CS3102 Theory of Computation

Non-regular Languages

- A language is regular if:
  - There is a NFA/DFA which accepts exactly the strings in the language
  - There is a regular expression which describes exactly the strings in the language
- A lanugage is non-regular if:
  - There is no NFA/DFA which accepts exactly the strings in the language (there will always be false positives/negatives)
  - You cannot write a regular expression which describes exactly the strings in the lanuage (it always misses some or describes too many)
- Examples:

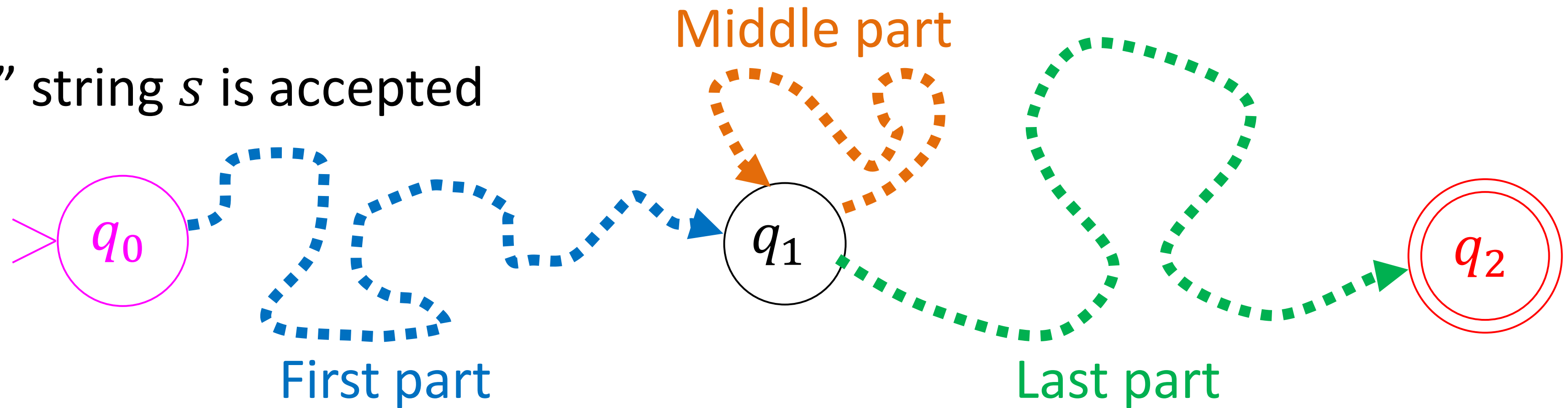
  Non-existence Proof!

  - $\{a^n b^n \text{ for } n \in \mathbb{N}\}$
  - $\{s \mid s \text{ is a palindrome}\}$

Proving Non-Regularity

- For a language to be regular, there must be a DFA for it
- That same DFA must work for *every* string in the language (no matter how long)
- If the language is infinite, there must be some string in the language larger than the machine's size
- This long string it must visit at least one state twice on its way to the final state (pigeonhole principle)

