4. Syntax Analysis

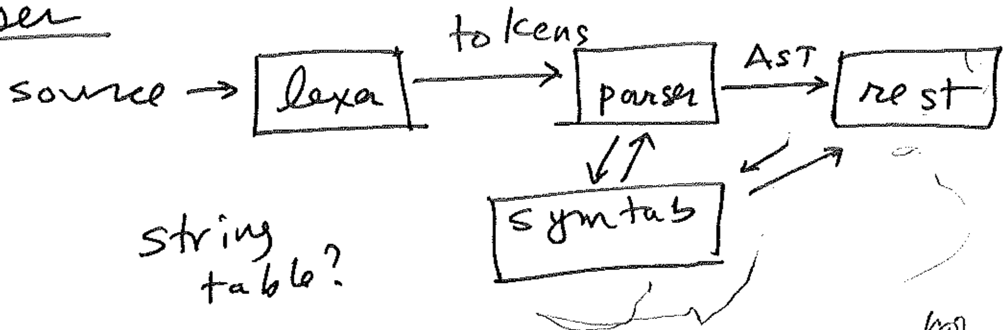
CMPS-104A
ch. 4 = 1
Dragon 4:191

BNF: Backus-Naur Form.

- precise syntax
- automate parser
- detect errs.
- iter develop.

συνταξις = arrange together

Parser



Universal parser.
ambig CF : $O(n^3)$ McWhorter
unambig CF : $O(n^2)$ Earley.
$G = \langle V_N, V_T, P, S \rangle$ LR(k) : $O(n)$ — Knuth.
LALR(1)

Representative Grammars

LR(k)
bottom up
shift reduce

left rec

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow i$

LL(k)
top down
recursive descent

$E \rightarrow T E'$
$E' \rightarrow + T E'$
$E' \rightarrow$
$T \rightarrow F T'$
$T' \rightarrow * F T'$
$T' \rightarrow$
$F \rightarrow (E)$
$F \rightarrow i$

CMPS - 104A
ch 4 = 2
Diagon 4.194

ambiguious

$E \to E + E$
$E \to E * E$
$E \to (E)$
$E \to i$

need prec/assoc rules
or mult parse trees
%left '+'
%left '*'

4.1.3 Handling Syntax Errors

- req: locate src coord
- want: fixup?
- avoid: cascade (if possible)

ERRORS
lexical - scanner
syntactic - parser
semantic - symtab & codgen
logical - cc can't help.
- lint sometimes
if (a = b) ...

LL/LR - detect errors immediately
- 1st non-viable prefix

ERROR Recovery

panic : - discard input sym
until find synch token
end }

~~ca~~
- recover & continue

CMPS-104A
ch4 ⟹ 3
Dragon 4.196

error pdns in yacc

pop stack until shift error
~~i~~gnore ~~tokens~~ until shift

```
ex:   block : '{' stmts '}'
            | '{' error }
            ;
```

4.2 Context-Free Grammars

$$G = \langle V_N, V_T, P, S \rangle$$

S — start symbol (goal)
P — set of rules.
V_T — terminals
V_N — nonterminals.

token = terminal + lexeme
V_N imposes hierarch. structure.

$S \in V_N$ $\quad P = \{(A \to \beta) \mid A \in V_N \wedge \beta \in V^*\}$

$V = V_N \cup V_T$

4.2.2 Theoretical Notation (bison is different)

(1) V_T: $a, b, c, \ldots, 0, 1, \ldots$ bold face
$+, *, (,), \ldots$ id, if, ...

(2) V_N: upcase $A, B, C, \ldots$ $\rightarrow$
S = start sym |
lc italic *expr*, *stmt*, ...

