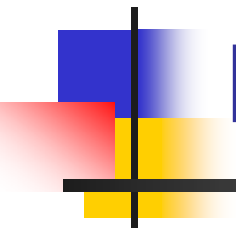
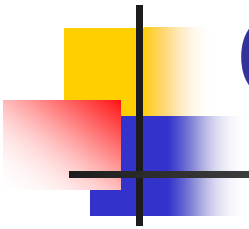


Approximate Search of **Regular Expressions** Using **Bit-Parallel** Algorithms

Kristo Tammeoja

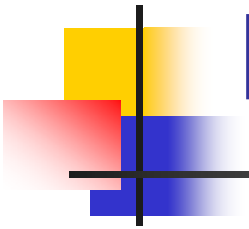
Jaak Vilo

Teooriapäevad Rõuges, 2007



Contents

- Regular expression (RE) syntax
- Glushkov's automaton
- Existing bit-parallel algorithms
  - Exact matching
  - Approximate matching
- New feature added
  - Error-free regions



Regular expression

- Syntax
  - (,)
  - |
  - Quantifier
    - *, +, ?, {m,n}, {m,}
  - Character classes (example [a-z])

Regular expression

- Syntax
  - (,)
  - |
  - Quantifier
    - *, +, ?, {m,n}, {m,}
  - Character classes (example [a-z])
- Matching as used in presentation
  - Regular expression A*
  - AAAAA match
  - BAAAC no match

