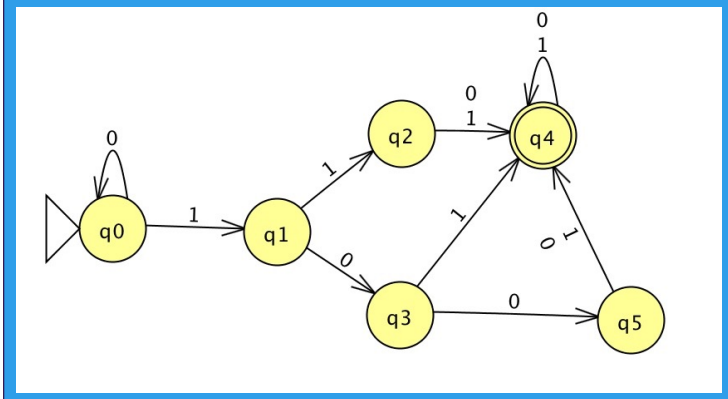
Theory of Computation and Programming Languages



DFAs

CS51 – Spring 2026

Today we will start talking about automata, fundamental models of computation.

Models of computation

- We will see some *very* simple models of computation.
  - Much simpler than modern computers.
- Why study simple models?
  - They are much easier to understand.
  - They allow us to reason and think about them.
  - Even if simple, many have similar capabilities as modern computers.
    - Just not as fast.
- Reasoning about simple models can allow us to reason about computational power of computers.
- Answer questions like, what are the capabilities and limitations of computers?
  - Are there problems that we could NEVER (even given an infinite amount of time, memory, etc.) answer with a computer?

2

These models of computation are much simpler than modern computers to the point that you might be wondering why we even study them. First, because of their simplicity, they are much easier to understand and thus allow us to reason and think about them. Their simplicity can also be deceptive; you might think they are less powerful but in fact many have similar capabilities as modern computers, they are just not as fast. Putting these arguments together, reasoning about simple models allows us to reason about computational powers of any computer, even modern ones. That way, we will be able to answer not only what are the capabilities but also what are the limitations of computers. For example, are there problems that we could never answer with a computer no matter how fancy it is in terms of memory, processor, and even if we give it an infinite amount of time to solve a problem?

Languages

- A language L is a set of strings.
- Strings in that language are built up from an alphabet of characters, denoted as Σ.
- We want to ask a simple question, like:
  - Given a string of characters in Σ does it belong to the language L?

3

To start modeling our first automaton, we will need to establish some terminology about languages. We define a language L as a set of strings. The strings in that language are bult up from an alphabet of characters denoted as sigma. We will be tackling simple questions, like given a string of characters from the alphabet sigma, does this string belong to the language L. An example of a sigma would be the english alphabet with its 26 characters. If for example, our language L is that of actual English, the string 'computer' belongs to it but the string 'fgejghfe' is gibberish and does not belong to L despite comprising of valid characters from our English alphabet.

Deterministic Finite Automata (DFAs)

- We have a set of **states**, indicated by circles.

- We have a **start state** where computation begins (indicated by a triangle)

- We have a collection of **final states** (indicated by a concentric circle)

- For **each** state and **each** letter in our alphabet, we have a transition to another state
- Note it's critical that for each state there is **exactly one** transition for each letter in the alphabet.
- If not, then if we encountered that letter we wouldn't know what to do!

4

Now that we defined what a language is, we will start building our first automaton (machine) that can answer this type of question. Specifically, today we will see a type of automaton called deterministic finite automaton or DFA for short. DFAs have a set of states which we denote by circles. We tend to label these states so that we distinguish them, e.g, q0. Out of all the states that the DFA has, one is very special and is called the start state. The computation begins at the start state. Visually, we distinguish the start state among all states by attaching a triangle next to it. We also have a collection of possibly more than one final states where the computation ends and we reach what is known as an accept state. Visually, final states are indicated by a concentric circle. The important characteristic about DFAs is that for *each* state, they need to define how we transition to another state for *each* letter in our alphabet. Transitions are indicated with arrows and show how the computation progresses. I can't stress it enough: it is critical that for each state in a DFA there is exactly one transition for each letter in the alphabet. If not, then if we encounter that letter we wouldn't know what to do.