CMPS-104A

ch 4 ⇒ 4

Dragon 4.199

(3) upcase end $X, Y, Z \in V$

i.e. $X, Y, Z \in V_N \cup V_T$

(4) lower end

$u, v, w, \ldots \in V_T^*$

(5) l.c. Ελληνικα (begin)

$\alpha, \beta, \gamma, \cdots \in V^*$

(6) $\left.\begin{array}{l} A \to \alpha \\ A \to \beta \\ A \to \gamma \end{array}\right\} \longrightarrow A \to \alpha \mid \beta \mid \gamma \mid \cdots$

(7) usu LHS of 1st rule is S

bison		meta
lc	$\in V_N$	:
uc	$\in V_T$	|
'x'	$\in V_T$	;

4.2.3 Derivations

- construction of a program
- rewriting rules

$A \to \beta$ produces

$\alpha \Rightarrow \beta$ derives

$\alpha \stackrel{*}{\Rightarrow} \beta$ derives in zero⁺ steps.

ETF example

$E \stackrel{*}{\Rightarrow} a * b + c$

left most deriv

$$
\begin{aligned}
E &\Rightarrow E + T \\
&\Rightarrow T + T \\
&\Rightarrow T * F + T \\
&\Rightarrow F * F + T \\
&\Rightarrow i * F + T \\
&\Rightarrow i * i + T \\
&\Rightarrow i * i + F \\
&\Rightarrow i * i + i
\end{aligned}
$$

LL(k)

rightmost deriv

$$
\begin{aligned}
E &\Rightarrow E + T \\
&\Rightarrow E + F \\
&\Rightarrow E + i \\
&\Rightarrow T + i \\
&\Rightarrow T * F + i \\
&\Rightarrow T * i + i \\
&\Rightarrow F * i + i \\
&\Rightarrow i * i + i
\end{aligned}
$$

parse

LR(k)

CMPS - 104A
ch 4 = 5
Dragon 4. 201

Parse Trees & Derivation

Parsing $\equiv$ Derivation$^{(-1)}$

LL(k) ETF
leftmost

$\cancel{E \Rightarrow TE'}$
$\cancel{\Rightarrow FT'E'}$
$\cancel{\Rightarrow iT'E'}$ ~~(null deriv)~~
$\cancel{\Rightarrow iE'}$

$$
\begin{aligned}
E &\Rightarrow TE' \\
&\Rightarrow FT'E' \\
&\Rightarrow iT'E' \\
&\Rightarrow i * FT'E' \\
&\Rightarrow i * iT'E' \quad \text{(null deriv)} \\
&\Rightarrow i * iE' \\
&\Rightarrow i * i + TE' \\
&\Rightarrow i * i + FT'E' \\
&\Rightarrow i * i + iT'E' \\
&\Rightarrow i * i + iE' \\
&\Rightarrow i * i + i
\end{aligned}
$$

ambig grammar ~~(righ...)~~

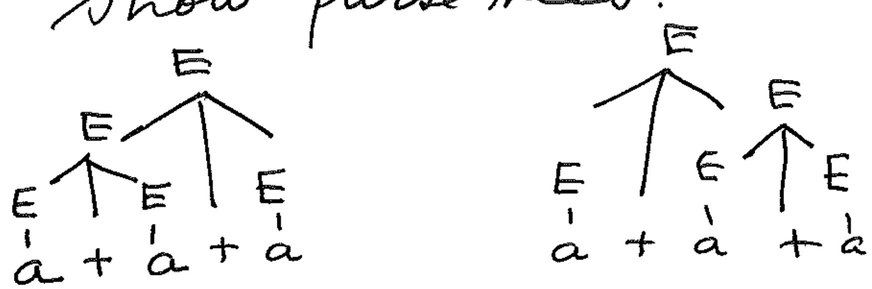
rightmost

$E \rightarrow E + E$
$E \rightarrow a$

$E \Rightarrow E + E$
$E \Rightarrow E + E + E$
⋮

show parse trees.

left → Reduce
right → shift



CMPS-104A
ch 4 ⇒ 6
Dragon 4.204

example balanced parens

scan.l

```
"("   return '('
")"   return ')'
.     /**/
```

parse.y

```
S : '(' S ')' S
  |
  ;
```

Power

$L(G) = \{x^n y^n\}$ $S \to xSy$
$S \to$

~~FA can't count~~
PDA = FA + stack

vanWyngaarden. $x^n y^n z^n$

why sep? (1) modularity
(2) lex usu simple.
(3) regex more concise
(4) lex scanner more efficient code
(5) scan white + comments

Eliminate Ambiguity

ambig ETF → unambig ETF

The Dangling else.