Regular expression

1 error allowed

R(E|G)(EX) * **1:R(E|G)(EX)*** **1:R<E|G>(EX)***

1:R(E|G)<EX>*

Regular expression

1 error allowed

R(E|G)(EX) * **1:R(E|G)(EX)*** **1:R<E|G>(EX)***

R E

R G

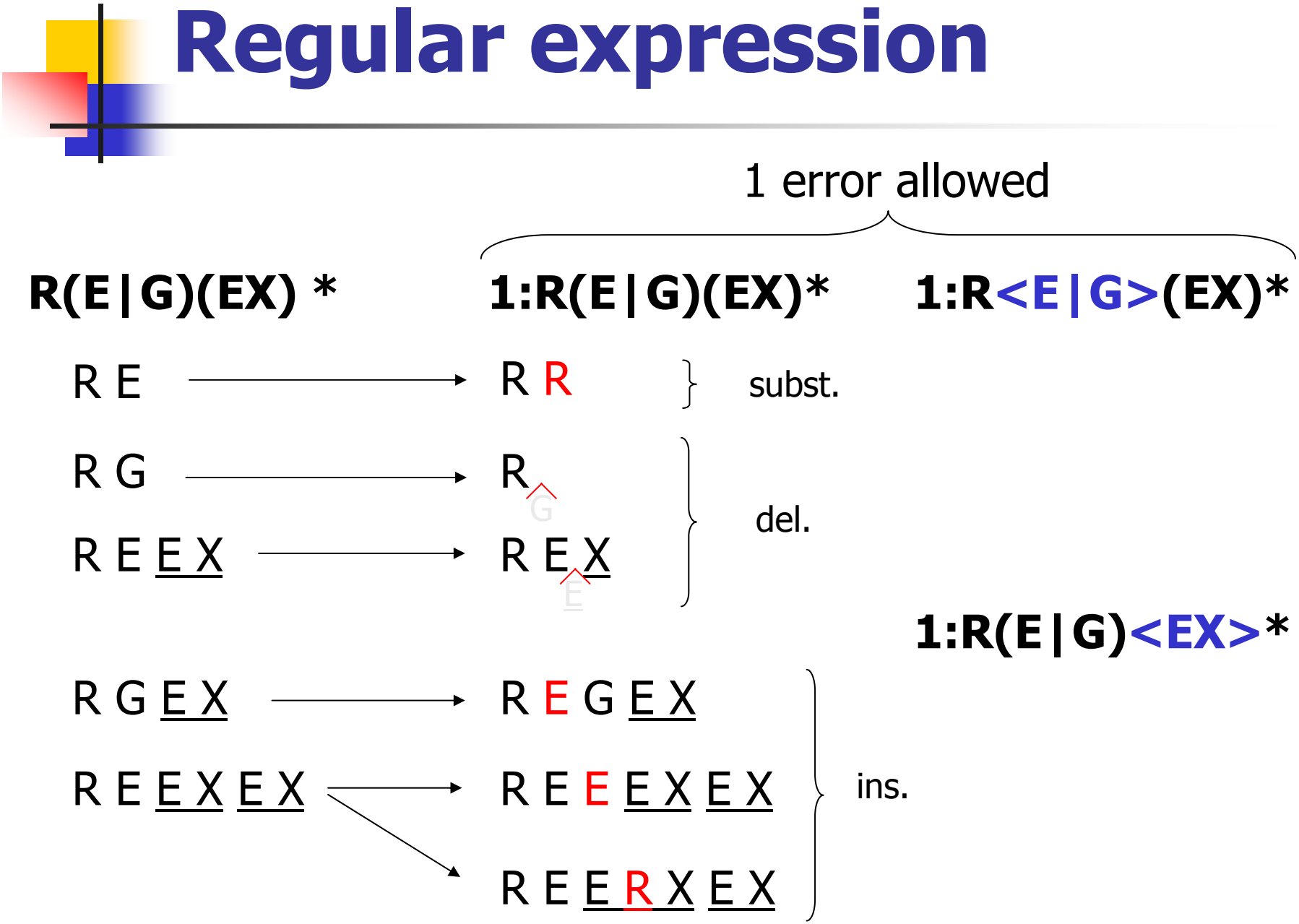
R E E X

1:R(E|G)<EX>*

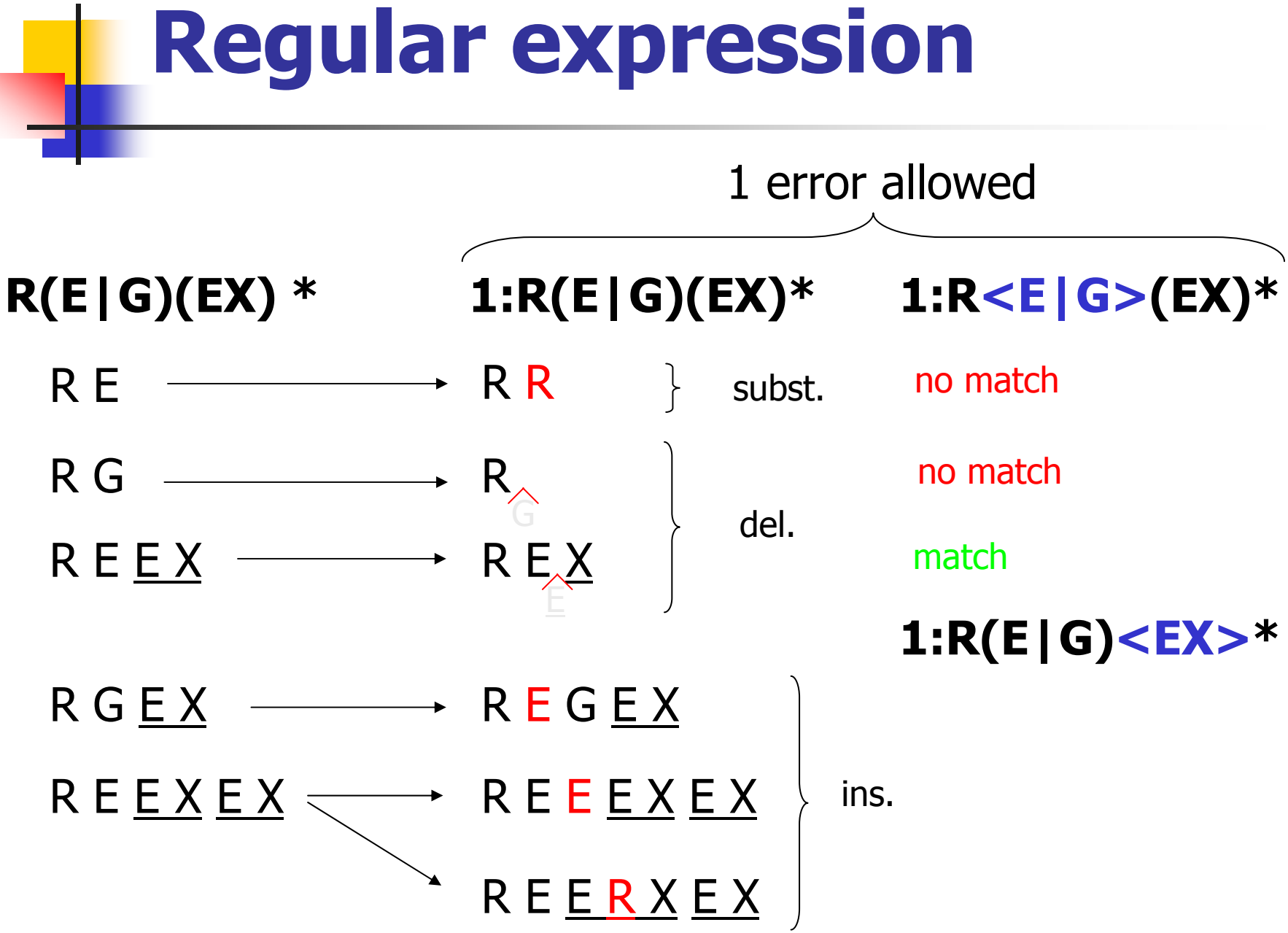
R G E X

R E E X E X

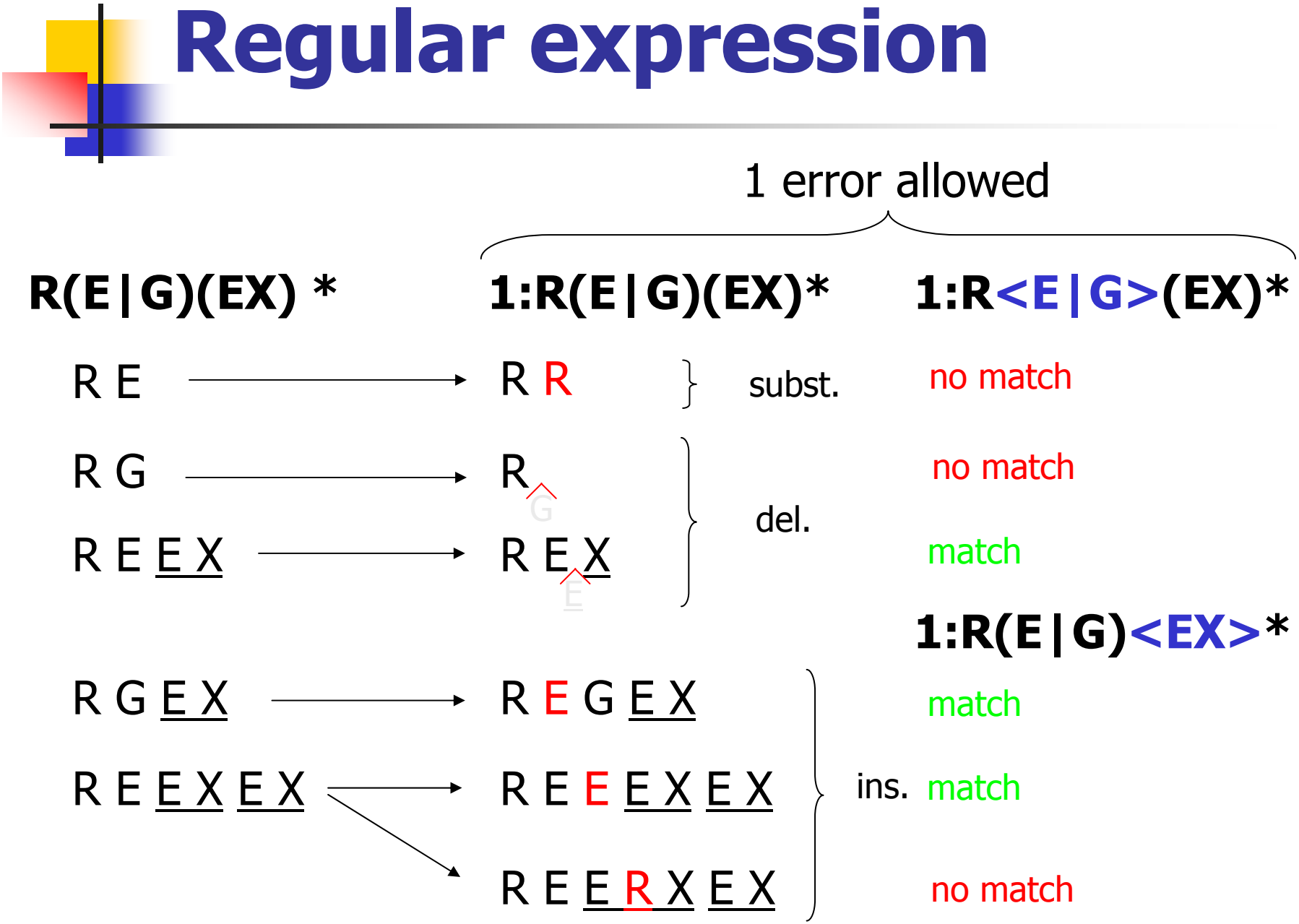
Regular expression



Regular expression



Regular expression

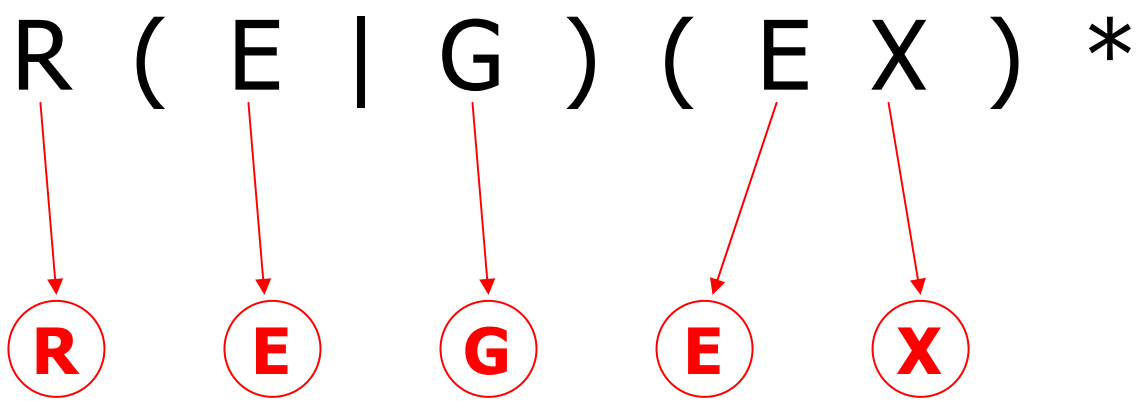


Glushkov's automaton

R (E | G) (E X) *

Glushkov's automaton

- Character in RE = **state** in automaton



Glushkov's automaton

- Character in RE = **state** in automaton
  + one state for the beginning of the RE

