Simple Examples of Finite Automata

What we had so far: automaton for recognizing unsigned binary integers. So far, we had an automaton that recognizes unsigned binary integers provided that the only other possible symbol is letter $a$.

![Automaton for recognizing unsigned binary integers](image)

Automaton for recognizing general unsigned integers. The main difference is that now:

- in addition to digits 0 and 1, we can have 2, 3, 4, ..., 9, and
- in addition to the symbol $a$, we can have all other symbols on the keyboard: $a$, $b$, $c$, etc.

Transitions are the same, so all we need is add the new symbols to the arrows describing these transitions:

![Automaton for recognizing general unsigned integers](image)
What about general integers – which may be signed? In general, we can also have signed integers, e.g., \(-19\) or \(+26\). In this case, the first symbol can be a sign.

Once we read a sign, this is not an integer yet, the next symbol should be a digit. We can thus say that we are in a special state $s_i$. In this state:

- if we see a digit, i.e., $0, \ldots, 9$, then we know what we have so far is an integer, so we go to state $i$;

- on the other hand, if the next symbol is not a digit, then we know that we have an error, so we go to the error state $e$.

The resulting automaton has the following form:

Comment. If we end up in a state which is not a final state, the word is rejected – in this case, the state is not an unsigned integer. Please note the following:

- In many previous examples, when the state was rejected, it was because we reached a sink state.

- However, here, if all we type in is the sign $+$, then we end up in a state $s_i$. If this is the only symbol we type in, then the word $+$ is rejected, because $s_i$ is not a final state. But, as you see, $s_i$ is not a sink state.

First example. Let is trace how the integer 90 will be recognized. In the beginning, we are in the start state:
Then, we read digit 9: 90. What we have read is already an integer, so we are in state $i$.

Now, we read 0: 90. We are still in the integer state:
We have read all the digits, we are in the final state, so the word 90 is recognized as an integer.

**Second example.** Let us trace how the integer −19 will be recognized. In the beginning, we are in the start state:

Then, we read the sign −19 and go into the sign state:
Then, we read the digit 1: $-19$. What we have read so far, i.e., number $-1$, is already an integer, so we are in the state $i$.

Finally, we read the digit 9: $-19$, and are still in the state $i$: 
We have read all the symbols, we end up in the state $i$, which is a final state, so we conclude that what we have read is an integer.

**Fixed-point real numbers: first approximation.** Let us now consider fixed-point real numbers, i.e., numbers of the type 1.35, $-1.35$, +1.35, 1., $-1.$, +1., etc. Java also allows to have numbers like .5, $-$.5, +.5, but, for simplicity, let us first ignore this possibility, and consider only the cases when we have digits before the decimal point. Such numbers can be described as follows:

- first, we have an integer; e.g., for $-1.35$, this integer is $-1$;
- then, we have a decimal point;
- finally, we have either nothing else or an unsigned integer; in the above example, this unsigned integer is 35.

How can we describe this as a finite automaton?

- First, we follow all the steps of the above automaton for recognizing integers.
- However, the state $i$ is no longer the final state, since $-1$ is not a correct real number in Java.
- Instead, we have to see a period. Once we see a period, it is already a real number. We can therefore say that we are in a state $r$.
- After the period, we can only have digits. If we have anything else, it is an error.

In terms of an automaton, this can be described as follows:
Example. Let us show how the number $-1.35$ will be recognized. We start in the start state:

Then, we read the minus sign $-1.35$, and go into the state $si$: 
After that, we read the digit 1: \(-1.35\), and get into the state \(i\):

Then, we read a period, and get into the state \(r\):
After that, we read digit 3 and stay in \( r \):

Finally, we read the digit 5: \(-1.35\)
We read all the symbols, we are in the final state $r$, so the word $-1.35$ is a fixed-point real number.

*Comment.* If you see this graph in the very beginning, it sounds complicated, but as built it step by step, hopefully, it does not sound that difficult.

*Practice.* Check how you can trace whether different words are accepted or rejected by these two automata. Try your hand on designing automata for other cases:

- for all unsigned integers – taking into account that we can have numbers of the type $-0.35$, with no digits before the decimal points;
- for all real numbers, including floating point Java numbers like $-1.35e-6$.
- for names of variables: in the usual Java arrangement, such names must start with a small letter and be followed by letters, digits, or an underscore symbol_.

Some such questions may be assigned as a homework, some other questions may be on the test.