Languages of NFAs

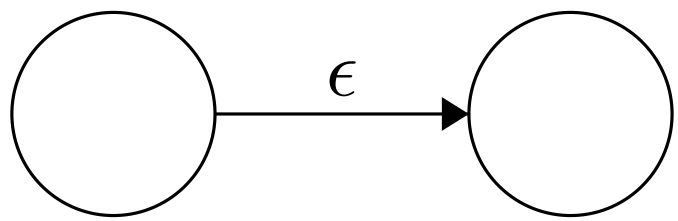
Are NFAs more powerful than DFAs; i.e. is there some NFA M, where there is no DFA that accepts L(M)?

- Every DFA is an NFA with only a single choice of transition at each state.
- Every NFA can be converted to a DFA that accepts exactly the same language.
- Class of languages of NFAs $\equiv$ Class of languages of DFAs $\equiv$ Regular Languages.

$\Rightarrow$ NFAs accept exactly the class of Regular languages.

ϵ-NFAs

An ϵ-NFA extends NFAs by also allowing a change of state on ϵ, the empty string; i.e. a change of state by ϵ-transition does not consume an alphabet symbol.



More complex? More powerful? More paths to follow?

Note: ϵ is not an alphabet symbol; i.e. $\epsilon \notin \Sigma$.

Simulating an ϵ-NFAs

Let S be a subset of states of an NFA. The ϵ-closure(S) is the set of all states reachable from a state in S by 0 or more ϵ-transitions.

```
states = e-closure({q0})
while NOT EOF do
  read ch
  states = e-closure(Union(delta(q, ch)
                         for each q in states))
end while
return states INTERSECT A != NULL
```

Convert ϵ-NFA to a DFA

We use the same technique as NFA to DFA, Subset Construction:

- Same basic idea but must consider the ϵ-closure of sets of states.
- Start with the ϵ-closure of the start state - this subset of states is the label for the start state of the DFA.
- Then determine the ϵ-closure of the set of state reachable on each alphabet symbol, etc.

Note: This conversion method could be automated (implement it).

$\Rightarrow$ Class of languages of ϵ-NFAs $\equiv$ Regular languages.

Scanning

Is C a regular language?

What do we use to build C programs?

- C keywords
- identifiers
- literals
- operators
- comments
- punctuation

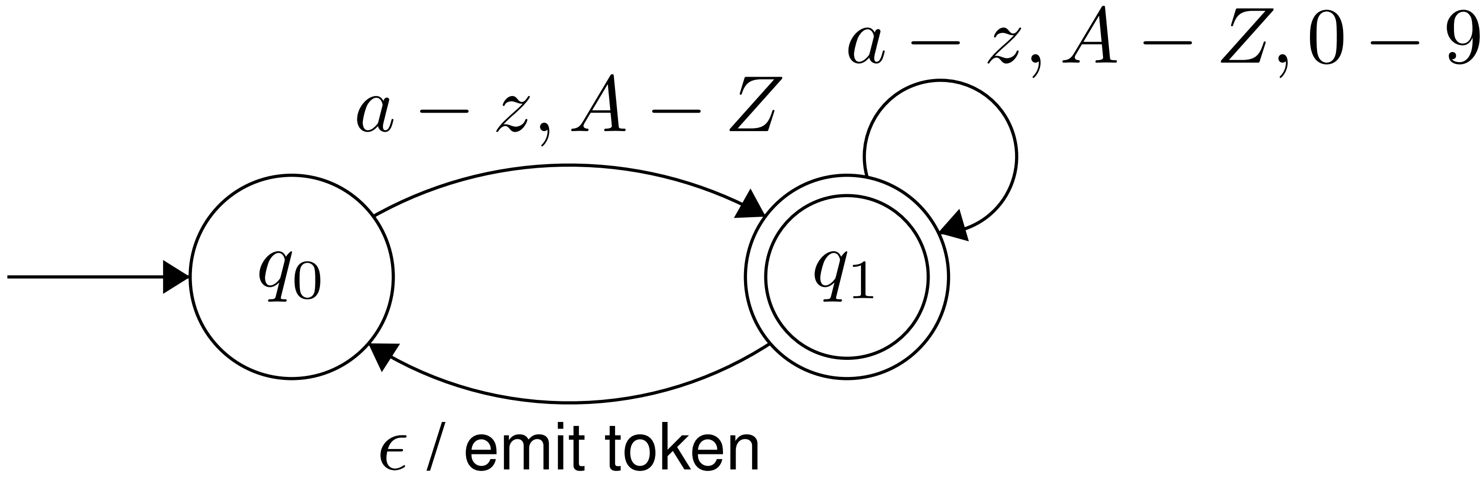
The above are all regular languages, so Union is also regular.

