COMP3131/9102: Programming Languages and Compilers

Jingling Xue

School of Computer Science and Engineering
The University of New South Wales
Sydney, NSW 2052, Australia

`http://www.cse.unsw.edu.au/~cs3131`

`http://www.cse.unsw.edu.au/~cs9102`

Week 9 (Mon): DFAs and NFAs

Week 2:

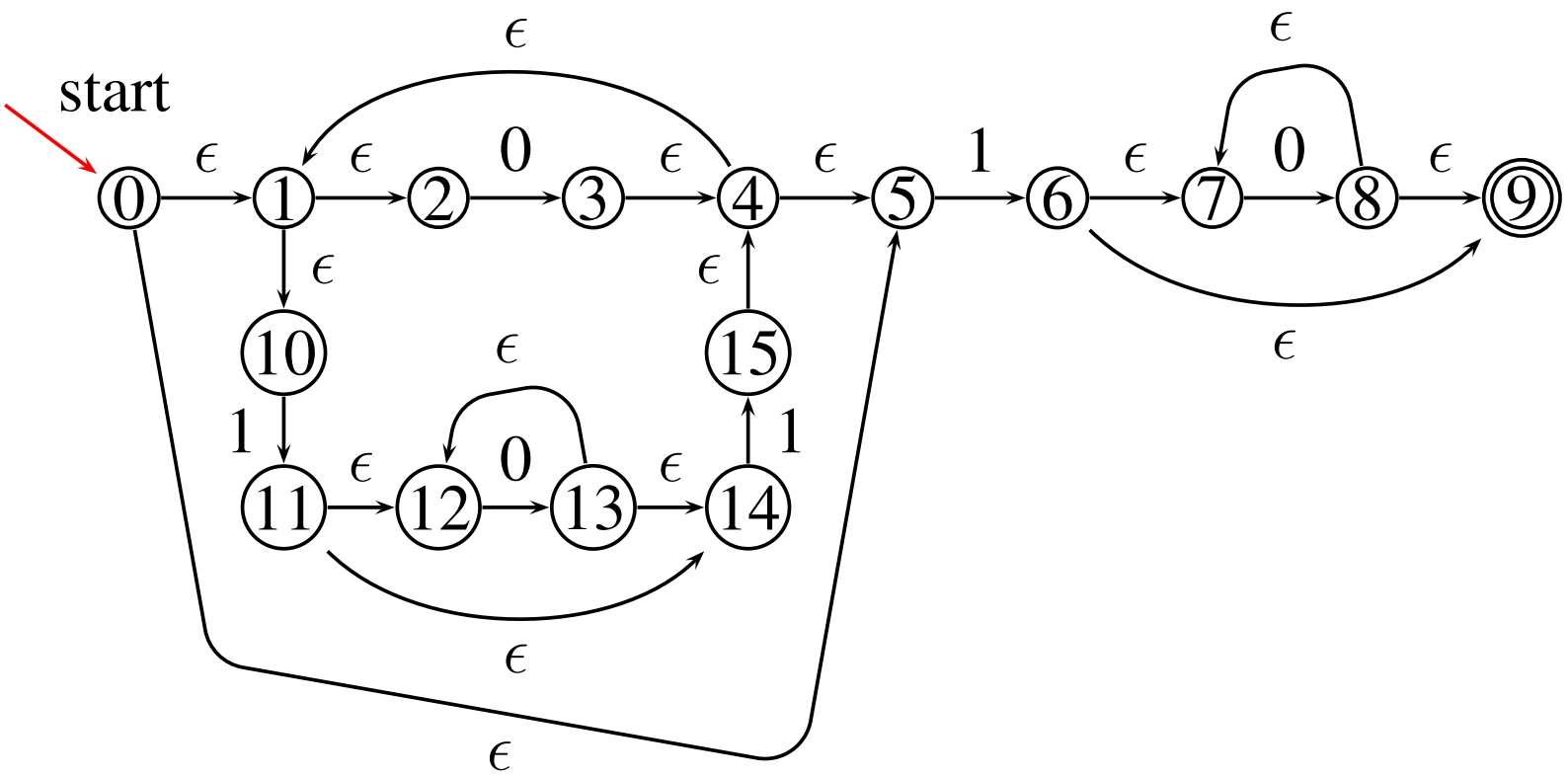
1. Definitions of REs, DFA and NFA
2. REs $\Longrightarrow$ NFA (*Thompson's construction, Algorithm 3.3, Red Dragon/Algorithm 3.23, Purple Dragon*)

