Formal Languages

- **A mechanism to aid description of programming language constructs**
- **A way to describe difficulty of computation problems formulated as language recognition problems**

Theory of Computation

- **Are some things inherently harder to compute than others?**
- **Are some classes of machines inherently more powerful that others?**
- **Are there some things that cannot be computed?**

Formal Languages

- **FL is a set of strings (called *sentences*) over some finite alphabet of symbols, called *terminals***
  - **Not necessarily a finite set of strings**
- **A FL can be defined by a grammar**
  - **A set of rules LHS ::= RHS where LHS and RHS are strings of symbols and non-terminals**
  - **The set of all strings that can be generated from the Start symbol by repeatedly replacing a rule's left-hand side with it's right hand side**

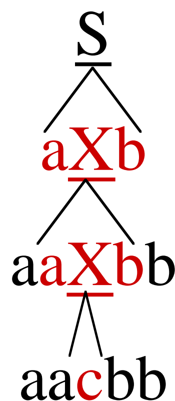
Example

- **Alphabet: {a, b}**
- **Grammar**

  S::= acb | aXb

  X:= c | aXb

- **A Derivation**



- **A language**
  - **{acb, aacbb, aaacbbb, …} i.e. a^n c b^n**

Classes of Languages

- **We can define a whole class of languages by giving a set of restrictions on the grammar**
- **E.g, Regular languages are those that can be described by a grammar where every rule is of the form**

<non-terminal> ::= <terminal> <non-terminal>

or <non-terminal> ::= <terminal>

- **E.g., S ::= aA ab, aab, aaab, ...**

A::= aA | aB

B::= b

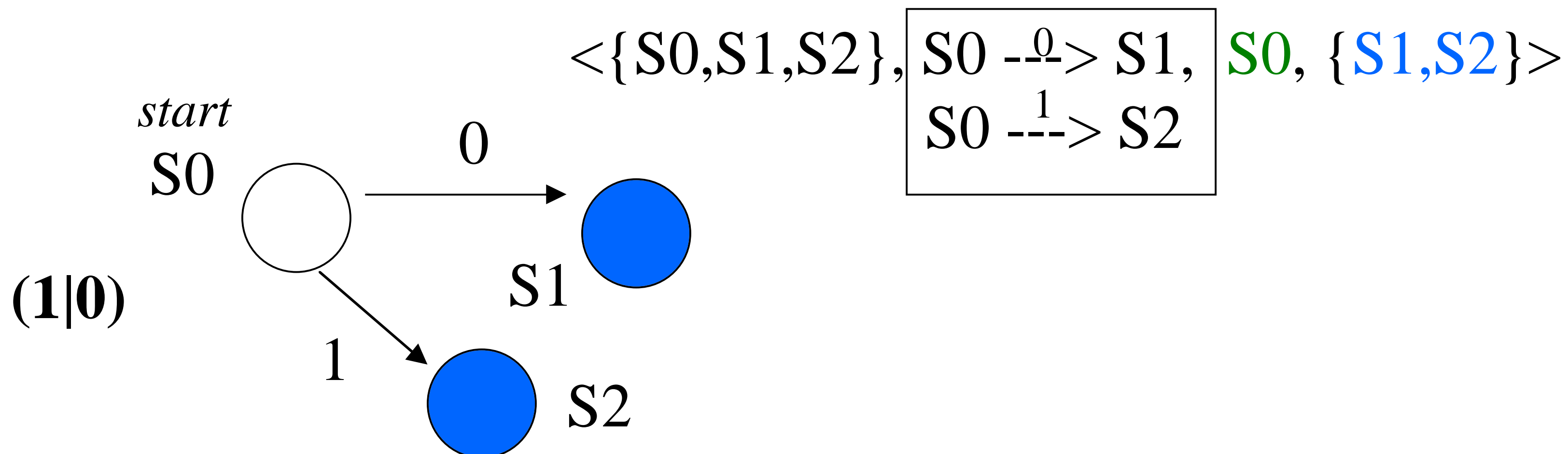
Automata

- **A language can also be defined by giving a machine, or *automaton* that looks at a string and decides if it is in the language**
- **There ate classes of automata that correspond to classes of grammars**
- **E.g., the class of automata corresponding to regular grammars is the class *Finite State* automata**

Finite State Automaton (FSA)