R (E | G) (E X) *

Glushkov's automaton

- Character in RE = **state** in automaton
  + one state for the beginning of the RE
- **Transitions** show which characters/positions can precede each other

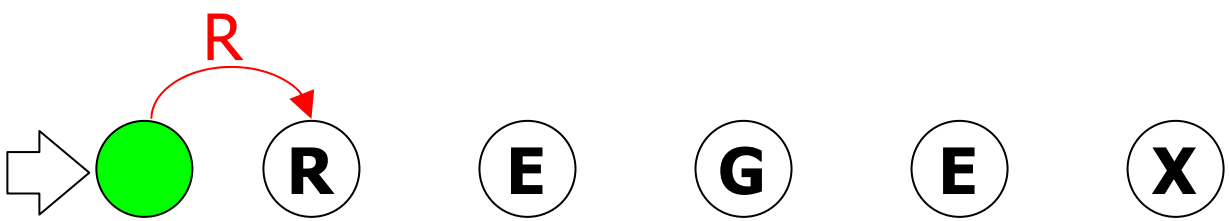
R (E | G) (E X) *

R...

Glushkov's automaton

- Character in RE = **state** in automaton
  + one state for the beginning of the RE
- **Transitions** show which characters/positions can precede each other

R (E | G) (E X) *

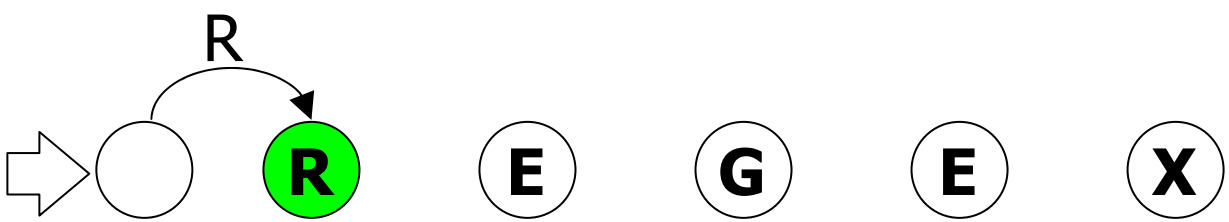


R...

Glushkov's automaton

- Character in RE = **state** in automaton
  + one state for the beginning of the RE
- **Transitions** show which characters/positions can precede each other

R (E | G) (E X) *

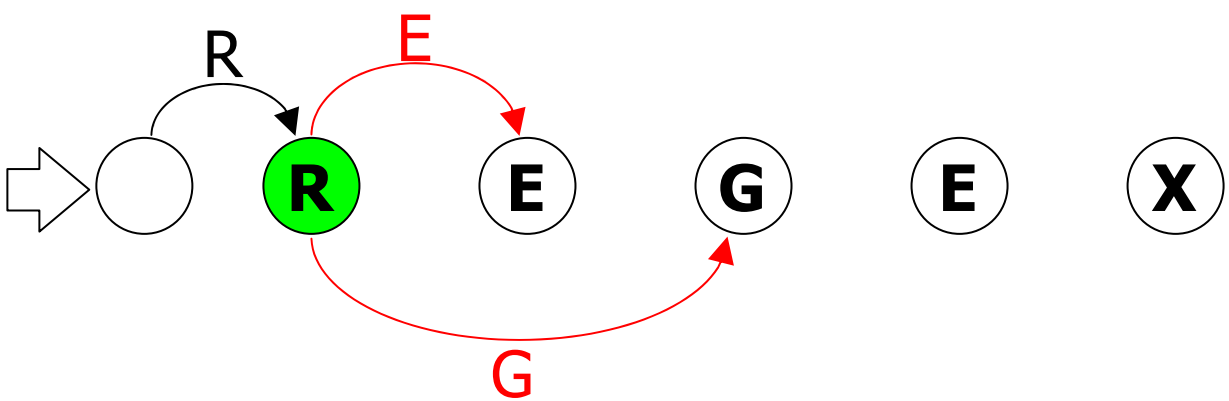


R...

Glushkov's automaton

- Character in RE = **state** in automaton
  + one state for the beginning of the RE
- **Transitions** show which characters/positions can precede each other

R (E | G) (E X) *

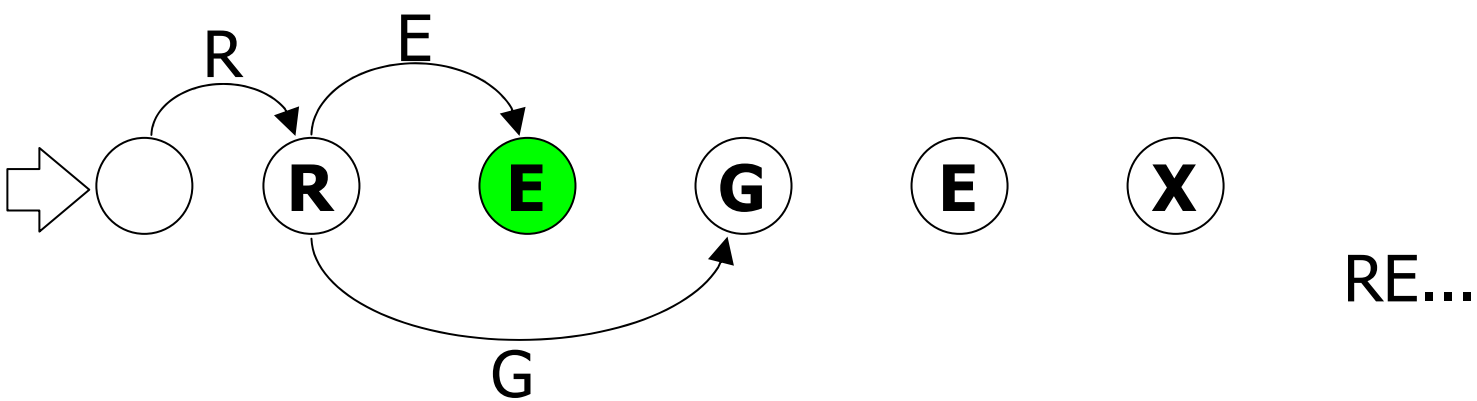


RE...
RG...

