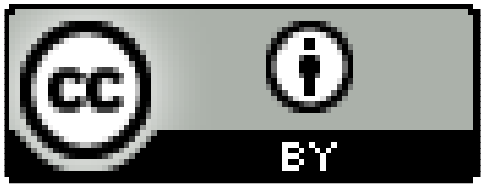
Regular Expressions

Mechanics



Notebook #1

```
Site     Date     Evil (millivaders)
----     ----     ------------------
Baker 1  2009-11-17       1223.0
Baker 1  2010-06-24       1122.7
Baker 2  2009-07-24       2819.0
Baker 2  2010-08-25       2971.6
Baker 1  2011-01-05       1410.0
Baker 2  2010-09-04       4671.6
   ⋮        ⋮                ⋮
```

Notebook #2

```
Site/Date/Evil
Davison/May 22, 2010/1721.3
Davison/May 23, 2010/1724.7
Pertwee/May 24, 2010/2103.8
Davison/June 19, 2010/1731.9
Davison/July 6, 2010/2010.7
Pertwee/Aug 4, 2010/1731.3
Pertwee/Sept 3, 2010/4981.0
  ⋮          ⋮          ⋮
```

```
'(.+)/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/(.+)'
```

This pattern matches:

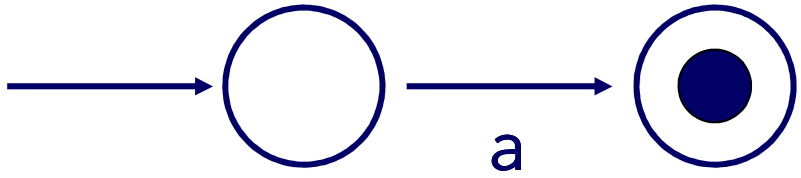
- one or more characters
- a slash
- a single upper-case letter
- one or more lower-case letters
- a space
- one or two digits
- a comma if one is there
- a space
- exactly four digits
- a slash
- one or more characters

How?

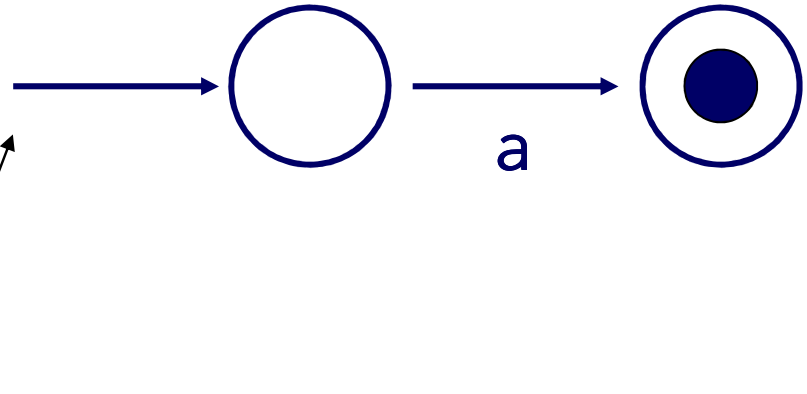
How?

Using *finite state machines*

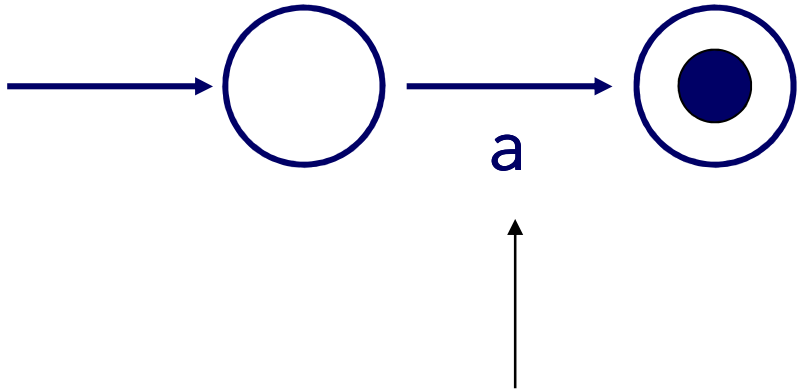
Match a single 'a'



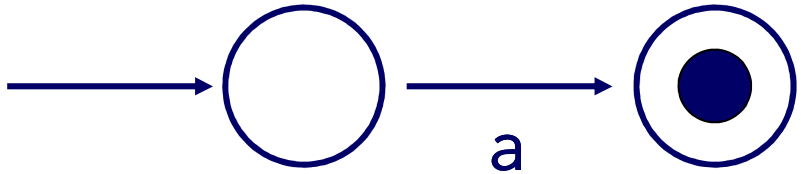
Match a single 'a'



Match a single 'a'



