Principles of Programming Languages

Topic: Formal Languages, Part C

Reminder: Recitation

- **All sections as scheduled**
  - **Today section 03 and 04 (04 already met today)**
  - **Tuesday section 01 and 02 (01 already met last Tuesday)**

Review: Regular Expressions

- "Language" = "set of strings"
- Languages can be infinite sets. If a language is infinite we can't just list all strings in the language
- Formalisms for specifying a language
  - Grammars
  - Regular Expressions
  - Automata

Regular Expressions

- Formalism for describing simple PL constructs:
  - e.g., identifiers, numbers
- Simplest sort of grammatical structures
- Regular expressions actually are expressions
  - value of an RE is a set of strings
  - operators in an RE take sets of strings as their operands

Regular Expressions

- **For the alphabet a, b, c:**

RE	Meaning
a	{ “a” }
ε	{“”}
a | b	{“a”} ∪ {“b”} = {“a”, “b”}
a b	any from {“a”} followed by any from {“b”} = {“ab”}
a*	zero or more from {“a”} in sequence = {“”, “a”, “aa”, ...}
a+	one or more from {“a”} in sequence = {“a”, “aa”, ...}

Regular Expressions

- **For the alphabet a, b, c:**

RE	Meaning
(a | b) c	any from {“a”, “b”} followed by any from {“c”} = {“ac”, “bc”}
ab | ac	{ “ab”, “ac” }
(a | b)(a | d)	{“aa”, “ad”, “ba”, “bd”}
(aa | ab) | ((a|b)(a|d))	{“aa”, “ad”, “ba”, “bd”, “ab”} only one “aa”
(abc | ε) d	{“abcd”, “d”}
(a | ε)(b | ε)	
ε*	
(a | b)+	
a b+	

Regular Expressions

- **Which of these are in the language defined by** ***_ * (a | b) (a | b | 1 | 2)****

  _ _ 3ab   a21   a   _ ab _ 2

- **Describe the language 1(0|1)* in English**
- **Write an RE which describes the language "binary numbers with at least two bits, with ones and zeros alternating.**

RE's for PLs

- **Let *letter* stand for a|b|c|…|a|A|... |Z and *digit* stand for 0|1|2|3|4|5|6|7|8|9**
- **Which of these are in the language *digit* * *. digit* $^+$**

  0.6 -3.8 .25 23.
- **Which of these are in the language *(letter | _)(letter | digit)*****

  aA3 a3A _xyz a_B 9x _9x

Uses of Regular Expressions

- Theoretical constructs
- Describe parts of programming languages
- General pattern matcher for strings
  - Unix utility grep: find all lines in a file that contain a RE

```
grep (Lou|Louis)b(“”|Ib|Irab)Steinberg
```

would* find

```
Lou Steinberg: cs314
Prof.  Louis I Steinberg
```

but not

```
Prof.  Louis I. Steinberg
```

* actually this does not work: grep uses different RE syntax

Formal Languages

- Three related formalisms:
  - Grammar
  - Regular Expression
  - Automata

New: Automata

- **Another formalism for reasoning about the difficulty of computational problems**
  - **it inputs a string, one character at a time**
  - **it outputs a boolean: is the string in the language?**
- **An automaton is also:**
  - **A way to structure programs that recognize patterns in strings**
  - **A way to specify a token in a programming language**

Automaton

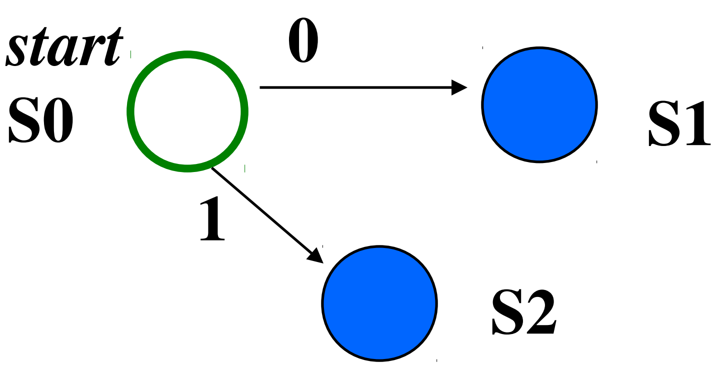
- An automaton has
  - input: one symbol at a time
  - A (finite) set of states that it can be in
  - a set of transition rules: (current char, state) → next state
  - may have other memory (stack, tape) – *but not in 314*
- Classes of automata
  - based on what other memory
    - none → Finite State Automaton – FSA or FA
  - class of automaton <=> what kind of languages it can recognize
    - FAs recognize same languages as REs

Finite State Automaton (FA)

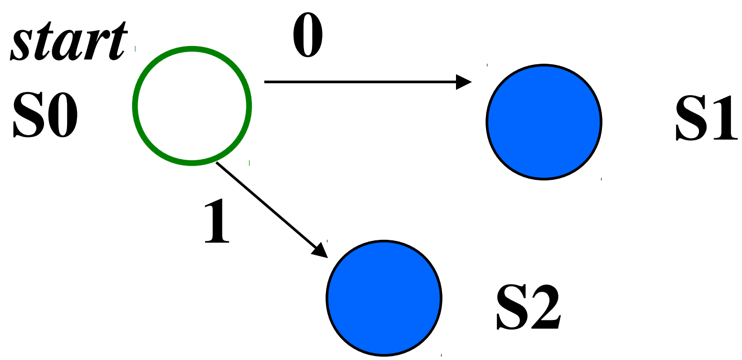
- **Described by**

  <set of states, labeled transitions, start state, final state(s)>
- **Example:**

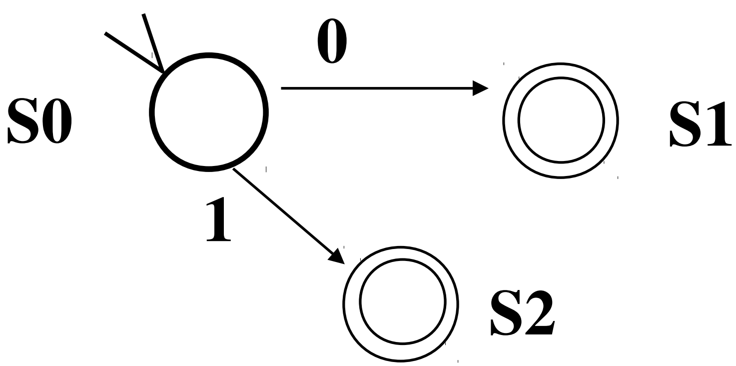
  <{S0,S1,S2}, $\boxed{\begin{array}{l} S0 \xrightarrow{0} S1 \\ S0 \xrightarrow{1} S2 \end{array}}$ **, S0, {S1,S2}>**

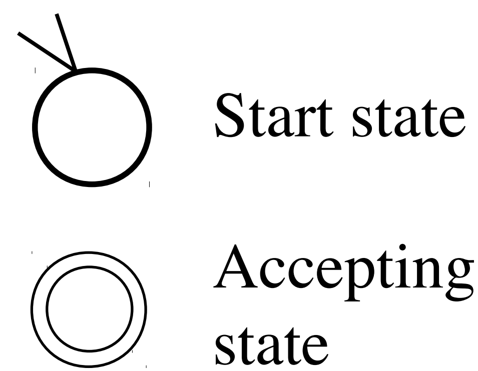


Another Convention for Drawing

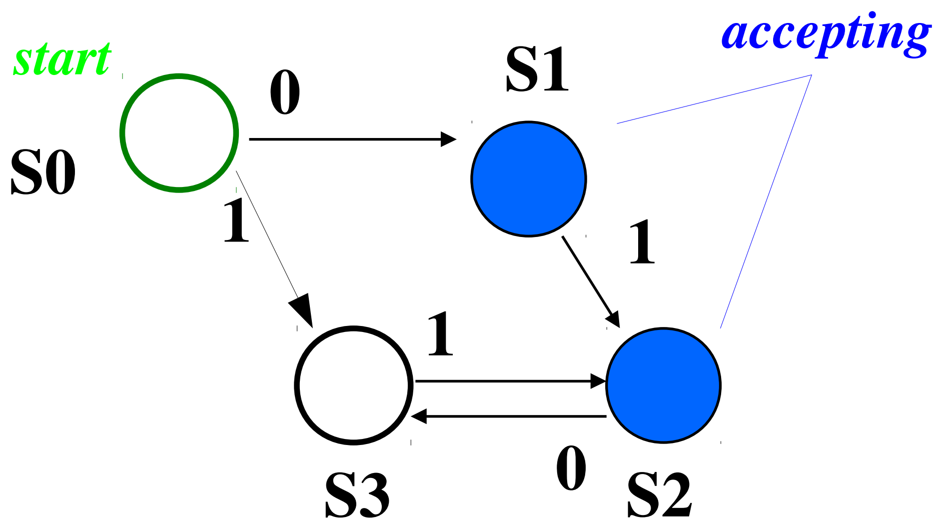


Can also be drawn as:





Finite State Automaton (FA)

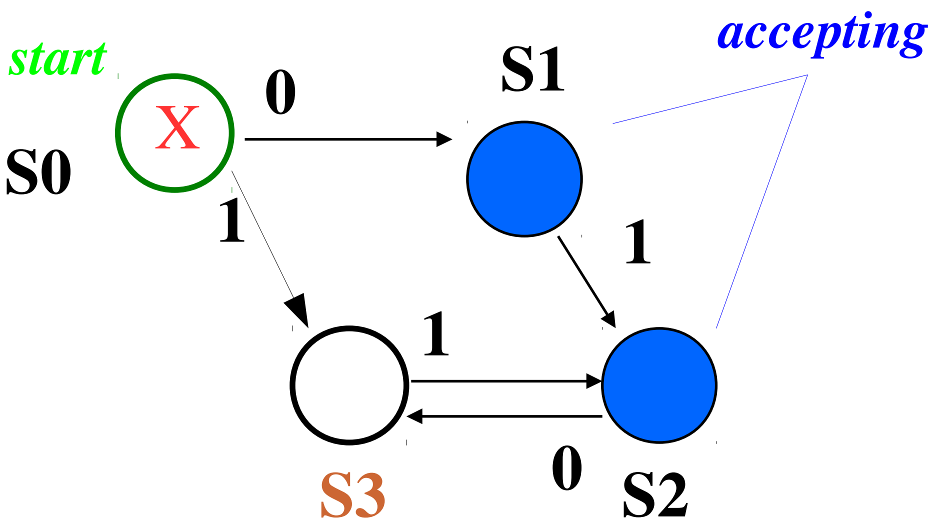


transition table

states:	inputs: 0	1
S0	**S1**	**S3**
S1	**---**	**S2**
S2	**S3**	**---**
S3	**---**	**S2**

1 1 0 1

Finite State Automaton (FA)

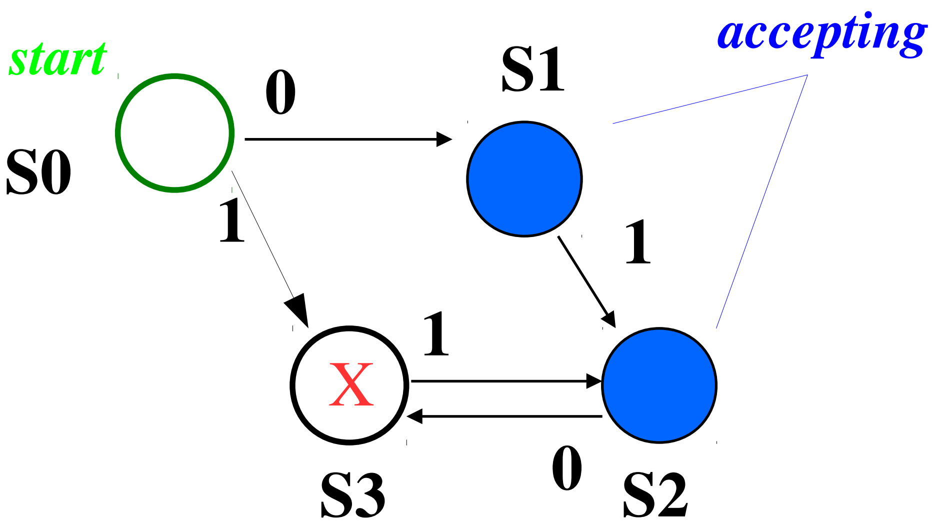


transition table

states:	inputs: 0	1
S0	S1	S3
S1	---	S2
S2	S3	---
S3	---	S2

1 1 0 1

Finite State Automaton (FA)

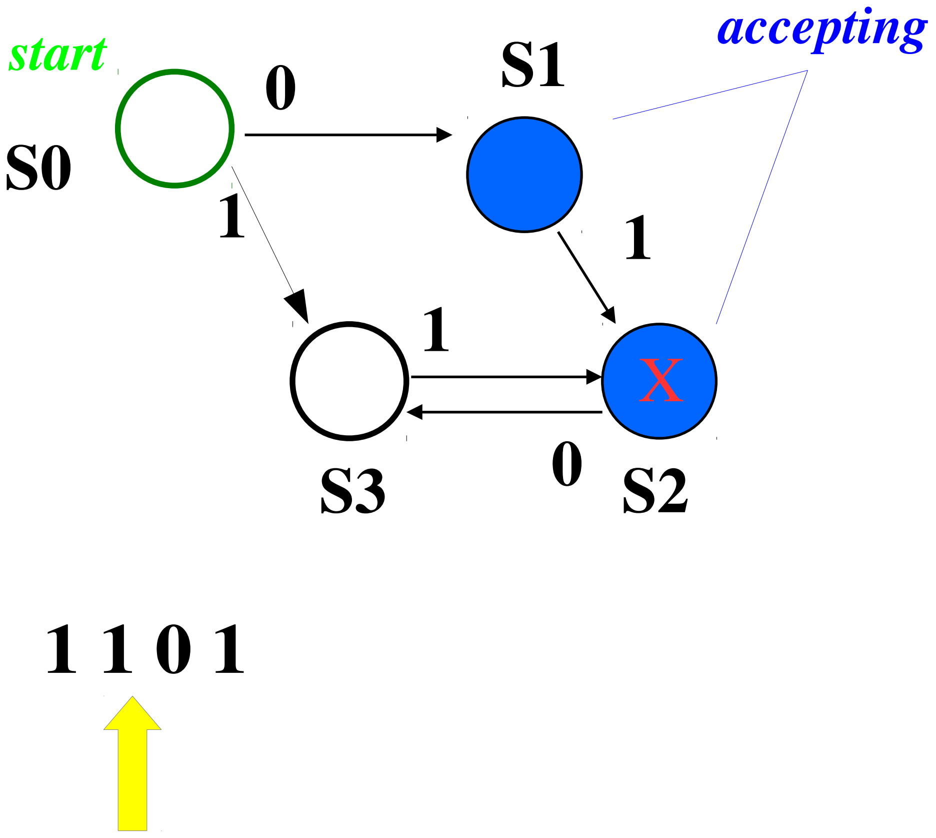


transition table

states:	inputs: 0	1
S0	S1	S3
S1	---	S2
S2	S3	---
S3	---	S2

1 1 0 1

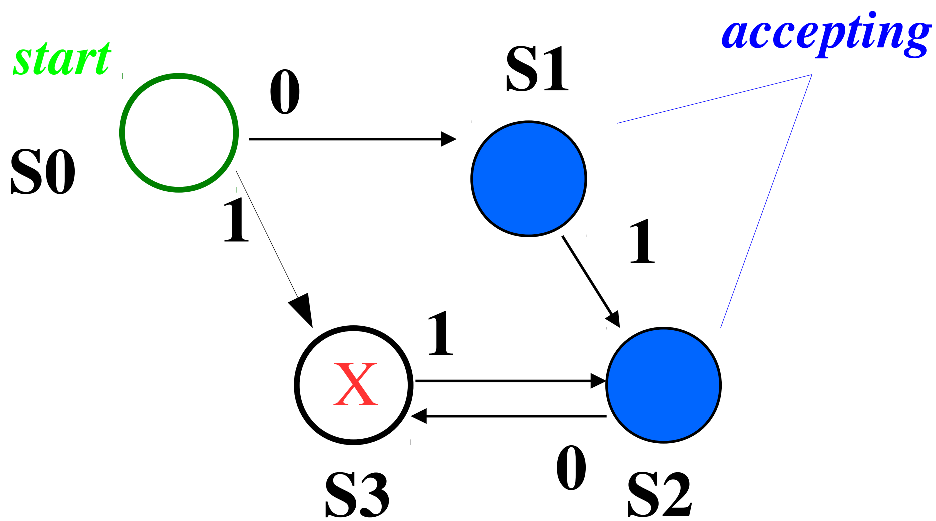
Finite State Automaton (FA)



transition table

states:	inputs: 0	1
S0	**S1**	**S3**
S1	**---**	**S2**
S2	**S3**	**---**
S3	**---**	**S2**

Finite State Automaton (FA)