Computing on a DFA

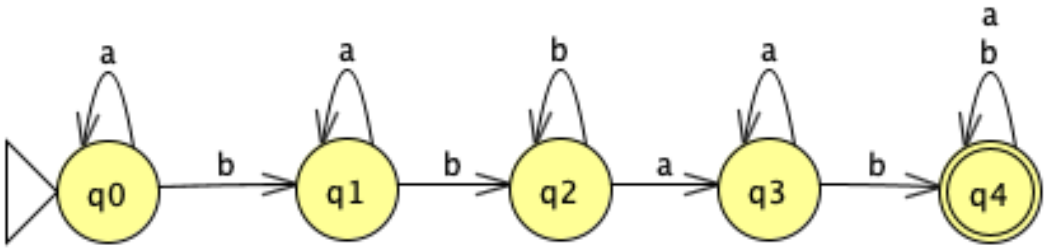
- We have a string as input on a tape.
- We start at the beginning of the string.
- We read a symbol from the tape and transition to the state indicated by the model.
- If we end in a final state (i.e. when we get to the end of the string we are in a final state), we accept the string.
- Otherwise, if when we get to the end of the string on the tape and we're in a non-final state we reject.
- All strings accepted by a DFA define a language.
- We will use JFLAP https://www.jflap.org/ to create DFAs and evaluate whether a DFA recognizes a specific string.

5

Computing on a DFA unfolds as follows. We assume that the string we want to determine whether it belongs to a language is given to us as input on a tape. We start at the beginning of the string, that is the first character or first cell of the tape. We read a symbol from the tape and transition to the state as indicated by our DFA, that is we follow the arrow that matches the letter we just read. If we end up in a final state, that is we read the whole string and we are in a final state, we accept the string. If we ran out of characters and don't end up in a final state, then we reject that string as it does not belong to L. All strings accepted by a DFA define a specific language that that DFA describes. All the visualizations and simulations we will see together are built using JFLAP.

Example

- You are given the following DFA over the alphabet {a, b} and the string abbbaabab.
- Start in q0.

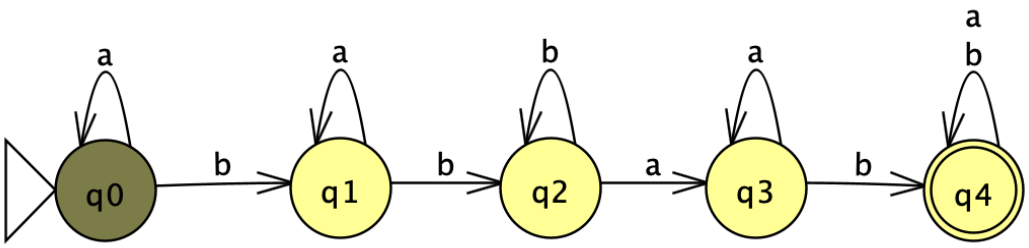


6

Let's see an example of a DFA which someone built for us. Note that q0 is the start state, there are 6 states altogether, q0-q4, and one final state, q4. For each state, we HAVE TO have an arrow for each letter of the alphabet (in this case the alphabet is just two characters, a and b). Remember, the goal here is to receive a string of characters from this alphabet and determine whether the string belongs to the language defined by this DFA. Say we are given the string abbbaabab. We start in q0 reading the first character a.

Example

- **a**bbbaabab
- a -> q0

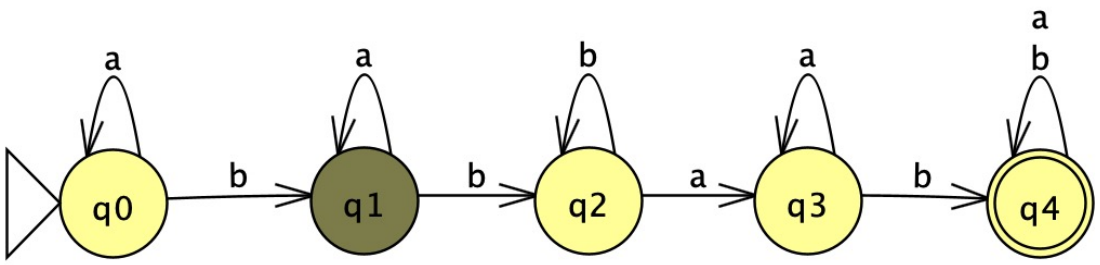


7

q0 tells us that when we read an a we transition back to itself (kind of like a self-loop). So after having read the first character, we remain in q0.

