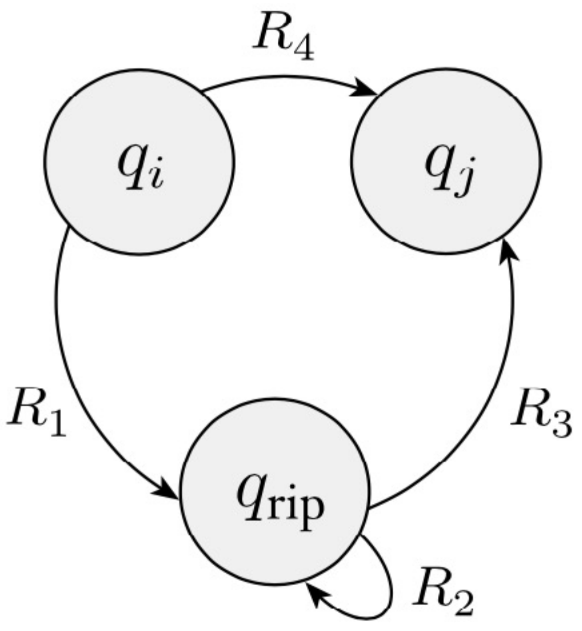
Nonregular Languages

Schedule ahead

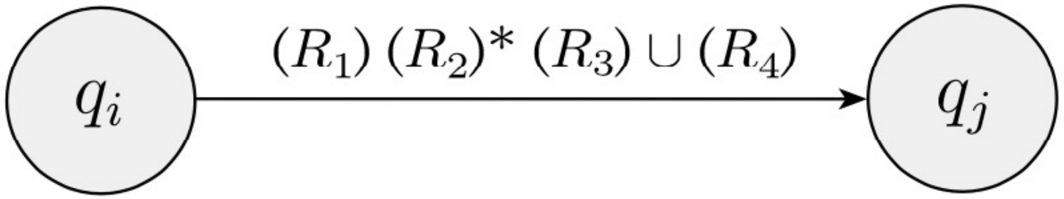
- Wed Feb 9 (today): nonregular languages, part I
- Fri Feb 11:
  - Nonregular languages, part II
  - HW3 in, HW4 out
- Mon Feb 14: In-class exercises on PL + discussions on HWs
- Wed Feb 16: introduce CFG
  - (If needed, more discussions on HWs)
- Fri Feb 18: Review for Mid-term 1
  - HW4 in
- Mon Feb 21: HOLIDAY (no class)
- Wed Feb 23: Mid-term 1

Example 2 (“left over” from last lecture)

"Rip and Repair" – constructing an equivalent GNFA with one fewer state when k > 2

