

Christian Colombo
(joint course design with Gordon J. Pace)
University of Malta

March 2018

RUNTIME VERIFICATION FROM THEORY TO PRACTICE AND BACK

Regular Expressions

Regular Expressions

- Regular expressions can also be used to write specifications either by giving the wrong or the good behaviour.
- Examples of specifications of wrong behaviour ($\neq e$ means any event except for e):
 - $(\text{login}; (\text{read+write})^*; \text{logout})^*; (\text{read+write})$
 - $(\neq \text{approveAccount})^*; \text{transfer}$
- Examples of specifications of good behaviour (matching prefixes of the regular expression):
 - $(\text{login}; (\text{read+write})^*; \text{logout})^*$
 - $\neq \text{transfer}^*; \text{approveAccount}; \neq \text{transfer}^*; \text{transfer}$

Monitoring Regular Expressions

- As with finite state automata, we may want to quantify universally over the target of events:
 - `foreach target (User u)
 (u.login(); u.transfer(..);
 u.logout())*;
 u.transfer(..)`
- Verification can be done either by:
 - By using the residual algorithm or
 - Transforming into an automaton

Regular Expressions

RE ::= ? (Any) | 0 (Nothing)
| 1 (End)
| a (Proposition)
| !a (All the propositions except a)
| RE + RE (Choice)
| RE ; RE (Sequence)
| RE* (Repetition)
| (RE) (Bracketed expression)

Regular Expressions

RE ::= ? (Any) | 0 (Nothing)
| 1 (End)
| a Matches the event a
| !a Matches any event but a
except a)
| RE + RE
| RE ; RE
| RE* (Repetition)
| (RE) (Bracketed expression)

Matches if nothing happens, i.e. an empty string

RE	?	(Any)		0	(Nothing)
	1	(End)			
	a		(Propositi		
	!a		(All the pr		
except a)					Nothing can match (like reaching a bad state)
	RE + RE	(Choice)			
	RE ; RE	(Sequence)			
	RE*	(Repetition)			
	(RE)	(Bracketed expression)			

Regular Expressions

RE ::= ? (Any) | 0 (Nothing)
| 1 (End)

| a (Matches either or
| !a (

except a)

| RE + RE (Choice
| RE ; RE (Sequence
| RE* (Repetition
| (RE) (Bracket)

Matches the
subsequent behaviour
of the two REs

Matches zero or more
repetitions of RE

Exercise

- Express properties 2, 5, 6, and 10 of the Financial Transaction System in terms of regular expressions (you can choose whether to specify “matching” or “non matching” expressions)

Example

Property 2 - The transaction system must
be initialised before any user logs
in.

```
property matching {
    (!USER_login)* ;
ADMIN_initialise ; ?*
}
```

```
property not matching {
    (!ADMIN_initialise)*; USER_login
}
```

More Advanced Example

Property 5 – Once a user is disabled, he or she may not withdraw from an account until being made activate again.

```
property foreach target (UserInfo u) matching {  
  ( (!makeDisabled)* ; makeDisabled ; (!  
withdrawFrom)* ; makeActive )* }  
  
property foreach target (UserInfo u) not  
matching {  
  (?)* ; makeDisabled ; (!makeActive)* ;  
withdrawFrom  
}
```

Monitoring Regular Expressions using Residuals

Residuals

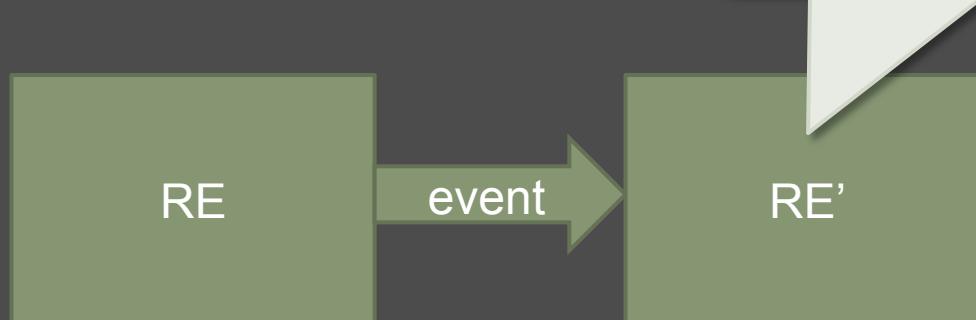
One way of monitoring REs is by modifying the RE for each observed event