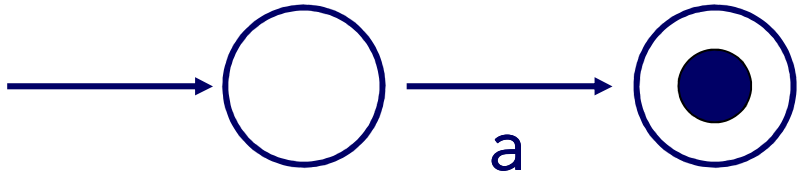
Match a single 'a'



must be here

at the end

Match a single 'a'

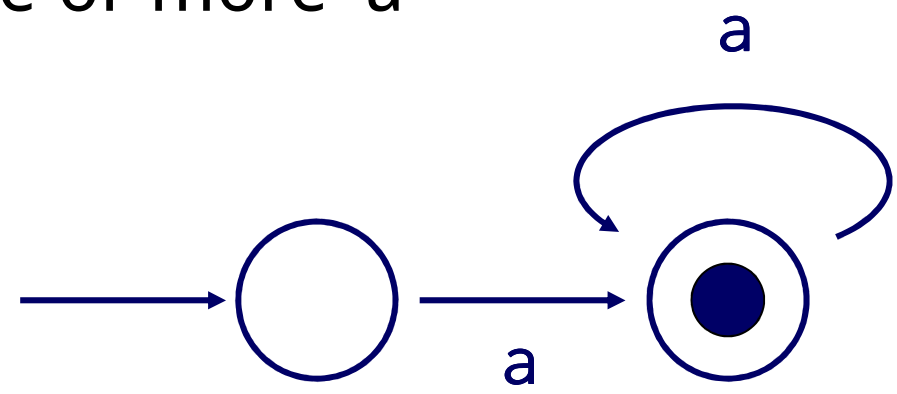


a

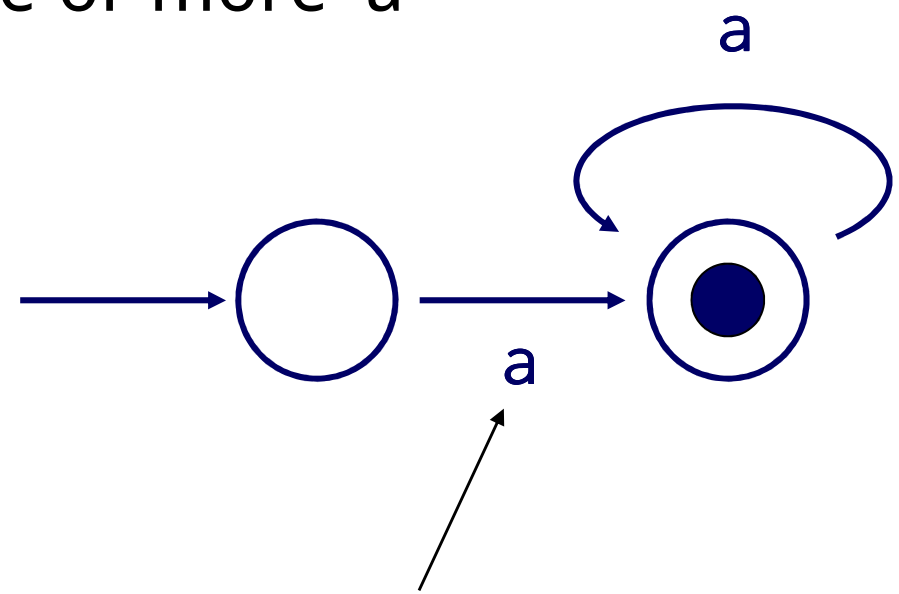
must be here

at the end

Match one or more 'a'