$S \to$ *if* E *then* S
| *if* E *then* S *else* S
| X
| while E do S

parse tree for

prec else
~~left~~ right assoc

If E then X else if E then X else X

CMPS-104A
ch4 $\Rightarrow$ 7
Dragon 4.212

unamb if else

$S \rightarrow M \mid U$
$M \rightarrow$ if E then M else M
$\mid$ X
$U \rightarrow$ if E then M else U
$\mid$ if E then S

---

skip left rec

Non CF: ex: $L = \{wcw \mid w \in (a|b)^*\}$
PL not CF $\longrightarrow$ ~~decl~~
declarations are Ctx Sens.

---

skip top down

CMPS-104A
ch 4 = 8
Dragon

4.5 BottomUp

Reduction: replace seq $\in V^*$ at top of stack by LHS symbol.

fomally $S \overset{*}{\underset{rm}{\Rightarrow}} \alpha A w \Rightarrow \alpha \beta w$

then pos after α is a handle

$\alpha \in V^*$
$A \in V_N$
$\beta \in V^*$
$w \in V_T^*$

always @ top of stack

parsing actions

- shift
- reduce
- accept
- error

conflicts

- shift/reduce
- reduce/reduce

4.6 LR parsing

- most PL are LR(k) (if they are CF)
- most general non-backtracking
- earliest detect syntax error.
- $LR(k) \supset LL(k)$

LR(0) item = rule with a dot @ tos.

$A \to .XYZ$
$A \to X.YZ$
$A \to XY.Z$
$A \to XYZ.$ ← reduction.

stack • unscanned

closure (I)

I : set of items.

$\forall (A \to \alpha . B \beta) \in I$

and $(B \to \gamma) \in P$

add $(B \to . \gamma)$ to I

```
alg closure (I) {
    J = I
    loop {
        ∀ (A → α . B β) ∈ J {
            ∀ (B → γ) ∈ P {
                if (B → . γ) ∉ J
                    add (B → . γ) to J
            }
        }
    } until done
    ret J
}
```

Kernel items all elems of form $(A \to X . \beta)$

nonkernel items $(A \to . \cancel{\beta})$

~~alg~~ GOTO (I, X)

I = set of items

$X \in V$

$GOTO(I, X) = \forall (A \to \alpha . X \beta) \in I$

$\to (A \to \alpha X . \beta)$

CMPS-104A
ch4 =10
Dragon 244

Compute LR(0) machine

$G = \langle V_N, V_T, P, S \rangle$

$G' = \langle V_N', V_T', P', S' \rangle$ ← augmented grammar.

$V_N' = V_N \cup \{S'\}$, $S' \notin V$

$V_T' = V_T \cup \{\$\}$, $\$ \notin V$

$P' = P \cup \{S' \to \$ S \$\}$

alg: construct LR(0)

```
C = closure ({S' -> $.S$})
repeat
      ∀ (I ∈ C) {
            ∀ (X ∈ V') {
                  if (GOTO (I,X) ≠ Φ ∧ not ∈ C) {
                        add it to C
                  }
            }
      }
} until done
```

---

page 249

refer to handout

LR parsing

LR(0)
#1.

configuration is

$$(\$ s_0 X_1 s_1 X_2 s_2 \dots X_m s_m \bullet a_i a_{i+1} \dots a_n \$)$$

represents a sentential form

$$X_1 \dots X_m a_i \dots a_n.$$

where $S \Rightarrow$

$(\$ s_0 \circ a_0 a_1 a_2 \dots \$)$

LR(0): an LR(0) item is a rule with a dot

ex:

rule	items
$A \to XYZ$	$A \to .XYZ$
	$A \to X.YZ$
	$A \to XY.Z$
	$A \to XYZ.$
$A \to \bullet$	$A \to .$

LR(0) machine is canonical collection of sets of items in G.

assume $G = \langle V_n, V_t, P, S \rangle$

then augmented grammar

$$G' = \langle V_n', V_t', P', S' \rangle$$

where $V_n' = \{S'\} \cup V_n$

$V_t' = \{\$\} \cup V_t$

$P' = P \cup \{(S' \to \$ S \$)\}$

and $S', \$ \notin V$.

