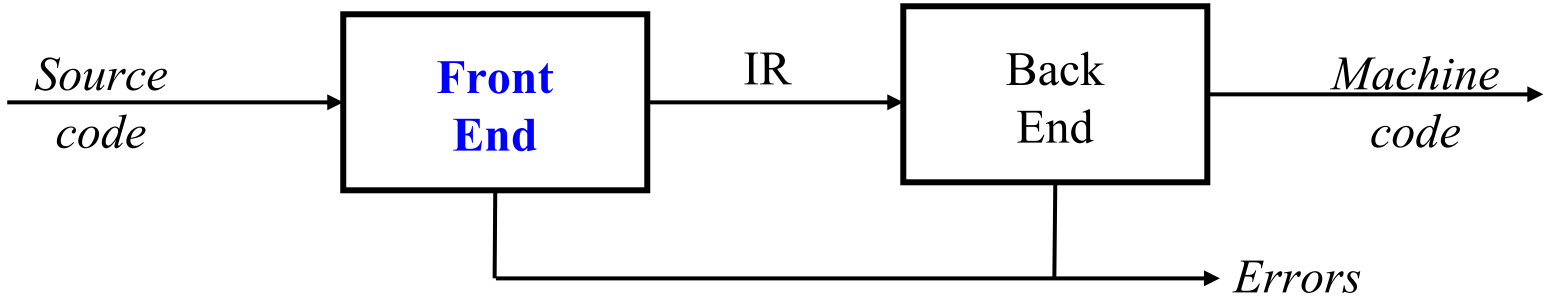
Chap 2. Scanning

COMP321 컴파일러

2007년 가을학기

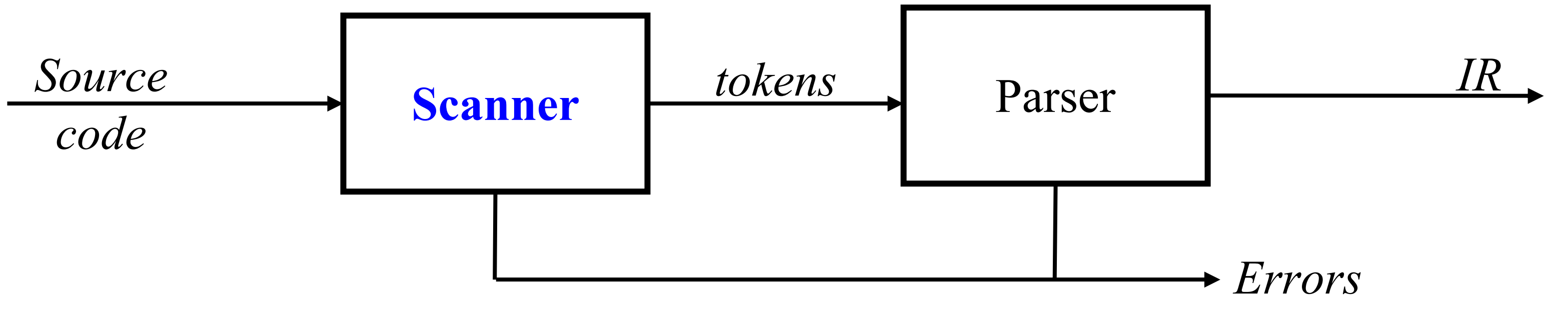
경북대학교 전자전기컴퓨터학부

The Front End



- The purpose of the front end is to deal with the input language
  - **syntax check**: code ∈ source language?
  - **semantics check**:
    - Is the program well-formed (semantically) ?
  - Build an **IR version** of the code for the rest of the compiler

Scanner



- Maps stream of characters into **words**
- **token**: basic unit of syntax
- *x* = *x* + *y* ;   becomes
  <id, *x*> <eq, => <id, *x*> <pl, +> <id, *y*> <sc, ; >
- Scanner **discards white space & (often) comments**

2.1 Introduction

Scanner

- also known as **lexical analyzer**
  - a stream of characters → a stream of words (**tokens**)

- 궁극적인 문제는 **pattern matching**
  - regular expression : pattern 표현 방법
  - lexical analysis : pattern matching 수행

- 응용 분야
  - UNIX grep command
  - Web search
  - find in word processors

왜 scanner 를 분리하는가?

- scanner / parser를 분리하는 이유
  - blank, new line, comment 제거를 전담
  - lexical rule을 적용해서 automation 가능
    - automata 이론 적용에 편리
  - **parser의 부담을 줄인다.**
    - parser는 syntax check만 해도 heavy-weighted !

1.	*goal*	$\rightarrow$	*expr*	
2.	*expr*	$\rightarrow$	*expr* *op* *term*	
3.		|	*term*	
4.	*term*	$\rightarrow$	<u>number</u>	
5.		|	<u>id</u>	
6.	*op*	$\rightarrow$	+	
7.		|	–	

parser 가 할 일

<u>number</u> → 0 | 1 | 2 | … | 9
| 1 number
| 2 number
| …
| 9 number

<u>id</u> → …

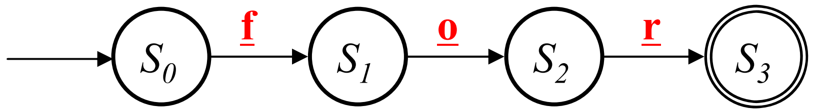
scanner가 해 줄 수 있는 일

2.2 Recognizing Words

Hand-Written Scanner

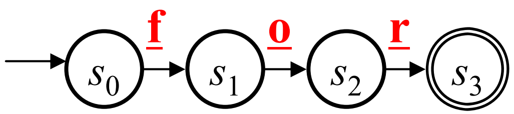
- for the word "for"
  - *NextChar*( ) : **function** to input the next character

- c ← NextChar( )
  if (c ≠ 'f')
    then *do something else*
    else
      c ← NextChar( )
      if (c ≠ 'o')
        then *do something else*
        else
          c ← NextChar( )
          if (c ≠ 'r')
            then *do something else*
            else *report success*

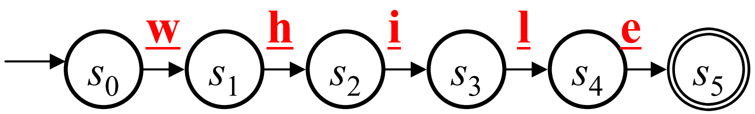


Hand-Written Scanners

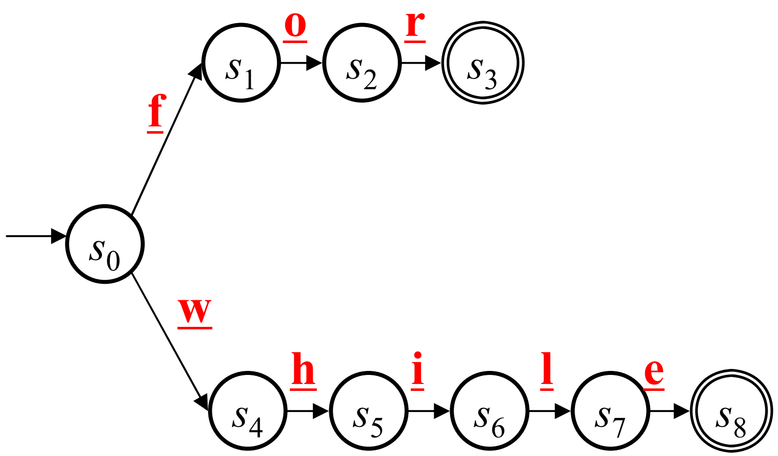
- "for" case



- "while" case

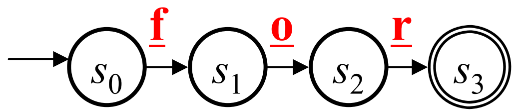


- "for" and "while" case



Finite Automata

- transition diagram → more formalized → finite automata
  - FA와 transition diagram은 equivalent !
- FA is a five-tuple $(S, \Sigma, \delta, s_0, S_F)$
  - S is the set of states. (must be finite)
    - $S = \{ s_0, s_1, s_2, s_3 \}$

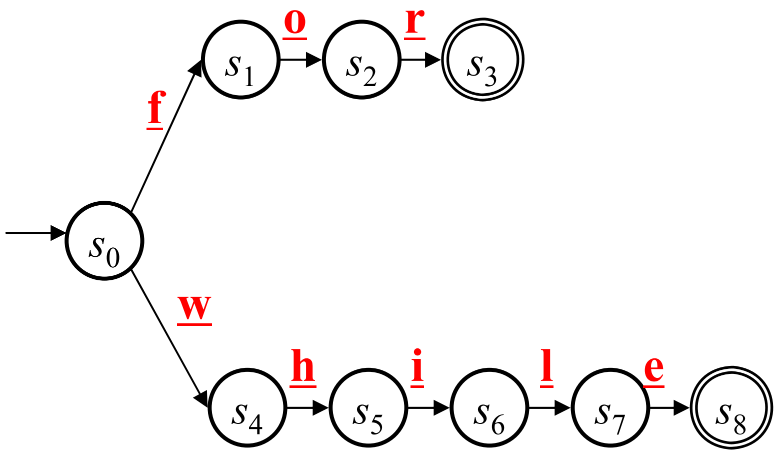


  - Σ is the alphabet. (must be finite)
    - $\Sigma = \{ \underline{f}, \underline{o}, \underline{r} \}$
  - $\delta(s, c)$ is a transition function
    - $\delta = \{ s_0 \xrightarrow{f} s_1, s_1 \xrightarrow{o} s_2, s_2 \xrightarrow{r} s_3 \}$
  - $s_0 \in S$ is the designated start state.
  - S_F is the set of final states. $S_F \subseteq S$
    - $S_F = \{ s_3 \}$

Finite Automata

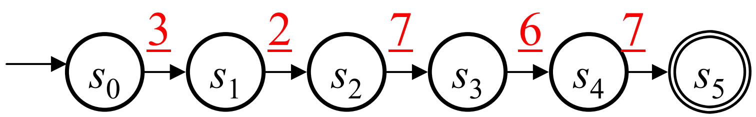
- another example
  - $S = \{ s_0, s_1, \ldots, s_8, s_e \}$
  - $\Sigma = \{ \underline{e}, \underline{f}, \underline{h}, \underline{i}, \underline{l}, \underline{o}, \underline{r}, \underline{w} \}$
  - $\delta = \{ s_0 \xrightarrow{\underline{f}} s_1, s_0 \xrightarrow{\underline{w}} s_4, \ldots \}$
  - s_0
  - $S_F = \{ s_3, s_8 \}$
- s_e : the designated error state
- FA accepts a word $x_1 x_2 x_3 \ldots x_n$
  - $\delta(\delta(\ldots\delta(\delta(s_0, x_1), x_2), \ldots, x_{n-1}), x_n) \in S_F$
- lexical error : $\delta(s_i, x_j)$ is undefined

  or the word ends at a non-final state.

