Tutorials

- Tutorials to start in Week 2 (i.e., next week)
- Tutorial questions are already available on-line

Assignment 1: Scanner

- +5 $\Longrightarrow$ two tokens: + and 5
  the scanner understands how tokens are formed but not anything else
- Maximal munch or longest match principle:

```
x = i---j;
```

COMP3131/9102: Programming Languages and Compilers

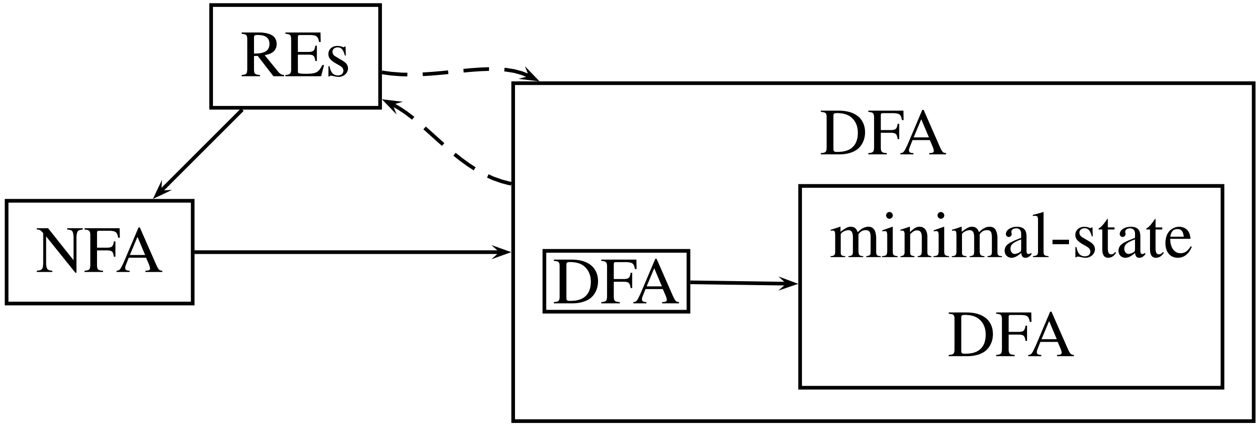
Jingling Xue

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`

The Big Picture



The two conversions in dashed arrows are not covered:

- REs $\rightarrow$ DFA (pages 135 – 141, Red Dragon/§3.7, Purple Dragon)
- DFA $\rightarrow$ REs: Chapter 3, J. Hopcroft, R. Motwani and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 2nd Edition, 2001. See `www-db.stanford.edu/~ullman/ullman-books.html`.
- DFA $\rightarrow$ minimal-state DFA (pages 141 – 144, Red Dragon/§3.9.6, Purple Dragon)
- Tools: `http://jflap.org/`

Week 1 (2nd): Regular Expressions, DFA and NFA

Today:

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)

Applications of Regular Expressions

- Anywhere when patterns of text need to be specified
  - Specifying restriction enzymes
  - Google analytics
- Unix system, database and networking administration:

  grep, fgrep, egrep, sed, awk
- HTML documents: Javascript and VBScript
- Perl:

  J. Friedl, Mastering Regular Expressions, O'reilly, 1997
- Token Specs for scanner generators (lex, Jflex, etc.)
- `http://www.zytrax.com/tech/web/regex.htm`

Applications of Finite Automata (i.e., Finite State Machines)

- Hardware design (minimising states $\Longrightarrow$ minimising cost)
- Language theory
- Computational complexity
- Scanner generators (lex and Jflex)
- Automata tools:
  `https://www.microsoft.com/en-us/research/project/automata/`

Alphabet, Strings and Languages

- Alphabet denoted Σ: any finite set of symbols
  - The binary alphabet $\{0,1\}$ (for machine languages)
  - The ASCII alphabet (for high-level languages)
- String: a finite sequence of symbols drawn from Σ:
  - Length $|s|$ of a string s: the number of symbols in s
  - ϵ: the empty string ($|\epsilon| = 0$)
- Language: any set of strings over Σ; its two special cases:
  - $\emptyset$: the empty set
  - $\{\epsilon\}$

Examples of Languages

- $\Sigma = \{0, 1\}$ – a string is an instruction
  - The set of M68K instructions
  - The set of Pentium instructions
  - The set of MIPS instructions
- $\Sigma =$ the ASCII set – a string is a program
  - the set of Haskell programs
  - the set of C programs
  - the set of VC programs

Terms for Parts of a String (Figure 3.7 of Text)

TERM	DEFINITION
prefix of s	a string obtained by removing 0 or more trailing symbols of s
suffix of s	a string obtained by removing 0 or more leading symbols of s
substring of s	a string obtained by deleting a prefix and a suffix from s
proper prefix suffix, substring of s	Any nonempty string x that is, respectively, a prefix, suffix or substring of s such that $s \neq x$

String Concatenation

- If x and y are strings, xy is the string formed by appending y to x
- Examples:

x	y	xy
key	word	keyword
java	script	javascript

- ϵ is the identity: $\epsilon x = x\epsilon = x$

Operations on Languages (Figure 3.8 of Text)