Week 9:

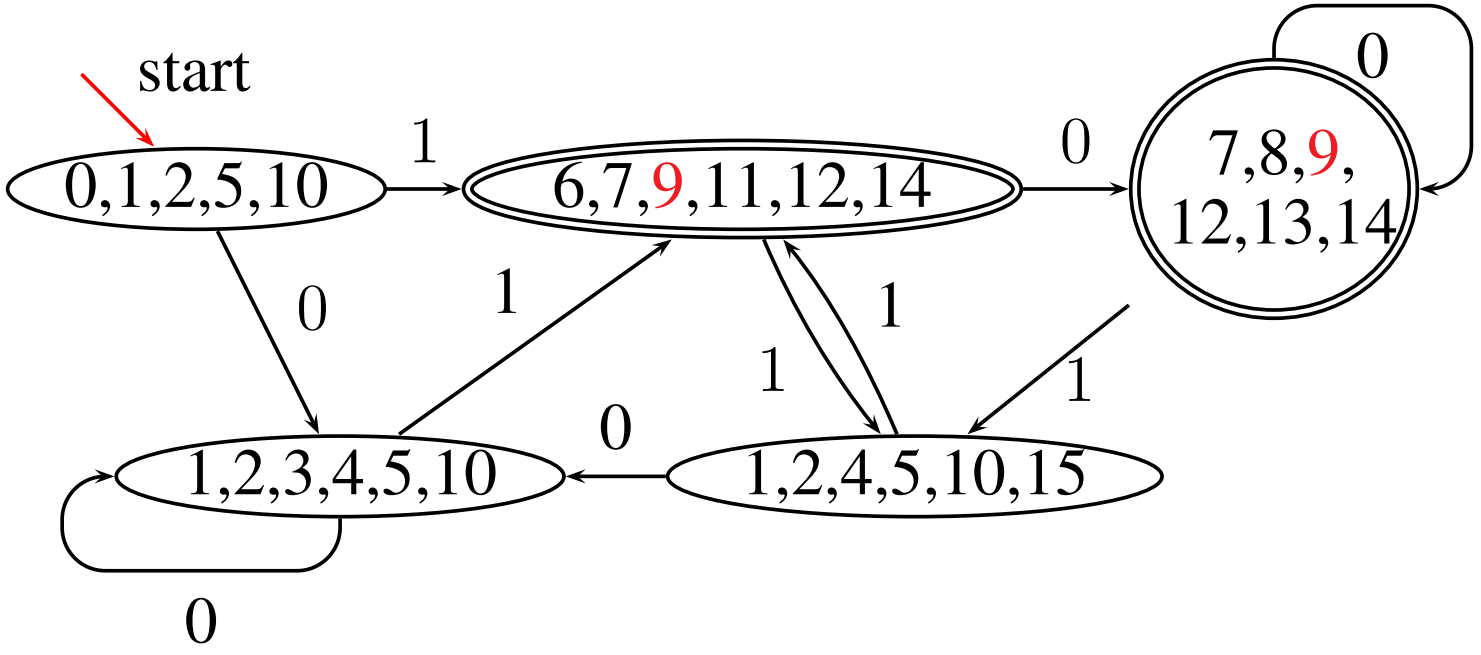
1. NFA $\Longrightarrow$ DFA (*subset construction, Algorithm 3.2, Red Dragon/Algorithm 3.20, Purple Dragon*)
2. DFA $\Longrightarrow$ minimal-state DFA (*state minimisation, Algorithm 3.6, Red Dragon/Algorithm 3.39, Purple Dragon*)
3. Scanner generators
   - How to use them (straightforward)
   - How to write them (the most techniques introduced today)

Example: RE $\Longrightarrow$ NFA

- Regular expression: $(0|10^*1)^*10^*$
- NFA:



Example: DFA (Cont'd)