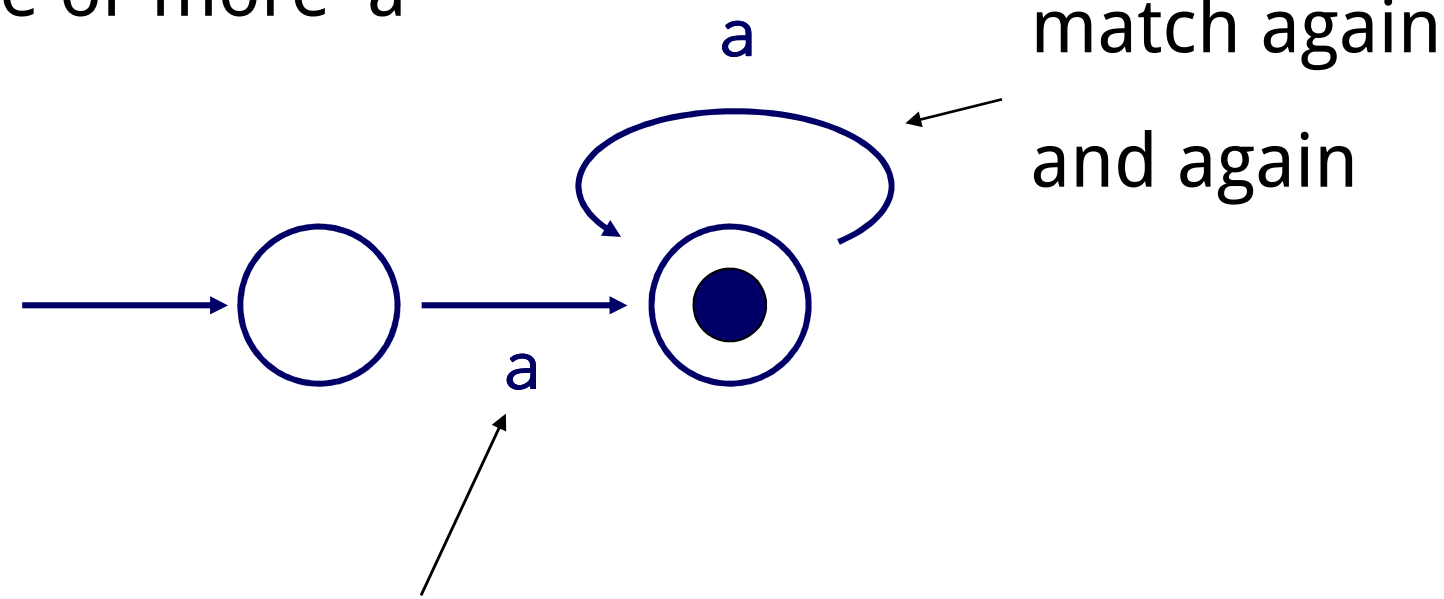
Match one or more 'a'



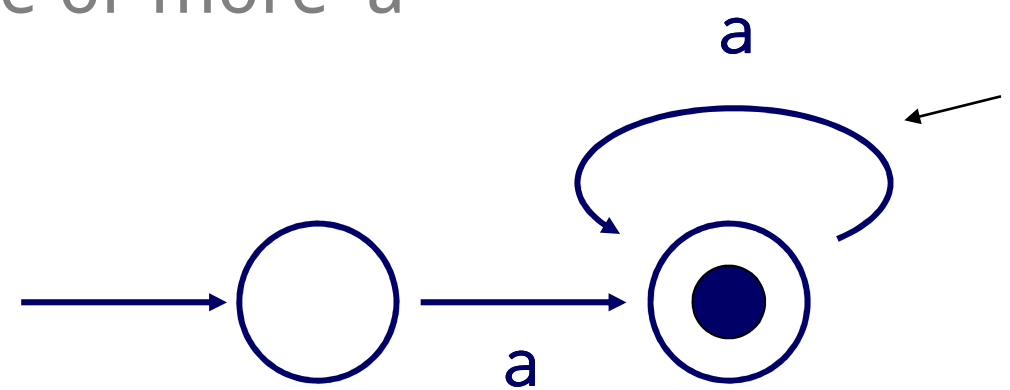
match this

as before

Match one or more 'a'



Match one or more 'a'



match again

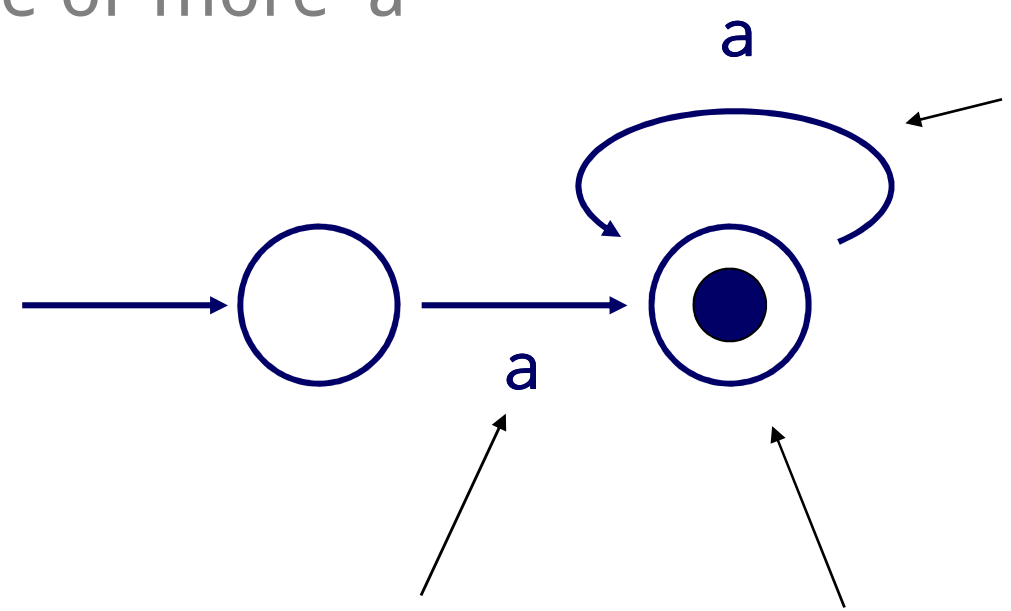
and again

match this

as before

don't have to stop here the first time, just have to be here at the end

Match one or more 'a'