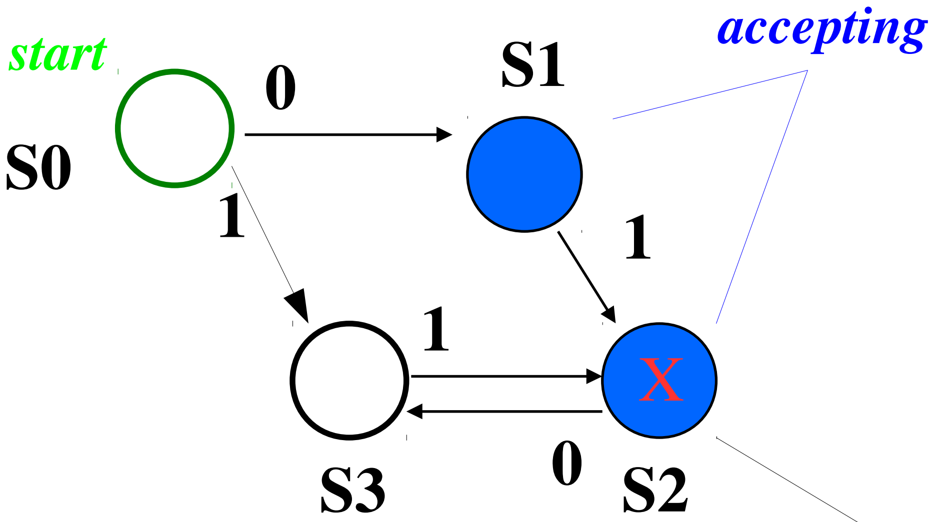


transition table

states:	inputs: 0	1
S0	**S1**	**S3**
S1	**---**	**S2**
S2	**S3**	**---**
S3	**---**	**S2**

1 1 0 1

Finite State Automaton (FA)



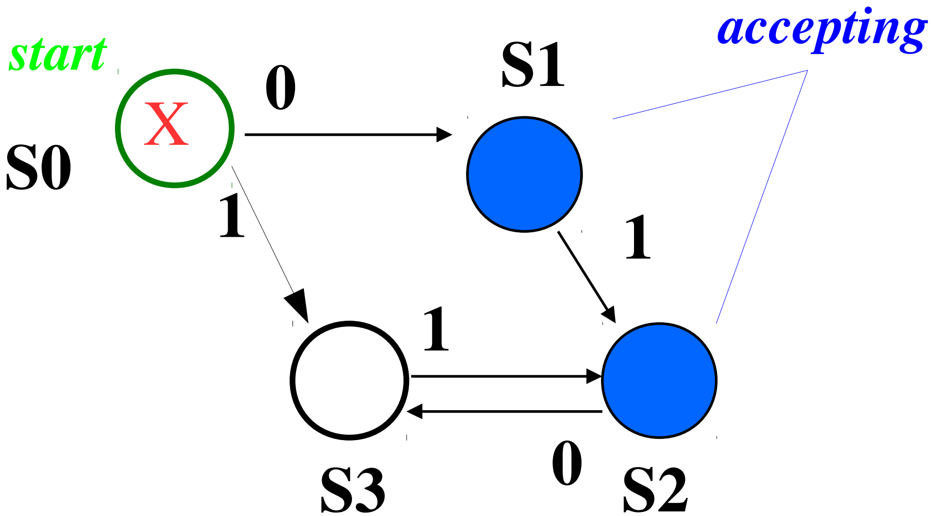
transition table

states:	inputs: 0	1
S0	**S1**	**S3**
S1	**---**	**S2**
S2	**S3**	**---**
S3	**---**	**S2**

1 1 0 1

Ends in an accepting state: the string 1101 is in the language of this FA

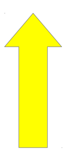
Finite State Automaton (FA)



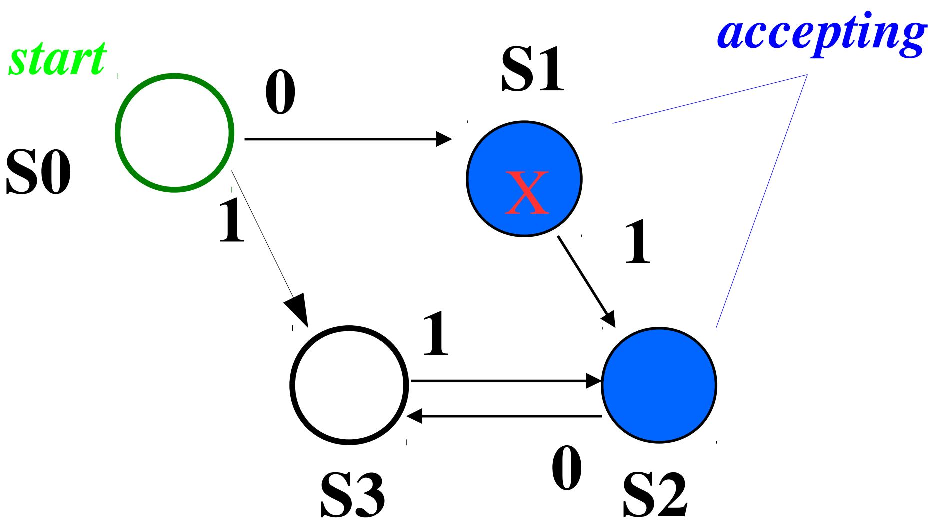
transition table

states:	inputs: 0	1
S0	**S1**	**S3**
S1	**---**	**S2**
S2	**S3**	**---**
S3	**---**	**S2**

0 1 0



Finite State Automaton (FA)

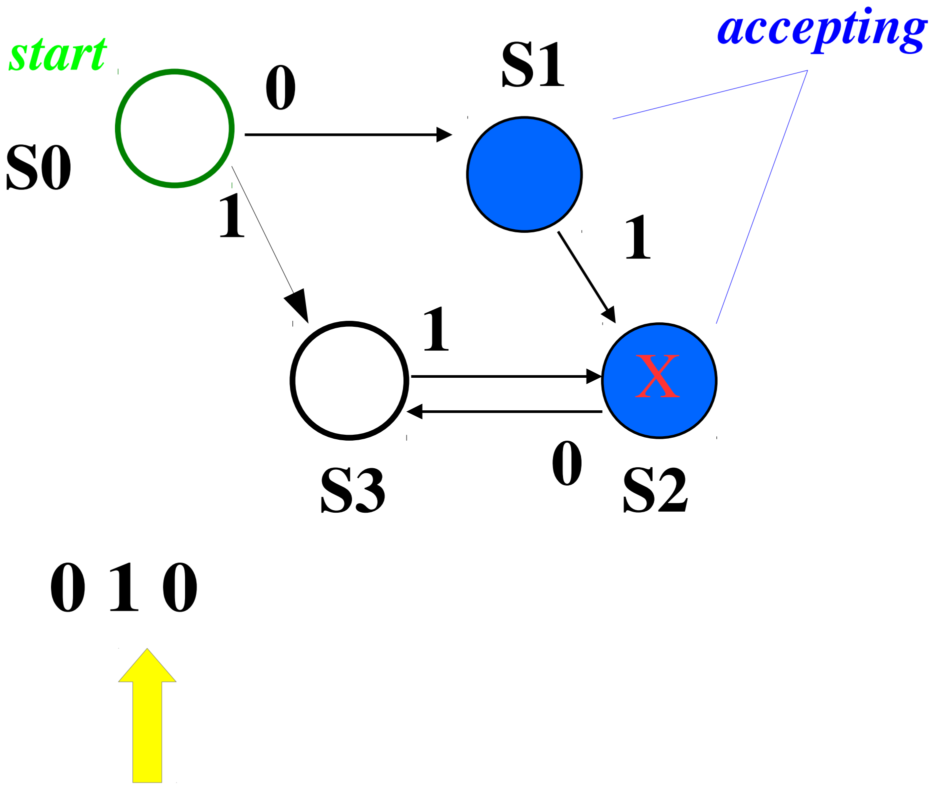


transition table

states:	inputs: 0	1
S0	S1	S3
S1	---	S2
S2	S3	---
S3	---	S2

0 1 0

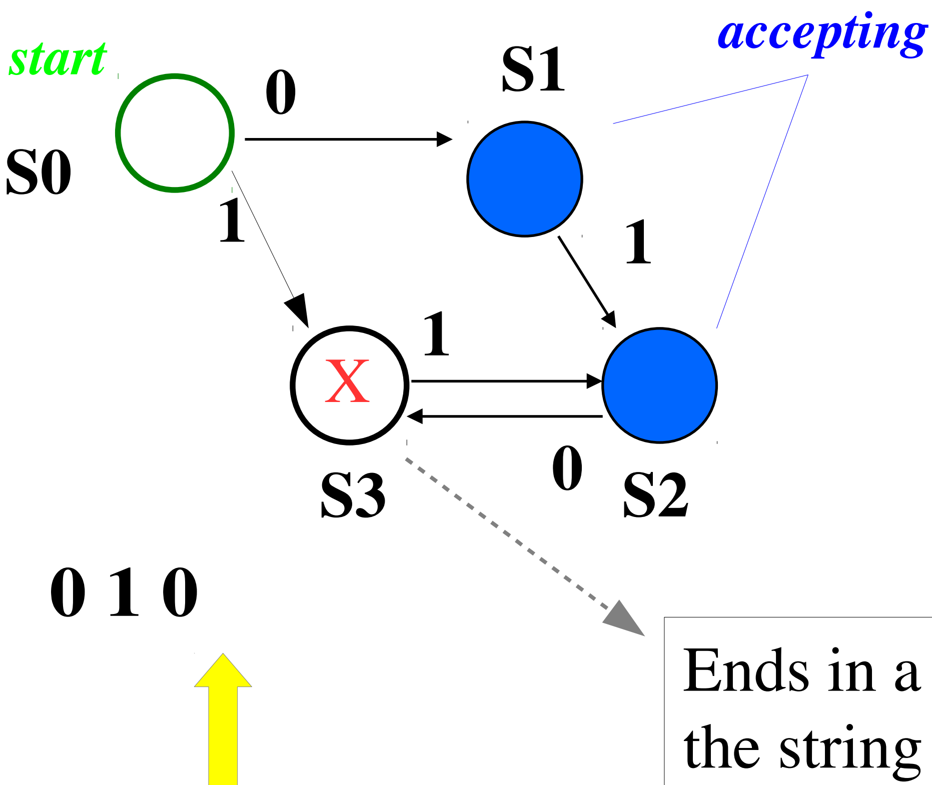
Finite State Automaton (FA)



transition table

states:	inputs: 0	1
S0	S1	S3
S1	---	S2
S2	S3	---
S3	---	S2

Finite State Automaton (FA)