OPERATION	DEFINITION
union: $L \cup M$	$L \cup M = \{s \mid s \in L \text{ or } s \in M\}$
concatenation: LM	$LM = \{st \mid s \in L \text{ and } t \in M\}$
Kleene Closure: L^*	$L^* = \cup_{i=0}^{\infty} L^i = L^0 \cup L \cup LL \cup LLL \ldots$ where $L^0 = \{\epsilon\}$ (0 or more concatenations of L)
Positive Closure: L^+	$L^+ = \cup_{i=1}^{\infty} L^i = L \cup LL \cup LLL \ldots$ (1 or more concatenations of L)

Examples: Operations on Languages

- $L = \{a, \ldots, z, A, \ldots, Z, \_\}$
- $D = \{0, \ldots, 9\}$

EXAMPLE	LANGUAGE (THE SET OF)
$L \cup D$	
L^3	
LD	
L^*	
$L(L \cup D)^*$	
D^+	

Examples: Operations on Languages

- $L = \{a, \ldots, z, A, \ldots, Z, \_\}$
- $D = \{0, \ldots, 9\}$

EXAMPLE	LANGUAGE
$L \cup D$	letters and digits
L^3	all 3-letter strings
LD	strings consisting of a letter followed by a digit
L^*	all strings of letters, including the empty string ϵ
$L(L \cup D)^*$	all strings of letters and digits beginning with a letter
D^+	all strings of one or more digits

Regular Expressions (REs) Over Alphabet Σ

- Inductive Base:
  1. ϵ is a RE, denoting the RL $\{\epsilon\}$
  2. $a \in \Sigma$ is a RE, denoting the RL $\{a\}$

- Inductive Step: Suppose r and s are REs, denoting the RLs $L(r)$ and $L(s)$. Then (next slide):
  1. $(r)|(s)$ is a RE, denoting the RL $L(r) \cup L(s)$
  2. $(r)(s)$ is a RE, denoting the RL $L(r)L(s)$
  3. $(r)^*$ is a RE, denoting the RL $L(r)^*$
  4. (r) is a RE, denoting the RL $L(r)$

REs define regular languages (RL) or regular sets

Precedence and Associativity of “Regular” Operators

- Precedence:
  - “$*$” has the highest precedence
  - “Concatenation” has the second highest precedence
  - “$|$” has the lowest precedence
- Associativity: — all are left-associative
- Example:

$$(a)|((b)^*(c)) \equiv a|b^*c$$

Unnecessary parentheses can be avoided!

An Example (Following the Definition of REs)

- Alphabet: $\Sigma = \{0, 1\}$
- RE: $0(0|1)^*$
- Question: What is the language defined by the RE?
- Answer:

$$
\begin{array}{rcl}
L(0(0|1)^*) & = & L(0)L((0|1)^*) \\
 & = & \{0\}L(0|1)^* \\
 & = & \{0\}(L(0) \cup L(1))^* \\
 & = & \{0\}(\{0\} \cup \{1\})^* \\
 & = & \{0\}\{0, 1\}^* \\
 & = & \{0\}\{\epsilon, 0, 1, 00, 01, 10, 11, \ldots\} \\
 & = & \{0, 00, 01, 000, 001, 010, 011, \ldots\}
\end{array}
$$

The RE describes the set of strings of 0's and 1's beginning with a 0.

More Example Regular Expressions: $\Sigma = \{0, 1\}$

RE	LANGUAGE
1	$\{1\}$
$0\|1$	$\{0, 1\}$
1^*	$\{\epsilon, 1, 11, 111, \ldots\}$
1^*1	$\{1, 11, 111, \ldots\}$
$0\|0^*1$	the set containing 0 and all strings consisting of zero or more 0's followed by a 1.

Notational Shorthands

- One or more instances $^+$: $r^+ = rr^*$
  - denotes the language $(L(r))^+$
  - has the same precedence and associativity as *
- Zero or one instance $?$: $r? = r|\epsilon$
  - denotes the language $L(r) \cup \{\epsilon\}$
  - written as $(r)?$ to indicate grouping (e.g., (12)?)
- Character classes:

$$[A - Za - z\_][A - Za - z0 - 9\_]^*$$

Regular Expressions for VC (or C)

TOKEN	RE
Identifiers	$\textbf{letter}(\textbf{letter}\mid\textbf{digit})^*$
Integers	$\textbf{digit}^+$
Reals	A bit long but can be obtained from the following page by substitutions

- In the VC spec, **letter** includes "_"
- In Java, letters and digits may be drawn from the entire Unicode character set. Examples of identifiers are:

abc $\qquad \alpha\beta\gamma \qquad$ 中文

Regular Grammars for Integers and Reals in VC

- Integers:

```
      digit: 0|1|2|...|9
intLiteral: digit+
```

- Reals:

```
        digit: 0|1|2|...|9
     fraction: .digit+
     exponent: (E|e)(+|-)?digit+
 floatLiteral: digit* fraction exponent?
             | digit+.
             | digit+.?exponent
```

Regular grammars are a special case of CFGs (Week 2).