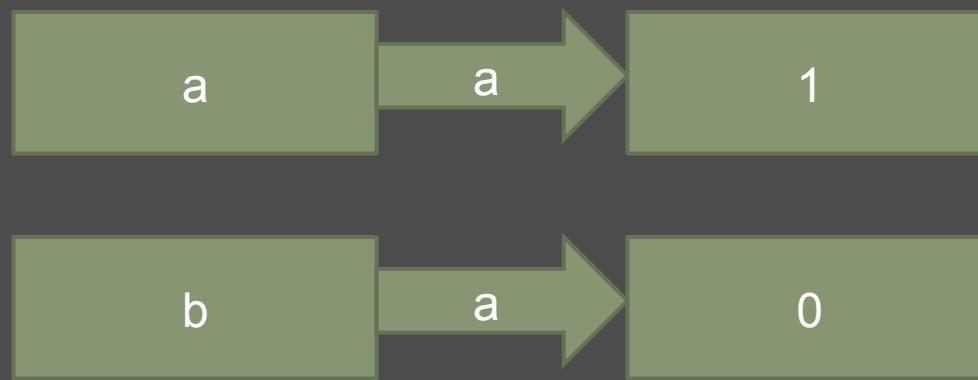
Residuals

One way of monitoring REs is by modifying the RE for each observed event

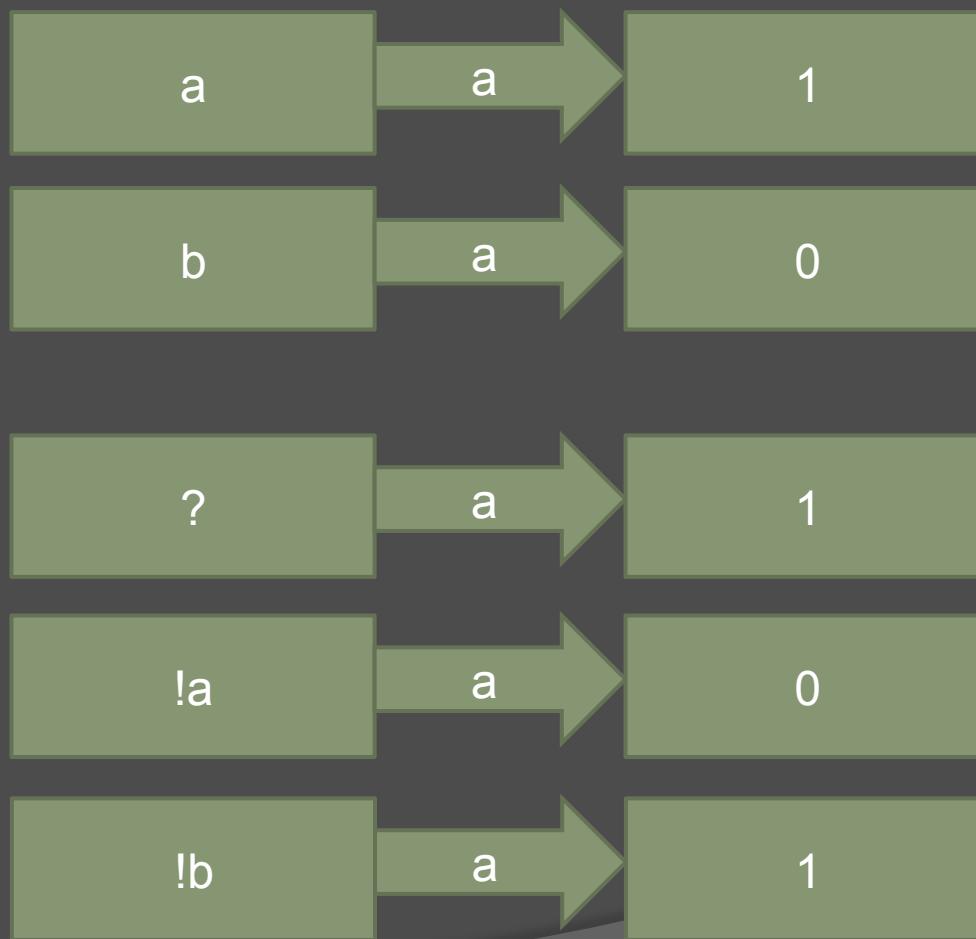
What remains to be satisfied



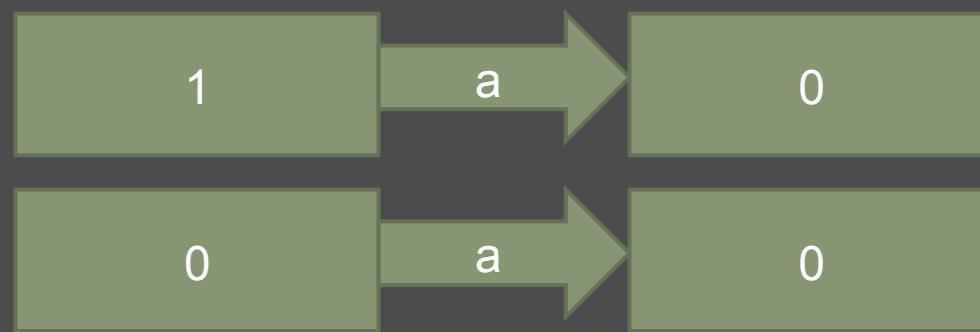
Residuals - Basic



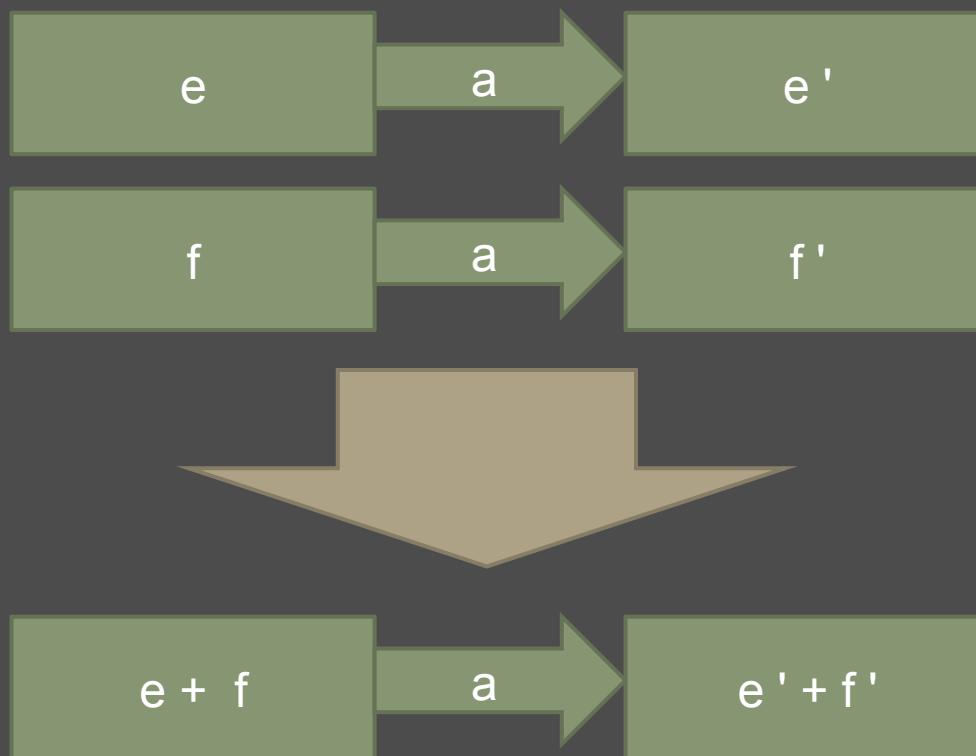
Residuals - Basic



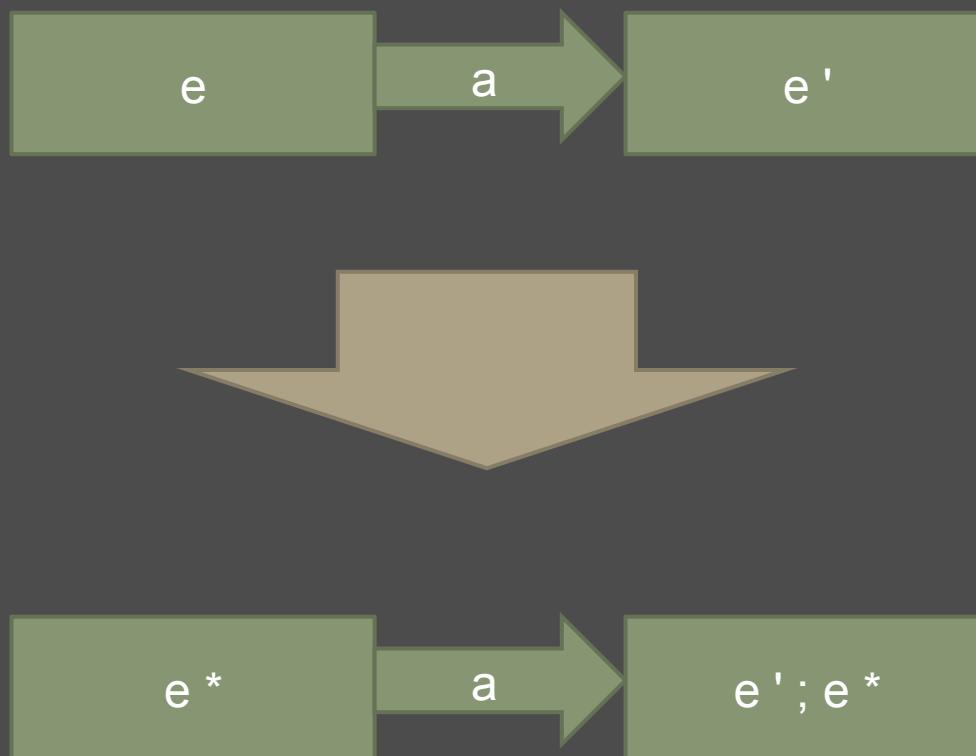
Residuals - Basic



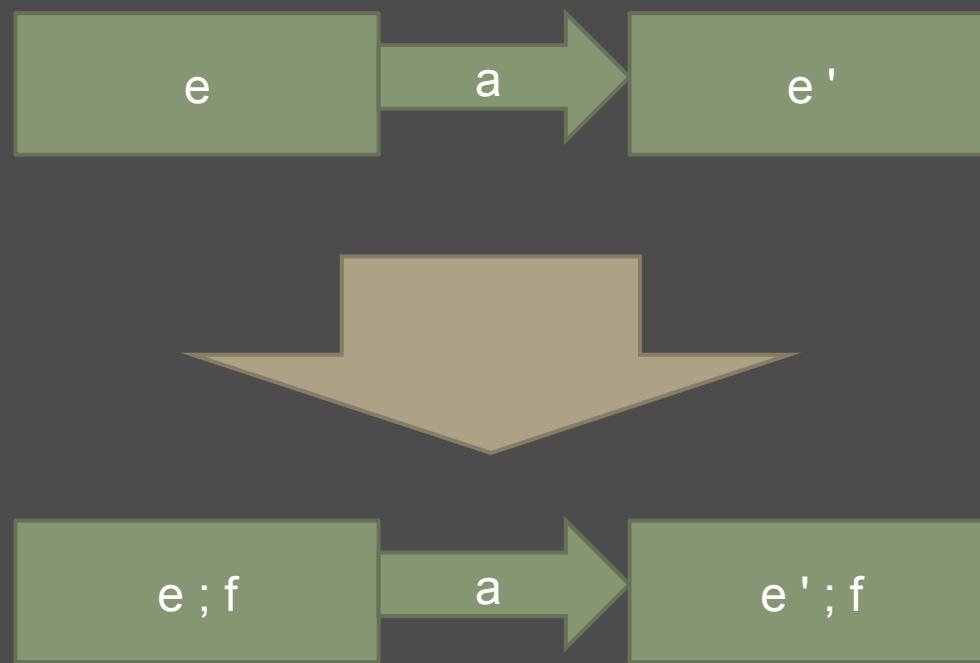
Residuals – Or



Residuals – Star

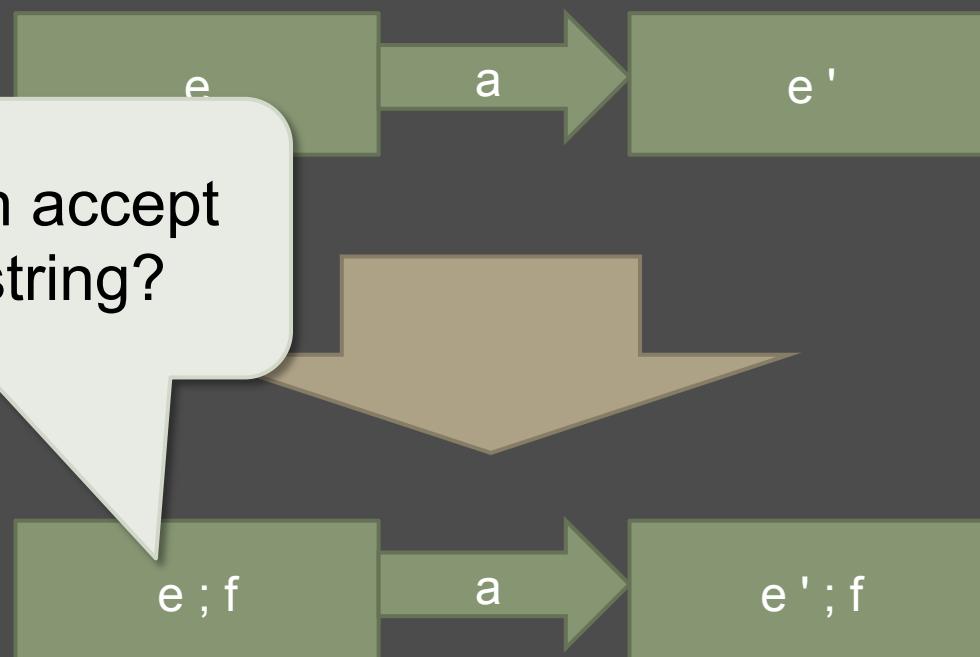


Residuals – Sequence

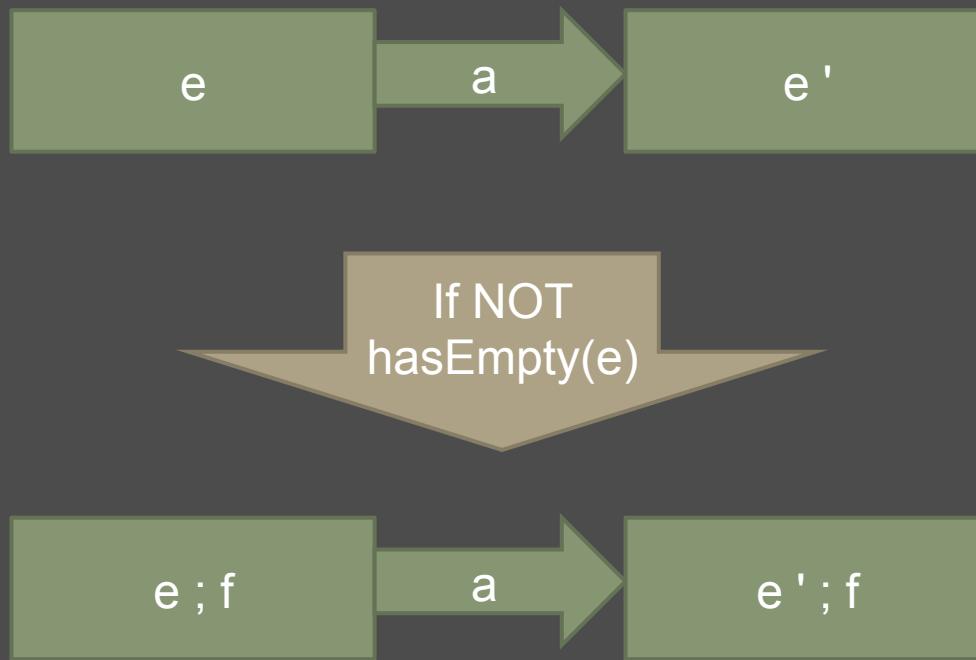


Residuals – Sequence

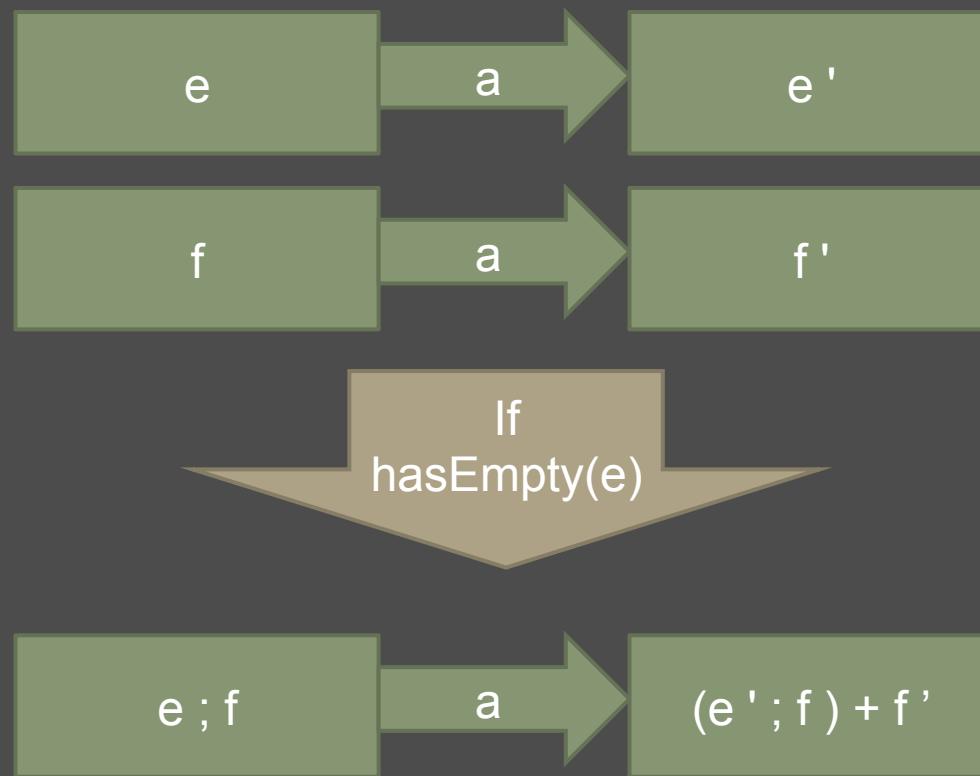
What if e can accept
an empty string?



Residuals – Sequence



Residuals – Sequence



Accepting an Empty String

`hasEmpty (?) = false`

`hasEmpty (0) = false`

`hasEmpty (1) = true`

`hasEmpty (a) = false`

`hasEmpty (!a) = false`

`hasEmpty (RE1 + RE2) = hasEmpty (RE1) or`

`hasEmpty (RE2)`

`hasEmpty (RE1 ; RE2) = hasEmpty (RE1) and`

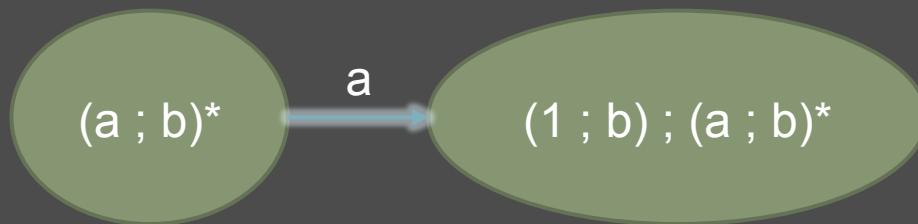
`hasEmpty (RE2)`

`hasEmpty (RE*) = true`

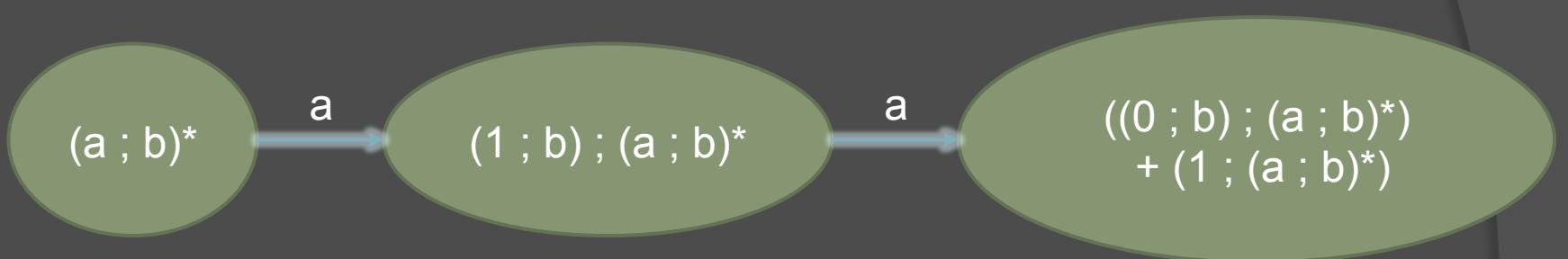
Residuals Example

- ◎ Apply residuals on $(a ; b)^*$ over the following event sequences:
 1. a, b, a, b
 2. b
 3. a, a
 4. a, b, b