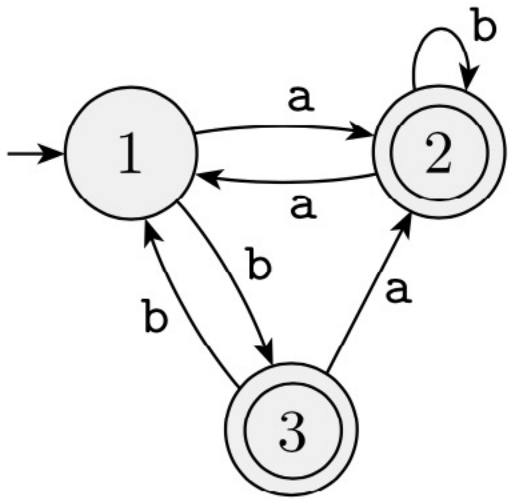


before

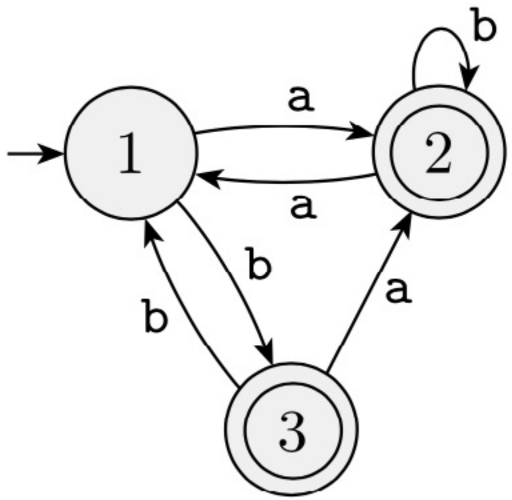


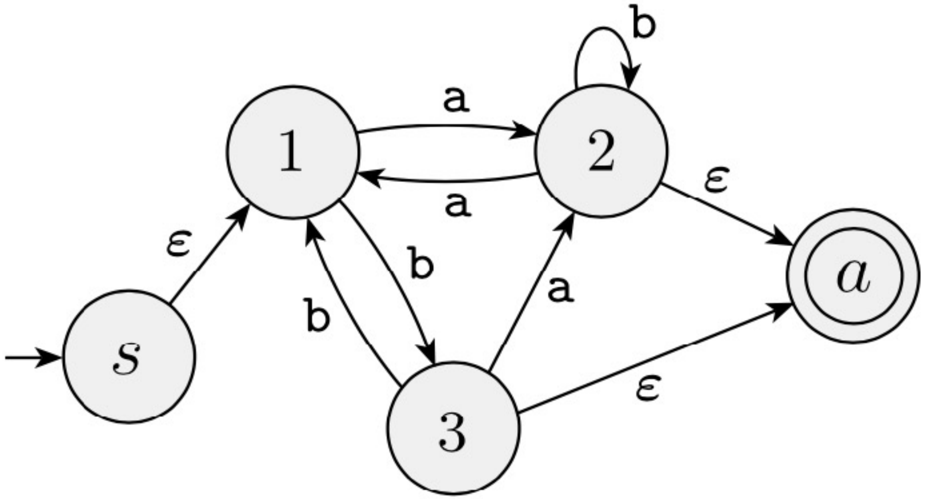
after

Example 2

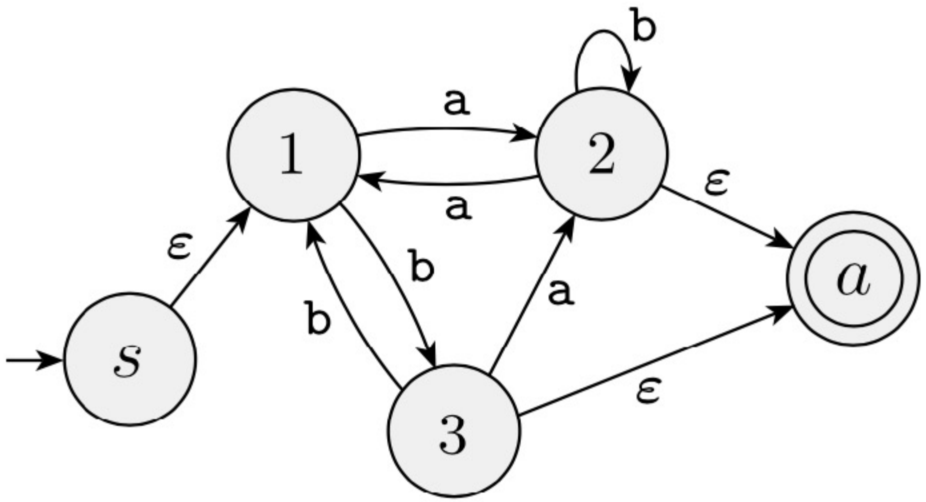


