Cursēd Regular Expressions

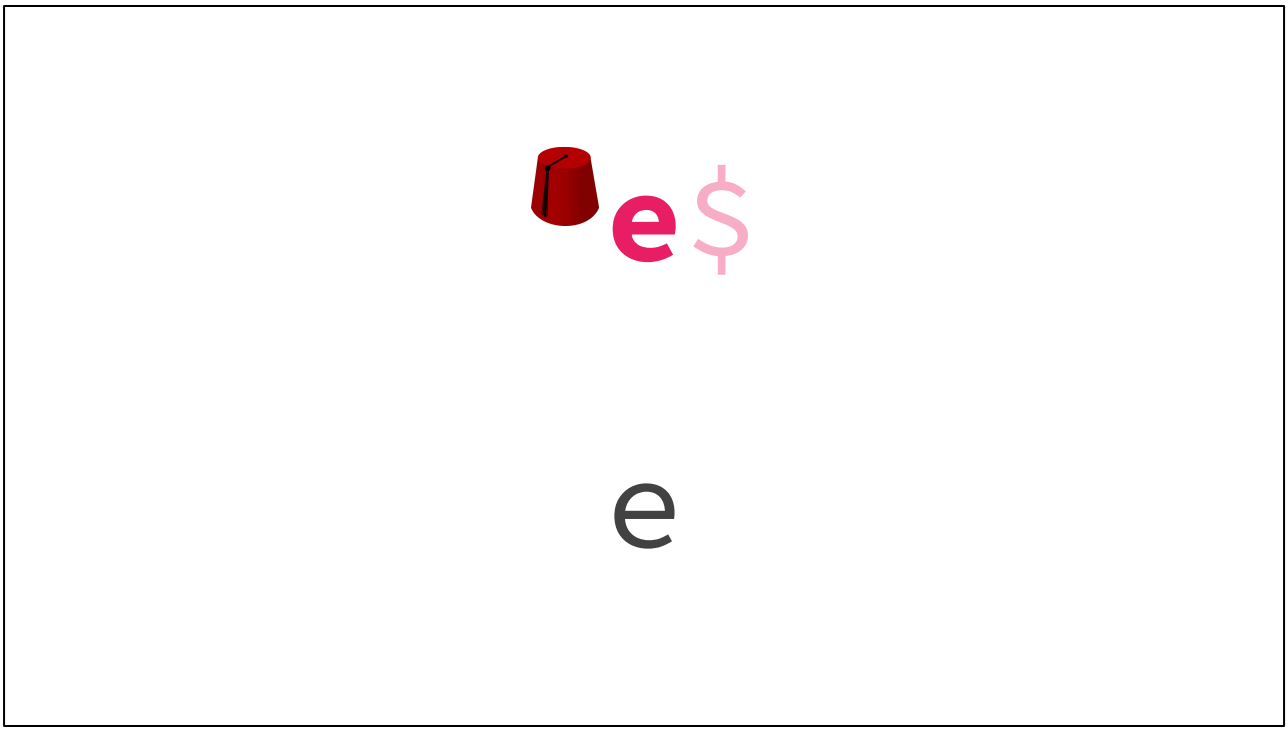
@LunaSorcery

Hi everyone, I'm going to talk about some Cursed Regular Expressions.
I know the computer scientists among you are just thinking "that's a tautology" but I assure you it gets worse.

Regular Expressions?

a notation for describing patterns of text, as text

So what are regular expressions? They're a text notation for representing patterns of text.
They're used a lot in computing to verify user input and to extract data from large datasets.
That's pretty non-explanatory so I'll show you what I mean.



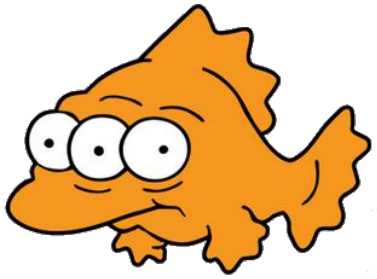
So the regular expression is here at the top in pink, and the text at the bottom is the text it matches.
The letter ‘e’ matches against itself, the circumflex and dollar are special characters which match the beginning and end of a text string.
Without them this would match anything that even contains the letter e, with them we only match *exactly* the letter e.
And here’s how it looks with realhats [CLICK, pause for laughter]

^(b|c)at$

bat

We can do more interesting matching - we can use the vertical pipe operator to represent an ‘or’ operation.
The section in parentheses will match against either the letter b or c.
So this expression will match against a [CLICK] ‘bat’, and also a [CLICK] ‘cat’

^fi*sh$



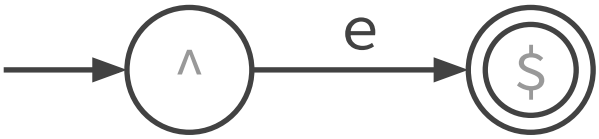
We can also match varying counts of letters with the asterisk, and that'll match any number of the letter immediately preceding it.
So this can match against a [CLICK] 'fish', but it can also match a fish with [CLICK] three eyes, and also a [CLICK] fish with no eyes.

