— from Sebesta book ch3.

— a grammar for a small language

G

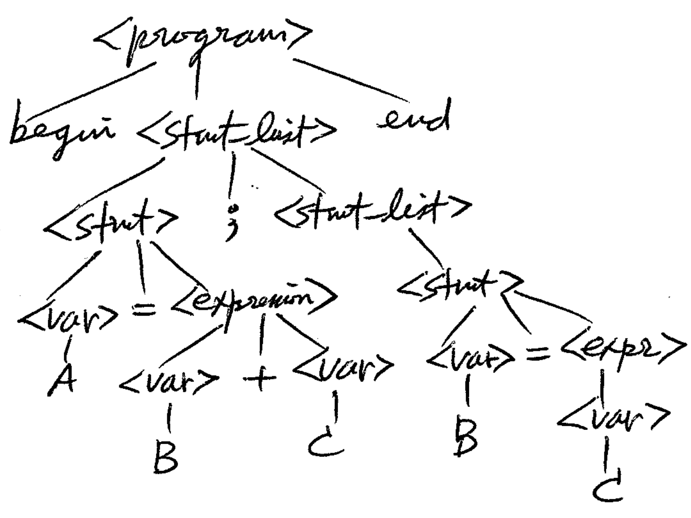
<program> → begin <stmt-list> end
<stmt-list> → <stmt>
  | <stmt> ; <stmt-list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
  | <var> − <var>
  | <var>

---

a sample program (string)

begin
  A = B+C ;
  B = C
end

— Show a parse tree



— The given string (program) is in the lang. defined by the Grammar, ∵ ∃ a parse tree or, ∃ a derivation.

— Show the left-most derivation

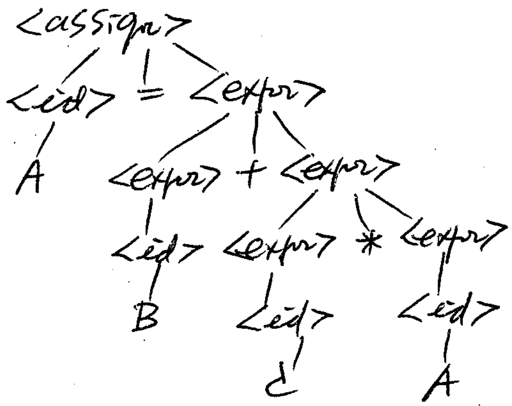
<program> ⇒ begin <stmt_list> end
⇒ begin <stmt> ; <stmt_list> end
⇒ begin <var> = <expr> ; <stmt_list> end
⇒ begin A = <expr> ; <stmt_list> end
⇒ begin A = <var> + <var> ; <stmt_list> end
⋮
⇒ begin A=B+C ; B=C end

— a simple assignment statement

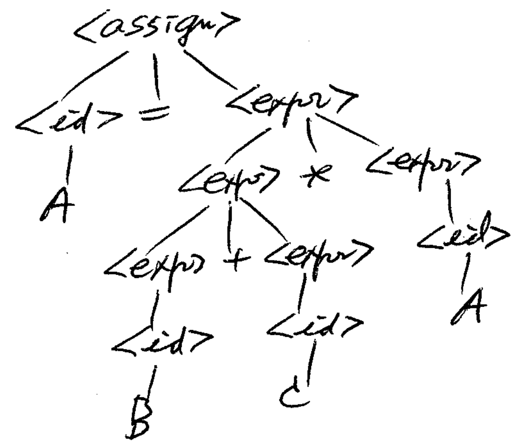
G
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <expr> + <expr>
| <expr> * <expr>
| (<expr>)
| <id>

ambiguous ✓
∵ (+, * / ()) same precedence way

string: A = B + C * A — ∃ more than 1 parse trees ⇒ G is ambiguous



or



⇒ write unambiguous G for <assign> statement:

- Keep precedence of ops (+ / * / () ↑)
- Keep associativity
  +, * — left assoc. ⇒ so, G should be left-rec.