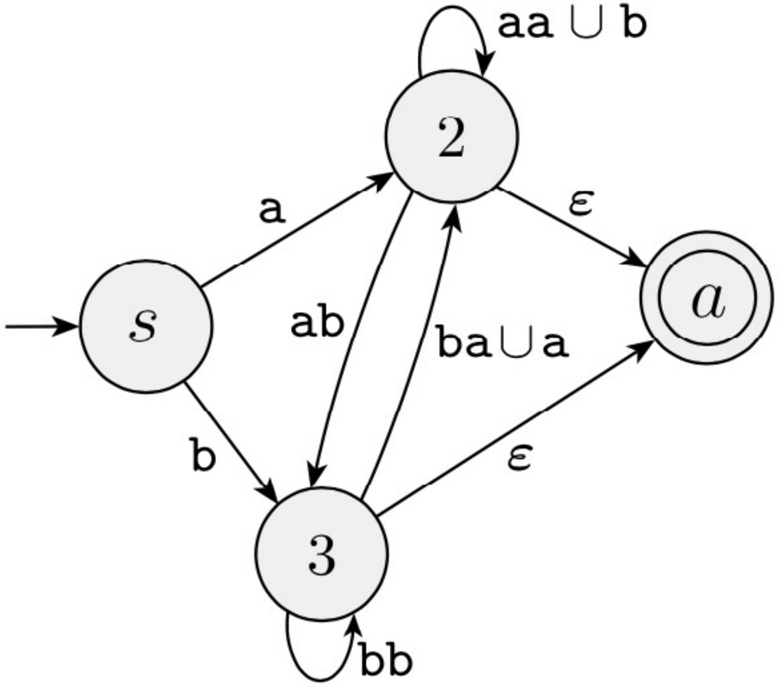
Example 2



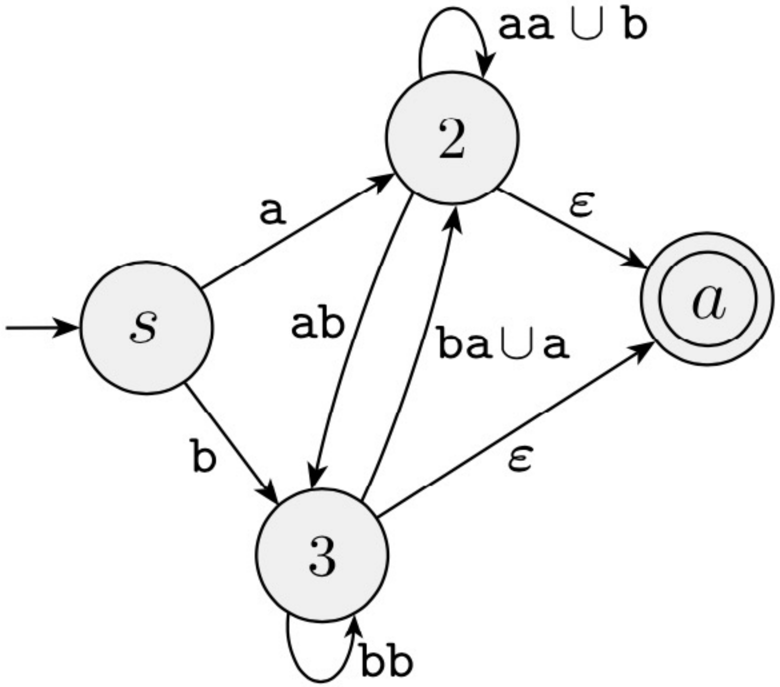


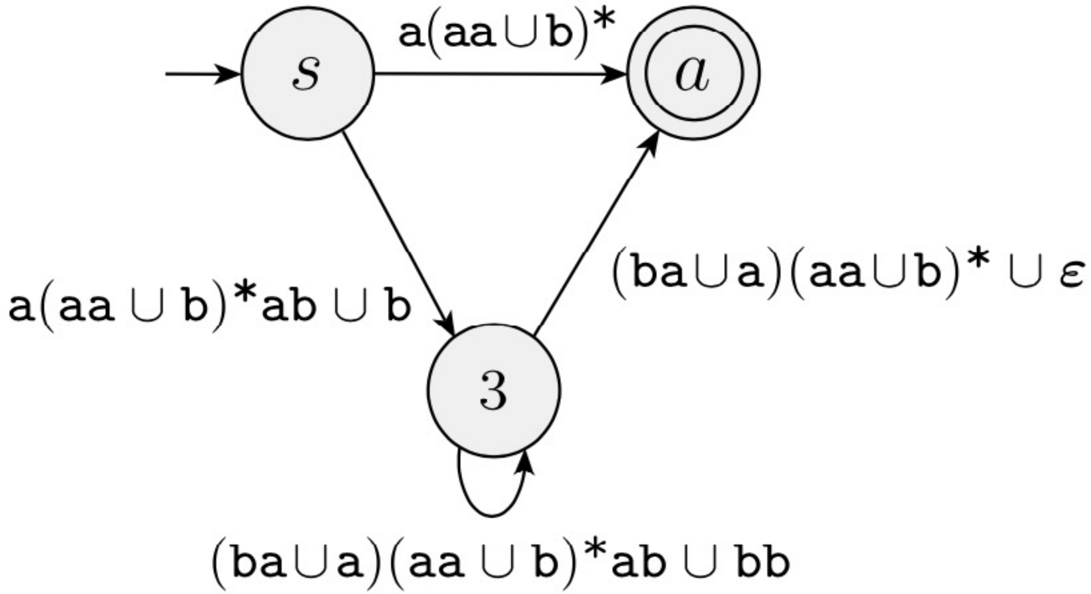
Example 2



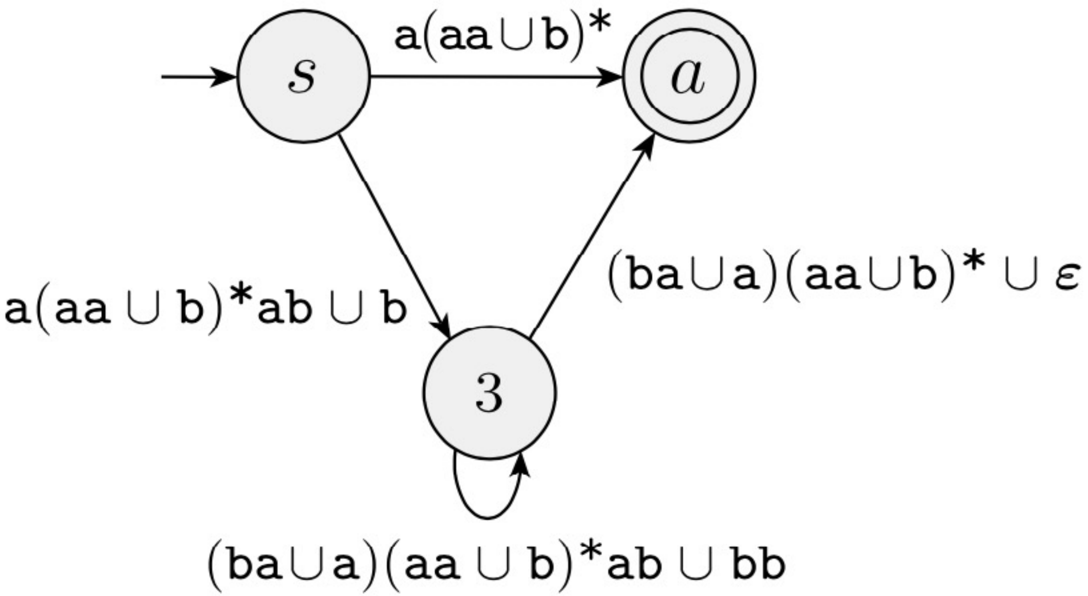


