Runtime Verification
From theory to practice and back

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

January 2014

Part I

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 (!e means any event except for e):
  - `(login; (read+write)*; logout)*; (read+write)`
  - `(!approveAccount)*; transfer`
- Examples of specifications of good behaviour (matching prefixes of the regular expression):
  - `(login; (read+write)*; logout)*`
  - `!transfer*; approveAccount; !transfer*; 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)

R sions

Matches any event

RE ::= ? (Any) (End)

| a (

Matches the event a

| !a (A

| RE + RE (Ch

| RE ; RE (S

Matches any event but a

| RE* (R

| (RE) (Bracketed expression)

R ons

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

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

| a (Proposition)

| !a (All the prop

| RE + RE (Choice)

| RE ; RE (Sequence)

| RE* (Repetition)

| (RE) (Bracketed expression)

Nothing can match
(like reaching a bad state)

Regular Expressions

RE ::= ? (Any) (End)

| a (P1

| !a (All th positions except a)

| RE + RE (Choice)

| RE ; RE (S

| RE* (Repeti

| (RE) (Bracke

Matches either or

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 )*}

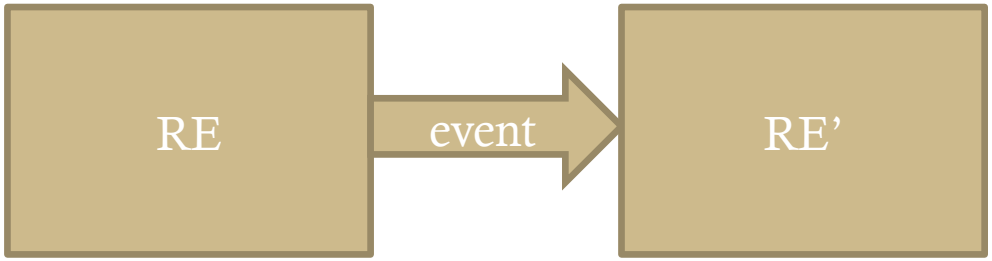
property foreach target (UserInfo u) not matching {
    (?)* ; makeDisabled ; (!makeActive)* ; withdrawFrom
}
```

Part II

Monitoring Regular Expressions using Residuals

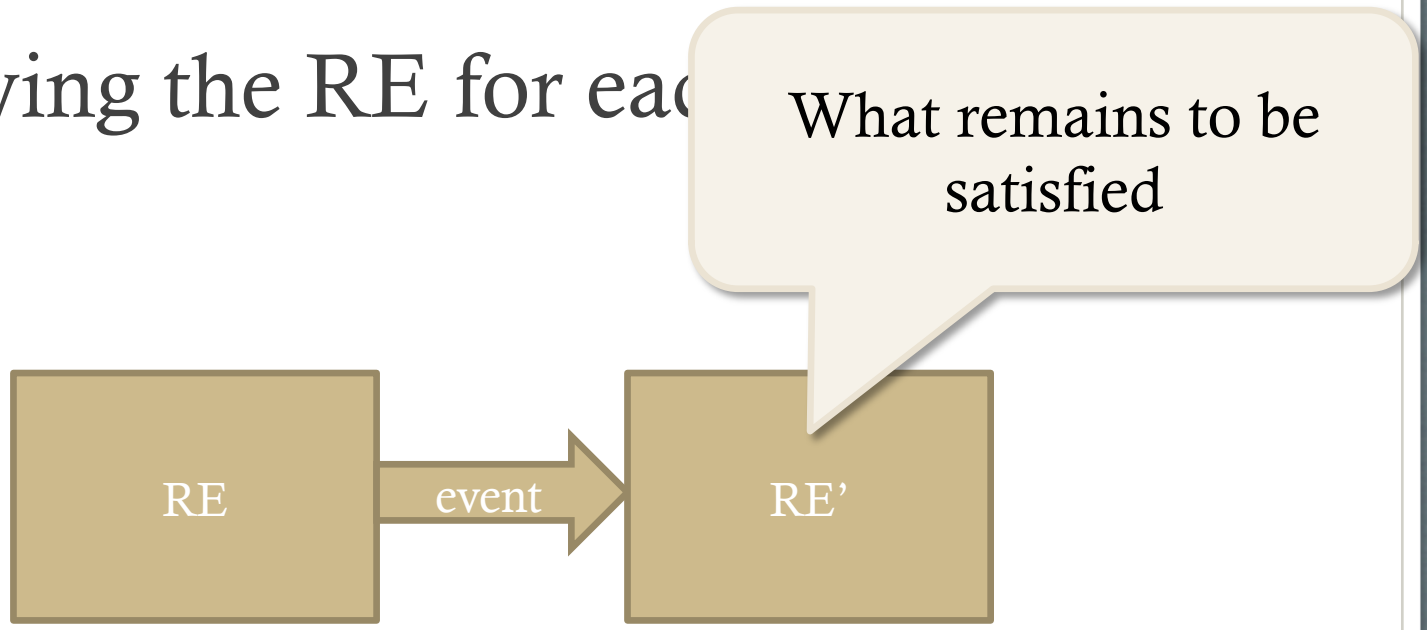
Residuals