Glushkov's automaton

- Character in RE = **state** in automaton
  + one state for the beginning of the RE
- **Transitions** show which characters/positions can precede each other

R (E | G) (E X) *

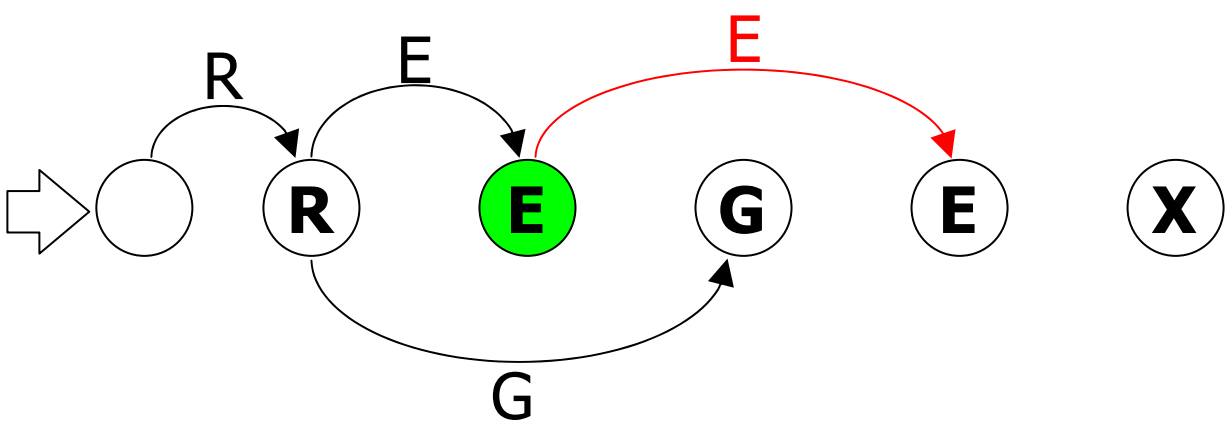


RE...

Glushkov's automaton

- Character in RE = **state** in automaton
  + one state for the beginning of the RE
- **Transitions** show which characters/positions can precede each other

R (E | G) (E X) *

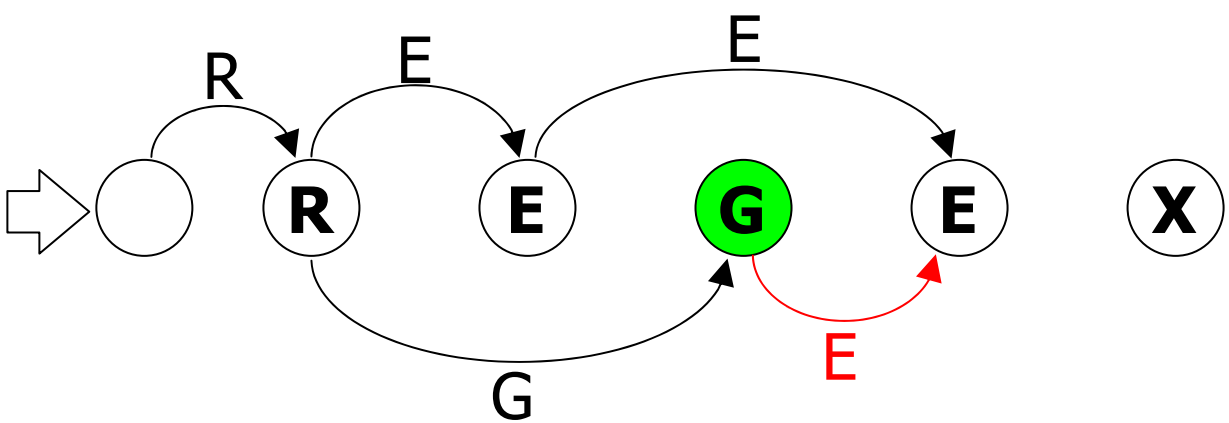


R**E**E...

Glushkov's automaton

- Character in RE = **state** in automaton
  + one state for the beginning of the RE
- **Transitions** show which characters/positions can precede each other

R (E | G) (E X) *

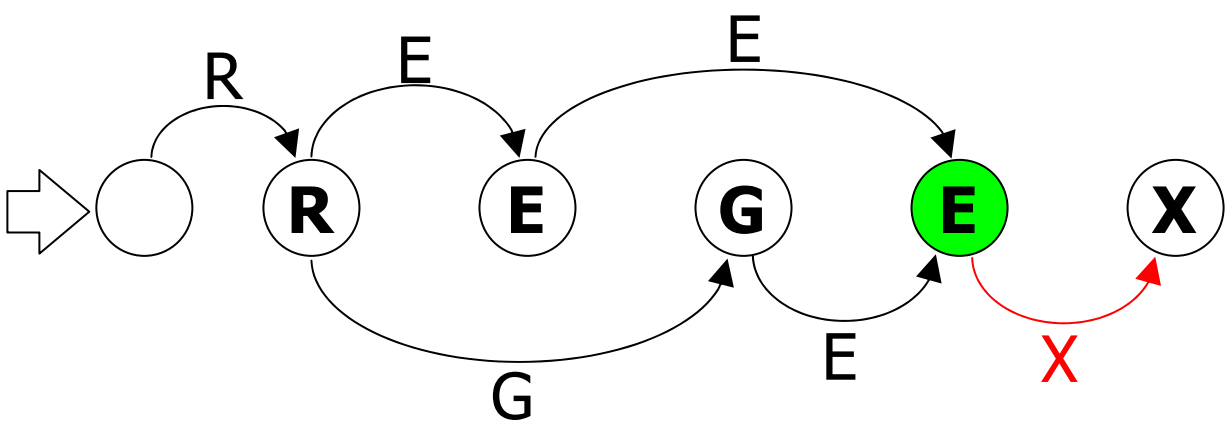


RGE...

Glushkov's automaton

- Character in RE = **state** in automaton
  + one state for the beginning of the RE
- **Transitions** show which characters/positions can precede each other

R (E | G) (E X) *

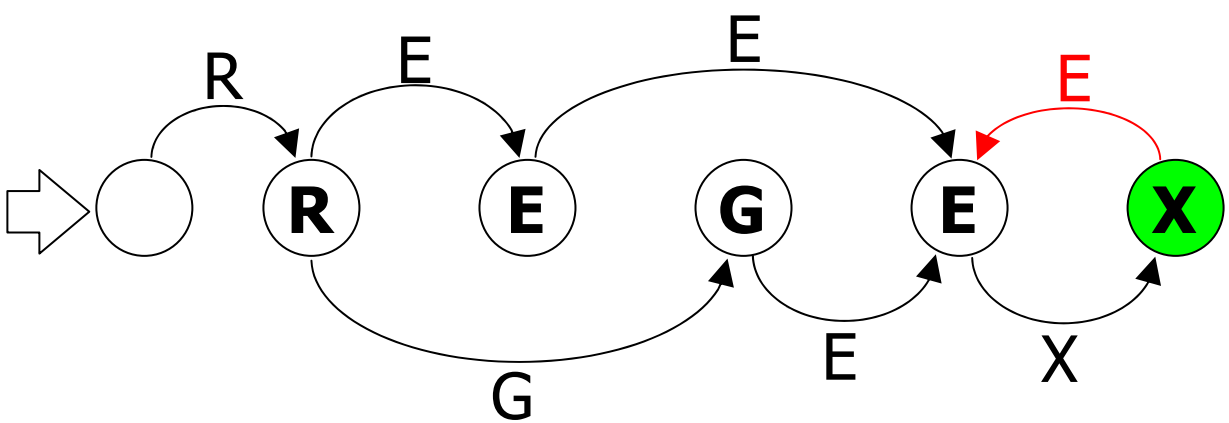


RG**E**X...