The syntax for regular expressions has a ton more features I don't have time to go into, but that's all you need to know for the rest of the talk.

Regular Expressions are Finite State Machines

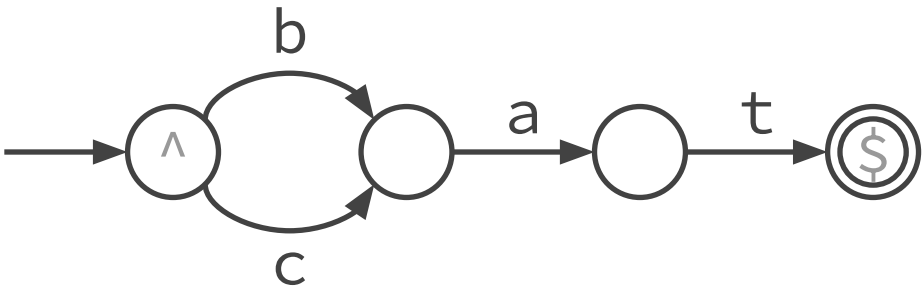
But here's the cool thing. These are just finite state machines!
Let's have a look at those same examples again.

^e$



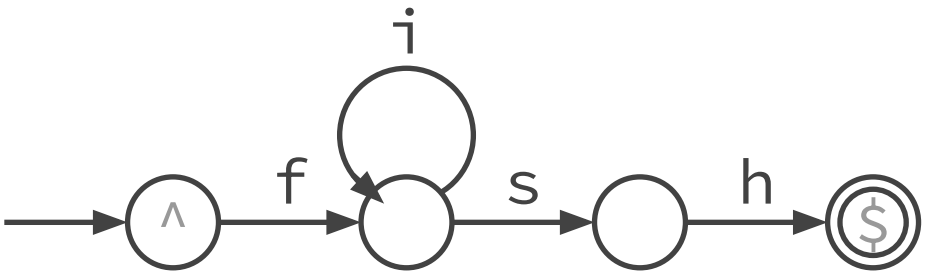
We have a starting state and an end state, and a pathway between the two.
We start in one state and take an input of E to transition to the other state.
Nice and simple.

^(b|c)at$



In this case, we start on the left, we can take an input of either B or C, and then we have to take inputs A and T to get to the win state.
Everyone on board?