- The algorithm used is known as the subset construction, because a DFA state corresponds to a subset of NFA states
- There are at most 2^n DFA states, where n is the total number of the NFA states

Subset Construction: The Operations Used

OPERATION	DESCRIPTION
ϵ-closure(s)	Set of NFA states readable from NFA state s on ϵ-transitions
ϵ-closure(T)	Set of NFA states readable from some state s in T on ϵ-transitions
move(T, a)	Set of NFA states to which there is a transition on input a from some state s in T

- s: a NFA state
- T: a set of NFA states

Subset Construction: The Algorithm

Let s_0 be the start state of the NFA;
DFAstates contains the only unmarked state ϵ-closure(s_0);
`while` there is an unmarked state T in DFAstates `do begin`
    `mark` T
    `for` each input symbol a `do begin`
        $U := \epsilon$-closure(move(T, a));
        `if` U is not in DFAstates `then`
            Add U as an unmarked state in DFAstates;
        DFATrans$[T, a] := U$;
    `end`;
`end`;

Subset Construction: The Definition of the DFA

Let (Σ, S, T, F, s_0) be the original NFA. The DFA is:

- The alphabet: Σ
- The states: all states in DFAstates
- The start state: ϵ-closure(s_0)
- The accepting states: all states in DFAstates containing at least one accepting state in F of the NFA
- The transitions: DFATrans

Weeks 2 + 9: Regular Expressions, DFA and NFA

1. Definitions of REs, DFA and NFA $\surd$
2. REs $\Longrightarrow$ NFA (*Thompson's construction, Algorithm 3.3, Red Dragon/Algorithm 3.23, Purple Dragon*) $\surd$
3. NFA $\Longrightarrow$ DFA (*subset construction, Algorithm 3.2, Red Dragon/Algorithm 3.20, Purple Dragon*) $\surd$
4. DFA $\Longrightarrow$ minimal-state DFA (*state minimisation, Algorithm 3.6, Red Dragon/Algorithm 3.39, Purple Dragon*)
5. Scanner generators
   - How to use them (straightforward)
   - How to write them (the most techniques introduced today)

