# HW #1

* Code for stubbed methods was written and organized identical to given concatenation method
* Breakdown is given below showing proof equations side-by-side with implemented code
* Testing was done with various examples shown on page 3
* Automata drawings are found on page 4. They were created using the .get\_delta\_as\_dictionary() function after being run through various methods.

## Code Breakdown

Each method utilizing two DFAs/NFAs will check that their alphabets are identical and combine them if necessary. Additionally, methods utilizing two DFAs/NFAs will use tuples to specify which one a numbered state is referring to.

### Union

|  |  |
| --- | --- |
| Create local working variables. | (QA, SigmaA, deltaA, q0A, FA) = self.\_\_convert\_to\_nfa()  (QB, SigmaB, deltaB, q0B, FB) = other.\_\_convert\_to\_nfa()  Sigma = SigmaA  *# Note: SigmaA == SigmaB == Sigma* |
| Iterate over all states in both NFAs and add to new *Q*. Add new start state, which is a set of start states from both NFAs | q0 = frozenset({(1, q0A), (2, q0B)})  Q.add(q0)  for q in QA:  Q.add((1, q))  for q in QB:  Q.add((2, q)) |
| Iterate over all states in both NFA’s final states and add to new *F*. | for q in FA:  F.add((1, q))  for q in FB:  F.add((2, q)) |
| Iterate over all states in *M* for all letters in Sigma and set the new next state to all possible next states given by δ1. | for q in QA:  for a in Sigma:  delta[((1, q), a)] = frozenset({(1, r) for r in deltaA[(q, a)]}) |
| Iterate over all states in *N* for all letters in Sigma and set the new next state to all possible next states given by δ2. | for q in QB:  for a in Sigma:  delta[((2, q), a)] = frozenset({(2, r) for r in deltaB[(q, a)]}) |
| Set the next state for our start state to the set of possible start states for the empty string. | delta[(q0, '')] = q0  # Note: We already set q0 = {(1, q0A), (2, q0B)}, so we can just set the delta to q0 |
| Iterate over all letters in Sigma and set the next state of the start state to the empty set. | for a in Sigma:  delta[(q0, a)] = frozenset({}) |
| Return our new NFA. | return DFA("NFA", Q, Sigma, delta, q0, F) |

## Star

|  |  |
| --- | --- |
| Create local working variables. | (QA, SigmaA, deltaA, q0A, FA) = self.\_\_convert\_to\_nfa()  Sigma = SigmaA |
| Iterate over all states in *M* and add to new *Q*. Add new start state, which is a set of start states from *M*. | q0 = frozenset({q0A})  Q.add(q0)  for q in QA:  Q.add(q) |
| Iterate over all states in *M*’s final states and add to new *F*. Add new start state. | F.add(q0)  for q in FA:  F.add(q) |
| Iterate over all states in *M* for all letters in Sigma and set the new next state to all possible next states given by δ1. | for q in QA:  for a in SigmaA:  delta[(q, a)] = frozenset({r for r in deltaA[(q, a)]}) |
| Iterate over all final states in *N* for the empty string and set the new next state to the start state. | for q in FA:  delta[(q, '')] = q0 |
| Set the next state for our start state to the set of possible start states for the empty string. | delta[(q0, '')] = q0  # Note: We already set q0 = {q0A}, so we can just set the delta to q0 |
| Iterate over all letters in Sigma and set the next state of the start state to the empty set. | for a in Sigma:  delta[(q0, a)] = frozenset({}) |
| Return our new NFA. | return DFA("NFA", Q, Sigma, delta, q0, F) |

## Complement

|  |  |
| --- | --- |
| Create local working variables. Our new NFA will be identical to *M* with inverted final states. | (QA, SigmaA, deltaA, q0A, FA) = self.\_\_convert\_to\_nfa()  (Q, Sigma, delta, q0, F) = (QA, SigmaA, deltaA, q0A, set()) |
| Iterate over all states in *M* and add all states not in *F*1 to *F*. | for q in QA:  if q not in FA:  F.add(q) |
| Return our new NFA. | return DFA("NFA", Q, Sigma, delta, q0, F) |

