Closure of Regular Languages

Union, Concatenation, Kleene Star

The class of regular languages is closed under these three operations by definition.

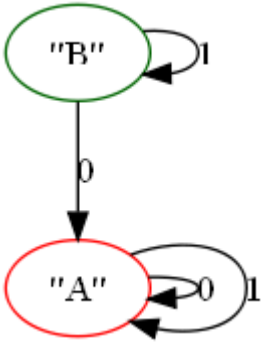
Complementation

Take a DFA for L and change the status - final or non-final - of all its states. The resulting DFA will accept exactly those strings that the first one rejects. It is, therefore, a DFA for C(L).

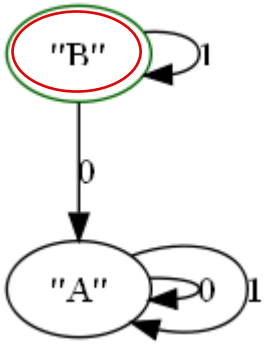
Thus, the complement of every regular language is regular.

Complement Example

Contains a “0”



Starts with a “1”



Intersection

The intersection $L \cap M$ of two regular languages must be regular, too. The quickest way to see it is by expressing intersection in terms of union and complementation, and then referring to the already established facts that the class of regular languages is closed under union and complementation.

The requisite set-theoretical identity is one of DeMorgan's laws:

$$L \cap M = C(C(L) \cup C(M))$$

Constructive Proof

A more direct proof is based on a construction that given two DFAs A and B, produces a third DFA C such that L(C) = L(A) ∩ L(B). The states of C are pairs (p,q) , where p is a state of A and q is a state of B. A transition labeled a leads from (p,q) to (p',q') iff there are transitions

$$p \xrightarrow{a} p' \qquad\qquad q \xrightarrow{a} q'$$

in A and B. The start state is the pair or original start states; the final states are pairs of original final states. The transition function

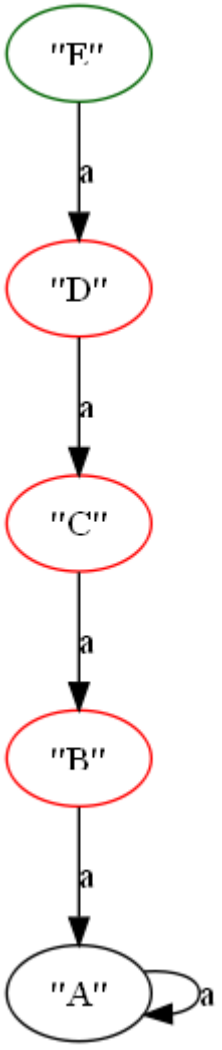
$$\delta_{A\cap B}(q,a) = (\delta_A(q,a), \delta_B(q,a))$$

This is called the *product construction*.