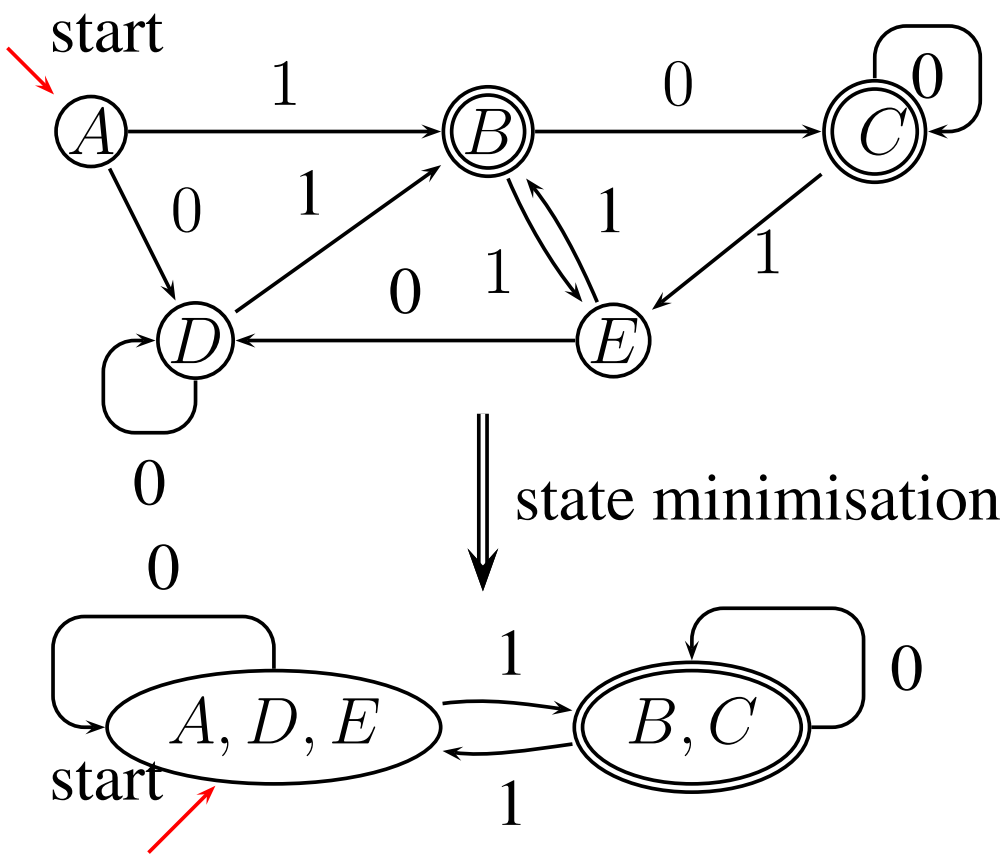
An Algorithm to Mimimise DFA Statements

Initially, let Π be the partition with the two groups:
(1) one is the set of all final states
(2) the other is the set of all non-final states

Let $\Pi_{new} = \Pi$
for (each group G in Π_{new}) {
partition G into subgroups such that two states s and t are in the same subgroup iff for all input symbols a, states s and t have transitions on a to states in the same group of Π_{new}
replace G in Π_{new} by the set of subgroups formed
}

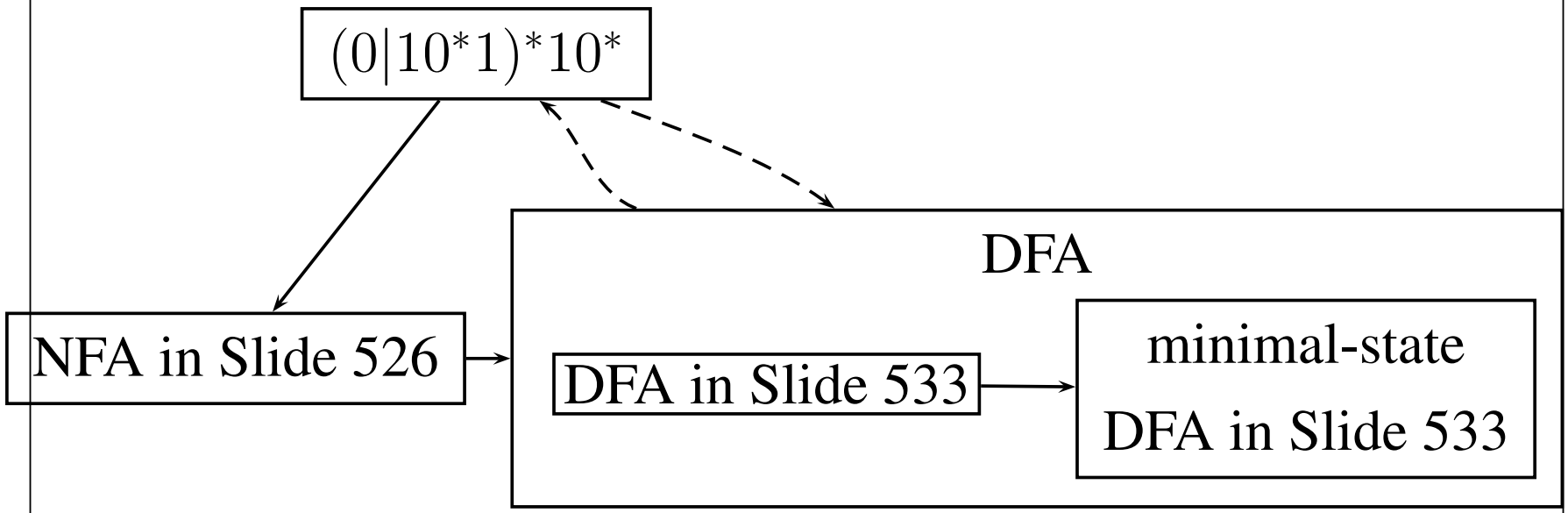
- Begins with the most optimistic assumption
- Also used in global value numbering (COMP4133)

Example (Cont'd): States Re-Labeled



Theoretical Result: every regular language can be recognised by a minimal-state DFA that is unique up to state names

Various Conversions for an Example



However, the conversions in dashed arrows are not covered.