G'
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <expr> + <term> | <term> — + lowest precedence
left rec.
<term> → <term> * <factor> | <factor> — *
<factor> → (<expr>) | <id> — () highest precedence

Sebesta book

— Dangling-else ambiguity — (Sebesta pp 124~)

<stmt> → if <expr> then <stmt>
  | if <expr> then <stmt> else <stmt>
  | ...

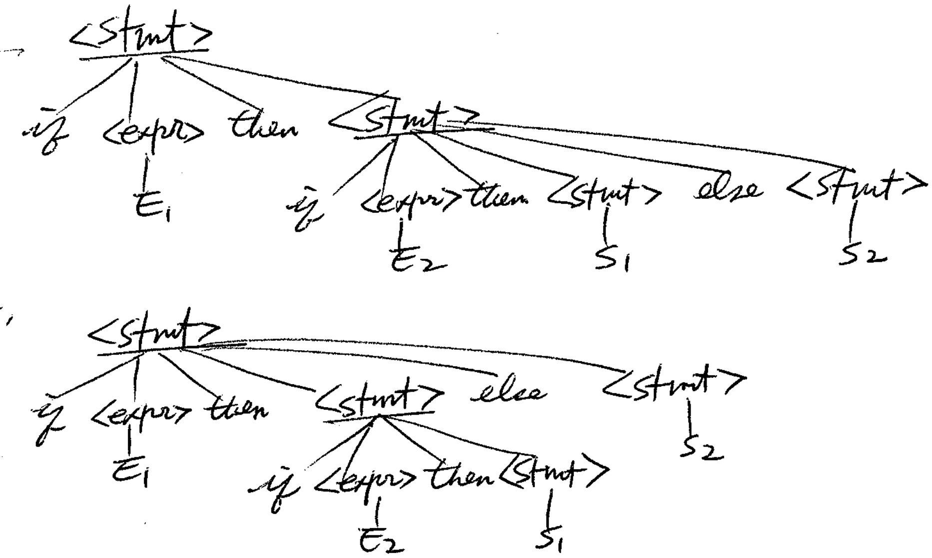
for if/if_else statement

ex) string:

if E1 then if E2 then S1 else S2 ??

— assume E1, E2, S1, S2 have ~~are~~ all terminals only.

ambiguous



C/C++ way by compiler

or,

⇒ Solutions:

1. Conditionals have closing keyword end. — Modula-2, Ada,

   <S> → if <expr> then <SL> end
   | if <expr> then <SL> else <SL> end

2. Compiler matches else with nearest unmatched if. — C/C++

3. Using different productions for if-state and if_else_state. — also lang. supports { }. Java

   <if_state> → if (<expr>) <stmt>
   <if_else_state> → if (<expr>) <stmt_no_short_if> else <stmt>