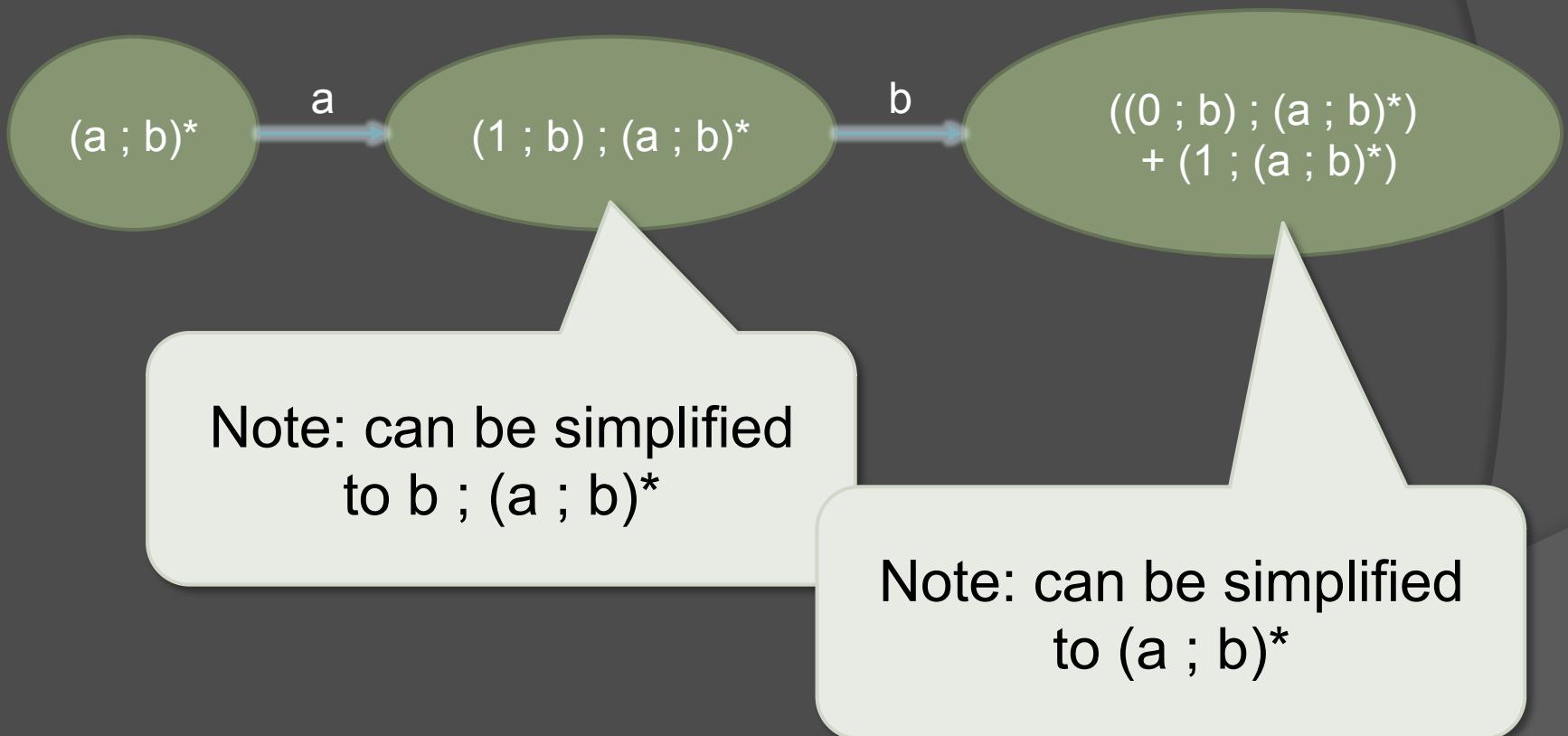
Residuals Example 1



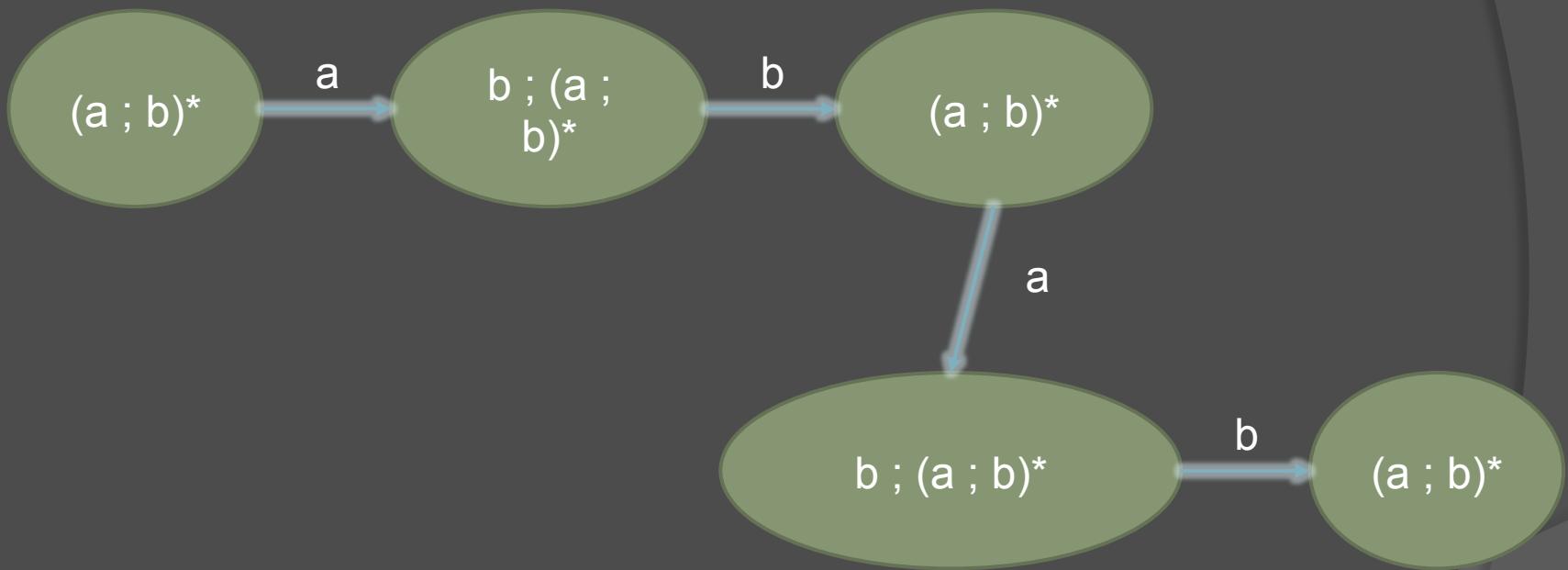
Residuals Example 1



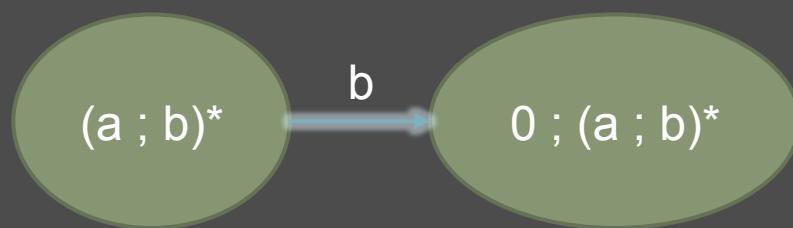
Residuals Example 1



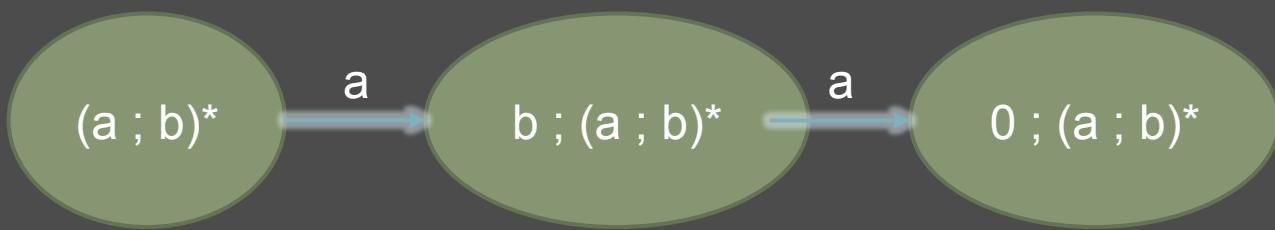
Residuals Example 1



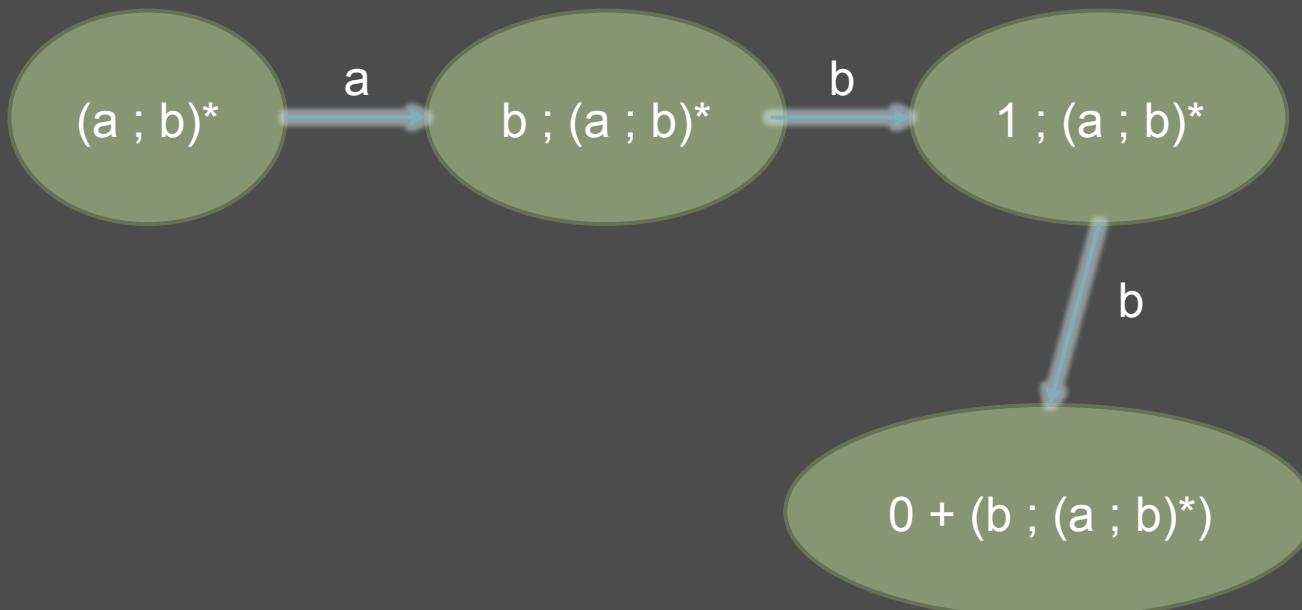
Residuals Example 2



Residuals Example 3



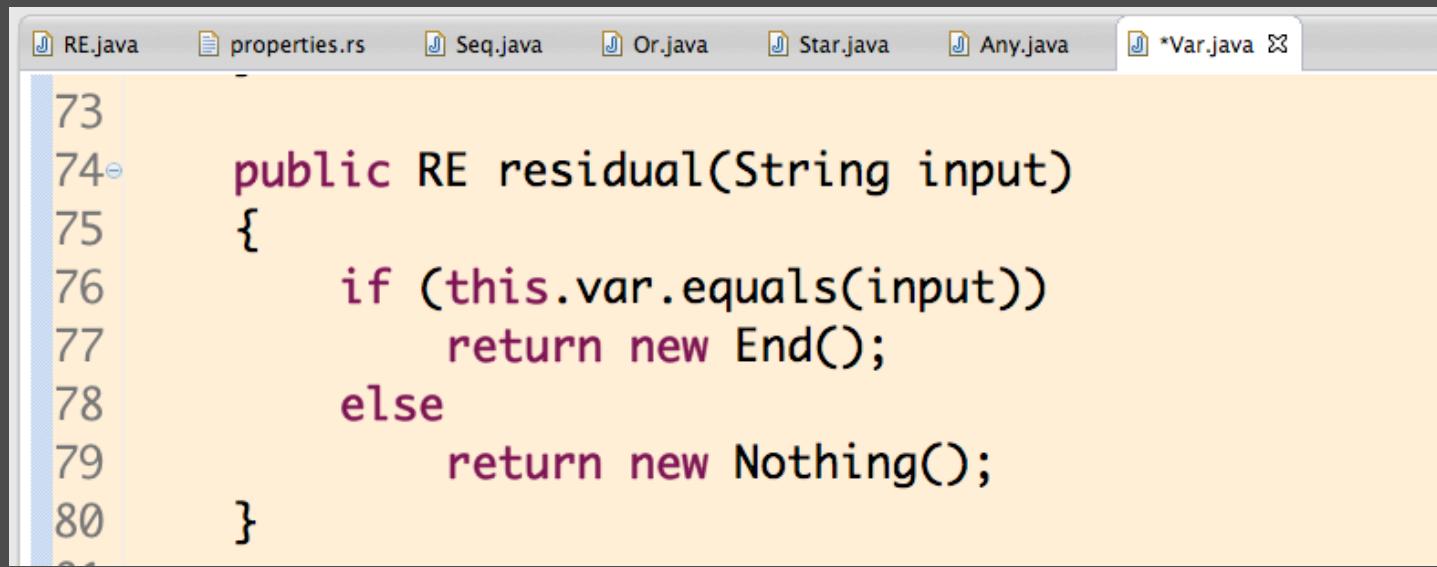
Residuals Example 4



Exercises

- Implement the residual function for all RE classes (i.e. fill in the method for the Or class)

Example



The screenshot shows a Java code editor with the following code:

```
73
74  public RE residual(String input)
75  {
76      if (this.var.equals(input))
77          return new End();
78      else
79          return new Nothing();
80  }
```

The code is part of a class named *Var.java. The tabs at the top show other files: RE.java, properties.rs, Seq.java, Or.java, Star.java, Any.java, and *Var.java.