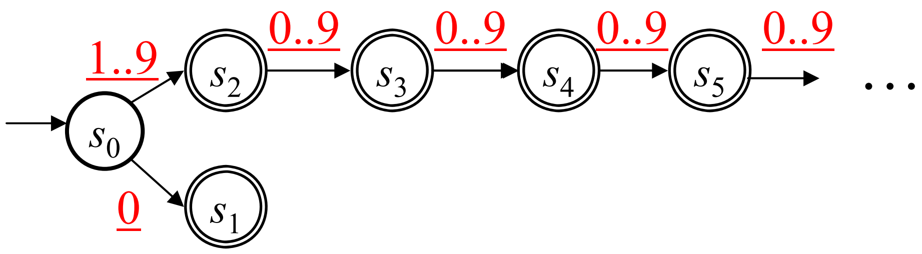


Recognizing Natural Numbers

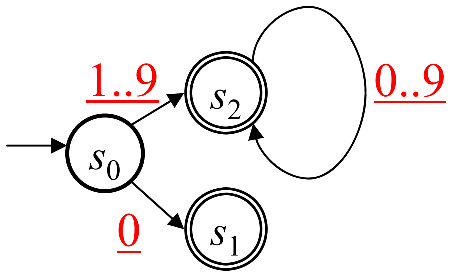
- a natural number, 32767



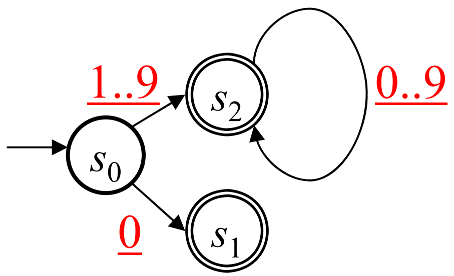
- any natural number : intuitive approach



- any natural number : correct answer with a cycle



Recognizing Natural Numbers



- $S = \{ s_0, s_1, s_2 \}$
- $\Sigma = \{ \underline{0}, \underline{1}, \underline{2}, \ldots, \underline{9} \}$
- $\delta = \{ s_0 \xrightarrow[\underline{0}]{} s_1,\ s_0 \xrightarrow[\underline{1..9}]{} s_2,\ s_2 \xrightarrow[\underline{0..9}]{} s_2 \}$
- $S_F = \{ s_1, s_2 \}$

δ	0	1..9	other
s_0	s_1	s_2	s_e
s_1	s_e	s_e	s_e
s_2	s_2	s_2	s_e
s_e	s_e	s_e	s_e

- implementation

ch ← NextChar( )
state ← s_0
while (ch ≠ eof **and** state ≠ s_e)
    state ← δ(state, ch)
    ch ← NextChar( )
end while
if (state ∈ S_F)
    then *report acceptance*
    else *report failure*

- automatic scanner construction ?
  - 가능 ! → next sections

2.3 Regular Expressions

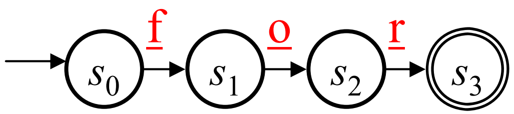
FA and RE

- F : a **finite automata** (FA)
- $L(F)$: a **language** accepted by an FA
  - the set of (all) words accepted by a FA
- RE : **regular expression** for an FA
  - $L(F)$ 를 표현하는 intuitive expression

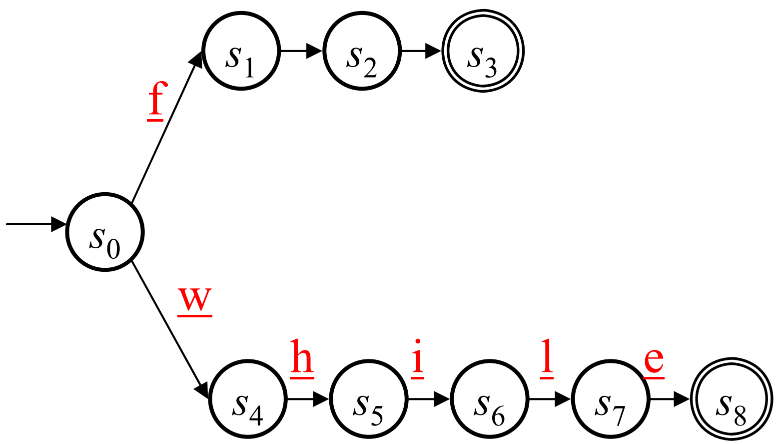
- $L(F)$를 정확하게 표현하는 방법 : FA itself
  - but, not intuitive, not efficient
- RE : $L(F)$를 직관적으로 표현하는 방법

RE Examples

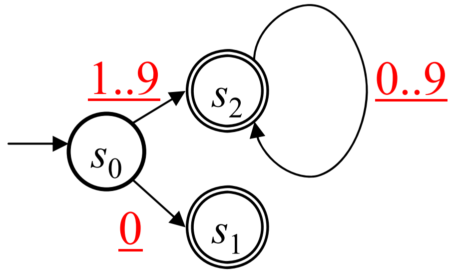
- RE : for



- RE : for | while



- RE ?



- 0 | ([1..9]) ([0..9])*
  - [1..9] = 1 | 2 | … | 8 | 9

- **Kleene closure *x****
  - zero or more occurrences of *x*

Regular Expression

- Σ is the **alphabet** augmented with empty string ε
- $L(r)$: the language accepted by a regular expression r
- ε is a RE denoting the set $\{ \varepsilon \}$
- If $\underline{a} \in \Sigma$, then $\underline{a}$ is a RE denoting $\{ \underline{a} \}$
- If x and y are REs denoting $L(x)$ and $L(y)$ then
  - (priority 3) alternation: $\boldsymbol{x \mid y}$ is an RE denoting $L(x) \cup L(y)$
  - (priority 2) concatenation : $\boldsymbol{xy}$ is an RE denoting $L(x)L(y)$
  - (priority 1) closure : $\boldsymbol{x^*}$ is an RE denoting $L(x)^*$
  - positive closure : $x^+ = xx^*$

$$x^* = \bigcup_{i=0}^{\infty} x^i \qquad x^+ = \bigcup_{i=1}^{\infty} x^i$$

RE Examples

Identifiers:

$Letter \rightarrow (\underline{a}|\underline{b}|\underline{c}| \dots |\underline{z}|\underline{A}|\underline{B}|\underline{C}| \dots |\underline{Z})$

$Digit \rightarrow (\underline{0}|\underline{1}|\underline{2}| \dots |\underline{9})$

$Identifier \rightarrow Letter\ (\ Letter \mid Digit\ )^*$

Numbers:

$Integer \rightarrow (\underline{+}|\underline{-}|\varepsilon)\ (\underline{0}|\ (\underline{1}|\underline{2}|\underline{3}| \dots |\underline{9})(Digit^*)\ )$

$Decimal \rightarrow Integer\ \underline{.}\ Digit^*$

$Real \rightarrow (\ Integer \mid Decimal\ )\ \underline{E}\ (\underline{+}|\underline{-}|\varepsilon)\ Digit^*$

$Complex \rightarrow \underline{(}\ Real\ \underline{,}\ Real\ \underline{)}$

Numbers can get much more complicated!