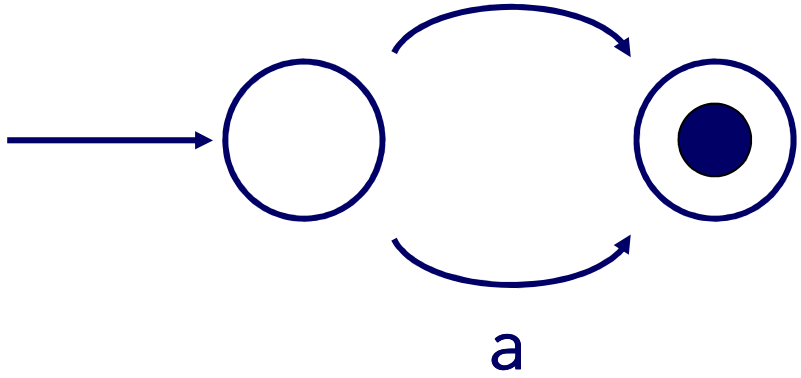
match again and again

match this as before

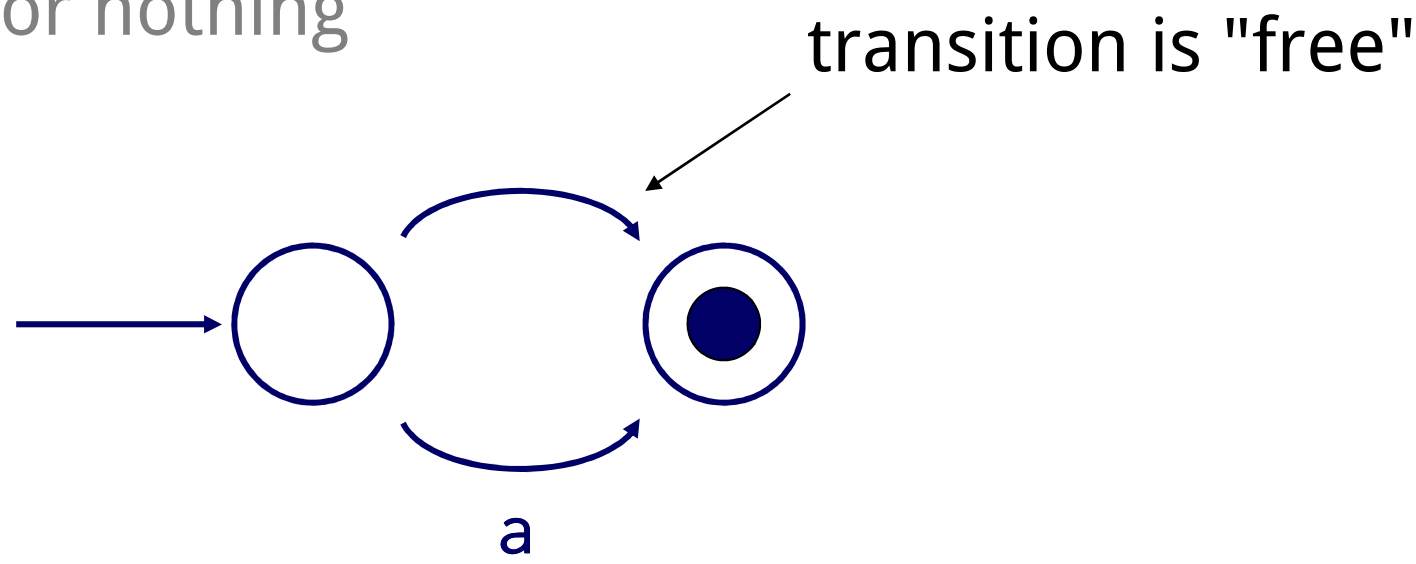
don't have to stop here the first time, just have to be here at the end