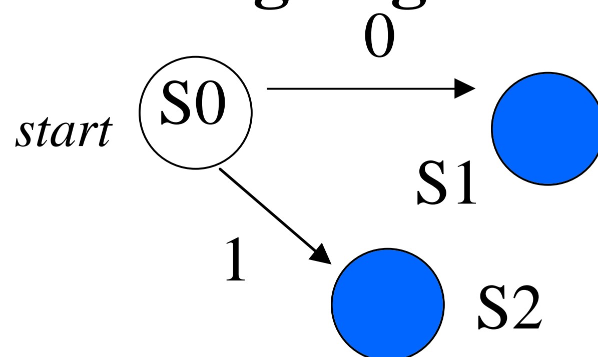
- **Recognizer of the language generated by a regular expression**
- **Described by**

<set of states, labelled transitions, start state, final state(s)>



FSA

- **FSA *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**
  - **Empty transitions signify illegal moves; can think of FSA going to a sink error state**

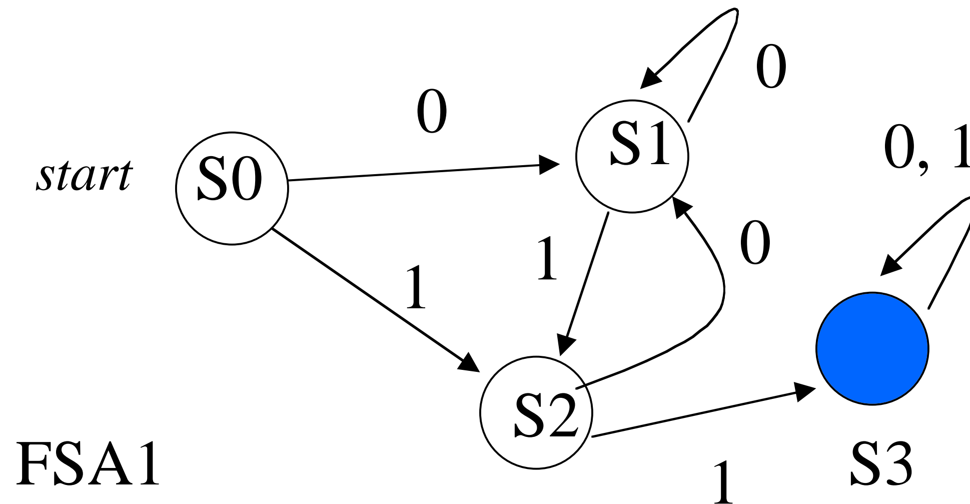


states:	inputs: 0	1
S0	S1	S2
S1	---	---
S2	---	---

transition table

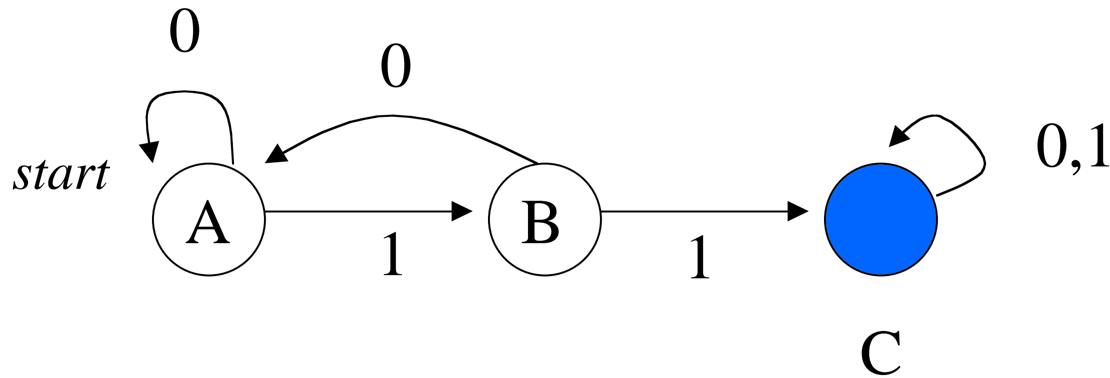
Examples