Simplifying REs

- ⦿ If not simplified, REs quickly become long and unreadable with residuals

e.g.: $(0 ; ((a + 0) + ?)) + (0 ; 1) = 0$

Implemented Simplification Rules

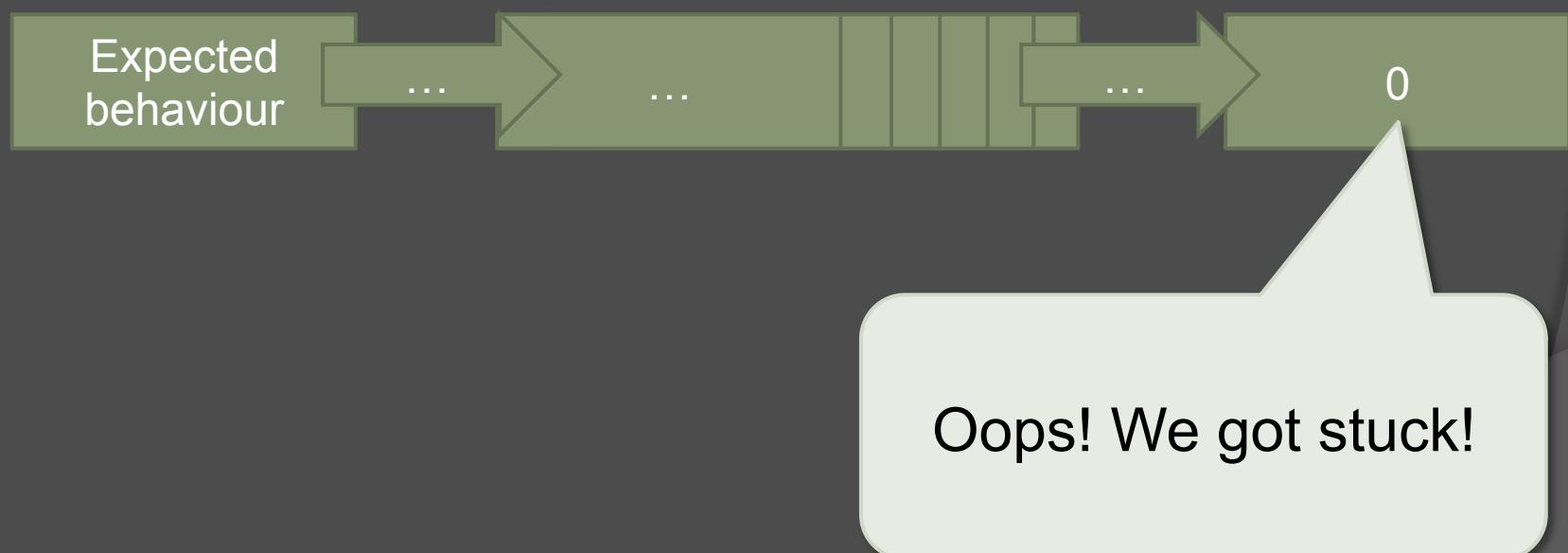
- ◎ $1 ; \text{RE} = \text{RE}$
- ◎ $0 ; \text{RE} = 0$
- ◎ $0 + \text{RE} = \text{RE}$

Detecting a Violation

- What does it mean to detect a violation?

Detecting a Violation

- What does it mean to detect a violation?



Detecting a Violation

- What does it mean to detect a violation when monitoring for unexpected behaviour?



Oops! We matched the unexpected behaviour!

(two ways of)

Monitoring Regular Expressions using Automata

Disadvantage of Residuals

- While using residuals it is quite easy to build a monitor...
- ... it involves significant overhead at runtime

(imagine how many times the rules would have to be checked for a significantly long regular expression)

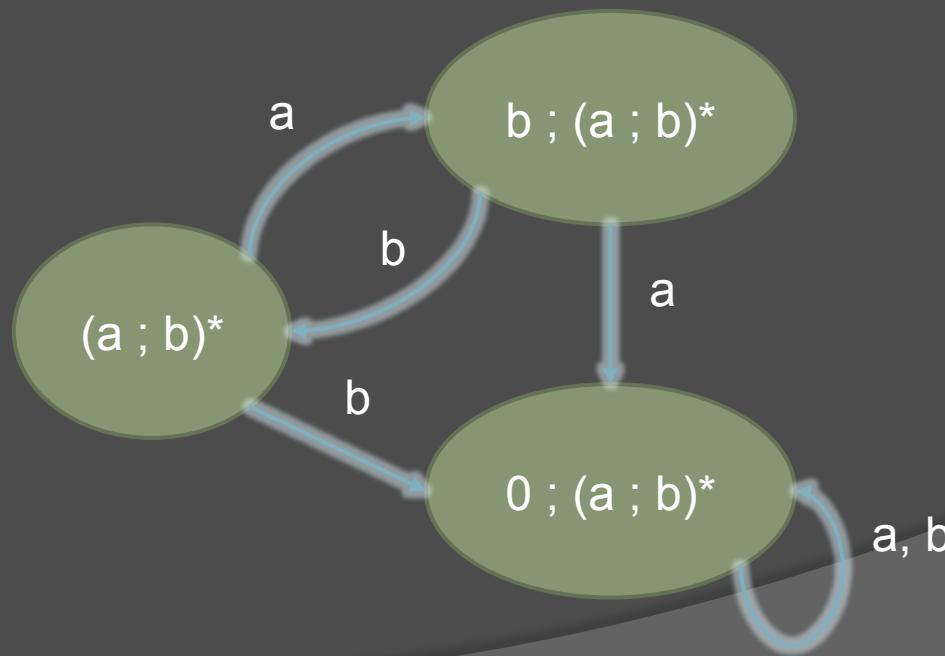
Generating Automata using Residuals

Generating an Automaton 1

- ➊ One way of generating an automaton is by applying residuals for any option at runtime and recording the results

Generating an Automaton 1

- One way of generating an automaton is by applying residuals for any option at runtime and recording the results



Caveats: REs can grow indefinitely

- ➊ $0 + 0 + 0 + 0 + 0 + 0 + 0 \dots + (a;b)^*$
- ➋ The automaton becomes infinite

Caveats: REs can grow indefinitely

- ➊ $0 + 0 + 0 + 0 + 0 + 0 + 0 \dots + (a;b)^*$
- ➋ The automaton becomes infinite
- ➌ Use normal form

RE Normal Form

- A RE is in Normal Form if
 1. All choices are nested on the right-hand side
 - e.g.: $a + (b + (c + d))$
 2. No repeated choices
 3. All choices are ordered (according to some order)
- Note: by associativity, idempotency, and symmetry, any RE can be converted into an RE in normal form

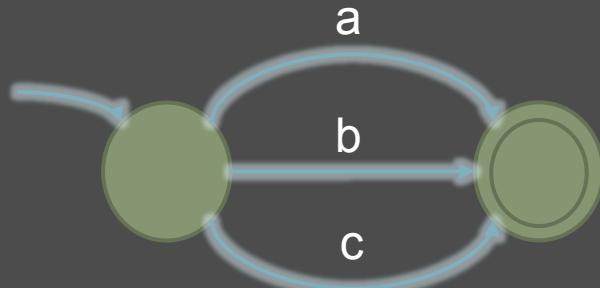
Generating Automata through a 1-to-1 operator mapping

Automata Equivalents

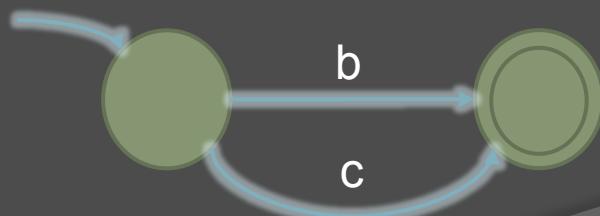
○ a



○ ?