$S' \notin V$
$\$ \in V$
$V = V_n \cup V_t$

LR(0)
#2

<u>build sets of items</u>

C ← {closure (S' → $. E $) }
~~loop~~
~~~~ ~~not~~
loop {
∀ I ∈ C and gr. sym X
where goto (I, X) ≠ Φ
and not in C
<u>do</u> add goto (I, X) to C.
until done

<u>closure (I)</u>

J ← I
loop { ∀ item ($A \to \alpha . B \beta$) in J
and ∀ ~~rule~~ $B \to \gamma$ where $B \to . \gamma$
not in J <u>do</u> add $B \to . \gamma$ to J
until done
return J

---

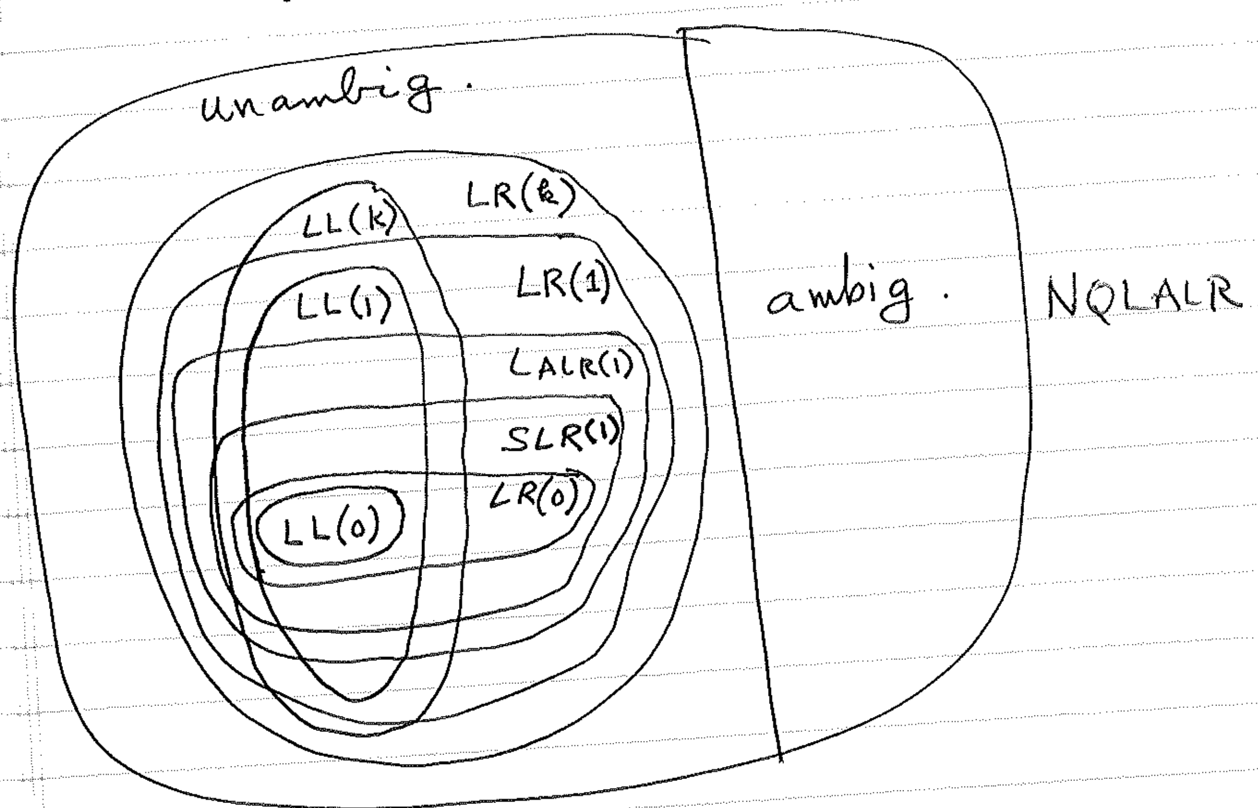
goto (I, X)

∀ ($A \to \alpha . X \beta$) ∈ I
add ($A \to \alpha X . \beta$) to goto
~~~~