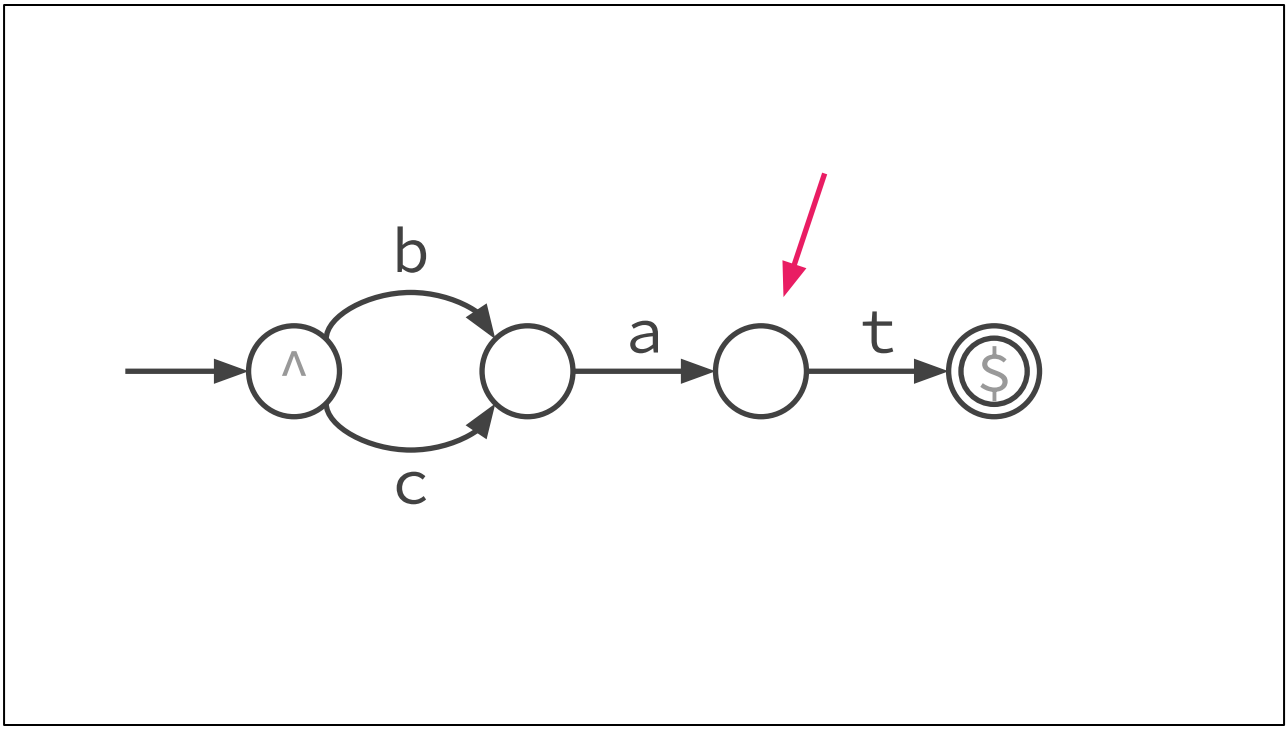
^fi*sh$



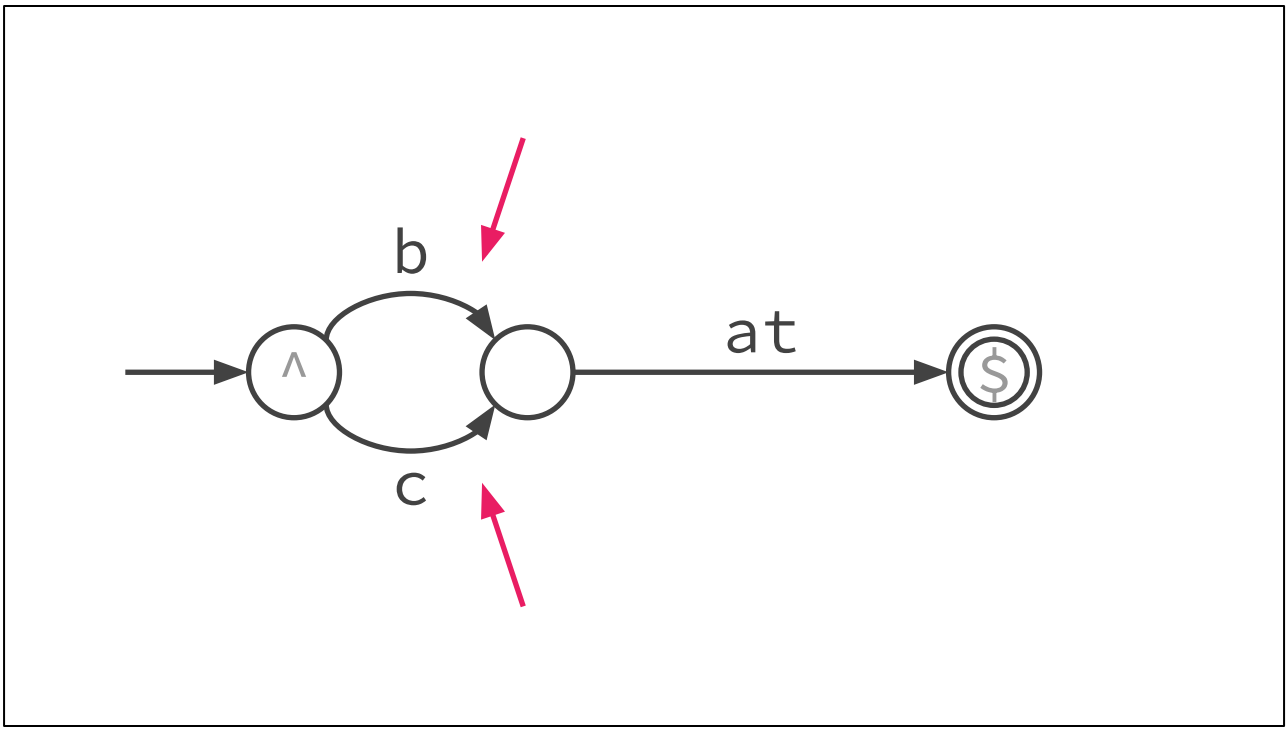
And then hopefully this also makes sense, the star syntax is equivalent to a state that loops back on itself like this.