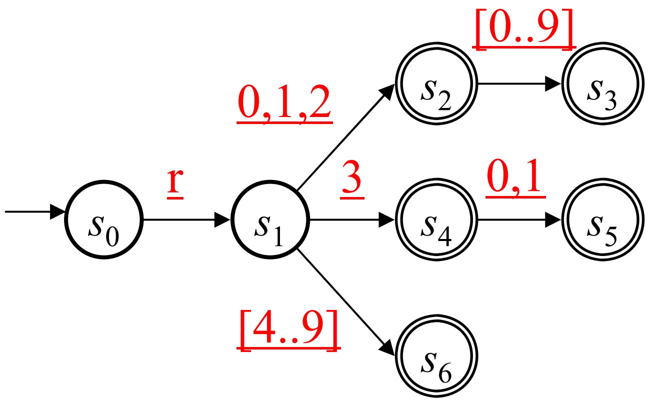
RE Examples

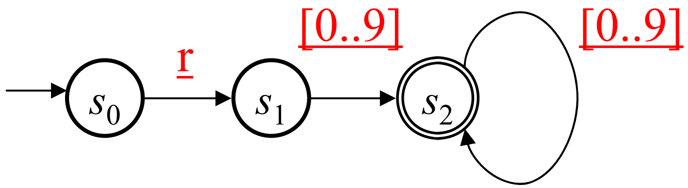
- quoted character string : " 와 " 로 묶인 string
  - [^c] : character c, ε 을 제외한 모든 alphabet
    → an example RE : "[^"]*"

- C string : \" 가능 !   → *complex RE ...*
- line comment : // 로 시작, \n 으로 끝
  → RE : //[^\n]*
- C-style comment : /* 로 시작, */ 로 끝
  → complex RE …

Limits of RE's

- we have 32 registers: r0, r1, r2, … , r30, r31
- **complex RE approach**
  - r0 | r00 | r1 | r01 | … | r10 | r11 | r12 | … | r30 | r31
  - 사람이 이해하기는 쉽지만, 구현은 복잡하다…





- **simple FA + extra check approach**
  - RE : $r[0..9]^+$
- 어느 쪽이든, 수행 시간은 거의 비슷함

Check Points

- **Regular expressions** can be used to specify
  the tokens recognized by a lexical analyzer

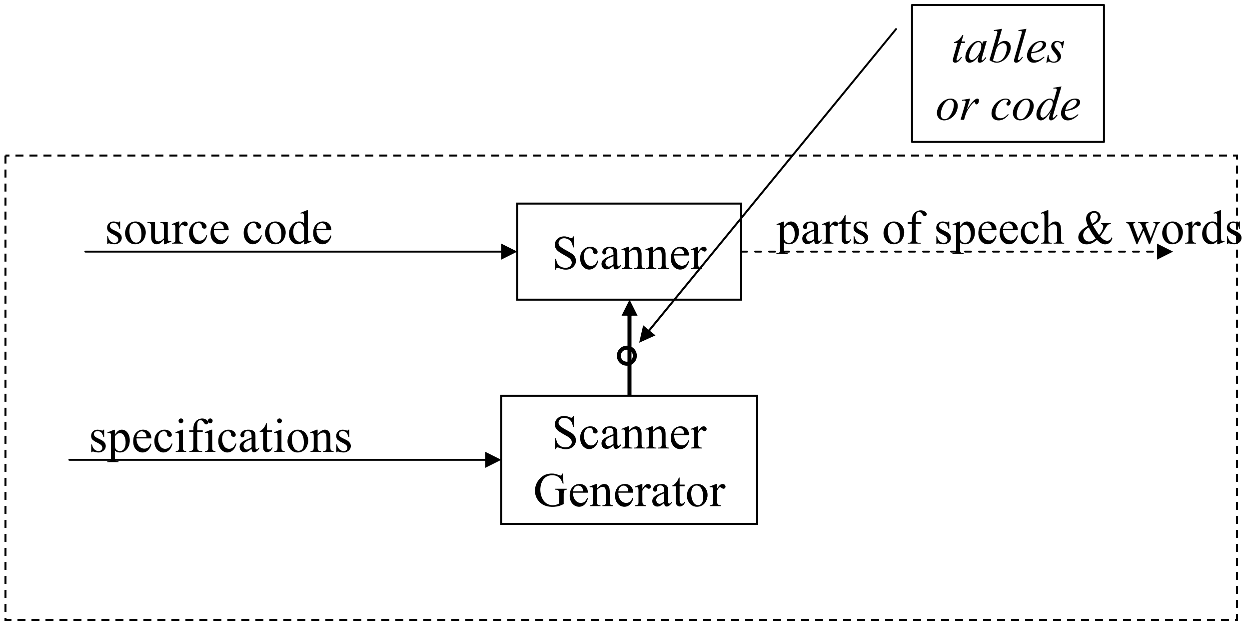
- Using results from **automata theory**
  and **theory of algorithms**,
  we can automatically build
  recognizers from regular expressions

- We study REs and associated theory
  to automate scanner construction !

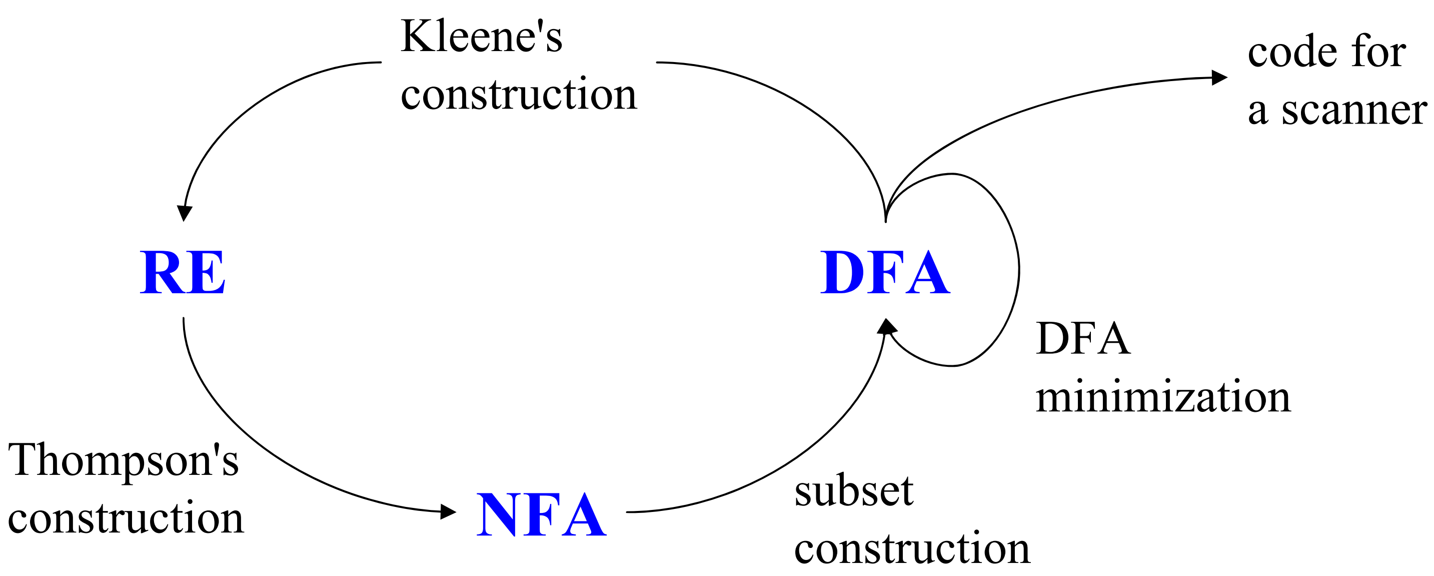
2.4 From RE to Scanner and BACK

Our Goal

- We will show
  how to construct **a finite state automaton**
  to recognize any RE



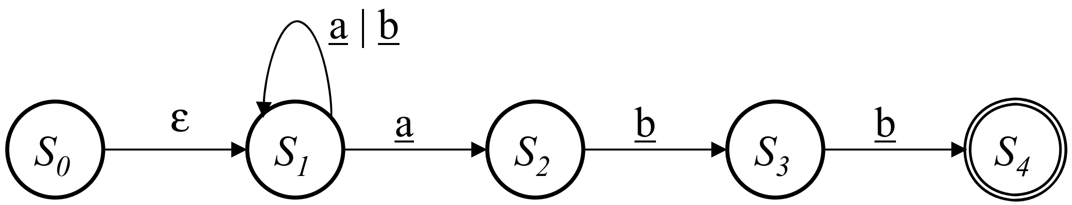
Global View



- NFA : non-deterministic finite automata
- DFA : deterministic finite automata
- RE를 공부한 이유 : FA를 자동으로 만들기 위해 !

NFA

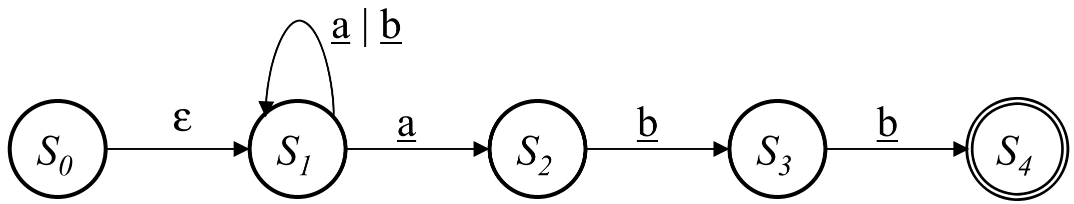
- Each **RE** corresponds
  to a deterministic finite automaton (**DFA**)
  - May be hard to directly construct the right DFA
- What about an RE such as (a | b)*abb ?



- This is a little different
  - s_0 has **a transition on ε**
  - s_1 has **two transitions on a**
- This is a non-deterministic finite automaton (NFA)

NFA

- ε 의 존재
  - hand-written FA 에서는 사실상 불필요
  - automated FA 생성에서는 필수
    - RE 끼리의 결합에 사용
- **multiple transition의 존재**
  - An NFA accepts a string x iff ∃ a path though the transition graph from s_0 to a final state such that the edge labels spell x
  - x를 accept 하는 path만 존재하면, accept 판정



DFA and NFA

- Why study NFAs?
  - They are the key to automating the RE $\rightarrow$ DFA construction
  - We can paste together NFAs with ε-transitions
- **DFA is a special case of an NFA**
  - DFA has **no ε-transitions**
  - DFA's transition function is **single-valued**
  - Same rules will work
- DFA can be simulated with an NFA
  - obvious !
- NFA can be simulated with a DFA
  - We will show it !