Glushkov's automaton

- Character in RE = **state** in automaton
  + one state for the beginning of the RE
- **Transitions** show which characters/positions can precede each other

R (E | G) (E X) *

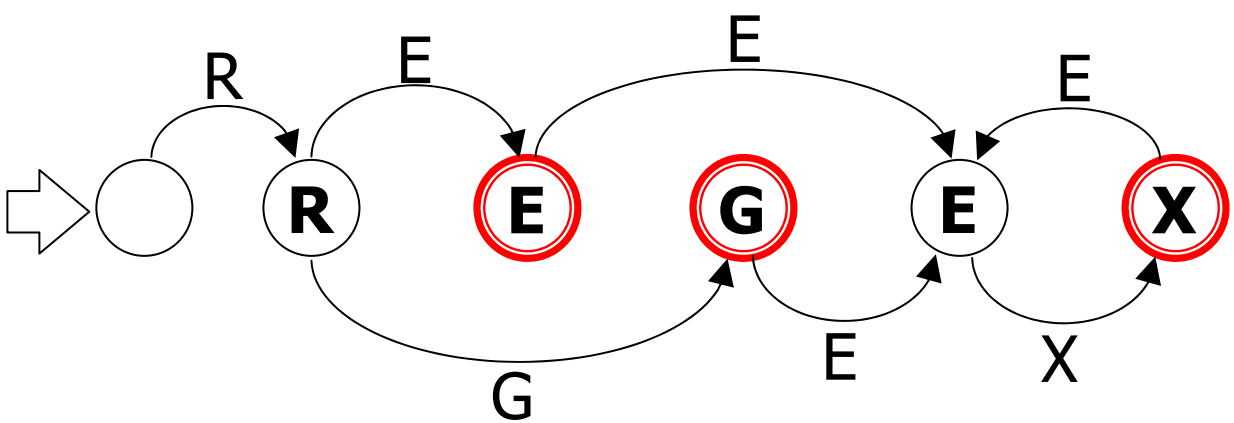


RGE**X**E...

Glushkov's automaton

- Character in RE = **state** in automaton
  + one state for the beginning of the RE
- **Transitions** show which characters/positions can precede each other

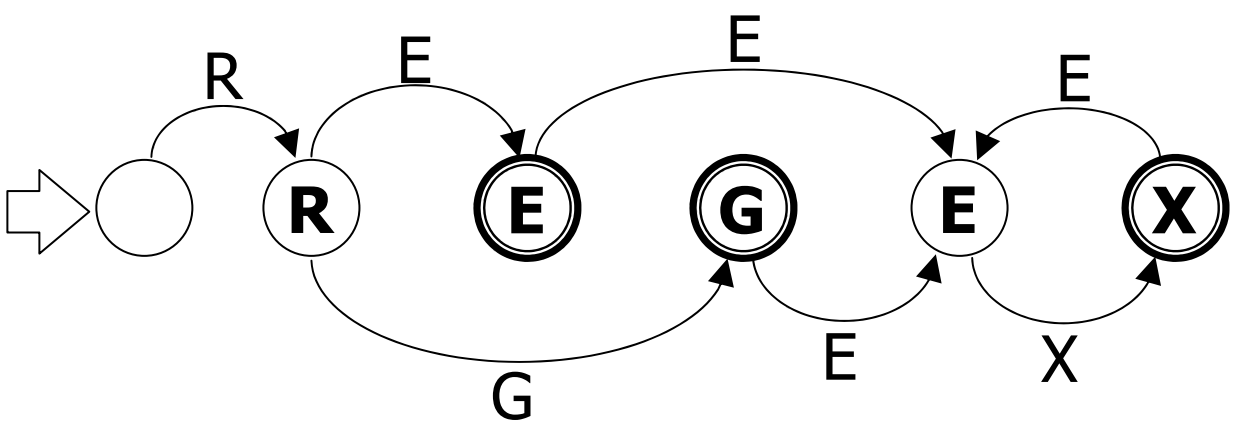
R (E | G) (E X) *



Glushkov's automaton

- All labels entering a node are labeled by the same character

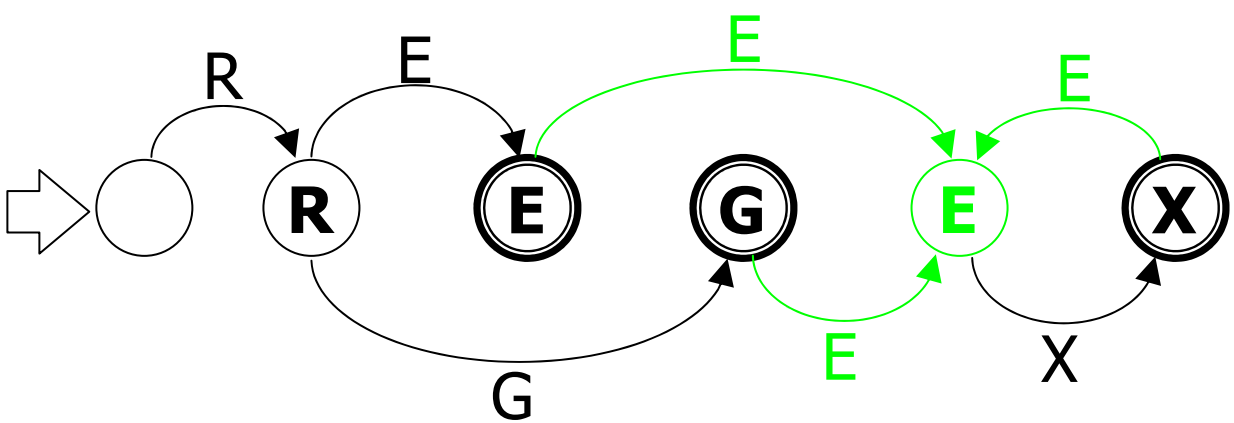
R (E | G) (E X) *



Glushkov's automaton

- All labels entering a node are labeled by the same character

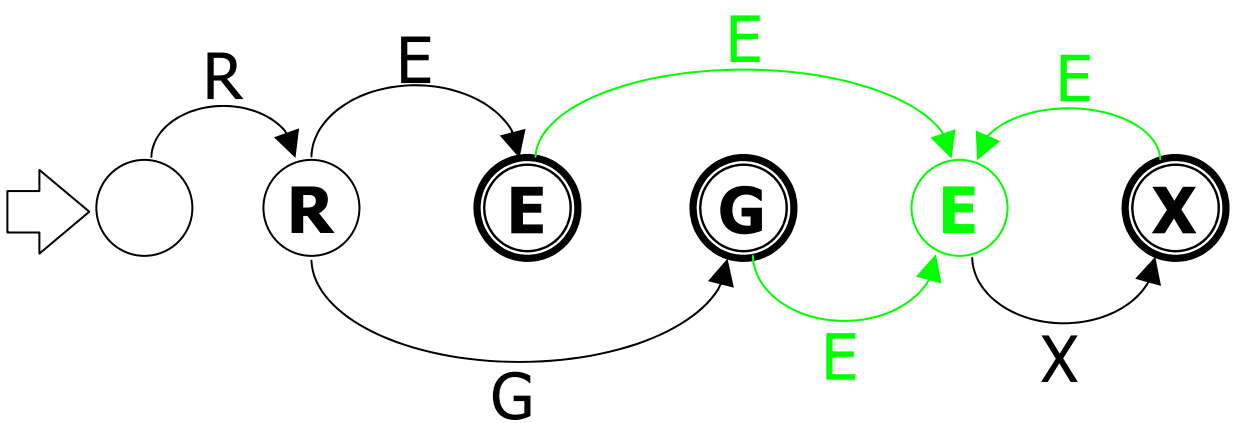
R (E | G) (E X) *



Glushkov's automaton

- All labels entering a node are labeled by the same character

  for example **after reading character 'E' only states with label 'E' can be active**