Example

- a**b**bbaabab
- b -> q1

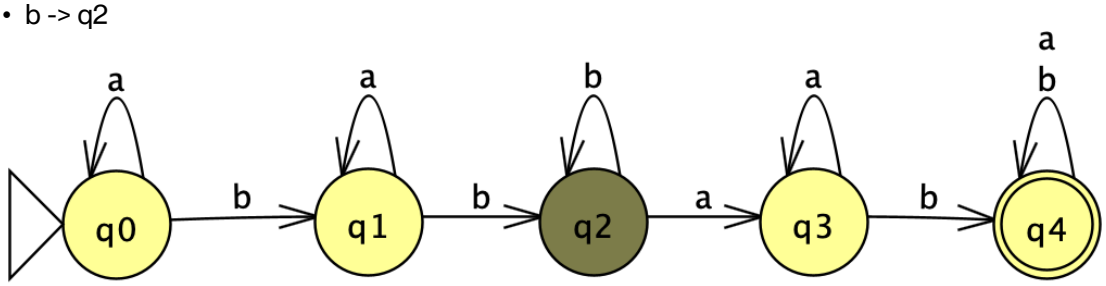


8

We move to the next character of our string in the tape. That is the character b. We were in q0 which says that if we read a b, we need to transition in q1.

Example

- ab**b**baabab
- b -> q2

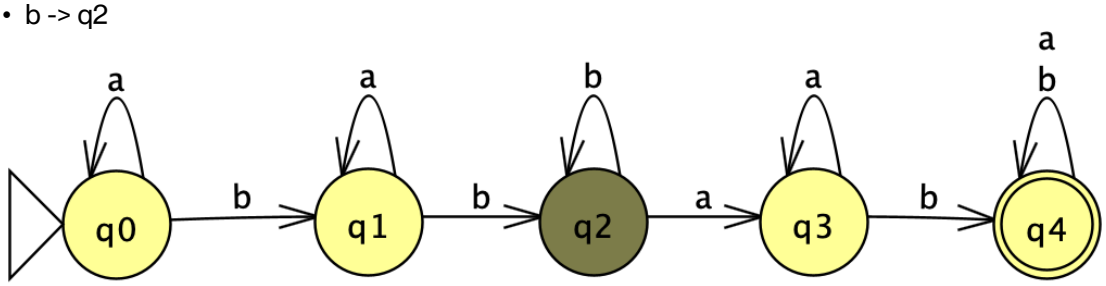


9

We move to the next character which is a b. We were in q1 which says that if we read a b, we need to transition to q2.

Example

- abb**b**aabab
- b -> q2

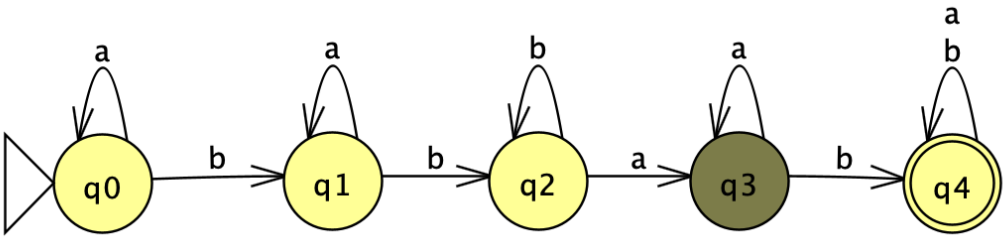


10

We read the next character which is again a b. We were in q2 which says that if we read a b, we stay in q2.

Example

- abbb**a**abab
- a -> q3

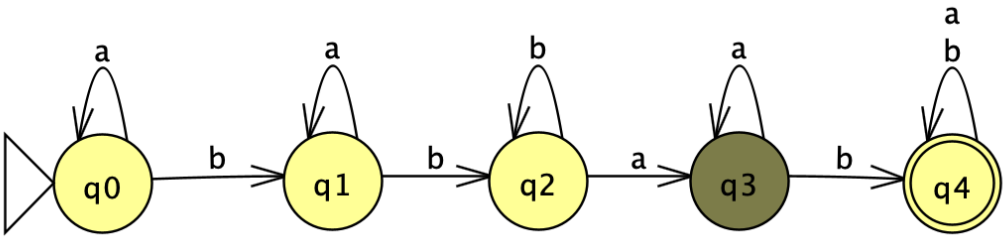


11

Our next character is a. We were in q2 which says that if we read an a we transition to q3.

Example

- abbbaabab
- a -> q3

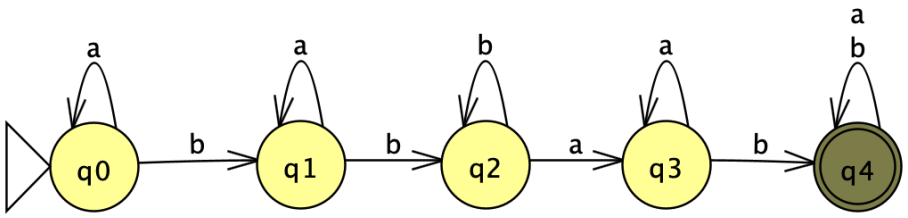


12

Our next character is again a. From q3 we stay in q3 with an a.

