

CS311—Computational Structures

Properties of Regular Languages

Lecture 5

What we know so far:

- RLs are closed under product, union and *
- Every RL can be written as a RE, and every RE represents a RL
- Every RL can be recognized by a NFA
 - and we know how to build it
- NFAs and DFA have the same “power”
- Every NFA can be turned in to a DFA
 - “the subset construction”
 - today’s homework: implement it, working in pairs

What's Next?

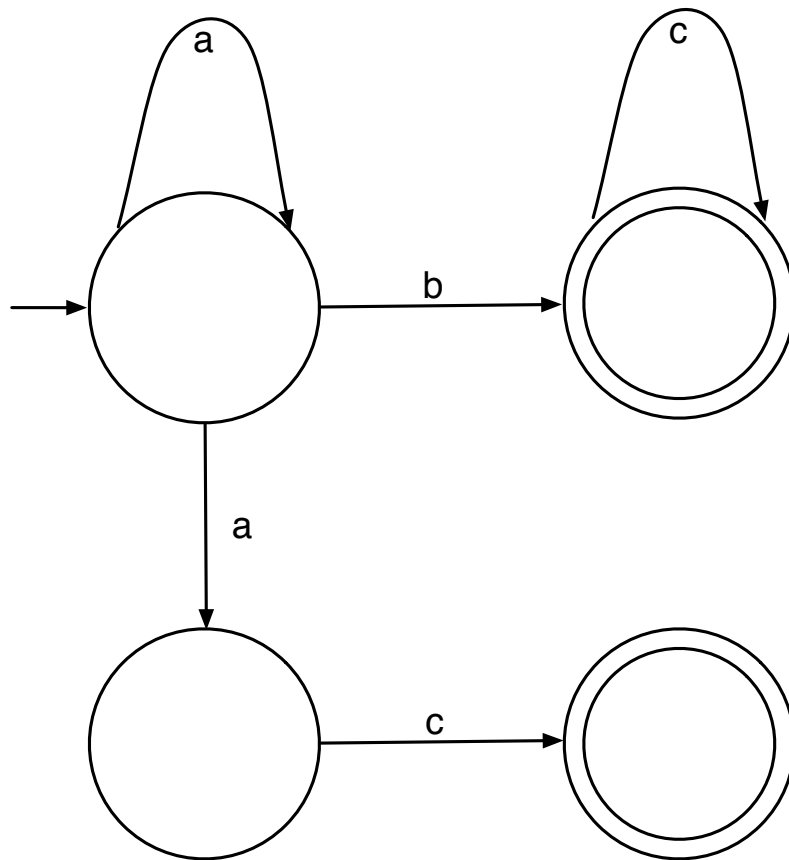
- How to turn a FA into a regular grammar
 - and vice-versa
- How to figure out whether a language is regular or not
 - Myhill-Nerode Theorem
 - minimum-state DFAs
 - Pumping Lemma

From FSM to Regular Grammar

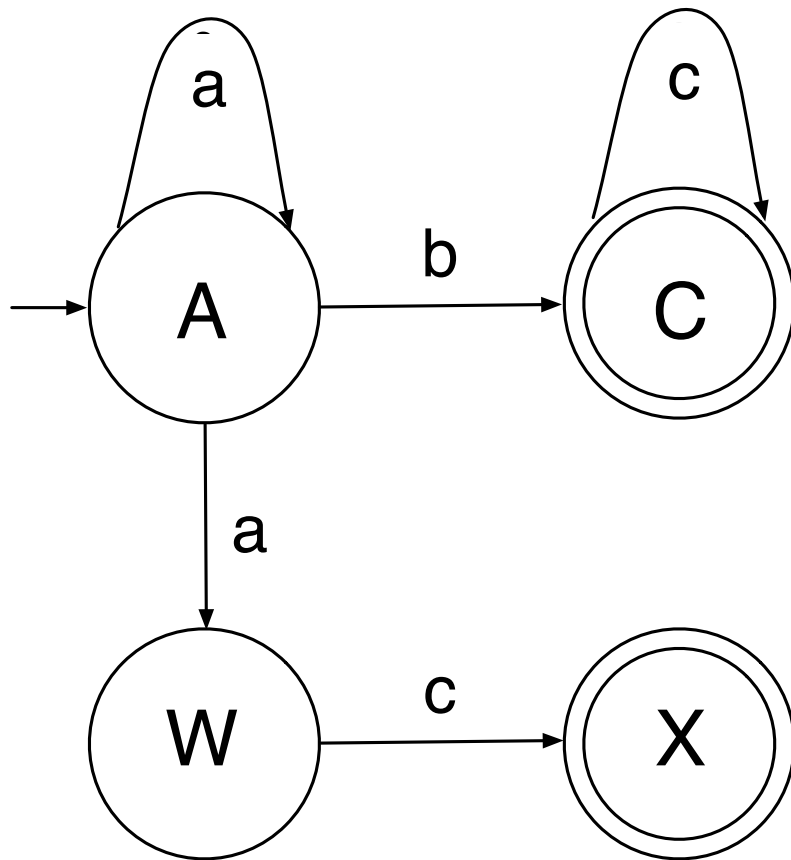
Hein Algorithm 11.11

1. Rename the states Q to a set of upper-case letters
2. The start symbol of the grammar is the name of the start state q_0 .
3. For each transition $(I) \xrightarrow{a} (J)$, create the production $I \rightarrow aJ$.
4. For each transition $(I) \xrightarrow{\Lambda} (J)$, create the production $I \rightarrow J$.
5. For each final state K , create the production $K \rightarrow \Lambda$.