Example 1

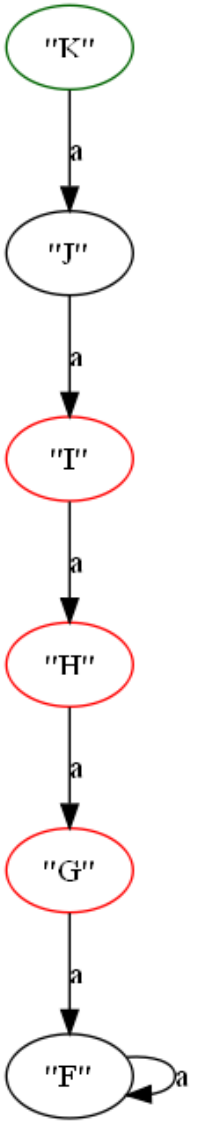
aa+aaa+aaaa

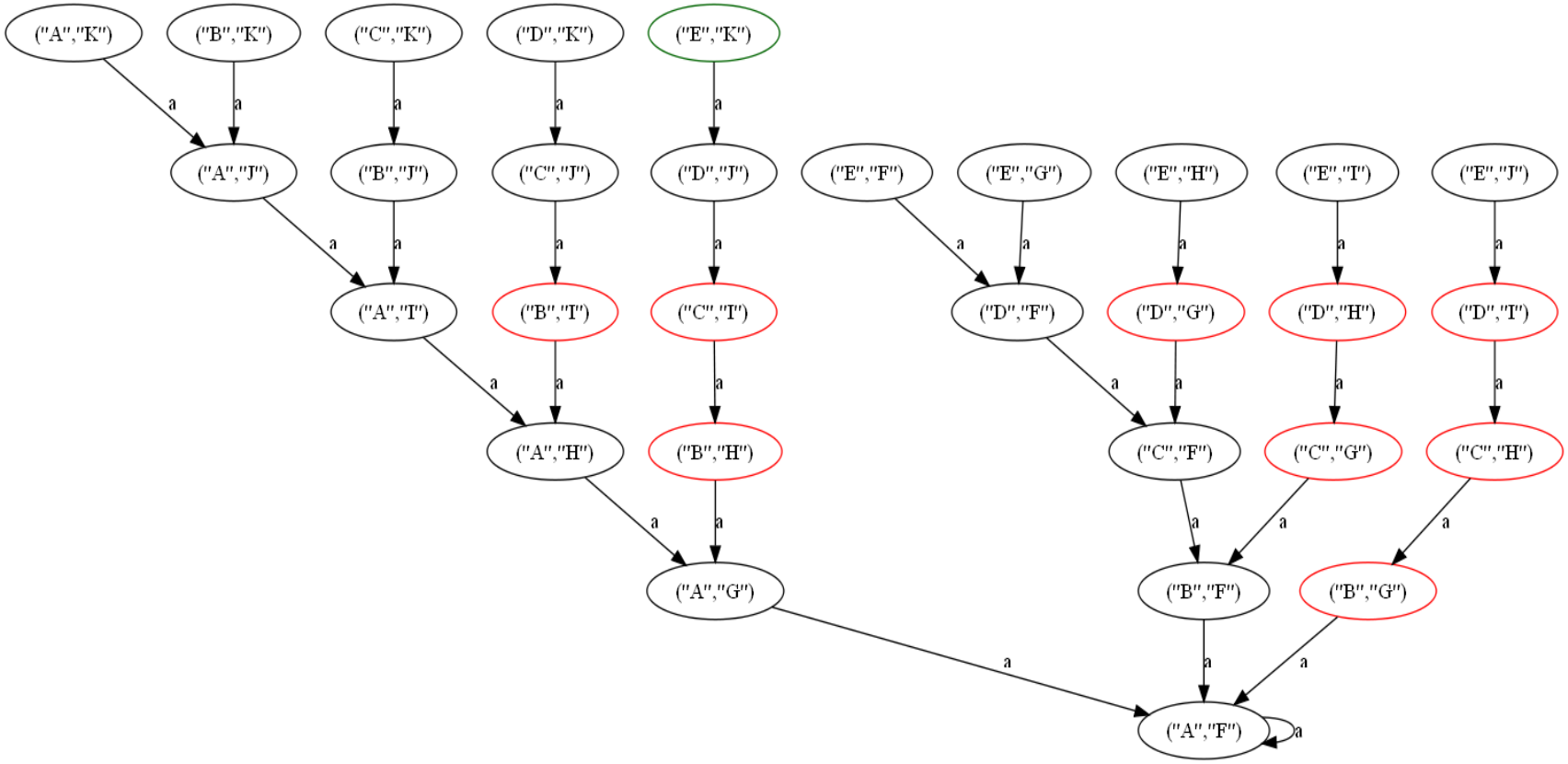
a+aa+aaa



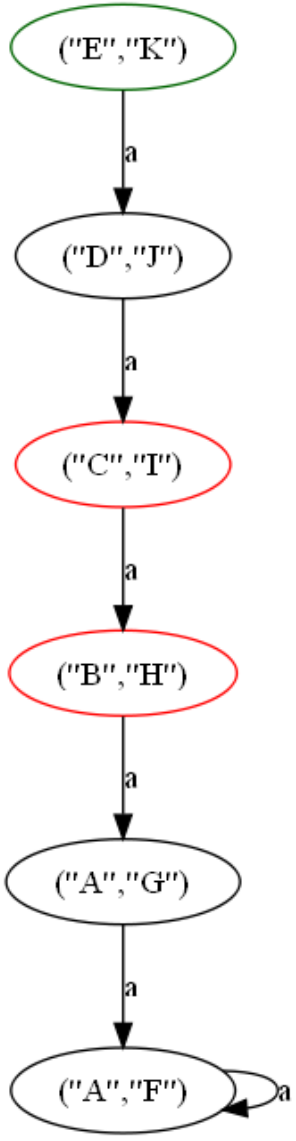
What is the intersection?