Example 2

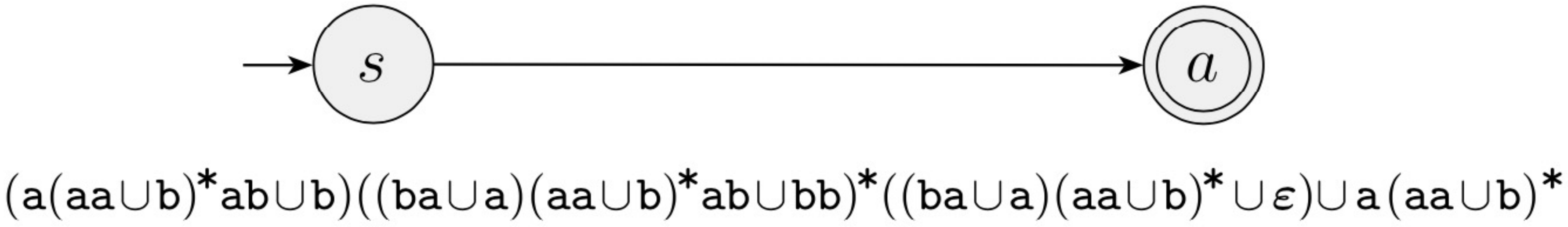




Example 2



s
a
3
$a(aa\cup b)^*$
$a(aa\cup b)^*ab\cup b$
$(ba\cup a)(aa\cup b)^*\cup\varepsilon$
$(ba\cup a)(aa\cup b)^*ab\cup bb$



