DFA and Regular Languages

1. Definitions

A Deterministic Finite Automaton (*DFA*, DFAs for plural) is an abstract computational machine composed of the following elements:

- An *alphabet*: (finite) set of characters allowed.
- A (finite) set of *states* (composed of a single *starting state*, and at least one *accepting state*).
- *Transitions* between states.

A *word* is simply a finite chain/sequence of characters from the DFA's alphabet. A *word* can contain from 0 (empty word) to many number of characters from the alphabet (with or without repetitions).

A *substring* is a contiguous sequence of characters contained in a word.

Examples - Consider the following alphabet: {"a", "b"}.
Some possible words that we can form over this alphabet are: "abba", "ab", "b".
Some possible substrings of the word "babb" are: "ab", "bb", "a".

We say that a given word is *accepted* by a particular machine if and only if:

1. The word only contains characters from the alphabet, and
2. You land on an accepting state after performing all the required transitions.

Otherwise, we say that the word is *rejected* by the machine.

The (sometimes infinite) set of all words that are accepted by a particular machine form its *recognized language*. They usually follow a special kind of *property*, dictated by the particular machine.

DFAs are widely used in computer science for tasks that involve pattern recognition and sequence validation. They are employed in text processing tools like search engines and spell checkers to efficiently match patterns or validate strings.

2. Representation

There are many ways to represent DFAs. Let's take a look at the most common, and easily interpretable one: the diagram representation.

In the diagram representation, the states are drawn as circles:

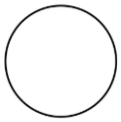

Figure 1: State in diagram representation

The starting state is a special kind of state where the validation logic of the machine will begin. Every DFA diagram must contain exactly one starting state. It is represented like a normal state, with an unannotated arrow pointing to it from outside:

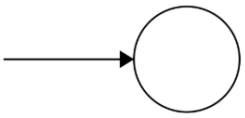

Figure 2: Starting state representation

A transition is represented by an arrow going from one state to another, annotated with the character of the alphabet on which the transition should happen:

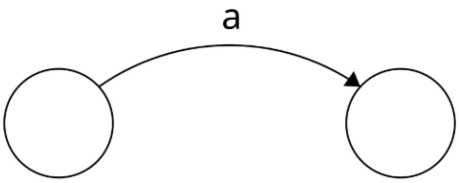

Figure 3: Transition representation

Sometimes an arrow is labeled with a comma-separated list of characters to indicate the same transition applies to each listed character, instead of drawing separate arrows.

An accepting state is another special kind of state. It is represented by drawing an additional circle inside an existing state. After performing each transition for every character of a given word, it is accepted only if you land on an accepting state. Every DFA diagram must have at least one accepting state.

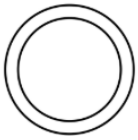

Figure 4: Accepting state representation

If you land on any other state after following every transition for a given word, that word is rejected by that machine.

Important note - Every state in a complete DFA must have exactly one outgoing transition for each character of the alphabet.

2.1. Example

The following figure is a fully complete diagram of a valid DFA. It works over the *alphabet* containing two letters: “a” and “b”. It has a *starting state* and a single *accepting state*. Every state has one *transition* for “a”, and one *transition* for “b”, meaning the diagram is valid and complete.

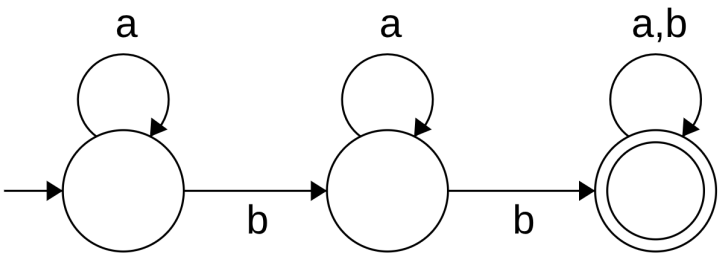

Figure 5: DFA for the language: words containing at least two “b”s

We can observe from the transitions that the accepted words must contain at least two “b”s. Therefore we say that it *recognizes* the language having the property that all (accepted) words must contain at least two “b”s.