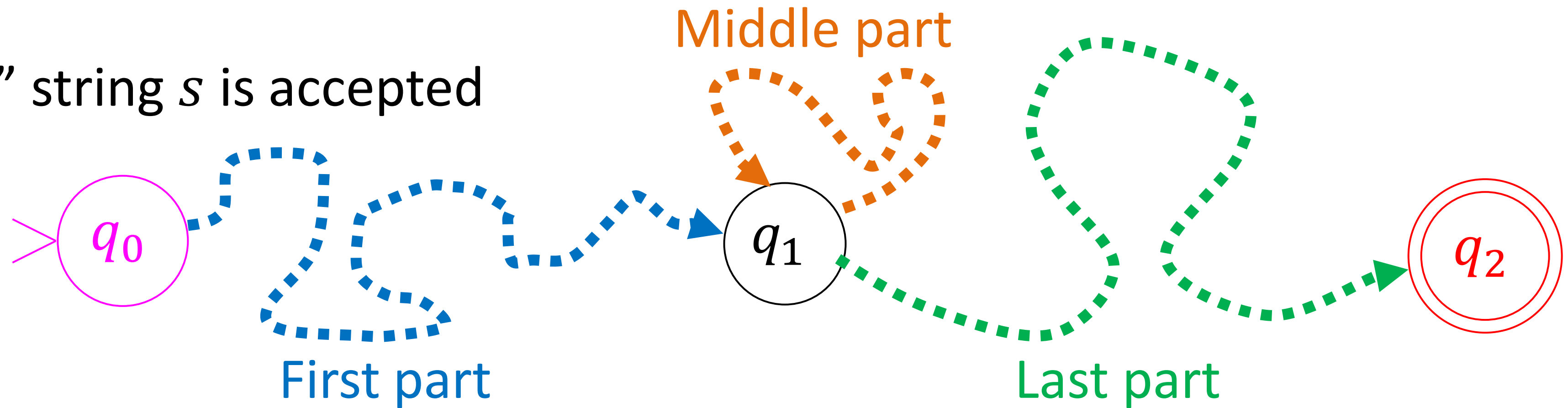
When "long" string s is accepted



"Pumping Lemma" Idea

- Any infinite regular language has a "long" string
- Any "long" string can be broken into 3 parts
  - A first part that takes you from start to **some state**
  - A middle part that takes you back to that **same state**
  - A last part that takes you to a final state
- Copying the middle part (or skipping) still takes us from start to a final state
- If we can't break up a "long" string into these parts which allows us to "pump" the middle, the language isn't regular

When "long" string s is accepted



Pumping Lemma

- If L is a regular language, then there is a number $p \in \mathbb{N}$ such that, for *any* string $s \in L$ where $|s| > p$, we can find strings x, y, z where $s = xyz$ satisfying all of:
  1. For each $i \geq 0$, $xy^iz \in L$
  2. $|y| > 0$
  3. $|xy| \leq p$
- Use contrapositive to show a language is not regular:
  - if you can find a long string where you can't do this, the language is not regular

$L = a^n b^n$ is not regular

- Let the "pumping length" be p
- Consider the string $a^p b^p$, note that $|a^p b^p| > p$, so if L is regular this string can be pumped
- If we had $a^p b^p = xyz$ there are 3 options for what y could be:
  1. $y \in a^+$
     - In this case, xy^iz has too many a's
  2. $y \in b^+$
     - In this case, xy^iz has too many b's
  3. $y \in a^+b^+$
     - In this case, xy^iz has a's and b's out of order
- Since $a^p b^p$ cannot be "pumped", L is not regular

$L = \{w \in \Sigma^* | w = w^R\}$ is not regular

- Let the "pumping length" be p
- Consider the string $a^p b^p a^p$, note that $|a^p b^p a^p| > p$, so if L is regular this string can be pumped
- If we had $a^p b^p a^p = xyz$ there are 3 options for what y could be:
  1. $y \in a^+$
     - In this case, xy^iz has too many a's before/after the b's
  2. $y \in a^+b^+ + \mathrm{b}^+\mathrm{a}^+$
     - In this case, xy^iz is not palindrome
  3. $y \in b^+$
     - In this case, $|xy| \geq p$ has too many b's
- Since $a^p b^p$ cannot be "pumped", L is not regular

$L = a^n b^m$ where $n \neq m$ is not regular

- Idea: Use closure properties!
- Assume toward reaching a contradiction that L is regular
- In this case, $\bar{L}$ is regular too (since complement preserves regularity)
- What strings are in $\bar{L}$?
  - Those that have a b before an a, i.e. $(a+b)^* ba (a+b)^*$
  - Those from $a^* b^*$ where the number of a's matches the number of b's
- Since $\bar{L}$ is regular, so is $\bar{L} \cap a^* b^*$ (since both are regular and intersection preserves regularity)
- $\bar{L} \cap a^* b^* = a^n b^n$ which we know isn't regular!