Automating Scanner Construction

To convert a specification into code:

1 Write down the **RE** for the input language

2 Build **a big NFA**

3 Build the **DFA** that simulates the NFA

4 Systematically **shrink the DFA**

5 Turn it into **code**

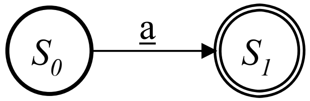
Scanner generators

- **Lex / Flex** work along these lines
- Algorithms are well-known and well-understood
- Key issue is interface to parser
- You could build one in a weekend!

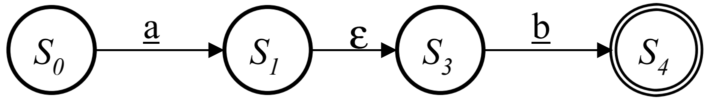
Thompson's Construction (RE → NFA)

Key idea

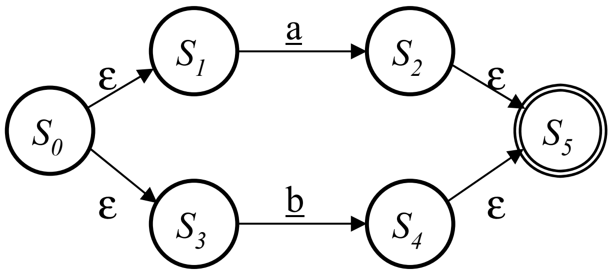
- NFA pattern for each symbol & each operator
- Join them with ε moves in precedence order



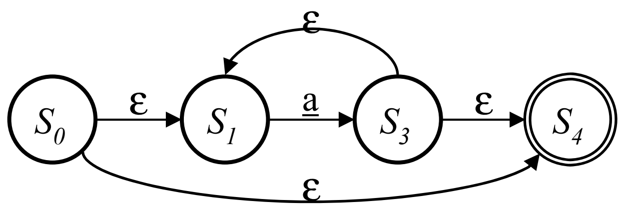
NFA for a



NFA for ab



NFA for a | b



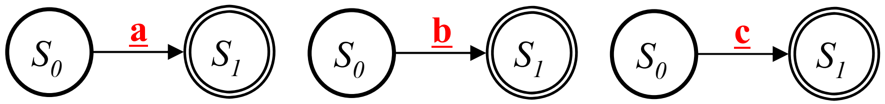
NFA for $\underline{a}^*$

Ken Thompson, CACM, 1968

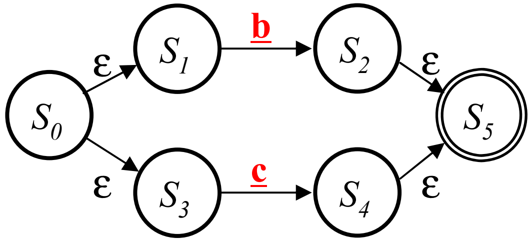
Example of Thompson's Construction

Let's try <u>a</u> (<u>b</u> | <u>c</u>)*

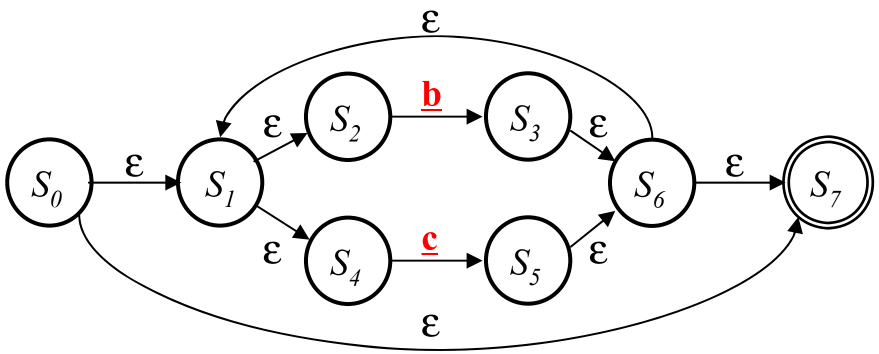
1. <u>a</u>, <u>b</u>, & <u>c</u>



2. <u>b</u> | <u>c</u>

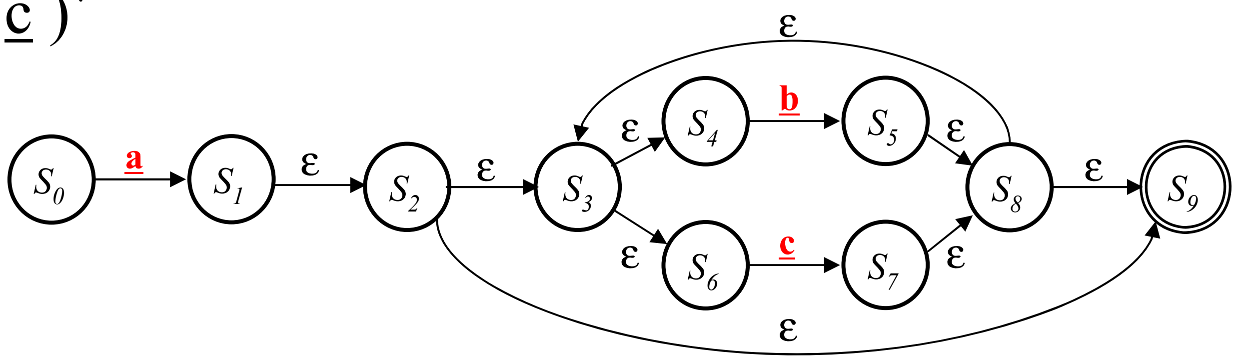


3. (<u>b</u> | <u>c</u>)*

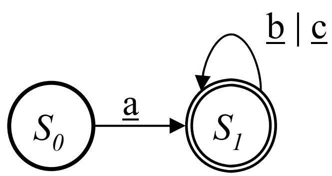


Example of Thompson's Construction

4. a (b | c)*



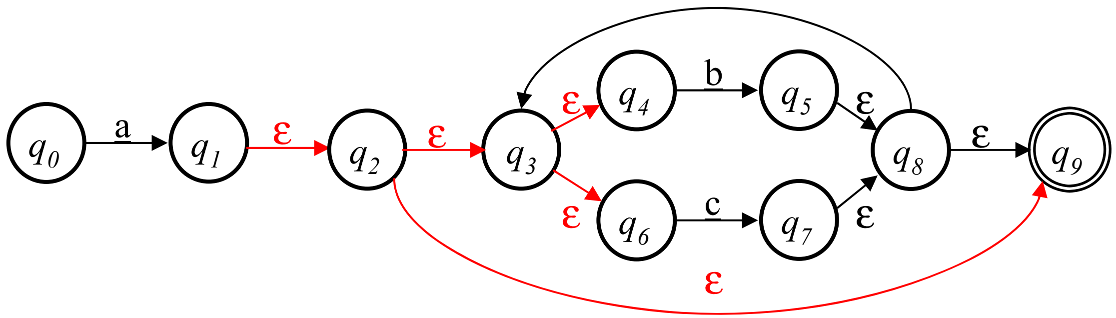
Of course, a human would design something simpler ...

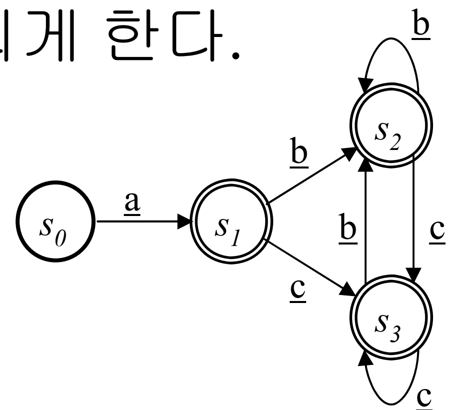


But, we can automate production of the more complex one ...

Subset Construction (NFA → DFA)

- subset construction : **NFA → DFA** algorithm
  - NFA : $(N, \Sigma, \delta_n, n_0, N_F)$
  - DFA : $(D, \Sigma, \delta_d, d_0, D_F)$
  - 핵심은 ***N* → *D*** 로의 계산
- 기본 아이디어
  - NFA에서 주어진 input으로 갈 수 있는 모든 state를 다 따라간다.
  - 그 모든 state를 **DFA의 하나의 state**가 되게 한다.