This works both ways

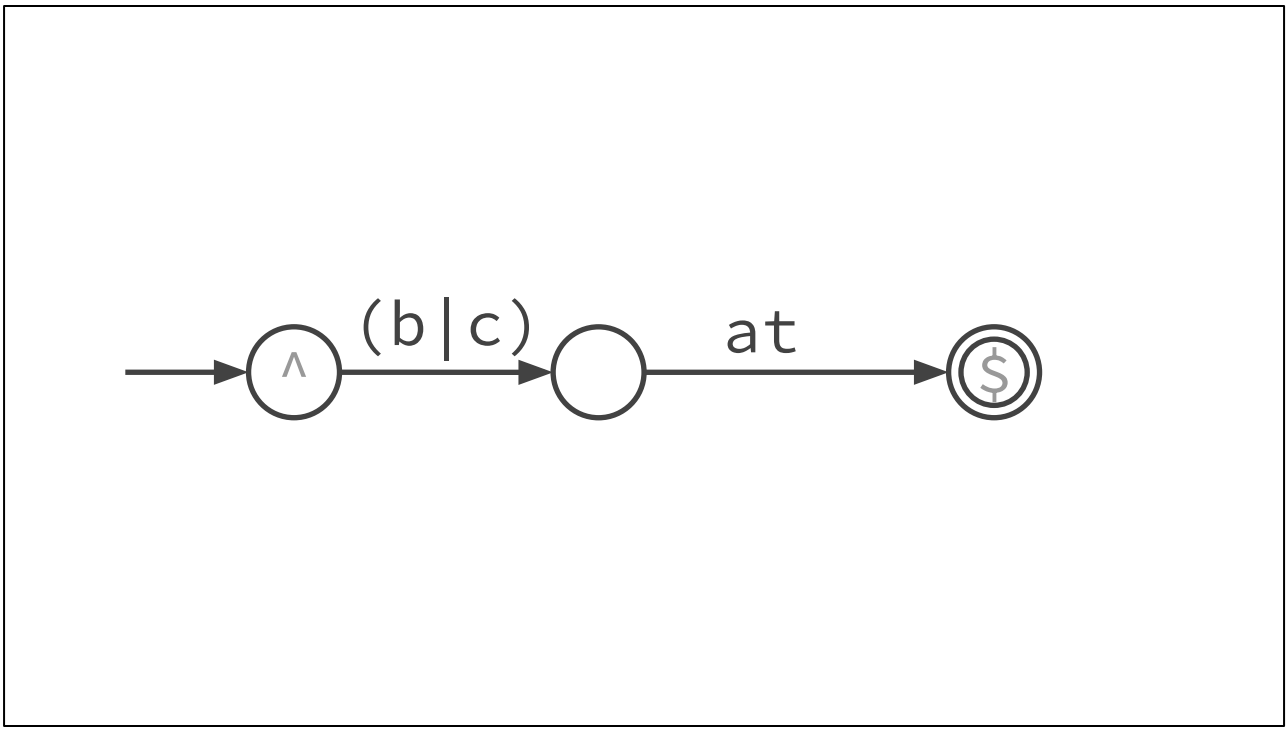
And this works both ways - so we can start at a finite state machine and turn it into a regular expression, that will only match against a series of inputs that takes you from the start state to the end state. Let’s have a look at how to do that.