Exact search

- Simulation of NFA = changing active states based on the character read from the text
- We use bit-vectors (one bit for each state) to hold active states

  $\delta(D, a)$
  - D – bit-vector of active states
  - a – character read
  - Returns new bit-vector

- $2^{|D|} \cdot |\Sigma|$ different sets of parameters
  - |D| – number of states in automaton
  - |Σ| - alphabet's size

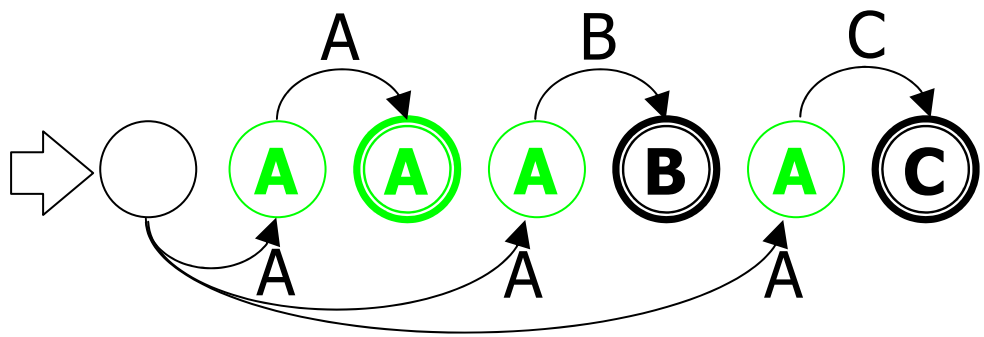
Exact search

- "After reading character 'E' only states with label 'E' can be active" so ...
- $\delta(D, a) = T[D] \,\&\, B[a]$
  - T[**D**] – states that can be reached **<u>from states in D</u>** by any character
  - B[**a**] – states that can be reached **<u>by character a</u>**

Exact search

- $\delta(D, a) = T[D] \,\&\, B[a]$

AA|AB|AC



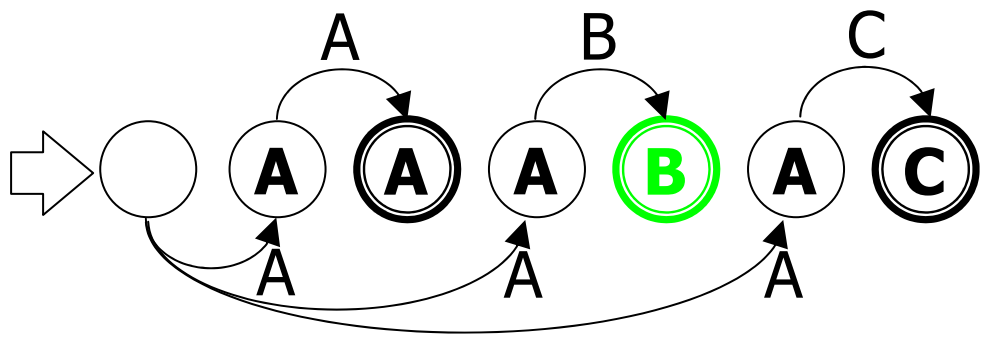
a	B[a]
'A'	0111010
'B'	
'C'	

D	T[D]
1000000	
0100000	
...	
0101010	
...	

Exact search

- $\delta(D, a) = T[D] \,\&\, B[a]$

AA|AB|AC



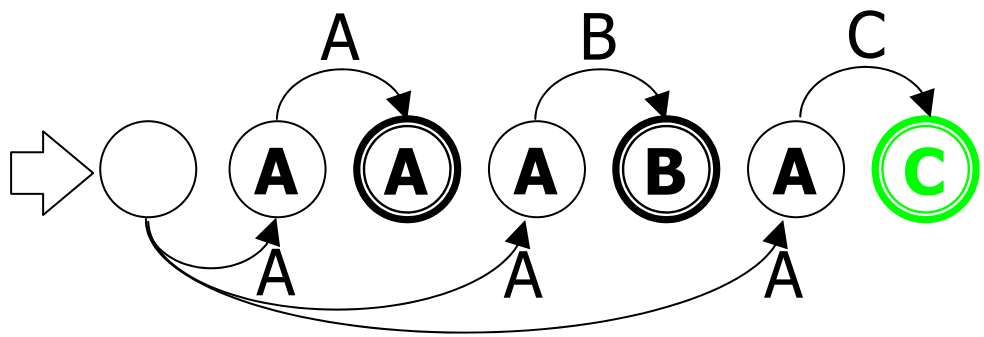
a	B[a]
'A'	0111010
'B'	0000100
'C'	

D	T[D]
1000000	
0100000	
...	
0101010	
...	

Exact search

- $\delta(D, a) = T[D] \,\&\, B[a]$

AA|AB|AC



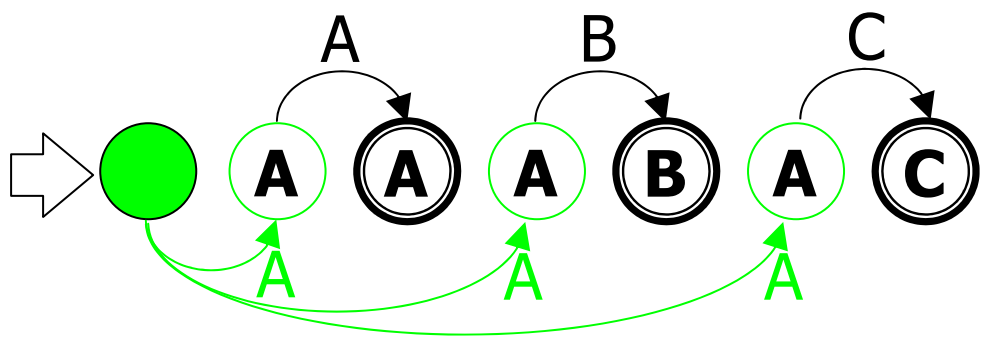
a	B[a]
'A'	0111010
'B'	0000100
'C'	0000001

D	T[D]
1000000	
0100000	
...	
0101010	
...	