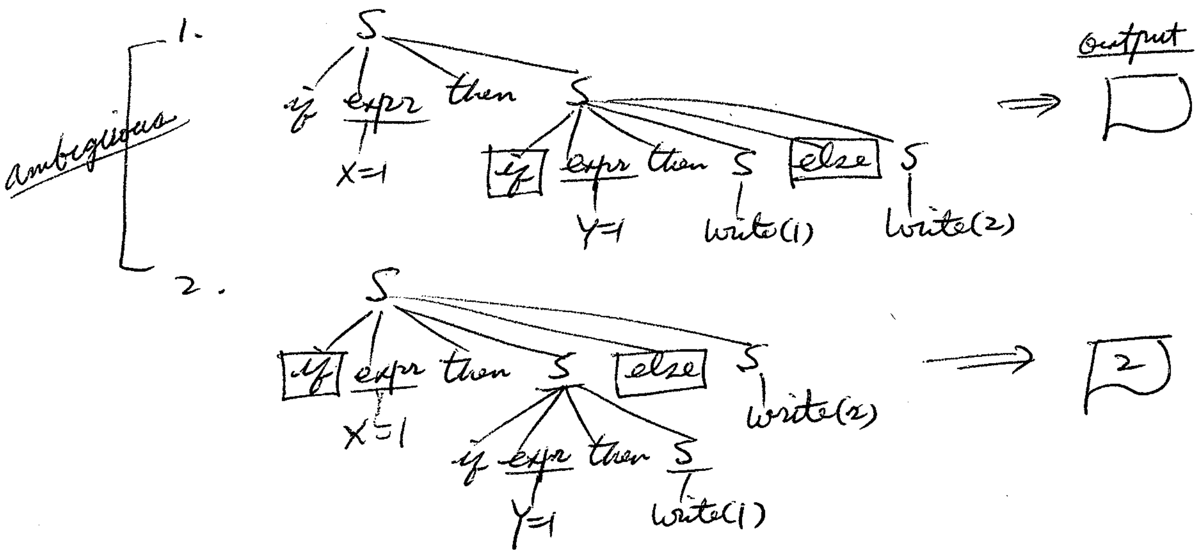
24) PASCAL Syntax

S → if expr then S
  | if expr then S else S

Assume: X=2
Y=1

Strip: if X=1 then if Y=1 then write(1) else write(2)

⇒ ∃ 2 parse trees



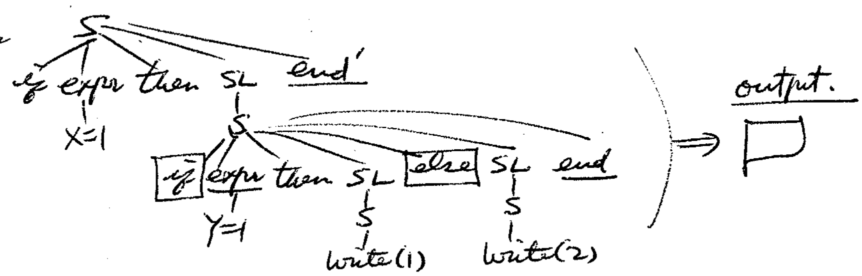
24) Modula-2 Syntax

S → if expr then SL end
  | if expr then SL else SL end

Assume: X=2
Y=1

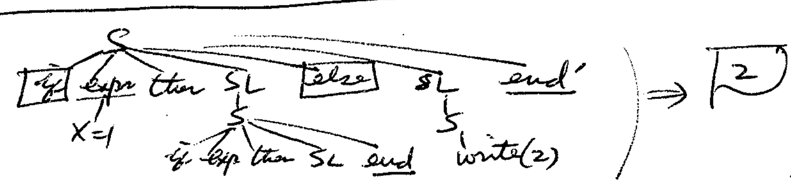
Strip: if X=1 then $\boxed{\text{if}}$ Y=1 then write(1) $\boxed{\text{else}}$ write(2) end end′

⇒ ∃ only 1 parse tree



Strip: $\boxed{\text{if}}$ X=1 then if Y=1 then write(1) end $\boxed{\text{else}}$ write(2) end′

⇒ ∃ only 1 parse tree



— variations of CFG notation

BNF

< > — non_terminal

::= — →

| — choice

eg) <E> ::= <E> + <T> | <T>

⇒ simplified way:

E → E + T | T — non_T upper case, T — low case

EBNF way

EBNF

non_T starts with capital

| — choice

( ) — group

{ } — $\emptyset$ or more times repeat

[ ] — optional ($\emptyset$ or 1 time)

syntax chart

→ — derivation

□ — non_T

○ — Terminal

---

eg) BNF

2 productions

<expr> ::= <expr> + <term> | <term>

<term> ::= <term> * <factor> | <factor>

<factor> ::= <digit>

<digit> ::= 0 | 1 | 2 | --- | 9

vs.