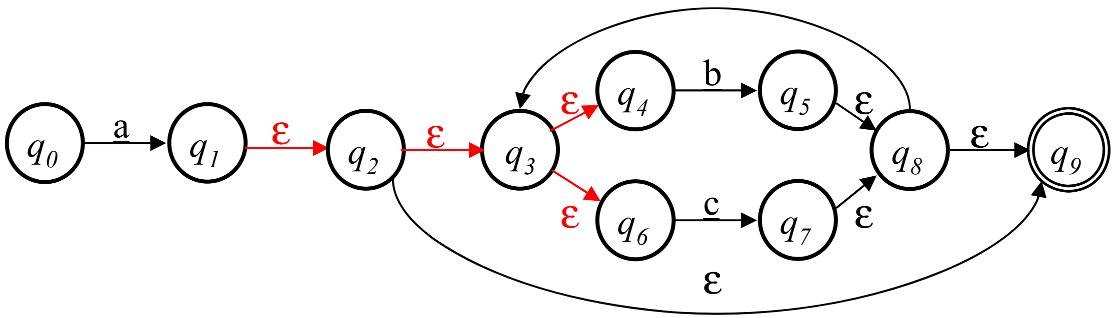


Subset Construction (NFA → DFA)

Need to build a simulation of the NFA

Two key functions

- *ε-closure*(s_i) is set of states reachable from s_i by ε
- *Delta*(s_i, a) is set of states reachable from s_i by a
  - $\varepsilon\text{-}closure(q_0) = \{ q_0 \} \rightarrow d_0$
  - $Delta(d_0, \underline{a}) = \{ q_1 \}$
  - $\varepsilon\text{-}closure(q_1) = \{ q_1, q_2, q_3, q_4, q_6, q_9 \} \rightarrow d_1$
  - $Delta(d_1, \underline{b}) = \{ q_5 \}$



Subset Construction (NFA $\rightarrow$ DFA)

The algorithm:

- Start state derived from s_0 of the NFA
- Take its ε-closure $\boldsymbol{S_0 = \varepsilon\text{-}closure(s_0)}$
- Take the image of S_0, $\boldsymbol{Delta(S_0, \alpha)}$ for each $\alpha \in \Sigma$, and take its ε-closure
- **Iterate** until no more states are added

Sounds more complex than it is…

Subset Construction (NFA $\rightarrow$ DFA)

- How many states in DFA ?
  - NFA has $N = \{ q_0, q_1, \ldots, q_k \}$
  - DFA has $D = \{ d_0, d_1, \ldots, d_m \} \subseteq 2^N$

- 2^N : power set of N = all possible subsets of N
  - example : $A = \{ 1, 2 \}$
  - $2^A = \{ \emptyset, \{1\}, \{2\}, \{1,2\} \}$

- $|N| = k \rightarrow |2^N| = 2^k$: finite !

Subset Construction (NFA $\rightarrow$ DFA)

The algorithm:

$q_0 \leftarrow \varepsilon\text{-}closure(q_0)$

$S \leftarrow \{ q_0 \}$

while (S is still changing)

  for each $s_i \in S$

    for each $\alpha \in \Sigma$

      $t \leftarrow \varepsilon\text{-}closure(Delta(s_i, \alpha))$

      if ($t \notin S$) **then**

        add t to S as s_j

      $T[s_i, \alpha] \leftarrow s_j$

- a fixed point iteration !

The algorithm halts:

- S contains no duplicates
- 2^N is finite : power set
- while loop adds to S, but does not remove from S *(monotone)*

→ the loop halts !

S contains all the reachable NFA states:

- It tries each character in each s_i.
- every possible NFA configuration 을 시도

→ *S and T form the DFA*

Subset Construction (NFA → DFA)

Example of a ***fixed-point* computation**

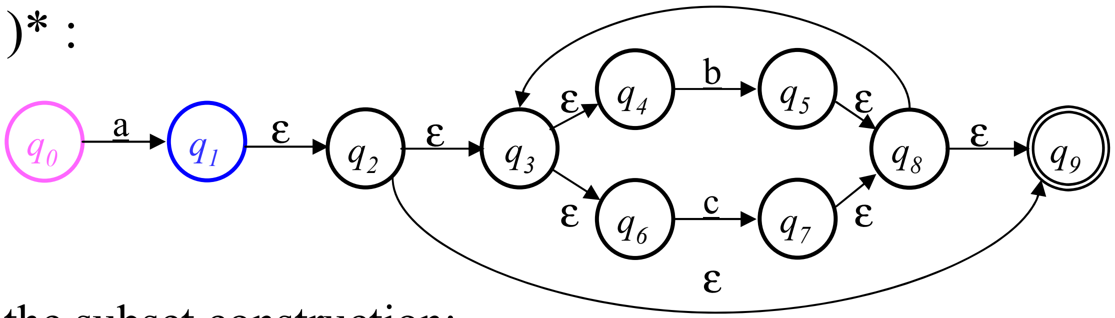
- Monotone construction of some finite set
- **Halts when it stops adding to the set**
- Proofs of halting & correctness are similar
- These computations arise in many contexts

Other fixed-point computations

- Canonical construction of sets of LR(1) items
  - Quite similar to the subset construction
- Many numerical analysis algorithms
  - example: Newton method

An Example

a (b | c)* :

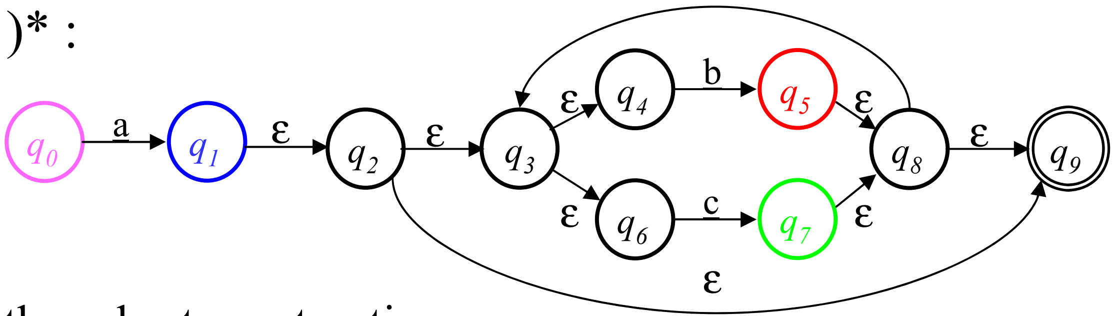


Applying the subset construction:

		ε-closure(Delta(s,*))		
	NFA states	a	b	c
s_0	q_0	$q_1, q_2, q_3, q_4, q_6, q_9$	*none*	*none*

An Example

a (b | c)* :

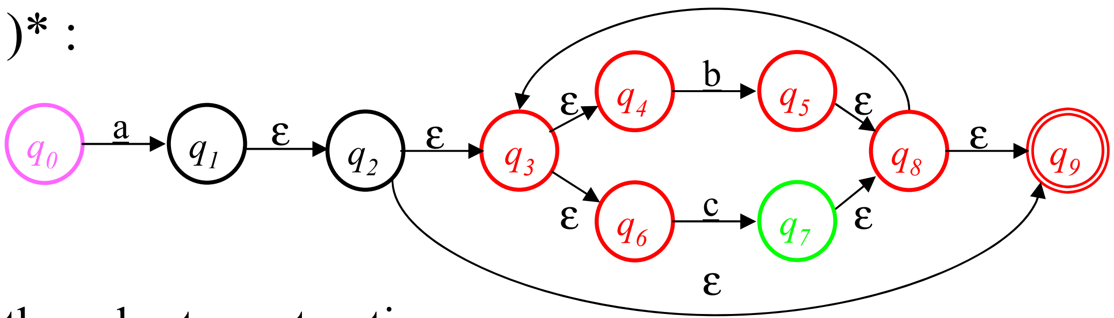


Applying the subset construction:

		ε-closure(Delta(s,*))		
	NFA states	a	b	c
s_0	q_0	$q_1, q_2, q_3, q_4, q_6, q_9$	*none*	*none*
s_1	$q_1, q_2, q_3, q_4, q_6, q_9$	*none*	$q_5, q_8, q_9, q_3, q_4, q_6$	$q_7, q_8, q_9, q_3, q_4, q_6$

An Example

a (b | c)* :

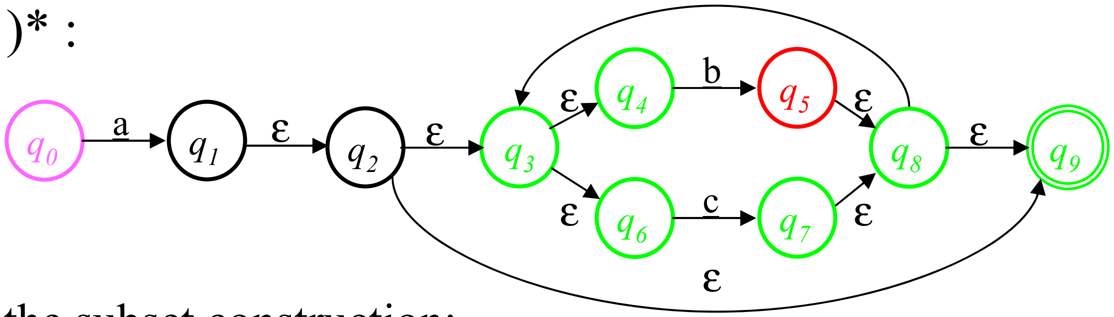


Applying the subset construction:

		ε-closure(Delta(s,*))		
	NFA states	a	b	c
s_0	q_0	$q_1, q_2, q_3, q_4, q_6, q_9$	*none*	*none*
s_1	$q_1, q_2, q_3, q_4, q_6, q_9$	*none*	$q_5, q_8, q_9, q_3, q_4, q_6$	$q_7, q_8, q_9, q_3, q_4, q_6$
s_2	$q_5, q_8, q_9, q_3, q_4, q_6$	*none*	$q_5, q_8, q_9, q_3, q_4, q_6$	$q_7, q_8, q_9, q_3, q_4, q_6$

An Example

a (b | c)* :