## Intersection

|  |  |
| --- | --- |
| Use DeMorgan’s Law  First get the complements of our NFAs. | M = self.compliment()  N = other.compliment() |
| Then get the complement of the union. | (Q, Sigma, delta, q0, F) = M.union(N).complement().\_DFA\_\_convert\_to\_nfa() |
| Return our new NFA. | return DFA("NFA", Q, Sigma, delta, q0, F) |

# Testing

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **DFAs** | **.recognize()** | **M** | **N** | **Mc** | **Nc** | **MuN** | **MiN** | **Ms** | **Ns** | **MN** | **NM** |
| M = DFA(‘hola|world’)  N = DFA(‘hello|world’)  Mc = M.complement()  Nc = N.complement()  MuN = M.union(N) MiN = M.intersection(N)  Ms = M.star() Ns = N.star()  MN = M.concat(N)  NM = N.concat(M) | ‘hola’ | T | F | F | F | T | F | T | F | F | F |
| ‘hello’ | F | T | F | F | T | F | F | T | F | F |
| ‘world’ | T | T | F | F | T | T | T | T | F | F |
| ‘hol’ | F | F | T | T | F | F | F | F | F | F |
| ‘hel’ | F | F | F | T | F | F | F | F | F | F |
| ‘wor’ | F | F | T | T | F | F | F | F | F | F |
| ‘holahola’ | F | F | T | F | F | F | T | F | F | F |
| ‘hellohello’ | F | F | F | T | F | F | F | T | F | F |
| ‘worldworld’ | F | F | T | T | F | F | T | T | T | T |
| ‘holahol’ | F | F | T | F | F | F | F | F | F | F |
| ‘hellohel’ | F | F | F | T | F | F | F | F | F | F |
| ‘worldwo’ | F | F | T | T | F | F | F | F | F | F |
| ‘helloworld’ | F | F | F | T | F | F | F | T | F | T |
| ‘holaworld’ | F | F | T | F | F | F | T | F | T | F |
| ‘hhh’ | F | F | T | T | F | F | F | F | F | F |
| ‘www’ | F | F | T | T | F | F | F | F | F | F |
| ‘xyz’ | F | F | F | F | F | F | F | F | F | F |
| ‘’ | F | F | T | T | F | F | T | T | F | F |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **DFAs** | **.recognize()** | **M** | **N** | **Mc** | **Nc** | **MuN** | **MiN** | **MMs** | **NNs** | **McNs** | **NcMs** |
| M = DFA(‘0’, {‘0’, ‘1’})  N = DFA(‘1’, {‘0’, ‘1’})  Mc = M.complement()  Nc = N.complement()  MuN = M.union(N) MiN = M.intersection(N)  MMs = M.concat(M).star() NNs = N.concat(N).star()  McNs = M.complement().concat(N).star()  NcMs = N.complement().concat(M).star() | ‘0’ | T | F | F | T | T | F | F | F | F | F |
| ‘1’ | F | T | T | F | T | F | F | F | F | F |
| ‘01’ | F | F | T | T | F | F | F | F | F | F |
| ‘10’ | F | F | T | T | F | F | F | F | F | F |
| ‘00’ | F | F | T | T | F | F | T | F | F | T |
| ‘11’ | F | F | T | T | F | F | F | T | T | F |
| ‘000’ | F | F | T | T | F | F | F | F | F | T |
| ‘111’ | F | F | T | T | F | F | F | F | T | F |
| ‘0000’ | F | F | T | T | F | F | T | F | F | T |
| ‘1111’ | F | F | T | T | F | F | F | T | T | F |
| ‘001001’ | F | F | T | T | F | F | F | F | T | F |
| ‘110110’ | F | F | T | T | F | F | F | F | F | T |
| ‘xyz’ | F | F | F | F | F | F | F | F | F | F |
| ‘’ | F | F | T | T | F | F | T | T | T | T |

## Diagrams

|  |  |
| --- | --- |
| M = DFA(‘0’, {‘0’, ‘1’})  Start = q0 | N = DFA(‘1’, {‘0’, ‘1’})  Start = q0 |
| M.complement()  Start = q0 | N.complement()  Start = q0 |
| M.star()  Start = q0 | M.union(N)  Start = q1 |
| M.complement().concat(N).star()  Start = q3 | M.complement().union(N.complement().concat(M).star())  Start = q7 |