2.1.1. Accepted words

We can check if a given DFA accepts or rejects a given word by following transitions for each character of the word: *consume* the input characters one by one and follow the corresponding transitions.

- Always begin at the starting state.
- Then take the first character of the word under test, and follow the arrow annotated with that character to land on the next state (possibly the same state).
- Repeat with the next character until you used every character of the word (in order).
- The final state you land on, after following every transition, is either an accepting state (accepted word) or not (rejected word).

Using the example DFA above (for words containing at least two “b”s):

- Example accepted words: “bab”, “abb”, “abaabbaaaaab”, “bb”, “abababaa”
- Example rejected words: “aba”, “aaaba”, “aaaaa”, “baa”, “a”, “b”, “baaaa”

The following figure shows a step-by-step detailed animation of state transitions for the words “aba” and “bab”.

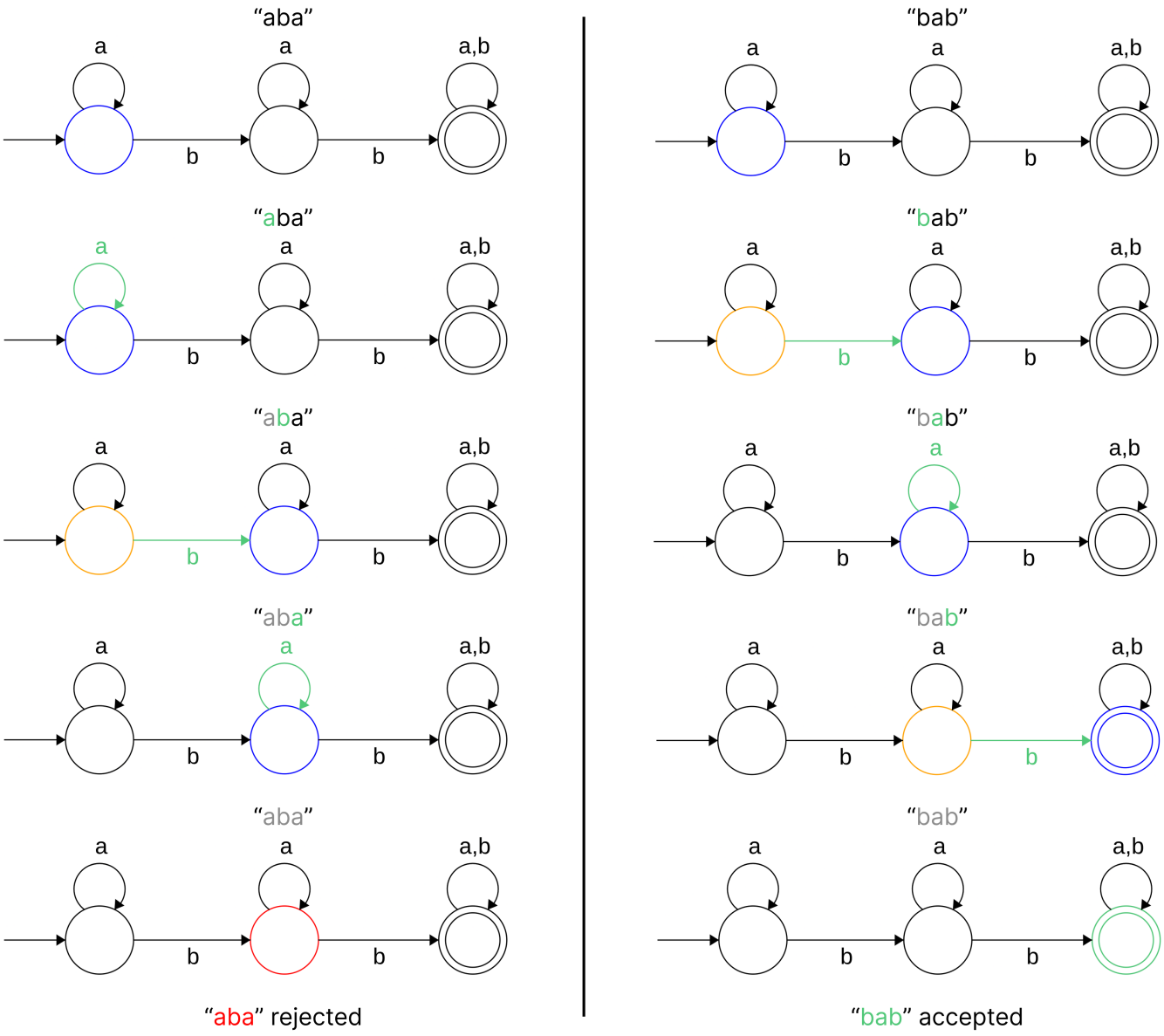

Figure 6: Transition animations for “aba” and “bab”

Note: a DFA can only process words made exclusively of characters from its alphabet (e.g. “acba” or “ab ba” are not valid words for the example DFA above).

3. From description to DFA

One usual problem type is where you are given a description of a language, and need to design a DFA that recognizes it.

Example: Design a DFA recognizing the language having the property that “a”s must be followed by at least one “b”.

Start by reasoning about states: each state encodes some information about what has been seen so far and what is required next. Decide whether the starting state should be accepting (i.e., whether the empty word should be accepted).