Finite Automata (or Finite State Machines)

A finite automaton consists of a 5-tuple:

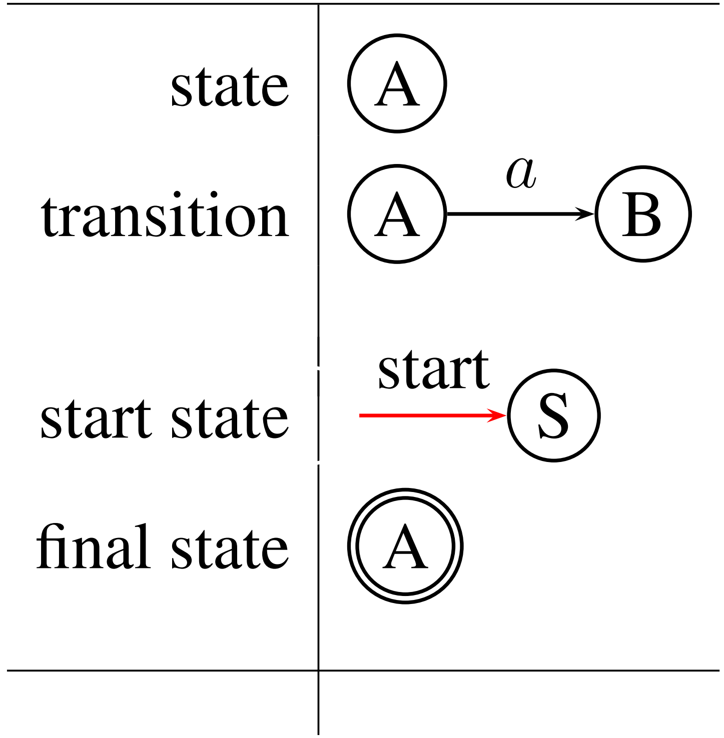
$$(\Sigma, S, T, F, I)$$

where

- Σ is an alphabet
- S is a finite set of states
- T is a state transition function: $T : S \times \Sigma \to S$
- F is a finite set of final or accepting states
- I is the start state: $I \in S$.

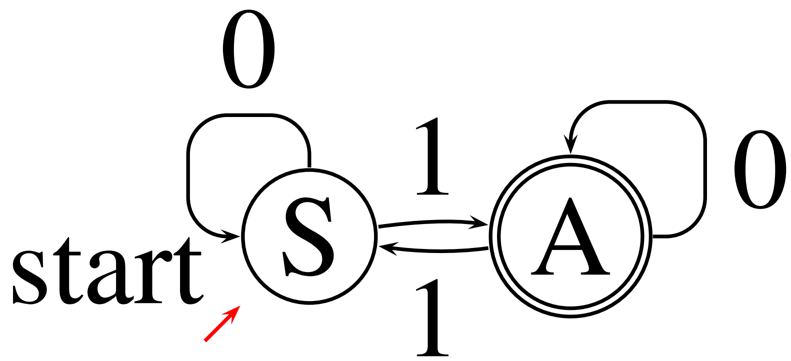
Representation and Acceptance

- Transition graph:



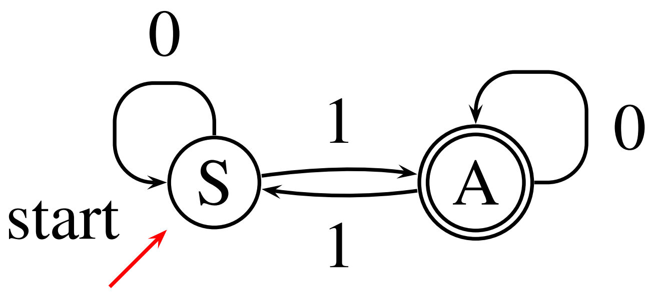
- Acceptance: A FA accepts an input string x iff there is some path in the transition graph from the start state to some accepting state such that the edge labels spell out x.

What Language does this FA accept?



Example 1

- The language: strings of 0 and 1 with an odd number of 1 (ϵ not included)



S: even number of 1's seen
A: odd number of 1's seen

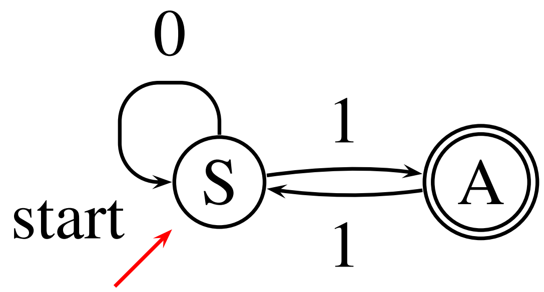
Σ	$\{0, 1\}$
S	$\{S, A\}$
T	$T(S, 0) = S, T(S, 1) = A, T(A, 0) = A, T(A, 1) = S$
F	$\{A\}$
I	S

- 01011 is recognised because

$$S \xrightarrow{0} S \xrightarrow{1} A \xrightarrow{0} A \xrightarrow{1} S \xrightarrow{1} A$$

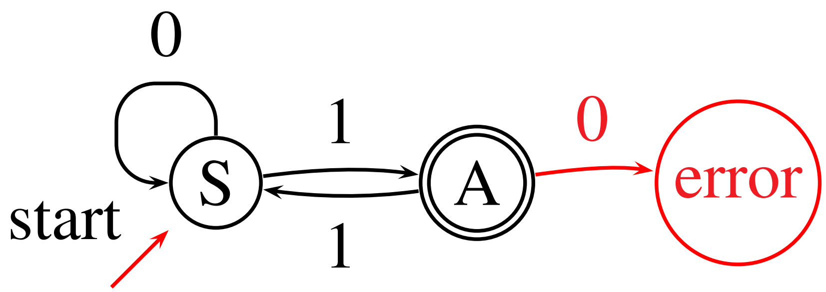
Implicit Error State

- By definition, T is a function from $S \times \Sigma$ to S, but ...



