Finite Automata that Recognize Union and Intersection of Regular Languages

Some terminology. In computer science, any combination of symbols is called a *word*, and any set of words is called a *language*.

For example, dyrbulschur is a word, and the set {dyrbulschur, abrakadabra} is a language. By the way:

- the empty string is a word, and
- the empty set is a language.

We say that an automaton *recognizes* a language L if this automaton goes into a final state if and only if it reads a word from this language. For example, the very first automaton that we had in this class recognizes the language of all unsigned binary integers

$$L = \{0, 1, 00, 01, 10, 11, \ldots\}.$$

A language that can be recognized by an automaton is known as *regular*.

These definitions you have to remember.

Comment. In these terms, what we learned so far is that:

- the set of all unsigned integers is regular;
- the set of all integers – signed or unsigned – is regular;
- the set of fixed-point real numbers is regular.

Warning. If you have friends in College of Liberal Arts, do not tell them what we call a language, they will laugh you off.

Union and intersection: a brief reminder. Suppose that you have two sets A and B. Then:

- the *union* $A \cup B$ is the set of all the elements that belongs either to A or to B or to both;
- the *intersection* $A \cap B$ is the set of all the elements that belongs both to A and to B.

Example: take $A = \{0, 2, 4, 6, 8, 10, 12\}$ and $B = \{0, 3, 6, 9, 12\}$; then:

- their union is $A \cup B = \{0, 2, 3, 4, 6, 8, 9, 10, 12\}$ and
- their intersection is $A \cap B = \{0, 6, 12\}$.

Where did these notations come from. The sign for the union $\cup$ resembles the letter U, and this is exactly where it came from. Before the computers, books were printed by hand. So, to describe the word *word*, someone needed to put together four small metal stamps corresponding to the letters w, o, r, and d. If you need formulas, you need other stamps. To make printing easier, people use letters to describe operations.

For example, union is naturally described by the first letter of this word – letter U. To be more precise, when this notion appeared – at the end of 19 century – the international language of science was not English, it was German. If you are an American professor and you want to publish your result so that people read it, you had to publish it in German; OK, if it is a minor thing, you publish it in a language like English. In Russia, the main physics journal was published in German, not in Russian. It all changed in 1933, when Hitler came to power in Germany and threw many scientists out of the country; then the center of science moved to the US.

Good news is that the German word for union starts with the same letter U, so we can now naively think that this notation comes from the English word *union*.

What about the intersection? People were creative: the sign $\cap$ for the intersection is nothing else but an inverted letter U. So, to describe intersection, you do not need to design a new stamp: take the stamp for letter U and place it upside down.

How to recognize the union of two regular languages: idea. Suppose that we have two regular languages A and B. This means that:

- there is an automaton recognizing the first language; we will denote this automaton by the same letter A; and
- there is an automaton recognizing the second language; we will denote this automaton by the same letter B.

How can recognize the language $A \cup B$?

A natural idea is to have both automata work in parallel. At the end, after we read all the symbols from the word:

- if one of these automata is in a final state, we accept the word; we know that it belongs to the union $A \cup B$;
- if none of the automata is in a final state, this means that the word is not in the language A and not in the language B – so it is not in the union.

To describe the state of the computation at each moment of time, we need to know the states of both automata. So, each state of the overall configuration is a *pair* consisting of:

- the state of the first automaton and
- the state of the second automaton.

When we read a symbol, each component of the state is transformed according to the rules of the corresponding automaton.

First example: description of the two regular languages. Let us consider the following two regular languages. In both cases, we consider a simple alphabet consisting of only two letters: a and b.

First example: the first regular language. The first language A is the set of all the words that contain only the letter a, i.e.,

$$A = \{\Lambda, a, aa, aaa, \ldots\};$$

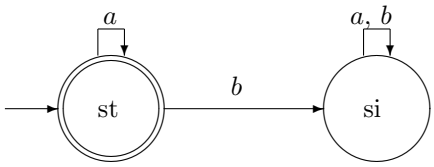
here, Λ (lambda capital) denotes an empty string. In terms of grade, this corresponds to a straight-A student.