○ !a



Automata Equivalents

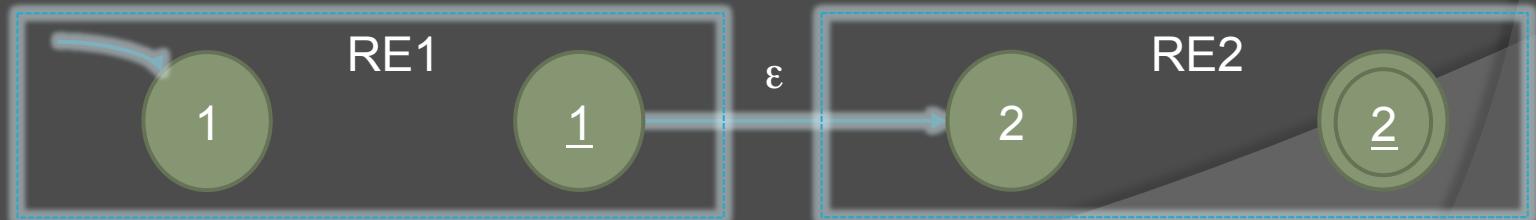
◎ 1



◎ 0

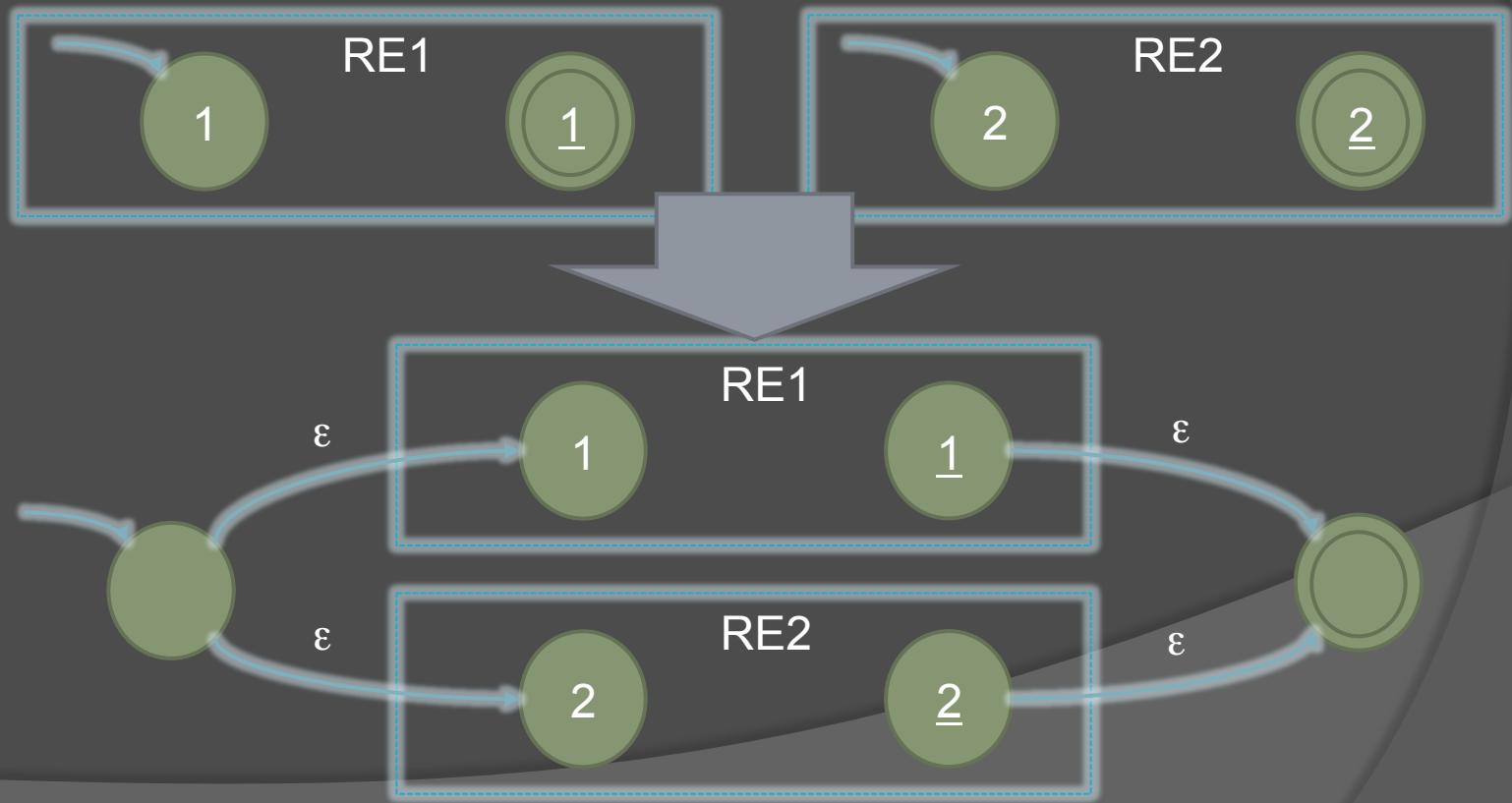
Automata Equivalents

◎ RE1 ; RE2



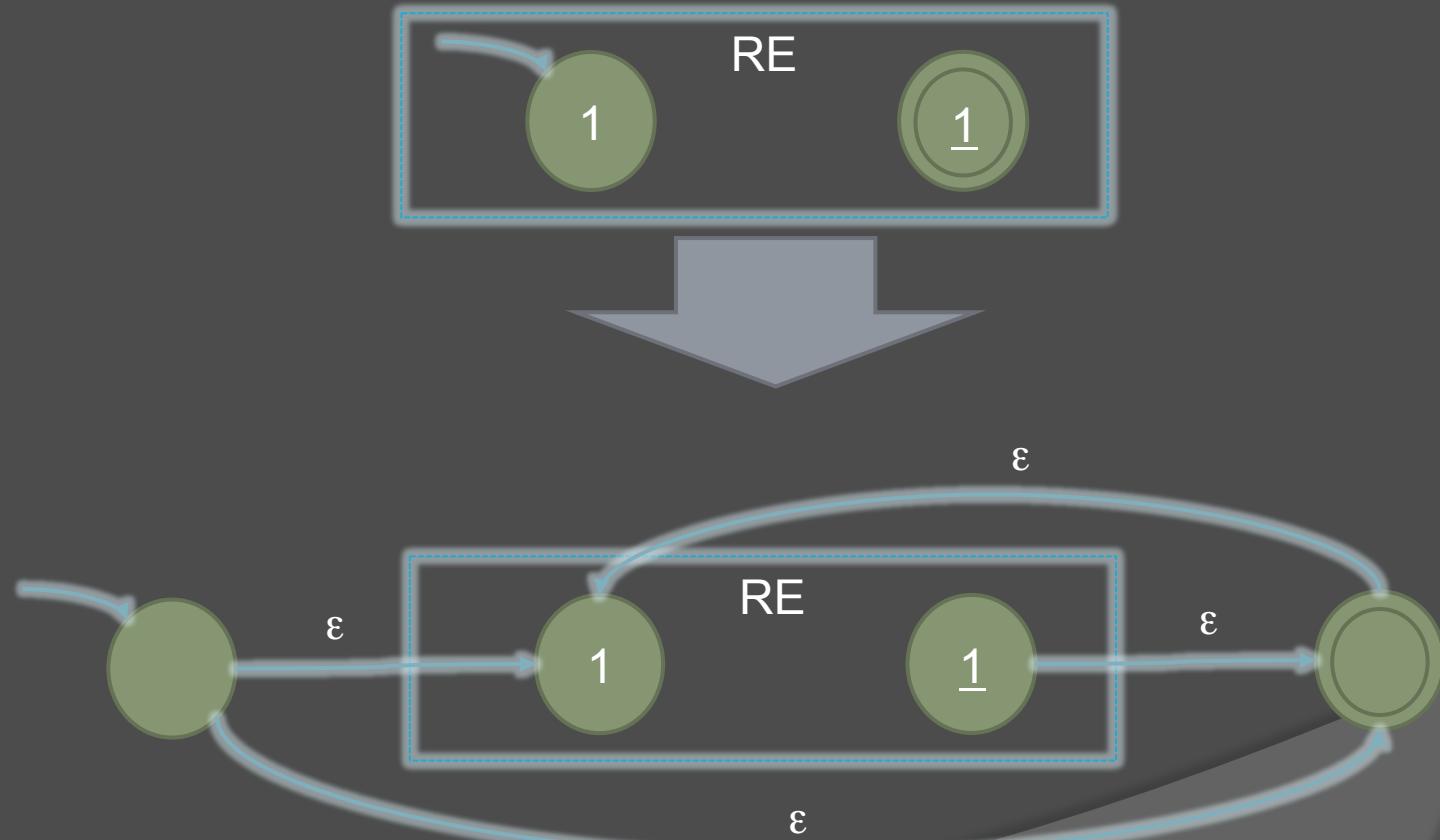
Automata Equivalents

- RE1 + RE2



Automata Equivalents

○ RE*

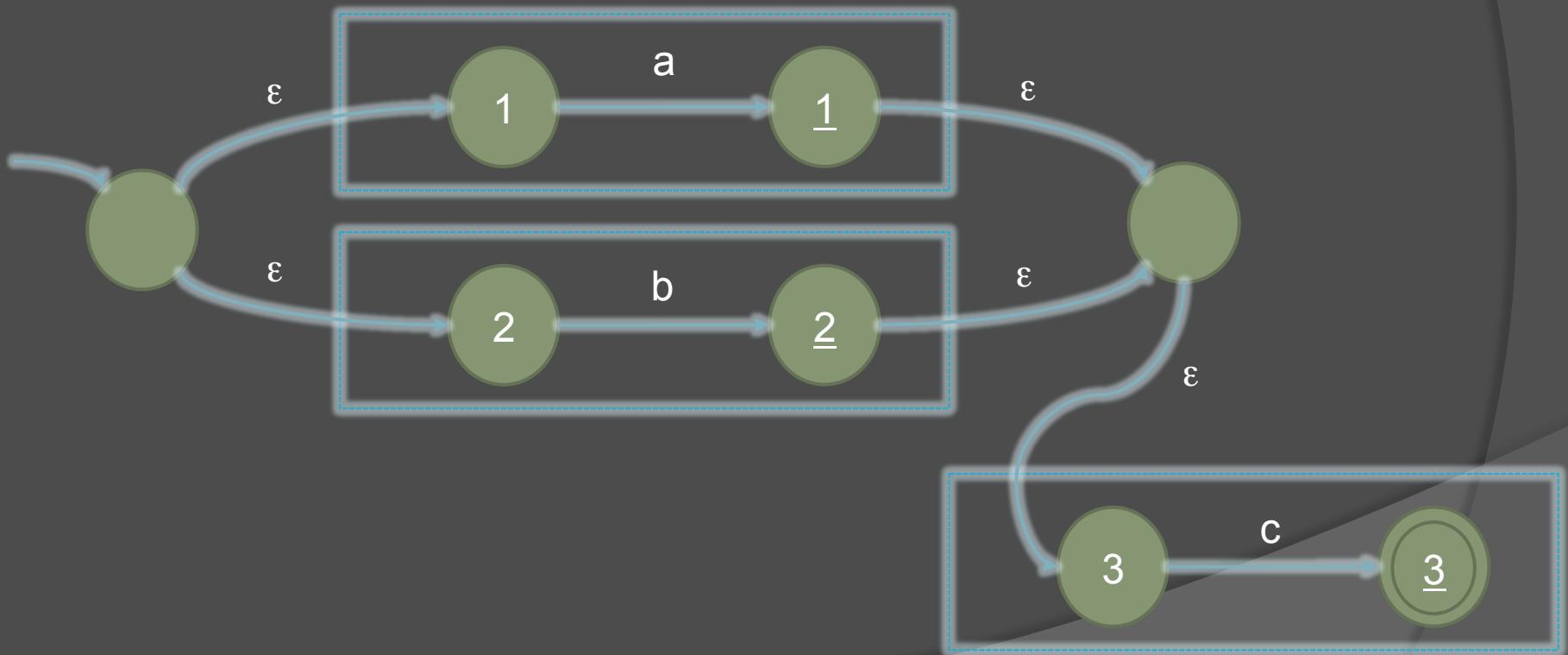


Example

◎ $(a + b) ; c$

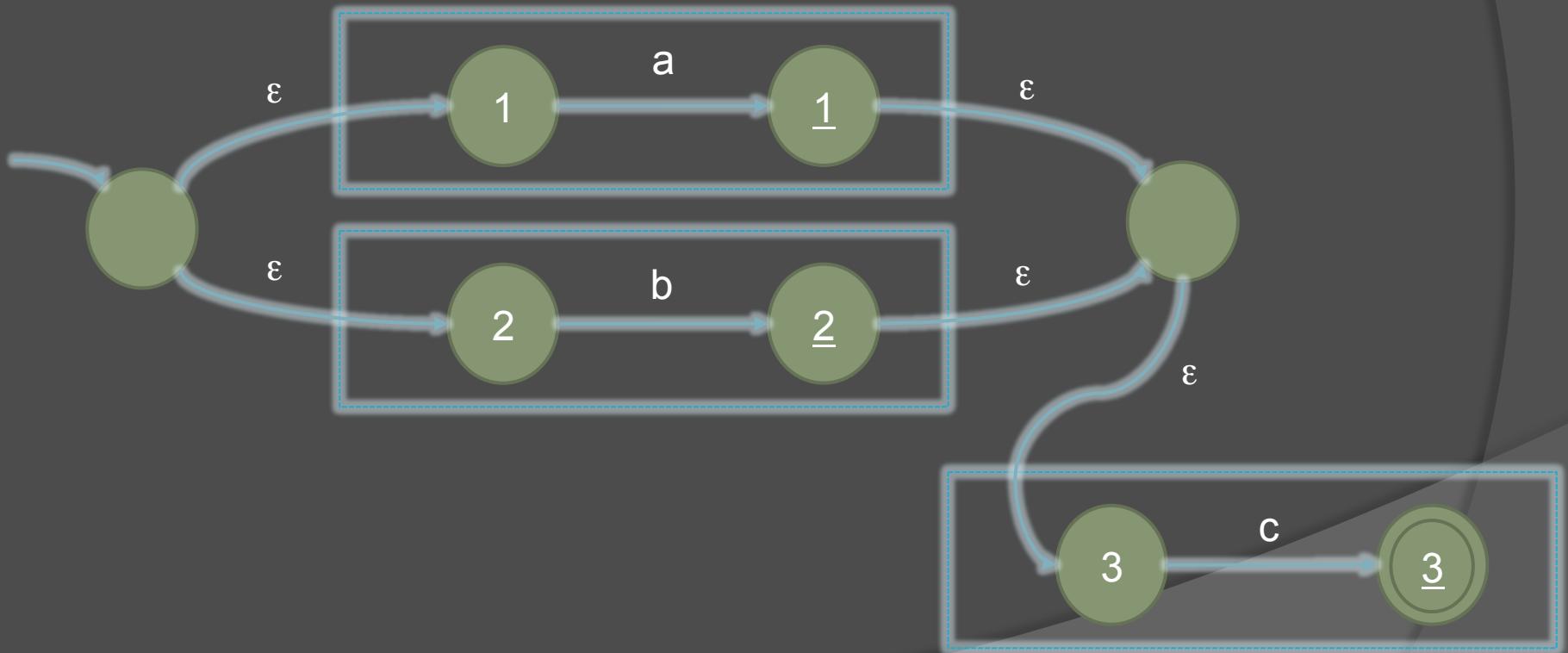
Example

◎ $(a + b) ; c$



Problem

- Monitors have to be deterministic!

