Compiler.

Project 1

team 9.
2017348 3 김희진

:-) define tokens.

① identifier ② keyword ③ seperator ④ operator
⑤ int literal ⑥ boolean literal
⑦ char literal ⑧ String literal

① Identifier. = id_start (id_start | numeric)*

id_start = alphabetic | _

alphabetic = (lower_case | upper_case)+

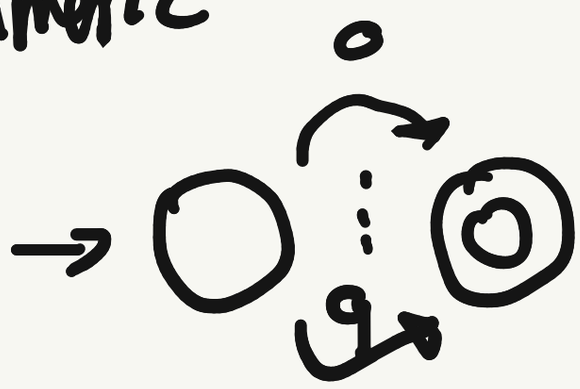
lower_case = a | b | ··· | y | z

upper_case = A | B | ··· | Y | Z

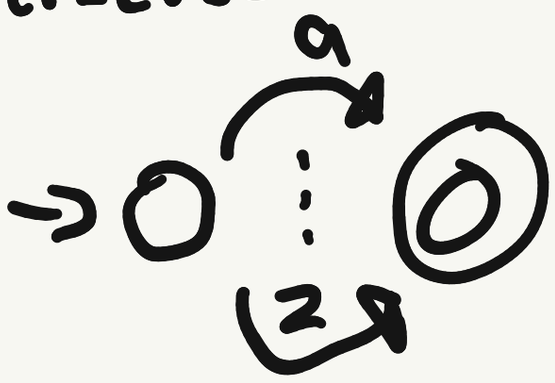
numeric = 0 | 1 | ··· | 8 | 9

DFA

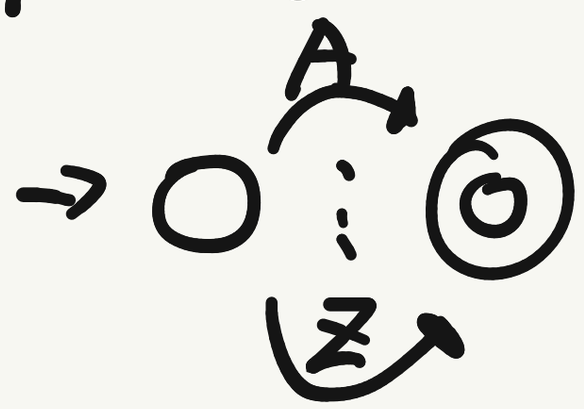
numeric



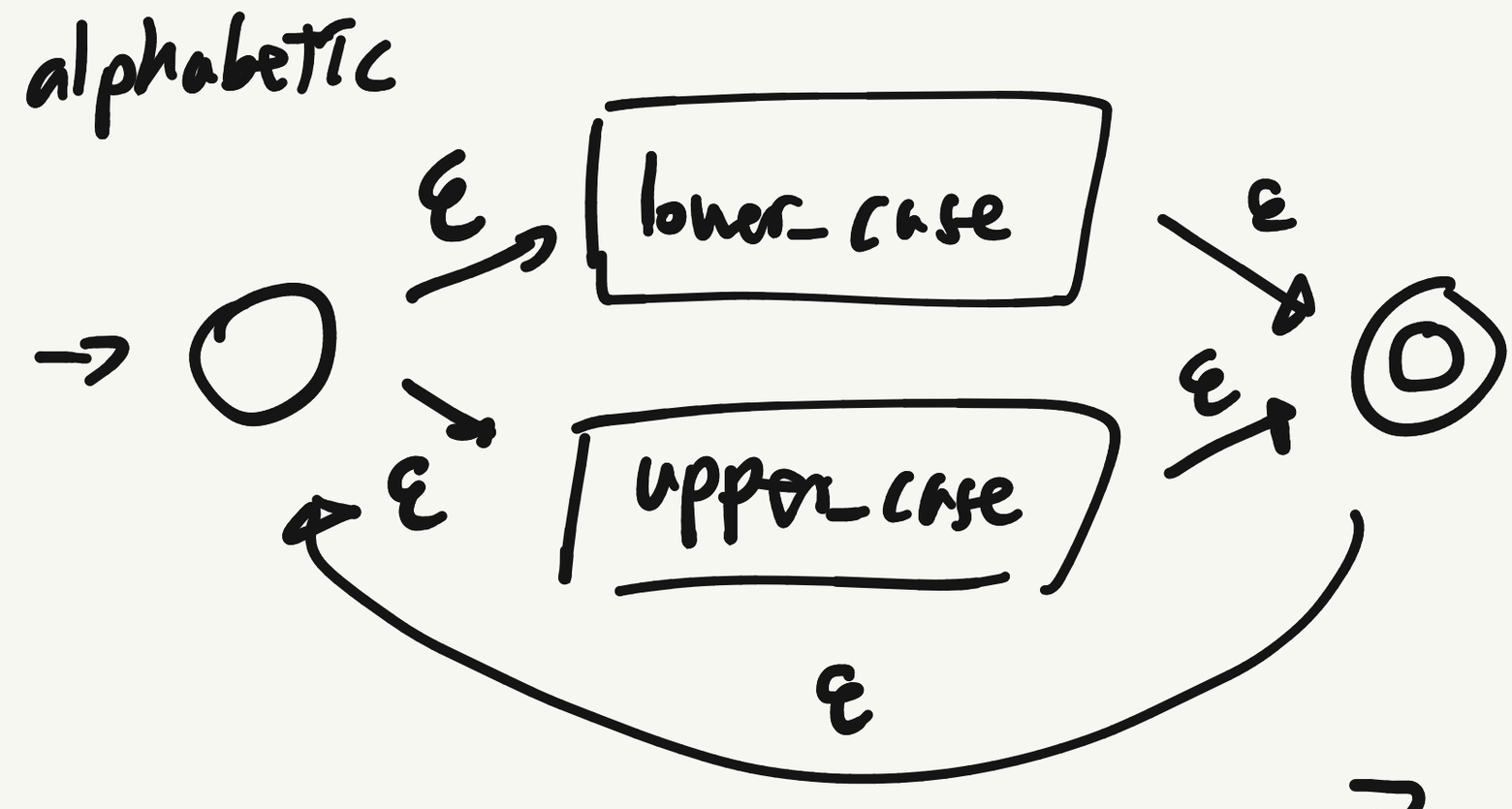
lower_case



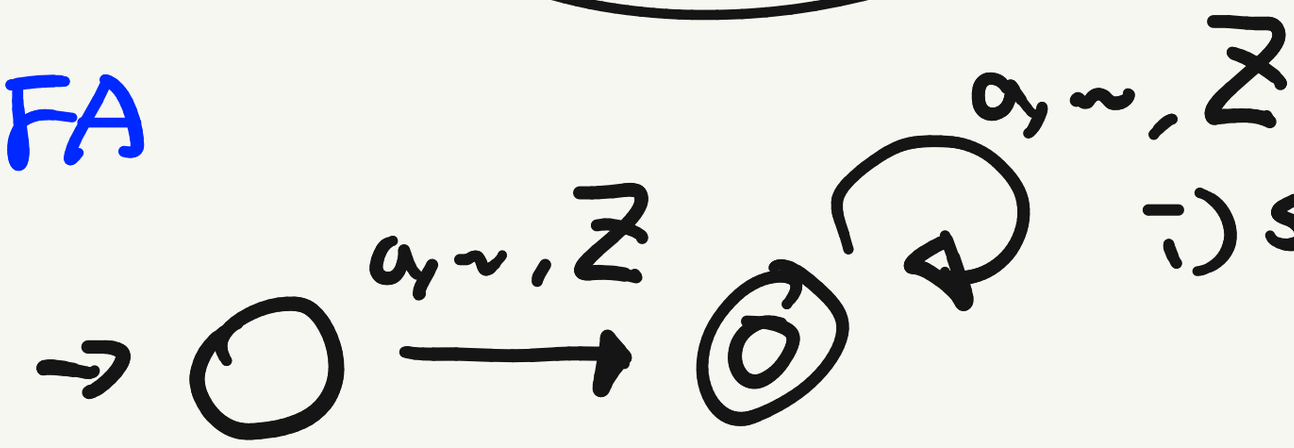
upper_case



NFA alphabetic



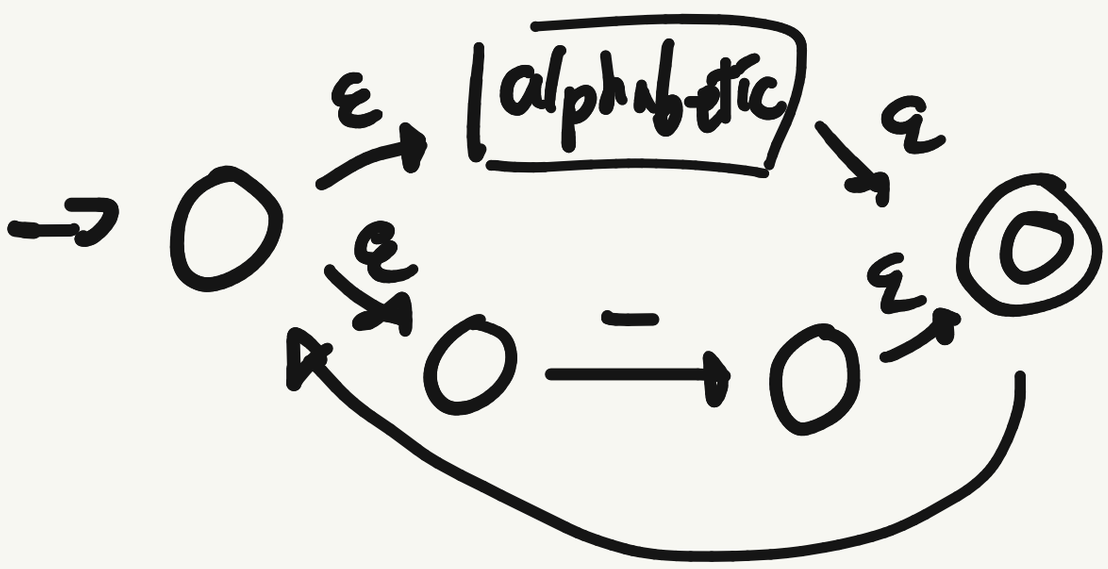
⇒DFA



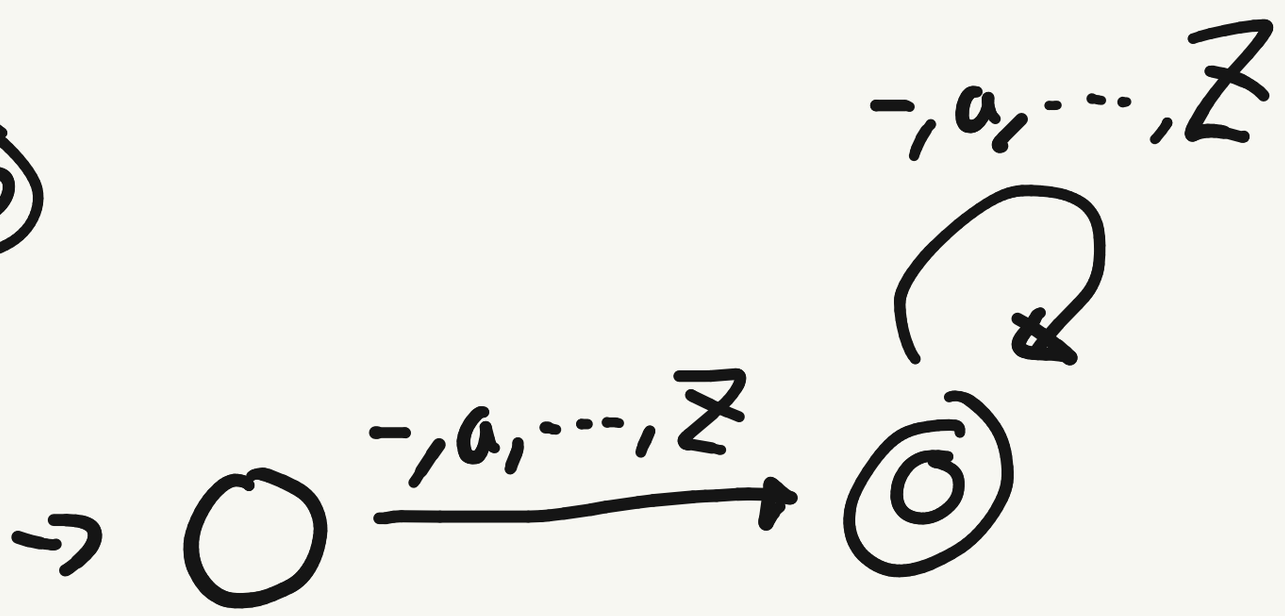
i) start state가 a~Z 중 하나라면 Final state 가능.