The language $L = \{\text{Valid C tokens}\}$ is regular.

LL^* is language of non-empty sequences of C tokens.

Unique Decomposition

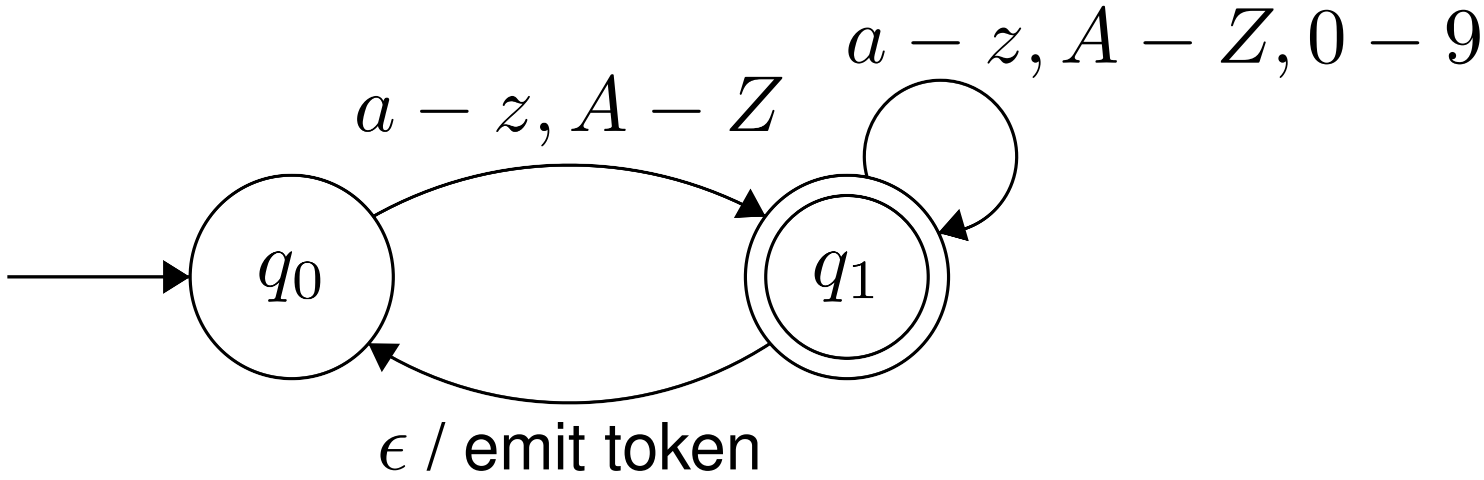
Consider an ϵ-NFA for valid C identifiers:



Given input $w =$ abcd is there only one decomposition of $w = w_1, \ldots, w_n$? When can we take the ϵ-transition?