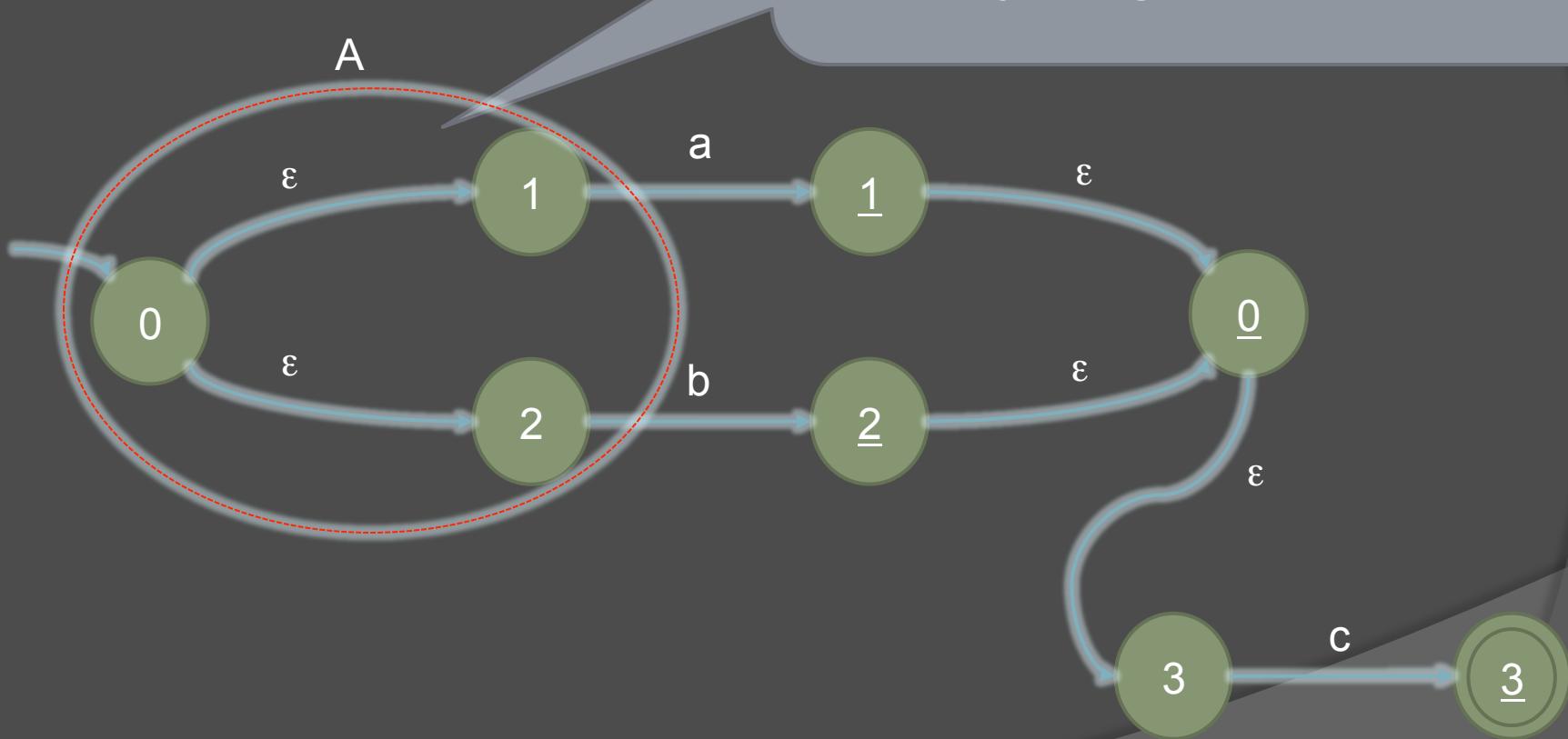


Determinisation in a Nutshell

1. Starting from a given set of states (initially the start states), add **any state reachable from this set on empty strings... call this set A**
2. For each transition leaving any state in A, **combine all possible destinations on each event e** and create a new state B.
3. Restart process from 1. on each B (unless B has already been processed earlier).

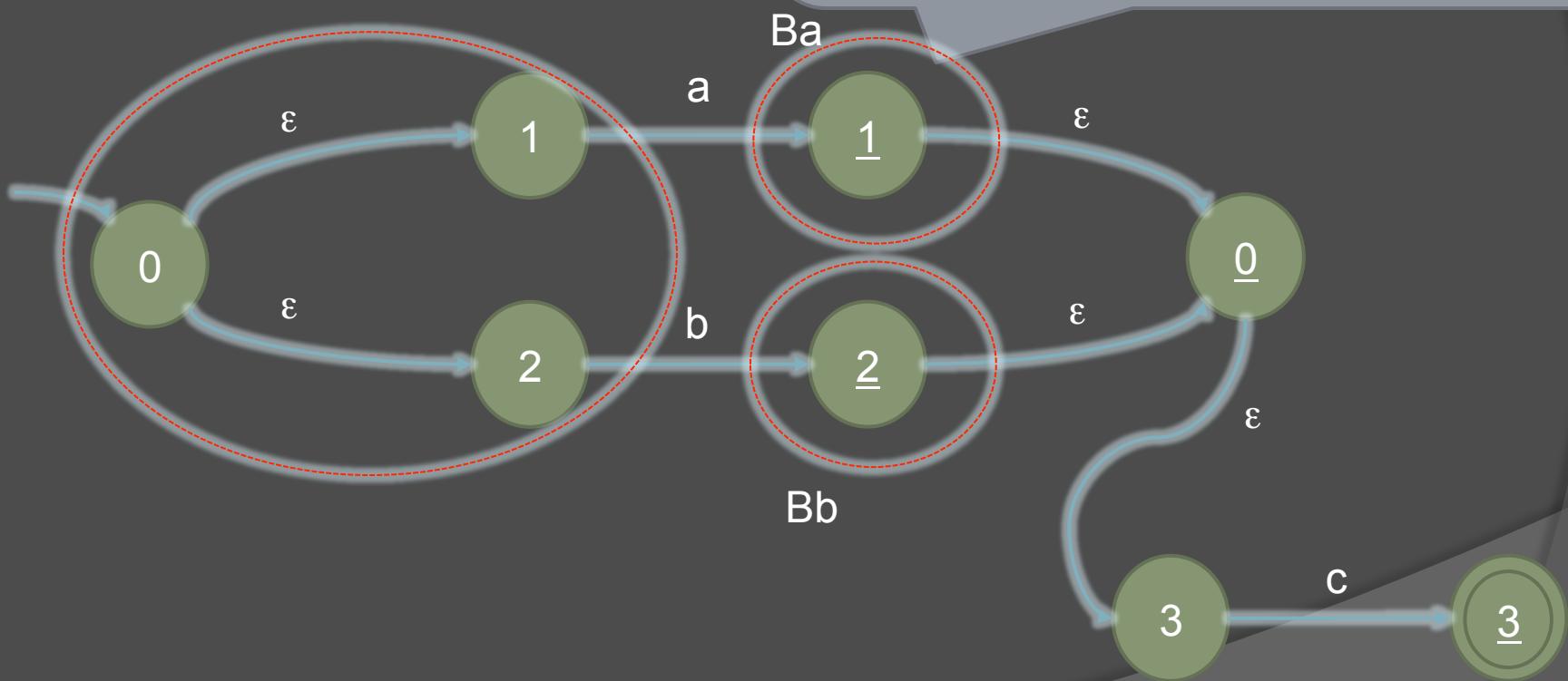
Example

1. Starting from a given set of states (initially all start states), add **any state reachable from this set on empty strings... call this set A**



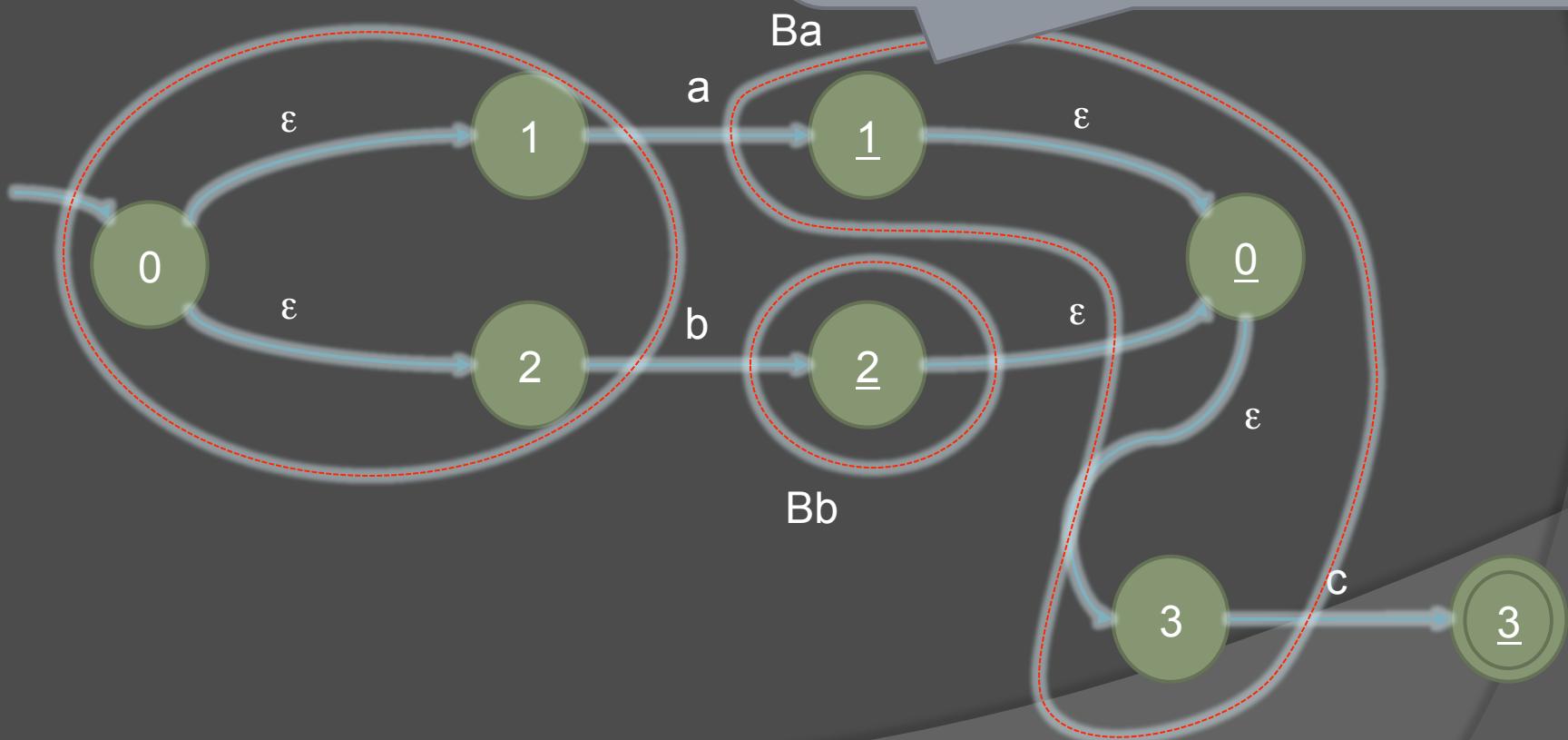
Example

2. For each transition leaving any state in A, **combine all possible destinations on each event e** and create a new state B.