Week 2: Regular Expressions, DFA and NFA

1. Definitions of REs, DFA and NFA $\surd$
2. REs $\Longrightarrow$ NFA (*Thompson's construction, Algorithm 3.3, Red Dragon/Algorithm 3.23, Purple Dragon*) $\surd$
3. NFA $\Longrightarrow$ DFA (*subset construction, Algorithm 3.2, Red Dragon/Algorithm 3.20, Purple Dragon*) $\surd$
4. DFA $\Longrightarrow$ minimal-state DFA (*state minimisation, Algorithm 3.6, Red Dragon/Algorithm 3.39, Purple Dragon*) $\surd$
5. Scanner generators
   - How to use them (straightforward)
   - How to write them (the most techniques introduced today)

Scanner Generators

- Scanners generated in C
  - lex (UNIX)
  - flex – GNU's fast lex (UNIX)
  - mks lex (MS-DOS and OS/2)
- Scanners generated in Java
  - Jflex
  - JavaCC (SUN Microsystems)

The Scanner Spec in Jflex

```
user code -- copied verbatim to the scanner file
%%
Jflex directives
%%
regular expression rules
```

How a Scanner Generator Works

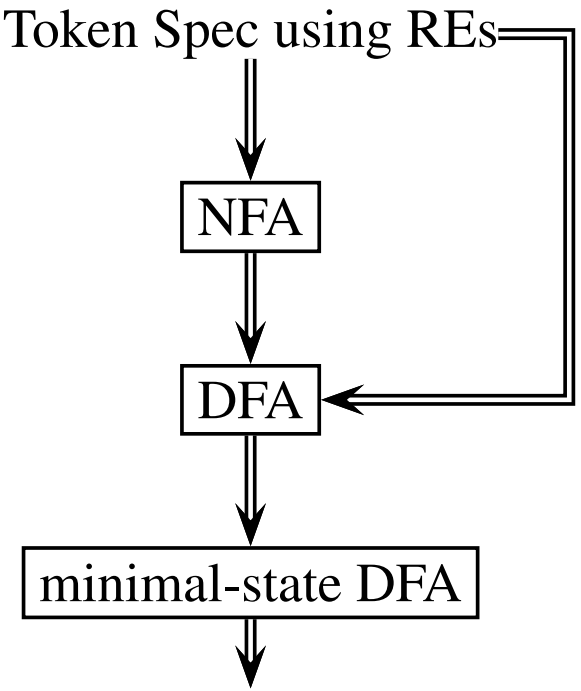


table-driven code (Jflex) – simulating a DFA on an input

or hard-wired code (Assignment 1 or §3.4 of either Dragon Book)

