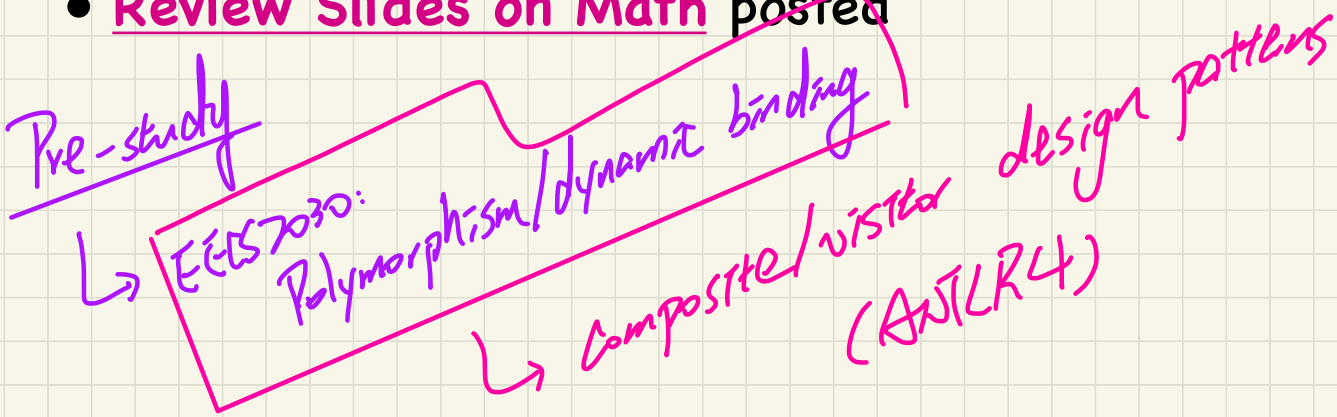
Lecture 12 - June 12

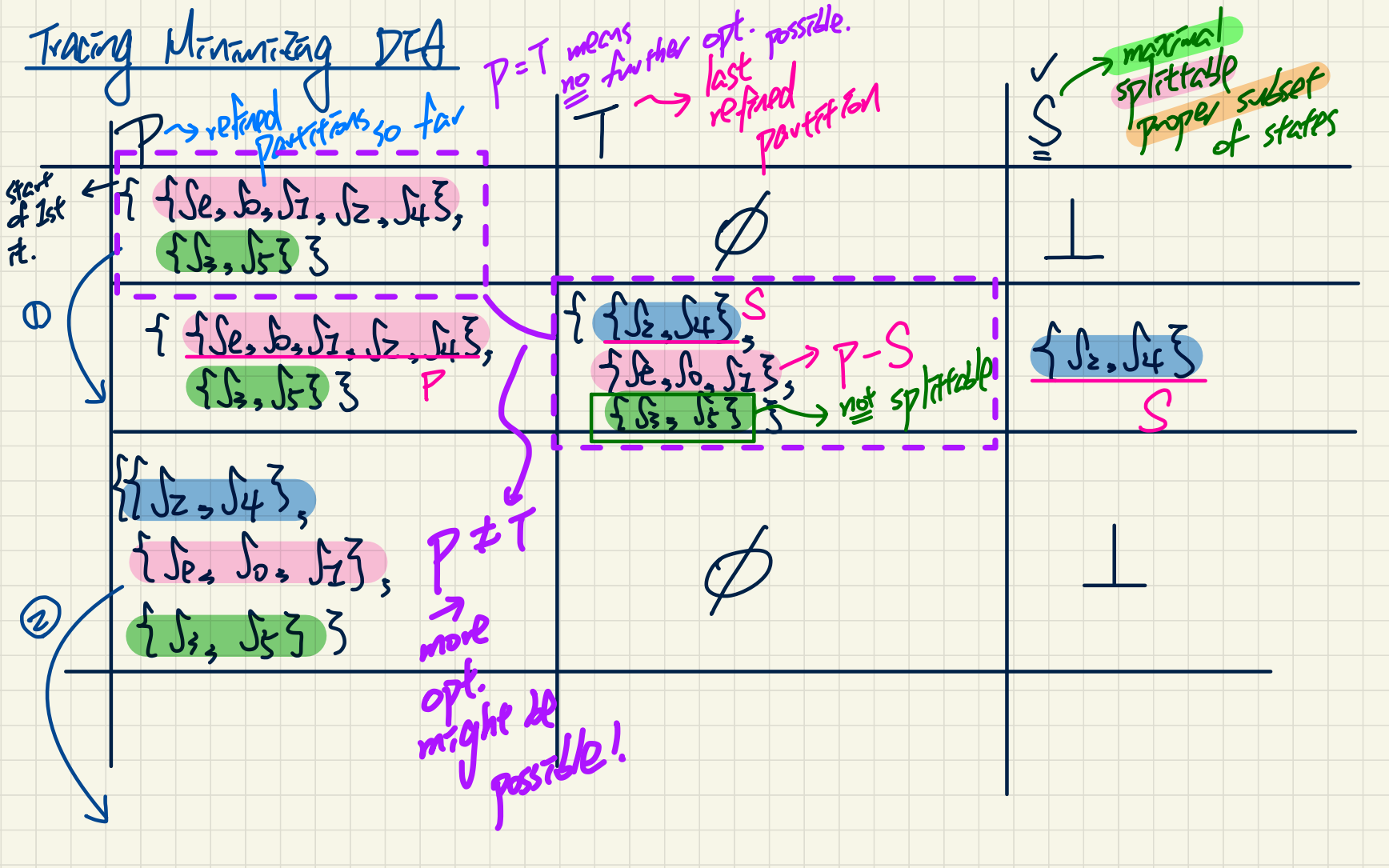
Lexical Analysis, Syntactic Analysis

Regular Expression to Minimized DFA
Implementing a Scanner
Parser in Context
Context Free Grammar (CFG): Intro