Example



Example



$A \rightarrow aA$

$A \rightarrow bC$

$A \rightarrow aW$

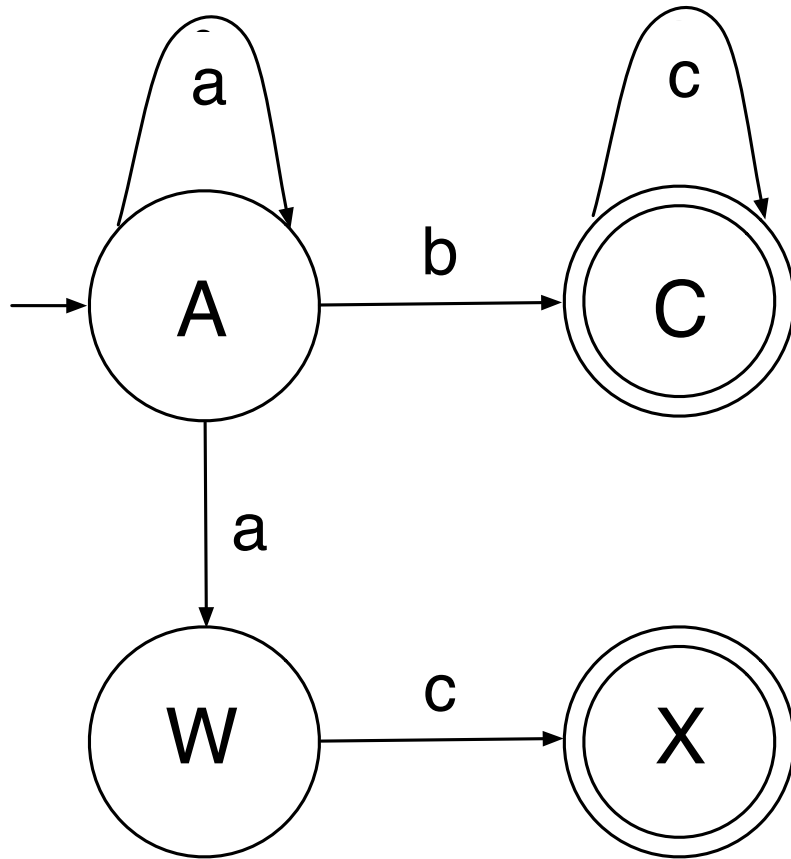
$C \rightarrow cC$

$C \rightarrow \Lambda$

$W \rightarrow cX$

$X \rightarrow \Lambda$

Example



$A \rightarrow aA \mid bC \mid aW$

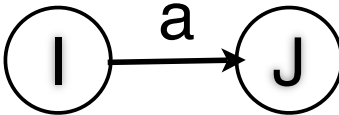
$C \rightarrow cC \mid \Lambda$

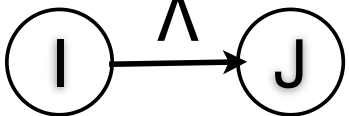
$W \rightarrow c$

From Regular Grammar to FSM

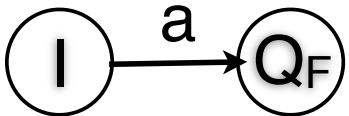
Hein Algorithm 11.12

1. Transform the grammar so that all productions are of the form $A \rightarrow x$ or $A \rightarrow xB$, where x is either a single letter or Λ .
2. The start state of the NFA is the grammar's start symbol.
3. Create state Q_F and add it to the set F of final states.