ii) 두번째부터 a~Z 혹은 처음과 a~Z는 동일

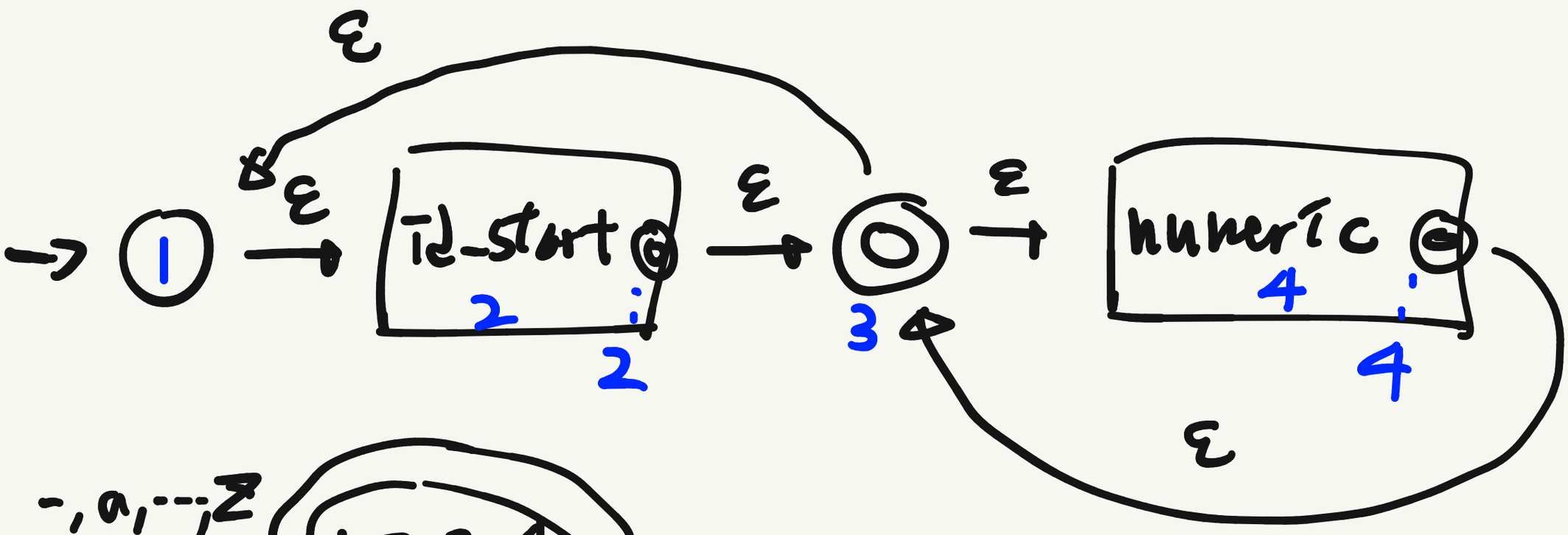
NFA Id_start

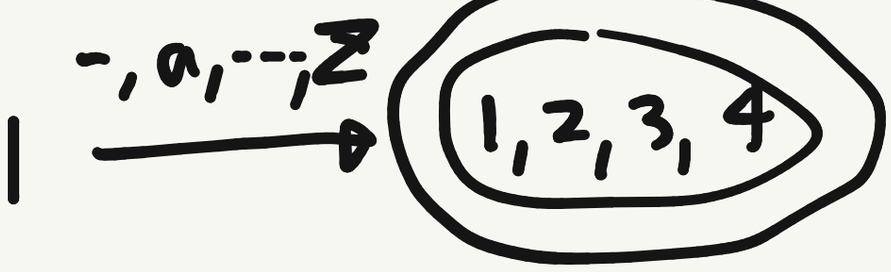


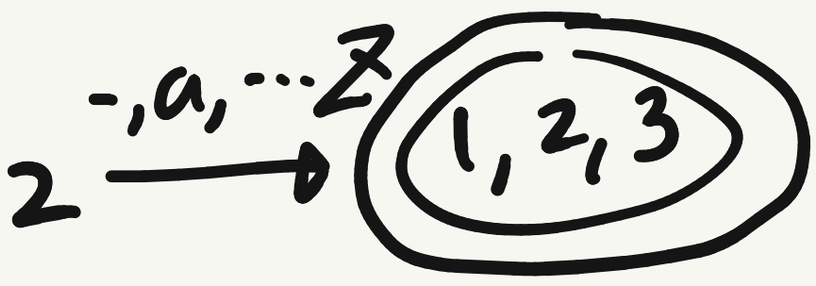
DFA Id_start

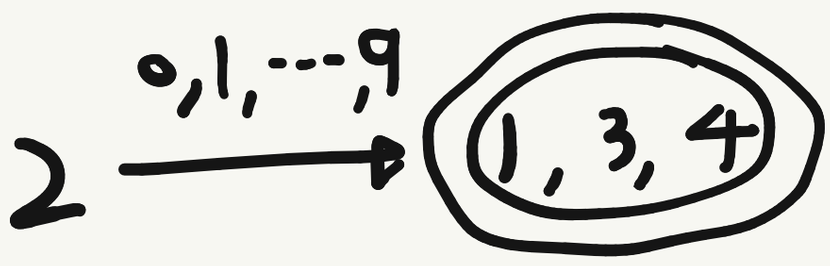


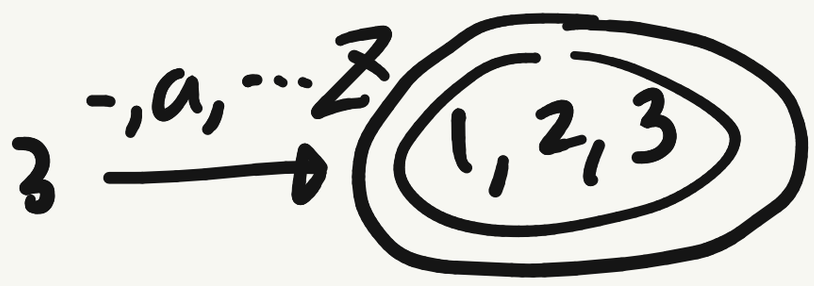
NFA identifier

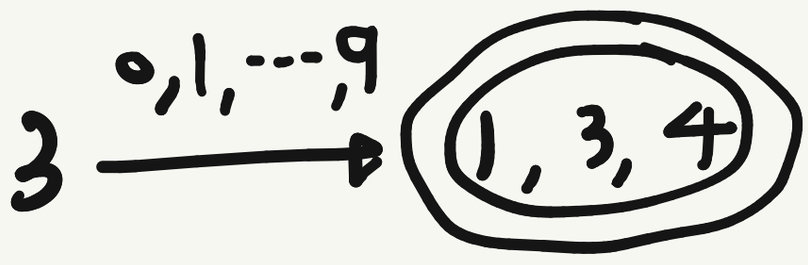


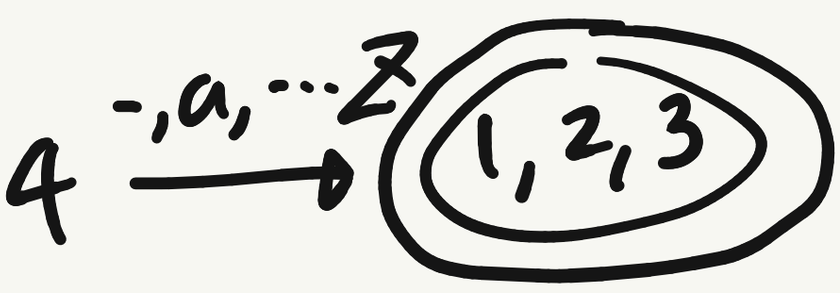


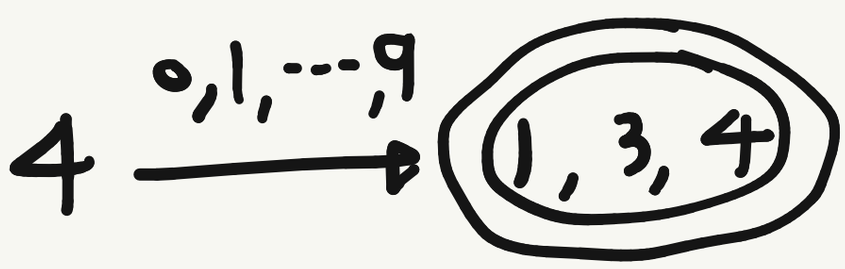












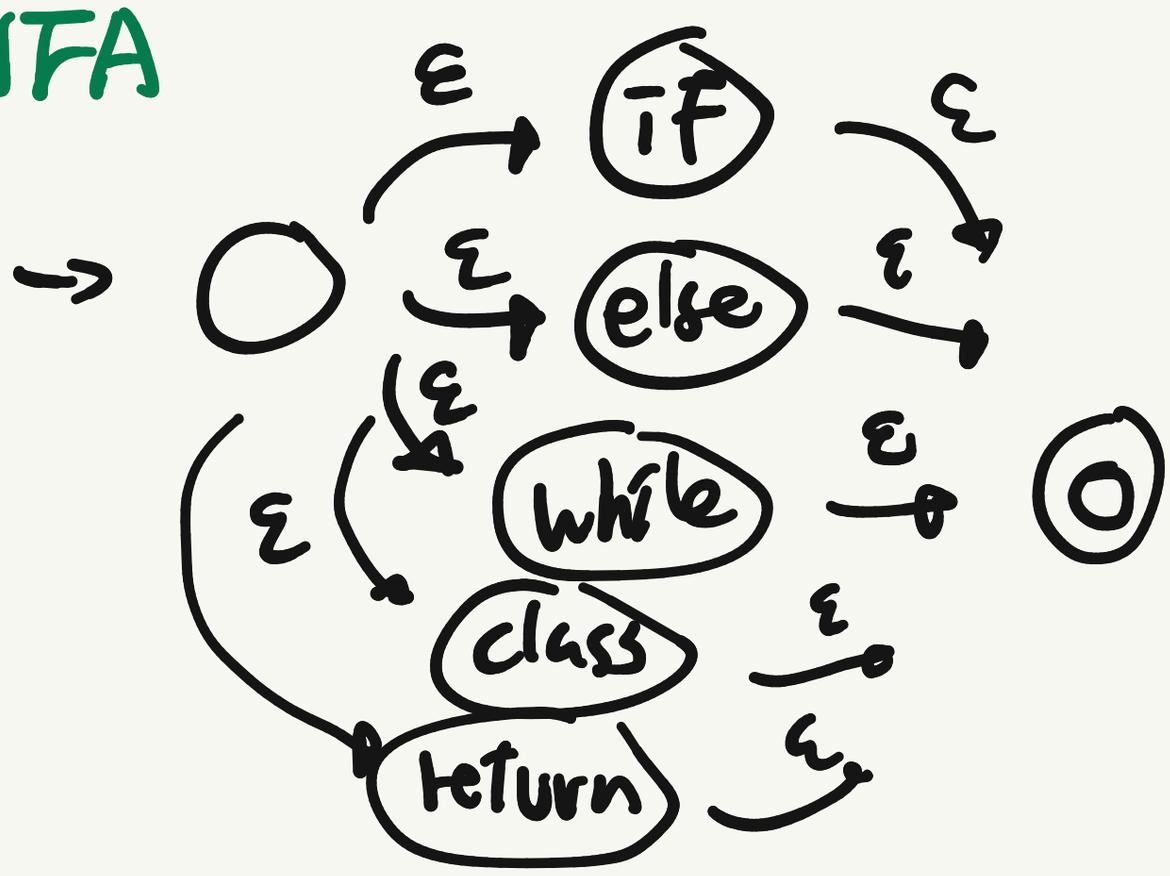
DFA identifier.