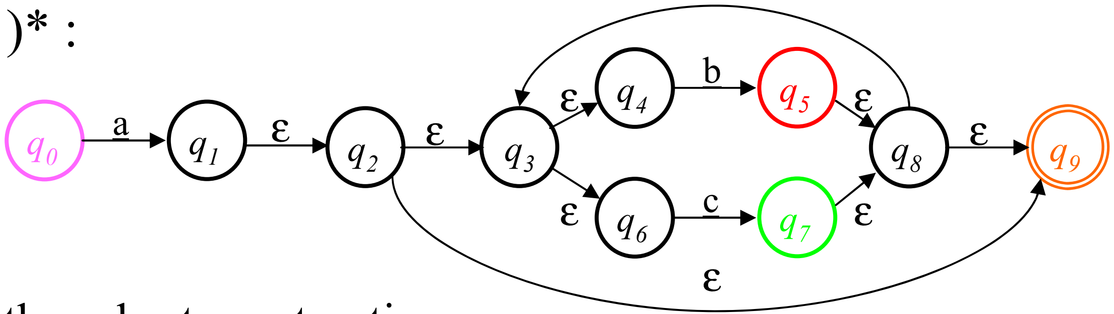


Applying the subset construction:

		ε-closure(Delta(s,*))		
	NFA states	a	b	c
s_0	q_0	$q_1, q_2, q_3, q_4, q_6, q_9$	*none*	*none*
s_1	$q_1, q_2, q_3, q_4, q_6, q_9$	*none*	$q_5, q_8, q_9, q_3, q_4, q_6$	$q_7, q_8, q_9, q_3, q_4, q_6$
s_2	$q_5, q_8, q_9, q_3, q_4, q_6$	*none*	$q_5, q_8, q_9, q_3, q_4, q_6$	$q_7, q_8, q_9, q_3, q_4, q_6$
s_3	$q_7, q_8, q_9, q_3, q_4, q_6$	*none*	$q_5, q_8, q_9, q_3, q_4, q_6$	$q_7, q_8, q_9, q_3, q_4, q_6$

An Example

a (b | c)* :

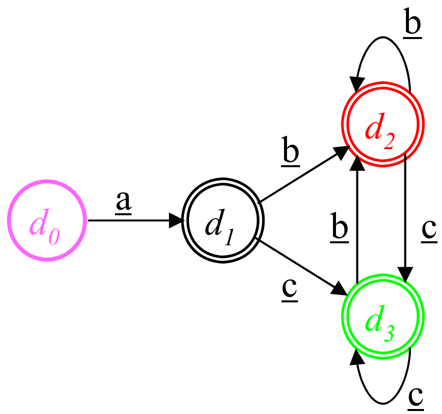


Applying the subset construction:

		ε-closure(Delta(s,*))		
	NFA states	a	b	c
s_0	q_0	s_1	*none*	*none*
s_1	$q_1, q_2, q_3, q_4, q_6, q_9$	*none*	s_2	s_3
s_2	$q_5, q_8, q_9, q_3, q_4, q_6$	*none*	s_2	s_3
s_3	$q_7, q_8, q_9, q_3, q_4, q_6$	*none*	s_2	s_3

An Example

The DFA for a (b | c)*

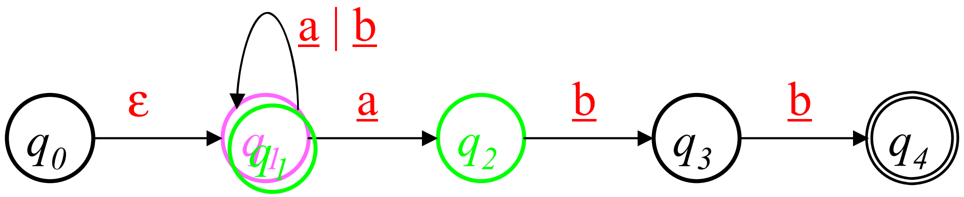


	a	b	c
d_0	d_1	–	–
d_1	–	d_2	d_3
d_2	–	d_2	d_3
d_3	–	d_2	d_3

- Ends up smaller than the NFA
- All transitions are deterministic
- Use same code skeleton as before

Another Example

- Remember (a | b)* abb ? – hand-driven NFA !

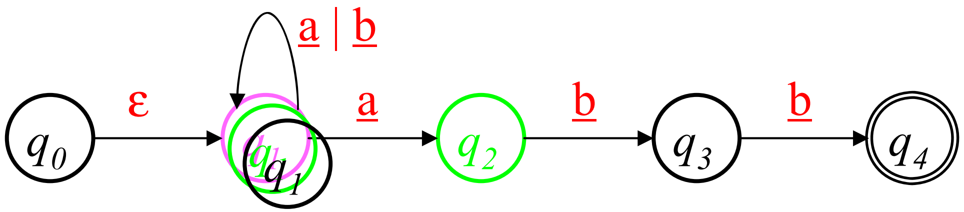


- subset construction

Iter.	State	Contains	ε-closure(Delta(s_i,a))	ε-closure(Delta(s_i,b))
0	s_0	q_0, q_1	q_1, q_2	q_1

Another Example

- Remember (a | b)* abb ?

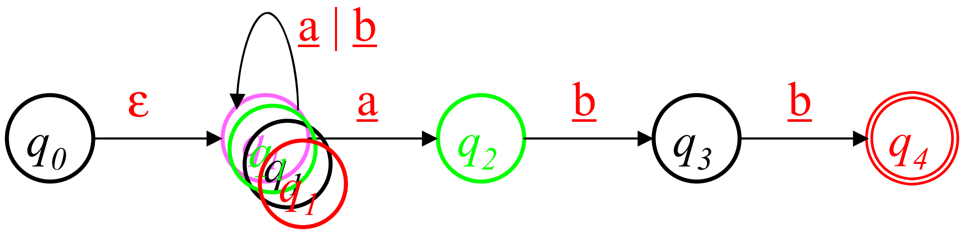


- subset construction

Iter.	State	Contains	ε-closure(Delta(s_i,a))	ε-closure(Delta(s_i,b))
0	s_0	q_0, q_1	q_1, q_2	q_1
1	s_1	q_1, q_2	q_1, q_2	q_1, q_3
	s_2	q_1	q_1, q_2	q_1

Another Example

- Remember (a | b)* abb ?

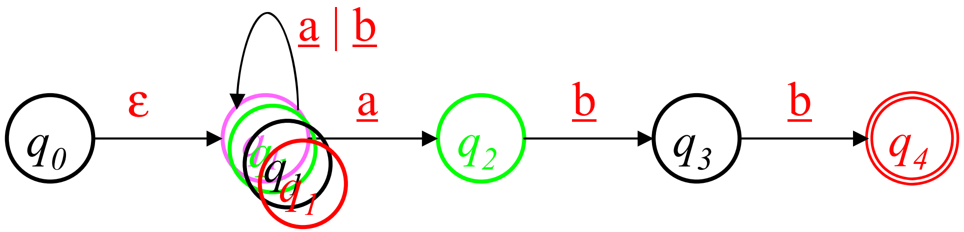


- subset construction

Iter.	State	Contains	ε-closure(Delta(s_i,a))	ε-closure(Delta(s_i,b))
0	s_0	q_0, q_1	q_1, q_2	q_1
1	s_1	q_1, q_2	q_1, q_2	q_1, q_3
	s_2	q_1	q_1, q_2	q_1
2	s_3	q_1, q_3	q_1, q_2	q_1, q_4

Another Example

- Remember (a | b)* abb ?

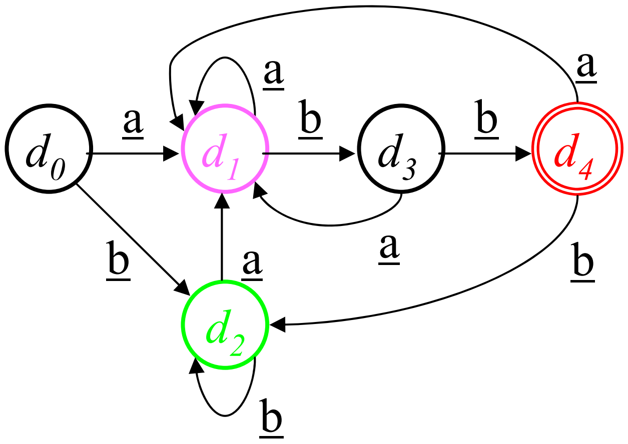


- subset construction

Iter.	State	Contains	ε-closure(Delta(s_i,a))	ε-closure(Delta(s_i,b))
0	s_0	q_0, q_1	q_1, q_2	q_1
1	s_1	q_1, q_2	q_1, q_2	q_1, q_3
	s_2	q_1	q_1, q_2	q_1
2	s_3	q_1, q_3	q_1, q_2	q_1, q_4
3	s_4	q_1, q_4	q_1, q_2	q_1

Another Example

- DFA for (a | b)* abb



	a	b
d_0	d_1	d_2
d_1	d_1	d_3
d_2	d_1	d_2
d_3	d_1	d_4
d_4	d_1	d_2

- Ends up smaller than the NFA
- All transitions are deterministic
- Use same code skeleton as before

DFA Minimization

- key idea
- Two states are **equivalent** if and only if:
  - The set of **paths leading to them are equivalent**
  - $\forall \alpha \in \Sigma$, **transitions on α lead to equivalent states**
  - α-transitions to distinct sets

    → states must be in distinct sets
- 실제 알고리즘 : 반대로 구현
  - 모든 **set**을 **equivalent** 하다고 가정하고,
  - equivalence가 깨어질 때마다, 새로운 **state** 추가