EBNF

1 production

Expr ::= Term { '+' Term }

Term ::= Factor { '*' Factor }

Factor ::= Digit

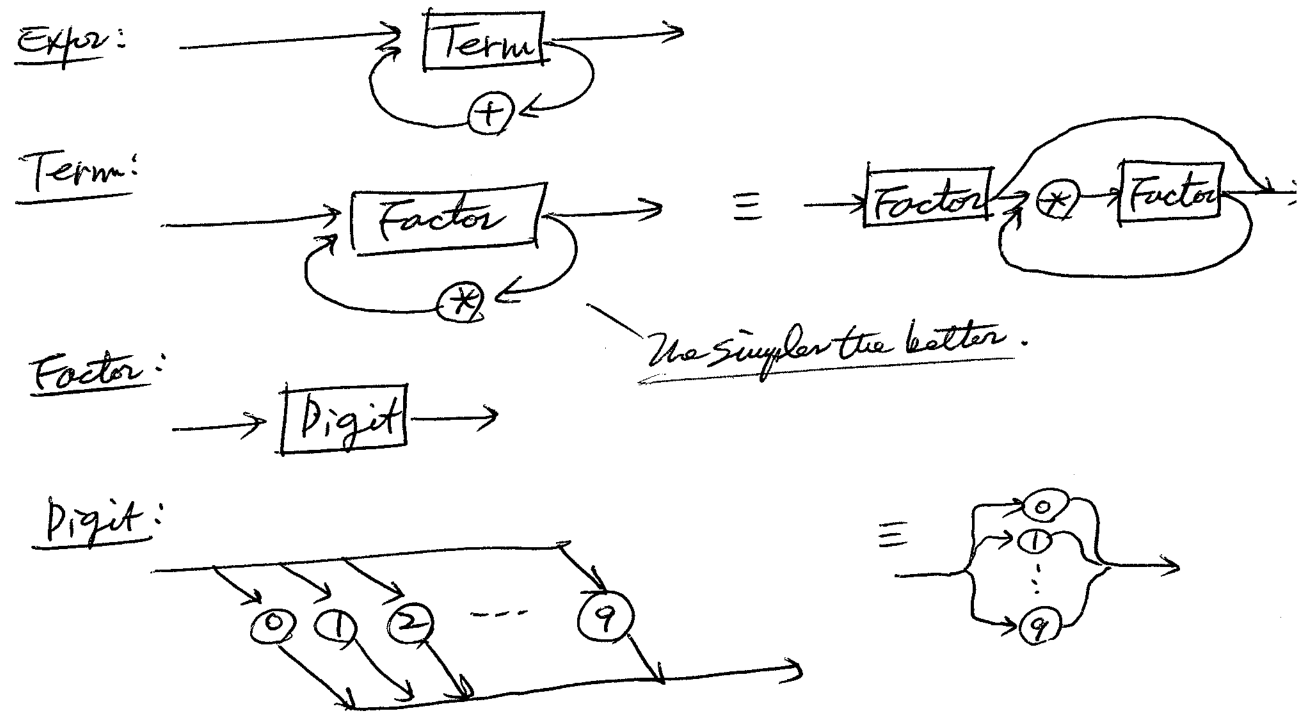
Digit ::= 0 | 1 | 2 | --- | 9

— Syntax Chart

each non_T has a chart

↓

Syntax chart



Convert EBNF → BNF

EBNF		BNF
optional:		
$S \to a\,b\,[c]\,d$	⟶	$S \to a\,b\,c\,d \mid a\,b\,d$
ϕ or more times:		
$S \to a\,b\,\{c\}\,d$	⟶	$S \to a\,b\,\underline{\underline{X}}\,d$ (new nonT) $X \to \varepsilon \mid c\,X$
mixed:		
$S \to a\,b\,\{c\}\,[d]\,e$	⟶	$S \to a\,b\,\underline{\underline{X}}\,\underline{\underline{d}}\,e \mid a\,b\,\underline{\underline{X}}\,e$ $X \to \varepsilon \mid c\,X$