② Keyword

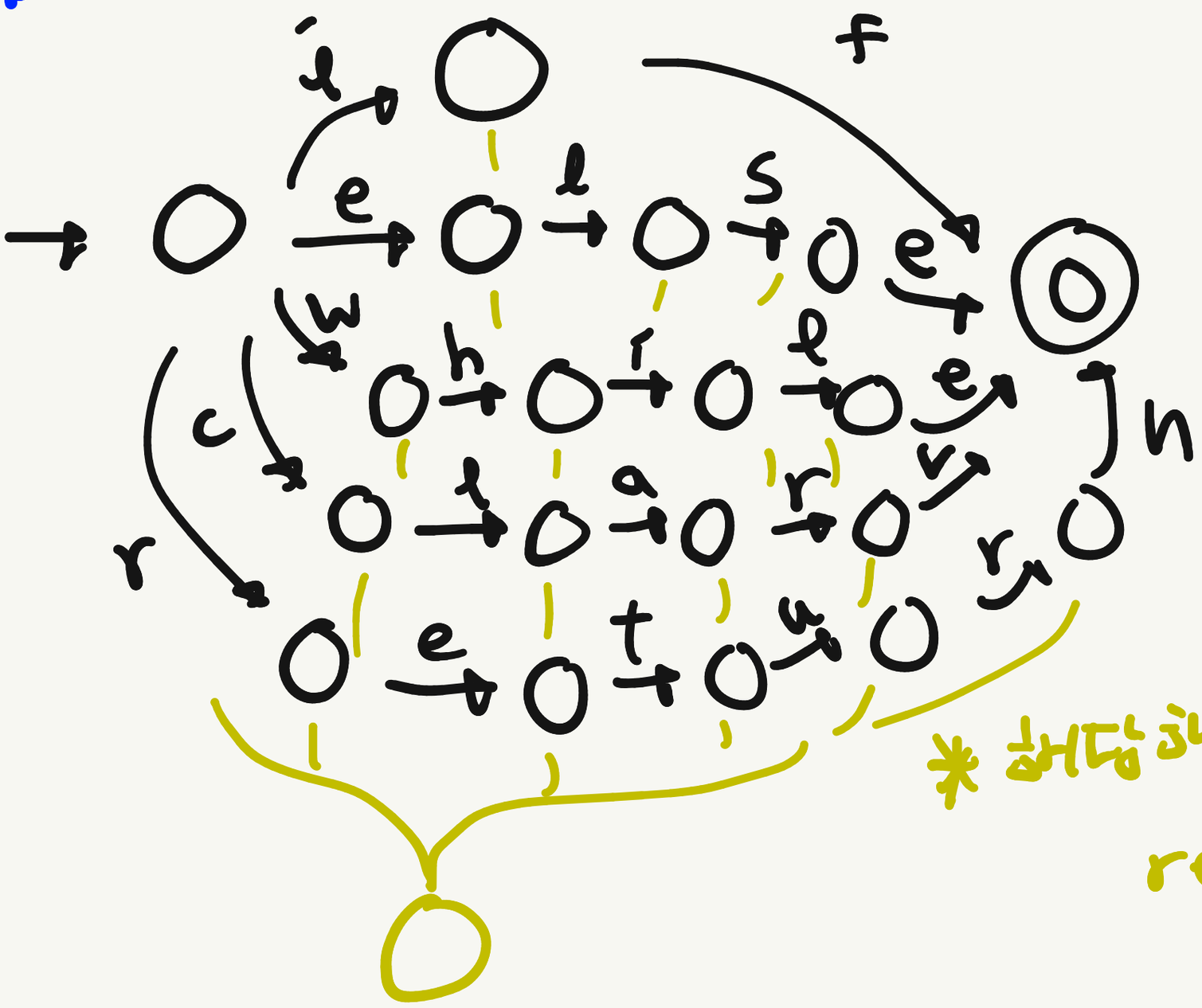
= if | else | while | class | return

$\Sigma = \{a, b, \cdots, z\}$

NFA



DFA



* 해당하지않는 모든 알파벳은 reject

③ Seperator

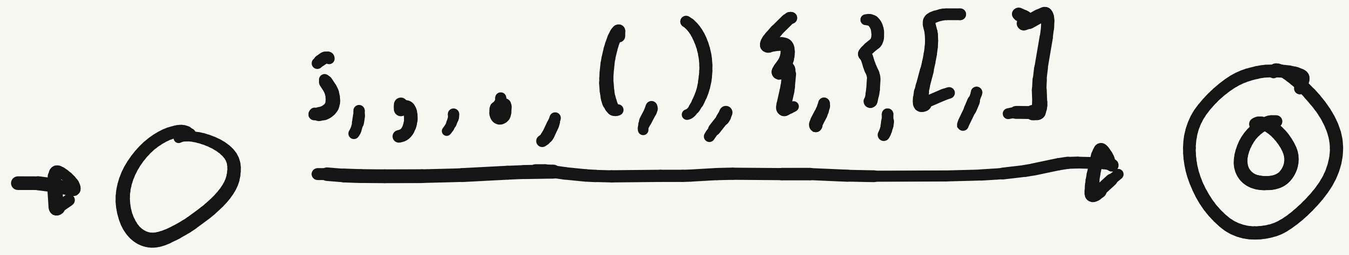
= ; | , | . | (|) | [|] | { | }

토큰의 일종인 토큰과 토큰을 구별하여주는 구별자 토큰.

ex) func(arg1, arg2) {
a = b;

}

DFA



각각의 이름을 대신하여 토큰 명을 지정 할수 있다.

ex) (: <Seperator, (>

=> <left_Paren, (>

paren, brace, bracket

④ operator

= Assign | Arithmetic | Comparison

Assign = =

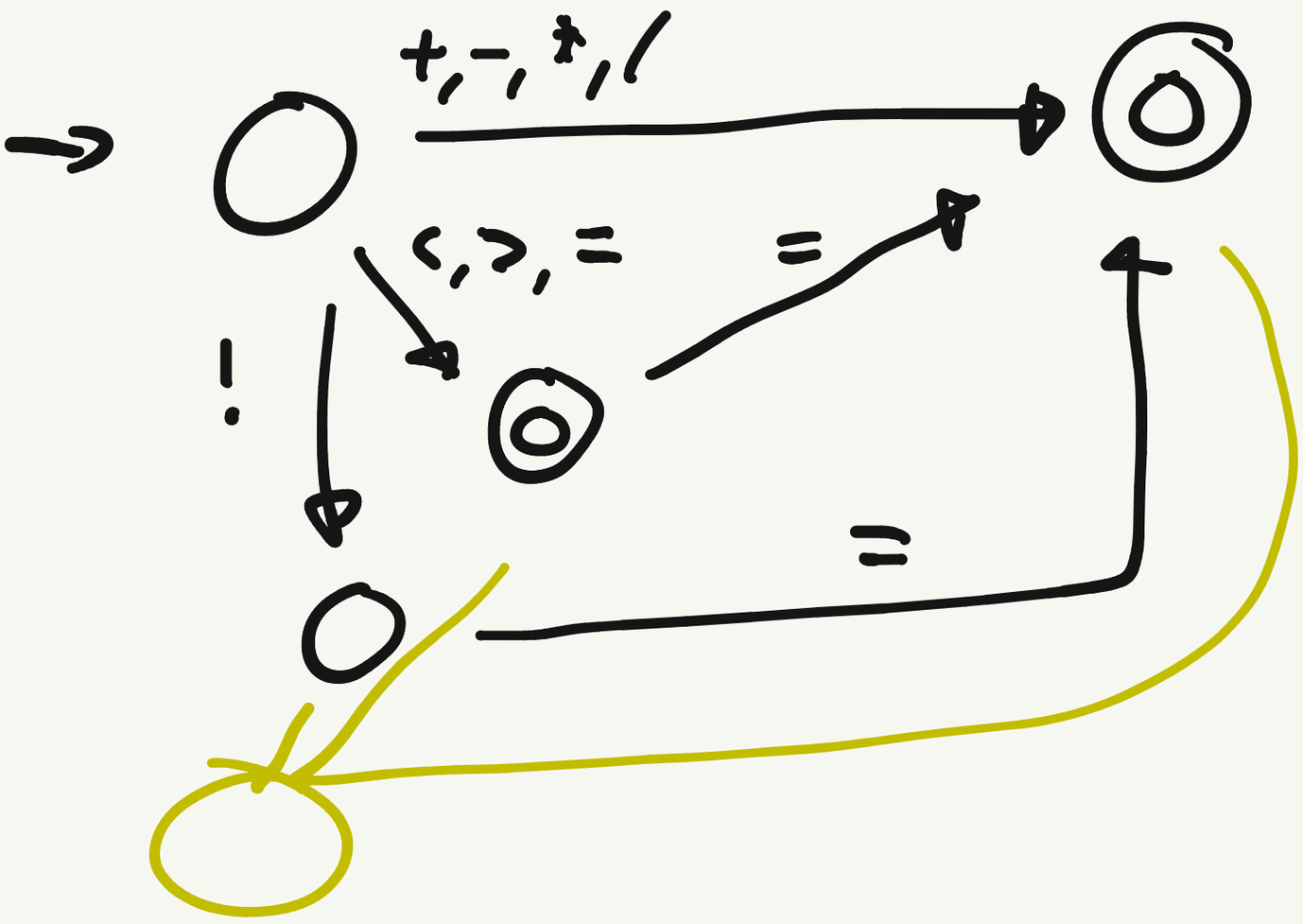
Arithmetic = + | - | * | /

Comparison = < | > | <= | >= | == | !=

operator token으로 동작하는 이유 :

소스코드 비슷하긴, 구문분석기를 통해 문법상 맞을 것을 확인할 생각함.