Example

- abbbaa**b**ab
- b -> q4

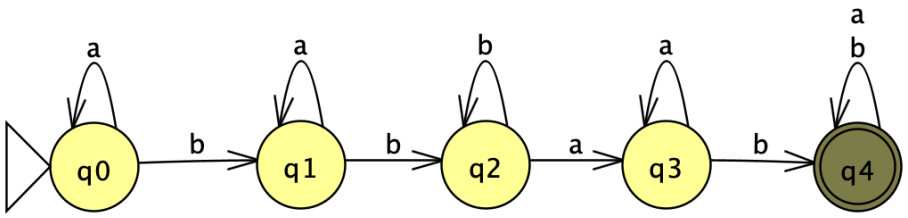


13

Next character is a b which takes us from q3 to q4

Example

- abbbaabab
- a -> q4

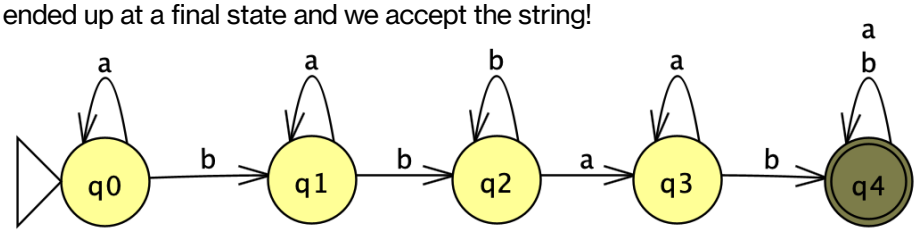


14

Note that q4 is a final state and any character we read from now on, be it a or b, still leaves us at q4. So we will continue reading the remaining characters, first with the penultimate a

Example

- abbbaaba**b**
- b -> q4
- We ended up at a final state and we accept the string!

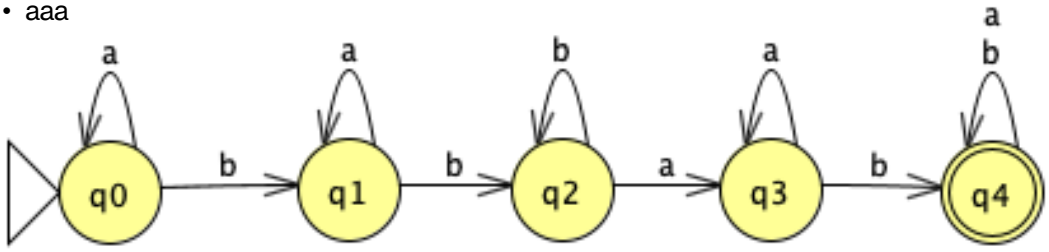


15

and then the final b. Since we read the entire string abbbaabab and ended up in a final state, we accept the string as belonging to the language described by this DFA.

Practice time

- What about the following strings?
- aabab
- bbb
- babbab
- aabbaab
- aaa

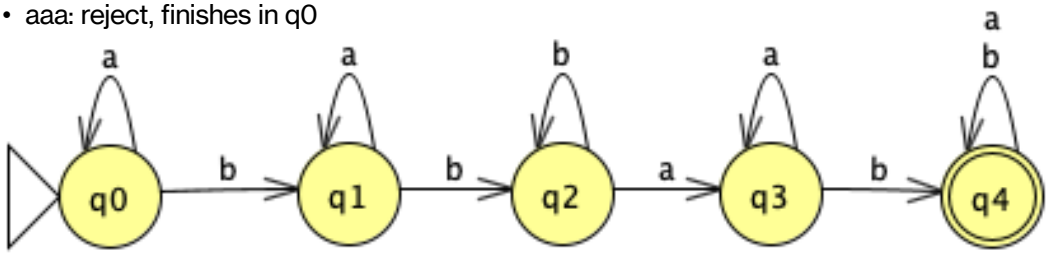


16

Your turn now. Do you think these strings belong to the language described by this DFA? For each of the five strings, state whether the DFA accepts it or rejects it.

Answer

- What about the following strings?
- aabab: reject, finishes in q2
- bbb: reject, finishes in q2
- babbab: accept
- aabbaab: accept
- aaa: reject, finishes in q0



17

Did you end up with these results?