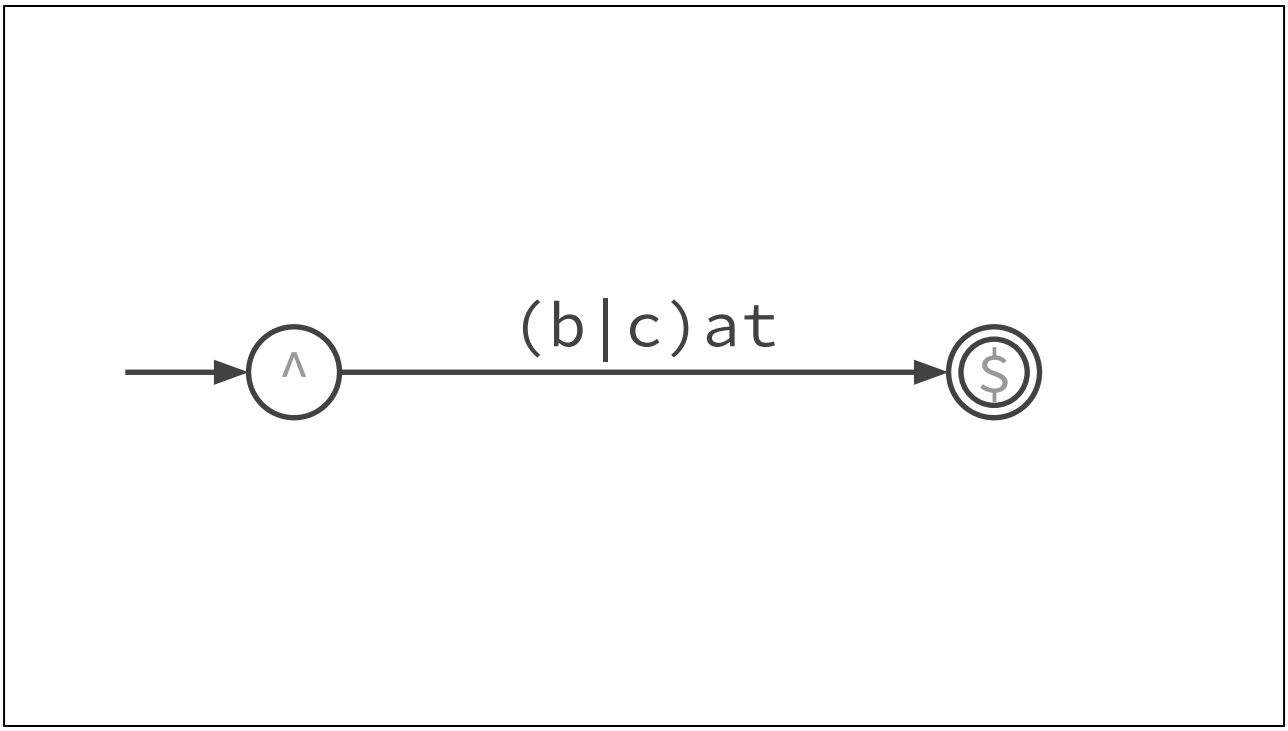
So the trick to this is to reduce the state machine into a simpler equivalent one, by progressively collapsing states and pathways into equivalent constructs.
If we look at this node [CLICK], it's only got one way in and one way out, so we can collapse it into a single pathway...



...like this.
Now let's look at these pathways [CLICK], notice how they both start and end at the same state? So we can collapse them into a single pathway with an 'or' statement on it…

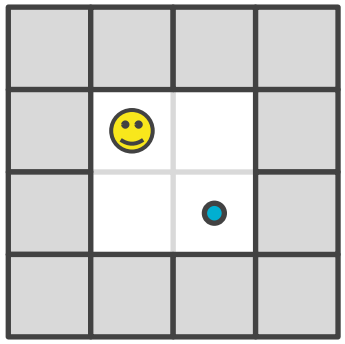


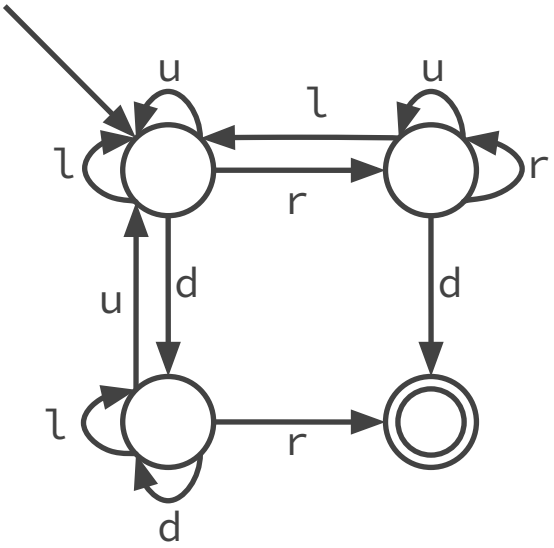
... like this.
And now as before we can drop that middle state...