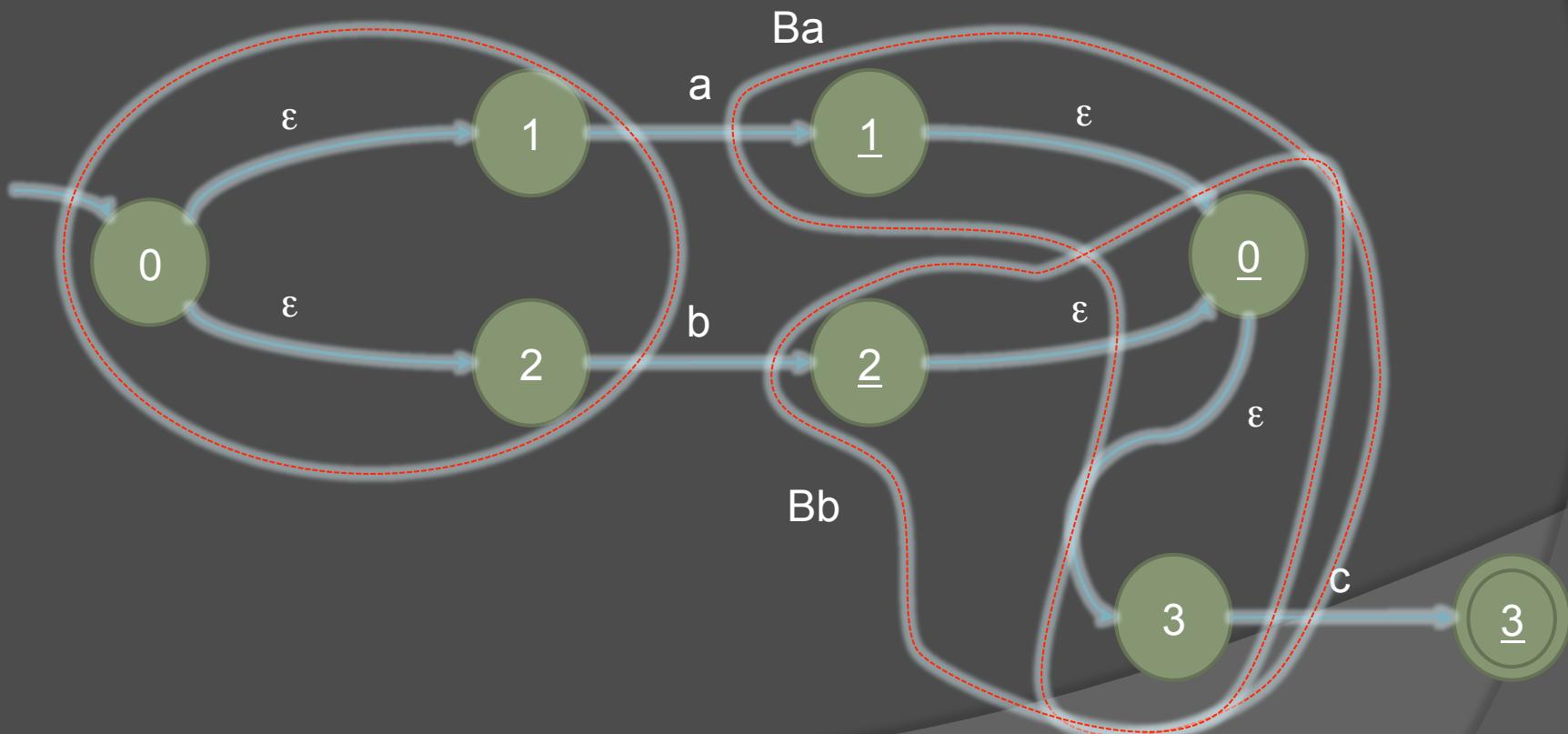


Example

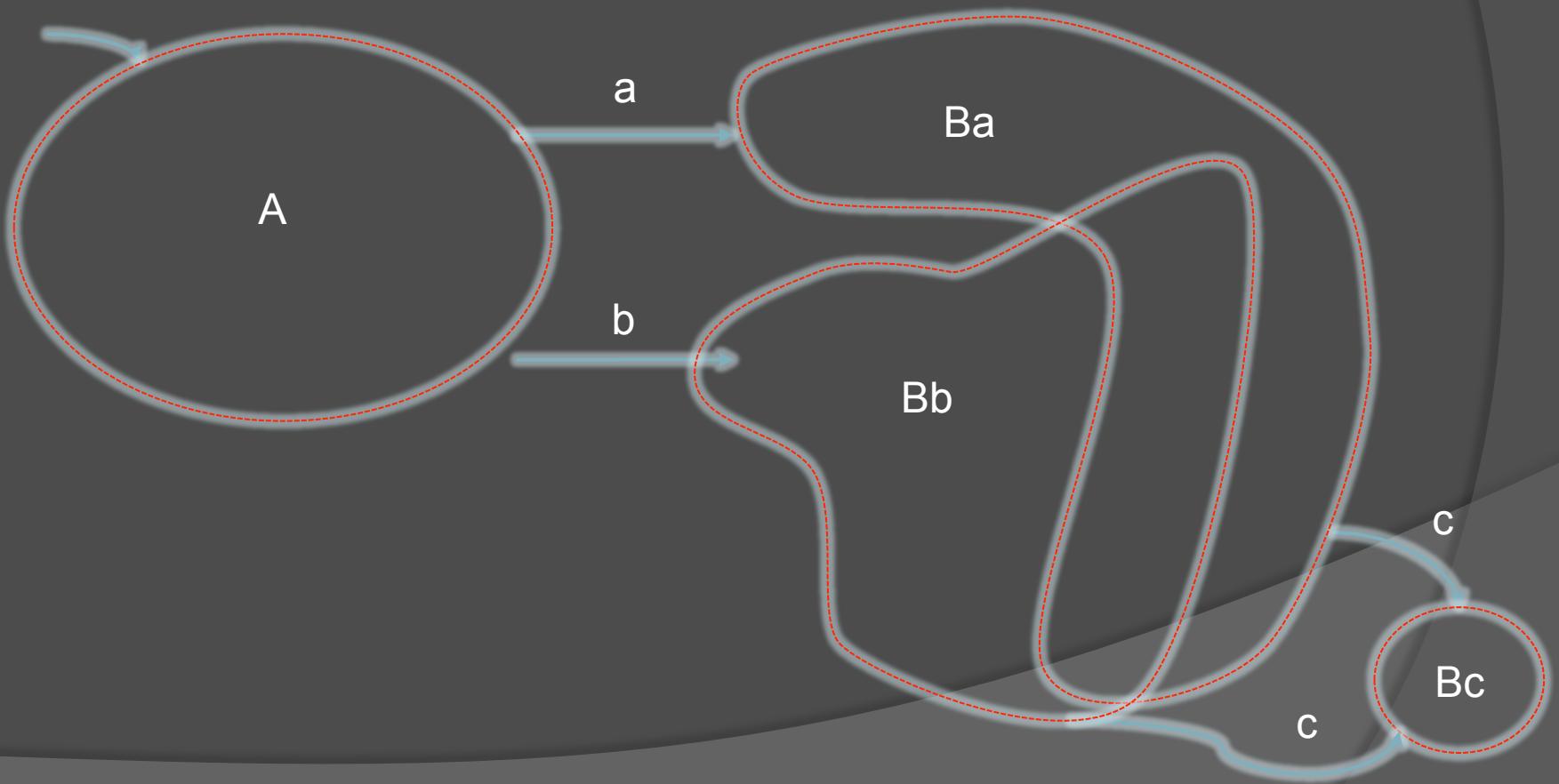
3. Restart process from 1 on each B (unless B has already been processed earlier).



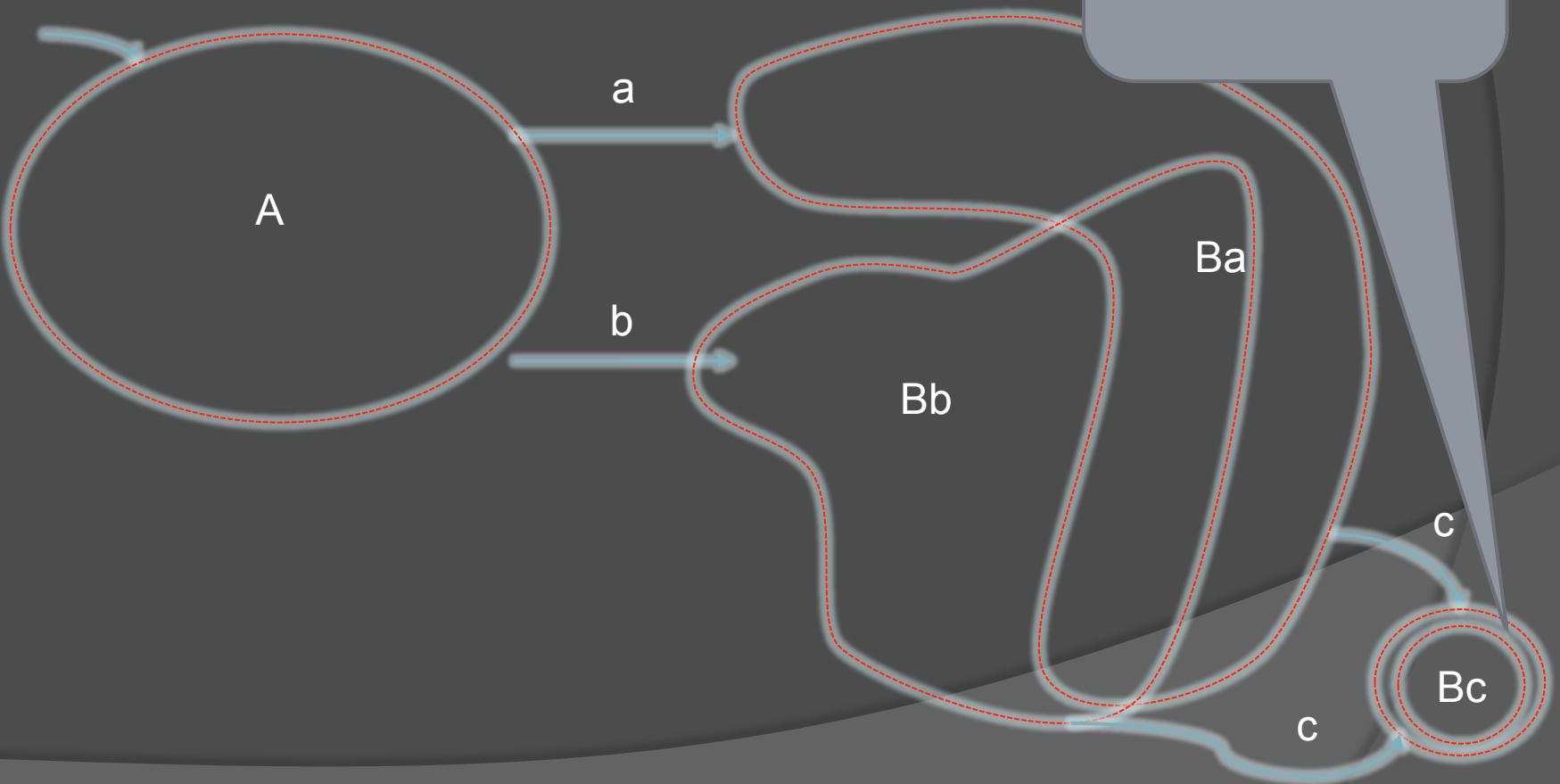
Example



Result



Result

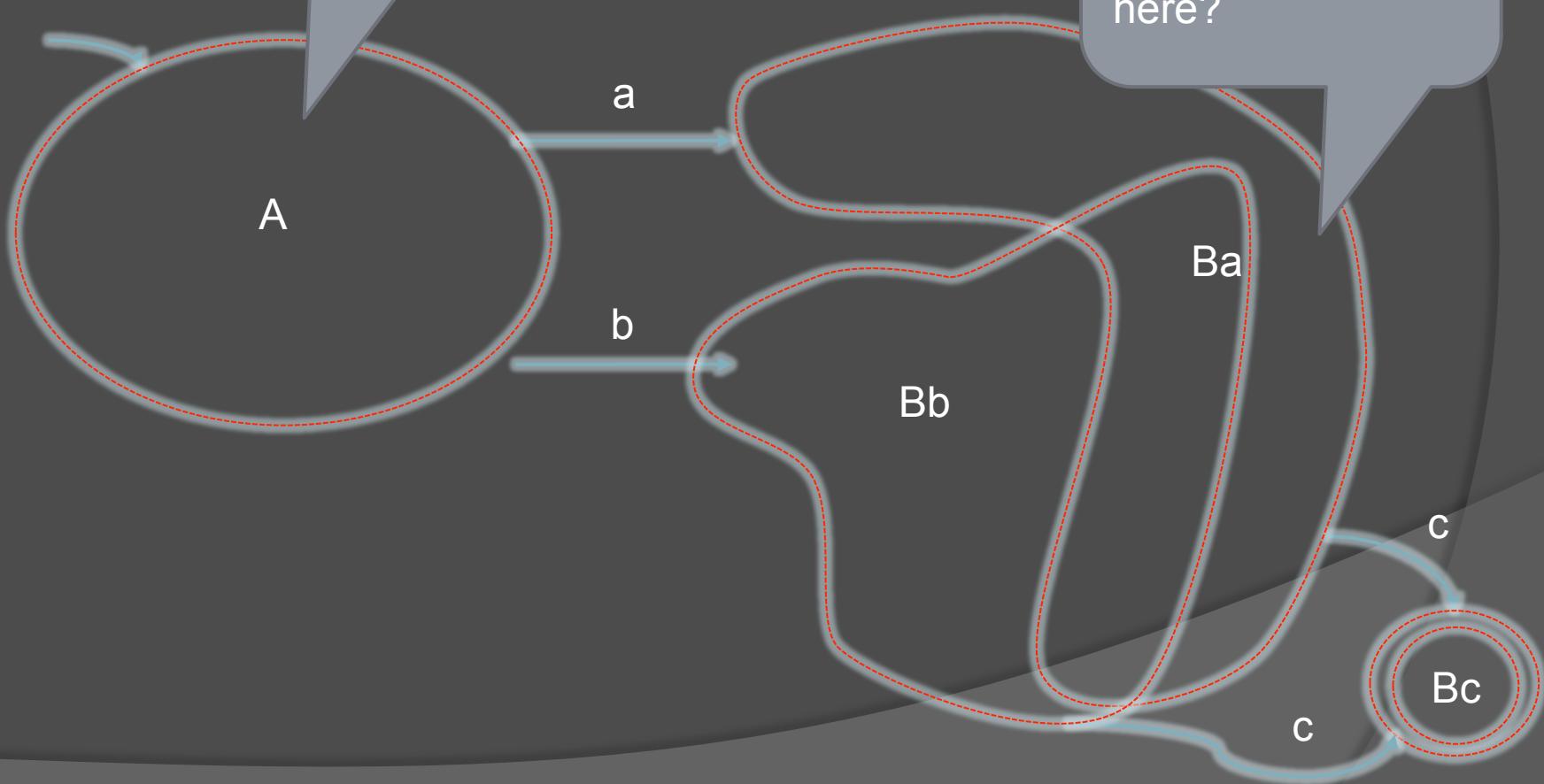


Determinisation

- ➊ Good news: the algorithm is provided for your assignment!