a
a+

Match 'a' or nothing

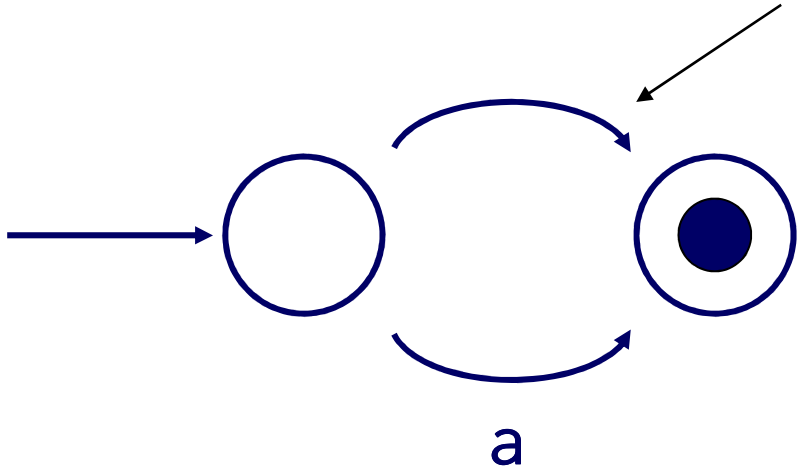


Match 'a' or nothing



Match 'a' or nothing

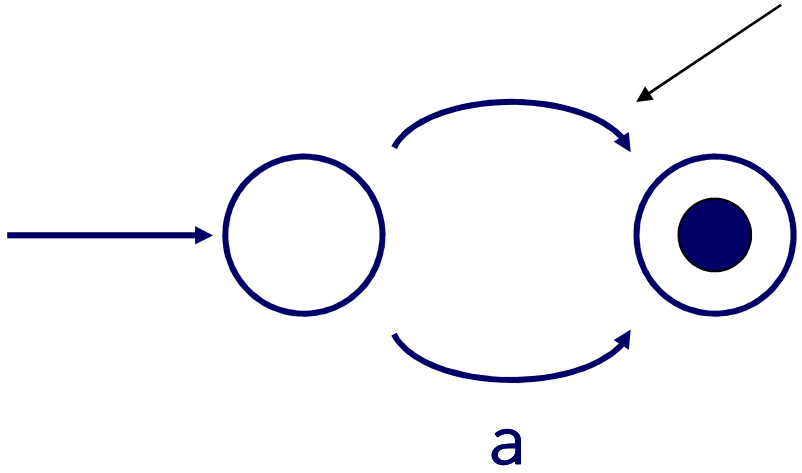
transition is "free"



So this is '(a|)'

Match 'a' or nothing

transition is "free"

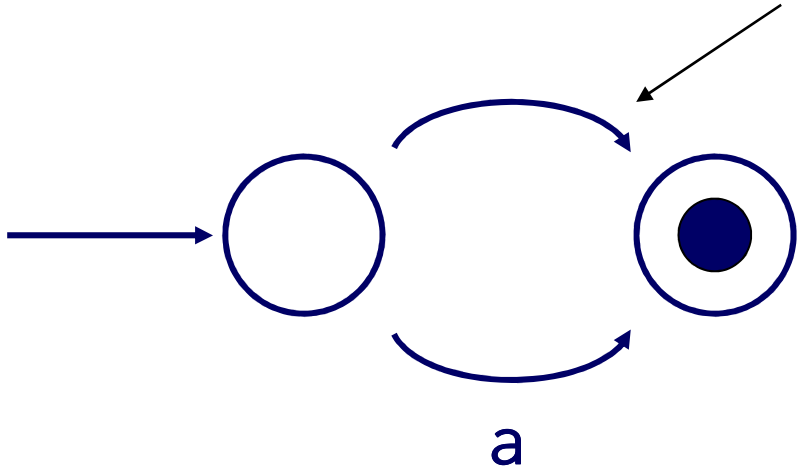


So this is '(a|)'

Which is 'a?'

Match 'a' or nothing

transition is "free"

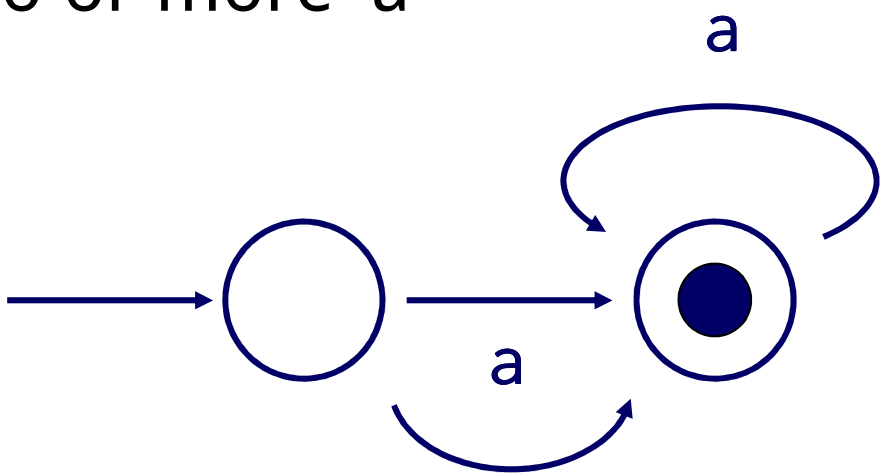


So this is '(a|)'

