

Comparing regular expressions and finite state machines

JANUARY 1, 2022 – BOB

This is the last in a series about regular expressions and finite state machines

- Regular expressions
- Finite state machines
- Comparing regular expressions and finite state machines

In this article I won't be introduce any big new things, but instead I'll be showing how two things I've already discussed in the previous two articles relate to each other. So if you're unsure about regular expressions or (deterministic) finite state machines (FSMs), please look at the previous articles before you read this one as I'll assume you're OK with them.



Regular expressions and finite state machines can be two sides of the same coin. [Image credit](#).

Two sides of one coin

You may have noticed similarities between regular expressions and FSMs as you read the previous two articles. They both look at a series of inputs, and at the end say whether that series was good or not.

If you don't use capture groups, then regular expressions and FSMs are exactly equivalent. When I was taught about them it was described as: regular expressions and FSMs are *isomorphic*. I think that this was because the lecturer had a Maths degree, and *isomorphism* is a Maths term for a two-way mapping or correspondence between two things.

Capture groups aren't allowed in the more pure Maths version of regular expressions, even though they are allowed in the version that's used in common programming languages. They require some kind of memory beyond the memory needed for keeping track of what has matched what, and so aren't like other features of regular expressions. If you remember in the article on FSMs, I stressed the fact that FSMs don't remember their past; they just know where they are now and what transitions are possible for which inputs.

Why does it matter?

The first reason I can think of is that it's just a bit cool, in a geeky kind of way. It's like knowing about Bruce Wayne and about Batman, and then learning that they're the same person. The two concepts that previously lived separately in your head kind of become two aspects of one concept.

The second reason is that it could be useful. An advantage of regular expressions compared to FSMs is that they take up hardly any space in source code, partly because you usually get help from libraries etc. when you want to process them. FSMs have the advantage of being able to be teased apart and the parts given helpful names etc. more than with regular expressions, and you can easily set up code to be executed when different bits of the input have been matched (i.e. when you enter different states).

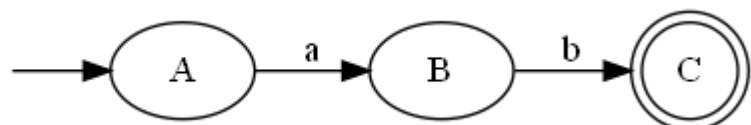
So (providing you don't use capture groups) you can translate a regular expression to an equivalent FSM or vice versa. Why you might want to do this will vary by your circumstance. Maybe the inputs used to arrive in one go e.g. as many characters in a string, and now arrive in several chunks e.g. messages from a queue. A regular expression would work well for the first context, but not for the second – a FSM would be better. Or maybe you had a regular expression and need to have code execute during the processing of the inputs – this is something that a FSM would let you do.

Or there's a regular expression that you just can't understand properly, but drawn out on paper as a FSM would be clearer. A FSM that used to need processing along the way but now doesn't might be able to be converted to a regular expression. Or maybe you want to switch between two or more valid inputs – switching between regular expressions is just a matter of switching between strings, whereas switching between FSMs involves switching between potentially complicated dictionaries.

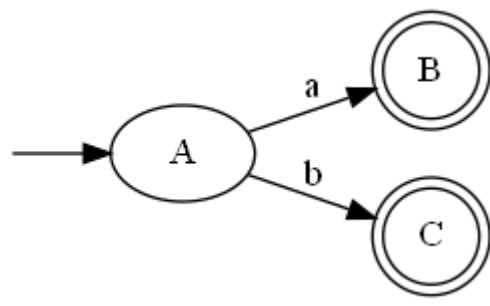
Going through the operators

To show you that regular expressions and FSMs are equivalent, I'll go through most of the elements that can make up a regular expression, and show how you could implement the same behaviour in a FSM.