DFA $\Sigma = \{+, -, *, /, =, !, >, <\}$



⑤ Int literal

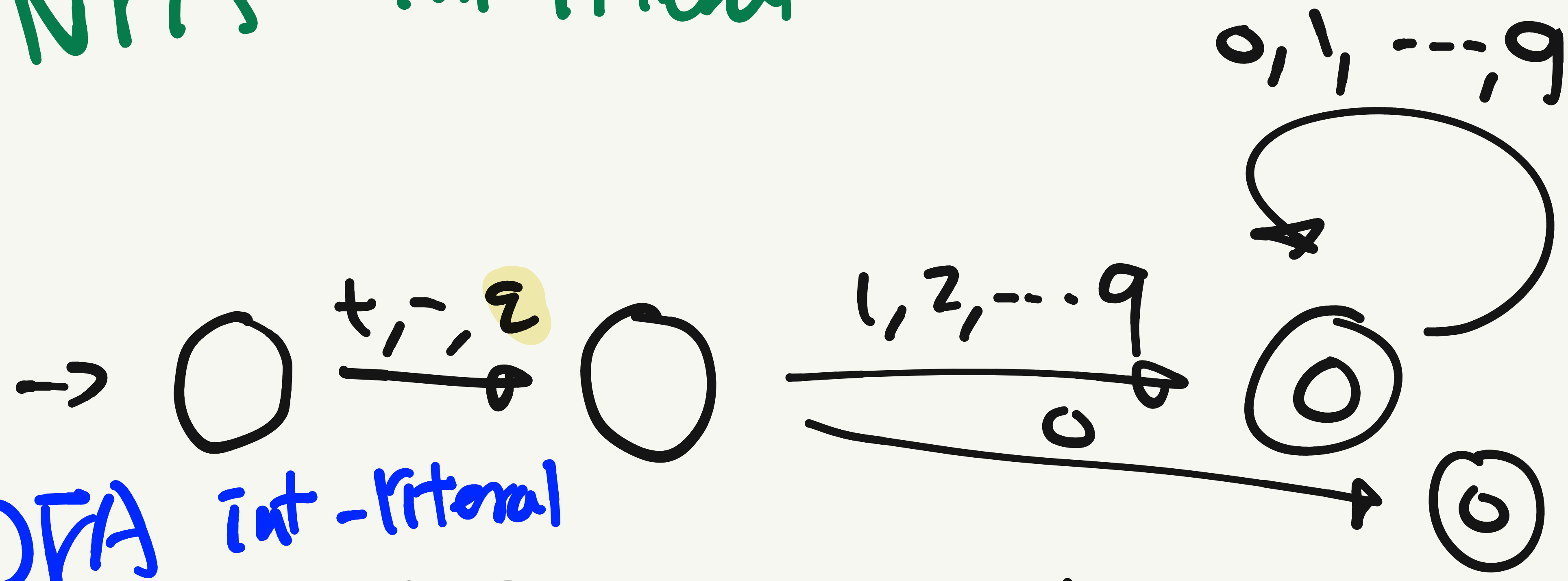
= (sign)(non-zero-digit)(digit)* | 0

sign = + | - | ε

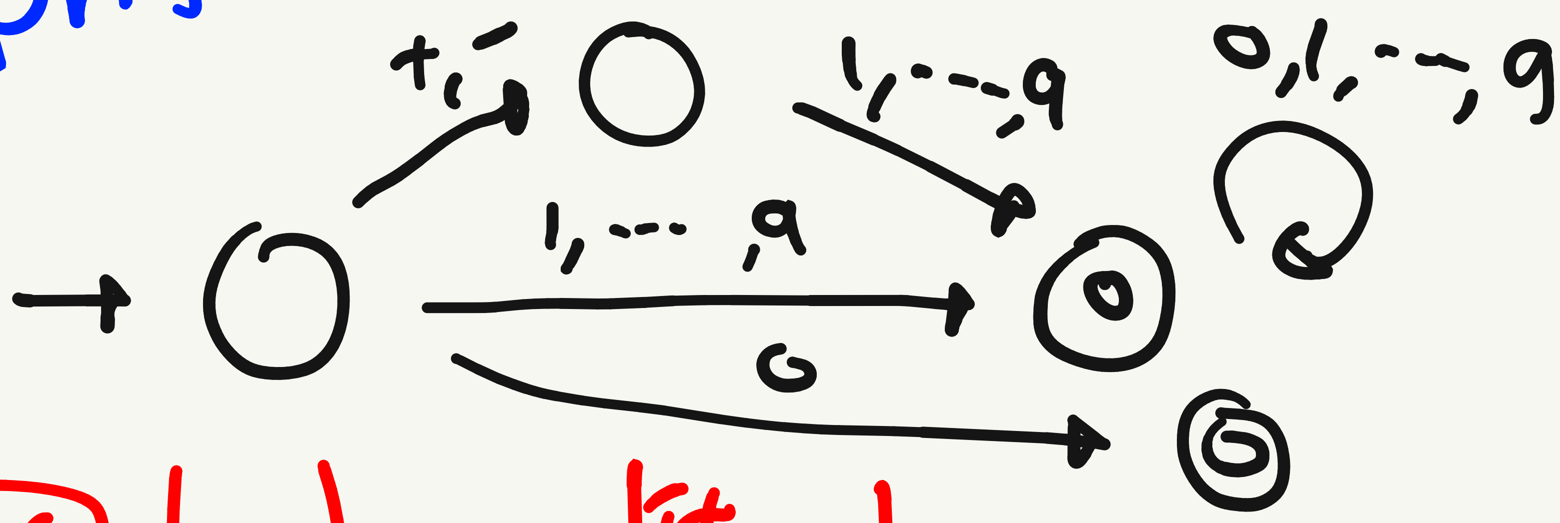
non-zero-digit = 1 | 2 | --- | 9

digit = 0 | non-zero-digit

NFA int-literal



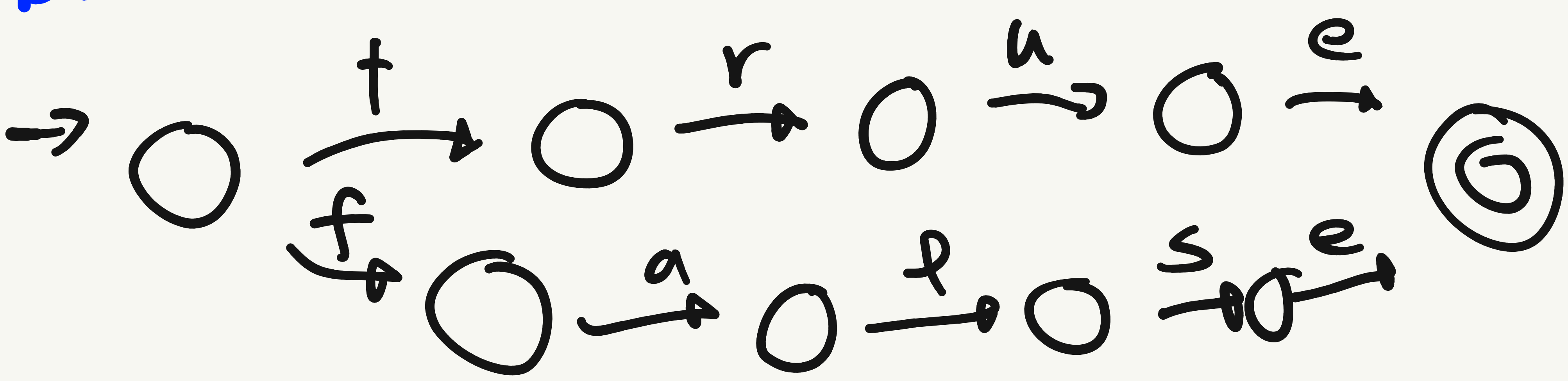
DFA int-literal



⑥ boolean-literal

= true | false (이 룰 리터럴로는 나머지 t/f 로 나타낼 수있는 표현은 이후에 고려한다)

DFA boolean-literal



⑦ char-literal

= 'lower_case' | 'upper-case' | 'space' | 'special' |

'escape-sequence'

space = 띄어쓰기

escape_sequnce = \n | \t | \v | \b | \r | \f | \a | \\ |

\' | \"