Binary numbers containing a pair of adjacent 1's: (0 | 1) * 1 1 (0 | 1) *



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

An Equivalent FSA

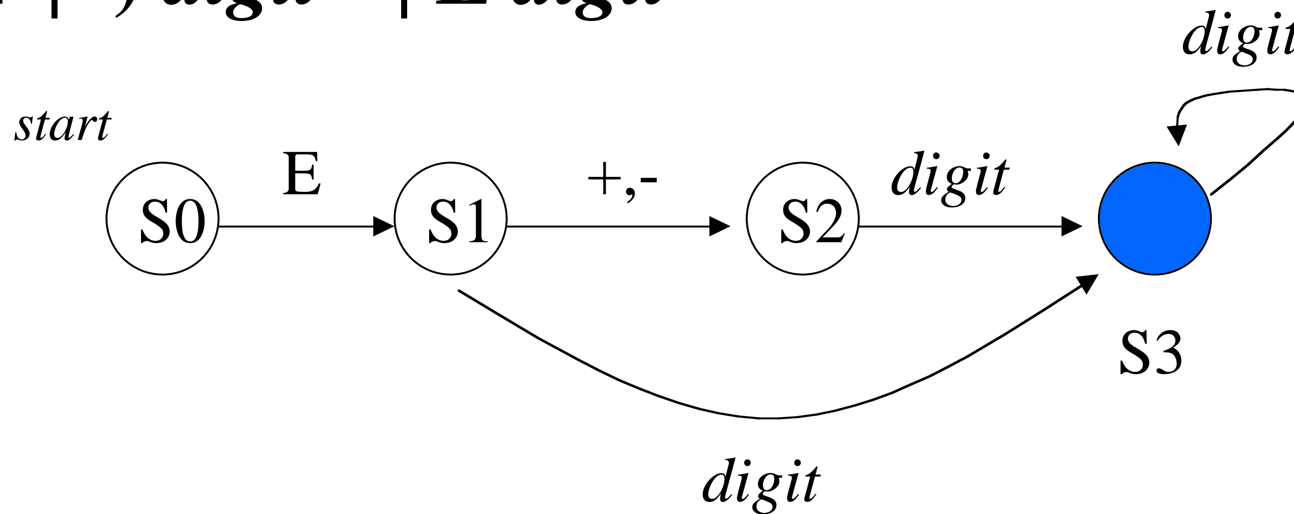


FSA1 and FSA2 recognize the same set of strings of terminals, the same language! Therefore FSAs are NOT UNIQUE.

Example

Exponent in scientific notation:

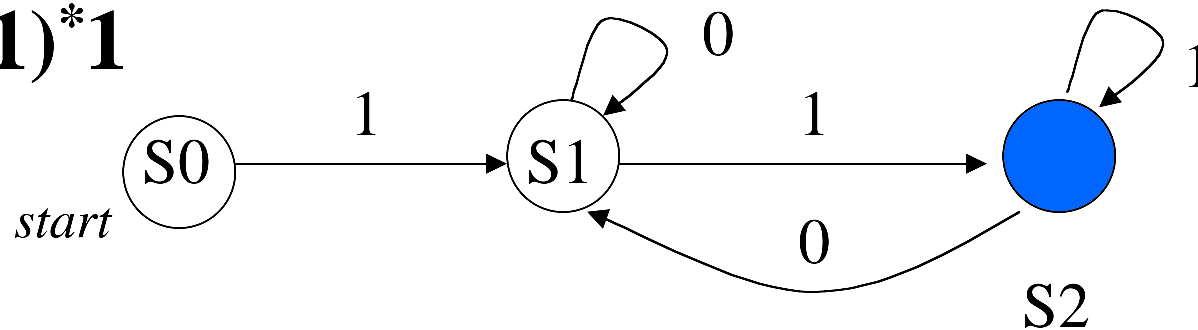
E (+ | -) *digit* $^{+}$ | E *digit* $^{+}$



Example

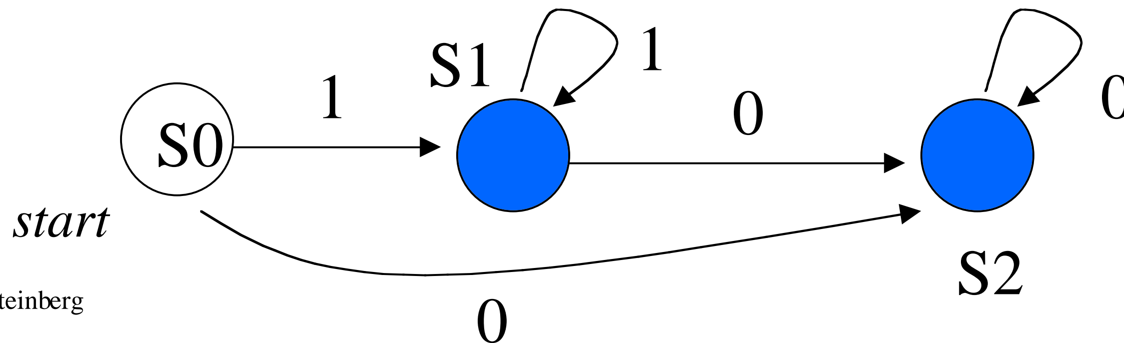
Binary numbers which begin and end with a 1,