This language can be recognized by the following automaton with two states:

- the start state st, meaning that so far we only had a's, and
- the sink state si, meaning that we have already seen some b's.

Here:

- In the beginning, we have an empty string, with no b's, so we are in the state st.
- If we are in the state st, and read a letter a, we stay in this state.
- If we are in the start state and we read a letter b, we go to the sink state.
- If we are in the sink state, then, no matter what we read, we stay in the sink state.



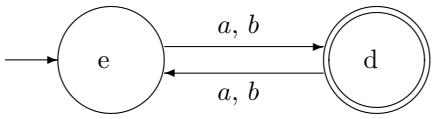
First example: the second regular language. As B, we can take the language of all the words that have odd number of symbols:

$$B = \{a, b, aaa, aab, aba, abb, baa, bab, bba, bbb, aaaaa, \ldots\}.$$

This language can be recognized by an automaton that has two states: even (e) and odd (d).

- In the beginning, we have an empty string, with 0 symbols. Zero is an even number, so we are in the state e.
- If we are in the even state, and we read one more symbol, the number of symbols becomes odd, so we go to the state d.
- If we are in the odd state, and we read one more symbol, the number of symbols becomes even, so we go to the state e.

The resulting automaton looks like this:



Comment. It is important to notice that while:

- in the past, we usually rejected a word because we went into a sink state,
- here is an automaton that does not have sink states at all, and still rejects words.

For example, the word aa will be rejected by this automaton, because we end up in a state e which is not final.

First example: how to describe the union. The states of the union are pairs of the states of two automata.

- The first automaton can be in two states: st and si.
- The second automaton can be in two states: e and d.

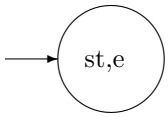
Thus, the configuration of these two automata can be in $2 \cdot 2 = 4$ possible states:

- when the first automaton is in the state st, we have states (st,e) and (st,d);
- when the first automaton is in the state si, we have states (si,e) and (si, d).

In the beginning:

- the first automaton is in the state st, and
- the second automaton is in the state e,

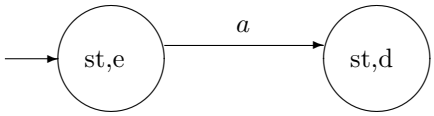
so the whole configuration is in the state (st,e).



When, in this state, we see the letter a, then:

- the first automaton remains in the state st, while
- the second automaton goes to state d.

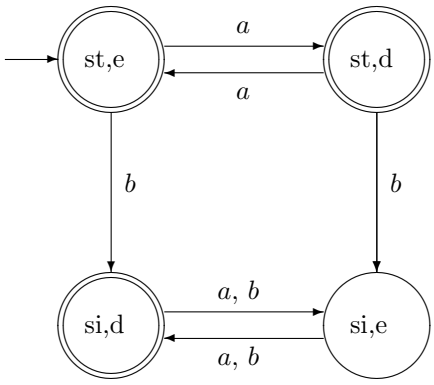
Thus, the whole configuration changes to (st,d).



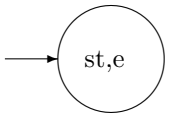
Similarly, we can describe all other transitions. At the end:

- if one of the two automata is in the final state, this means that the word is accepted by one of the automata, so this word is in the union of the two languages; thus, the states (st,e), (st,d), and (si,d) – for which at least one of the components is final – are final states of the resulting configuration;
- if none of the two automata is in the final state, the word is rejected; so, the state (si,e) is not final.

Thus, we arrive at the following automaton for recognizing the union of the two languages:



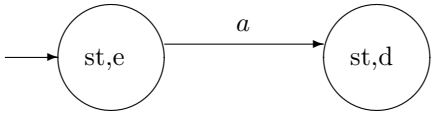
Just in case: the same procedure step by step. We start when both automata A and B are in their start states. The first automaton is in the state st, the second automaton is in the state e, so the whole configuration is in the state (st,e):



In this state, we see the letter a or the letter b. If we see the letter a, then:

- the first automaton remains in the state st, while
- the second automaton goes to state d.