Pumpable $\neq$ Regular

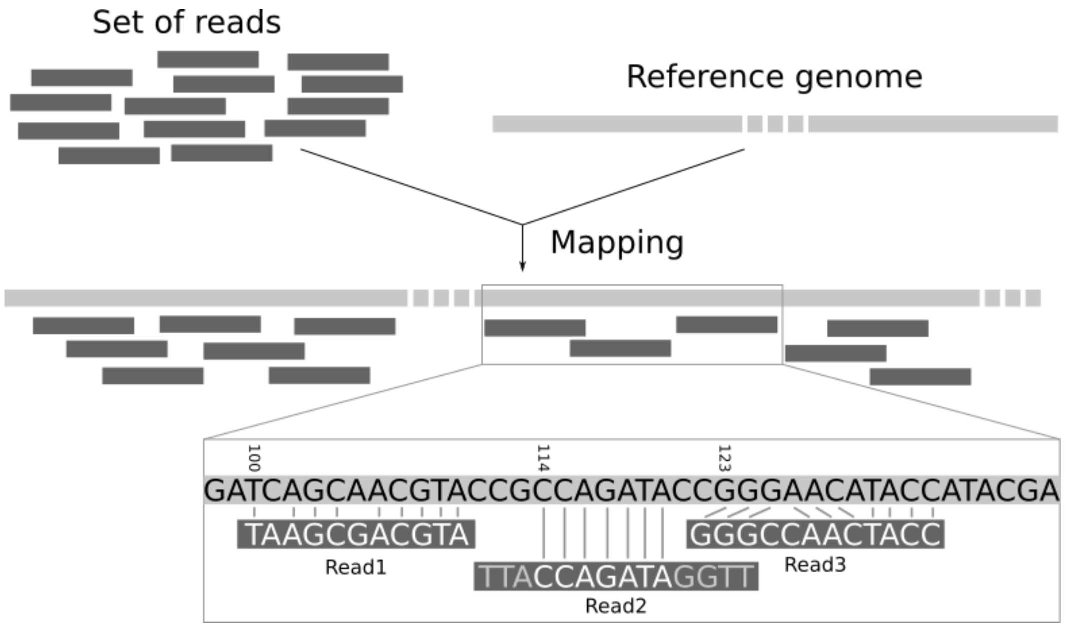
- The pumping lemma can only be used to show NON-regularity
  - If this languages is not pumpable, then it is not regular
- The pumping lamma cannot be used to prove a language is regular
  - Some non-regular languages are pumpable

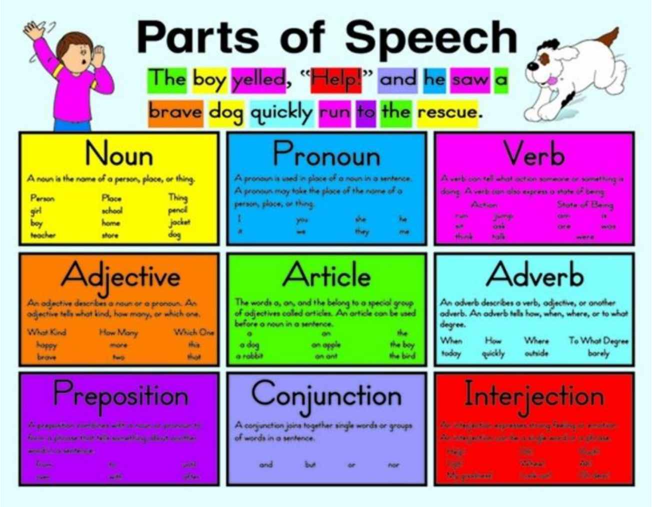
A pumpable non-regular language

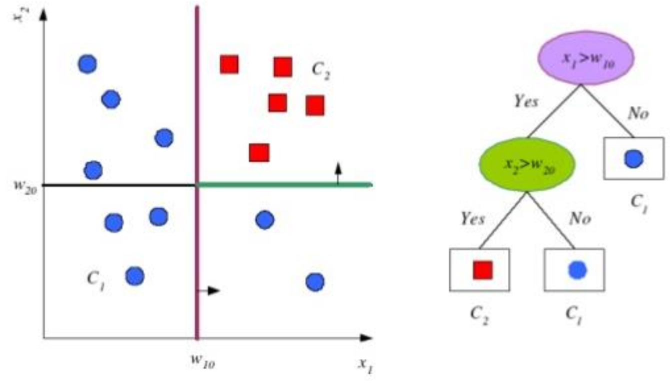
- $L = a^i b^j c^k$ where $i = 1 \Rightarrow j = k$
- $L \cap ab^*c^* = ab^n c^n$ which is not regular (it can't be pumped), so L isn't regular either
- I can pump everything of form ab^*c^* by letting $y = a$
- I can pump everything else by letting y have either just a's or just b's