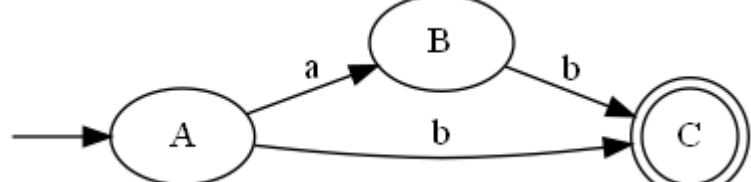
/ab/



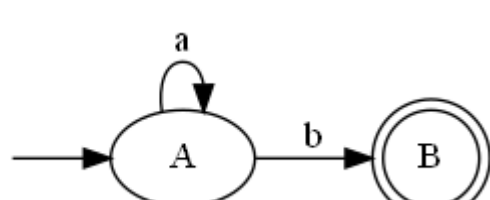
/a|b/



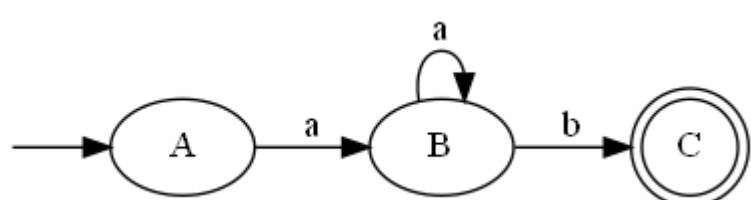
/a?b/



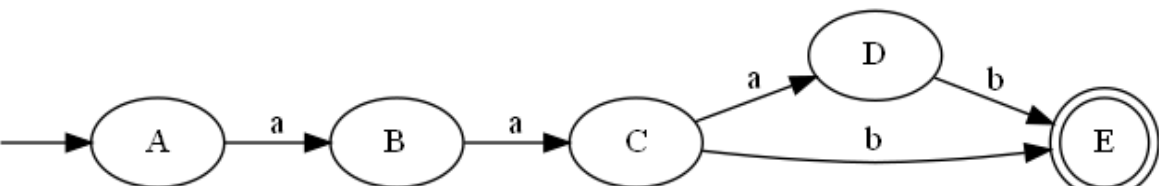
/a*b/



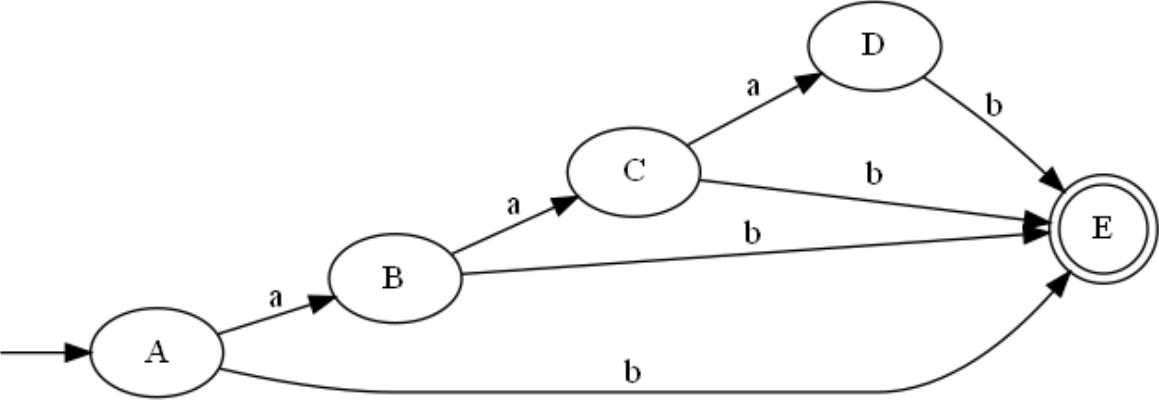
/a+b/



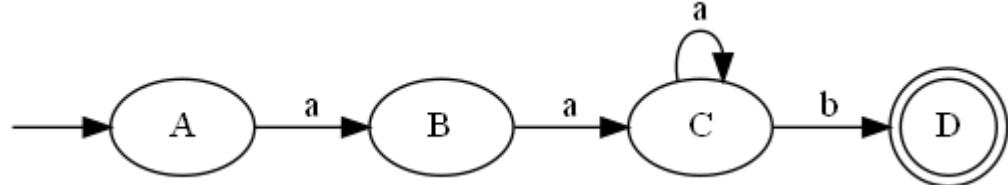
/a{2,3}b/



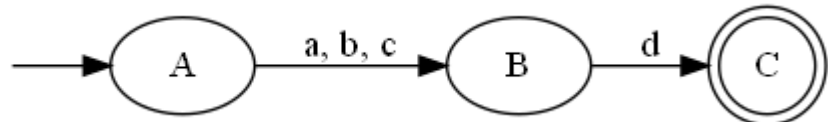
/a{3}b/



/a{2,b}/



/[abc]d/



Summing up

(Deterministic) finite state machines and regular expressions (without capture groups) do basically the same job. So you can usually translate one to the other. This means you could use whichever form makes more sense for the job in hand, a bit like [translating between cartesian and polar co-ordinates when working with complex numbers](#).

Share this:



Loading...

Related

Finite state machines
December 31, 2021
in "C#"


Regular expressions
December 30, 2021
in "Computer theory"

Fuzzy matching – example algorithms
April 22, 2023
in "Computer theory"

← PREVIOUS
Finite state machines

NEXT →
Exceptional ? Basics

2 thoughts on "Comparing regular expressions and finite state machines"

 lewiscowles
MARCH 2, 2022 AT 8:32 AM

Hi Bob,

Just catching up on the backlog now. What did you use to generate the graphics for the FSMs?

Looks like GraphViz, or maybe some wrapper around that, like plantuml.

Hope the new year is treating you well!
Best,
Lewis

★ Like

 Bob
MARCH 2, 2022 AT 9:21 AM




Hi Lewis, it's vanilla GraphViz. Things are generally good – thank you, and I hope they are with you too.

★ Like

Leave a comment

Write a comment...

Log in or provide your name and email to leave a comment.



Email (Address never made public)

Name

Website (Optional)

Email me new posts

InstantlyDailyWeekly

Email me new comments

Save my name, email, and website in this browser for the next time I comment.

Comment

Want to be told when there's new stuff?

If you'd like an email telling you when I've posted new stuff, go to the [sign-up page](#).