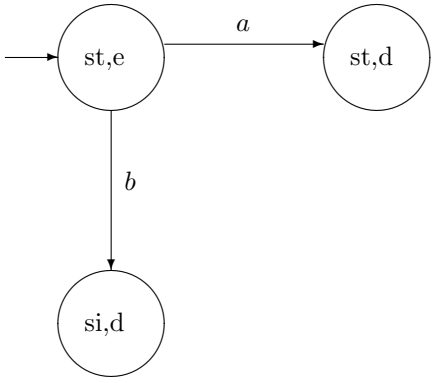
Thus, the whole configuration changes to (st,d).



If, in the state (st,e), we see the letter b, then:

- the first automaton changes to the state si, while
- the second automaton goes to state d.

Thus, the whole configuration changes to (si,d).

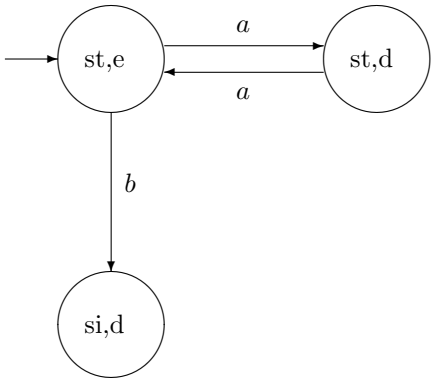


We considered all possible transitions when we are in the state (st,e). Let us now consider what will happen if we are in the next state that we considered: the state (st,d). We allow only two letters: a and b.

If, in the state (st,d), we read the symbol a, then:

- the first automaton remains in the state st, and
- the second automaton changes to the state e.

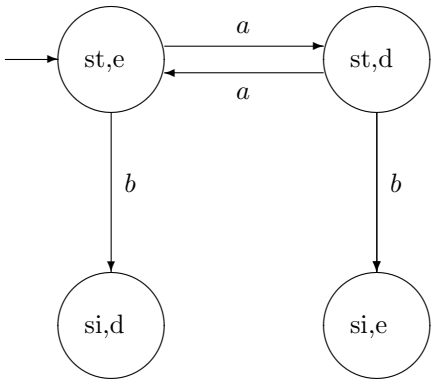
Thus, the configuration of the two automata changes to the state (st,e).



If, in the state (st,d), we read the symbol b, then:

- the first automaton changes to the state si, and
- the second automaton changes to the state e.

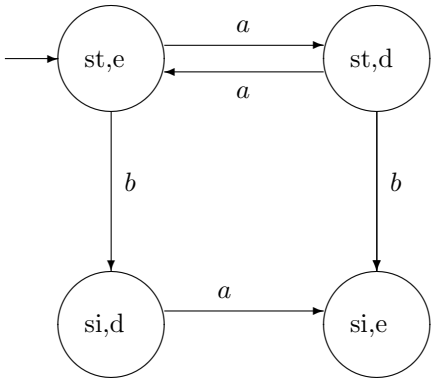
Thus, the configuration of the two automata changes to the state (si,e).



Let us now consider the state (si,d). If, in this state, we read the symbol a, then:

- the first automaton remains in the state si, and
- the second automaton changes to the state e.

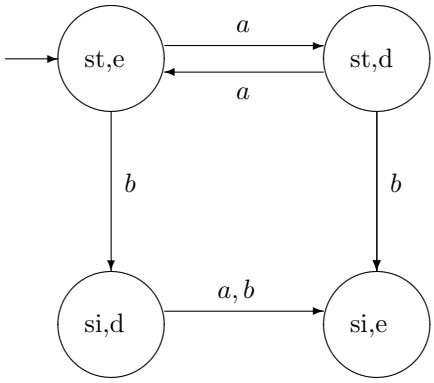
Thus, the configuration of the two automata changes to the state (si,e).



If, in the state (si,d), we read the symbol b, then:

- the first automaton remains in the state si, and
- the second automaton changes to the state e.