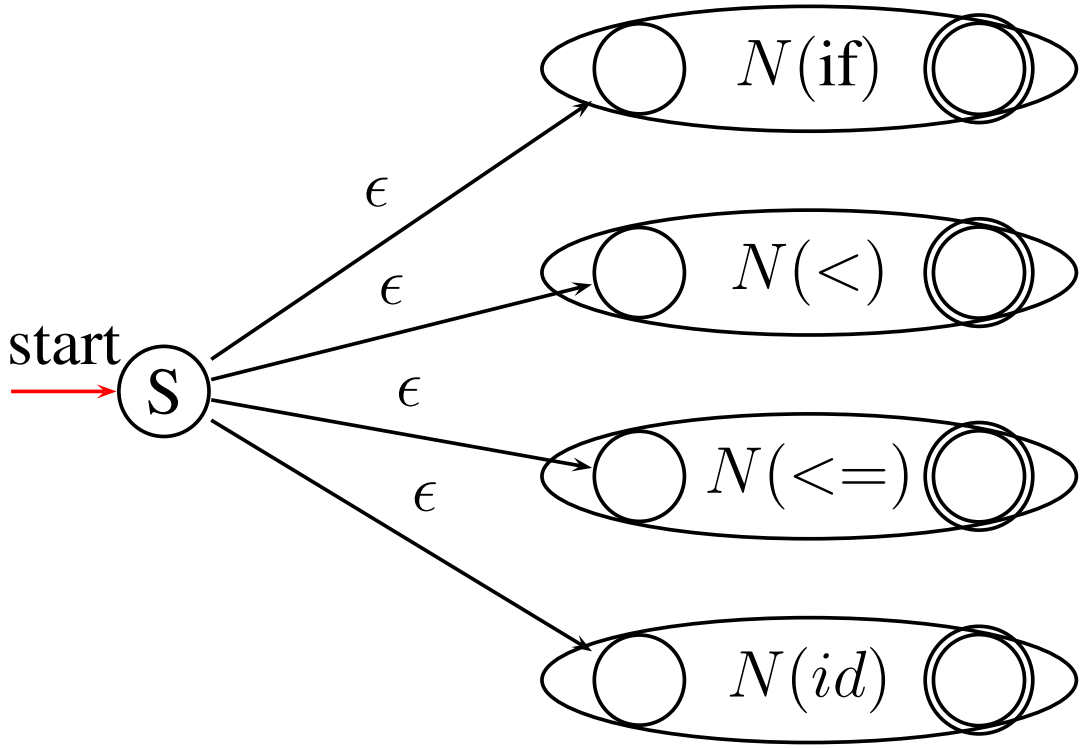
An Example: Spec

```
some user code
%%
LETTER=[A-Za-z_]
DIGIT=[0-9]
%%
"if"  { return new Token(Token.IF, "if", pos); }
"<"   { return new Token(Token.LT, "<", pos); }
"<="  { return new Token(Token.LE, "<=", pos); }
{LETTER}({LETTER}|{DIGIT})*
      { return new Token(Token.ID, "itsSpelling", pos); }
```

Two rules:

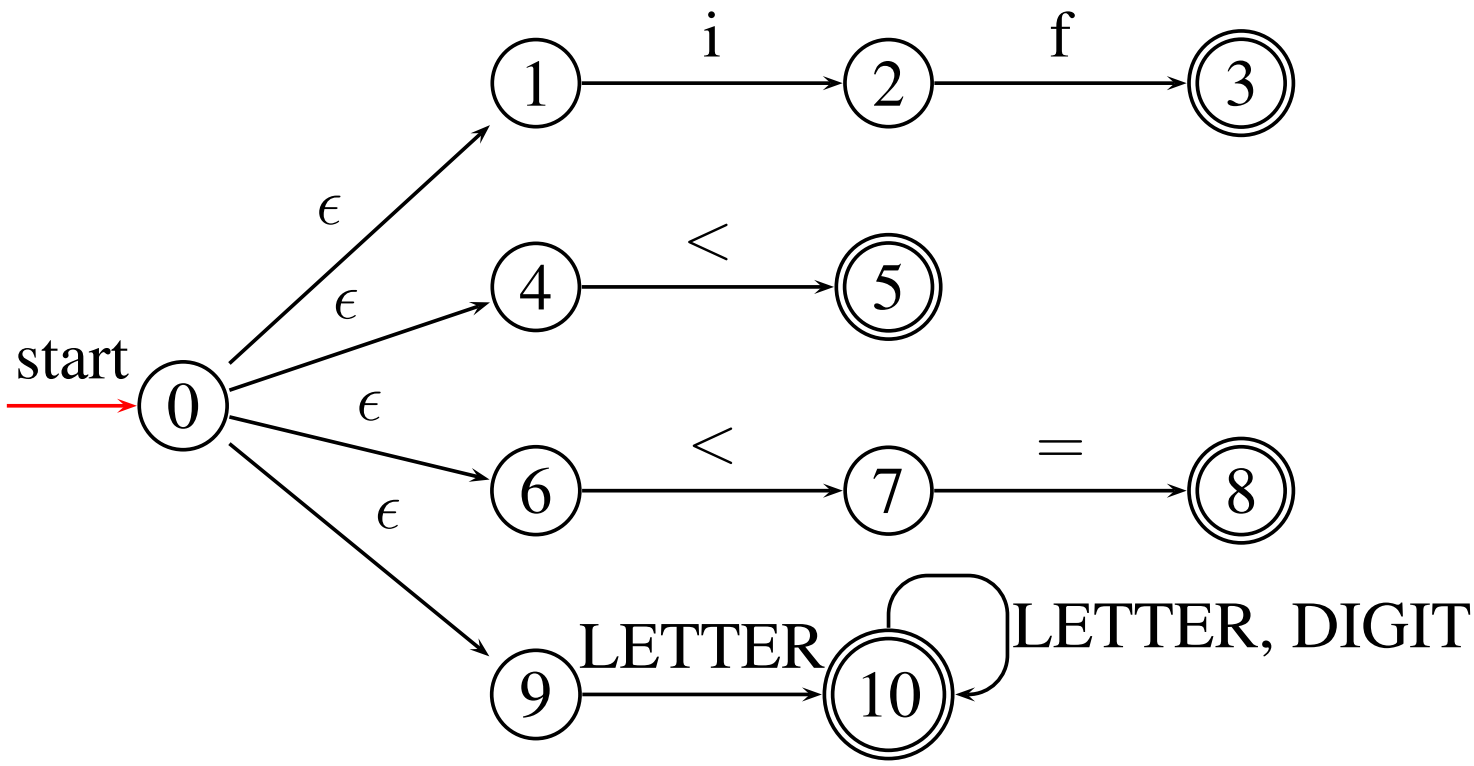
- The first pattern used when more than one are matched – “if” as a keyword not as an id
- The longest prefix of the input is always matched – ”$<=$ as one token