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

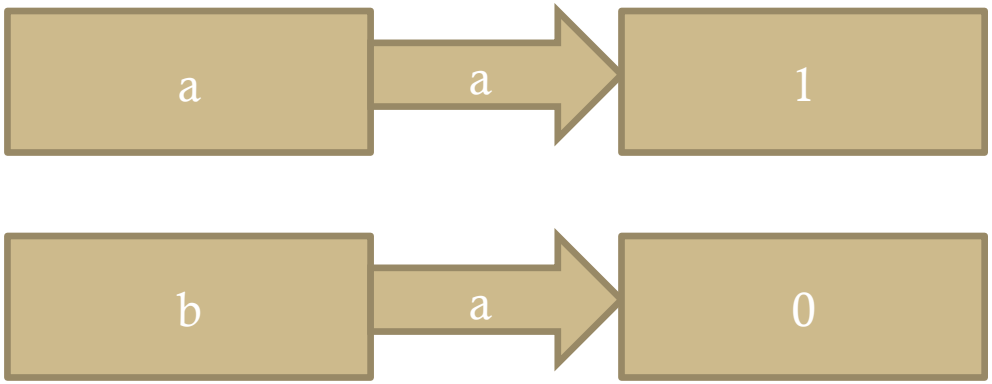


Residuals

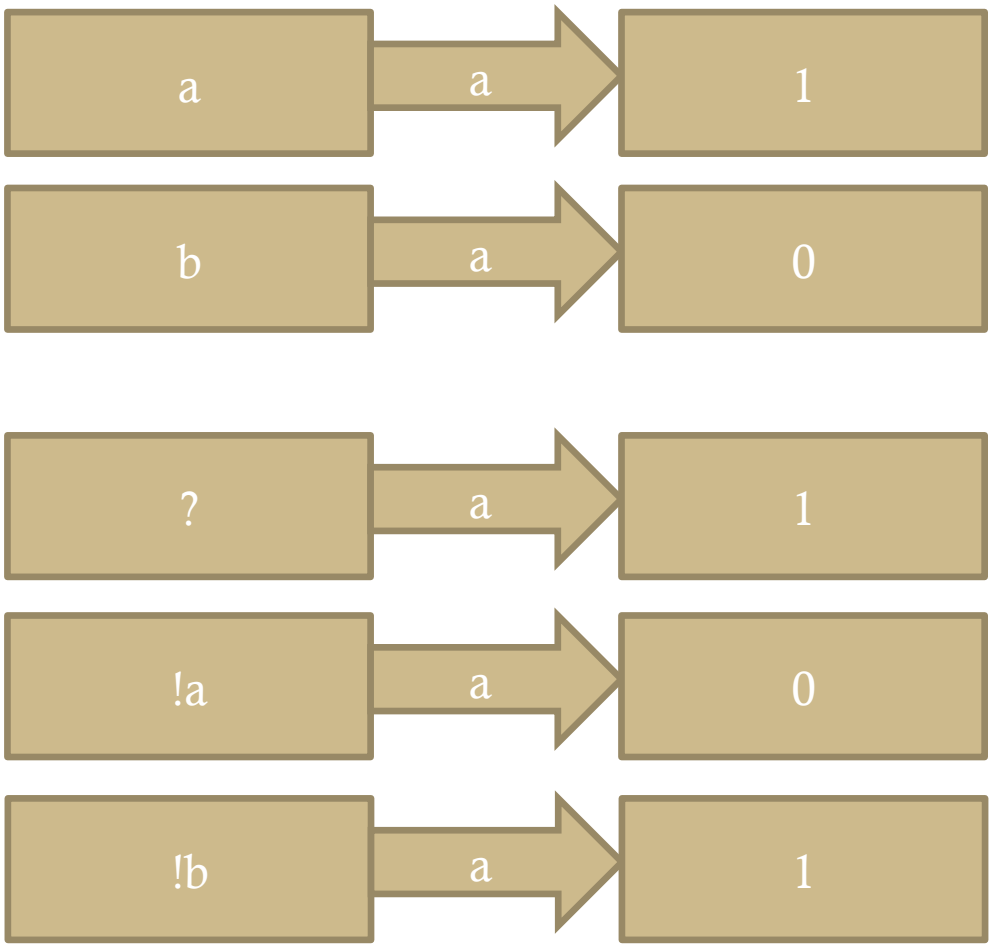
One way of monitoring REs is by modifying the RE for eac event



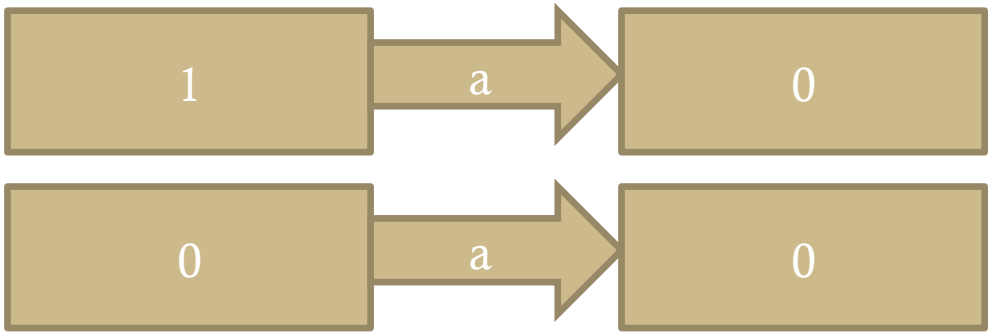
Residuals - Basic



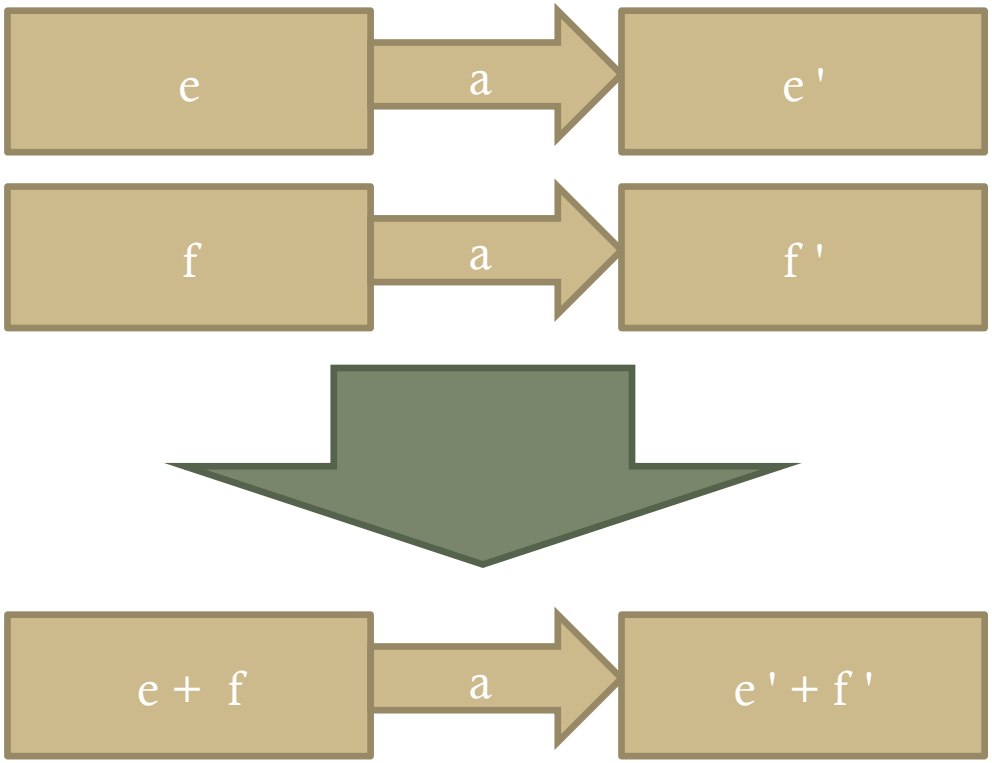
Residuals - Basic



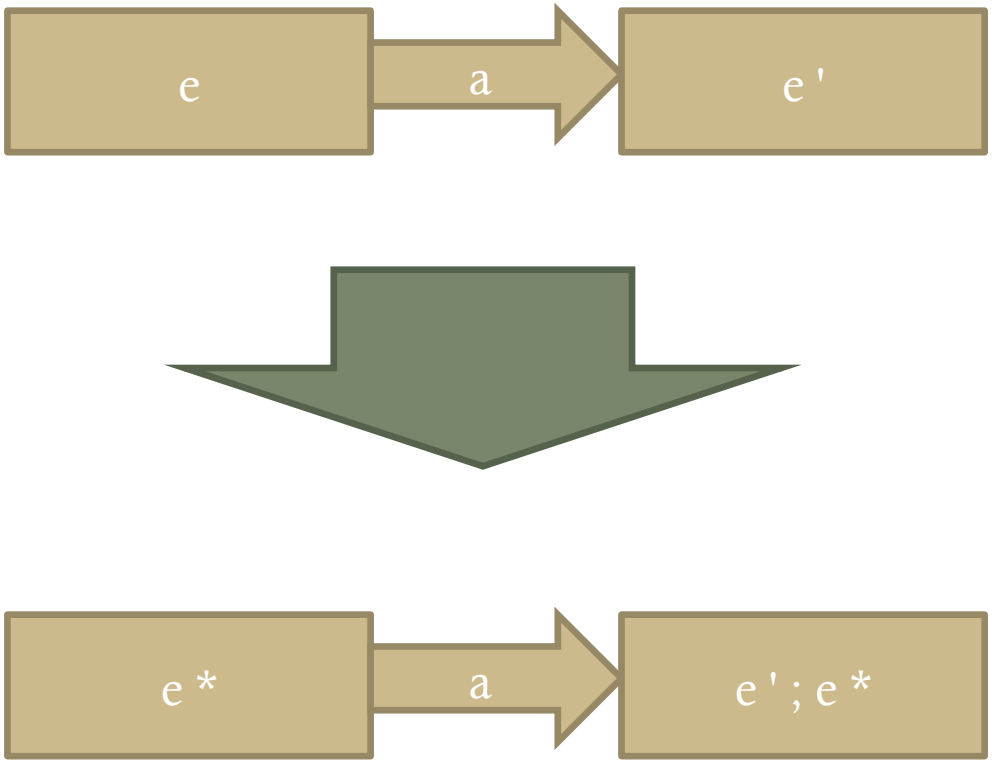
Residuals - Basic



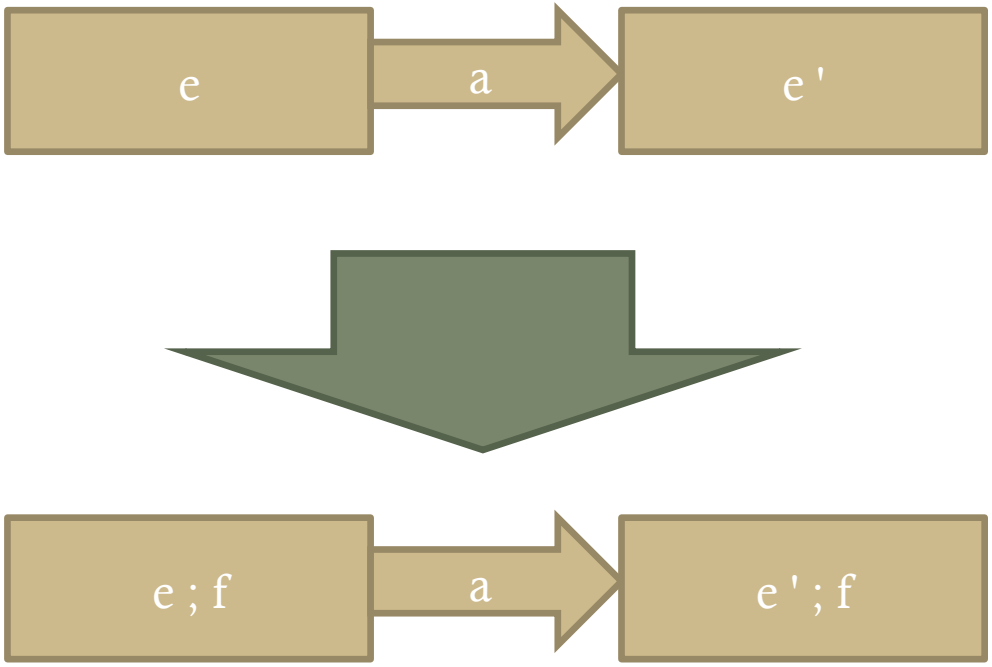
Residuals – Or



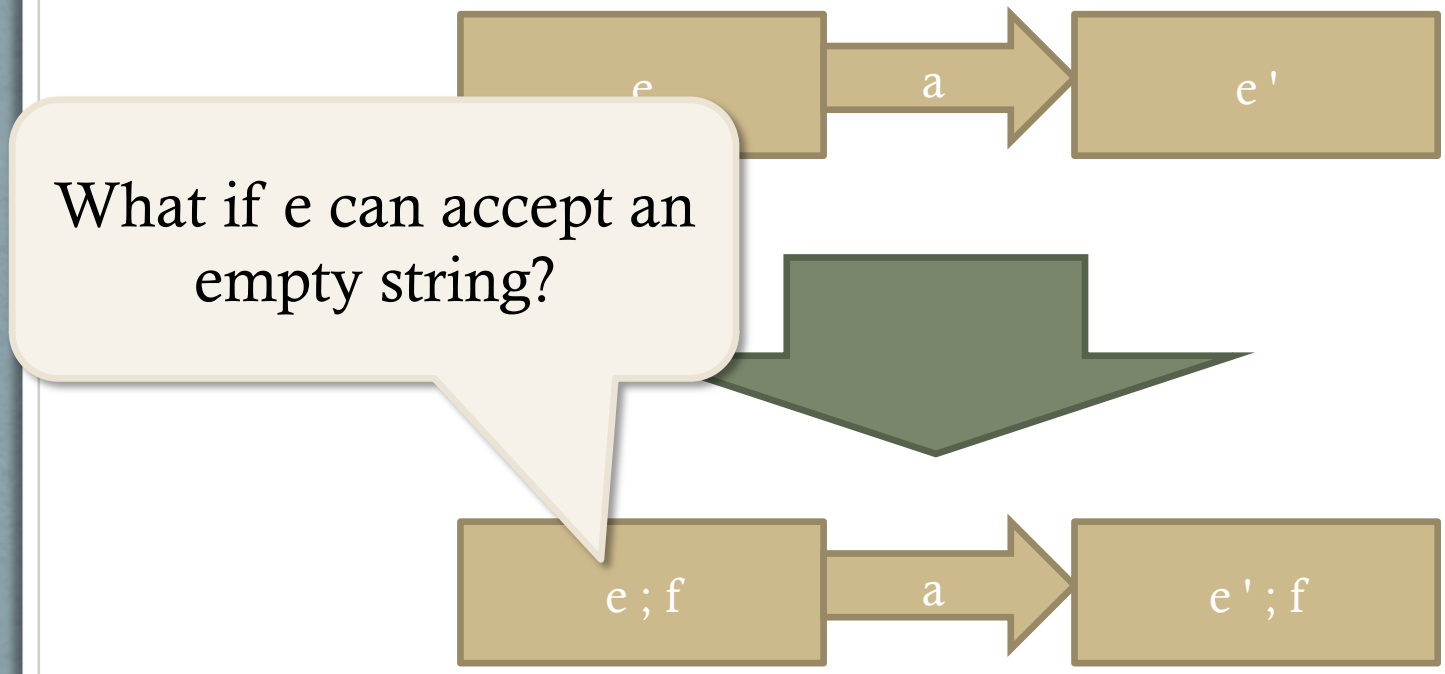
Residuals – Star



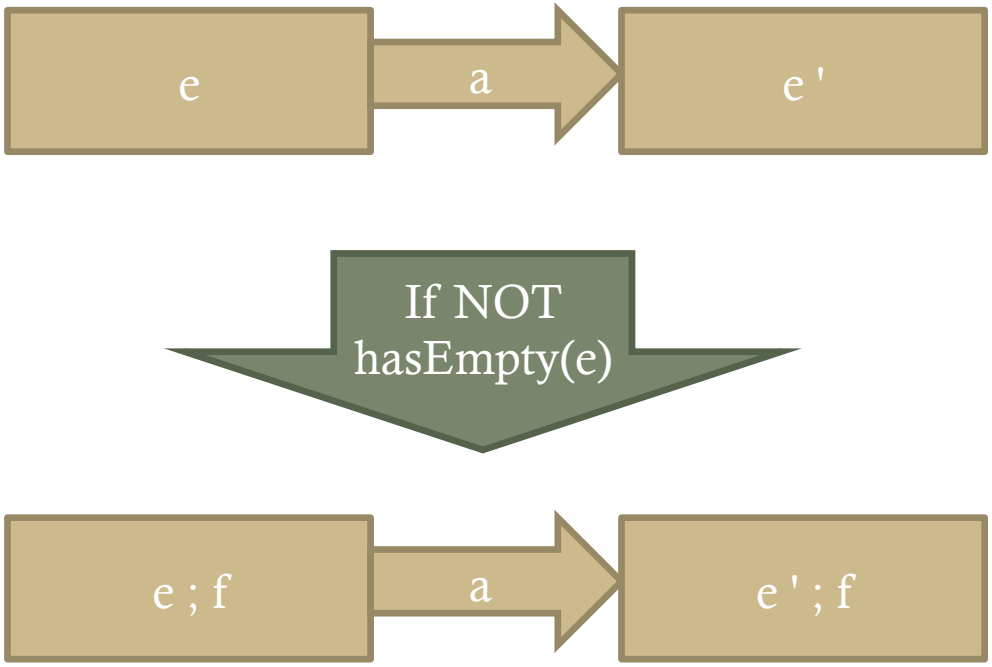
Residuals – Sequence



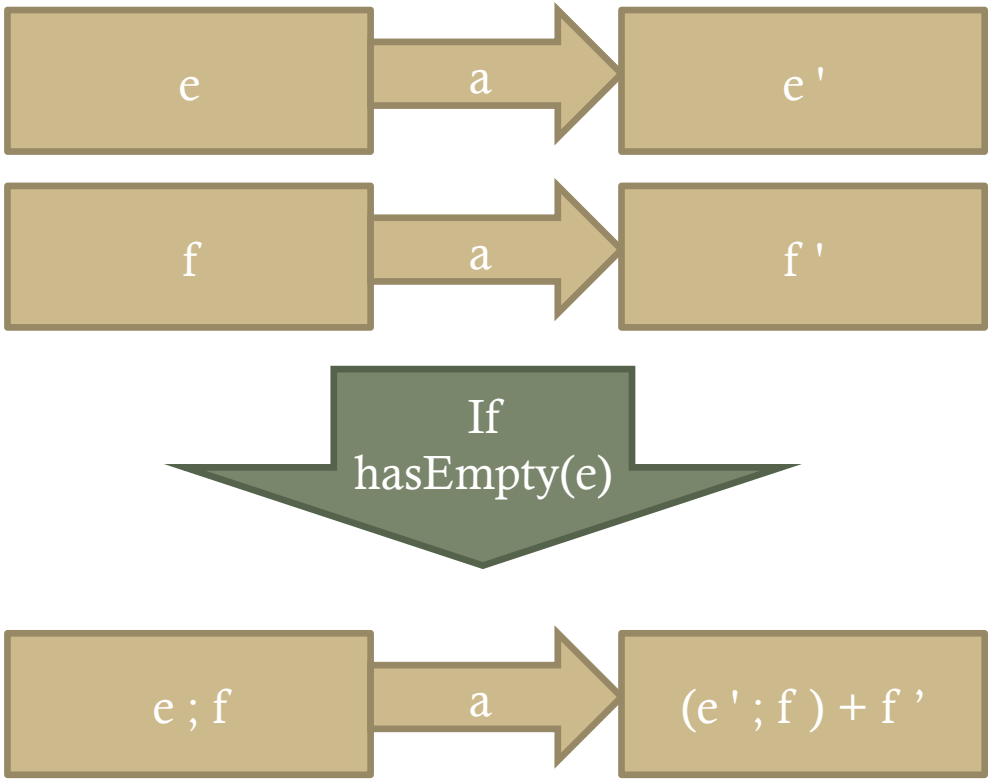
Residuals – Sequence



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)

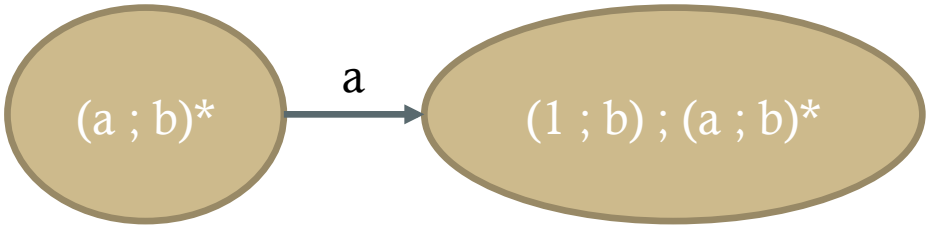
hasEmpty (RE1 ; RE2) = hasEmpty (RE1) and hasEmpty (RE2)