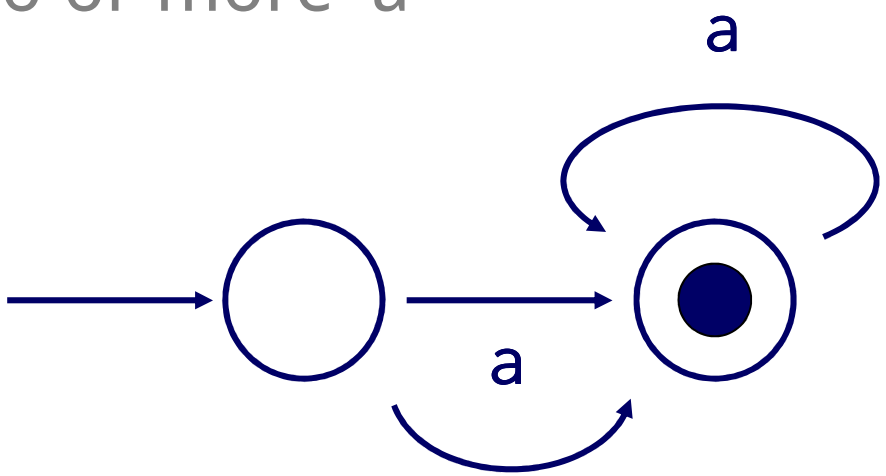
Which is 'a?'

a
a+
a?

Match zero or more 'a'

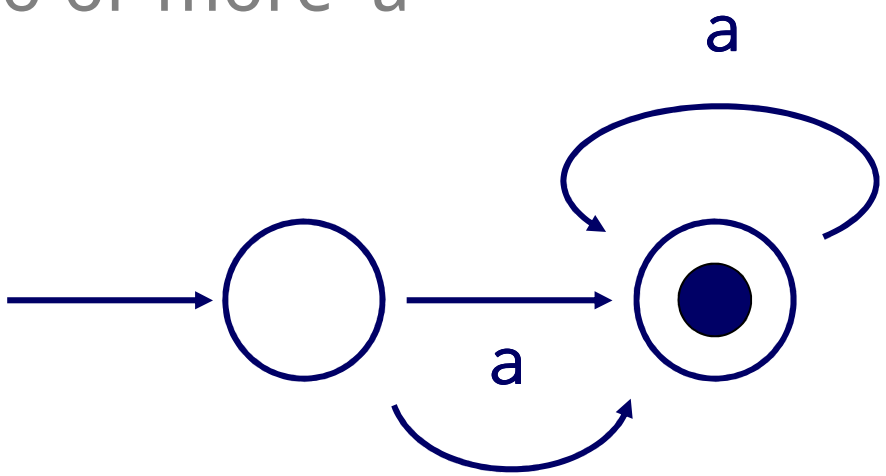


Match zero or more 'a'



Combine ideas

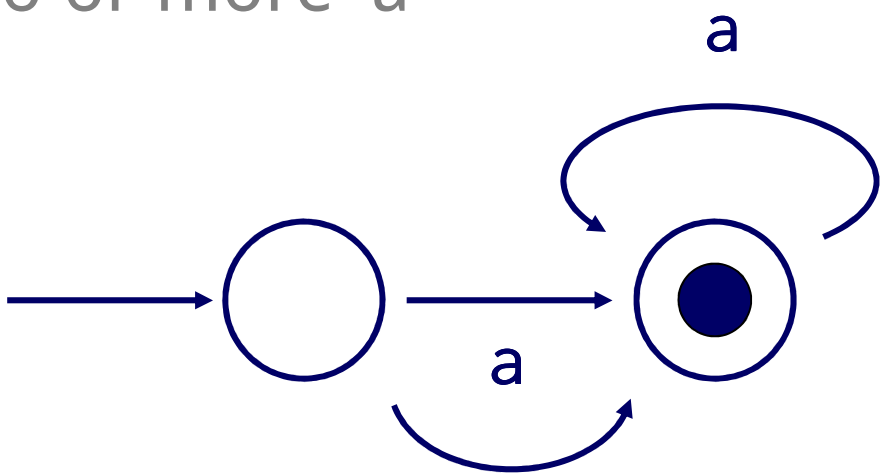
Match zero or more 'a'



Combine ideas

This is 'a*'

Match zero or more 'a'