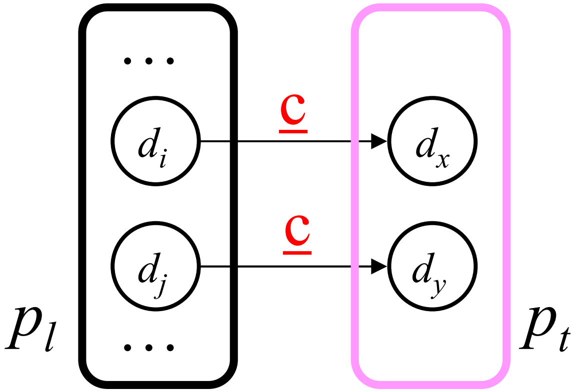
DFA Minimization

- input : a DFA with states $D = \{ d_0, d_1, \ldots, d_n \}$
- output : another DFA with states $P = \{ p_0, p_1, \ldots, p_m \}$
  - p_l contains a set of one or more DFA states d_i's
    - $p_l = \{ d_i \}$ or $p_l = \{ \ldots, d_i, d_j, \ldots \}$
  - P covers D : $\bigcup_{l=1}^{m} p_l = D$

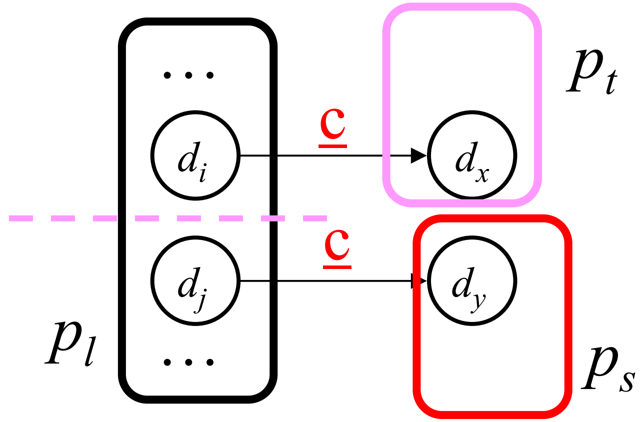
- key idea
  - Discover sets of equivalent states
  - Represent each such set with just one state

DFA Minimization

- equivalence test
  - let $d_i \in p_l$ and $d_j \in p_l$
  - for all $c \in \Sigma$, we know that: $d_i \xrightarrow{c} d_x, d_j \xrightarrow{c} d_y$
  - if $d_x \in p_t$ and $d_y \in p_t$: equivalent
  - otherwise, split p_l into two states



equivalent 만족



split 필요 !

DFA Minimization

Details of the algorithm

- Group states into maximal size sets, *optimistically*
- Iteratively subdivide those sets, as needed
- States that remain grouped together are equivalent

→ **fixed point iteration !**

Initially, two states:

- $p_0 = D_F$: **final states** in original DFA
- $p_1 = D - D_F$: **non-final states**

DFA Minimization

The algorithm:

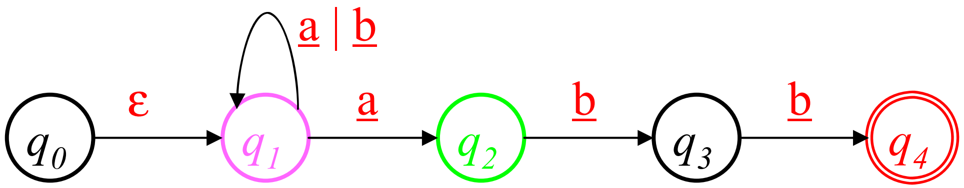
$P \leftarrow \{ D_F, D - D_F \}$
while (P is still changing)
  $T \leftarrow \emptyset$
  for each set $p \in P$
    for each $\alpha \in \Sigma$
      if split needed,
        split p into p_1 and p_2
        $T \leftarrow T \cup p_1 \cup p_2$
      else
        $T \leftarrow T \cup p$
  if $T \neq P$ **then**
    $P \leftarrow T$

Why does this work?

- Partition $P \in 2^D$
- Start off with 2 subsets:
  D_F and $D - D_F$
- *While* loop takes $P_i \rightarrow P_{i+1}$ by splitting 1 or more sets
- P_{i+1} is at least one step closer to the partition with $|D|$ sets
- Maximum of $|D|$ splits

An Example

- Remember (a | b)* abb ?

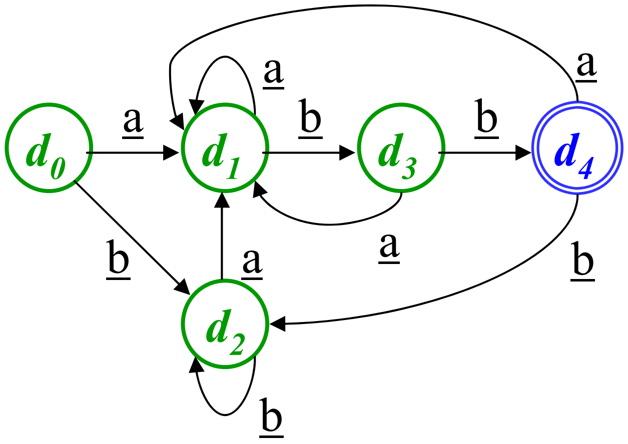


- subset construction

Iter.	State	Contains	ε-closure(Delta(s_i, a))	ε-closure(Delta(s_i, b))
0	s_0	q_0, q_1	q_1, q_2	q_1
1	s_1	q_1, q_2	q_1, q_2	q_1, q_3
	s_2	q_1	q_1, q_2	q_1
2	s_3	q_1, q_3	q_1, q_2	q_1, q_4
3	s_4	q_1, q_4	q_1, q_2	q_1

An Example

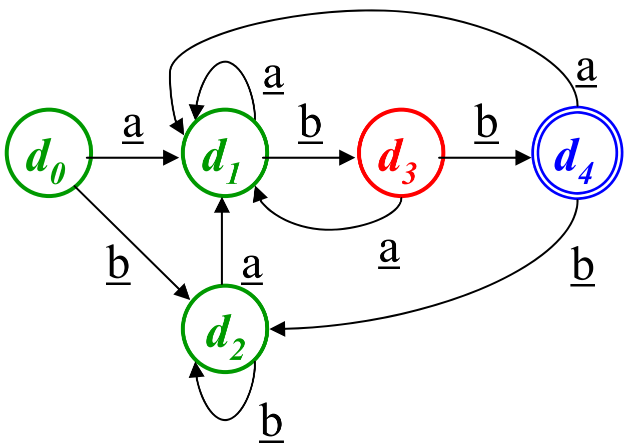
- DFA for (a | b)* abb



	Current Partition	*target*	*Split on* a	*Split on* b
P_0	$\{d_4\}$ $\{d_0,d_1,d_2,d_3\}$	$\{d_0,d_1,d_2,d_3\}$	none	$\{d_0, d_1, d_2\}$ $\{d_3\}$

An Example

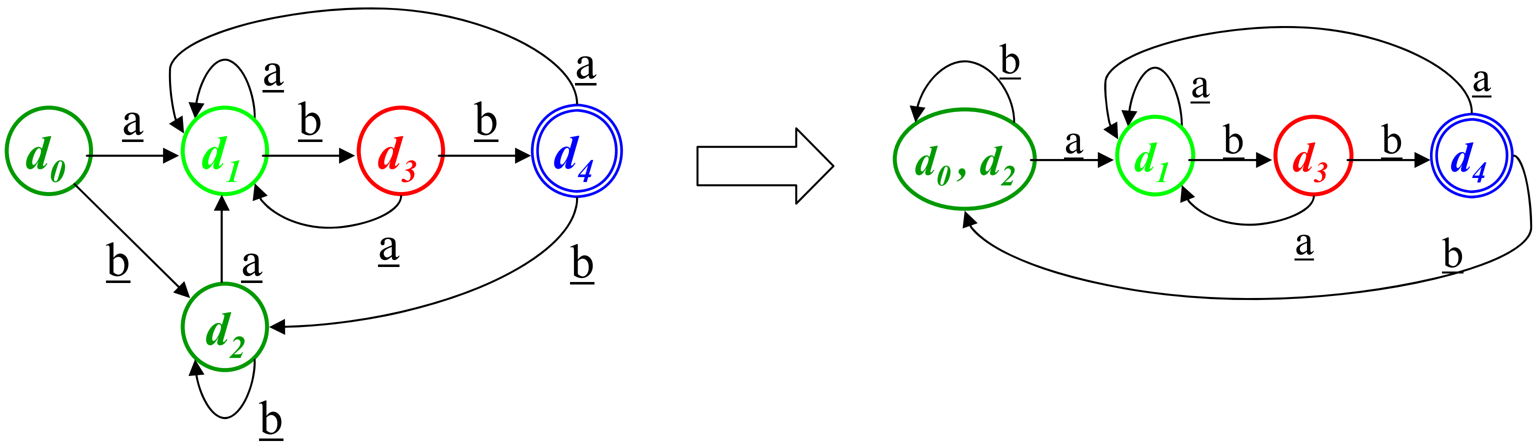
- DFA for (a | b)* abb



	Current Partition	*target*	*Split on* a	*Split on* b
P_0	$\{d_4\}$ $\{d_0,d_1,d_2,d_3\}$	$\{d_0,d_1,d_2,d_3\}$	none	$\{d_0, d_1, d_2\}$ $\{d_3\}$
P_1	$\{d_4\}$ $\{d_3\}$ $\{d_0,d_1,d_2\}$	$\{d_0,d_1,d_2\}$	none	$\{d_0, d_2\}$ $\{d_1\}$