hasEmpty (RE*) = true

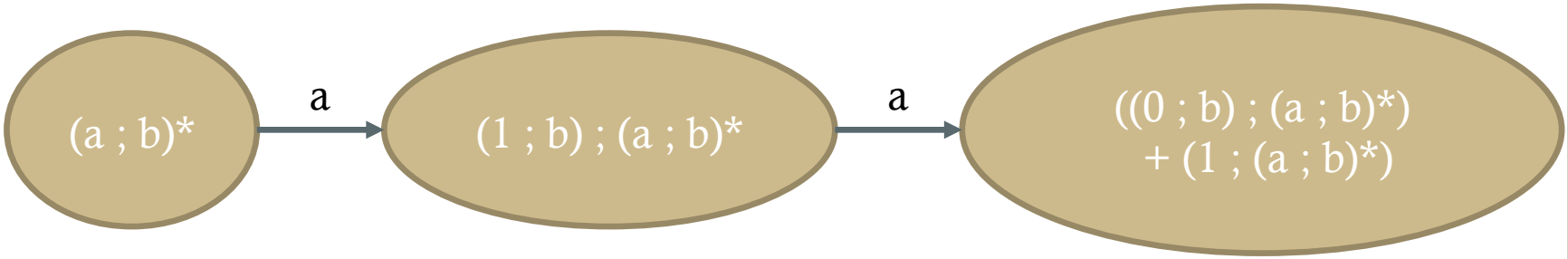
Residuals Example

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

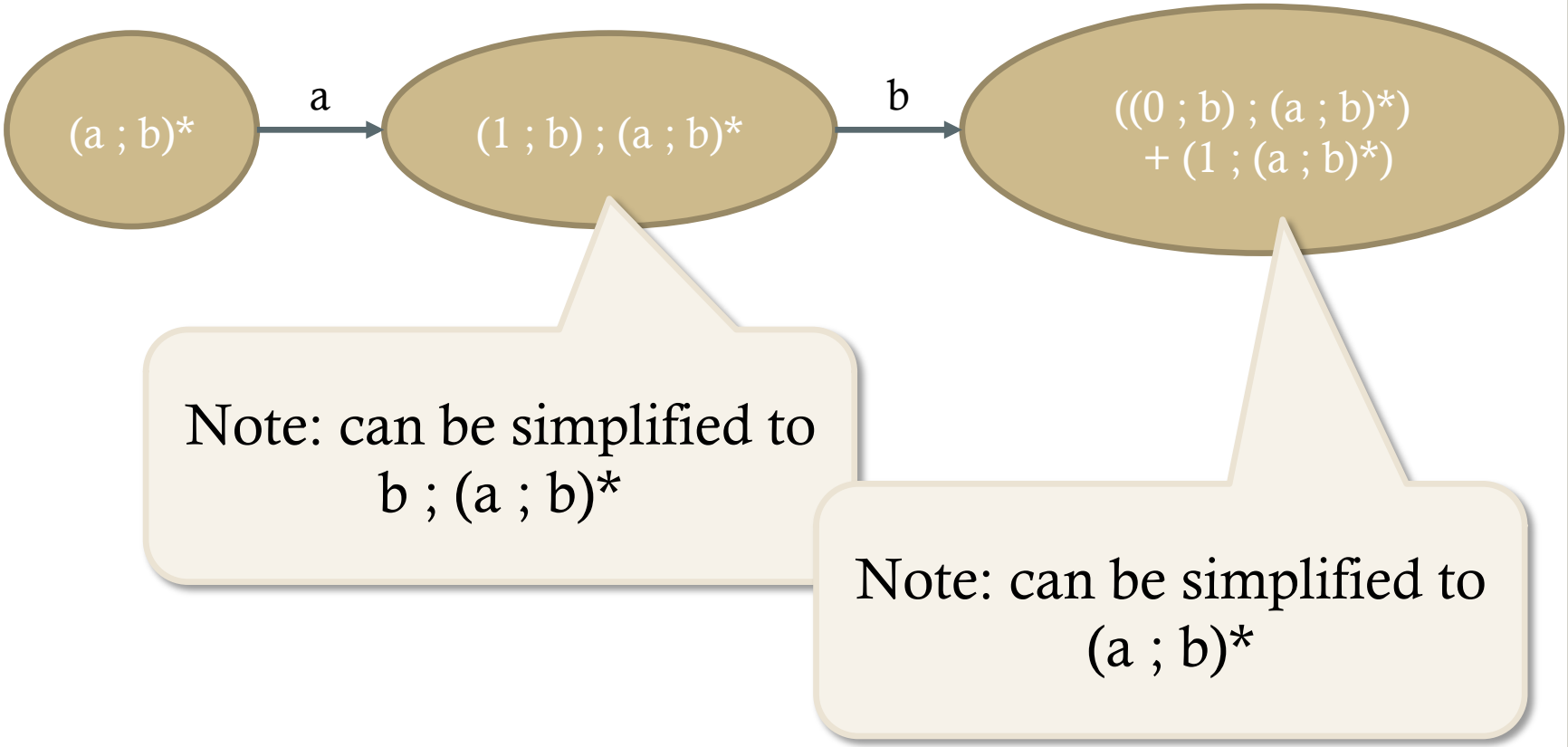
Residuals Example 1



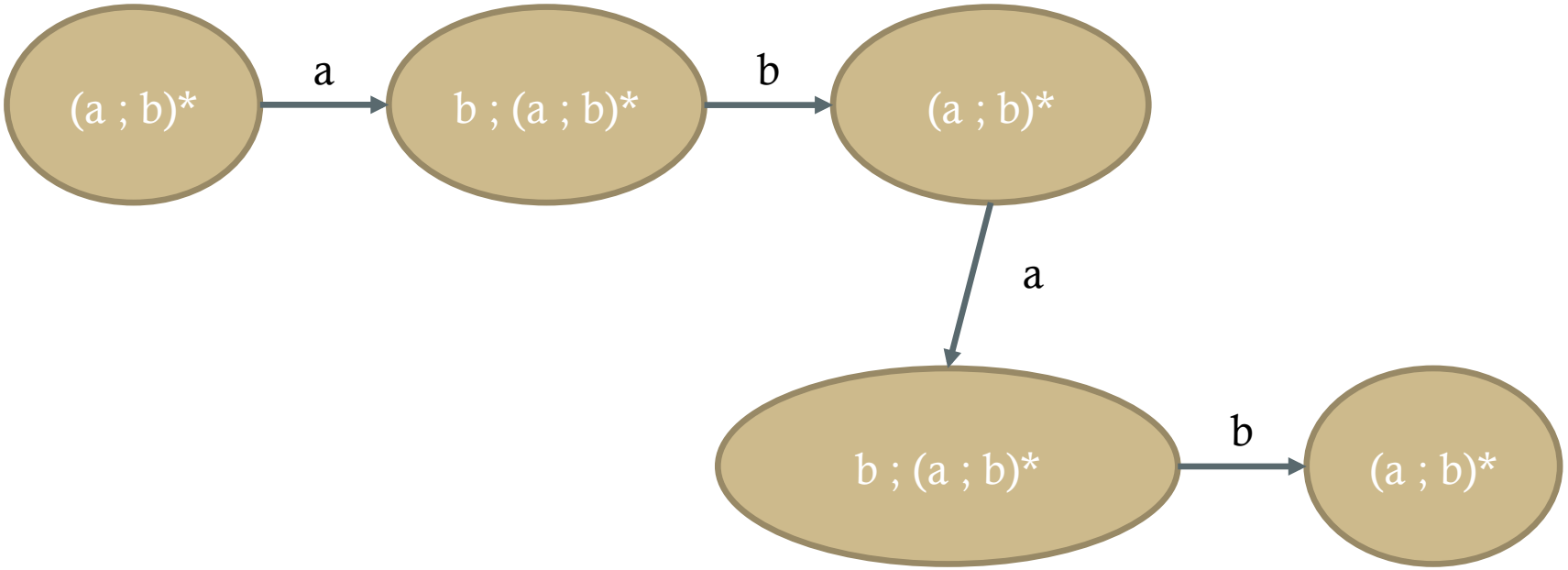
Residuals Example 1



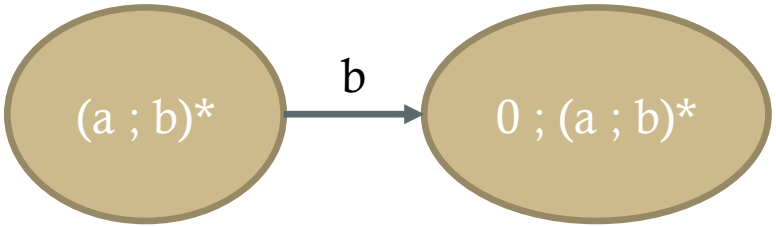
Residuals Example 1



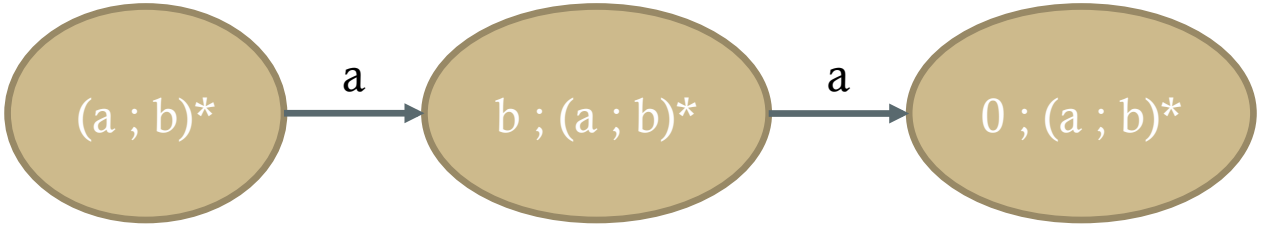
Residuals Example 1



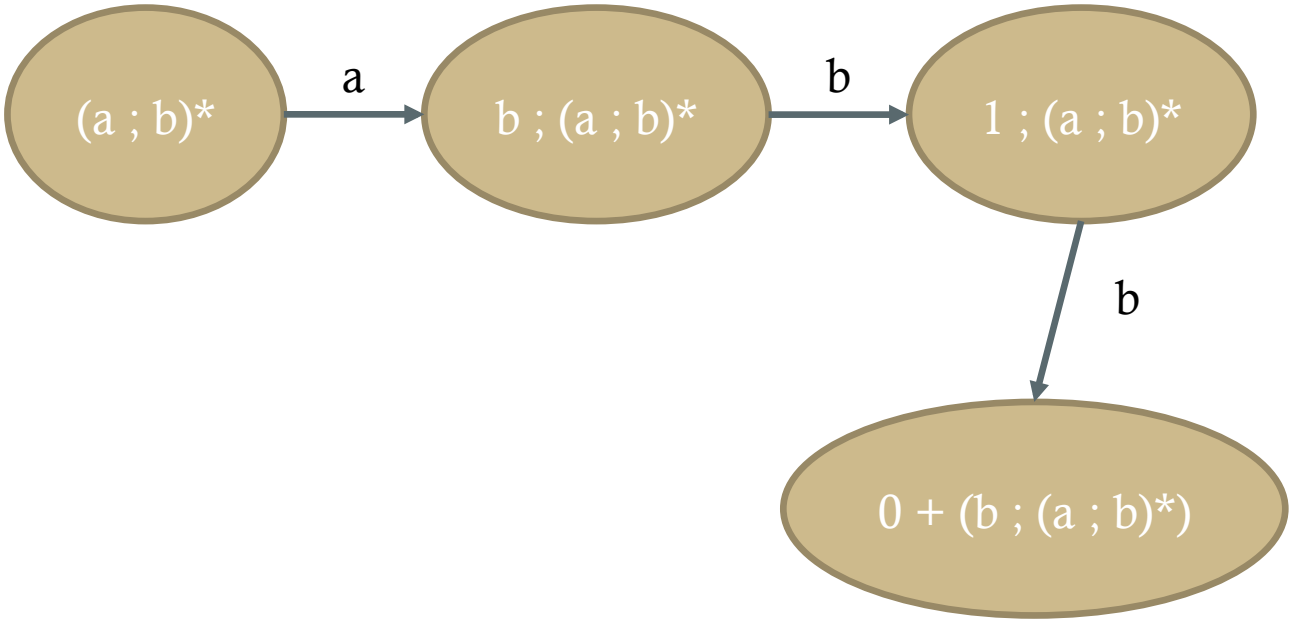
Residuals Example 2



Residuals Example 3



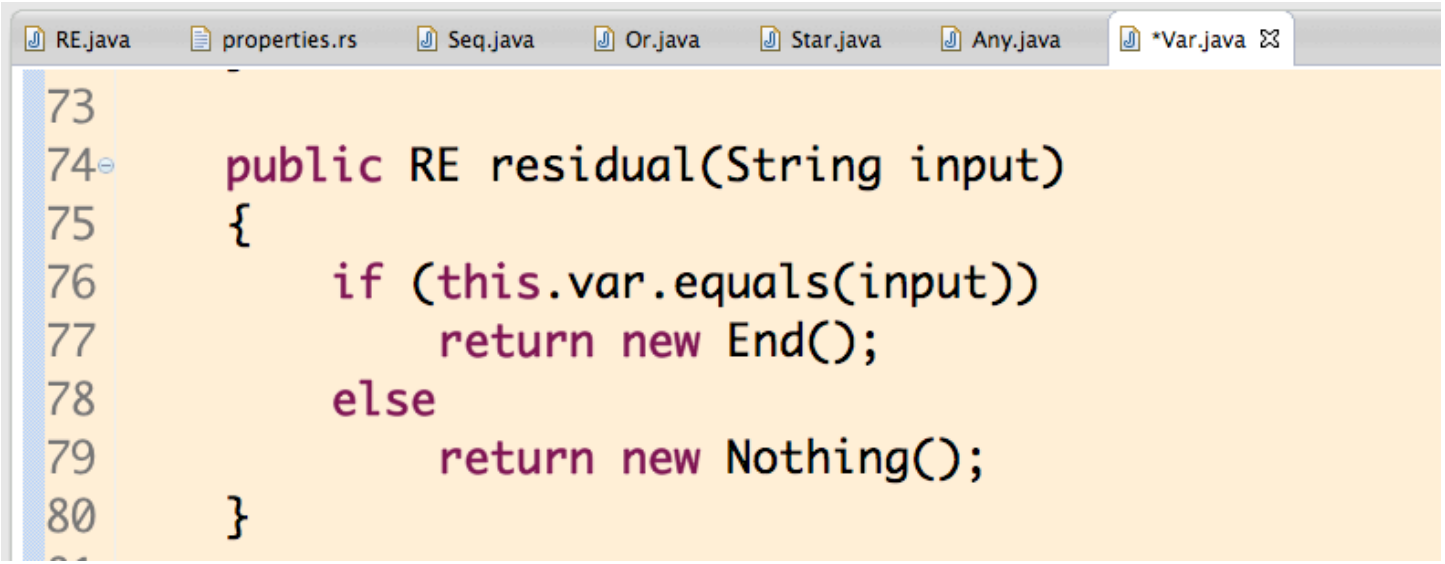
Residuals Example 4



Exercises

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

Example


RE.java
properties.rs
Seq.java
Or.java
Star.java
Any.java
*Var.java
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 }