An Example: NFA

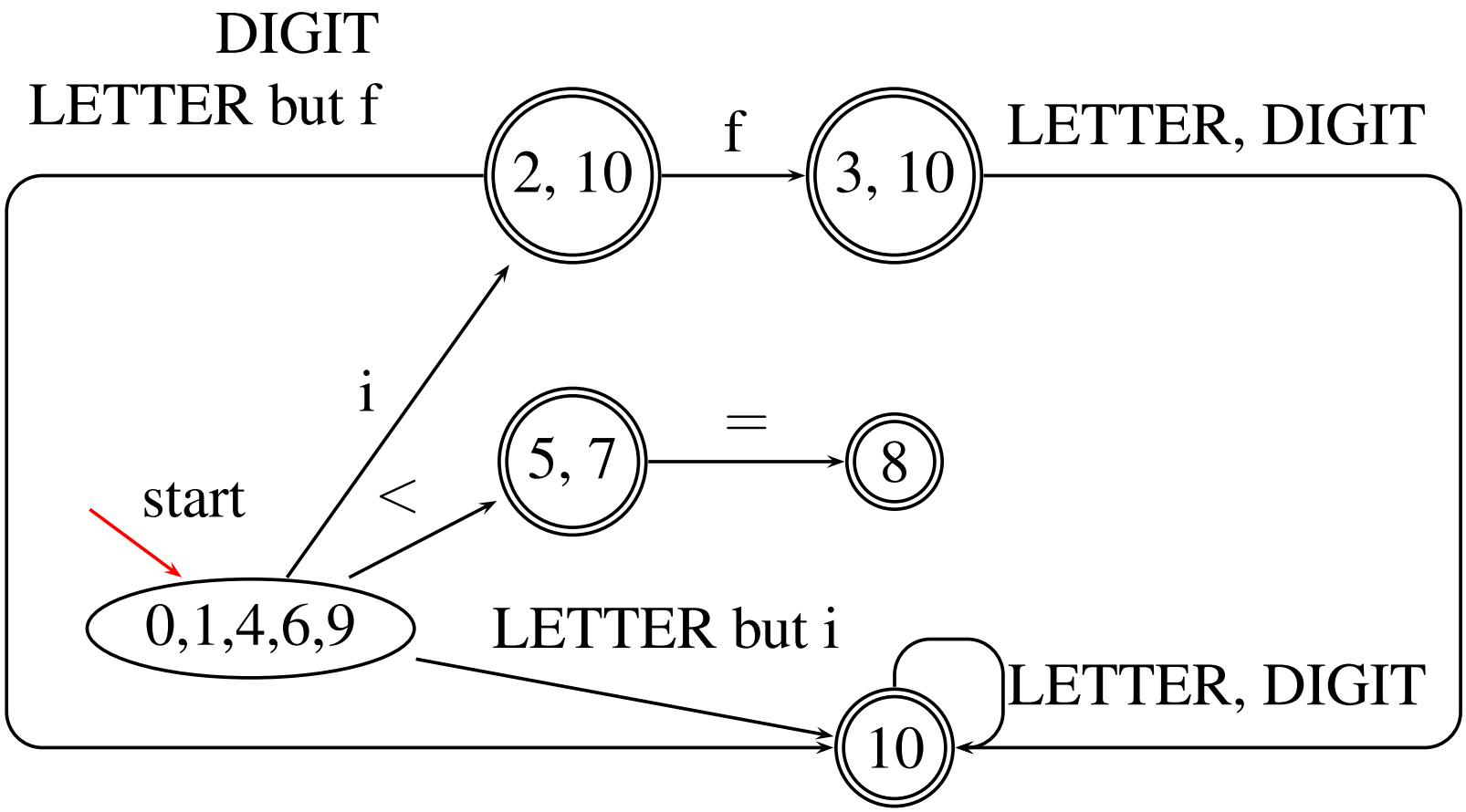


A DFA can also be used for each pattern.

An Example: NFA



An Example: DFA



Already a minimal-state DFA!

A DFA Represented as a Transition Table

State	Character						
	$<$	$=$	i	f	**LETTER** but i	**LETTER** but f	**DIGIT**
(0,1,4,6,9)	(5,7)		(2,10)		(10)		
(2,10)				(3,10)		(10)	(10)
(5,7)		(8)					
(8)							
(10)			(10)	(10)	(10)	(10)	(10)
(3,10)			(10)	(10)	(10)	(10)	(10)

- **Letter** = $\{i\} \cup$ "**letter** but i"
- Character classes reduce the table size
- The blank entries are errors
- The tables are usually sparse (pages 146 – 177 of text for compression techniques)

The Scanner Driver for Simulating a DFA

```
state = initial_state
while (TRUE) {
  next_state = T[state][current_char];
  if (next_state == ERROR)  // cannot move any further
    break;
  state = next_state;
  if (current_char == EOF)  // input exhausted
    break;
  current_char = getchar(); // fetch the next char
}
Backtrack to the most recent accepting state
if (such a state exists)
  /* return the corresponding token
     reset current_char to the first after the token
  */
else
  lexical_error(state);
```

- There should be a column in the transition table for EOF
- Need to backtrack

The Output of Running Jflex on a Sample Scanner Spec

- Scanner.l: the spec for the scanner generator Jflex

```
jflex Scanner.l

Constructing NFA : 267 states in NFA