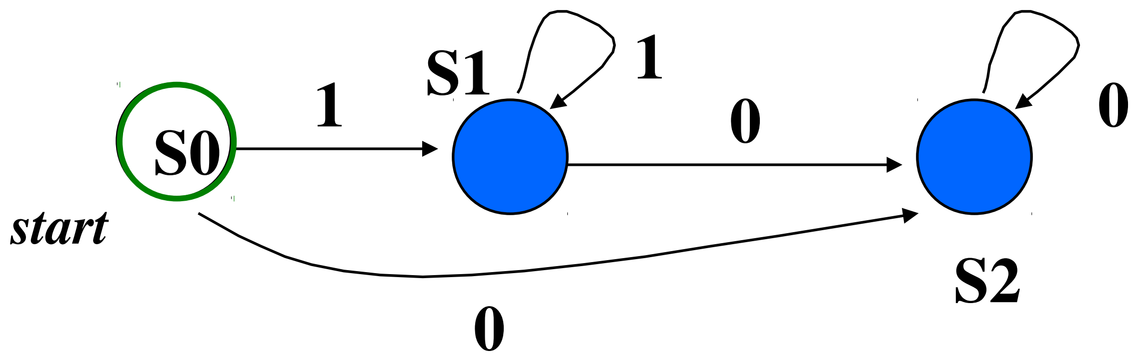
transition table

states:	inputs: 0	1
S0	**S1**	**S3**
S1	**---**	**S2**
S2	**S3**	**---**
S3	**---**	**S2**

Ends in a **non**accepting state: the string 010 is **not** in the language of this FA

Finite State Automaton (FA)

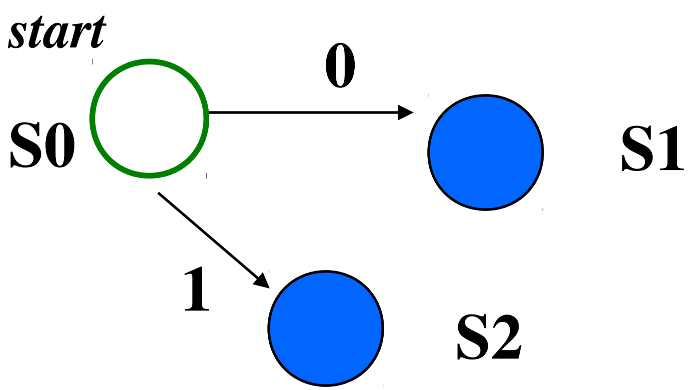
Binary numbers containing at least one digit, in which all the 1's precede all the 0's:



Recognizes: (0^+) | ($1^+ 0^*$)

Finite State Automaton (FA)

- FA *accepts* or *recognizes* an input string iff there is a path from its start state to a final state such that the labels on the path are the terminals in that string.



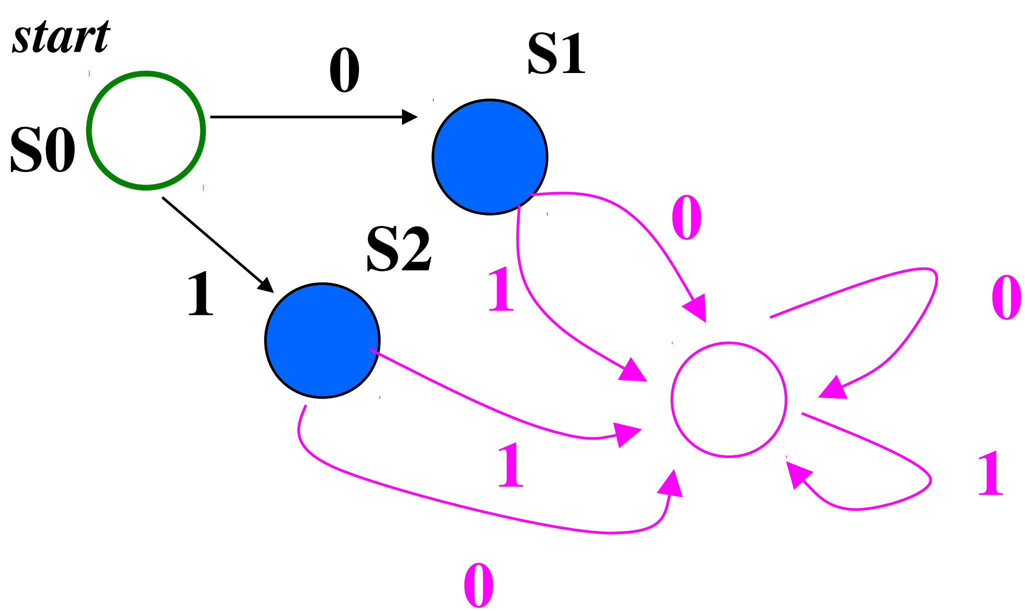
states:	inputs: 0	1
S0	**S1**	**S2**
S1	**---**	**---**
S2	**---**	**---**

transition table

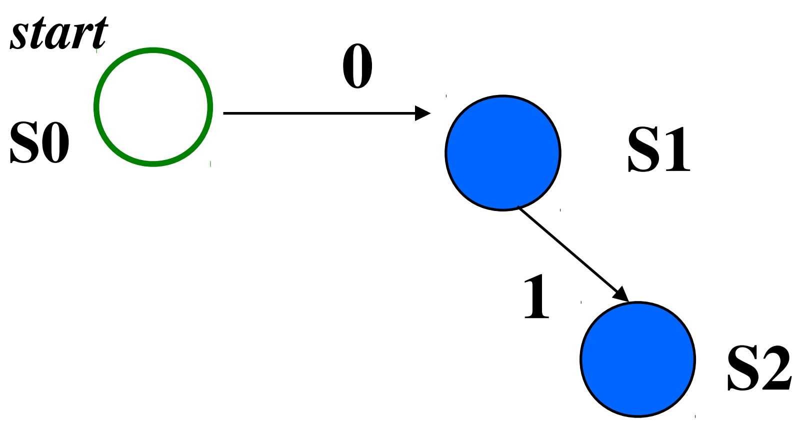
- **What strings are recognized?**

Finite State Automaton (FA)

- If there is no transition given for a state/input pair, it implicitly leads to a permanent failure state
  - i.e., recognition fails: the string is not in the language



Finite State Automaton (FA)

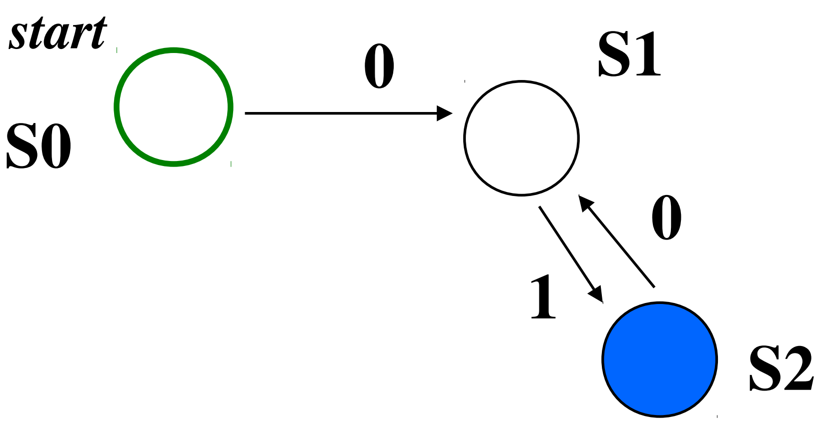


state:	input: 0	1
S0	S1	---
S1	---	S2
S2	---	---

transition table

- **What strings are recognized?**

FA



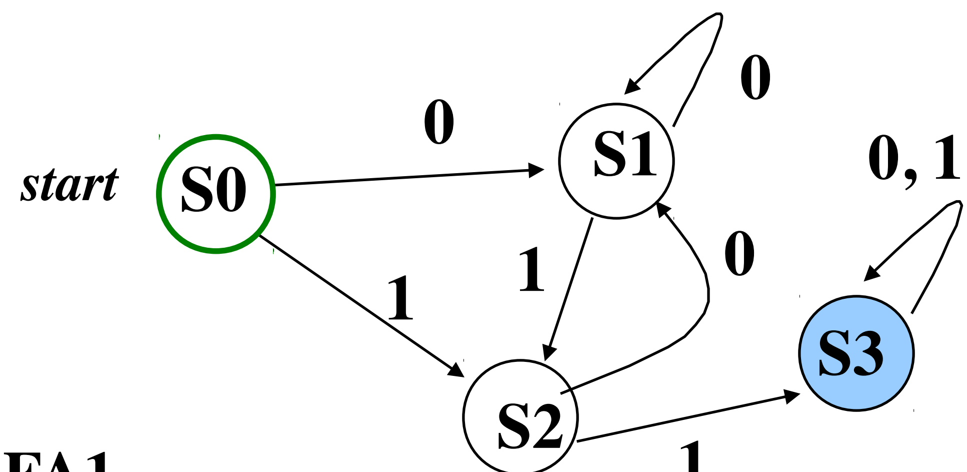
state:	input: 0	1
S0	**S1**	**---**
S1	**---**	**S2**
S2	**S1**	**---**

transition table

- **What strings are recognized?**

Finite State Automaton (FA)