Simplifying REs

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

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

Implemented Simplification Rules

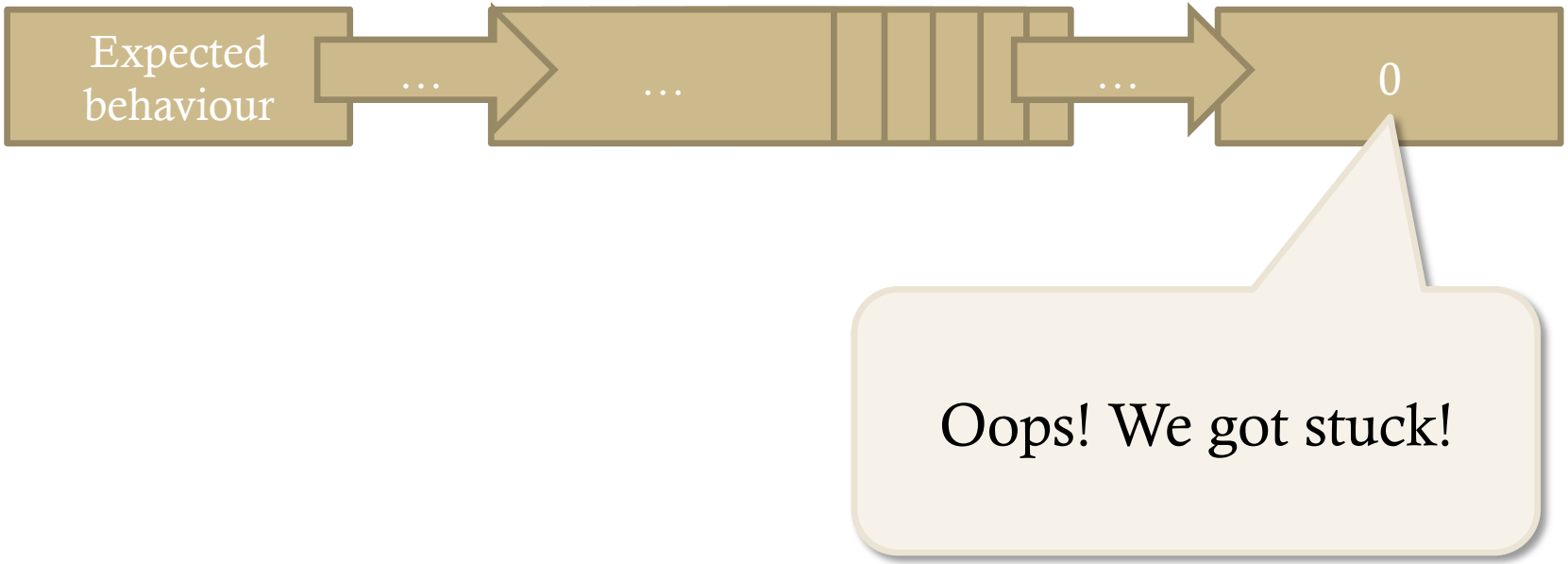
- 1 ; RE = RE
- 0 ; RE = 0
- 0 + RE = RE

Detecting a Violation

- What does it mean to detect a violation?

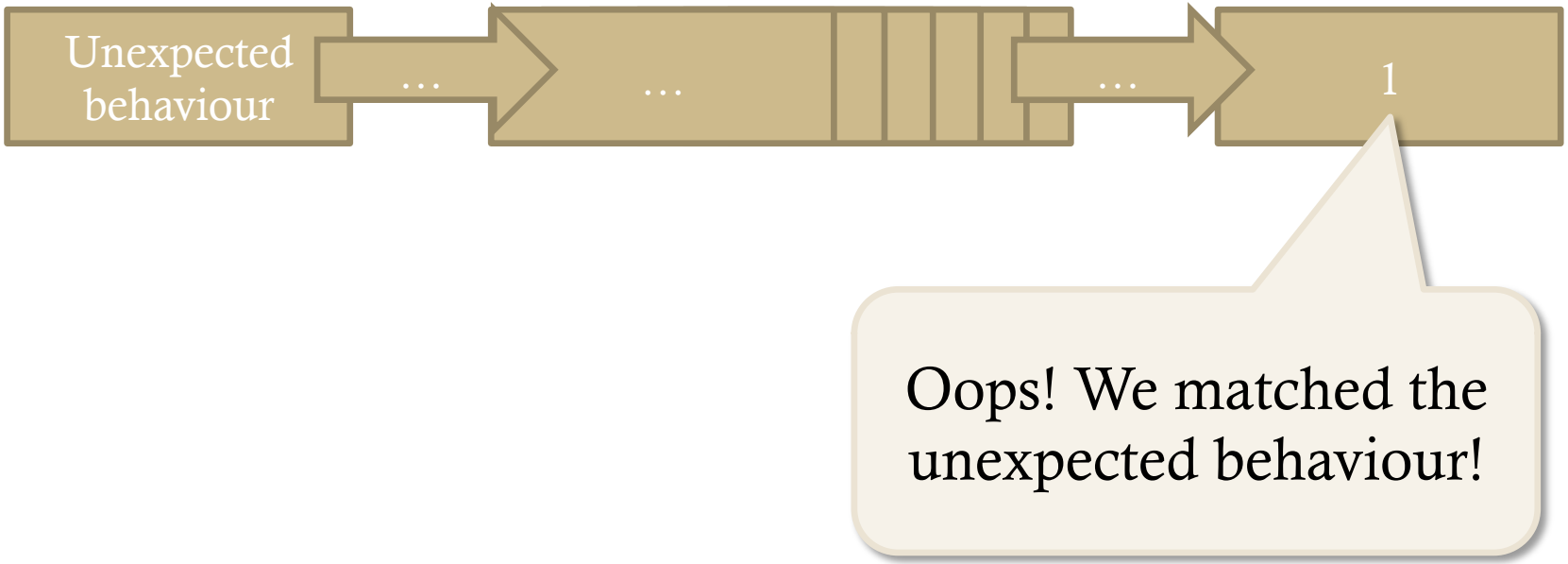
Detecting a Violation

- What does it mean to detect a violation?



Detecting a Violation

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



Part III

(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)

Part IIIa

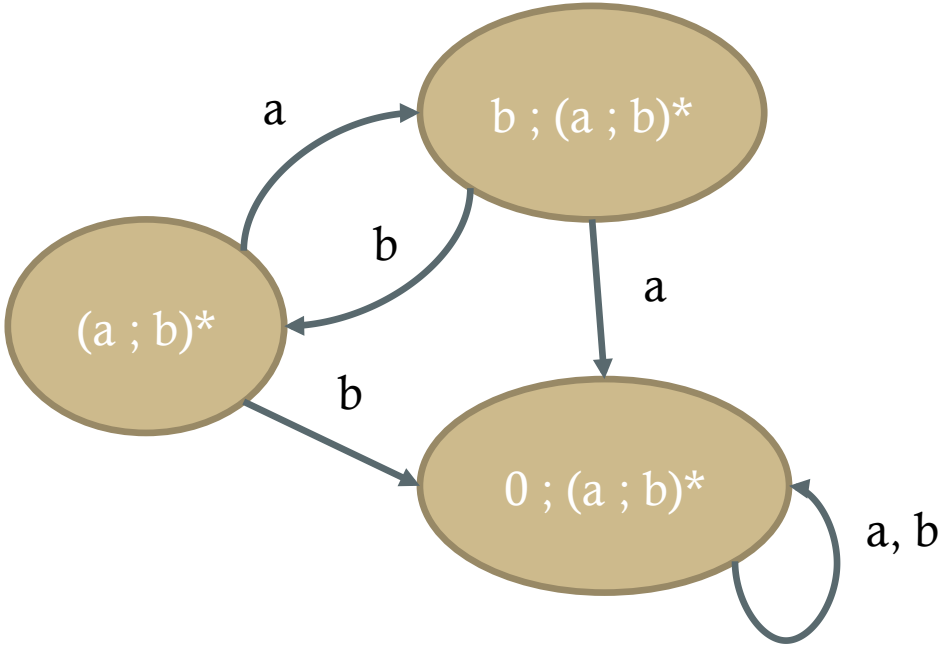
Generating Automata using Residuals

Generating an Automaton 1

- 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

- 0 + 0 + 0 + 0 + 0 + 0 + 0 … + (a;b)*