$1(0|1)^*1$



All binary numbers containing at least one digit, where all their 1's precede all their 0's

$0^+ | 1^+0^*$



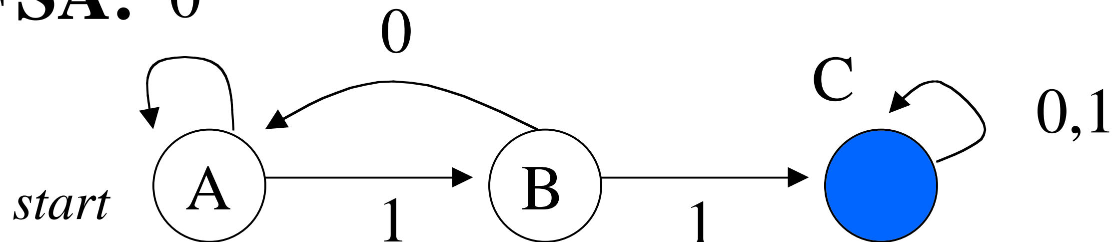
Nondeterministic FSAs

- **Allow more than one transition with same label**
- **Allow ε transition**

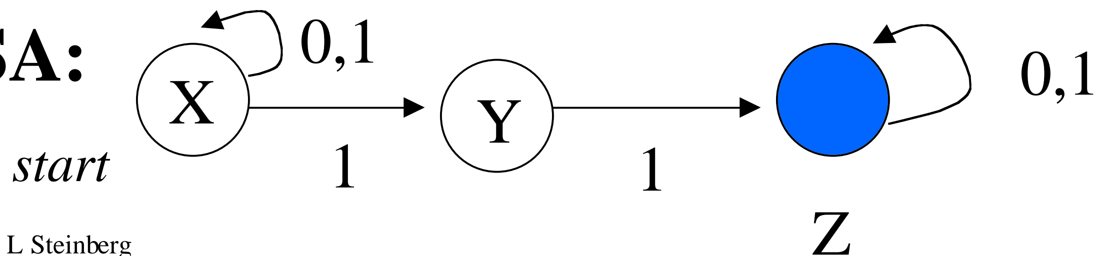
---

e.g., $\mathbf{(0 \mid 1)^{*}\ 1\ 1\ (0 \mid 1)^{*}}$

DFSA:



NFSA:



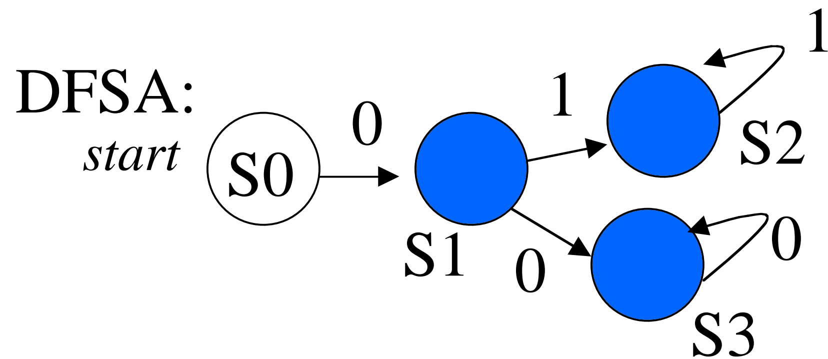
NFSAs

- **Recognize a sentence in a language by progressing from initial state to a final state**
  - **Think of following many threads of computation at same time; one must lead to a final state for a recognition to occur**
- **class of languages recognizable by NFSAs is *SAME* as class of languages recognizable by DFSAs.**
- **There are algorithms to build NFSA directly from RE**