An Example

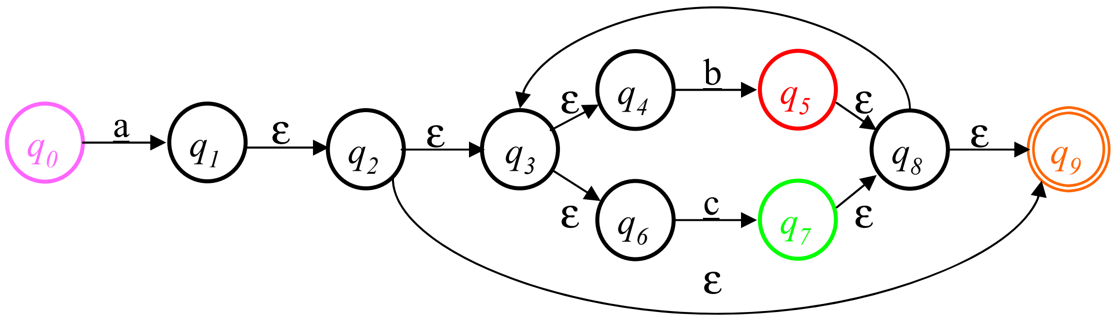
- DFA for (a | b)* abb



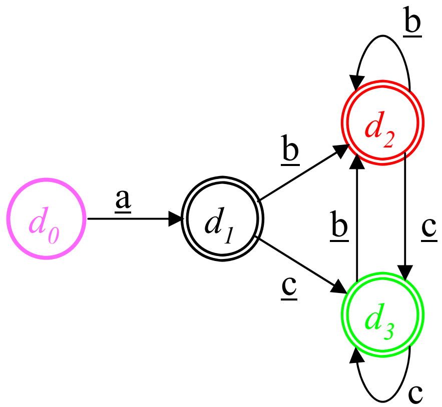
	Current Partition	*target*	*Split on* a	*Split on* b
P_0	$\{d_4\}$ $\{d_0,d_1,d_2,d_3\}$	$\{d_0,d_1,d_2,d_3\}$	none	$\{d_0, d_1, d_2\}$ $\{d_3\}$
P_1	$\{d_4\}$ $\{d_3\}$ $\{d_0,d_1,d_2\}$	$\{d_0,d_1,d_2\}$	none	$\{d_0, d_2\}$ $\{d_1\}$
P_2	$\{d_4\}$ $\{d_3\}$ $\{d_1\}$ $\{d_0,d_2\}$		none	none

Another Example

- NFA for a (b | c)* :

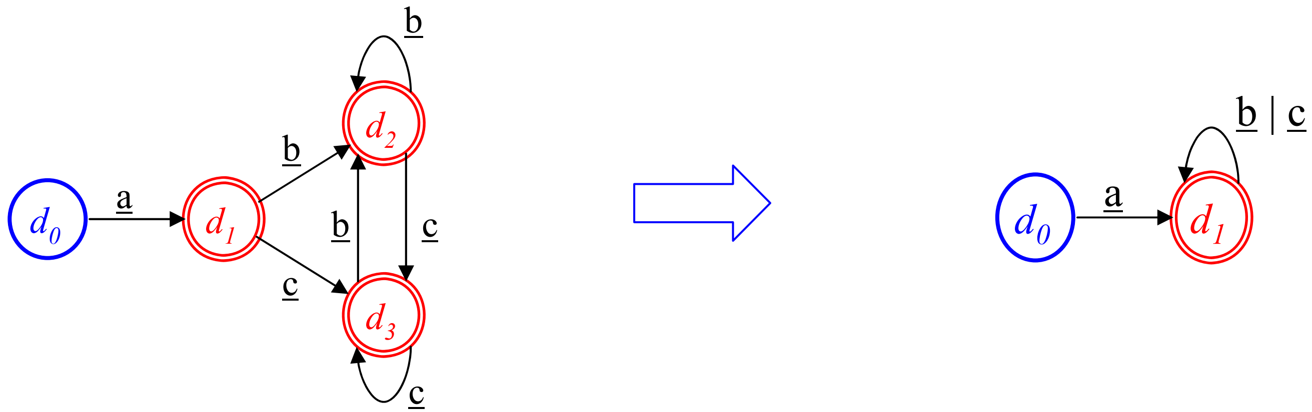


- DFA for a (b | c)* :



Another Example

- DFA for a (b | c)* :



	Current Partition	*Split on* a	b	c
P_0	{ d_1, d_2, d_3} {d_0}	*none*	*none*	*none*

DFA to RE

- DFA : a graph !
- use **dynamic programming technique**
  - R^k_{ij} : RE for all paths **from state *i* to state *j***
    using only states from 1 to *k*
  - iterate on k, i, j and finally get R^n_{1n}

- see Figure 2.10 for the algorithm
  - see Algorithm Text book for behind idea !

DFA as Scanner

- DFA를 scanner로 쓸 때의 문제점
  - DFA : for a single word
  - scanner : sequence of words
  - 어디서 끊어야 하는가?
    - 예: in C, '<' and '<=' are both valid
- 해결책
  - language-level : 모든 word는 delimiter 로 끝난다.
    - '< =' → '<' and '=', '<=' → '<='
  - recognizer level : scanner에서 **longest match**를 선택
    - error 발생 시, 최후의 valid final state로 backup
    - 문제점: final state를 계속 trace 해야 된다

DFA as Scanner

- 또 하나의 문제점
  - 2개 이상으로 인식될 때, 어느 쪽을 택하나?
    - 'if' : keyword and a valid identifier
  - 해결책 1: in most programming languages
    - 'if' is always keyword → **reserved word**
    - 이런 유형은 모두 priority를 부여해서 해결
  - 해결책 2: in ForTran
    - **context sensitive**
      - if i .eq. 3 goto 4 → if 문
      - if = 3 → identifier

2.5 Implementing Scanners

Table-Driven Scanners

- 모든 DFA table 과 동작하는 code 사용

ch ← NextChar( )

state ← s_0

while (char ≠ eof)

  state ← δ(state, ch)

  ch ← NextChar( )

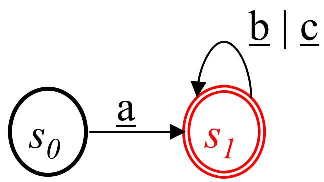
end while

if (state ∈ S_F)

  then report acceptance

  else report failure

- DFA for a (b | c)* :



- DFA table:

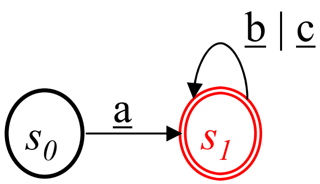
	a	b	c
s_0	s_1	s_e	s_e
s_1	s_e	s_1	s_1

- example input
  - "abbcc" + delimiter

Direct-Coded Scanners

- table 대신,
  code 형태로 직접 쓰는 방식
  - 장점 : faster
  - 단점 : longer and complicated
- 실제로는 이 방식을 선호
  - scanner generator가 사용
  - 편의상, goto를 많이 씀

- DFA for a (b | c)* :



- Example Code:

```
     goto s0
s0:  ch ← NextChar( )
     if (ch = 'a')
        then goto s1
        else goto error
s1:  ch ← NextChar( )
     if (ch = 'b' or ch = 'c')
        then goto s1
        else if (ch = eof)
           then report acceptance
           else goto error
error:
     report failure
```

Scanner Result

- 보통, **<type, word/value>** pair로 return

- operator :   <+, NULL>, <*, NULL>
- keyword :   <if, NULL>
- identifier :   <iden, "hello">
- number :   <int, 36>
- string :   <string, "world">

- 저장해 둘 필요 있음 → **symbol table**

Handling Keyword

- 대부분의 programming language 에서, keyword 와 identifier 정의는 겹침
  - **keywords ⊆ identifier**
  - 어떻게 효과적으로 처리할 것인가?
- 모든 keyword를 scanner가 인식한다
  - large scanner, but faster
- **symbol table lookup**
  - 모두 identifier로 인식
  - symbol table 에서, keyword 인지 search
  - speed-up : no linear or binary search, use hashing !

Handling Numbers

- scanner 처리가 끝나면,
  number 라는 사실보다, number value가 중요하다.
  - 어떻게 value를 계산할 것인가?

- final state 에서 계산
  - input 전체를 새로 읽어야 한다
  - buffer를 사용해도, 여전히 2번 읽어야 한다
- each state 마다 계산
  - complicated, but faster

- see Lex or Flex for more details

Handling Strings

- string 인식 ? "[^"]*"
  - 그 내용이 중요하다
  - each state에서 buffer를 update 해야 한다
- 문제점 ?
  - 대형 program 에서는 **duplicated string**이 많다
    → string 저장 공간 증가
  - 해결책: symbol table 에 저장

2.6 Advanced Topics

ForTran : a Nightmare

- no reserved word
  - if (x) = 1 : 배열 if의 x 번째 값을 1로
  - if (x) 120, 130 : x값이 negative → goto 120, else 130
- no delimiters
  - do 9 i = 1, 23 : i값이 1 ~ 23 반복해서 9번까지 수행
  - do9i=1,23 : same
  - do9i=1.23 : do9i 에 1.23 assign
  - do 9 i = 1.23 : same
- and more !
- 해결책 : **two-pass scanners**

Building Scanners

The point

- All this technology lets us **automate scanner construction**
- Implementer writes down the regular expressions
- Scanner generator builds **NFA**, **DFA**, **minimal DFA**, and then writes out the (table-driven or direct-coded) **code**
- This reliably produces fast, robust scanners

For most modern language features, this works

- You should think twice before introducing a new feature that defeats a DFA-based scanner
- 실패한 예들: insignificant blanks, non-reserved keywords