Formal definition of DFAs

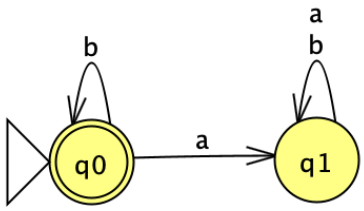
- A Deterministic Finite Automaton (DFA) is a finite state machine that accepts or rejects finite strings of symbols and produces the same unique computation for each unique input string. For any given finite input string, the DFA will halt and either accept or reject the string. A DFA, M, is said to recognize a language, $L(M)$, which is the set of all strings that M accepts.
- Formally, a DFA is described by a 5-tuple $(Q, \Sigma, \delta, q0, F)$
  - Q is a finite set of states
  - Σ is a finite set of input symbols also known as the alphabet
  - δ is a state transition function ($\delta : Q \times \Sigma \rightarrow Q$)
  - $q0$ is the start state ($q0 \in Q$)
  - F is a set of accept states ($F \subseteq Q$).

18

This example illustrated how a DFA works. Formally, a DFA is a finite state machine that accepts or rejects finite strings of symbols and produces the same unique computation for each unique input string. For any given finite input string, the DFA will halt and either accept or reject the string. A DFA, M, is said to recognize a language, $L(M)$, which is the set of all strings that M accepts. We would need five components to describe a DFA. The set of states, the alphabet, transitions for each state for each character, what is the start state and the set of accept (final) states.

Practice Time

- What language is described by this DFA?

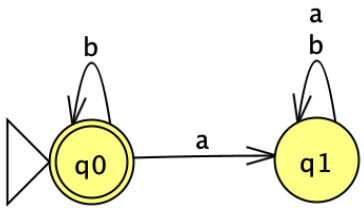


19

Let's have fun with a few automata. You are given the automaton shown here. What language is described by this DFA? Try different strings to determine that.

Answer

- All strings of a's and b's that don't contain any a.

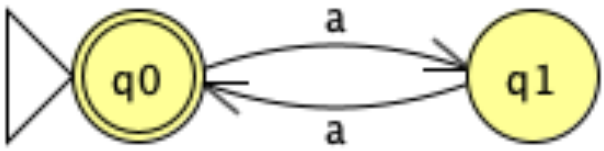


20

It would be the language over the alphabet {a,b} that contains all strings of a's and b's that don't contain any a.

Practice Time

- What language is described by this DFA?

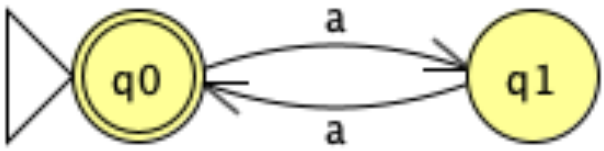


21

How about this DFA?

Answer

- All strings of a's that contain an even number of a's

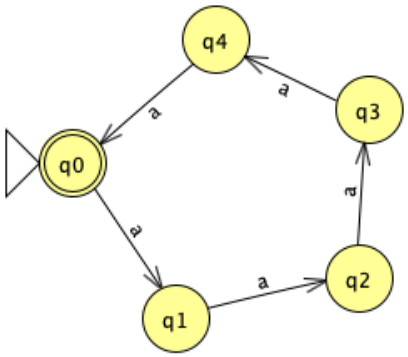


22

It would be the language over the alphabet {a} that contains all strings of a's that contain an even number of a's

Practice Time

- What language is described by this DFA?

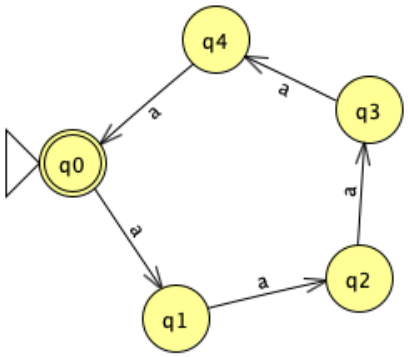


23

This looks a bit more complex. What do you think?

Answer

- All strings of a's that are multiples of 5

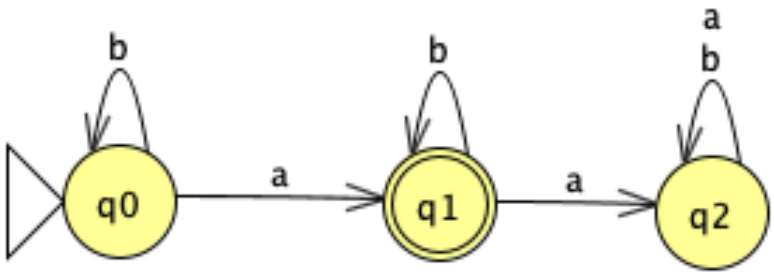


24

It would be the language over the alphabet {a} that contains all strings of a's that are multiples of 5

Practice Time

- What language is described by this DFA?