Unique Decomposition

Consider an ϵ-NFA for valid C identifiers:



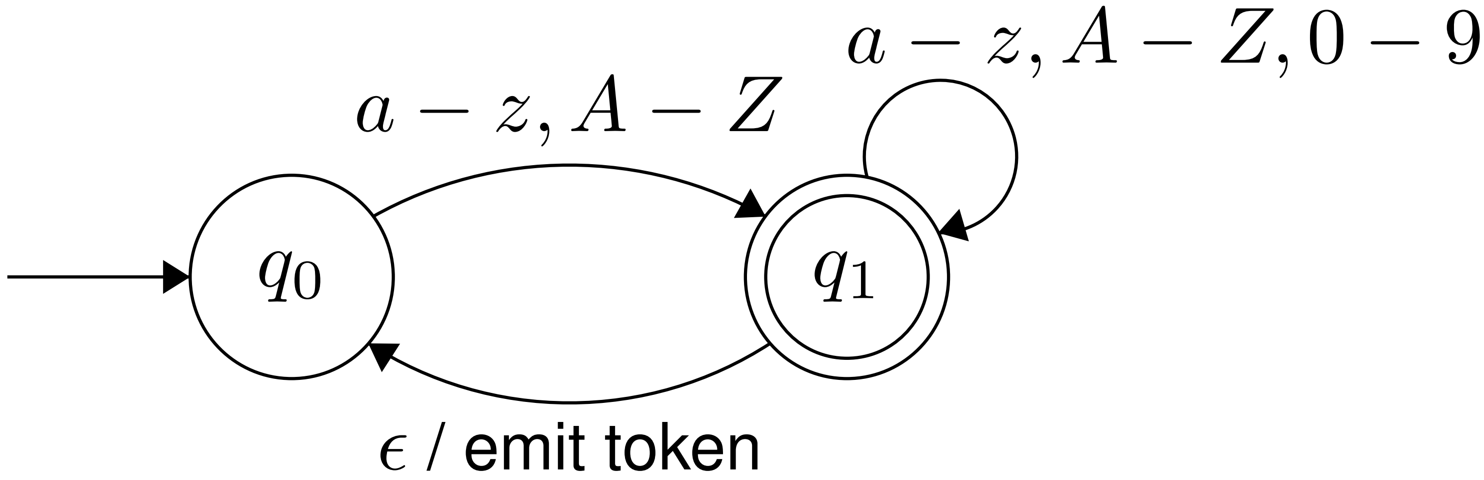
Given input $w =$ abcd is there only one decomposition of $w = w_1, \ldots, w_n$? When can we take the ϵ-transition?

No! Input could be decomposed into 1, 2, 3 or 4 tokens!

When should we use the ϵ-transition?

Unique Decomposition

Consider an ϵ-NFA for valid C identifiers:



Given input $w = \texttt{abcd}$ is there only one decomposition of $w = w_1, \ldots, w_n$? When can we take the ϵ-transition?

No! Input could be decomposed into 1, 2, 3 or 4 tokens!

When should we use the ϵ-transition? **Longest possible token**

To remove decomposition ambiguity, we could decide to only take the ϵ-transition (emit token) when there is no other choice. This emits the longest possible token at iteration.

Given $L = \{$aa,aaa$\}$ and input string $w =$ aaaa what happens?

To remove decomposition ambiguity, we could decide to only take the ϵ-transition (emit token) when there is no other choice. This emits the longest possible token at iteration.

Given $L = \{$aa,aaa$\}$ and input string $w =$ aaaa what happens?

Emits token: aaa, then crash (ERROR).

But emitting tokens: aa,aa would have been a valid decomposition.

What should we do?

Maximal Munch Algorithm

- Run DFA (without ϵ-transitions) until no non-error transitions available.
- If in an accepting state, emit token found.
  Else *backup* DFA to most recently seen accepting state, emit token, resume from here.
- ϵ-transition back to start state.

Implementation: will need a variable to track "most recent accepting state".

Simplified Maximal Munch Algorithm

Same as Maximal Munch except: if NOT in accepting state when no non-error transitions available, simply crash (ERROR) - no backtracking.

- Run DFA (without ϵ-transitions) until no non-error transitions available.
- If in an accepting state, emit token found, ϵ-transition back to start state.
  Else ERROR

Exercise: Give an example where MM finds a valid decomposition but Simplified MM gives ERROR.