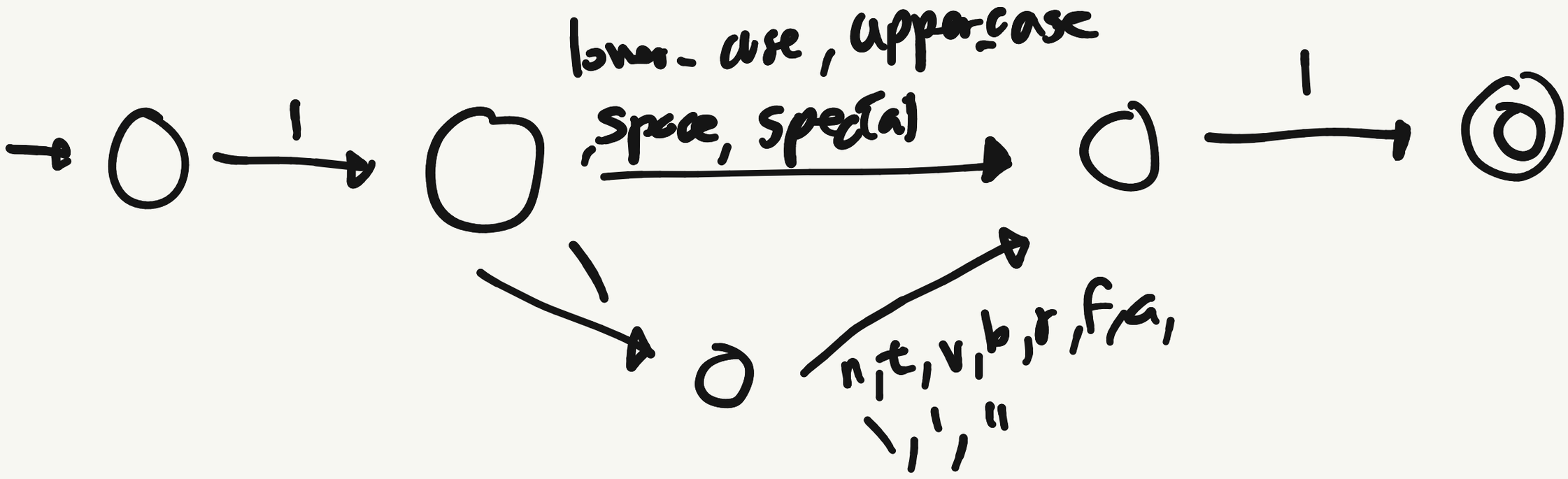
special = ! | % | ^ | & | * | (|) | [|] | { | } | - | + | = | | | ~ |

} | : | < | > | / | # | @ | ` | _

', ", \ 도 제약조 기호이나 | char에는 이미 포함이 되어있으므로,

', ", \ 는 escape-sequnce에서 작성 가능하다

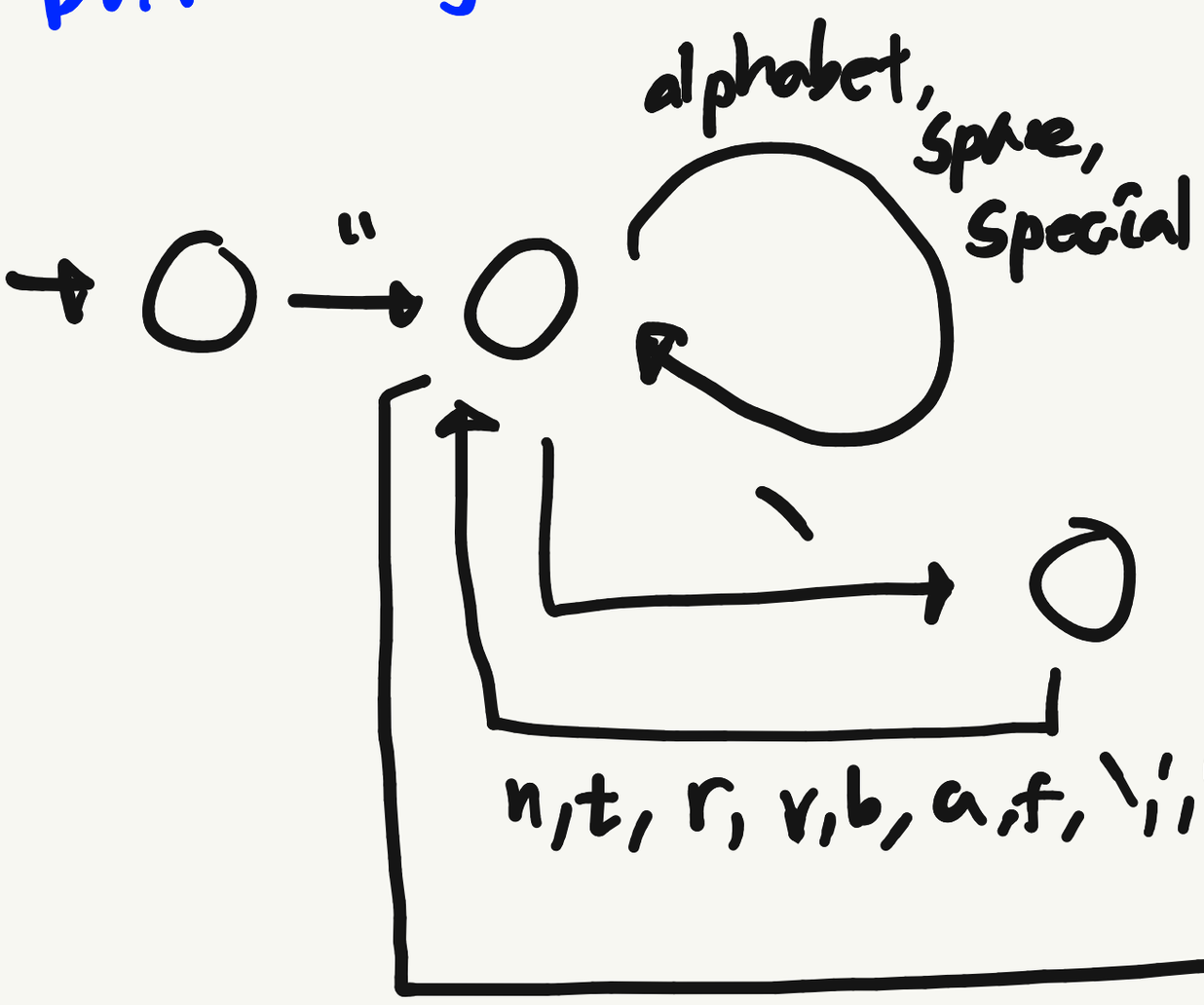
DFA char-literal



⑧ string-literal

= " (alphabet | space | special | escape-sequence)* "

DFA string-literal



⑨ Vtype

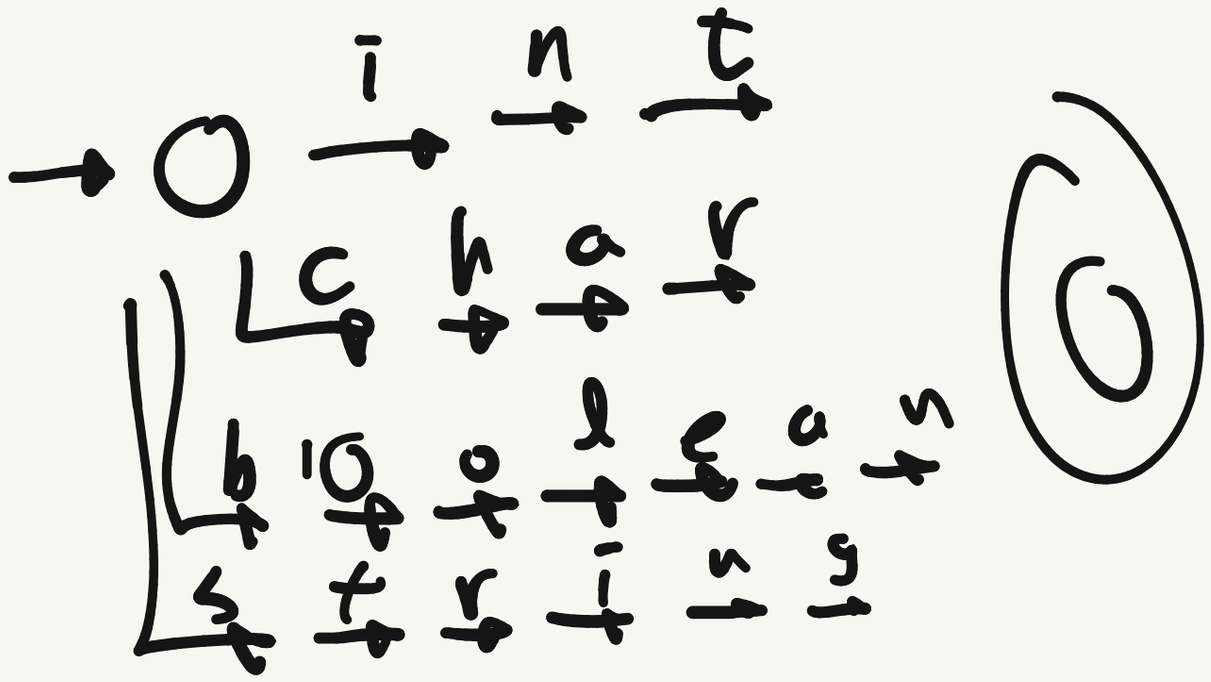
upper.