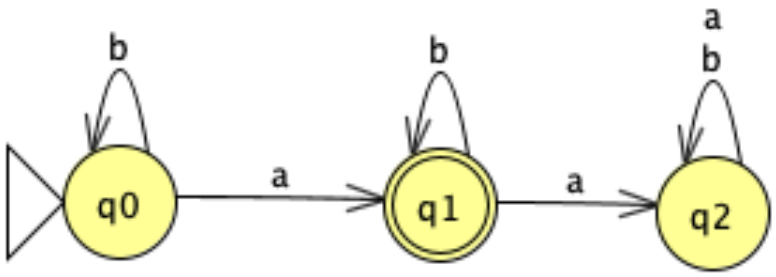


25

How about this DFA?

Answer

- All strings of a's and b's that contain a single a.

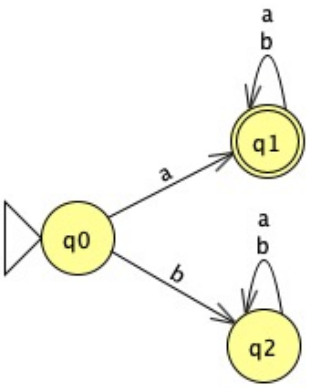


26

It would be the language over the alphabet {a,b} that contain a single a

Practice Time

- What language is described by this DFA?

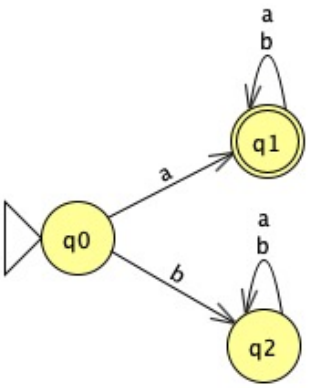


27

This one?

Answer

- All strings of a's and b's that start with a.



28

It would be the language over the alphabet {a,b} that start with an a.

Regular Expressions

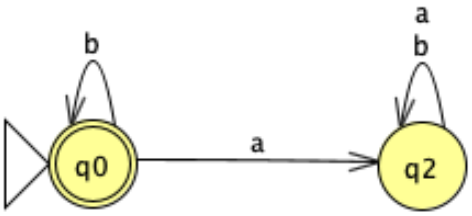
- Another way of expressing a language but *not* a computational model
- Regular expressions are built from :
  - * (Kleene star): 0 or more repetitions (highest precedence)
  - concatenation
  - | : union or "or" (lowest precedence)
  - () : used to enforce precedence
- For example,
  - a|b* denotes: a, b, bb, bbb, and so on
  - (a|b)* denotes: a, b, aa, ab, ba, bb, aaa, and so on, that is the set of all strings with no symbols other than "a" and "b"
- Let's revisit the examples we just saw and describe them using a regular expression.

29

Beyond DFAs, another way of expressing a language is to work with regular expressions. Note that regular expressions are not computation models. We can build regular expressions with a few rules. * is called the Kleene star and when it follows a character it denotes that this character will be repeated 0 or more times. We can concatenate two characters (we just put them nect to each other, we don't use any symbol between them). We can take the union (or) of two characters using |. We can also use parentheses to enforce precedence otherwise we go Kleene star>concatenation>union.
For example, the regular expression a|b* denotes strings like a, b, bb, bbb, bbbb, and so on
(a|b)* looks similar but it's quite different: a, b, aa, ab, ba, bb, aaa, and so on. This would be all strings that we can build using only as and bs.
Let's revisit the examples we just saw and try to describe them using regular expressions.

Example

- All strings of a's and b's that don't contain any a.
- Regular expression: b*

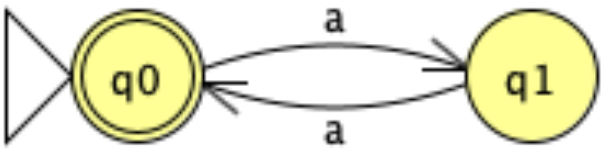


30

All strings of a's and b's that don't contain any a can be written as b* (note that q0 would accept even the empty string)

Example

- All strings of a's that contain an even number of a's
- Regular expression: (aa)*

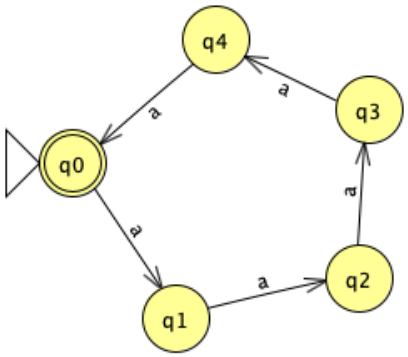


31

(aa)* would be the regular expression that corresponds to all strings of as that contain an even number of a's as we have multiples of twos.