Binary numbers containing a pair of adjacent 1's:

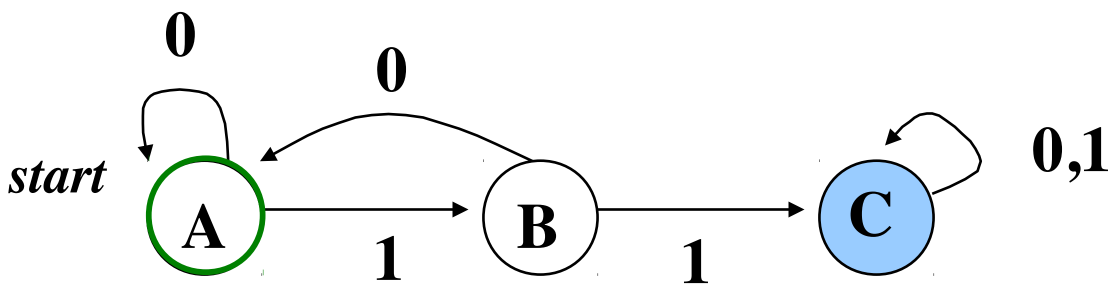


FA1

	0	1
S0	S1	S2
S1	S1	S2
S2	S1	S3
S3	S3	S3

Recognizes same language as RE $(0 \mid 1)^* 1\ 1\ (0 \mid 1)^*$

Binary numbers containing a pair of adjacent 1's:



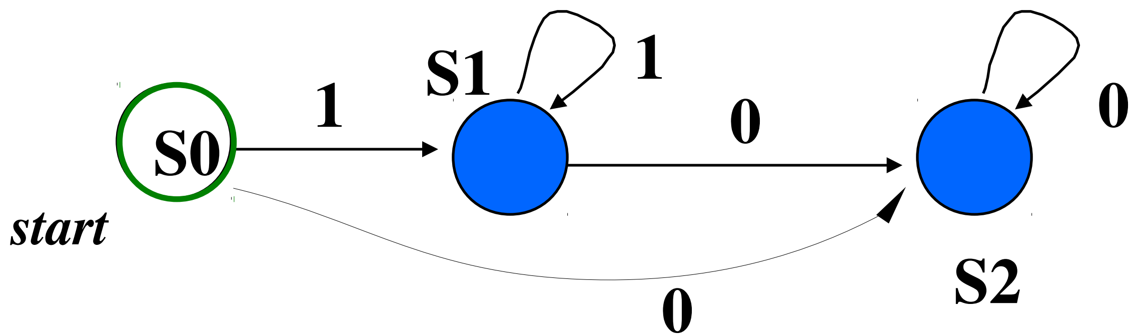
FA2

Recognizes: $(0 \mid 1)^{*}\ 1\ 1\ (0 \mid 1)^{*}$

FA1 and FA2 recognize the same set of strings, i.e., the same language! **Therefore, FAs are *not unique*.**

Finite State Automaton (FA)

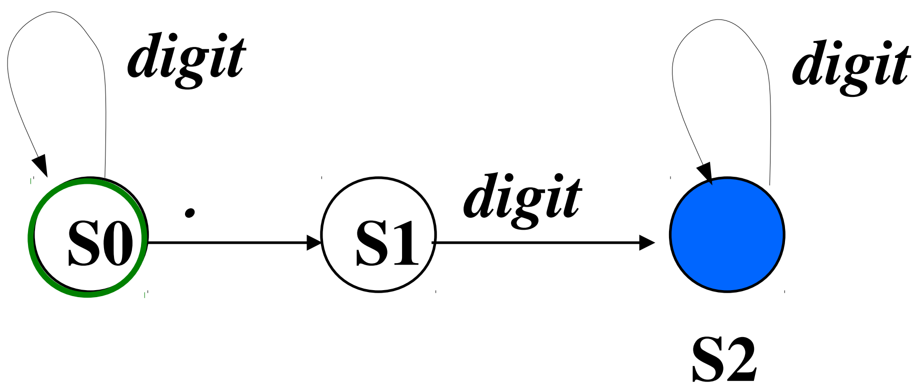
Binary numbers in which the 1's (if any) precede the 0's (if any):



Recognizes: $1^* \; 0^*$

Finite State Automaton (FA)

Real number: **12.34** **0.56** **.7** **but <u>not</u>** **8.**



Recognizes: ***digit *. digit +***

Deterministic & Nondeterministic FAs

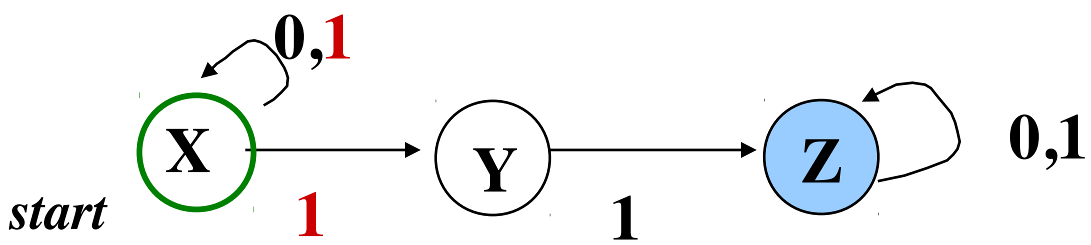
- **So far:**
  - **At most one transition for any state / character pair**
  - **Every transition consumes one input character**

  => *Deterministic* FA (DFA)
- ***Nondeterministic* FA (NFA):**
  - **For some state / character: more than one transition**

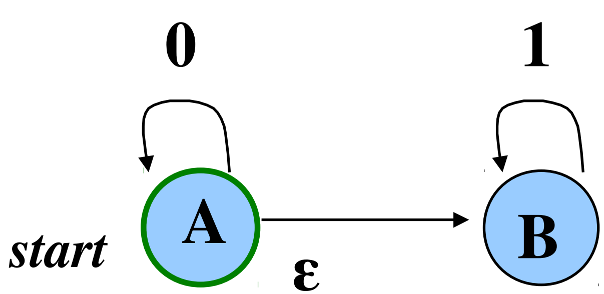
    and / or
  - **ε moves: transition that does not consume a character**
  - **NFA accepts a string if *ANY* sequence of allowed choices ends in an accepting state**

NFAs

- **Regular Expression: (0 | 1) * 1 1 (0 | 1) ***



- **Regular Expression: 0* 1***



RE => NFA => DFA => RE

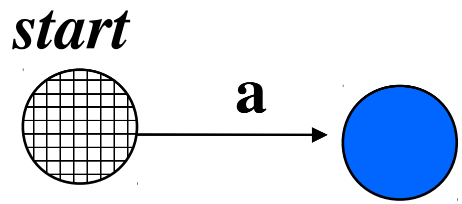
- **If there is a RE that recognizes L,**
  Then there is a NFA that recognizes L
- **If there is an NFA that recognizes L,**
  Then there is a DFA that recognizes L
- **If there is a DFA that recognizes L,**
  Then there is a RE that recognizes L

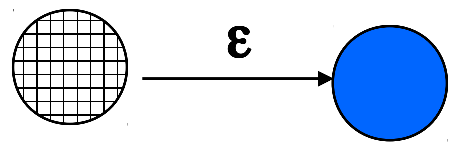
RE to NFA

- **Key idea:**
  - **Build an NFA for each operand**
  - **Put them togeher in a way that depends on the operator**