- The automaton becomes infinite

Caveats: REs can grow indefinitely

- 0 + 0 + 0 + 0 + 0 + 0 + 0 … + (a;b)*
- The automaton becomes infinite

- 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)

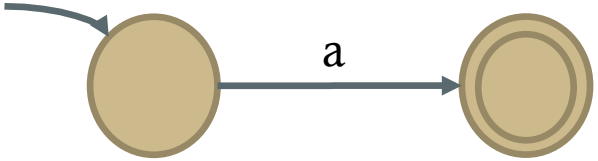
- Note: by associativity, idempotency, and symmetry, any RE can be converted into an RE in normal form

Part IIIb

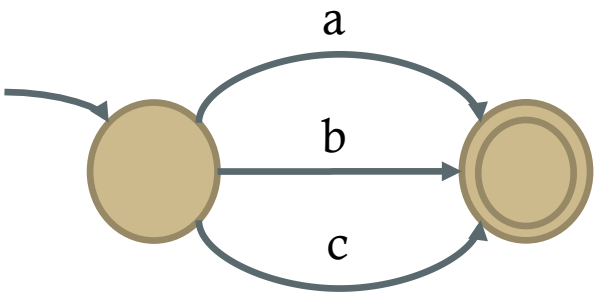
Generating Automata through a 1-to-1 operator mapping

Automata Equivalents

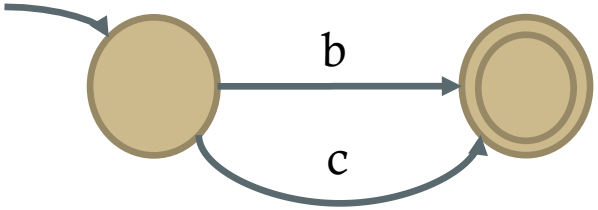
- a



- ?

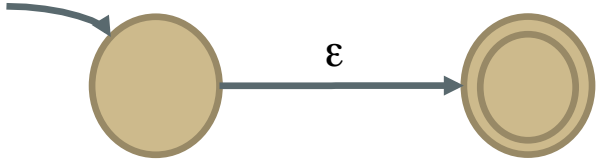


- !a



Automata Equivalents

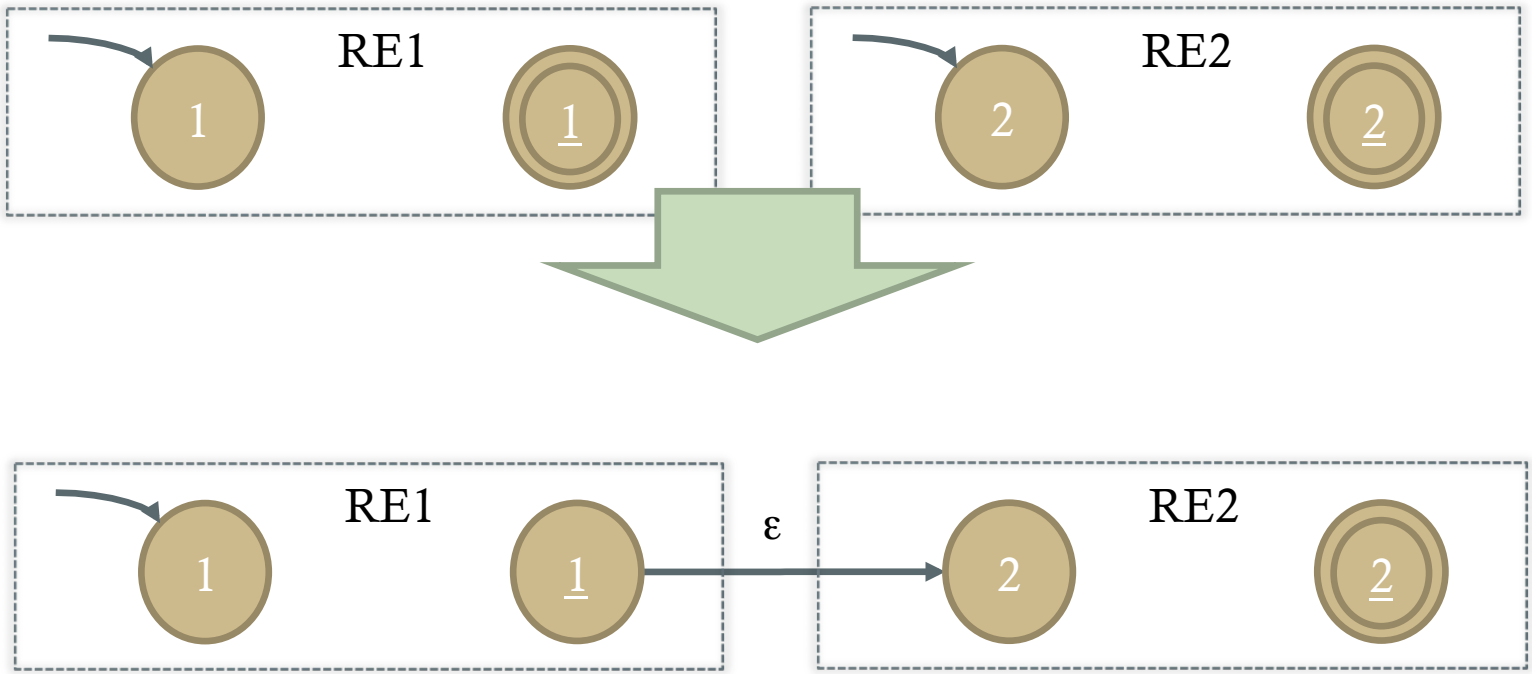
- 1



- 0

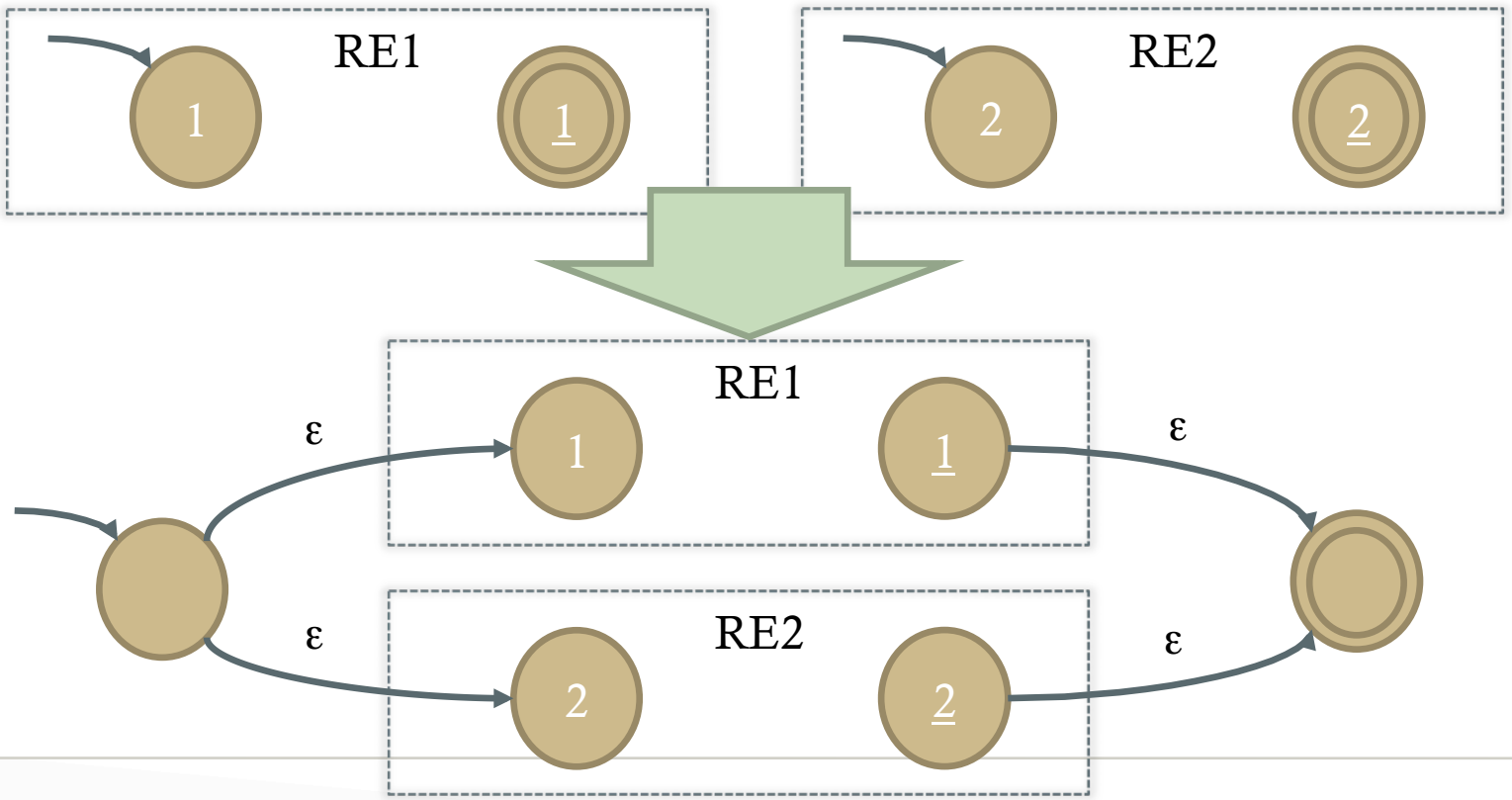
Automata Equivalents

- RE1 ; RE2



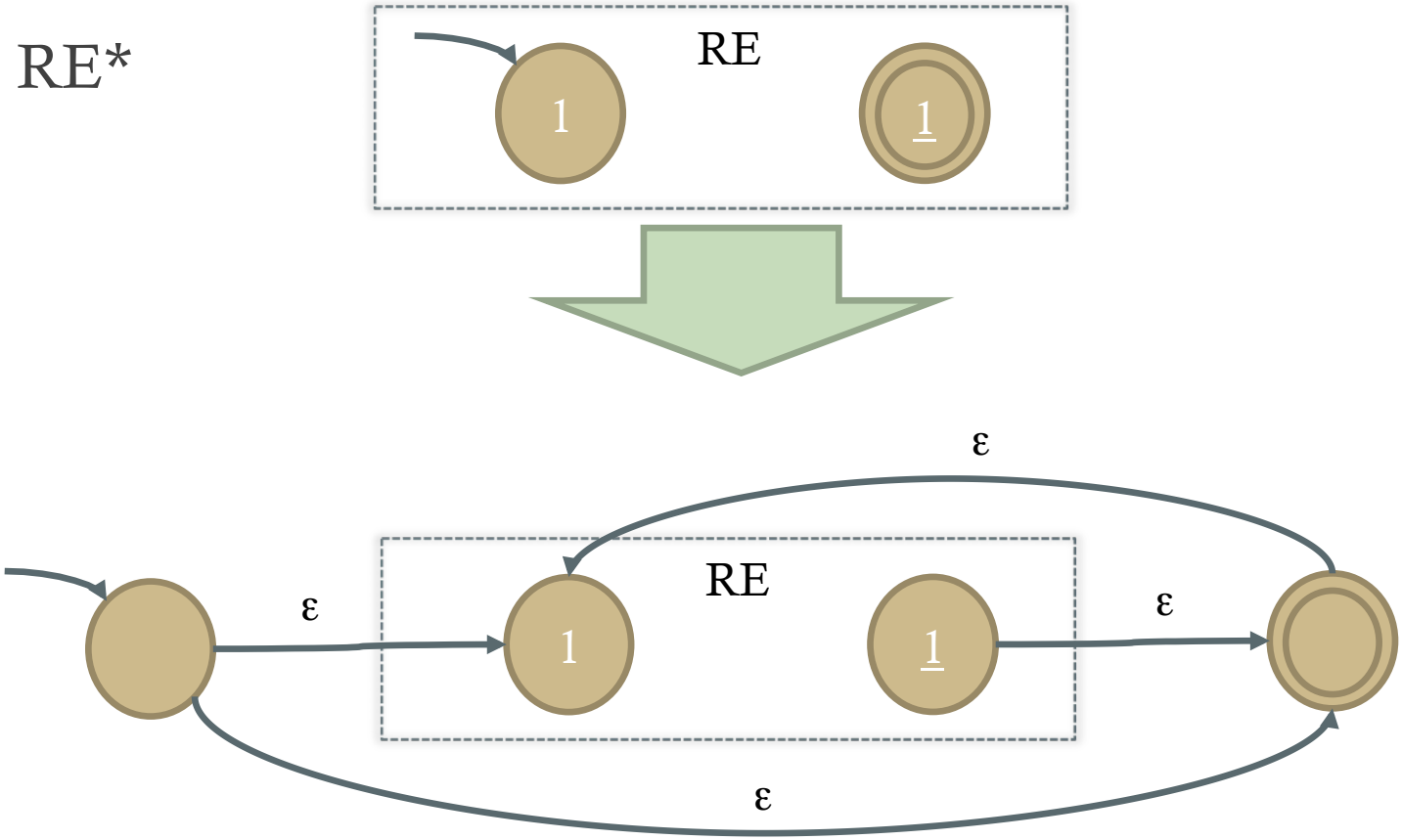