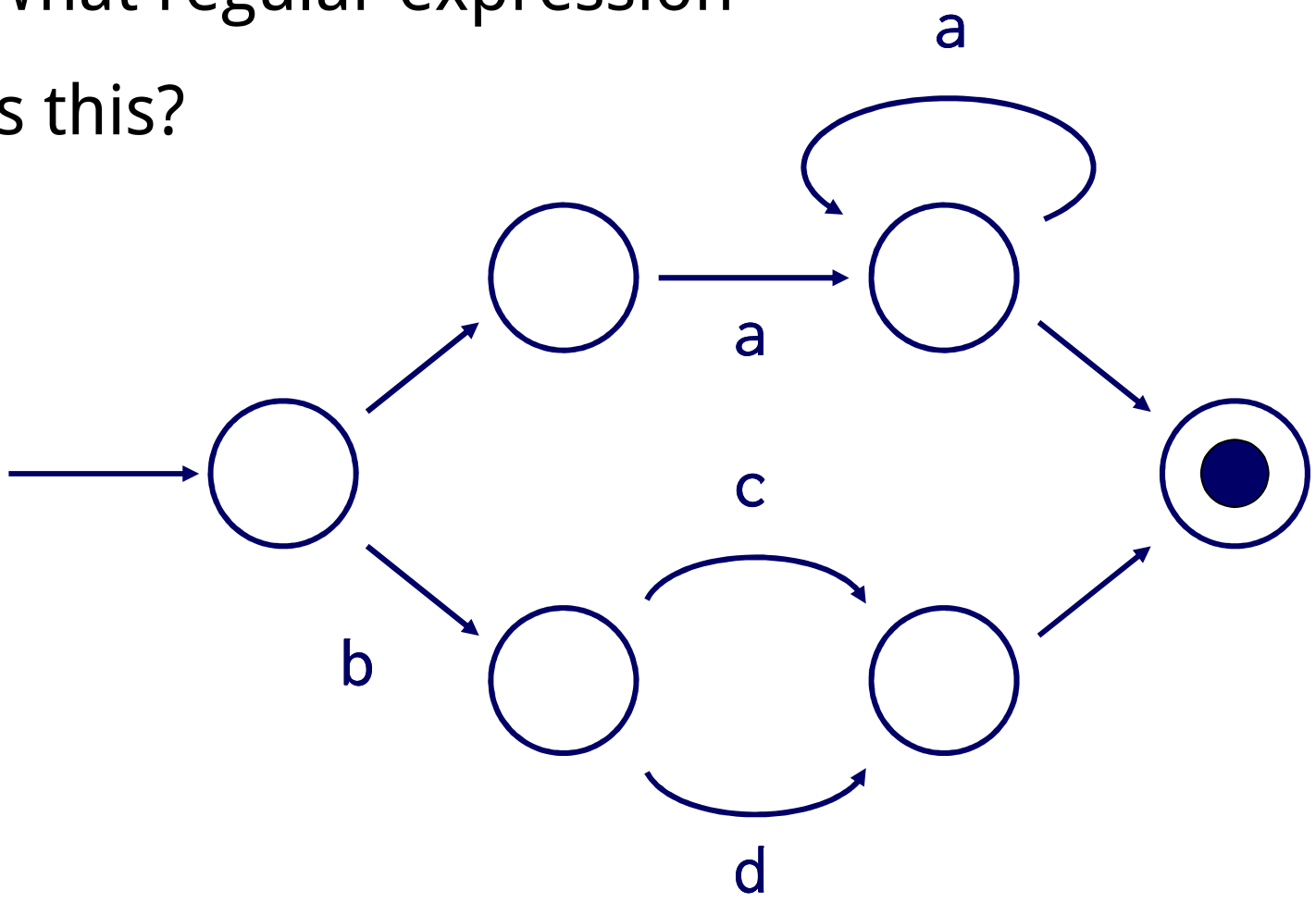


Combine ideas

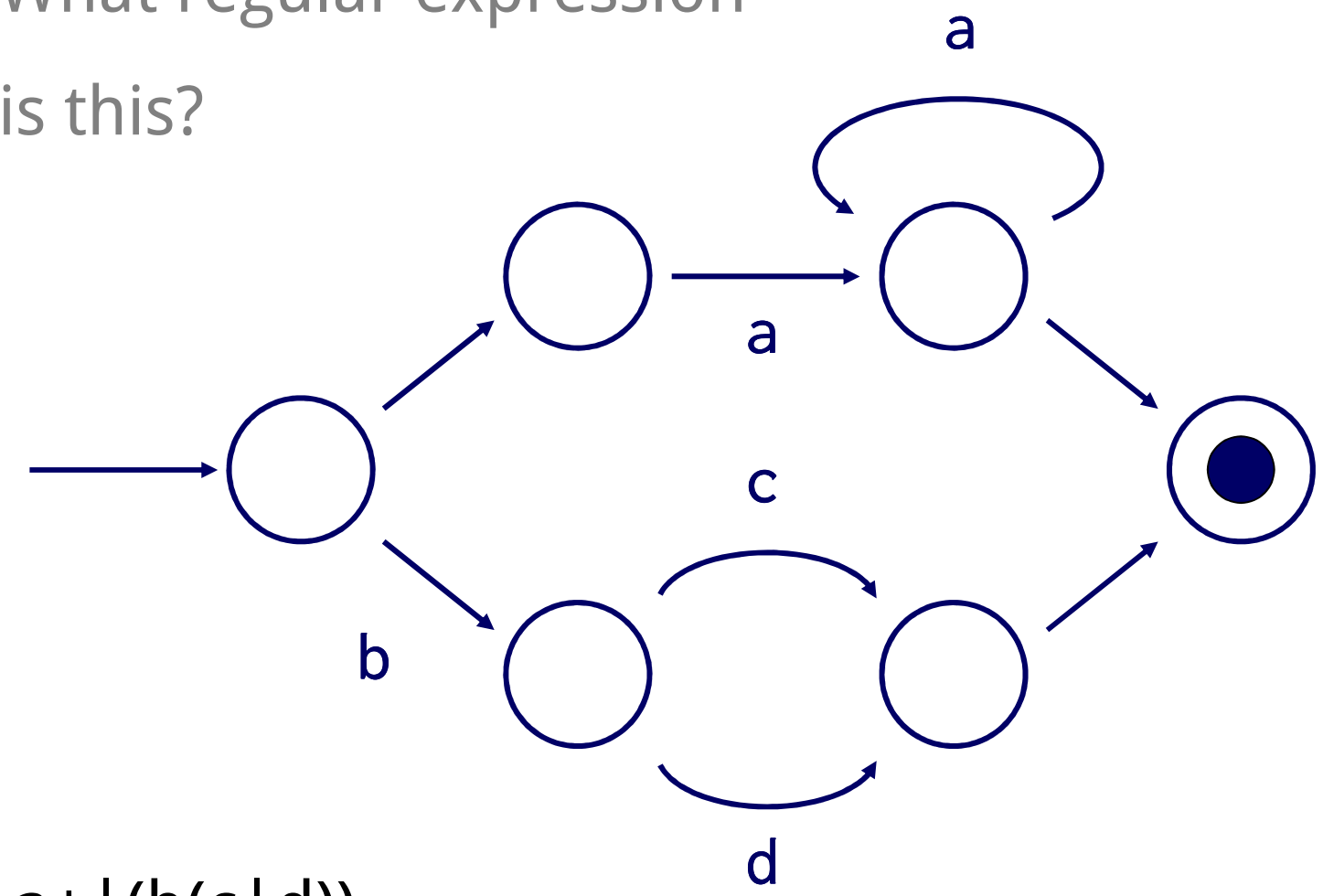
This is 'a*'

a
a+
a?
a*

What regular expression is this?



What regular expression is this?



a+|(b(c|d))

Action at a node depends only on:

Action at a node depends only on:

- arcs out of that node

Action at a node depends only on:

- arcs out of that node

- characters in target data

Action at a node depends only on:

- arcs out of that node

- characters in target data

Finite state machines have *no memory*

Action at a node depends only on:

- arcs out of that node

- characters in target data

Finite state machines have *no memory*

Means it is impossible to write a regular expression to check if arbitrarily nested parentheses match

Action at a node depends only on:

- arcs out of that node

- characters in target data

Finite state machines have *no memory*

Means it is impossible to write a regular expression

to check if arbitrarily nested parentheses match

"(((....)))" requires memory

Action at a node depends only on:

- arcs out of that node

- characters in target data

Finite state machines have *no memory*

Means it is impossible to write a regular expression to check if arbitrarily nested parentheses match

"(((....)))" requires memory (or at least a counter)

Action at a node depends only on:

- arcs out of that node

- characters in target data

Finite state machines have *no memory*

Means it is impossible to write a regular expression to check if arbitrarily nested parentheses match

"(((....)))" requires memory (or at least a counter)

Similarly, only way to check if a word contains each vowel once is to write 5! = 120 clauses

Why use a tool with limits?

Why use a tool with limits?

They're fast

Why use a tool with limits?

They're fast

- After some pre-calculation, a regular expression only has to look at each character in the input data once

Why use a tool with limits?

They're fast

- After some pre-calculation, a regular expression only has to look at each character in the input data once

It's readable

Why use a tool with limits?

They're fast

- After some pre-calculation, a regular expression only has to look at each character in the input data once

It's readable

- More readable than procedural equivalent

Why use a tool with limits?

They're fast

- After some pre-calculation, a regular expression only has to look at each character in the input data once

It's readable

- More readable than procedural equivalent

And regular expressions can do a lot more than what we've seen so far

created by

Greg Wilson

June 2010