...and we get our completed expression

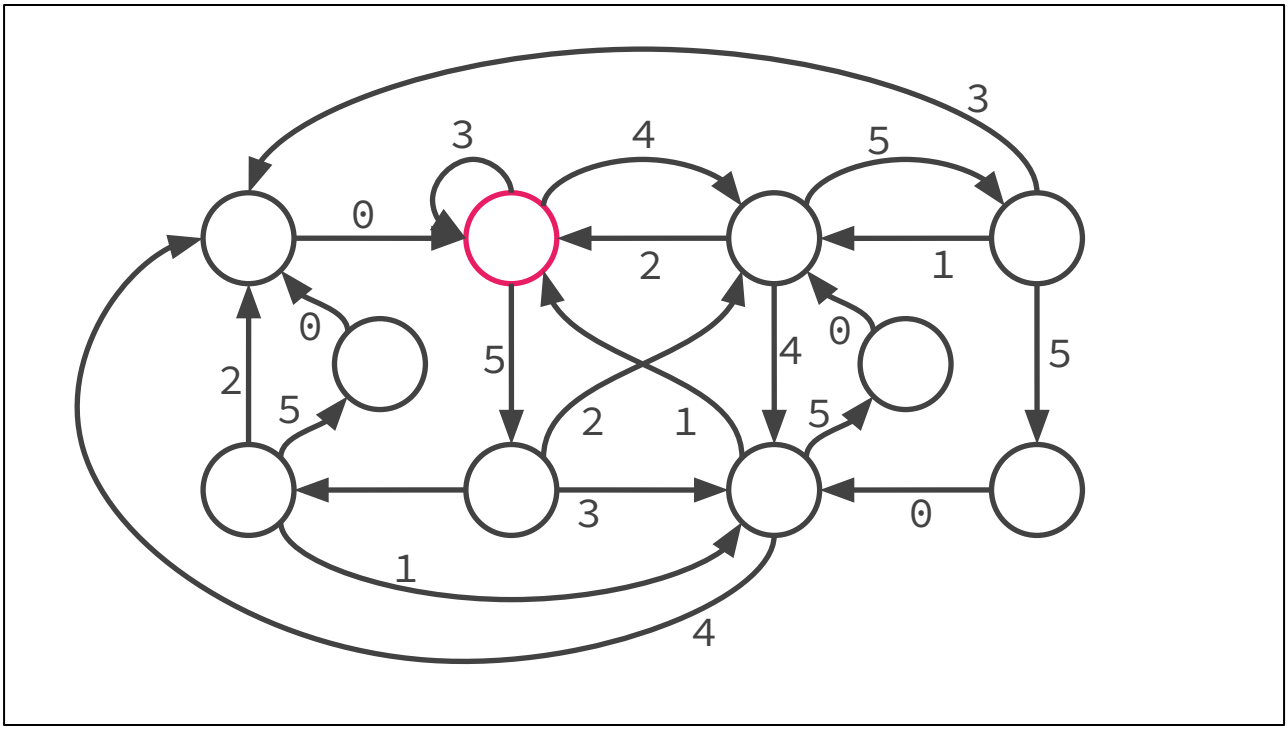
Alright, now let’s have some fun.





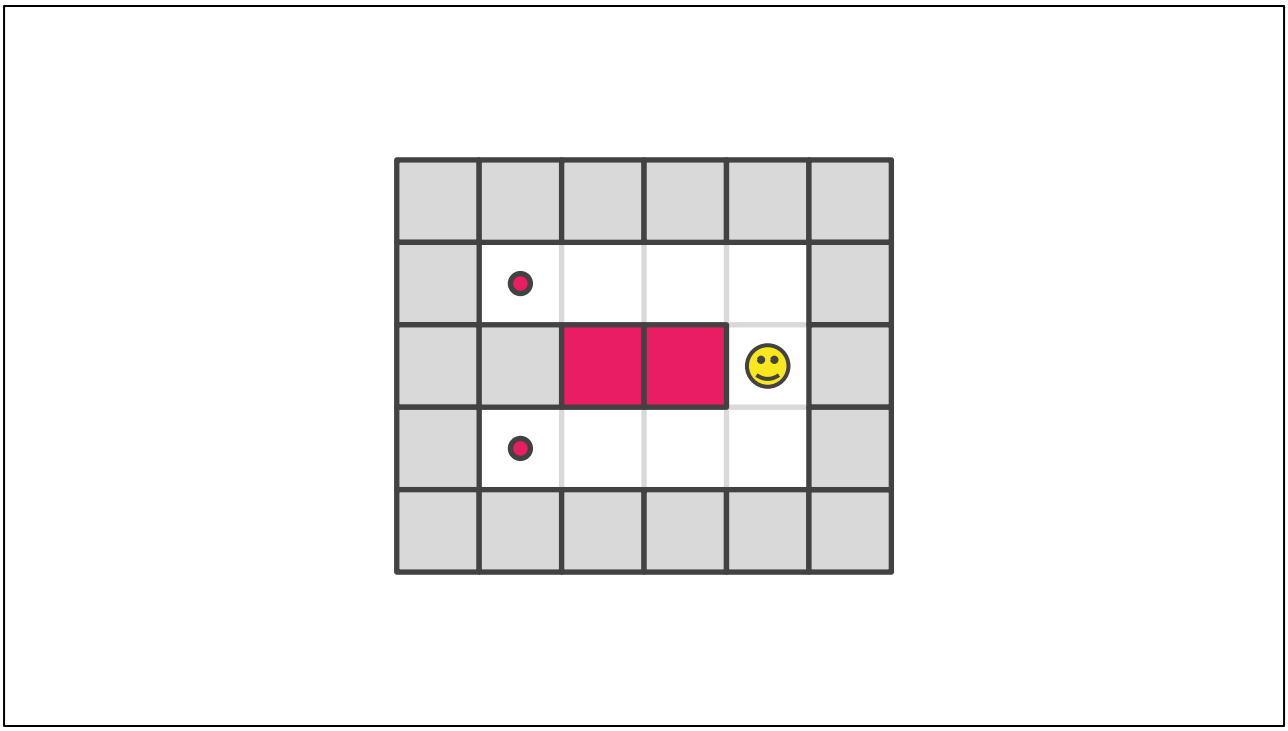

```
^(u|l|r(u|r)*l|d(d|l)*u)*(r(u|r)*d|d(d|l)*r)$
```