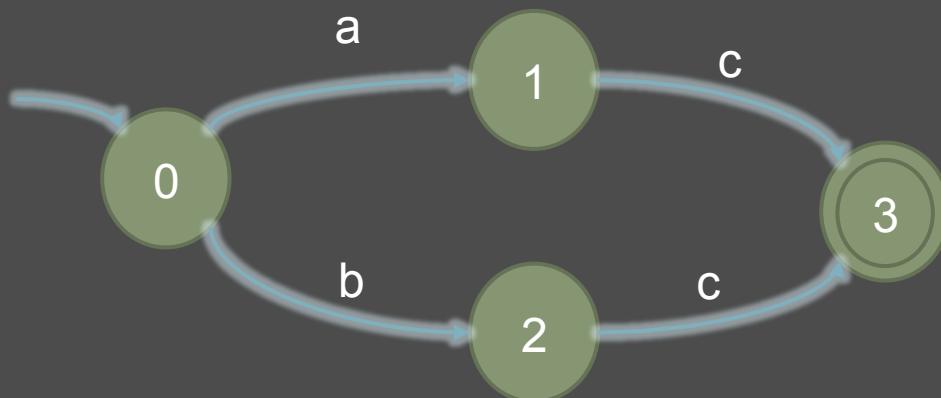
P

How should monitor react if observing “c” here?

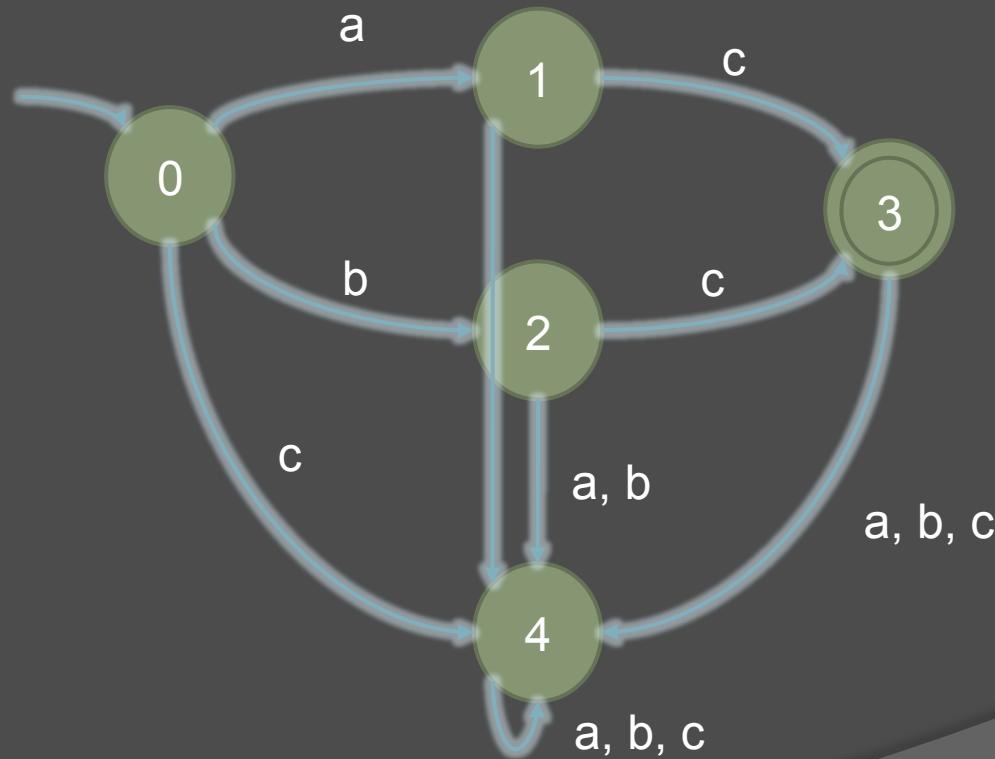


How should monitor react if observing “a” here?

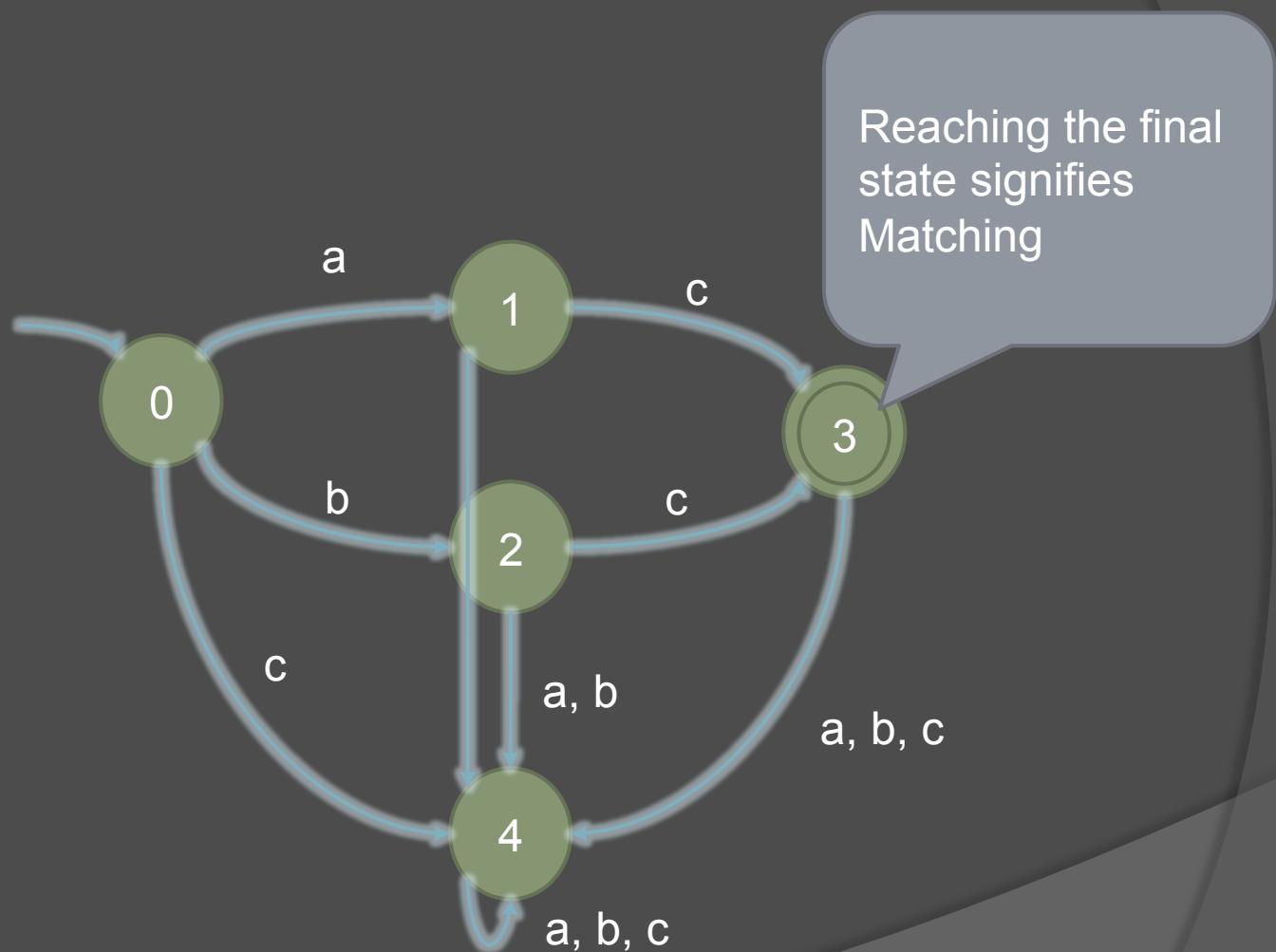
Totality



Totality



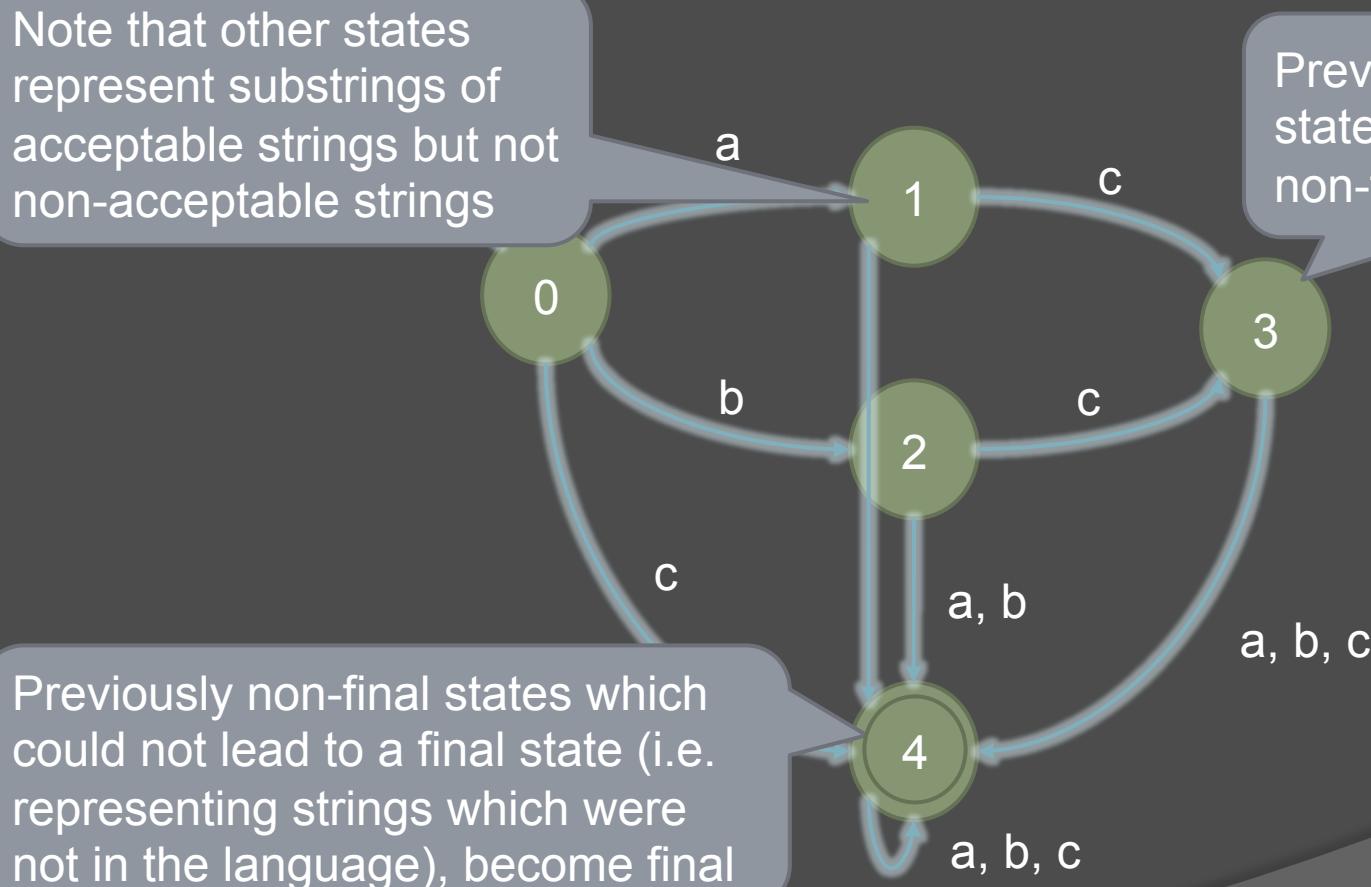
Violation



Complement if you to flag Non-Matching

Note that other states represent substrings of acceptable strings but not non-acceptable strings

Previously final states become non-final



Exercises

- Fill in the function `sequence()` in class `NFA.java` which connects two automata sequentially together