For the example, the empty word “”, does not contain any “a”: so in some way all of its “a”s (none) are followed by at least one “b”, so the empty word should be accepted and therefore the starting state is also an accepting state for that language.

Then you know you will have at least another state, as not every word should be accepted. That additional state could track when an “a” has been seen and a following “b” is required. When you consume an “a”, the machine should move to a state that enforces seeing a “b” next. Until a “b” is seen, words that leave the machine in that intermediate state will not be accepted.

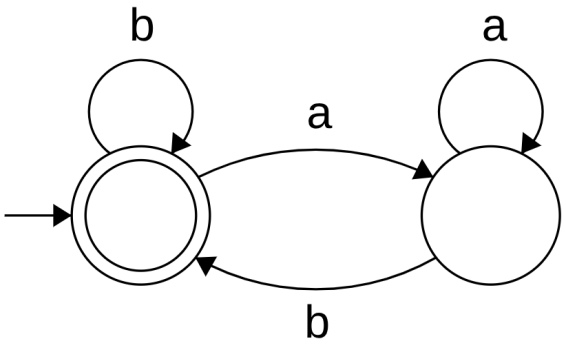

Figure 7: DFA for the language: “a”s must be followed by at least one “b”

Note - Not every language property can be expressed as a DFA. For example, there is no DFA that can recognize the language where every word has the same number of “a”s and “b”s. You can try to design one to convince yourself it is impossible (with a finite amount of state). A language is called *regular*, if it exists some DFA that recognizes it.

4. From DFA to description

Another usual problem type is the other way around: where you are given a diagram of a DFA, and need to interpret the property of the language it recognizes. That way is ofter harder to reason about.

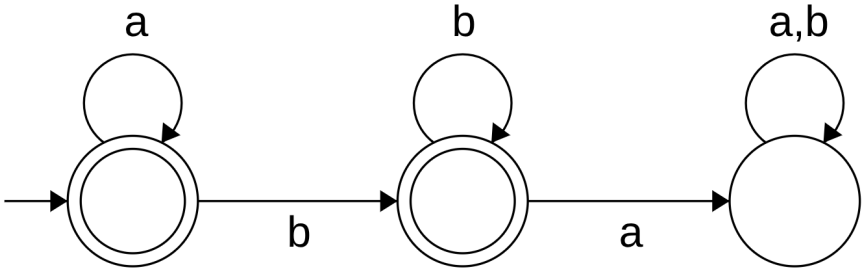

Figure 8: DFA for the language: word does not contain “ba” substring

A useful technique is to list some accepted and rejected words, look for patterns, and generalize the property shared by accepted words.

Another useful technique is state tracing: pick representative input strings and trace their path through the DFA, noting which states they end in. This helps identify the meaning of each state and the overall behavior of the automaton.