- If $T(s, a)$ is undefined at the state s on input a, then

$$T(s, a) = \text{error}$$



- The error state and transitions to it aren't drawn (by convention)

Deterministic FA (DFA) and Nondeterministic FA (NFA)

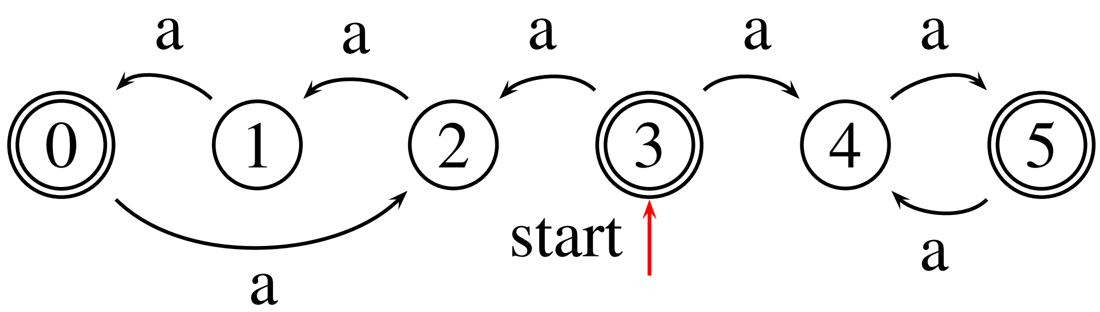
A FA is a DFA if

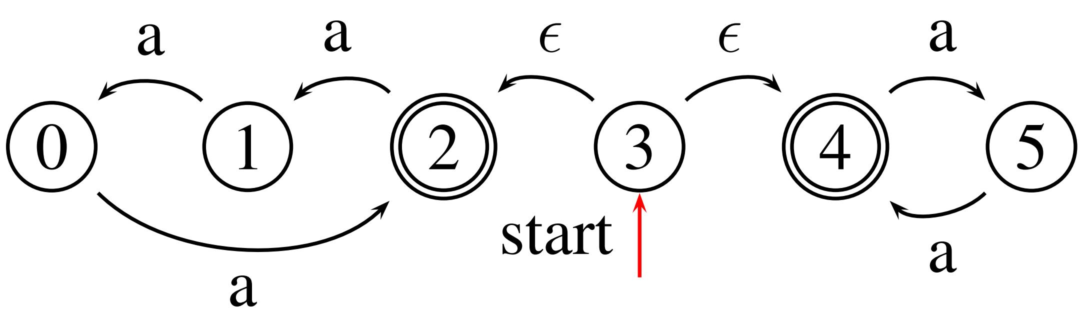
- no state has an ϵ-transition, i.e., an transition on input ϵ, and
- for each state s and input symbol a, there is at most one edge labeled a leaving s

A FA is an NFA if it is not a DFA:

- Nondeterministic: can make several parallel transitions on a given input
- Acceptance: the existence of some path as per Slide 84

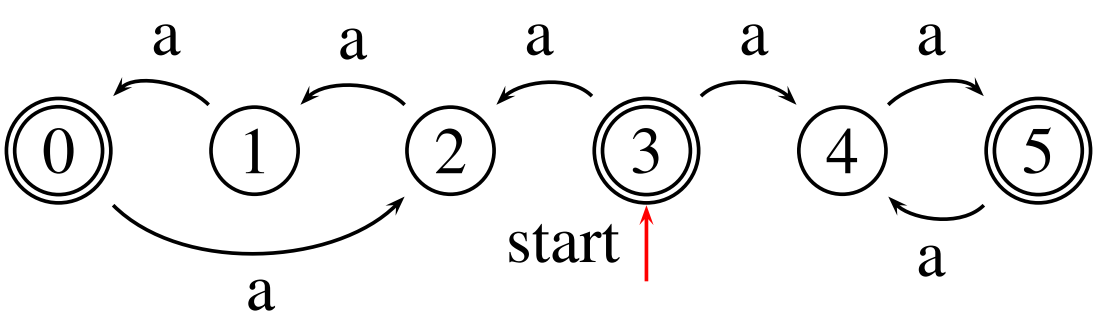
DFA or NFA? What are the Languages Recognised?