1

Andrew Appel

Modern Compile Impl



ambig

Rule . lookahead

R no prec $\to$ X

LA no pre $\to$ X

prec(R) $>$ prec(LA) $\to$ Reduce

prec(R) $<$ prec(LA) $\to$ Shift

prec(R) $=$ prec(LA)

assoc

left $\to$ Reduce

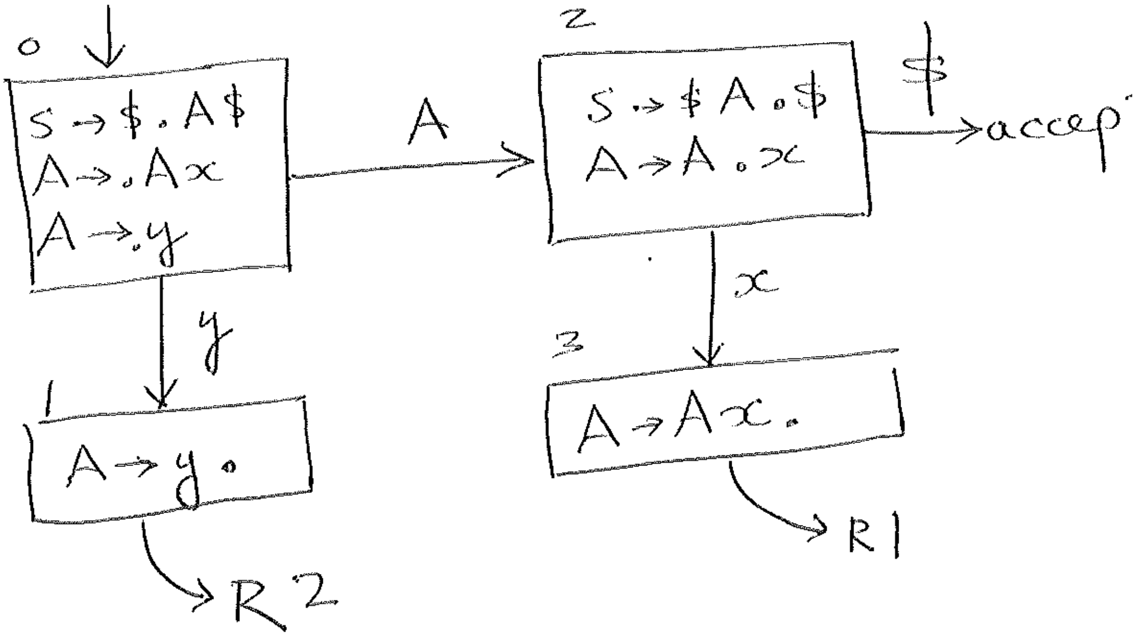
right $\to$ Shift

nonassoc $\to$ X

0. $S \to \$A\$$
1. $A \to Ax$
2. $A \to y$

LR(0)