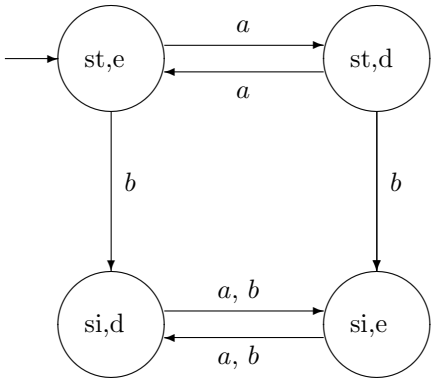
Thus, the configuration of the two automata changes to the state (si,e).



The only state from which we did not consider transitions is the state (si,e). In this state, whether we read a symbol a or a symbol b, the transition is the same:

- the first automaton remains in the state si, and
- the second automaton changes to the state e.

Thus, the configuration of the two automata changes to the state (si,d).



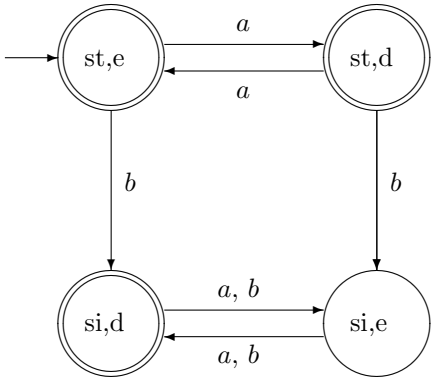
We are done with all the transitions: for each of the four states and each of the two symbols, we described where the automaton will go. All that remains is to decide which of the four states are final. In general, in a state (s,s′):

- if at least one of the states s and s′ is final for the corresponding automaton, then the state (s,s′) is final;
- if none of the two states s and s′ are final for the corresponding automaton, then the state (s,s′) is not final.

In our example:

- in the state (st,e) of the configuration, the first state st is the final state of Automaton A; thus, the state (st,e) is a final state of the configuration;
- in the state (st,d) of the configuration, the first state st is the final state of Automaton A and the second state d is the final state of the Automaton B; thus, the state (st,d) is a final state of the configuration;
- in the state (si,d) of the configuration, the second state d is the final state of Automaton B; thus, the state (si,d) is a final state of the configuration;
- in the state (si,e) of the configuration, none of the two states si and e is final; thus, the state (si,e) is not a final state of the configuration.

So, we arrive at the following description of the automaton for recognizing the union of the two languages:



First important comment: what if different automata use the same notations for their states? In some cases, the descriptions of Automata A and B may use the same letters. For example:

- Automaton A can use states s and e, and
- Automaton B can use states s, e, and i.

Then, the automaton for recognizing the union $A \cup B$ of the two languages will start in the state (s,s), where:

- the Automaton A is in the state that for this automaton was denoted by the letter s, and
- the Automaton B is in the state that for Automaton B was denoted by the letter s.

As we read symbols, we may get into all possible pairs of states:

- pairs (s,s), (s,e), and (s,i) in which Automaton A is in the state s, and
- pairs (e,s), (e,e), and (e,i) in which Automaton A is in the state e.

Second important comment: what if some pairs of states are not accessible? Sometimes, some pair of states cannot be reached no matter what word we read. In this case:

- it is OK to include this pair in the description of the union-recognizing automaton, and

- it is also OK *not* to include this pair in the description of the union-recognizing automaton.

Whether you include this pair of states or not, it will not affect the important thing:

- which words are recognized by the automaton, and
- which words are not recognized (i.e., rejected) by the automaton.