So here's a simple game level, we just want to get the smiley face to the blue dot, and we can move up, down, left and right.
[CLICK] Here's what the state transition diagram looks like, already a little unwieldy, especially since you can walk into walls and go nowhere.
[CLICK] and here's the regular expression which is already a bit of a mess at 45 characters long.
Incidentally most modern programming languages will have native support for regexes, so you could quite easily paste that in and use it to verify that a given series of inputs will solve the game.

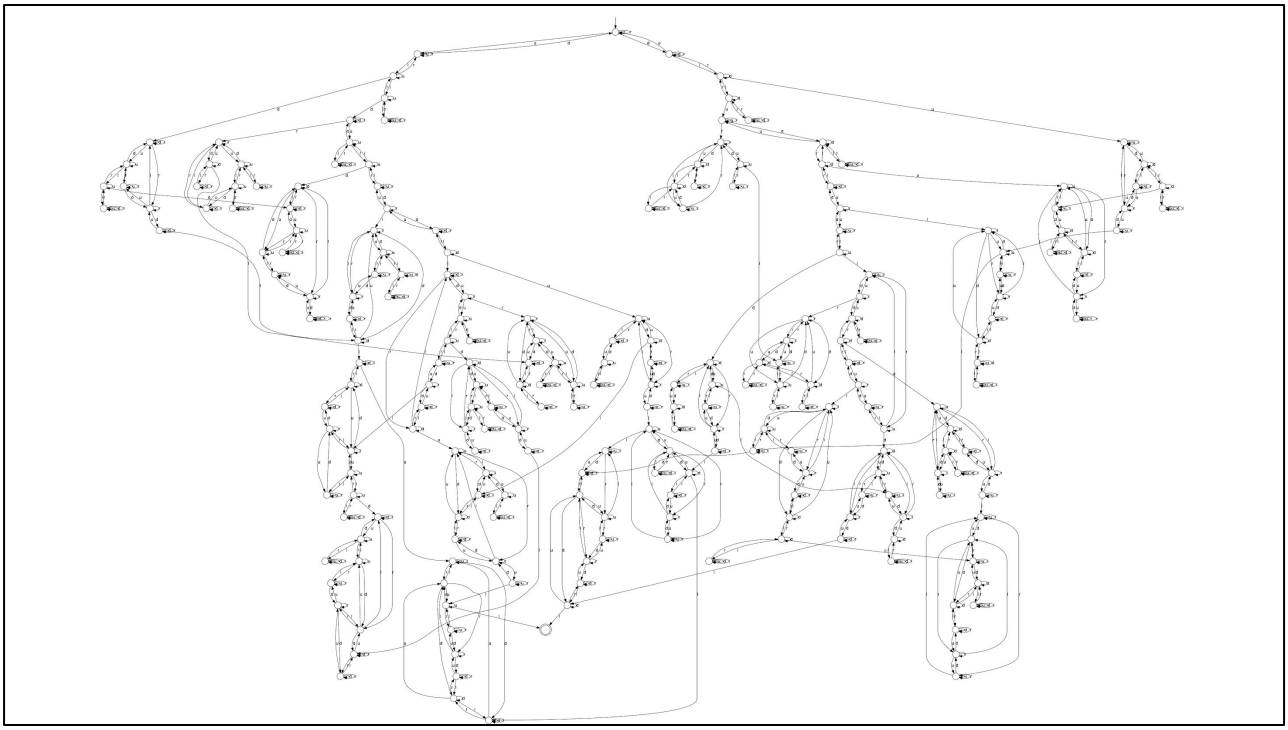


Here's a graph I nicked from Colin - I don't really understand how or why it works, but if you start at any node on here and wander around until you make a closed loop, you'll end up with a 3-ball juggling pattern in siteswap notation.