Exact search

- $\delta(D, a) = T[D] \,\&\, B[a]$

AA|AB|AC



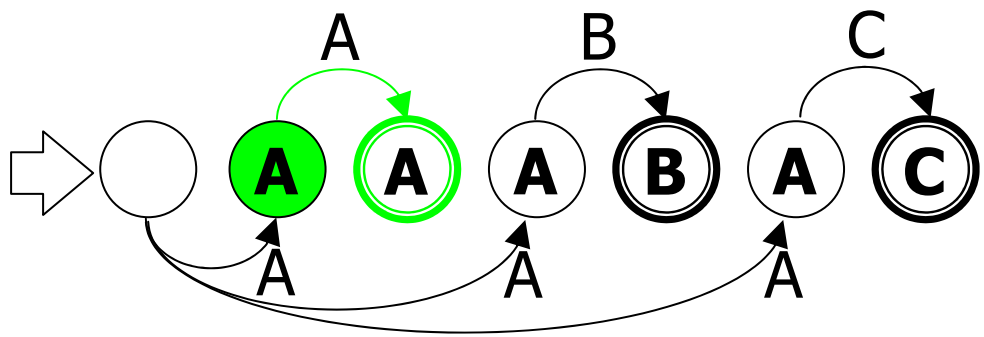
a	B[a]
'A'	0111010
'B'	0000100
'C'	0000001

D	T[D]
1000000	0101010
0100000	
...	
0101010	
...	

Exact search

- $\delta(D, a) = T[D] \,\&\, B[a]$

AA|AB|AC



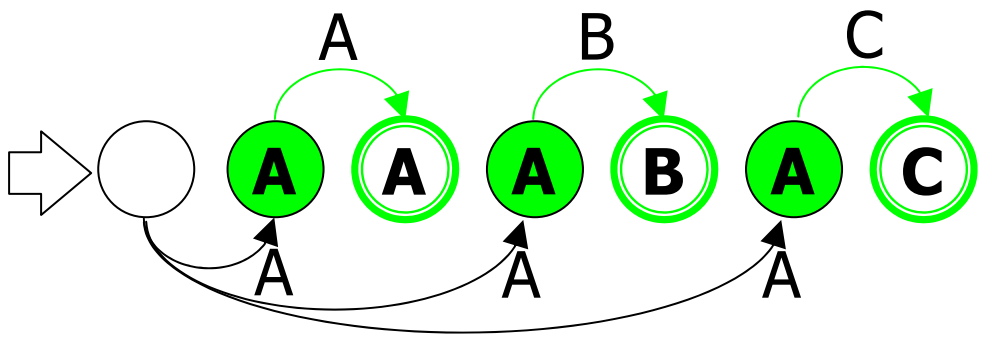
a	B[a]
'A'	0111010
'B'	0000100
'C'	0000001

D	T[D]
1000000	0101010
0100000	0010000
...	
0101010	
...	

Exact search

- $\delta(D, a) = T[D] \,\&\, B[a]$

AA|AB|AC



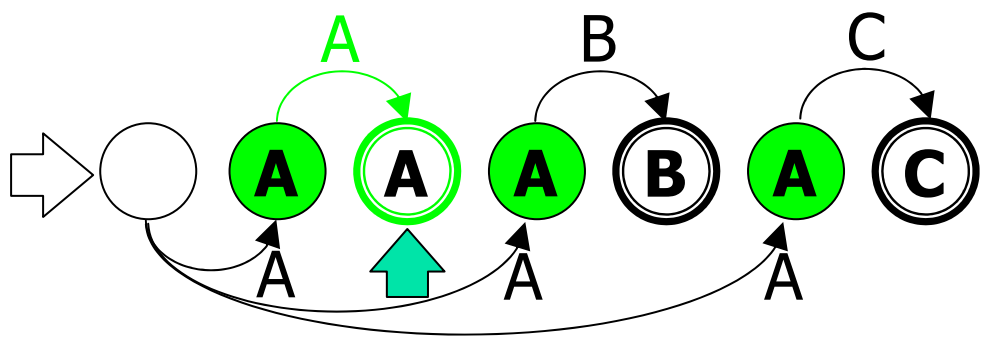
a	B[a]
'A'	0111010
'B'	0000100
'C'	0000001

D	T[D]
1000000	0101010
0100000	0010000
...	
0101010	0010101
...	

Exact search

- $\delta(D, a) = T[D] \,\&\, B[a]$

AA|AB|AC



a	B[a]
'A'	0111010
'B'	0000100
'C'	0000001

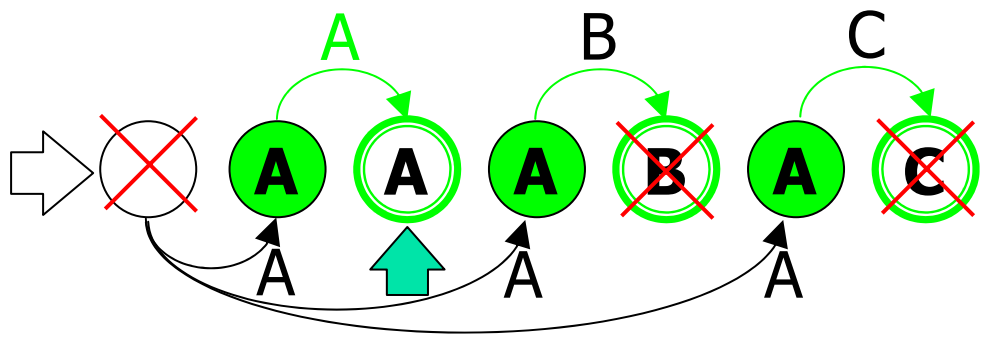
D	T[D]
1000000	0101010
0100000	0010000
...	
0101010	0010101
...	

$\delta(0101010, \text{'A'})$

Exact search

- $\delta(D, a) = T[D] \,\&\, B[a]$

AA|AB|AC

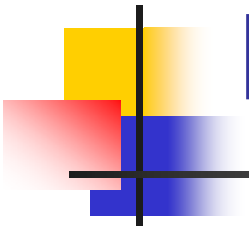


a	B[a]
'A'	0111010
'B'	0000100
'C'	0000001

D	T[D]
1000000	0101010
0100000	0010000
...	
0101010	0010101
...	

δ(0101010, 'A')

0010101	T[D]
& 0111010	B[a]
0010000	



Exact search

D ← 100..00 // initial state active

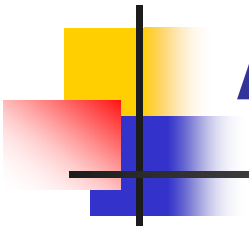
F ← bit-vector of final states

For pos ∈ 1 ... n **Do** // scanning text

  D ← T[D] & $B[t_{pos}]$

  If D & F ≠ 000..00 **Then** match

End of For



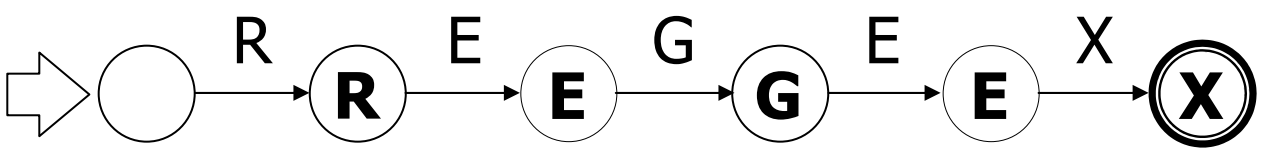
Approximate search

Errors

- Insertion
- Deletion
- Substitution

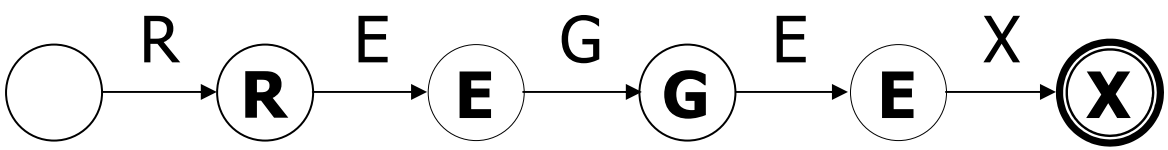
Approximate search

- When searching with k errors we make k+1 replicas of the automaton, one for each error-level
- Plus we need transitions for errors



No errors

? ? ? ? ?