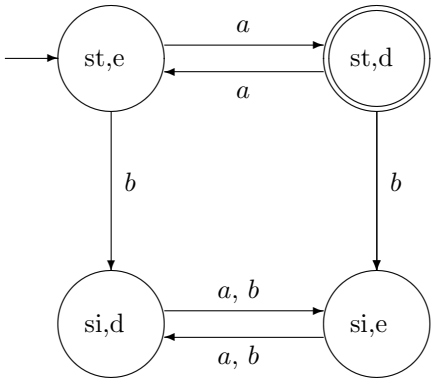
What about the intersection? For detecting intersection, transitions are the same, the difference is in when the word is accepted, i.e., which states are final:

- a word belongs to the union if it belongs to at least one of the languages; thus, the state consisting of two components is final if at least one of these components is final;
- in contrast, a word belongs to the intersection if it belongs to both languages; thus, the state consisting of two components is final if both its components are final.

For example, the word *bbb* is accepted by B but not by A, so:

- this word belongs to the union of the two languages, but
- it does not belong to their intersection.

In this example, only the state (st,d) is final:



Comment. By definition, each state which is final for the intersection is also final for the union.

How can we describe both union and intersection by one picture. The only difference between union and intersection is in which states are final:

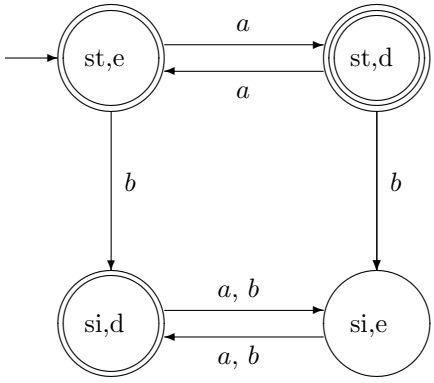
- all states which are final for intersection and also final for the union, but
- some states final for the union may not be final for the intersection.

So, to describe intersection on the picture of the union automaton, it is sufficient to describe which of the final-for-union states are also final for the intersection. This can be done by marking such states, e.g., by a triple circle.

This makes sense:

- in general, we emphasize the final state by using a double circle,
- so, it is reasonable to emphasize the states which are final both for the union and for the intersection by using triple circles.

In particular, in the above example, we get the following picture:

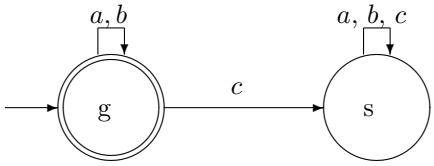


In this example:

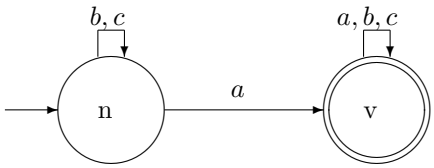
- the state (st,d) is denoted by a triple circle, meaning that it is final for the intersection, while
- the states (st,e) and (si,d) are denoted by a double circle, meaning that these states are final for the union, but not for the intersection.

Second example. Let us consider an alphabet consisting of three letters a, b, and c. Let us consider the following two languages.

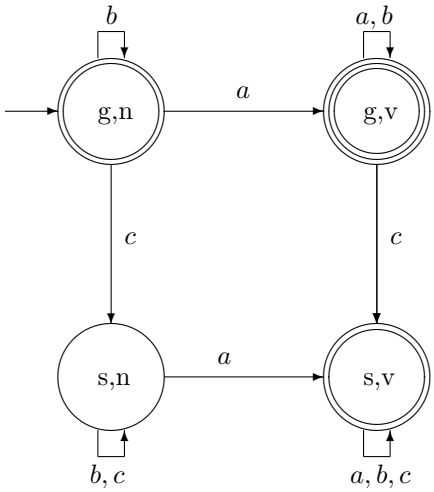
Second example: the first regular language. The first language A comes from interpreting these letters as grade; the language A corresponds to all sequences that have no c's. This language can be recognized by an automaton that has two states: *good* (g) and *satisfactory* (s). Transitions between these two states are as follows:



Second example: the second regular language. The second language B comes from the linguistic interpretation: it is the set of all the words that contain vowels – i.e., in this case, the vowel a. This language can be recognized by the following automaton, with two states: *vowel* (v) and *no vowel* (n):



Second example: union and intersection. The union and intersection of these two languages are described by the following automaton:



Practice. Draw two automata and describe their union and intersection.