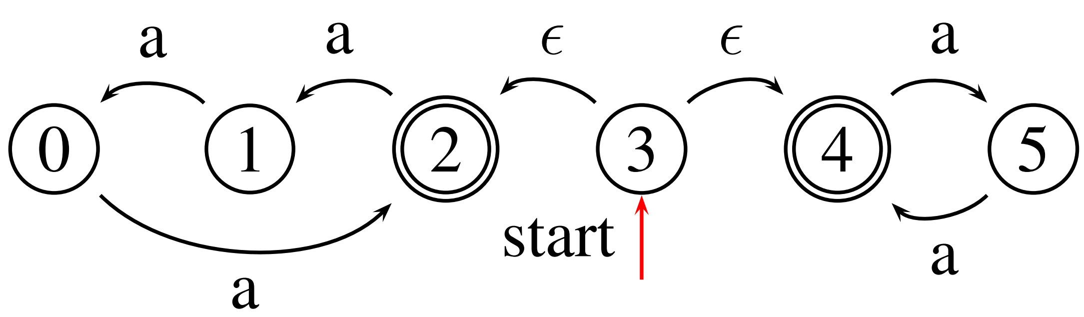


Two Examples

- NFA 1:

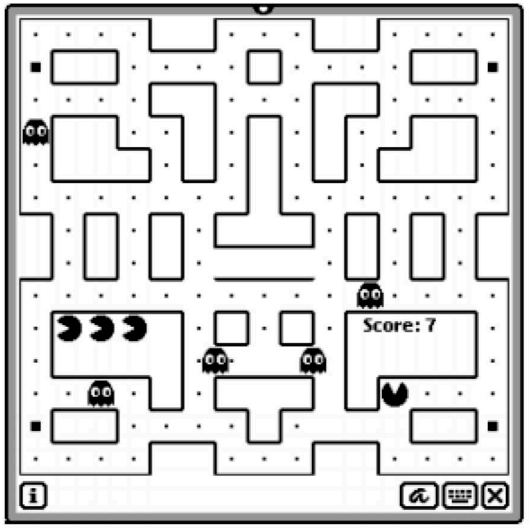


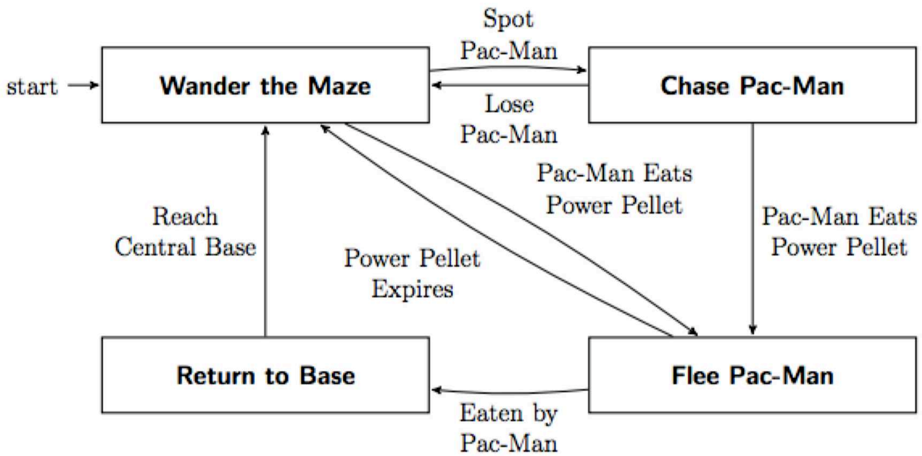
- NFA 2:



- The same language:
  the set of all strings of a's such that the length of each of these strings is a multiple of 2 or 3 (ϵ included)