Example

- All strings of a's that are multiples of 5
- Regular expression: (aaaaa)*

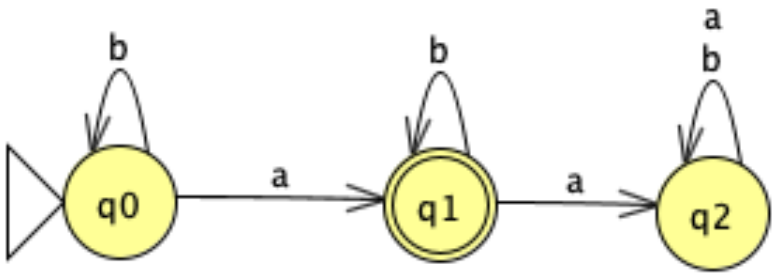


32

Similarly (aaaaa)* is for as that are multiple of 5s

Example

- All strings of a's and b's that contain a single a.
- Regular expression: b*ab*

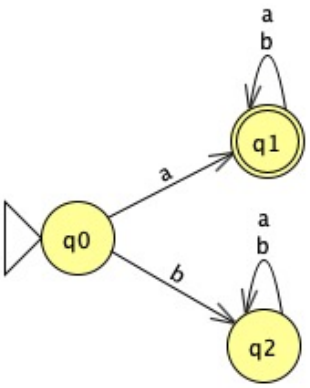


33

this is slightly more challenging. We would need to accept all strings as long as they contain one a. Since we don't care about how many bs precede or follow that single a, the regular expression would be b*ab*

Example

- All strings of a's and b's that start with a.
- a(a|b)*

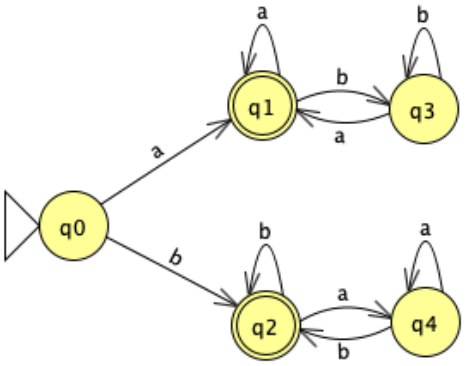


34

To start with a, we would need a(a|b)* since we don't care what happens after the first a in terms of as and bs

Practice Time

- What language is described by this DFA?
- What regular expression describes this language?

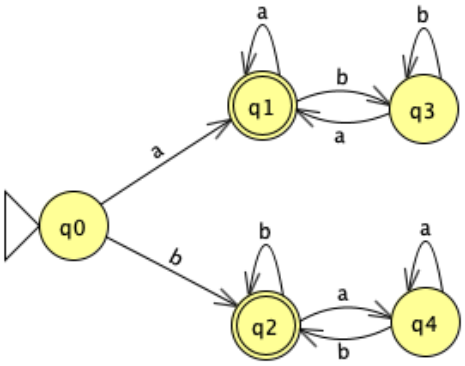


35

This is a new DFA. What language is described by this DFA and can you come up with a regular expression for it?

Answer

- Strings of a's and b's that start and end with the same letter
- (a(a|b)*a)|(b(a|b)*b)|a|b

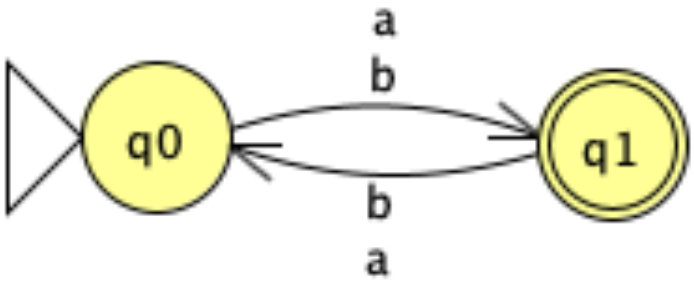


36

This would be all strings of as and bs that start and end with the same letter. The regular expression is quite complex. (a(a|b)*a) corresponds to all strings that start and end with a. b(a|b)*b corresponds with all strings that start and end with b. We take the union of the two. Why do we also need a|b? because we would need to accept even strings with a single character

Practice Time

- What language is described by this DFA?
- What regular expression describes this language?

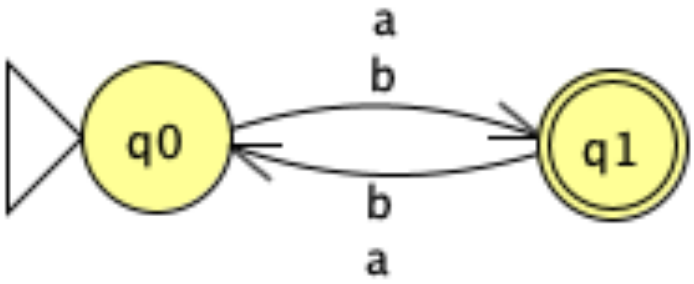


37

A new DFA again, same idea of what you have to do.