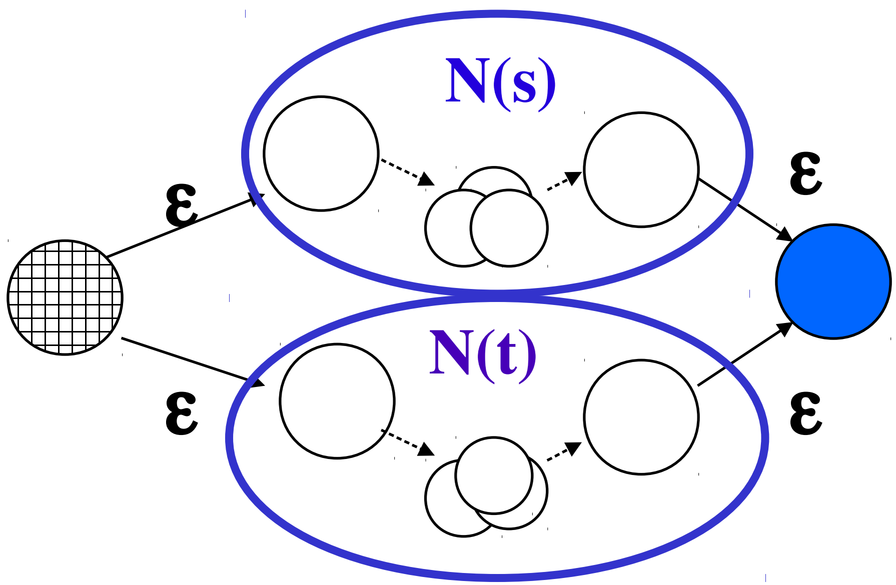
RE to NFA

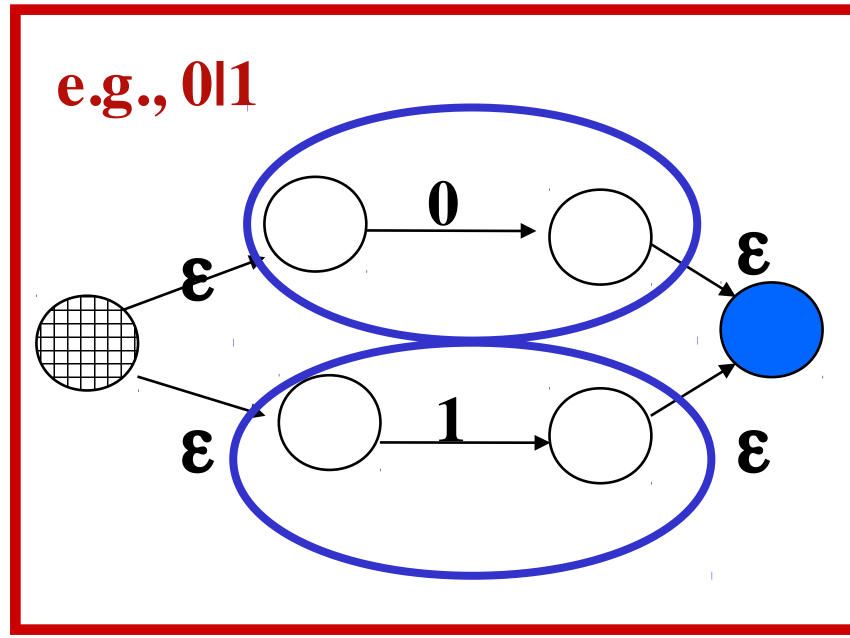
- **If RE is a NFA is:**
- **If RE is ε NFA is:**





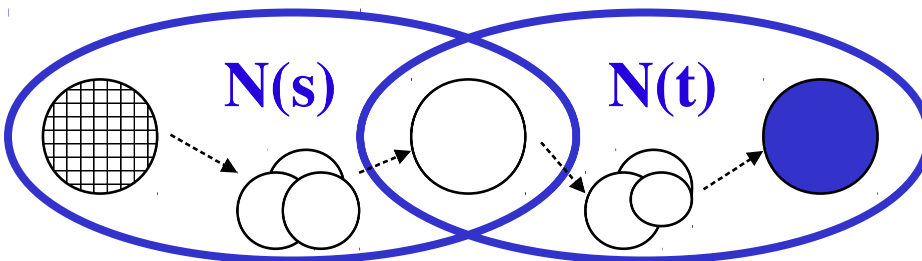
- **For s, t REs, construct s|t:**



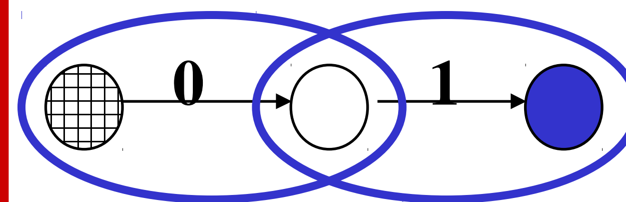


RE to NFA

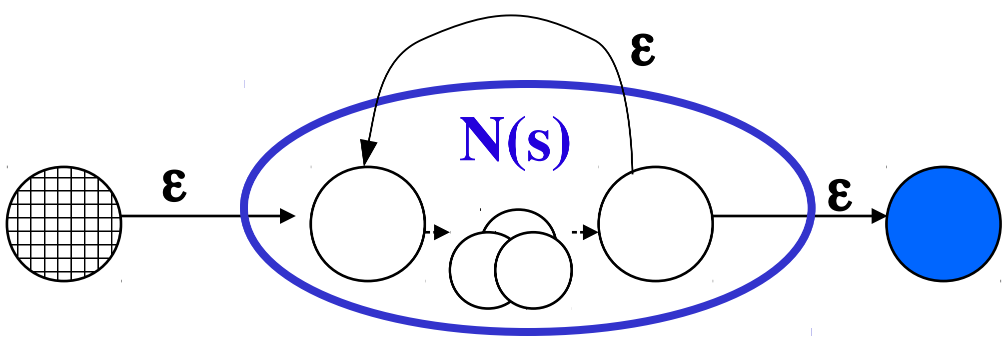
- For **s, t** REs, construct **st**:



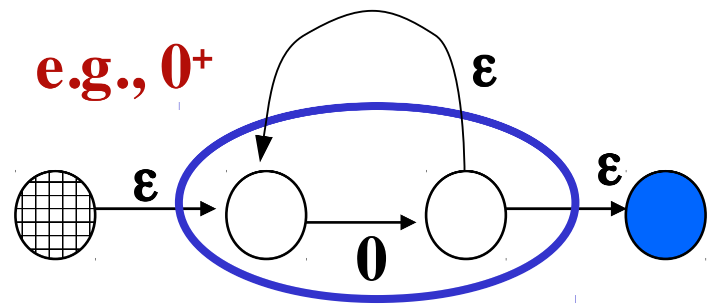
e.g., 01



- For **s** RE, construct **s**$^{+}$:

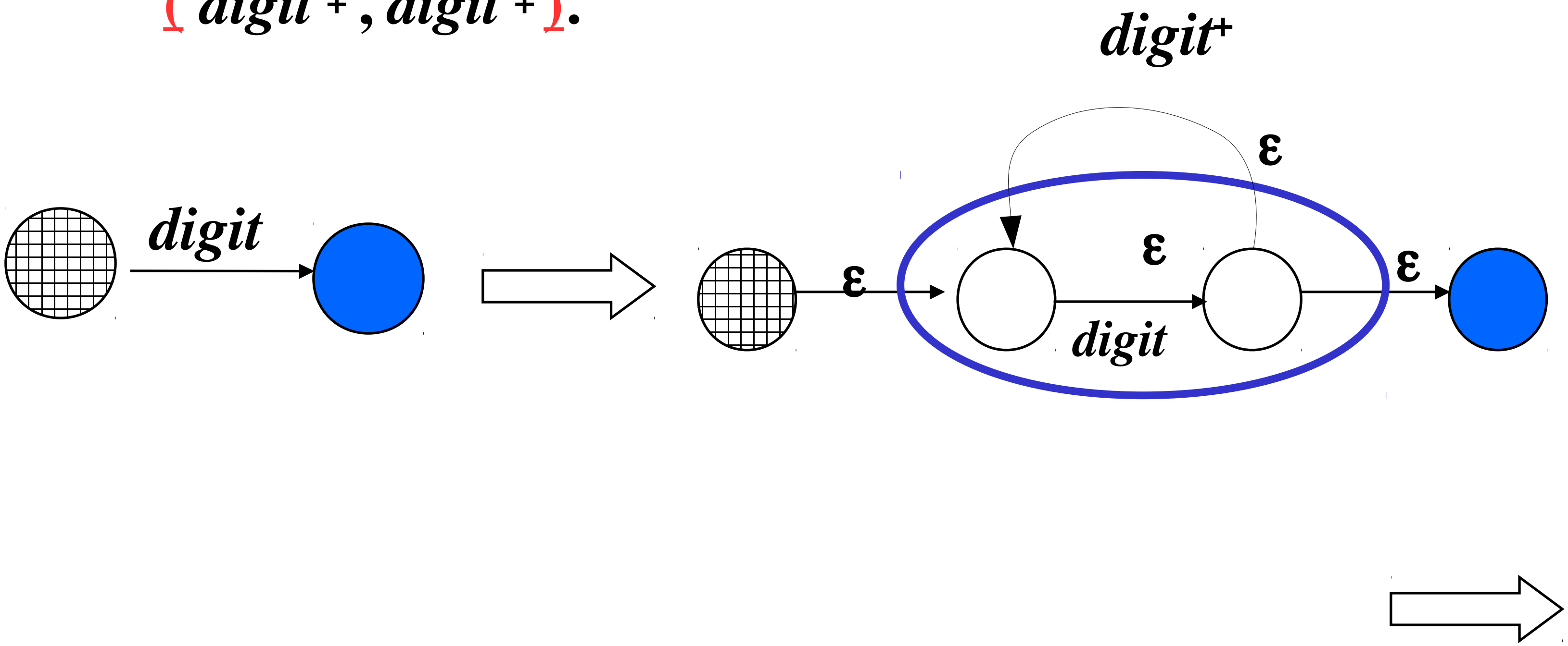


e.g., 0^{+}

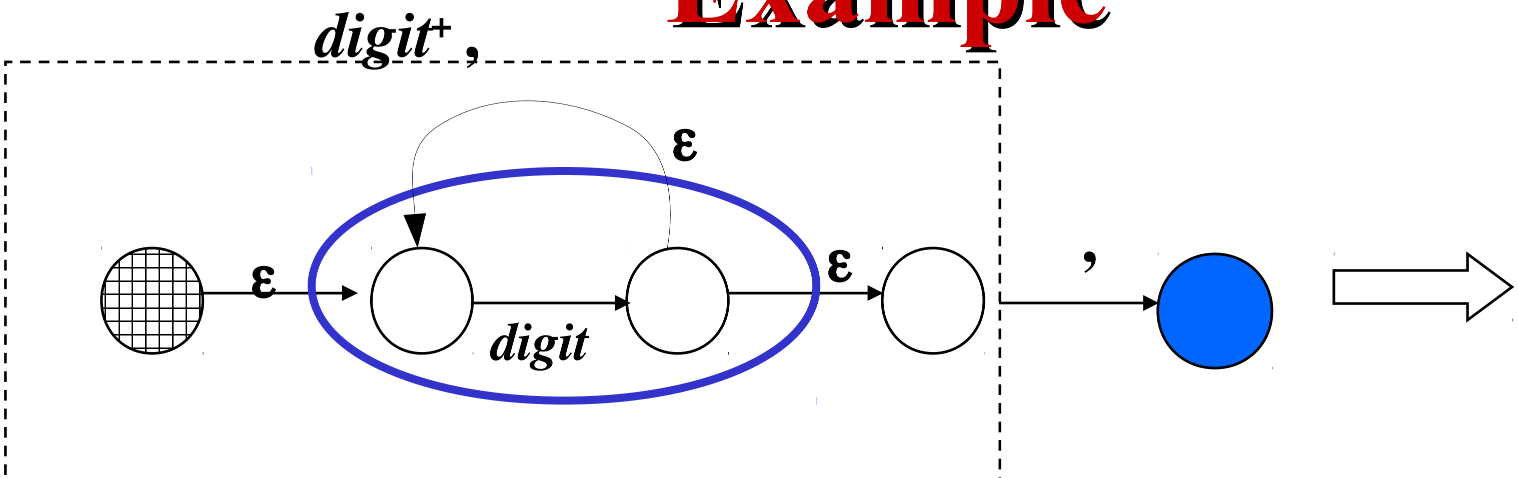


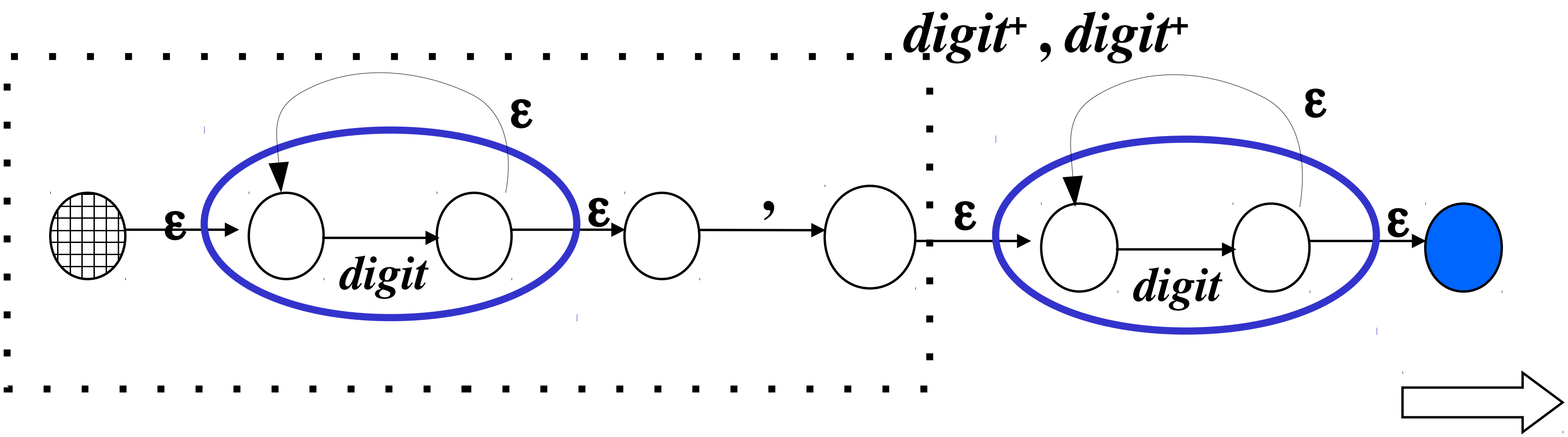
Example

- **Build the NFA for complex numbers using this RE:**
  (*digit* + , *digit* +).



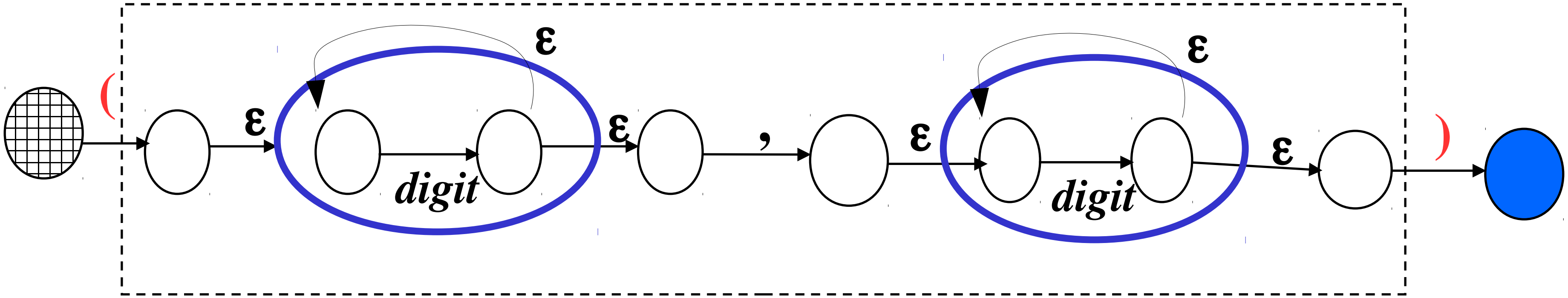
Example





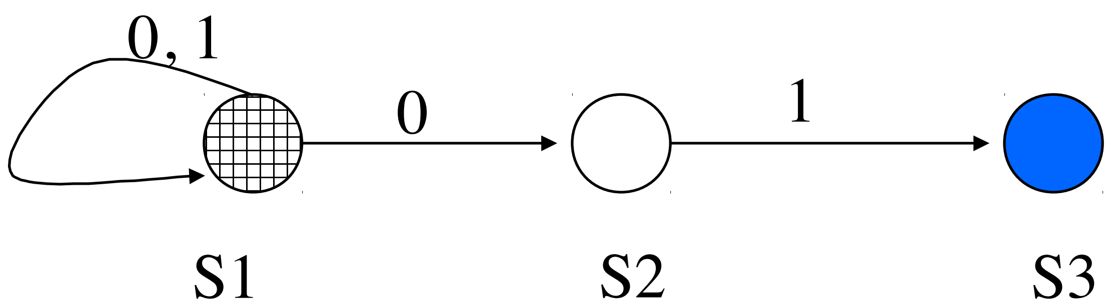
Example

(***digit*** $^{+}$ **,** ***digit*** $^{+}$)