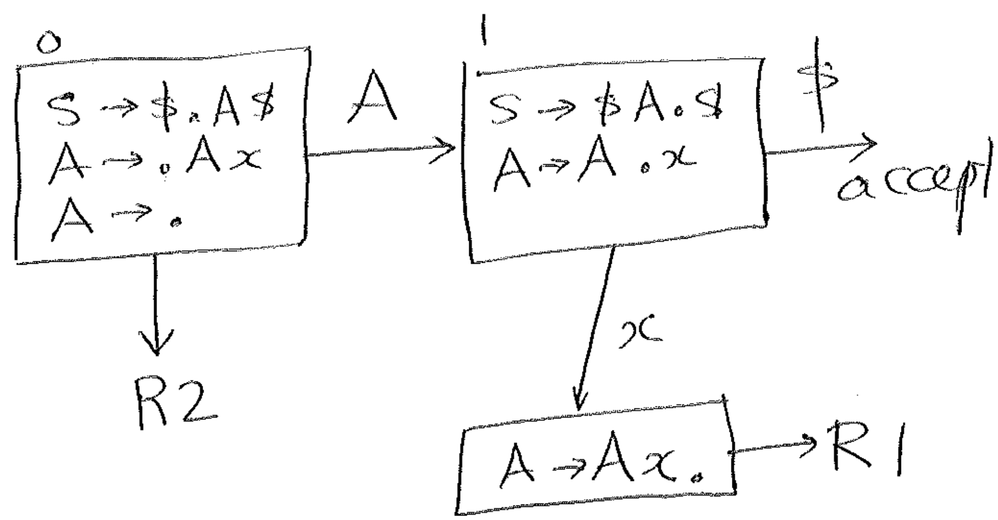
1



---

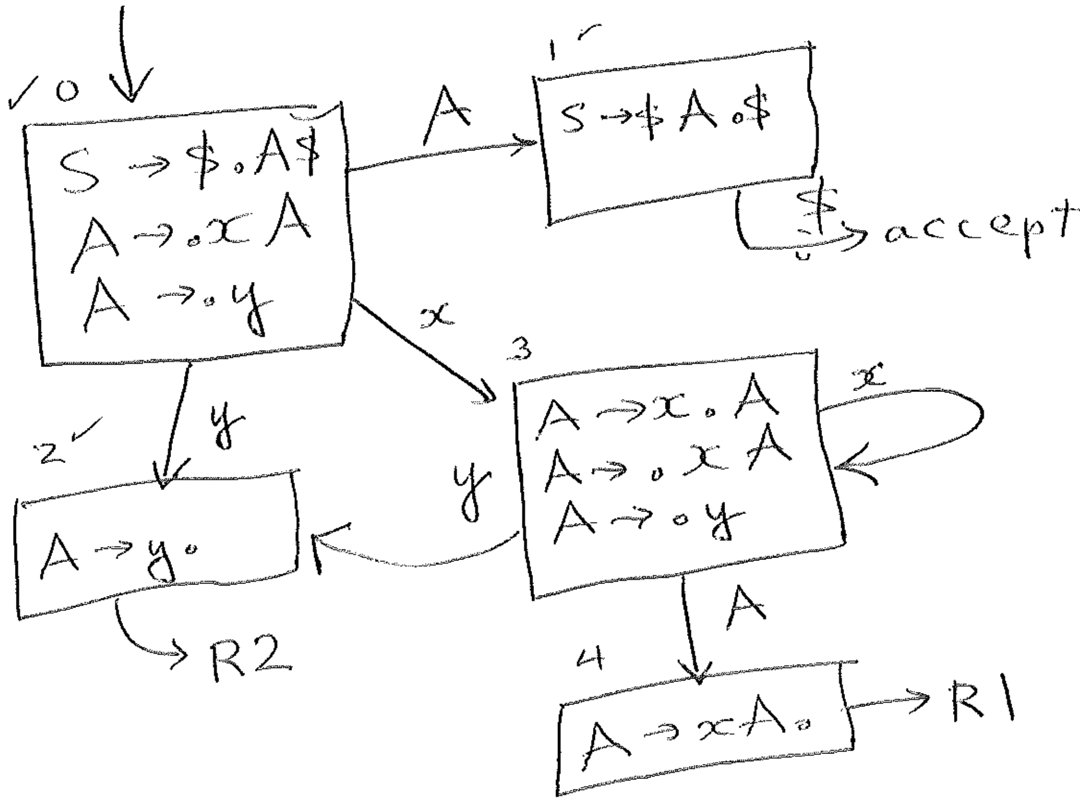
0. $S \to \$A\$$
1. $A \to Ax$
2. $A \to$