Make a new DFA where states of the new DFA are pairs of states form the old ones




("A","K")
("B","K")
("C","K")
("D","K")
("E","K")
("A","J")
("B","J")
("C","J")
("D","J")
("E","F")
("E","G")
("E","H")
("E","I")
("E","J")
("A","I")
("B","I")
("C","I")
("D","F")
("D","G")
("D","H")
("D","I")
("A","H")
("B","H")
("C","F")
("C","G")
("C","H")
("A","G")
("B","F")
("B","G")
("A","F")
a

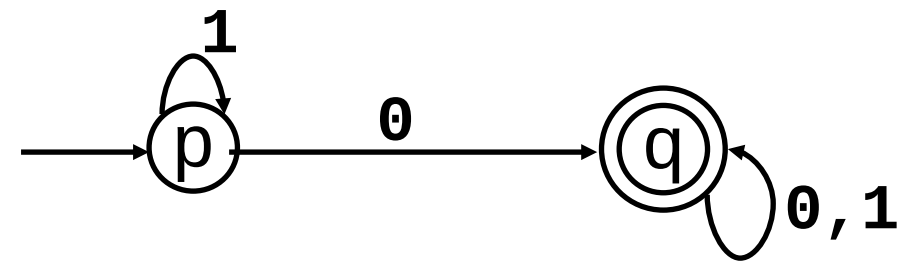
Reachable states only



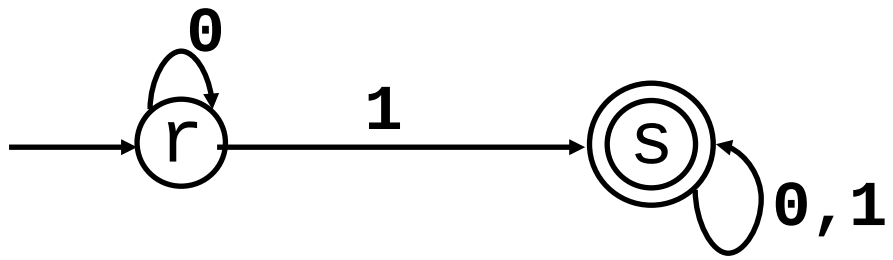
Intersection

{a,aa,aaa} ∩ {aa,aaa,aaaa}

Example 2



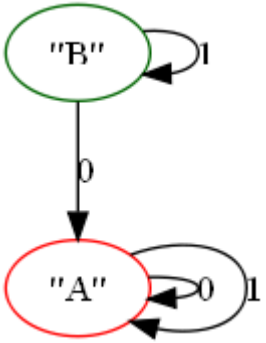
A – string contains a 0



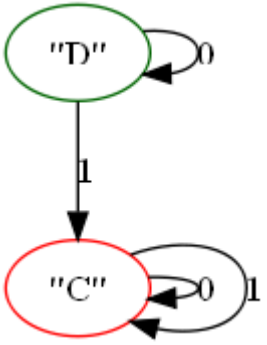
B – string contains a 1

C – string contains a
0 and a 1

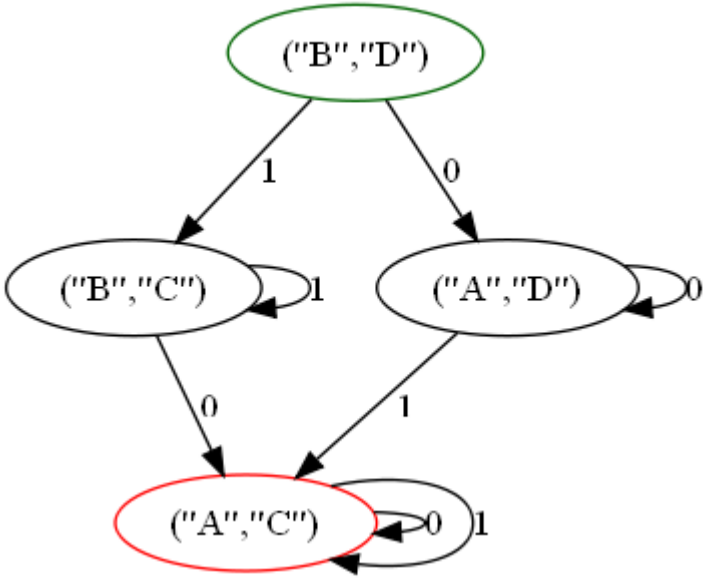
Contains a “0”