Real-Life DFAs

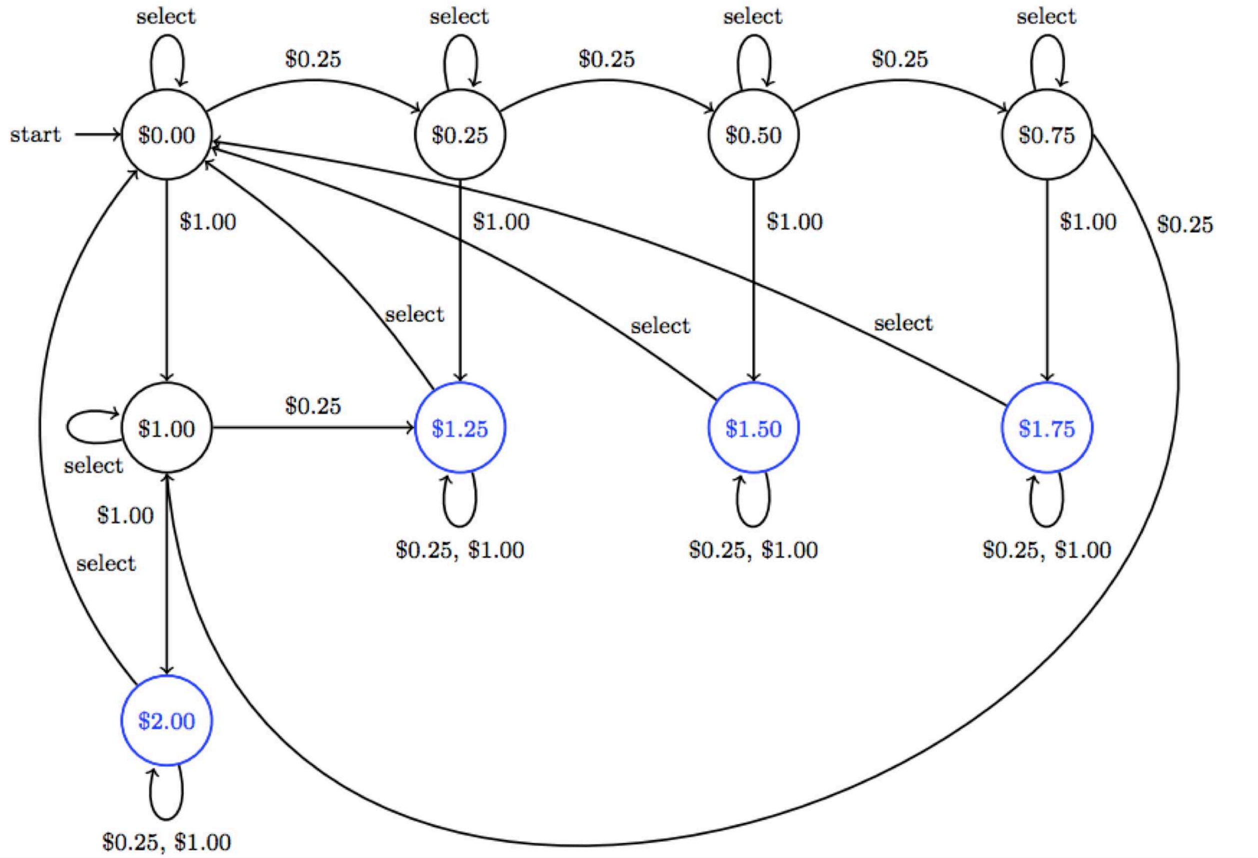




The ghosts in Pac-Man have four behaviors:

1. Randomly wander the maze
2. Chase Pac-Man, when he is within line of sight
3. Flee Pac-Man, after Pac-Man has consumed a power pellet
4. Return to the central base to regenerate

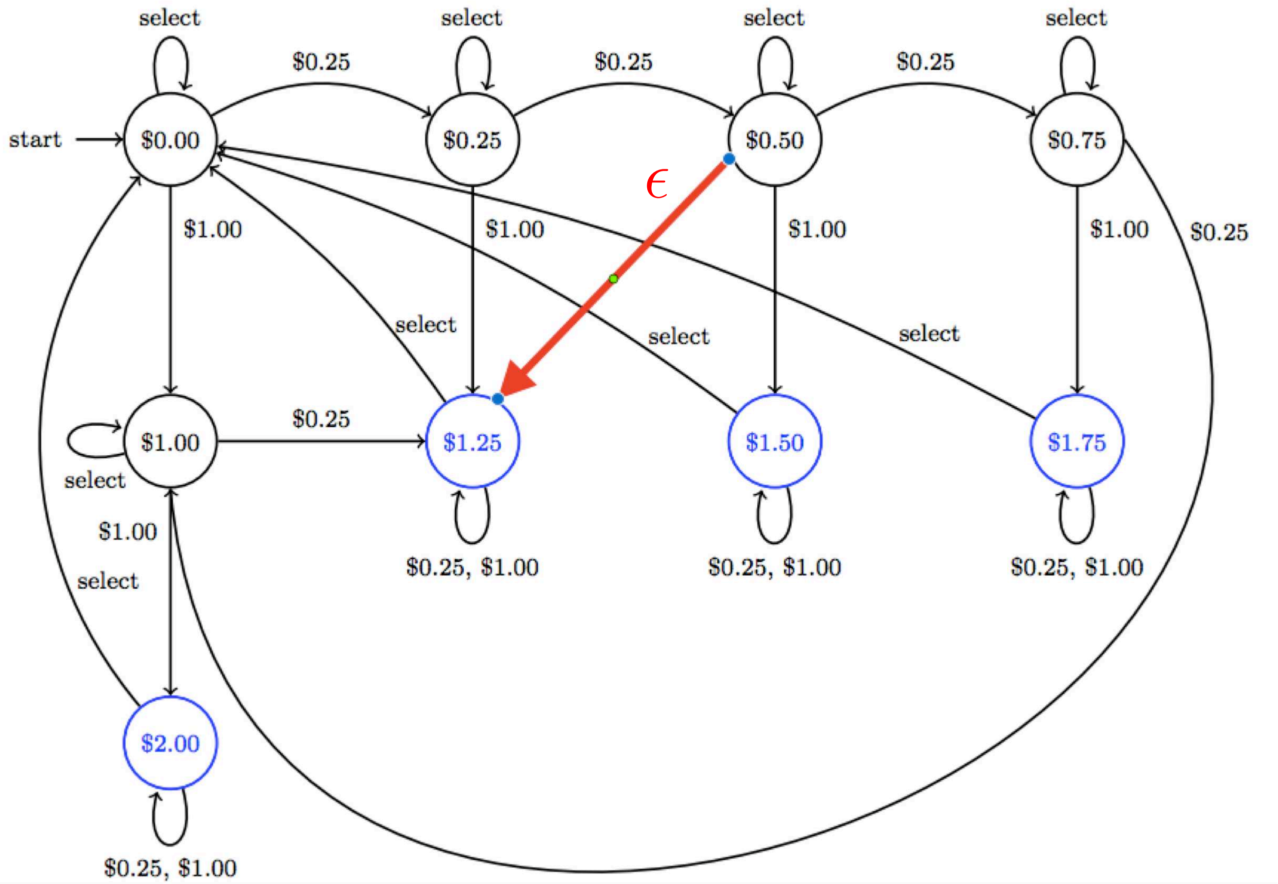
Real-Life DFAs



The behavior of a vending machine:

`accepts dollars and 25 cents, and charges $1.25 per coke.`

What About this Non-Real-Life NFA?



Week 1 (2nd): Regular Expressions, DFA and NFA

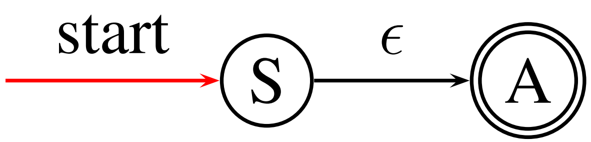
1. Definitions of REs, DFA and NFA $\surd$
2. REs $\Longrightarrow$ NFA (*Thompson's construction, Algorithm 3.3, Red Dragon/Algorithm 3.23, Purple Dragon*)

Thompson's Construction of NFA from REs

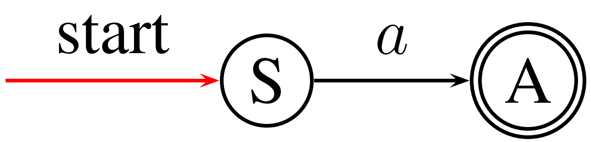
- Syntax-driven
- Inductive: The cases in the construction of the NFA follow the cases in the definition of REs
- Thompson's method is one of many available

Thompson's Construction

- Inductive Base:
  1. For ϵ, construct the NFA



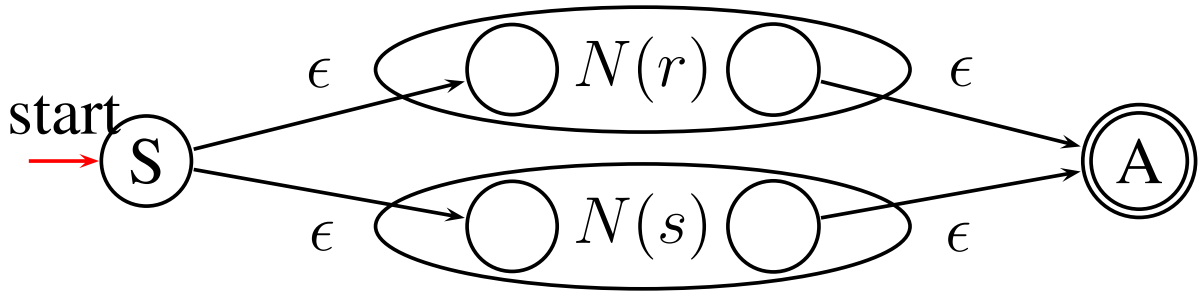
  2. For $a \in \Sigma$, construct the NFA



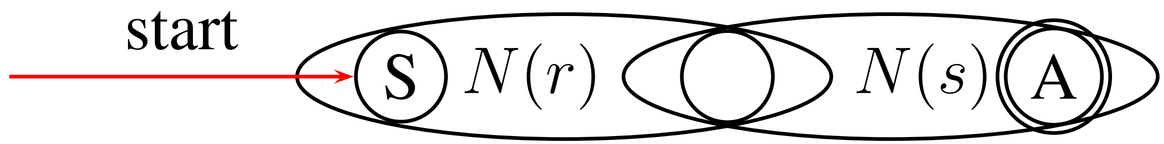
- Inductive step: suppose $N(r)$ and $N(s)$ are NFAs for REs r and s. Then

Thompson's Construction (Cont'd)

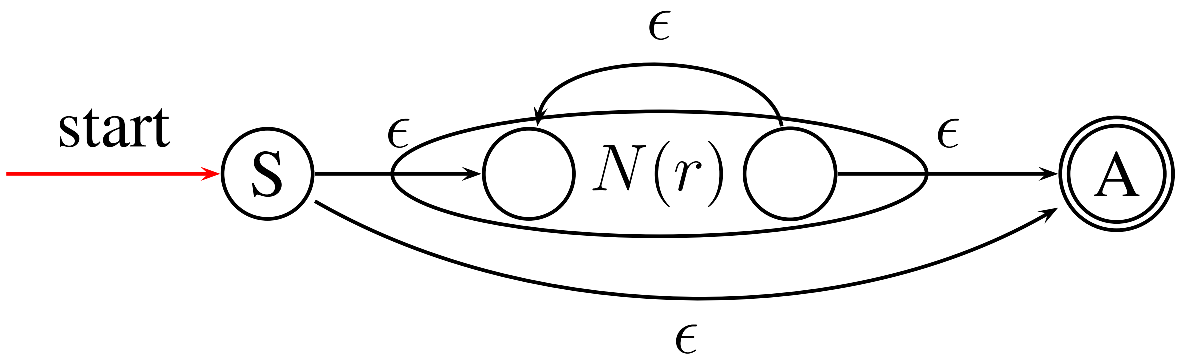
RE $r|s$:



RE rs :



RE r^* :



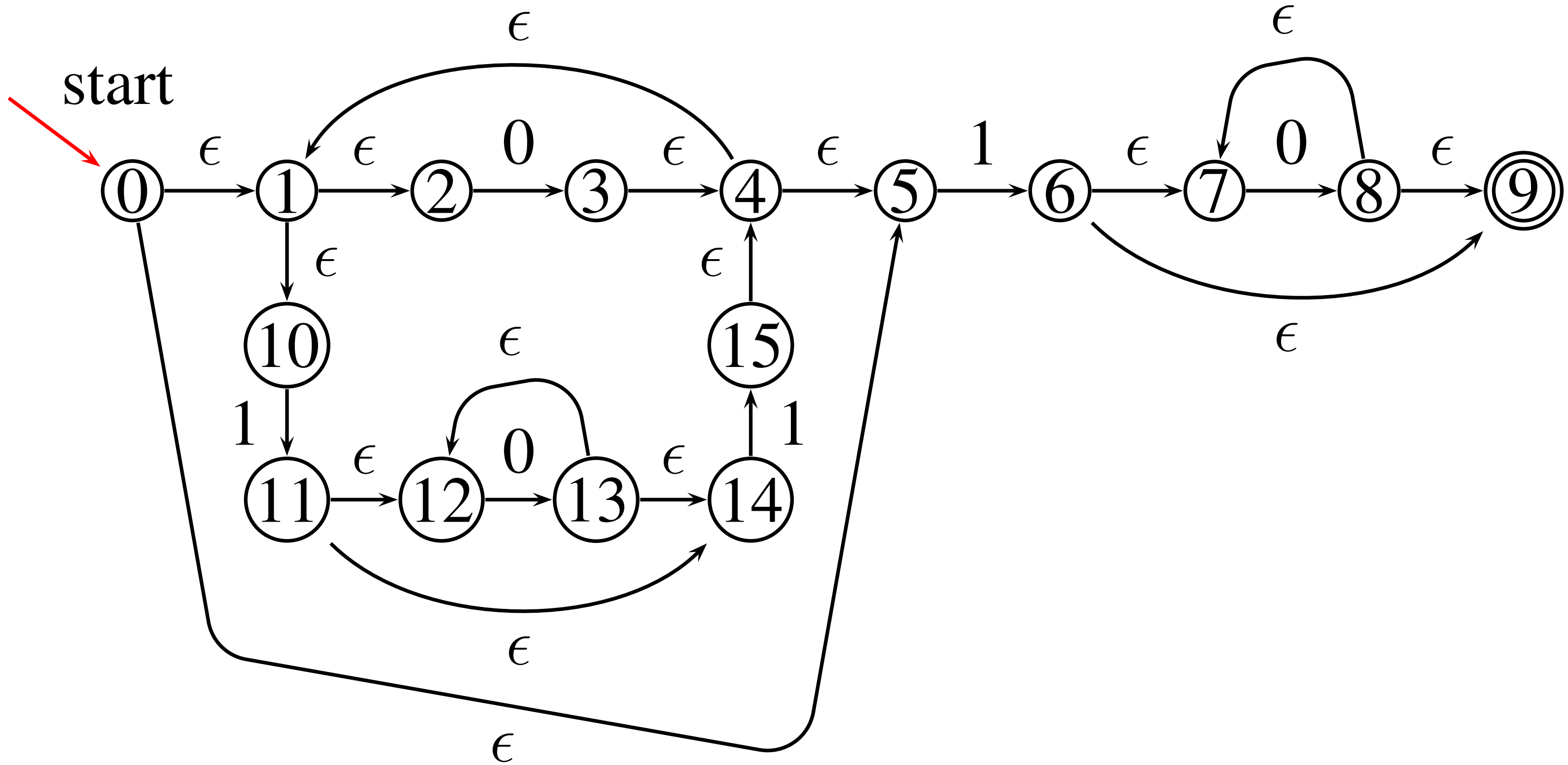
RE (r) : $N((r))$ is the same as $N(r)$

Example: RE $\Longrightarrow$ NFA

Converting $(0|10^*1)^*10^*$ to an NFA

Example: RE $\Longrightarrow$ NFA

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



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 2):

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

Chomsky's Hierarchy

Depending on the form of production

$$\alpha \to \beta$$

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	$\|\alpha\| \leq \|\beta\|$	Type 1	linear bounded automaton
Type 2	context-free grammar CFGs	$A \to \alpha$	Type 2	stack automaton
Type 3	Regular grammars	$A \to w \mid Bw$	Type 3	finite state automaton

Reading

- Sections 3.3 – 3.7 of either Dragon Book
- Week 2 tutorial questions (available on-line)

Next Lecture: Context-Free Grammars