Answer

- Strings of a's and b's that have an odd length
- (a|b)((a|b)(a|b))*

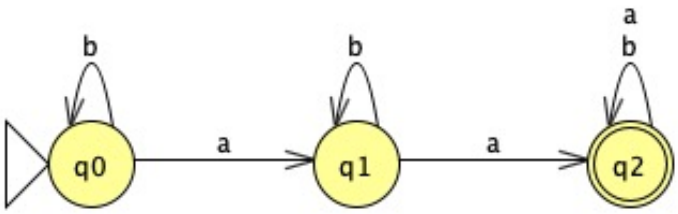


38

The language corresponds to all strings of as and bs that have an odd length. Since we again accept a single character we start with (a|b). We repeat this in multiples of two to maintain the odd length.

Practice Time

- What language is described by this DFA?
- What regular expression describes this language?

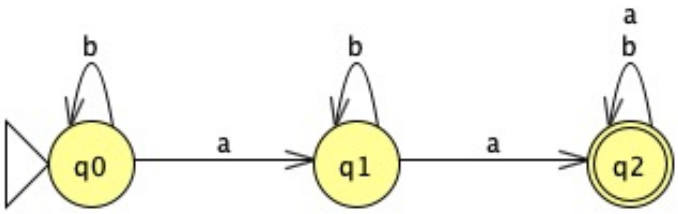


39

How about this?

Answer

- Strings of a's and b's that have two or more a's
- (a|b)*a(a|b)*a(a|b)*
- b*ab* a(a|b)*

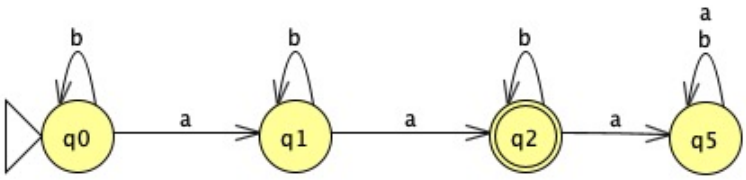


40

The language would be strings of a's and b's that have two or more a's. Look at the regular expression. We have two as'. before, in-between, and after them we can have any combo of a and b. Or we could have written it differently as b*ab* a(a|b)*

Practice Time

- What language is described by this DFA?
- What regular expression describes this language?

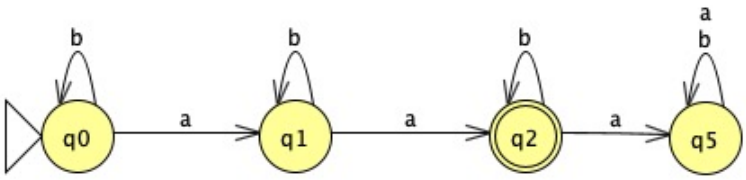


41

Almost there. How about this one?

Answer

- Strings of a's and b's that have exactly two a's
- b*ab*ab*

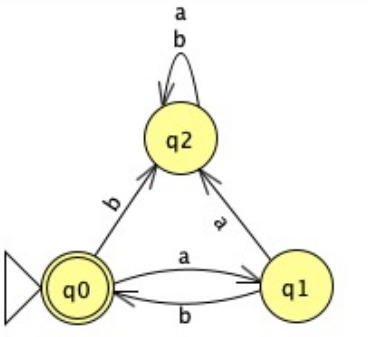


42

This is a variation of what we just saw and it's all strings that have exactly two a's. The regular expression needs to reflect now that we don't care about the b's before, in-between, and after those two a's

Practice Time

- What language is described by this DFA?
- What regular expression describes this language?

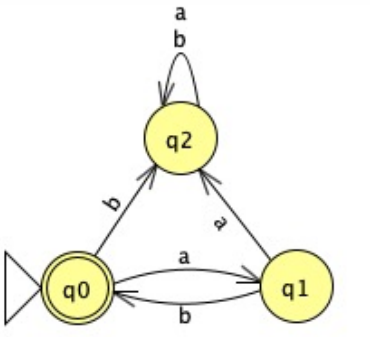


43

Final example

Answer

- Strings of a's and b's that have some number of repetitions of the sequence ab
- (ab)*



44

This would be all strings of as and bs that have some number of repetitions of the sequence ab. Which makes the regular expression (ab)* so much easier!

JFLAP examples:

- DFA examples

45

If you want to play with JFLAP, here's a link to the DFA examples we saw together.