= int | char | boolean | String

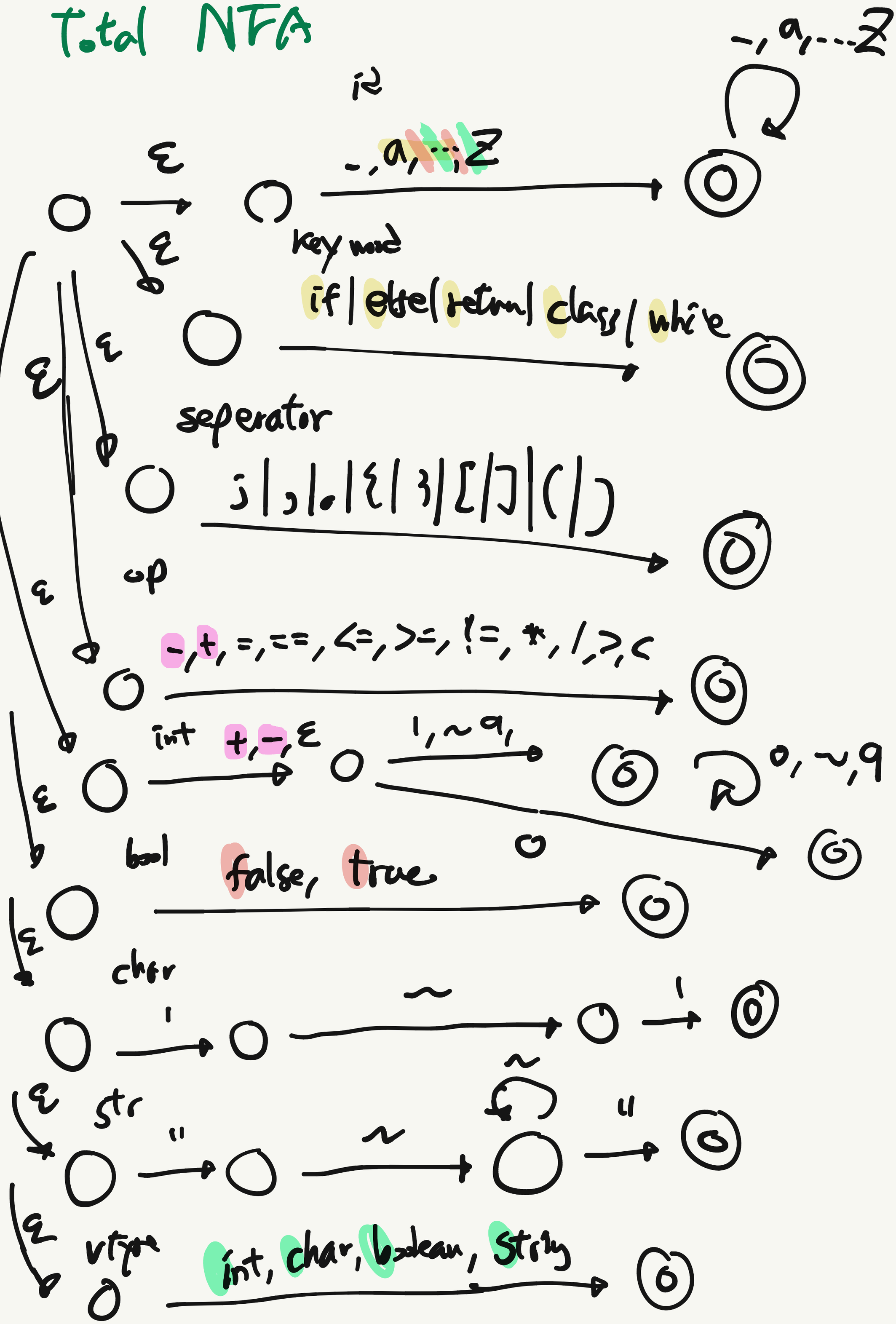
each DFA

Total DFA

DFA



Total NFA


ε
ε
_, a, ... z
key word
if | else | return | class | while
ε
ε
ε
seperator
; | , | . | { | } | [|] | (|)
op
ε
-, +, =, ==, <=, >=, !=, *, /, >, <
int
+, -, ε
1, ~ 9,
0, ~, 9
ε
bool
false, true
ε
char
'
~
'
ε
str
"
~
"
ε
virgo
int, char, boolean, String

겹치는 first symbol.

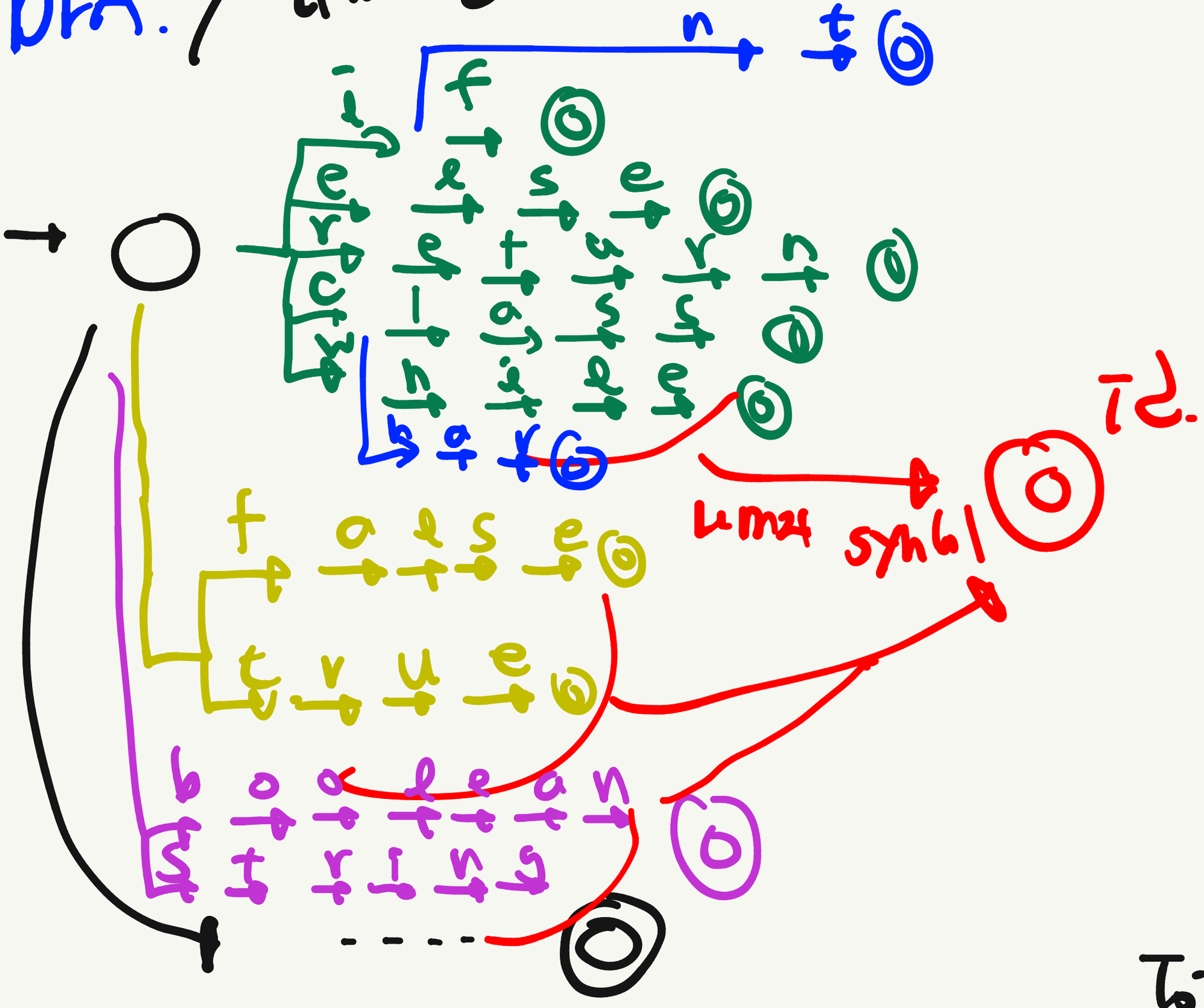
ID, K.W, VType => i, c

ID, K.W => w e r

ID, V.Type => b, S

ID, bool => f t

DFA. / 나머지는 ε 제거, 동일.