Announcements/Reminders

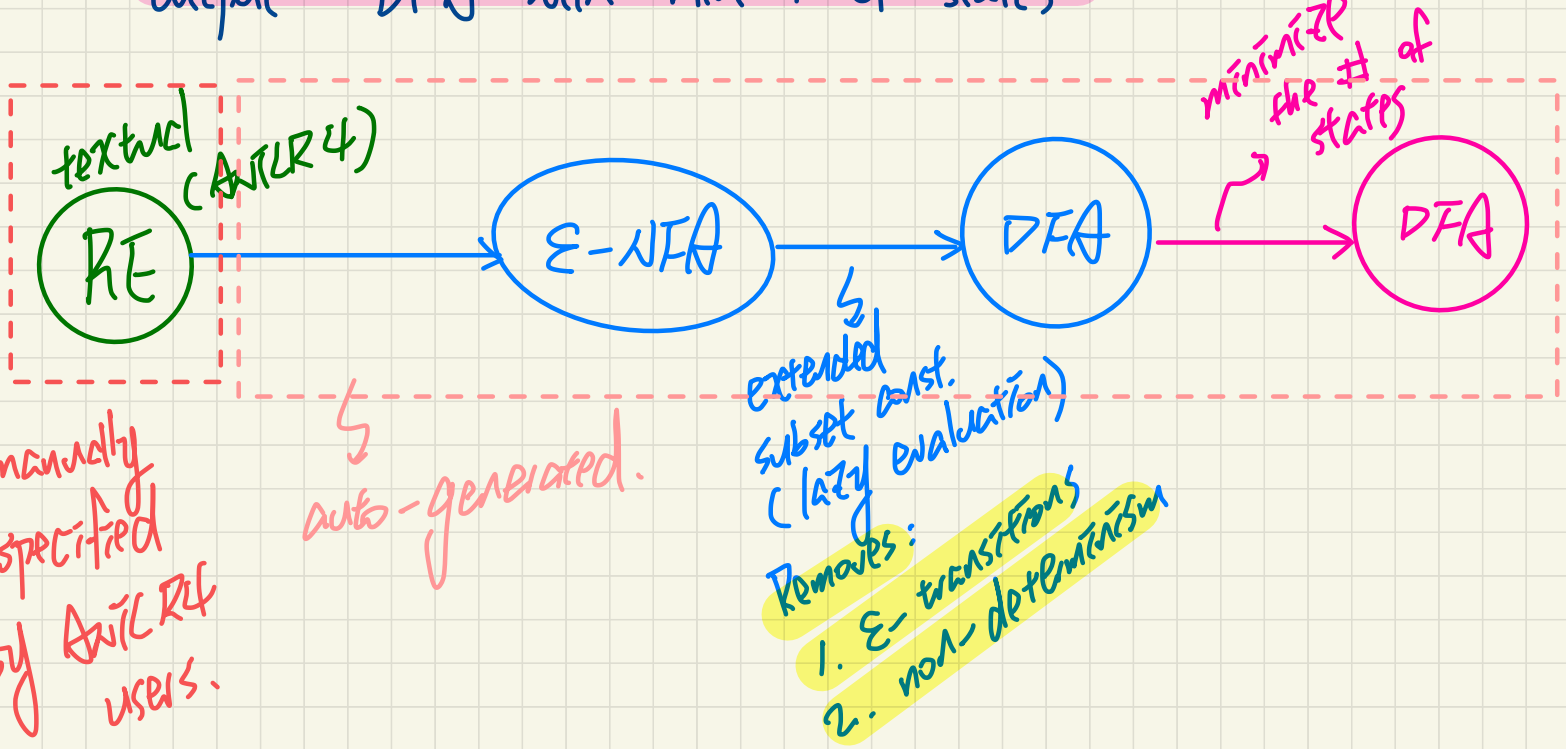
- **Assignment 1** due next Monday (June 16)
- **Assignment 2** to be released (June 16)
- **Required** for A2 and Project: ANTLR4 tutorial videos
- **Review Slides on Math** posted