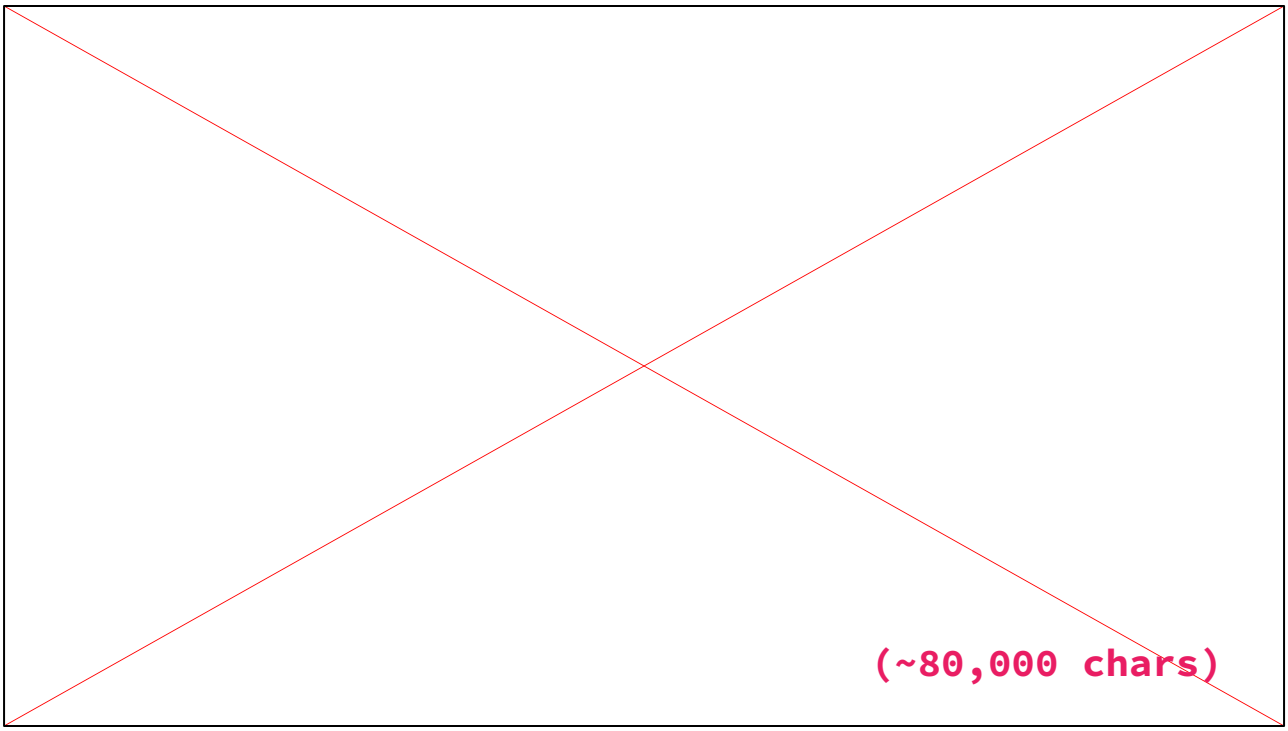
And here's the regex. Any input that matches against that will be a valid 3-ball juggling trick.



Here’s a simple level of Sokoban - everyone know Sokoban? The goal is to move the player around and push the pink boxes onto the dots.



[silence]



[pause]
I think this sets some sort of record for “most text on a single slide”, it’s 80,000 chars in 2pt font and is still falling off the bottom of the screen.

And more!

And those aren't the only impractical uses for this correspondence. Consider the Rubik's cube - it's technically got a finite number of states, a single "win" state, and well-defined transition functions between states. You could, for a given scrambled state, generate a regex that will match against sequences of moves that will solve the cube. Might take you a while though.

References

- *It's super effective! Solving Pokemon Blue with a single, huge regular expression* (Alex Clemmer, !!Con 2018)

  https://youtu.be/n-HTvjIueX0

This is the talk I learned about this from, proposing you could do the same for solving Pokemon Blue.

Thanks for your time!

@LunaSorcery

That’s all I’ve got, thank you!