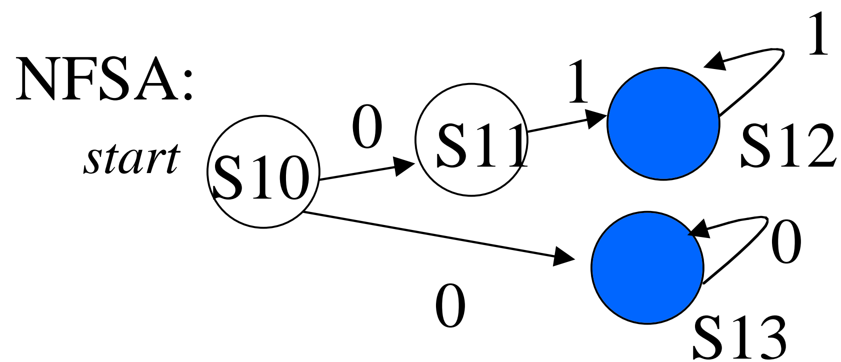
Example

$0^+ \mid 0\,1^+$ all binary numbers in with at least one 0, in which all 0's proceed all 1's

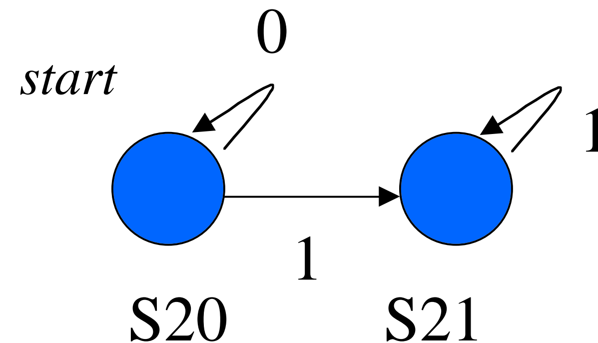
DFSA:



NFSA:



Why doesn't this NFSA work?



Chomsky Hierarchy

- Describes categories of languages which correspond to more and more powerful recognizing automata
- 4 level hierarchy
  - We've studied bottom two levels: regular and context-free languages

Type 3 (regular) Languages

- *Recognizer*: finite state automaton
- Can do simple recursive constructs
- Can't count (or match parentheses)
  - Not regular $\{a^n b^n , n>=1\}$
- Can be written with all right recursive or all left recursive rules
  - Nonterm ::= term | Nonterm term

Type 2 (context-free) Languages

- *Recognizer*: push down automaton
- BNF rules with 1 nonterminal on lefthandside
- Can't check *context*
  - Not context-free $\{a^n b^n c^n, n>=1\}$
  - Programming examples
    - Check that no variable is declared twice
    - Check difference between function calls and array accesses in Fortran (both use parentheses)

    DIMENSION …. F(10,10)….F(I,J)….

Type 1 (context-sensitive) Language

- *Recognizer*: linear bounded automaton
- Grammar rules can have more than 1 symbol on lefthandside as long as |rhs| >= |lhs|
- Can do parameter - argument matching (in number)
- Examples:

  $\{a^n b^m c^n d^m, m,n >= 1\}$

  $\{a^n b^n c^n, n >= 1\}$

Context-sensitive Example

1. T ::=	S	
2. S ::=	a S B C (2a) | a B C (2b)	
3. CB ::=	BC	--reverse B's and C's
4. aB ::=	ab	
5. bB ::=	bb	--expand B
6. bC ::=	bc	--expand C
7. cC ::=	cc	

Derive aabbcc:

T $\xrightarrow{1}$ S $\xrightarrow{2a}$ a S B C $\xrightarrow{2b}$ aaB C B C $\xrightarrow{3}$ aa B B C C $\xrightarrow{4}$ aabB C C $\xrightarrow{5}$ aabbCC $\xrightarrow{6}$ aabbcC $\xrightarrow{7}$ aabbcc

Type 0 (recursively-enumerable) Languages

- *Recognizer*: Turing machine
- All languages that can be recognized by a procedure
- Subclass of Type 0: Recursive languages, languages recognized by an algorithm that always halts

Turing Machines, Lightly

- **Abstract model of computation**
- **<finite set of states, alphabet, blank symbol, start state, final state, transition function>**
  - **transition function:**

  <state, tape symbol read> → <state, tape symbol wrote, {L,R,S}> where L,R,S means tape moves 1 square to the Left, Right, No move
- **TM Halting problem: Given a TM in an arbitrary configuration with nonblank symbols on its tape, will the TM eventually halt? -- unsolvable!**
  - **There cannot exist an algorithm to solve this problem for an arbitrary choice of Turing machine on arbitrary input, although for a specific TM with specific input, there may be a solution.**