Total DFA

OP, Int => +, −

구분자: ① OP뒤의 +, −

② +, − 뒤의 Integer.

⇒ ①, ②를 만족하면 Int Sign으로 간주.

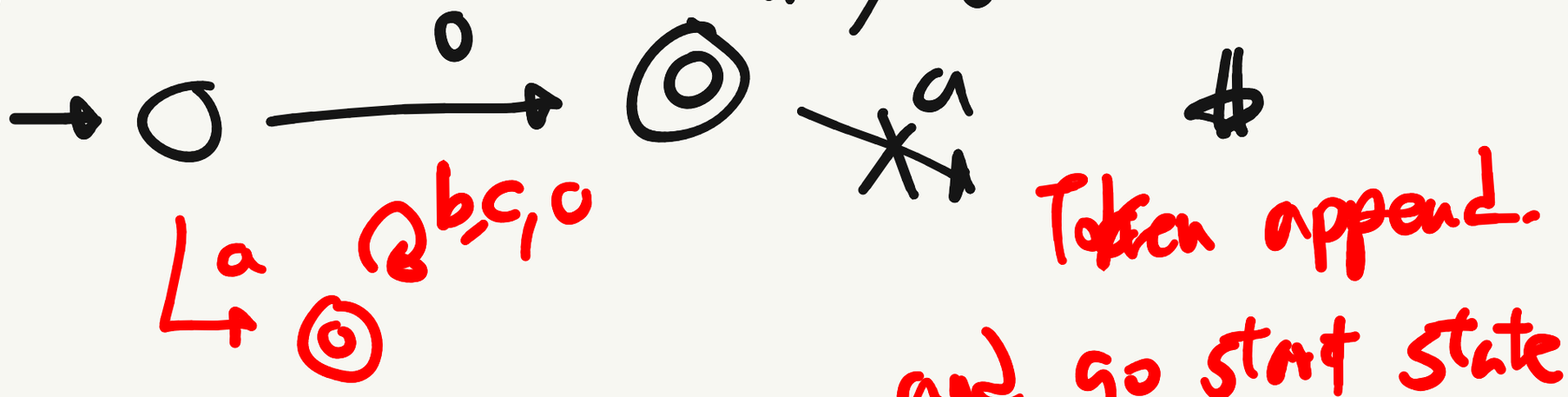
→ 구현 고려사항.

i) Input -0 ⇒ op, Int

⇒ Int token의 sign 붙이기.

ii) 0abc0 ⇒ Int 0, ID abc0

0abc0



ID.

⇒ 토큰을 append 하는 기준.

1. 다음 symbol 의 transition이 없음.

2. 다음 symbol 이 공백.

⇒ 이걸 detect.

1. 다음 symbol 이 transition 이 없을 경우.

또는 separator U whitespace 경우.

ex) 00(⇒ 0 int, 0 int, (int

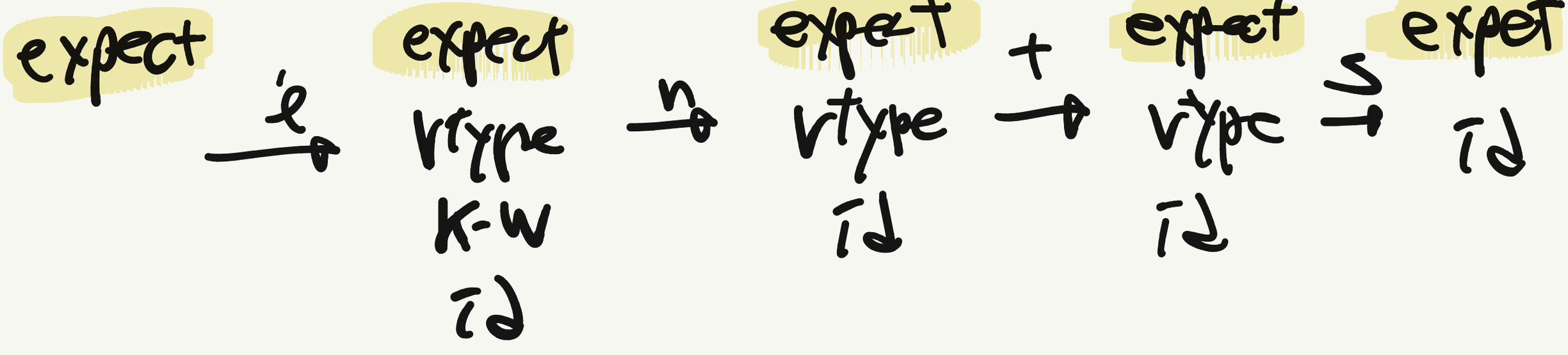
error : 00 ⇒ 올바르지 않은 input.

'x'123 ⇒ 'x' char, 123 int

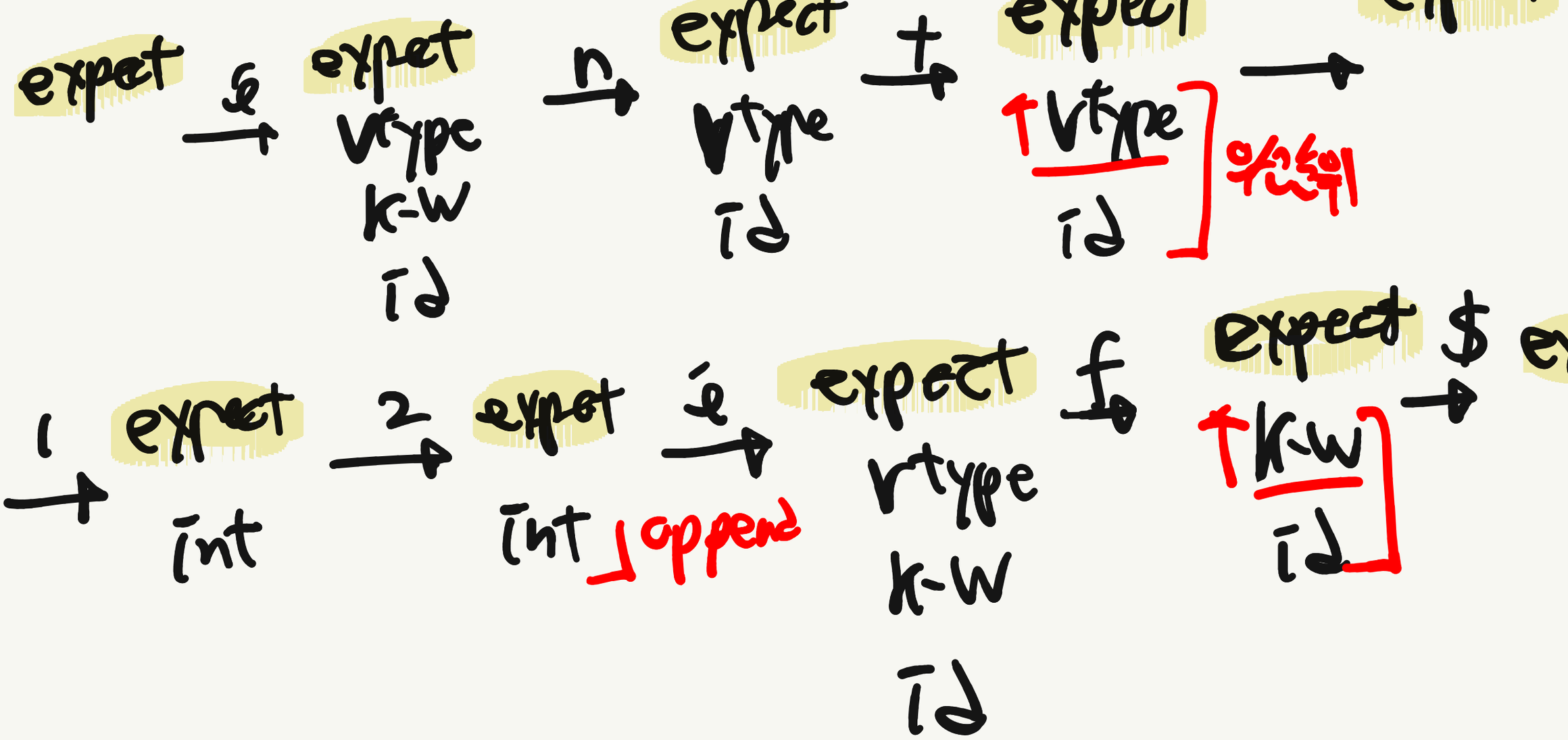
error : white로 구분 안했음.

구현 예시.

① input : int$



② input : int 12if$



∴ 반환되는 expect 의 index 는 달라지지 않는다.