Tracing Minimizing DFA
P → refined partitions so far
P=T means no further opt. possible.
≡
T → last refined partition
S
maximal splittable proper subset of states
start of 1st it.
{{q0, q1, q2, q4}, {q3, q5}}
∅
⊥
①
{{q0, q1, q2, q4}, {q3, q5}}
P
{{q2, q4}, {q0, q1}, {q3, q5}}
S
P-S
not splittable
{q2, q4}
S
{{q2, q4}, {q0, q1, q3}, {q5}}
P ≠ T
→ more opt. might be possible!
∅
⊥
②

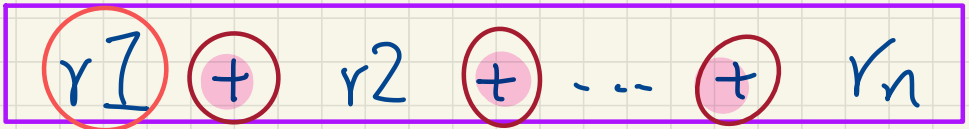
Input: Regular Expression

Output: DFA with min # of states



Scanner

n syntactic categories



r.e. for a particular category of tokens

a word recognized/accepted by either of the patterns is returned as a token

From RE to Scanner (1)

Regular Expression: r[0..9]+

Token Type (CharCat)

r	0, 1, 2, ..., 9	EOF	Other
Register	*Digit*	*Other*	*Other*

Transition

	Register	Digit	Other
s_0	s_1	s_e	s_e
s_1	s_e	s_2	s_e
s_2	s_e	s_2	s_e
s_e	s_e	s_e	s_e

Token Type (Type)

s_0	s_1	s_2	s_e
invalid	*invalid*	*register*	*invalid*

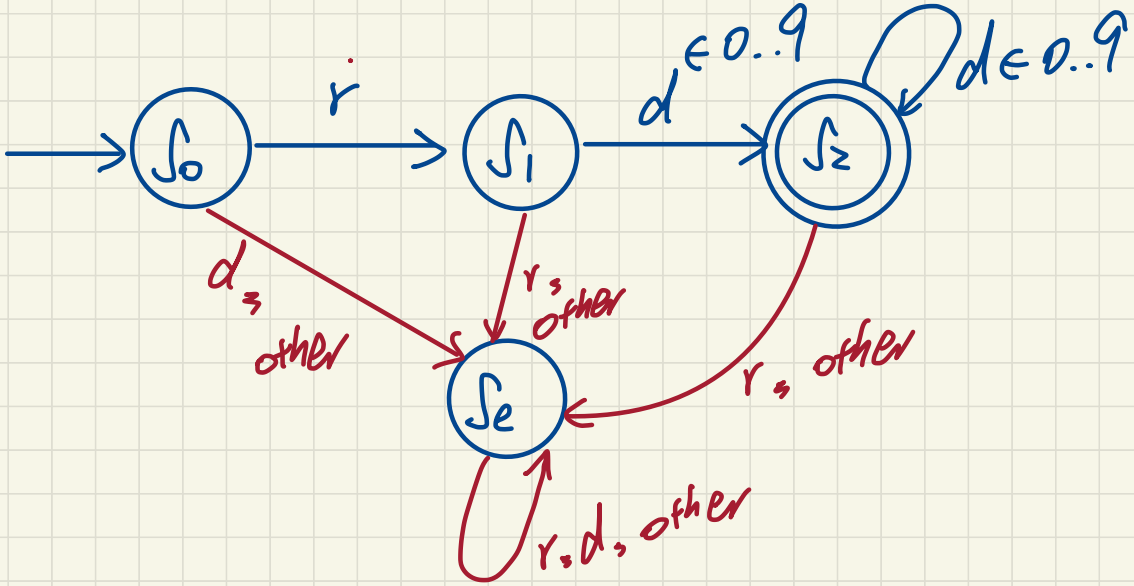
```
NextWord()
  -- Stage 1:  Initialization
  state := s0 ; word := ϵ
  initialize an empty stack s ; s.push(bad)
  -- Stage 2:  Scanning Loop
  while (state ≠ se)
    NextChar(char) ; word := word + char
    if state ∈ F then reset stack s end
    s.push(state)
    cat := CharCat[char]
    state := δ[state, cat]
  -- Stage 3:  Rollback Loop
  while (state ∉ F ∧ state ≠ bad)
    state := s.pop()
    truncate word
  -- Stage 4:  Interpret and Report
  if state ∈ F then return Type[state]
  else return invalid
  end
```