4. For each production $I \rightarrow aJ$, create the transition 

5. For each production $I \rightarrow J$, create the transition 

6. For each production $K \rightarrow \Lambda$, add K to the set of final states F

7. For each production $I \rightarrow a$, create the transition 

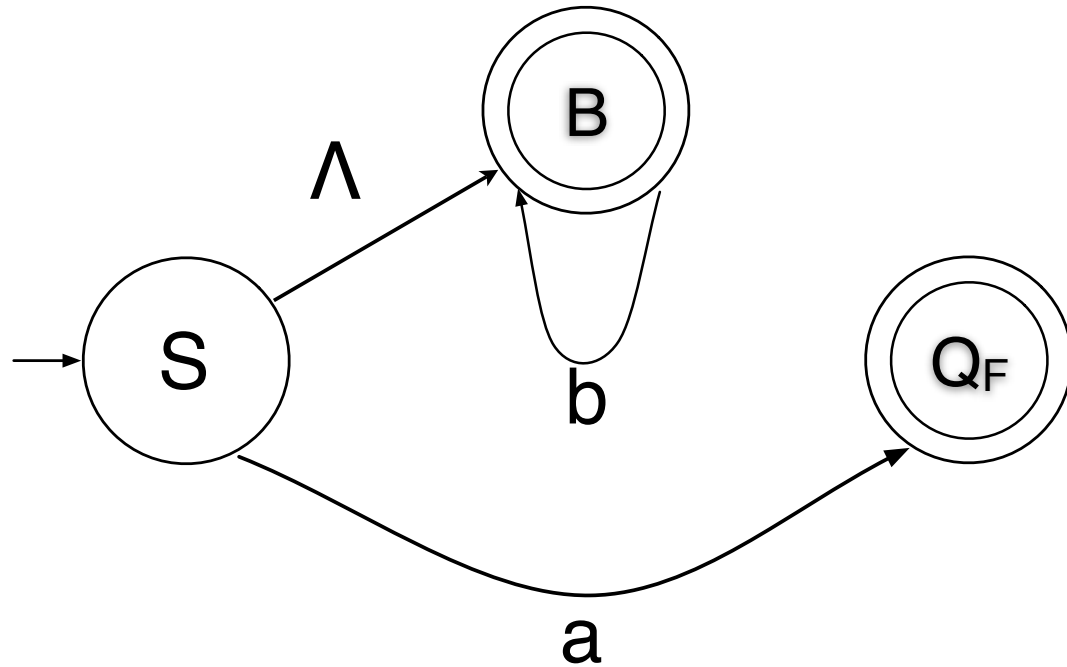
Example

$S \rightarrow a$

$S \rightarrow B$

$B \rightarrow \Lambda$

$B \rightarrow bB$



What's the language?

$a + b^*$

Language Indistinguishability

- Consider a language L over an alphabet A .
- Two strings $x, y \in A^*$ are L -indistinguishable if for all $z \in A^*$, $xz \in L$ whenever $yz \in L$. We write $x \equiv_L y$
- \equiv_L is an equivalence relation
- The *index* of L is the number of equivalence classes induced by \equiv_L

Example: $L = a + b^*$

- $a + b^* = \{ \Lambda, a, b, bb, bbb, bbbb, \dots \}$

$a \equiv_L b$?

$\Lambda \equiv_L b$?

$aa \equiv_L ab$?

$ab \equiv_L bb$?

- What are the equivalence classes of \equiv_L ?

1. $\{a\}$

2. $\{b, bb, bbb, bbbb, \dots\}$

3. $\{\Lambda\}$

4. everything else

Myhill-Nerode Theorem

- The equivalence relation \equiv_L characterizes exactly what the state of an automaton that accepts L needs to remember about the read portion of the input:
 - if the read portion of the input is x , then the state needs to remember the equivalence class $[x]$.
 - This is sufficient, because if $x \equiv_L y$, then it does not matter if the read portion of the input was x or y ; all that matters (for deciding whether to accept or reject) is the future portion of the input, say z , because $xz \in L$ iff $yz \in L$.
 - It is also necessary, because if $x \not\equiv_L y$, then there is some possible future portion z of the input such that xz needs to be accepted and yz rejected (or vice versa).

Theorem Statement (Part A)

If the index of a language A is k , then there is a k -state DFA M_A such that $L(M_A) = A$

Proof

Theorem Statement (Part B)

- For every k -state DFA M , the index of $L(M)$ is at most k .

Proof

- Consider a DFA M over A^* with k states
- Let $x, y \in A^*$, and define $x \equiv_M y$ iff M is in the same state after reading x and y .
- index of \equiv_M is k
- if $x \equiv_M y$, then $x \equiv_{L(M)} y$, because xz and yz must also lead to the same state
- Hence: there can be no more $\equiv_{L(M)}$ equivalence classes than there are \equiv_M equivalence classes,
 - i.e., the index of $L(M) \leq k$

Constructing a Minimal DFA

Hein Construction 11.10

Algorithm to Construct a Minimum-State DFA (11.10)

Given: A DFA with set of states S and transition table T . Assume that all states that cannot be reached from the start state have already been thrown away.

Output: A minimum-state DFA recognizing the same regular language as the input DFA.

1. Construct the equivalent pairs of states by calculating the descending sequence of sets of pairs $E_0 \supset E_1 \supset \dots$ defined as follows:

$$E_0 = \{\{s, t\} \mid s \text{ and } t \text{ are distinct and either both states are final or both states are nonfinal}\}.$$
$$E_{i+1} = \{\{s, t\} \mid \{s, t\} \in E_i \text{ and for every } a \in A \text{ either } T(s, a) = T(t, a) \text{ or } \{T(s, a), T(t, a)\} \in E_i\}.$$

The computation stops when $E_k = E_{k+1}$ for some index k . E_k is the desired set of equivalent pairs.

2. Use the equivalence relation generated by the pairs in E_k to partition S into a set of equivalence classes. These equivalence classes are the states of the new DFA.
3. The *start state* is the equivalence class containing the start state of the input DFA.
4. A *final state* is any equivalence class containing a final state of the input DFA.
5. The transition table T_{\min} for the minimum-state DFA is defined as follows, where $[s]$ denotes the equivalence class containing s and a is any letter: $T_{\min}([s], a) = [T(s, a)]$.