s
a
$(a(aa\cup b)^*ab\cup b)((ba\cup a)(aa\cup b)^*ab\cup bb)^*((ba\cup a)(aa\cup b)^*\cup\varepsilon)\cup a(aa\cup b)^*$

Nonregular languages

- Consider the language $B = \{0^n 1^n \mid n \geq 0\}$
- If we were to construct a DFA that recognizes B, the machine seems to need to remember how many 0s have been seen so far as it reads the input.
- Because the number of 0s isn't limited, the machine will have to keep track of an unlimited number of possibilities.
- But it cannot do so with any finite number of states.

Nonregular languages

- Just because a language appears to require unbounded memory doesn’t mean that it is necessarily nonregular.
- It happens to be true for language B; but other languages seem to require an unlimited number of possibilities and yet are regular.
- Example: consider two languages over the alphabet $\Sigma = \{0,1\}$
  - C = {w | w has an equal number of 0s and 1s}, and
  - D = {w | w has an equal number of occurrences of 01 and 10 as substrings}

- **C is not regular, but D is regular.**

The Pumping Lemma

Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Note

- When s is divided into xyz, either x or z maybe ε, but Condition 2 says that $y \neq \varepsilon$
- Condition 3 states that the pieces x and y together have length at most p

Proof IDEA

- Let $M = (Q, \Sigma, \delta, q_1, F)$ be a DFA that recognizes A
- We assign the pumping length p to be the number of states of M
- We want to show that any string s in A of length at most p maybe broken into three pieces xyz satisfying our three conditions

Case 1: no strings in A are of length $\geq p$. Then, the Lemma is vacuously true: the three conditions hold for all strings of length $\geq p$ if there aren't such strings

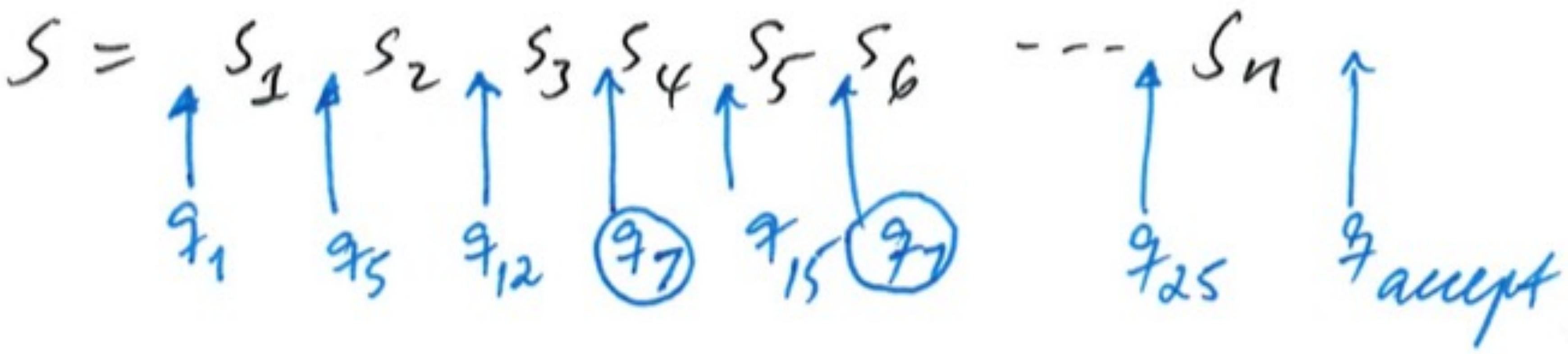
Case 2: if s in A has $|s| \geq p$: Consider the sequence of states that M goes through when computing with input s.