Contains a “1”



Contains both a “1” and a “0”



Difference

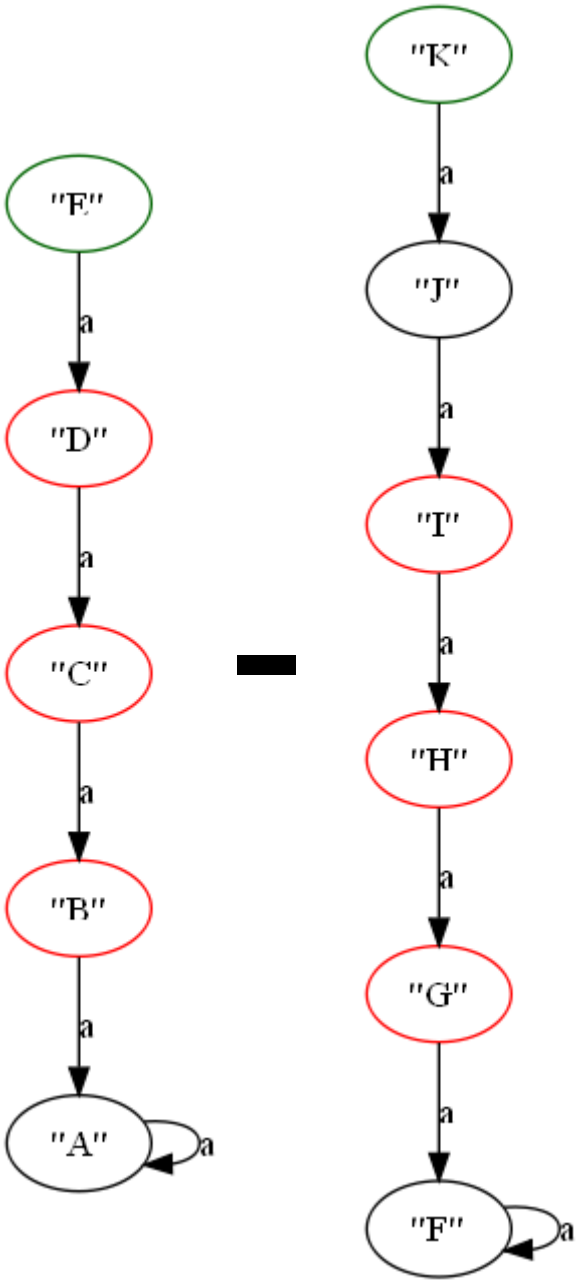
The identity:

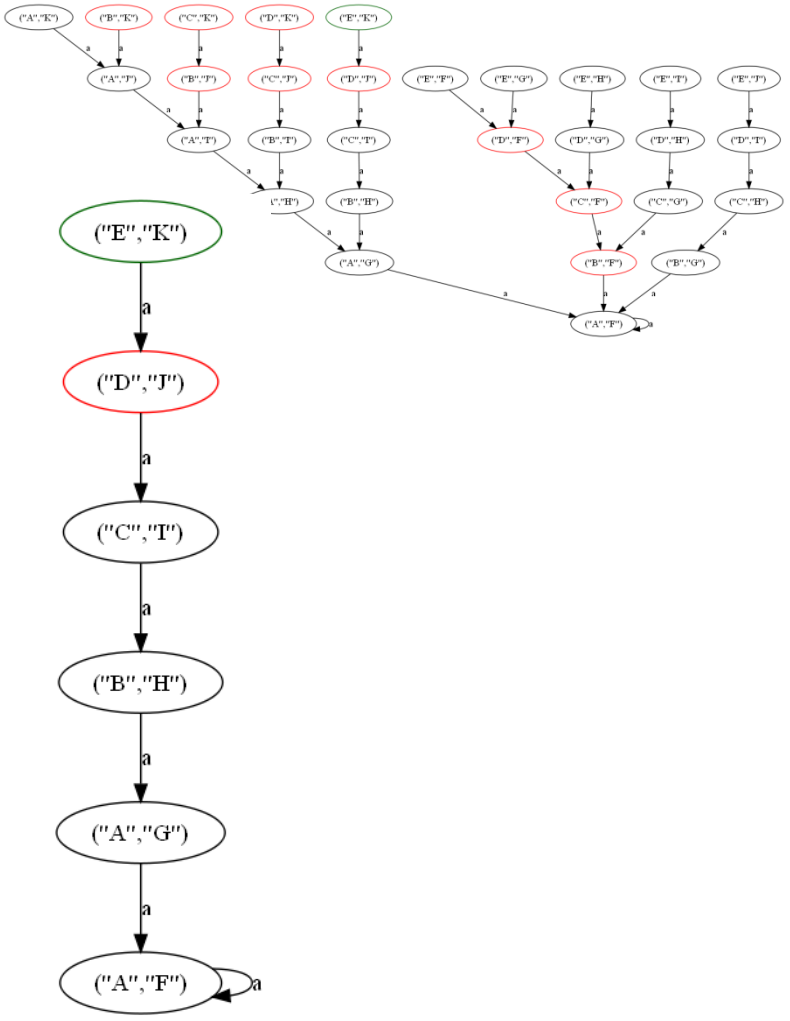
$$L - M = L \cap C(M)$$

reduces the closure under set-theoretical difference operator to closure under complementation and intersection.

Example Difference

$$L - M = L \cap C(M)$$





Reversal

As we saw in the homework, closure under reversal is most easily seen using ε-NFAs. If you take such an automaton for L, you need to make the following changes to transform it into an automaton for L^{Rev}:

1. Reverse all arcs
2. The old start state becomes the only new final state.
3. Add a new start state, and an ε-arc from it to all old final states.

Example Closure Construction

Given a language L, let L' be the set of all prefixes of even length of strings which belong to L. We prove that if L is regular then L' is also regular.

It is easy to show that prefix(L) is regular when L is (How?). We also know that the language **Even** of even length strings is regular (How?). All we need now is to note that

$L' = \mathbf{Even} \cap \text{prefix}(L)$

and use closure under intersection.