Converting NFA to DFA :
139 states before minimization, 106 states in minimized DFA
Old file ‘‘Scanner.java’’ saved as ‘‘Scanner.java~’’
Writing code to ‘‘Scanner.java’’
```

- Scanner.l.java: the scanner generated

```
javac Scanner.l.java
```

- java Scanner test.vc

Limitations of Regular Expressions (or FAs)

- Cannot "count"
- Cannot recognise palindromes (e.g., racecar & rotator)
- The language of the balanced parentheses

$$\{(^n)^n \mid n \geqslant 1\}$$

  is not a regular language
  - cannot build a FA to recognise the language for any n (can trivially build a FA for n=3, for example)
  - but can be specified by a CFG (Week 3):

$$P \rightarrow (P) \mid ( )$$

Chomsky's Hierarchy

Depending on the form of production

$$\alpha \rightarrow \beta$$

four types of grammars (and accordingly, languages) are distinguished:

GRAMMAR	KNOWN AS	DEFINITION	LANGUAGE	MACHINE
Type 0	unrestricted grammar	$\alpha \neq \epsilon$	Type 0	Turing machine
Type 1	context-sensitive grammar CSGs	$\lvert\alpha\rvert \leq \lvert\beta\rvert$	Type 1	linear bounded automaton
Type 2	context-free grammar CFGs	$A \rightarrow \alpha$	Type 2	stack automaton
Type 3	Regular grammars	$A \rightarrow w \mid Bw$	Type 3	finite state automaton

Reading

- Sections 3.3 – 3.7 of either Dragon Book
- Week 10 tutorial questions (available on-line)

Week 9 (2nd Lecture): Table-Driven LL(1) Parsing