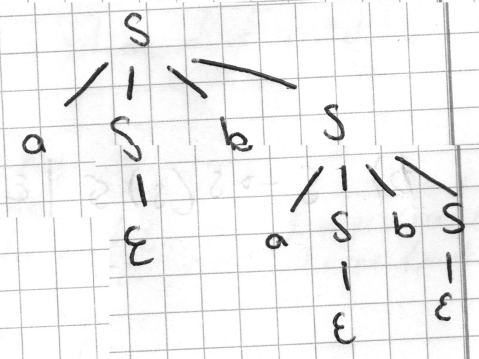
or



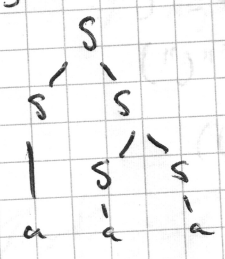
d) Ambigue : $S \to aSbS \mid bSaS \mid c$



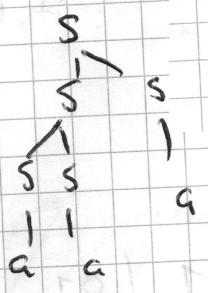
or



e) Ambigue : $S \to a \mid S+S \mid SS \mid S^* \mid (S)$



or



2.4 | Construct unambiguous grammars

b) Left-associative list of identifiers separated by comma.

↳ list → list , id | id

c) Right-associative list of identifiers separated by comma

↳ List → id , list | id

2.3.1 Construct a syntax-directed translation scheme that translates arithmetic expressions from infix to prefix

expr → {print("+")} expr + term
| {print("-")} expr - term
| term

term → {print("*")} term * factor
| {print("/")} term / factor
| factor

factor → digit {print(digit)} | expr

2.3.2 Construct a syntax directed translation scheme that translates from postfix to infix expressions.

expr → expr {print("+")} expr +
| expr {print("-")} expr -
| {print("(")} expr {print(")*(")} expr {print(")")} ↳ *
| {print("(")} expr {print(")/(")} expr {print(")")} ↳ /
| digit {print(digit)}

2.4.1 Construct descendant Parsers

```
def match (terminal):
    if lookahead != terminal
        # throw error
    else:
        lookahead = next()
```

a) $S \rightarrow +SS \mid -SS \mid a$

```
fun S():
    switch (lookahead)
        case "+":
            match("+");
            S();
            S(); break
        case "-":
            match("-")
            break
        case "a":
            match("a")
            break
        default:
            # Error.
            SyntaxError()
```

b) $S \rightarrow S(S)S \mid \varepsilon$

```
fun S():
    if lookahead == "(":
        match("(")
        S()
        match(")")
        S()
```

c) $S \rightarrow 0S1 \mid 01$

```
fun S():
    switch (lookahead)
        case "0":
            match("0")
            S()
            match("1")
            break
        case "1":
            break
        default:
            SyntaxError()
```