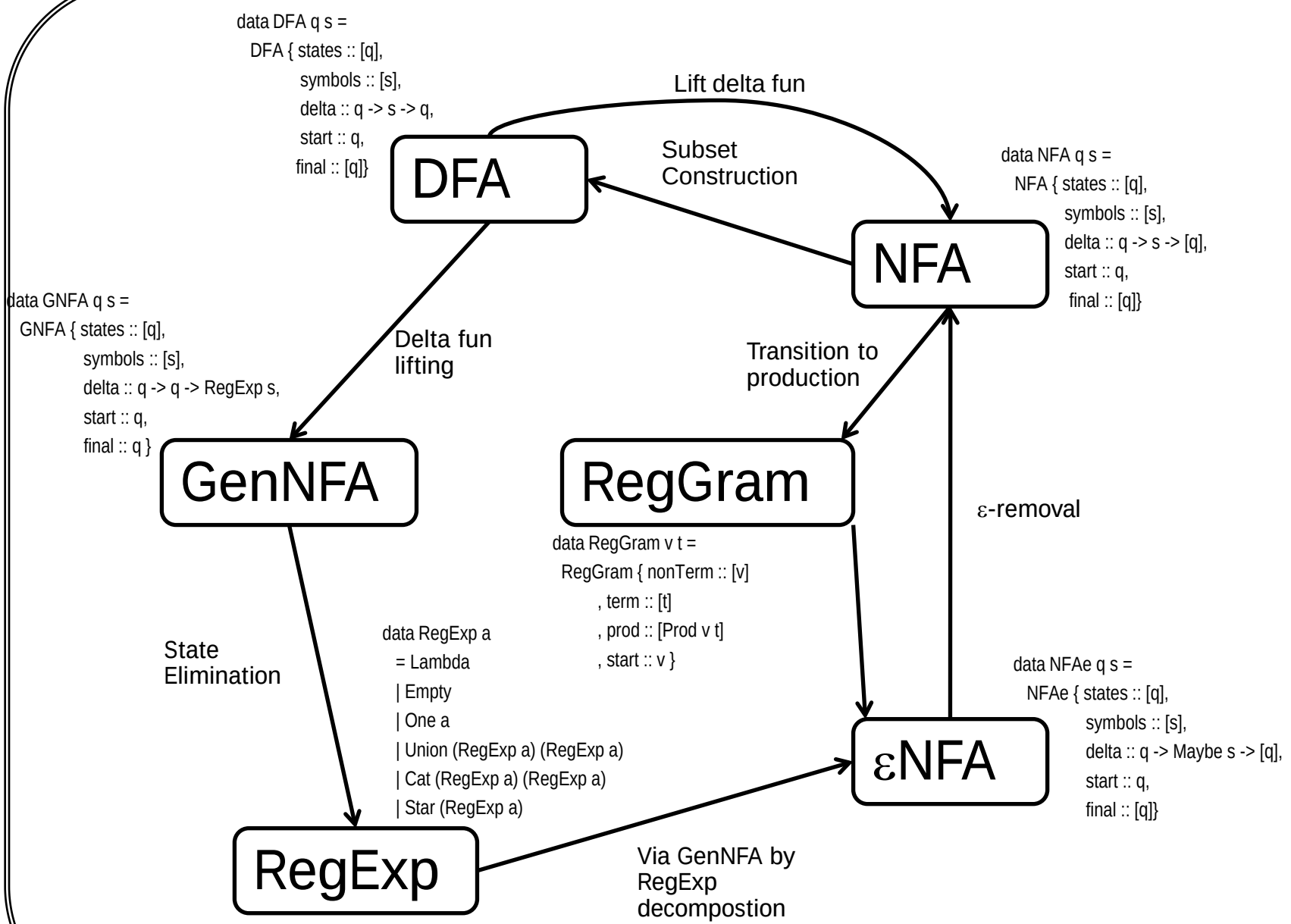
Conclusion

- We have studied the class of regular languages
- We saw many different ways to express a regular language
  1. Regular Expressions
  2. DFA
  3. NFA (More than one choice per symbol)
  4. Epsilon-NFA (L transitions)
  5. Generalized NFA (RegExp on transitions)
  6. Regular Grammars
- We showed that all were equally expressive
- Some were easier to use than others to describe some languages

Algorithms

We studied algorithms to transform one description into another

1. Subset construction
2. GenNFA expansion
3. State minimization
4. Λ-closure and removal


data DFA q s =
DFA { states :: [q],
symbols :: [s],
delta :: q -> s -> q,
start :: q,
final :: [q]}
Lift delta fun
DFA
Subset
Construction
data NFA q s =
NFA { states :: [q],
symbols :: [s],
delta :: q -> s -> [q],
start :: q,
final :: [q]}
NFA
data GNFA q s =
GNFA { states :: [q],
symbols :: [s],
delta :: q -> q -> RegExp s,
start :: q,
final :: q }
Delta fun
lifting
Transition to
production
GenNFA
RegGram
ε-removal
data RegGram v t =
RegGram { nonTerm :: [v]
, term :: [t]
, prod :: [Prod v t]
, start :: v }
State
Elimination
data RegExp a
= Lambda
| Empty
| One a
| Union (RegExp a) (RegExp a)
| Cat (RegExp a) (RegExp a)
| Star (RegExp a)
data NFAe q s =
NFAe { states :: [q],
symbols :: [s],
delta :: q -> Maybe s -> [q],
start :: q,
final :: [q]}
εNFA
RegExp
Via GenNFA by
RegExp
decompostion

Properties

We saw that Regular grammars have many properties

Closure properties

- Union
- Kleene – star
- Intersection
- Complement
- Reversal
- Difference

Uses

Regular languages and their algorithms have many uses in computer science

1. Searching for “rich” patterns in text
2. Describing the structure of tokens in computer languages
3. Describing the sequence of operators in protocols (understanding the exchange of information between machines)
4. Understanding complexity. What is easy, what is hard, what is not possible.
5. Hardware design. State machines model circuits
6. Model Checking. Finite state approximations of real programs

Not all languages are Regular

Some languages are not regular

The pumping lemma provides one means to show that a language is not regular.

How can we describe these languages?

This is the topic of the next section of the course