Automata Equivalents

- RE1 + RE2



Automata Equivalents

- RE*

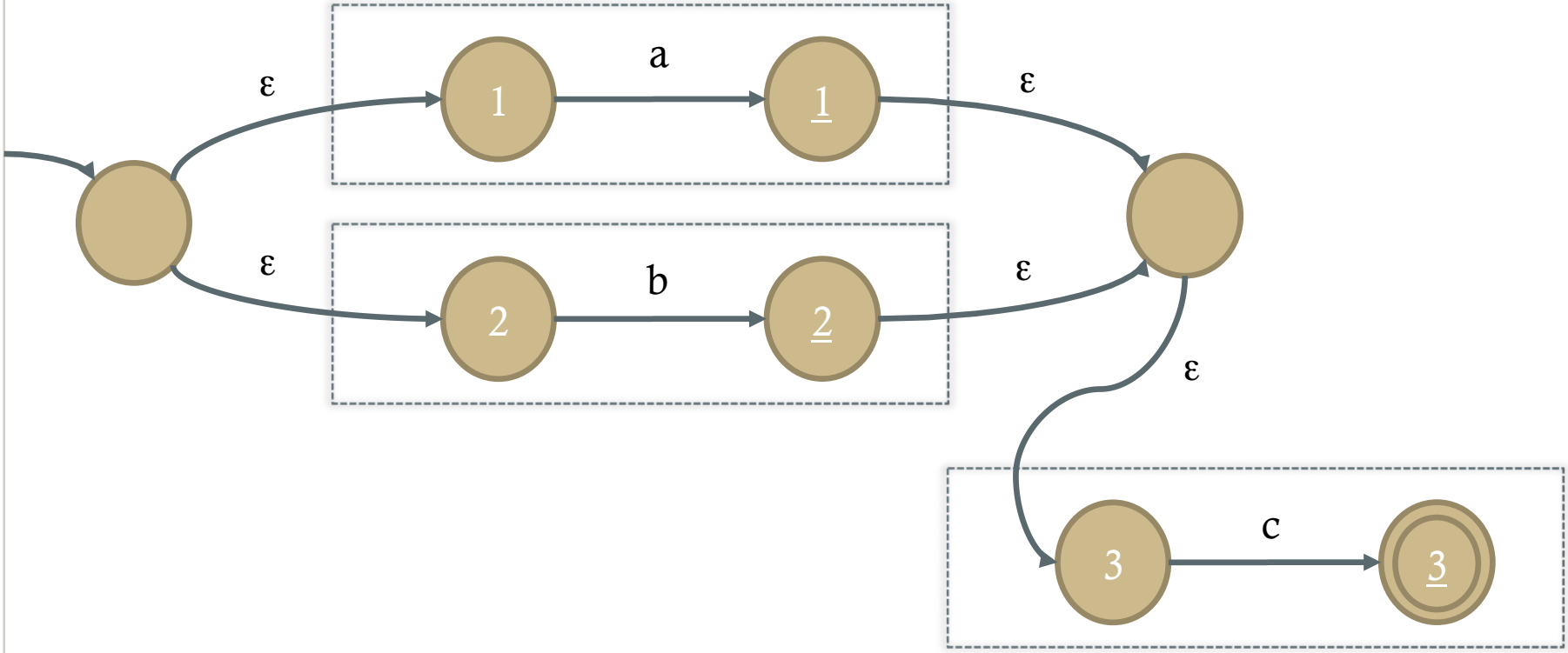


Example

- (a + b) ; c

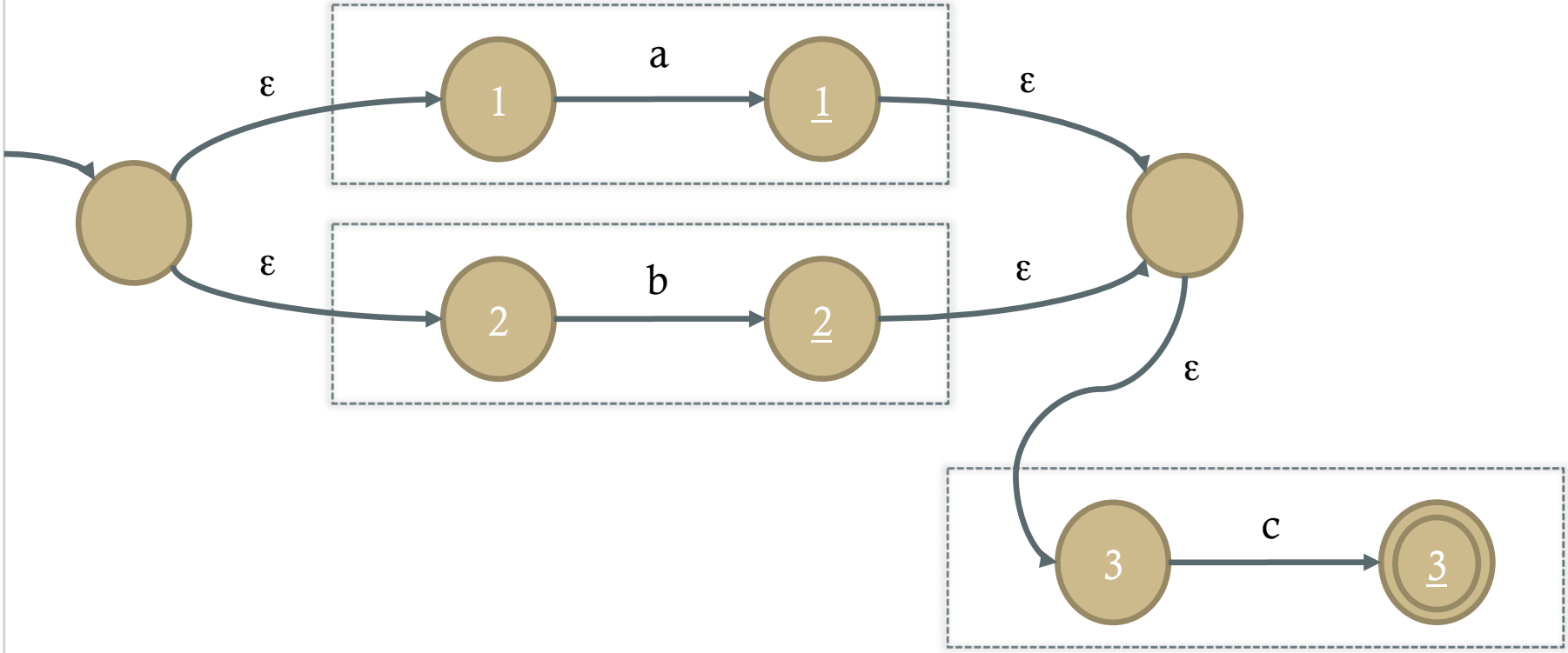
Example

- (a + b) ; c



Problem

- Monitors have to be deterministic!

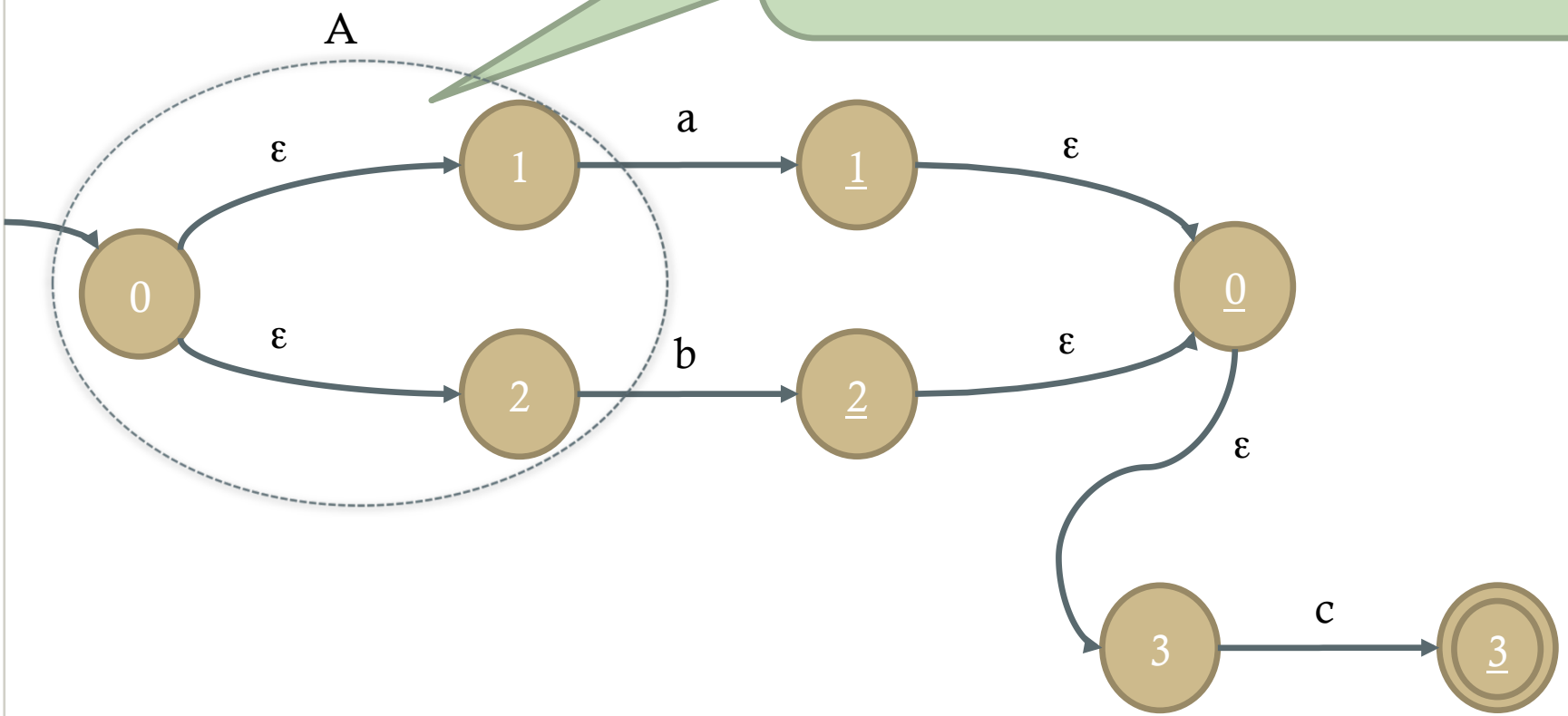


Determinisation in a Nutshell

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).

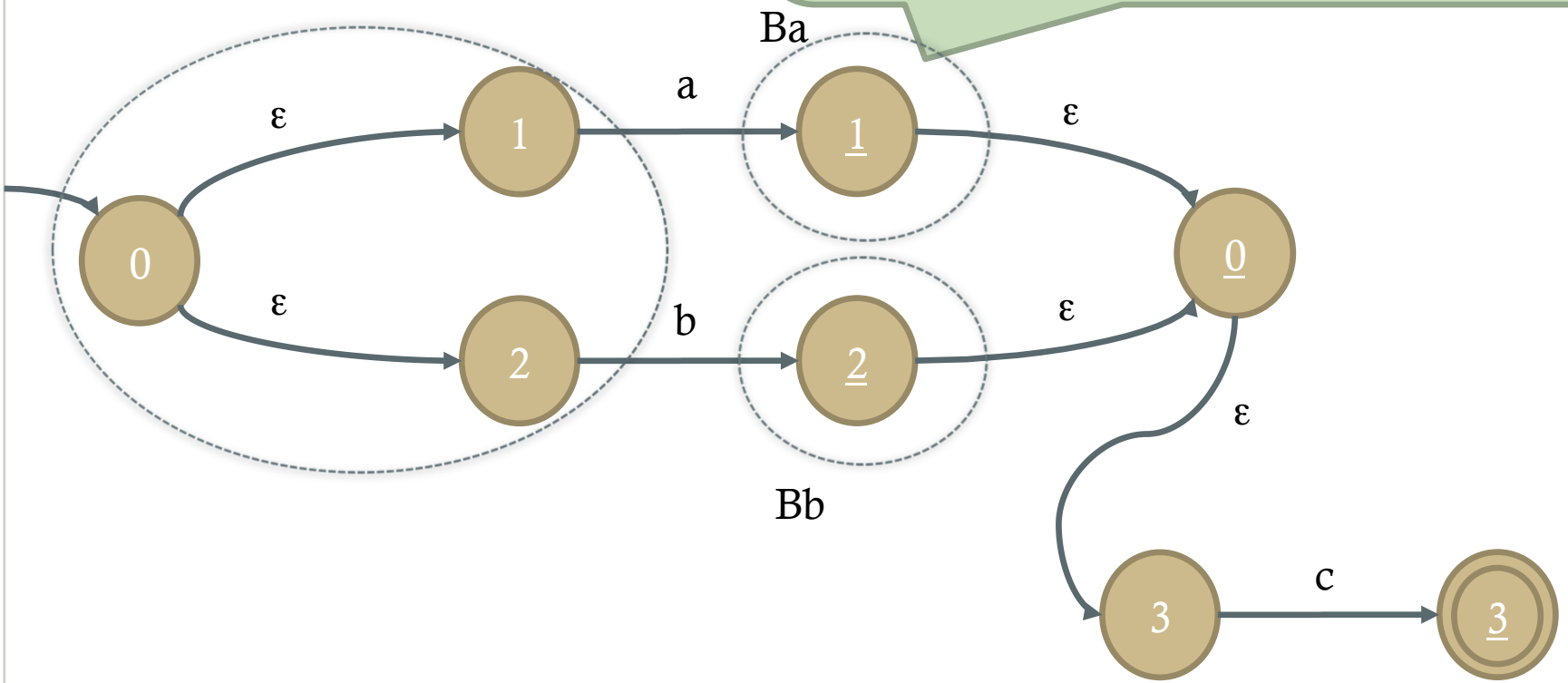
Ex

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**



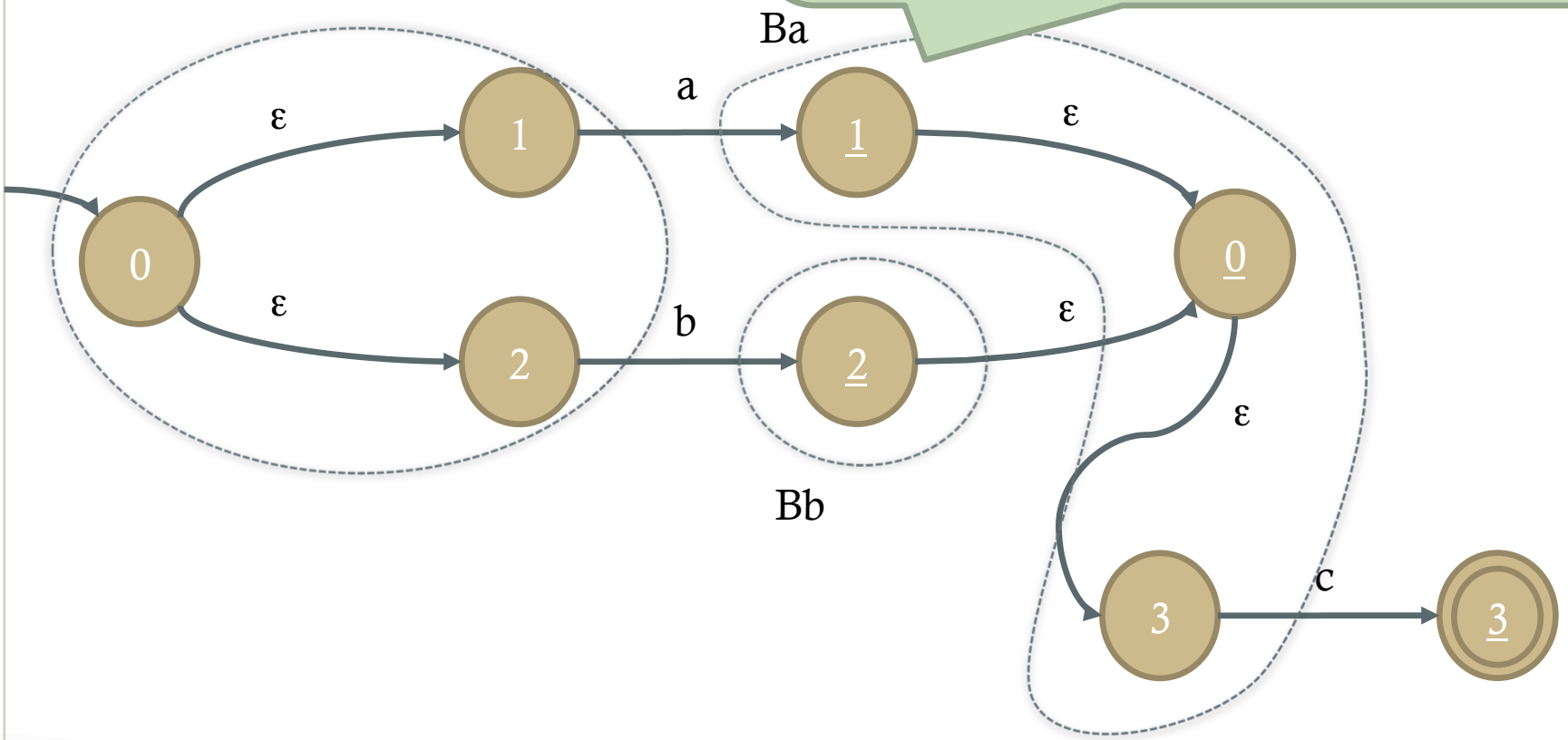
Ex

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

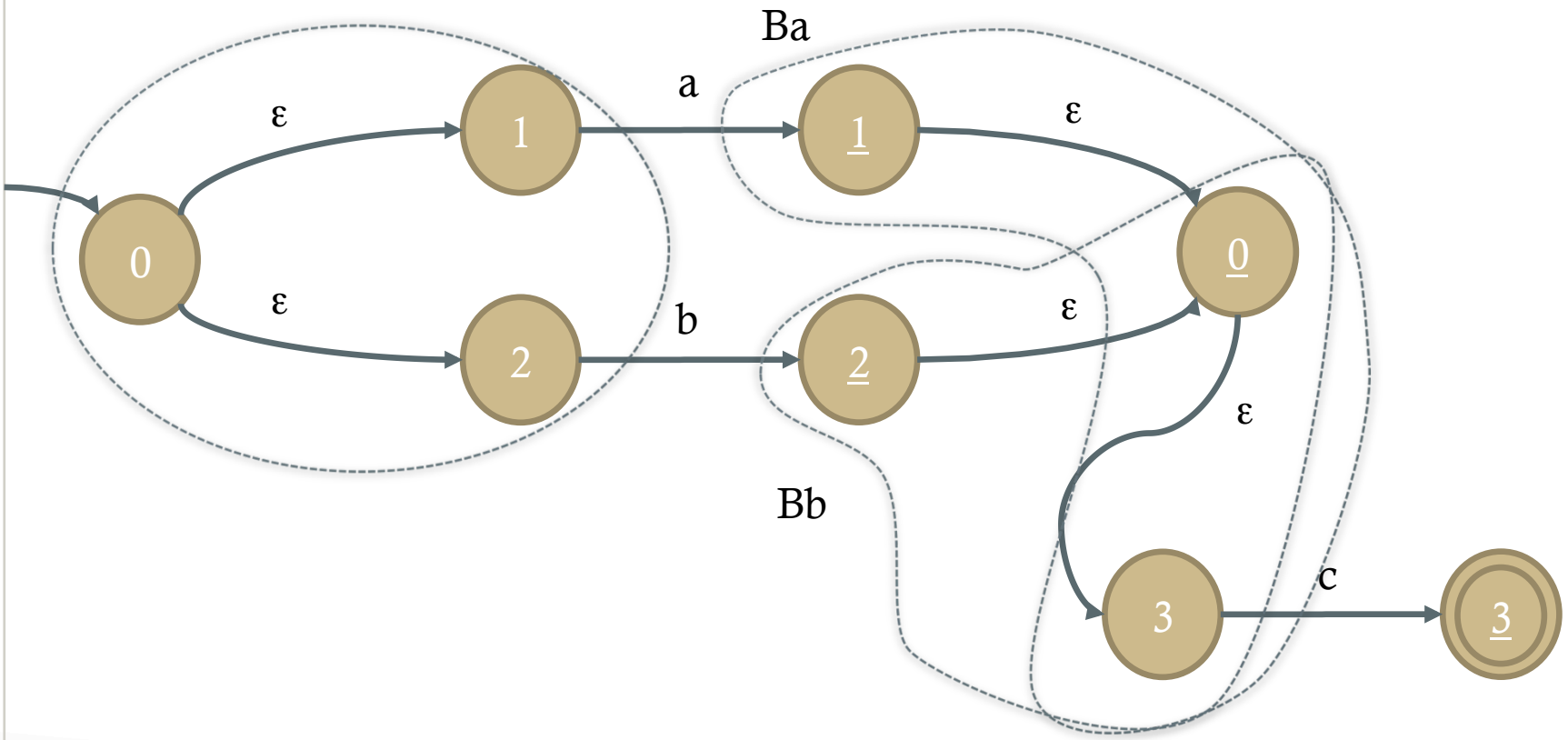


Ex

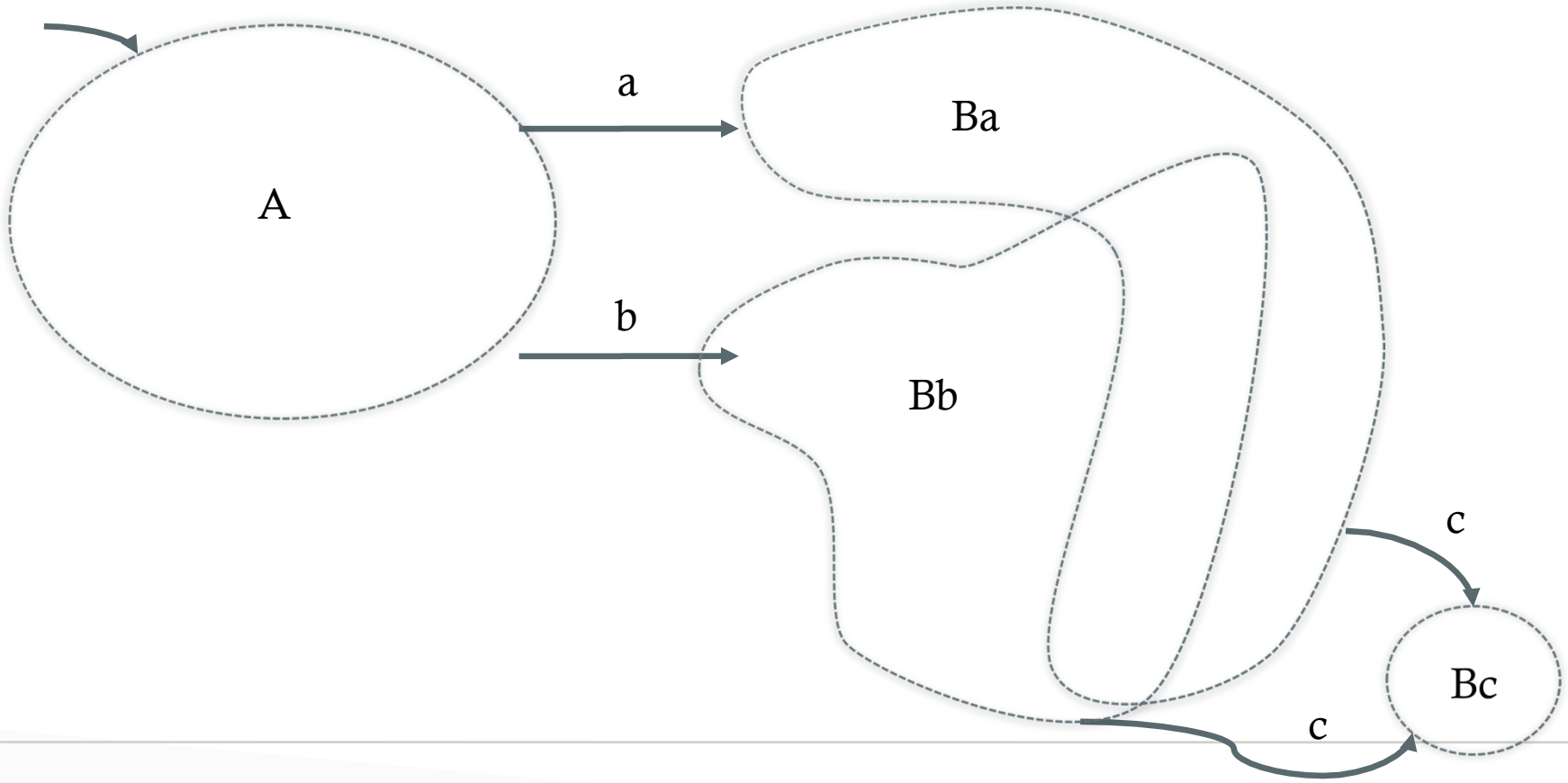
3. Restart process from 1 on each B (unless B has already been processed earlier).