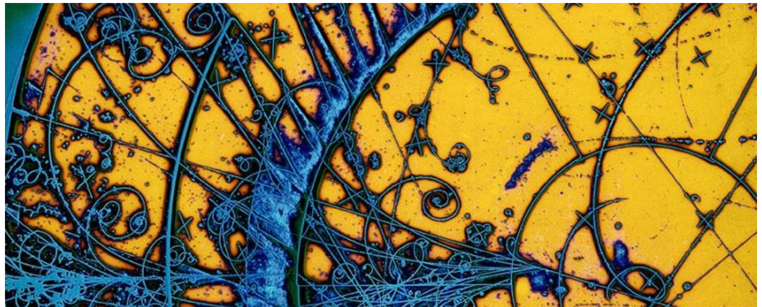
Regexes/NFAs in Applications

- Bioinformatics
  - Read alignment
- Virus scanning
  - ClamAV
- Natural Language processing
  - Part-of-speech tagging
- Machine Learning
  - Decision Tree models
- High Energy Physics
  - Particle collider path tracing









Date
May 22, 2018

Patent Number
9,977,652

Nathan James Brunelle

INVENTOR

SYSTEM, METHOD, AND COMPUTER-READABLE MEDIUM FOR HIGH THROUGHPUT PSEUDO-RANDOM NUMBER GENERATION

The Director of the United States Patent and Trademark Office has received an application for a patent for a new and useful invention. The requirements of law have been complied with, and it has been determined that a patent on the invention shall be granted under the law.

Therefore, this

United States Patent

Grants to the person(s) having title to this patent the right to exclude others from making, using, offering for sale, or selling the invention throughout the United States of America or importing the invention into the United States of America for the term of this patent, subject to the payment of maintenance fees as provided by law.

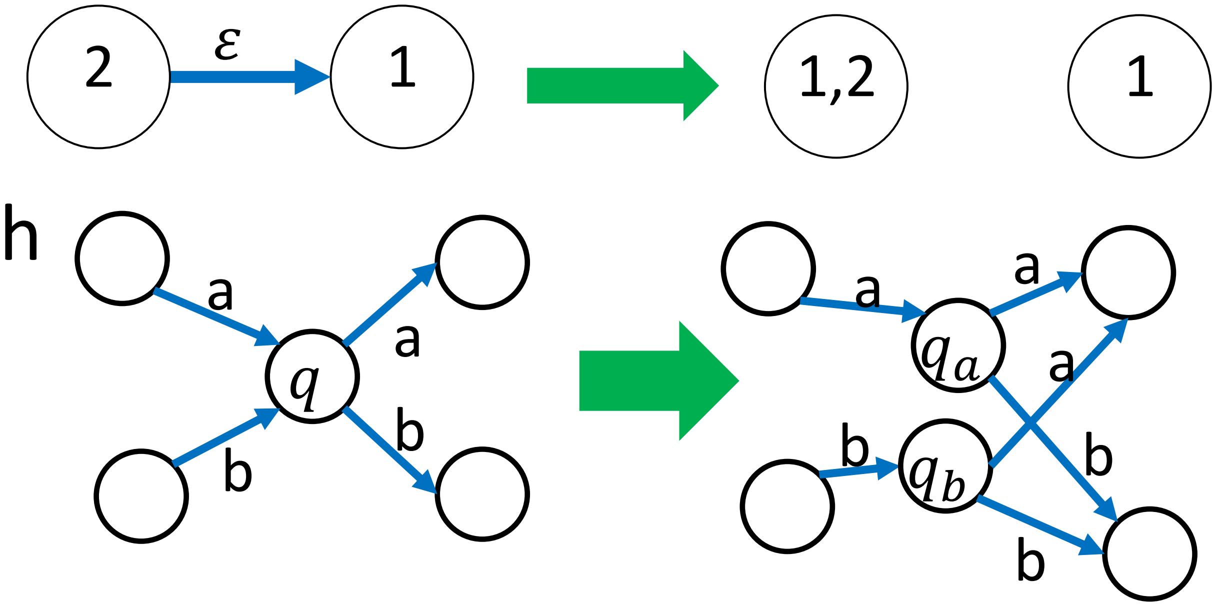
Acting Commissioner of Patents and Trademarks