2



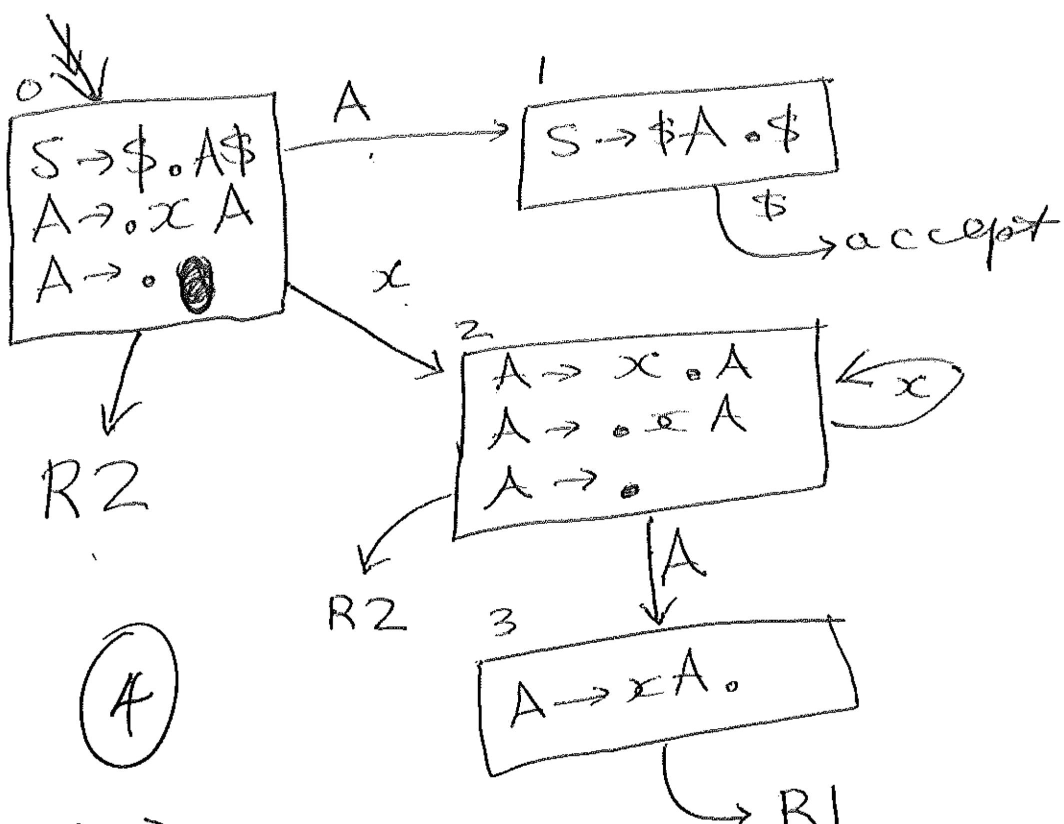
0. $S \to \$A\$$
1. $A \to xA$
2. $A \to y$

③



---

0. $S \to \$A\$$
1. $A \to xA$
2. $A \to$



SR conf
I0, I2
not LR(0)

SLR Lookahead

A | \$ ~~R~~

④

∴ R2 {\$}

NOT LR(0)

```
// $Id: ambiguous-else.y,v 1.1 2011-10-28 18:07:07-07 - - $

// Example of solving the problem of the dangling else with an
// ambiguous grammar and precedence declarations.

%verbose

%token IF WHILE
%right ELSE
%start program

%%

program     : program statement
            |
            ;

statement   : ifhead statement ELSE statement
            | ifhead statement %prec ELSE
            | whilehead statement
            | otherstmt
            ;

ifhead      : IF '(' expr ')'
            ;

whilehead   : WHILE '(' expr ')'
            ;

otherstmt   : expr ';'
            ;

expr        : 'x'
            ;

%%
```

```
// $Id: unambiguous-else.y,v 1.1 2011-10-28 18:07:07-07 - - $

// Example of solving the problem of the dangling else with an
// ambiguous grammar and precedence declarations.

%verbose

%token IF WHILE
%right ELSE
%start program

%%

program    : program statement
           |
           ;

statement  : closedstmt
           | openstmt
           ;

closedstmt : ifhead closedstmt ELSE closedstmt
           | whilehead closedstmt
           | otherstmt
           ;

openstmt   : ifhead closedstmt ELSE openstmt
           | ifhead statement
           | whilehead openstmt

ifhead     : IF '(' expr ')'
           ;

whilehead  : WHILE '(' expr ')'
           ;

otherstmt  : expr ';'
           ;

expr       : 'x'
           ;

%%

```