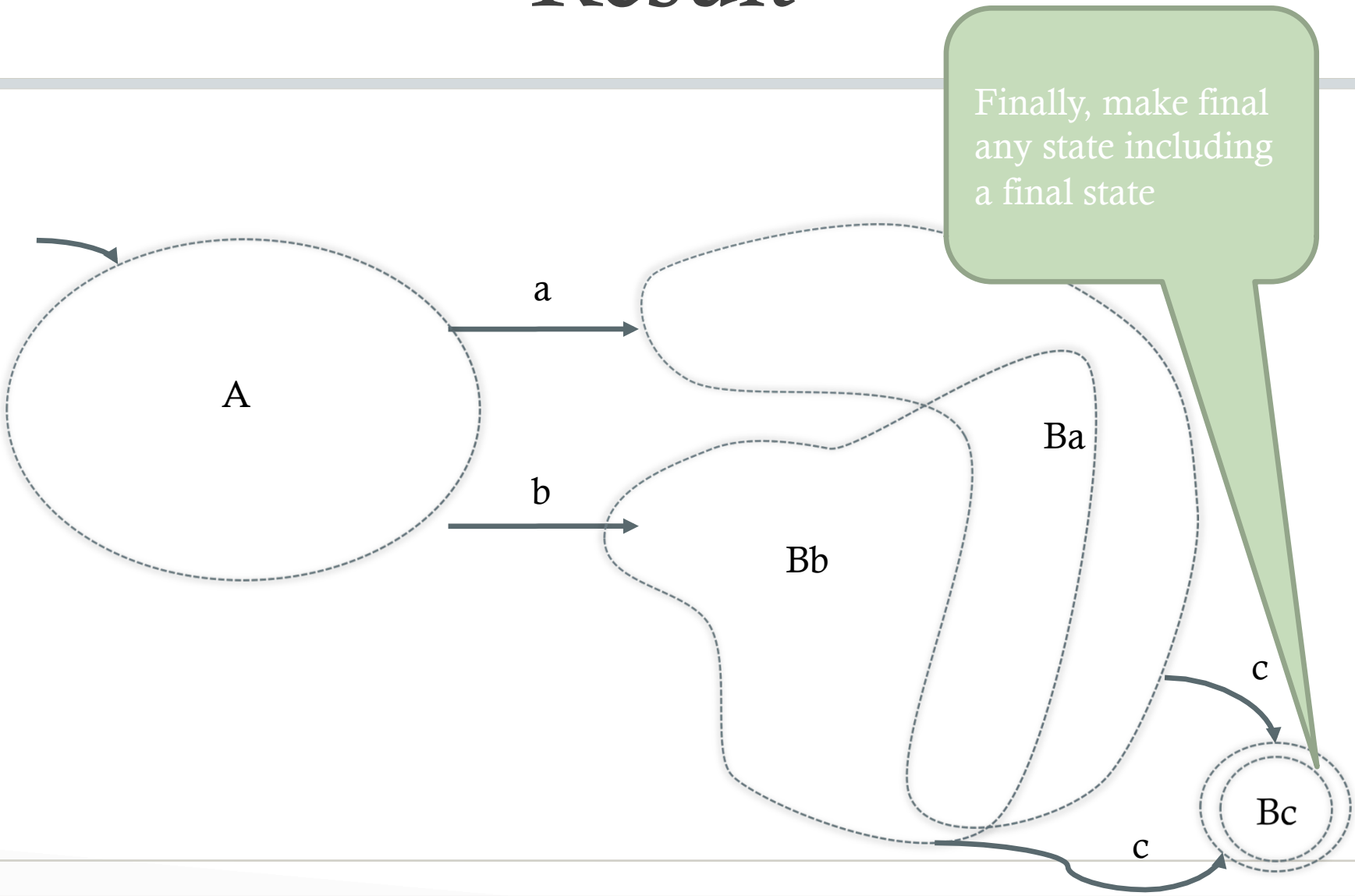
Example



Result



Result


Finally, make final any state including a final state
A
a
b
Ba
Bb
c
Bc
c

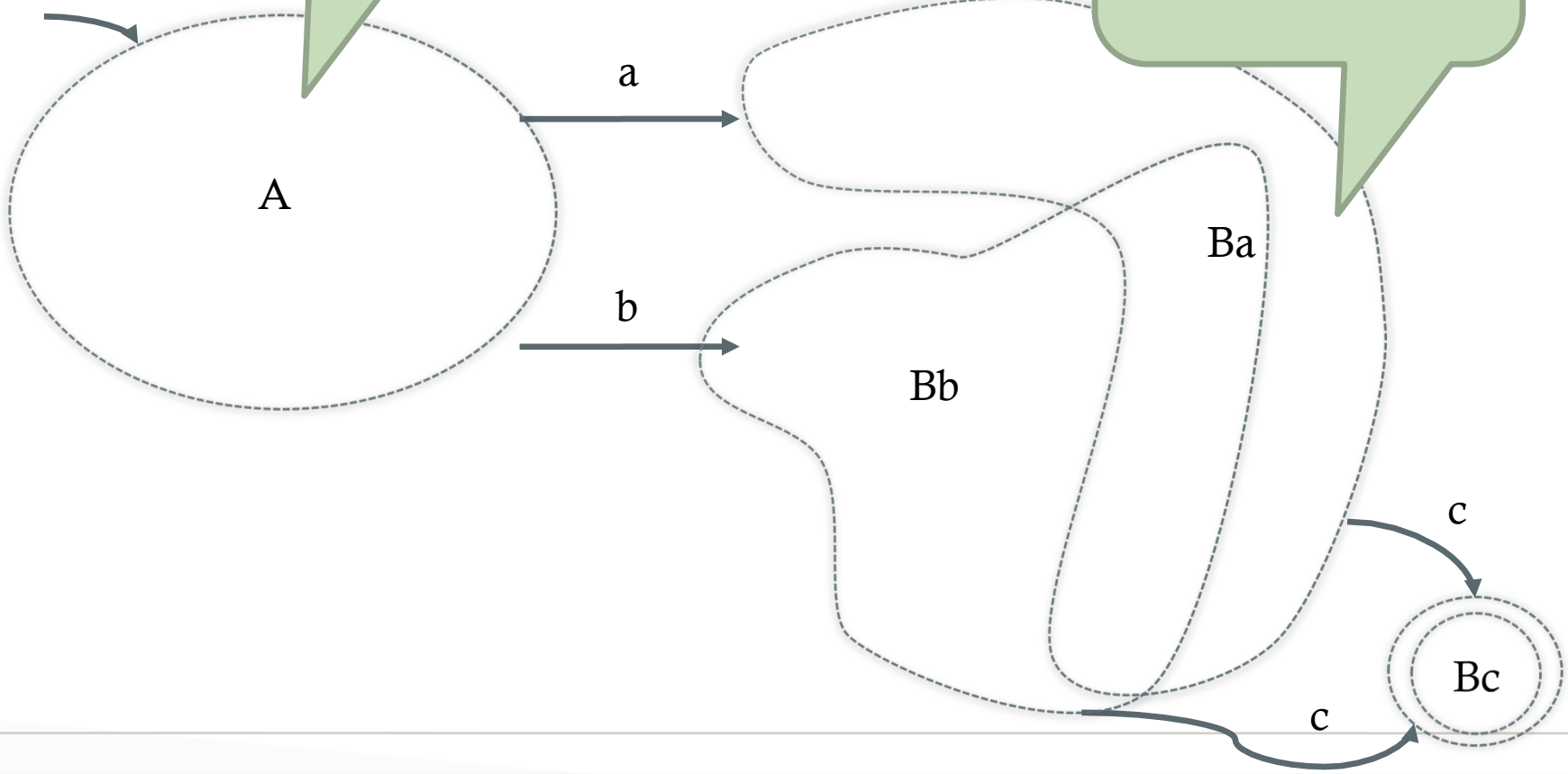
Determinisation

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

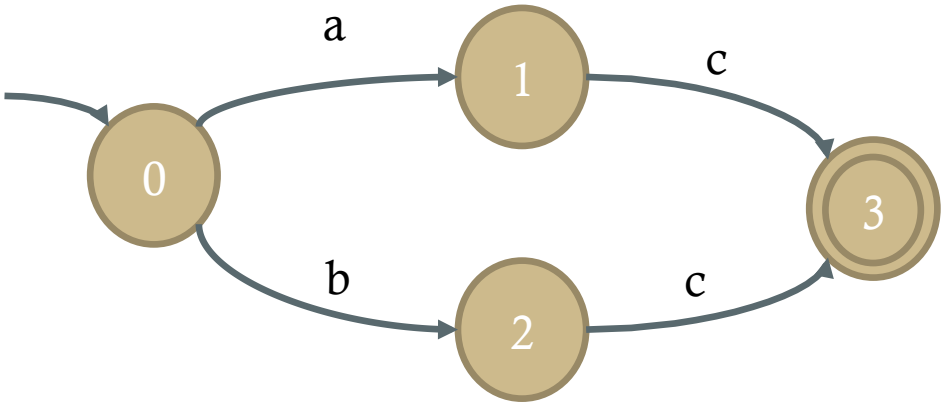
Problem

How should monitor react if observing "**c**" here?

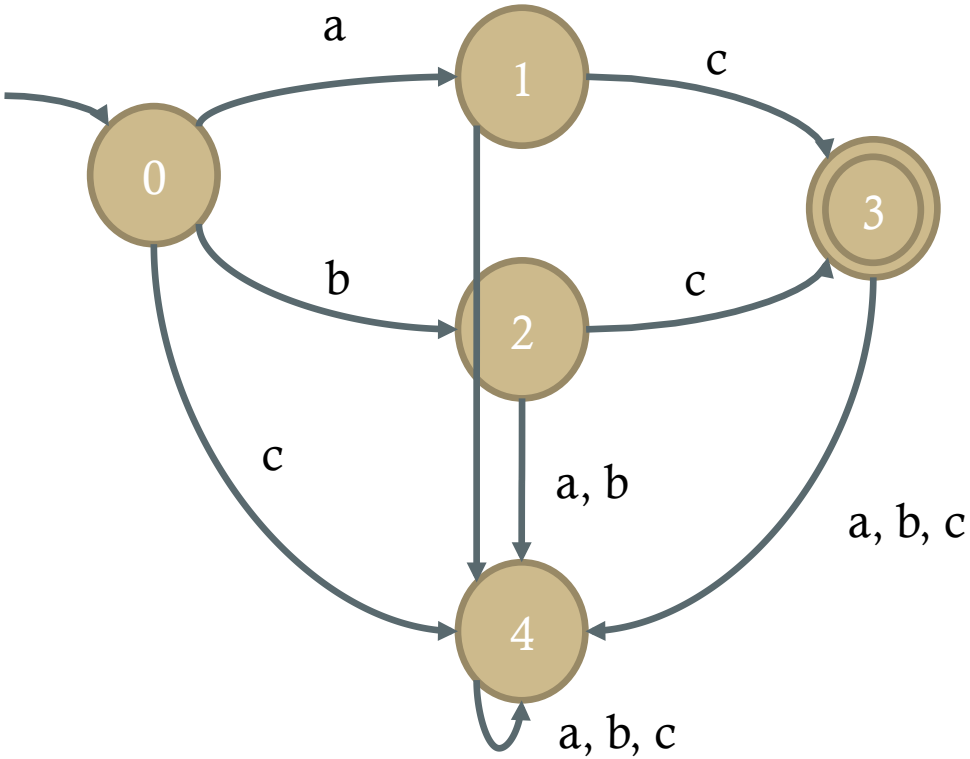
How should monitor react if observing "**a**" here?



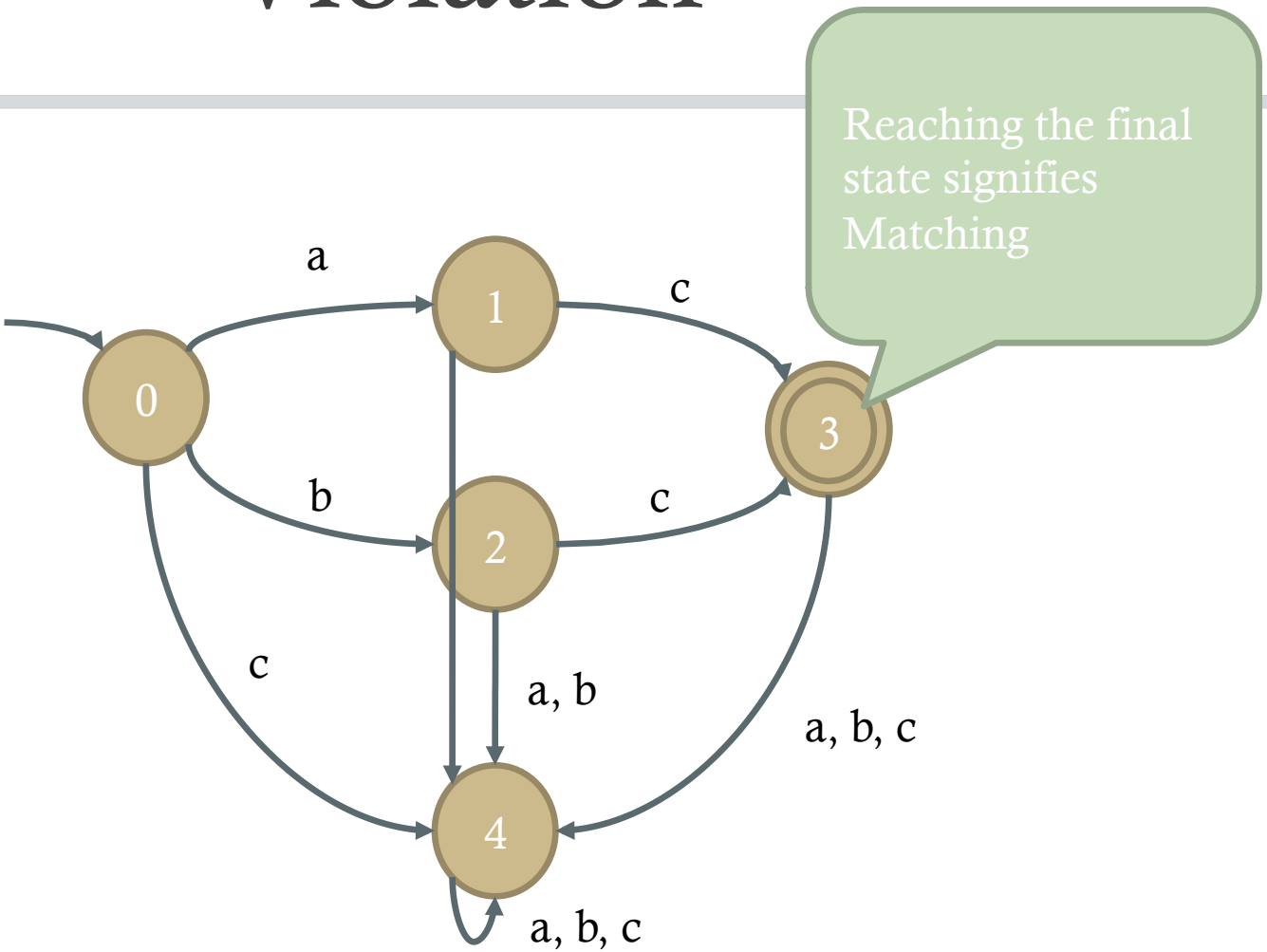
Totality



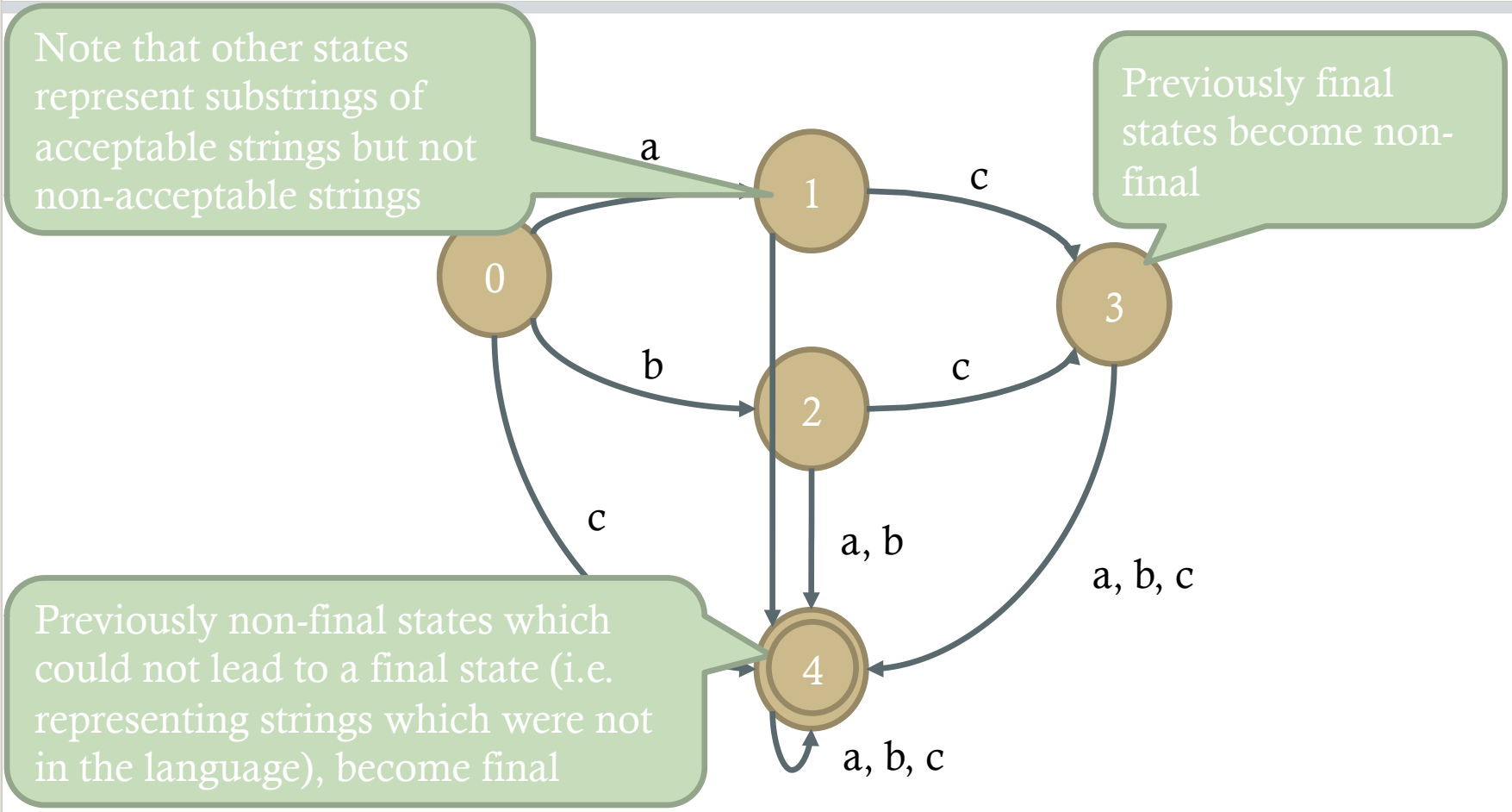
Totality



Violation


Reaching the final state signifies Matching
0
1
2
3
4
a
b
c
c
c
a, b
a, b, c
a, b, c

Complement if you to flag Non-Matching



Exercises

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