High-performance NFA simulation

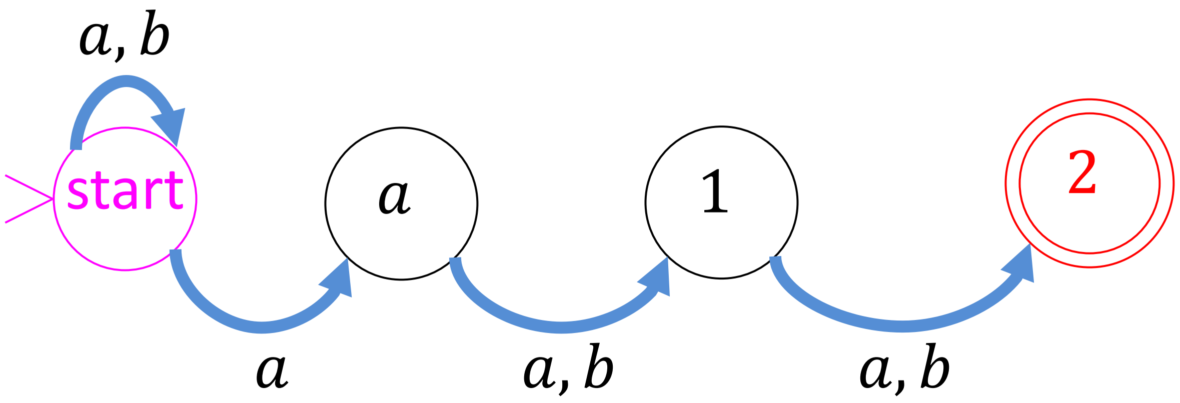
- Informally, how do NFAs transition?
- When is state q active?
- State q becomes active if when reading σ there is some state p that was active before which had a transition to q matching the input character σ
- Procedure: look at all of the active states, look at their outgoing transitions, check which ones match on σ, see if q is among those destinations