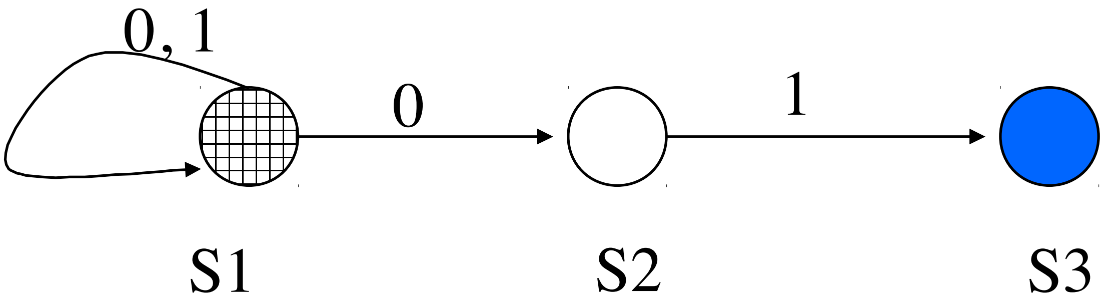


NFA to DFA

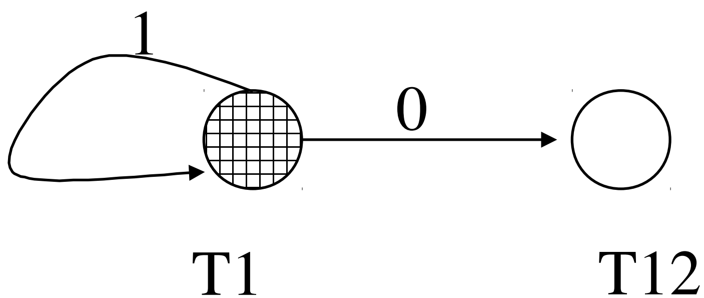
- **Key idea: Each state in DFA corresponds to a *set* of states in the NFA**
- **If you are in a given state of the NFA it means you would be in one of the corresponding states of the DFA, depending on non-deterministic choices**



NFA to DFA



- **Start in S1**
- **If in S1, 1=>S1 but 0=>S1 or S2**



Not all languages have an FA

- **Palindrome: a string that reads the same backwards as forwards**
  - **0110**
  - **0010100**
- **There is no DFA that accepts a string if and only if it is a palindrome**

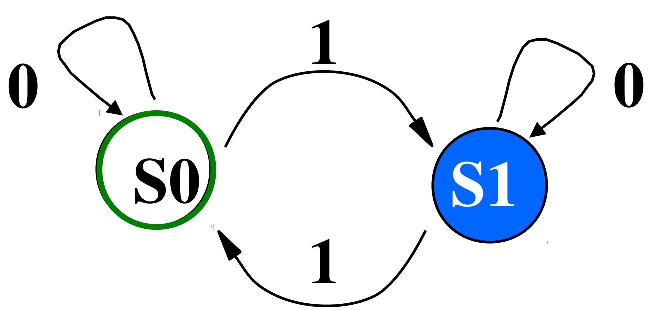
No FA for palindromes

- **Basic idea:**
  - **The only memory a FA has of what it has already seen in the string is what state it is in**
  - **For any specific FA, the number of states is fixed**
  - **So, for any FA, a long enough string will make it run out of states to record the string-so-far**

No DFA for palindromes

Claim 1:

- **Suppose we have**
  - **a DFA, *D*, and**
  - **two strings s1 and s2 that share a common suffix, i.e. for some number c, the last c characters of s1 and s2 are the same.**
- **E.g. D:**



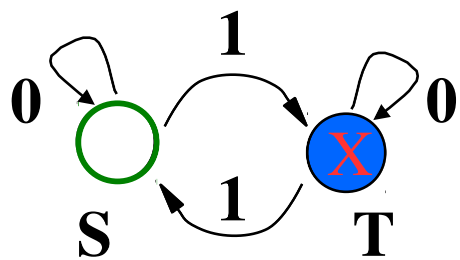
s1: 001 <u>100</u>

s2: 100 <u>100</u>

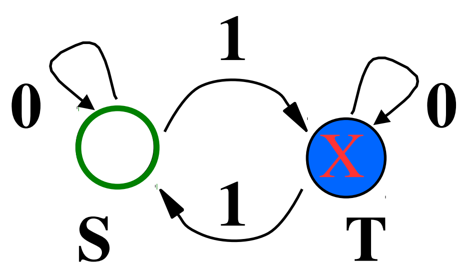
No DFA for palindromes

Claim 1 (continued):

- **Compare D processing s1 with D processing s2.**
  - **Suppose, on entering the suffix, D is in the same state, T, for both strings**



s1: 001 100



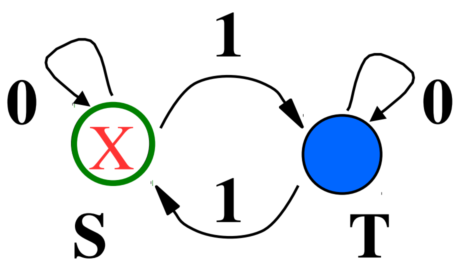
s2: 100 100

No DFA for palindromes

Claim 1 (continued):

- **Then D will end up in the same state when processing s1 as when processing s2**

D:



s1: 001 100

s2: 100 100

No DFA for palindromes

Claim 2:

- **For any given DFA D, there are at least two strings that result in the same state**
  - **Proof: Let N be the number of states in D. Consider any set of N+1 strings. There are more strings than states, so at least two strings have to share the same final state**

String number	String	Final State
1	01	T
2	110	S
3	111	T

No DFA for palindromes

Suppose that some DFA, P, does recognize the set "Palindromes"

- Let s and t be two strings such that P ends up in the same state on both. Let r be the reverse of s. But P processing sr is in some state T when it gets to r, and P processing tr is in that same state when it gets to r, so P has to wind up in the same state at the end of sr as of tr.

But sr is a palindrome and tr is not, and one state cannot be both accepting and non-accepting.

E.g. s: 001, t: 100, r: 100, sr: 001100, tr: 001001

No RE for palindromes

- There is no RE that describes the language “strings that are palindromes”
  - Proof: If there was such an RE it could be translated into a FA that accepts a string if and only if it is a palindrome, and we just proved there was no such FA

Tasks for REs and FAs

Things you need to know how to do for exams:

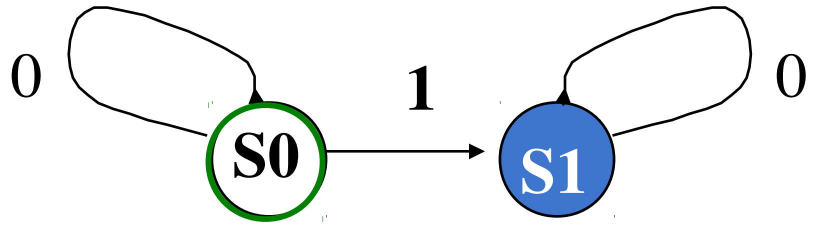
- **Recognition of a string**
  - **Is this given string in the language of this given RE or FA?**
- **Description of a language**
  - **Given an RE (FA), describe its language in English**
- **Codification of a language**
  - **Given a language described in English, find an RE and an FA that corresponds to it**

Tasks for REs and FAs

- **Recognition of a string**
  - **Given the RE 10*, which of these strings are in its language?**

    1, 00, 10, 1000, 01

  - **Given the following FA:**



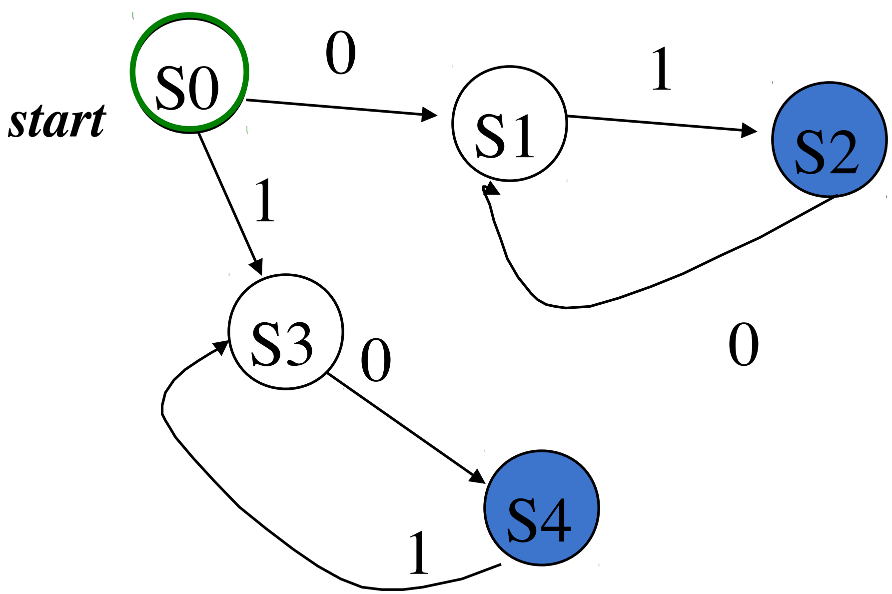
which of these strings is recognized by it:

1, 00, 10, 1000, 01, 011

Tasks for REs and FAs

- **Description of a language**
  - **What language is described by the following RE:**

    $(01)^+ \mid (10)^+$
  - **What language is recognized by the following FA:**



Tasks for REs and FAs

- **Codification of a language**
  - Complex constants are parenthesized pairs of integers Eg (3, 17)
  - Show an RE and an FA for complex constant

(* digit+, digit+ *)