Proof IDEA

- It starts with the start state q_1, then it goes to some state, say q_5, then say q_{10}, …., until it reaches q_{accept}
- If we let $n = |s|$, then the sequence $q_1, q_5, \ldots q_{accept}$ has length $n + 1$
- Because n is at least p, we know that $n+1 > p \equiv$ number of states of M
- Therefore, the sequence must contain a repeated state

- This result is an example of the pigeonhole principle – if p pigeons are placed into fewer than p holes, some hole has to have more than one pigeon in it.

Illustration (q_7 is the one that repeats)

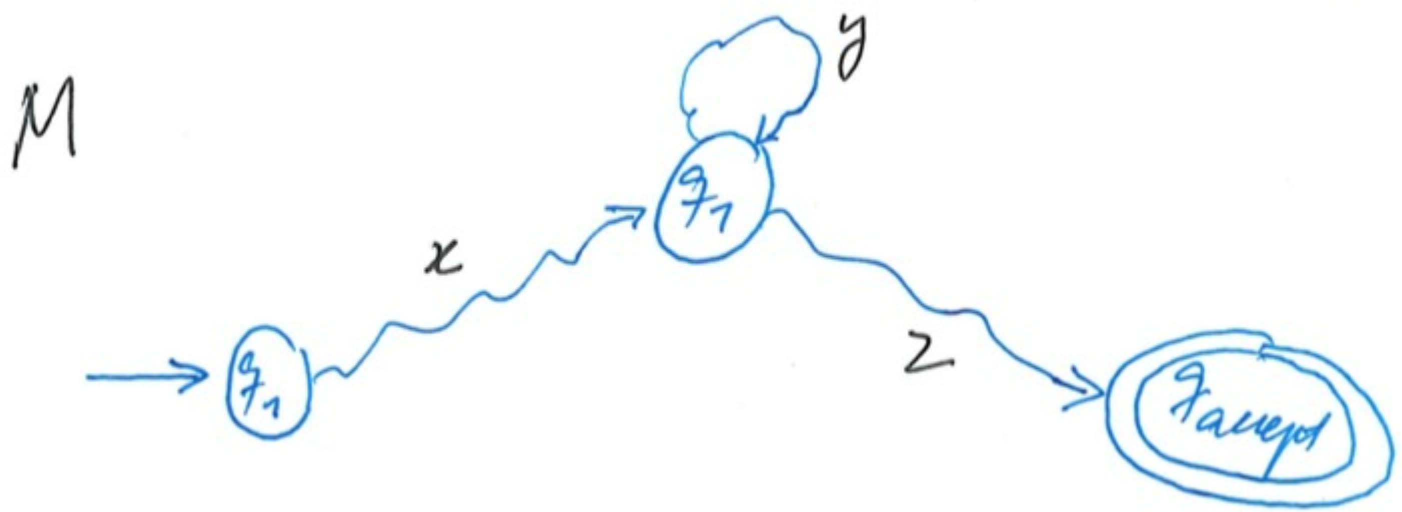


Dividing s into the three pieces x, y, and z

- x: part of s appearing before q_7
- y: part between the two appearances of q_7
- z: remaining part of s, after second occurrence of q_7

In other words

- x: takes M from q_1 to q_7
- y: takes M from q_7 back to q_7
- z: takes M from q_7 to q_{accept}



Let us see how this division of s satisfies the three conditions

- Suppose we run M on input $xyyz$
  - x takes M from q_1 to q_7, then the first y takes it from q_7 back to q_7, as does the second, and then z takes it to q_{accept}
    - OK
- Similarly, it will accept xy^iz for every $i > 0$
- For the case $i = 0$, $xy^iz = xz$, which is accepted for similar reason
- This establishes Condition 1.

How about Conditions 2 and 3?

- **Condition 2**: we see that $|y|>0$, as it was part of s that occurred between the two occurrences of q_7

- **Condition 3**: we ensure that q_7 is the first repetition in the sequence. Then by the pigeonhole principle, the first $p+1$ states in the sequence must contain a repetition. Therefore $|xy| \leq p$.