Example input: r2 EOF

word: r r2

state: s0 s0 s2

cat: Register Digit


s_0
s_1
s_2
s_e
r
d ∈ 0..9
d ∈ 0..9
d, other
r, other
r, other
r, d, other

From RE to Scanner (2)

Regular Expression: r[0..9]+

Token Type (CharCat)

r	0, 1, 2, ..., 9	EOF	Other
Register	*Digit*	*Other*	*Other*

Transition

	Register	Digit	Other
s_0	s_1	s_e	s_e
s_1	s_e	s_2	s_e
s_2	s_e	s_2	s_e
s_e	s_e	s_e	s_e

Token Type (Type)

s_0	s_1	s_2	s_e
invalid	*invalid*	*register*	*invalid*

```
NextWord()
  -- Stage 1:  Initialization
  state := s0 ; word := ϵ
  initialize an empty stack s ; s.push(bad)
  -- Stage 2:  Scanning Loop
  while (state ≠ se)
    NextChar(char) ; word := word + char
    if state ∈ F then reset stack s end
    s.push(state)
    cat := CharCat[char]
    state := δ[state, cat]
  -- Stage 3:  Rollback Loop
  while (state ∉ F ∧ state ≠ bad)
    state := s.pop()
    truncate word
  -- Stage 4:  Interpret and Report
  if state ∈ F then return Type[state]
  else return invalid
  end
```

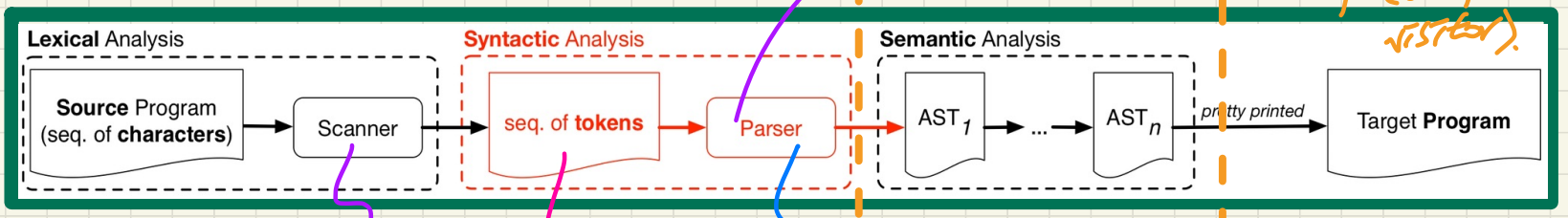
Example input: r24*3

word:

state:

cat:

Parser in Context



focus: tokens rather than characters.

design patterns (composite, visitor).

r1 + r2 + ... + rn

each token has a category (e.g. Register, Digit)

linear seq. of tokens

t1 t2 ... tn

top-down parsing or bottom-up parsing

non-linear ASTs

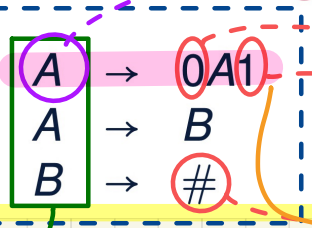
1. CFG
2. derivations
3. PTs

Context-Free Grammar (CFG): Terminology

The following language that is ***non-regular*** cannot be expressed/recognized by any RE/NFA/ε-NFA/DFA

$$\{0^n\#1^n \mid n \geq 0\}$$

can be described using a ***context-free grammar (CFG)***:



start variable

context-free grammar

↳ focus only on recursive pattern of strings

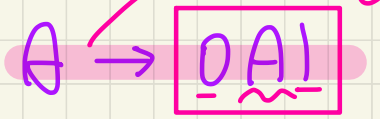
↳ no semantic analysis is done.

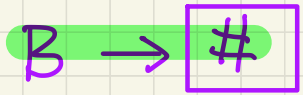
non-terminals (variables)

production rule.

terminals (tokens)

recursive case (more derivation steps expected)





base case (terminals only)