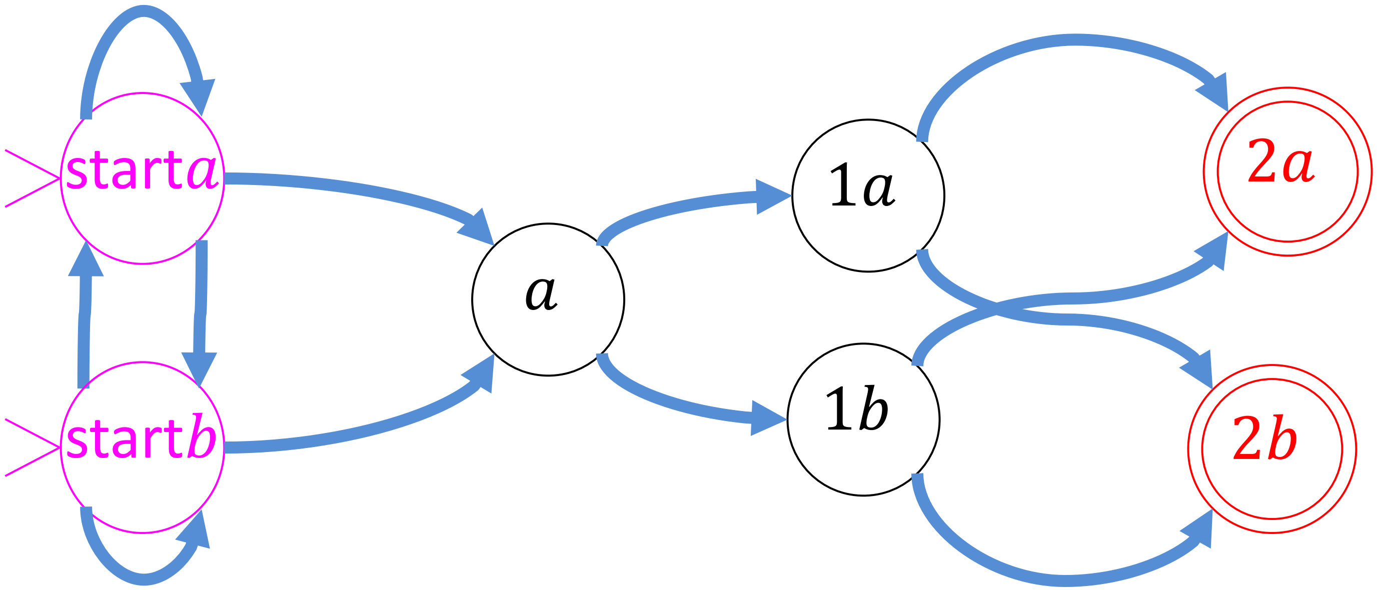
Homogeneous NFAs

- Restriction on NFAs
  - No ε transitions
  - All incoming transitions match on the same character for every state
  - Mutiple starts allowed
- Character matching is a property of the state rather than the transition



Example: a is third from last





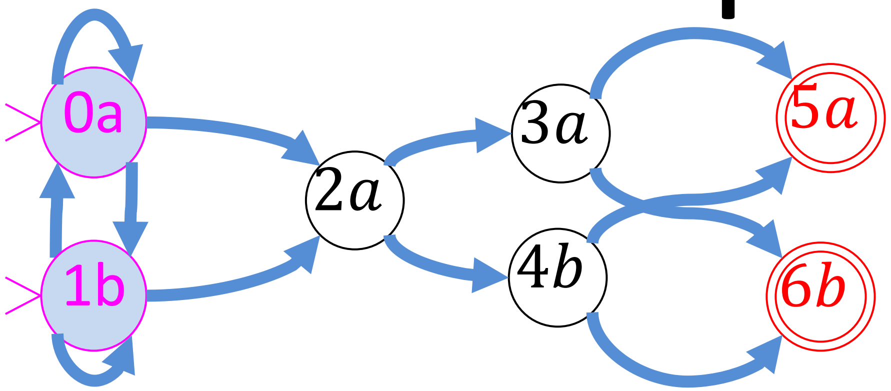
Homgeneous NFA transition

- When is state $q\sigma$ active?
- State $q\sigma$ becomes active if when reading σ there is some state p that was active before which had a transition to $q\sigma$ ~~matching the input character σ~~
- Procedure: look at all of the active states, look at their outgoing transitions, ~~check which ones match on σ~~, see if $q\sigma$ is among those destinations

Bit Parallel Execution

- Use memory lookups and bitwise operations to simulate NFAs
- act: A $|Q|$-bit string representing which states are active
- adj: A $|Q| \times |Q|$ binary matrix representing state adjacency
- $match$: A $|\Sigma| \times |Q|$ binary matrix representing which states match each character

Example: a is third from last



0. Start with which states are active

act:

0	1	2	3	4	5	6
1	1	0	0	0	0	0

1. Look at all rows i s.t. $act[i] = 1$

adj:

	0	1	2	3	4	5	6
0	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
2	0	0	0	1	1	0	0
3	0	0	0	0	0	1	1
4	0	0	0	0	0	1	1
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0

$trans\_to$:

1	**1**	**1**	**0**	**0**	**0**	**0**

2. Do bitwise OR on those rows to get adjacent states

3. Look up states matching current input

$match$:

	0	1	2	3	4	5	6
a	1	0	1	1	0	1	0
b	0	1	0	0	1	0	1

$next$:

0	1	2	3	4	5	6
1	0	1	0	0	0	0

4. Do bitwise AND on $trans\_to$ and $match[a]$

Bit Parallel Algorithm

- To find $next$ on character σ:

1. $trans\_to = \bigvee_{act[i]=1} adj[i]$
   – Find all states with a transition to them
   – All states adjacent to any active state
2. $next = match[\sigma] \wedge trans\_to$
3. Find all states active after reading σ
   – All states that match on σ and are adjacent to an active state
4. $act = next$
   – Do the transition