Up to 1 error

Approximate search

- R_0, R_1 – current bit-vectors
- R_0', R_1' – bit-vectors after processing character a

$R_0' = T[R_0]$ & $B[c]$

$R_1' = ?$

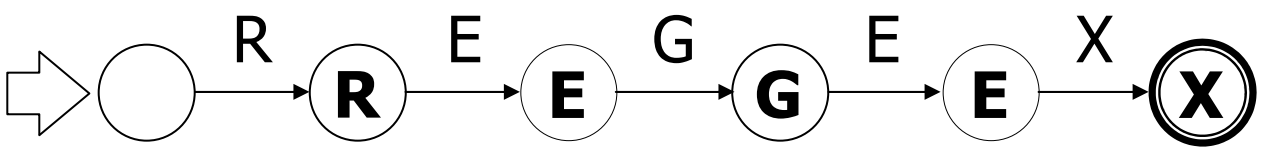
Approximate search

$R_1' = T[R_1] \;\&\; B[c] \mid \ldots$

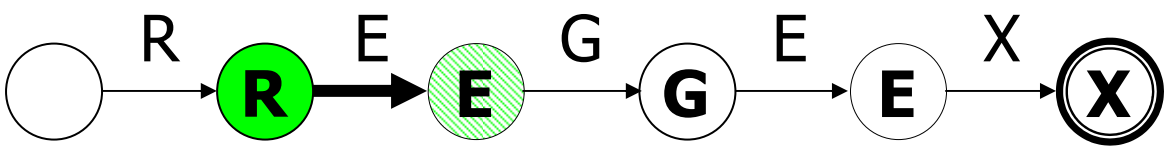
no errors

- Same as in exact search

EGEX



No errors



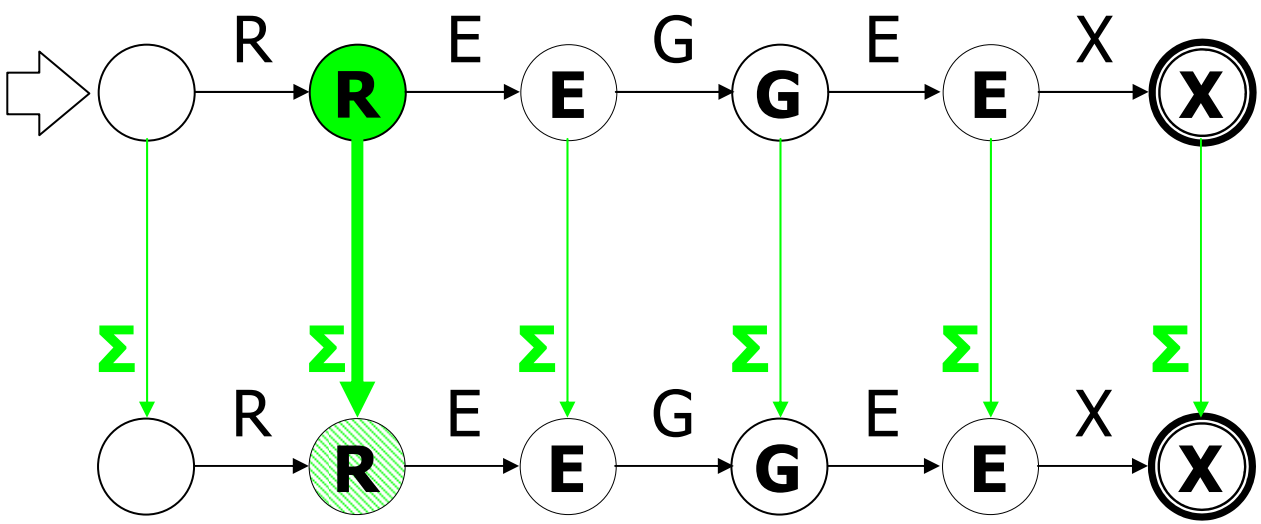
Up to 1 error

Approximate search

$$R_1' = \underbrace{T[R_1] \,\&\, B[c]}_{\text{no errors}} \mid \underbrace{R_0}_{\text{del}} \mid \ldots$$

- Active states remain the same

RAEGEX



No errors

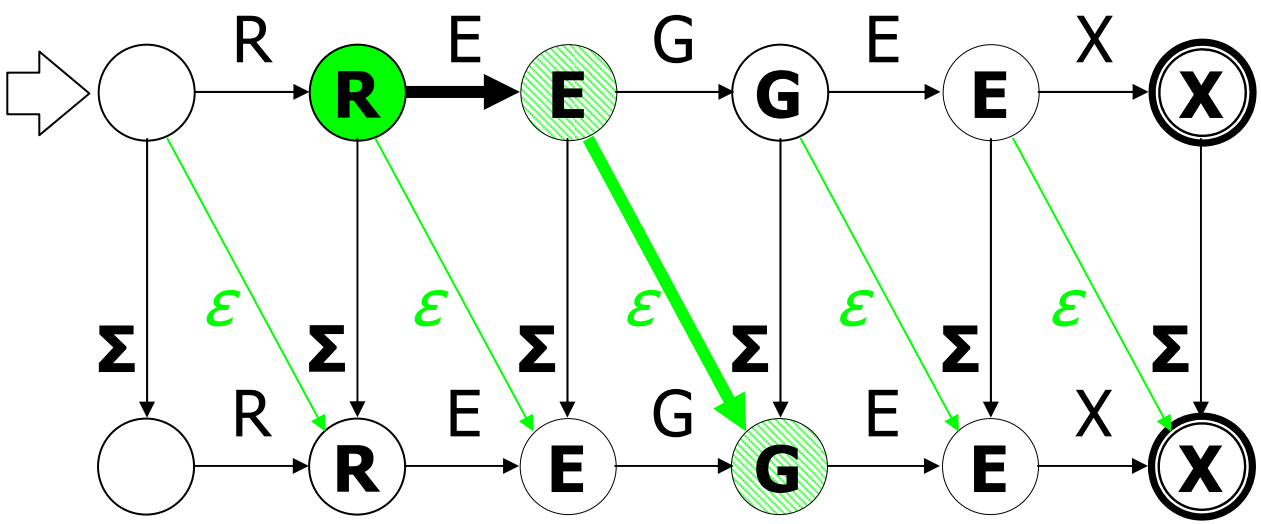
Up to 1 error

Approximate search

$$R_1' = \underbrace{T[R_1] \,\&\, B[c]}_{\text{no errors}} \mid \underbrace{R_0}_{\text{del}} \mid \underbrace{T[R_0']}_{\text{ins}} \mid \ldots$$

- Insert new character **after** the current one
- Just one step in automaton

REEX

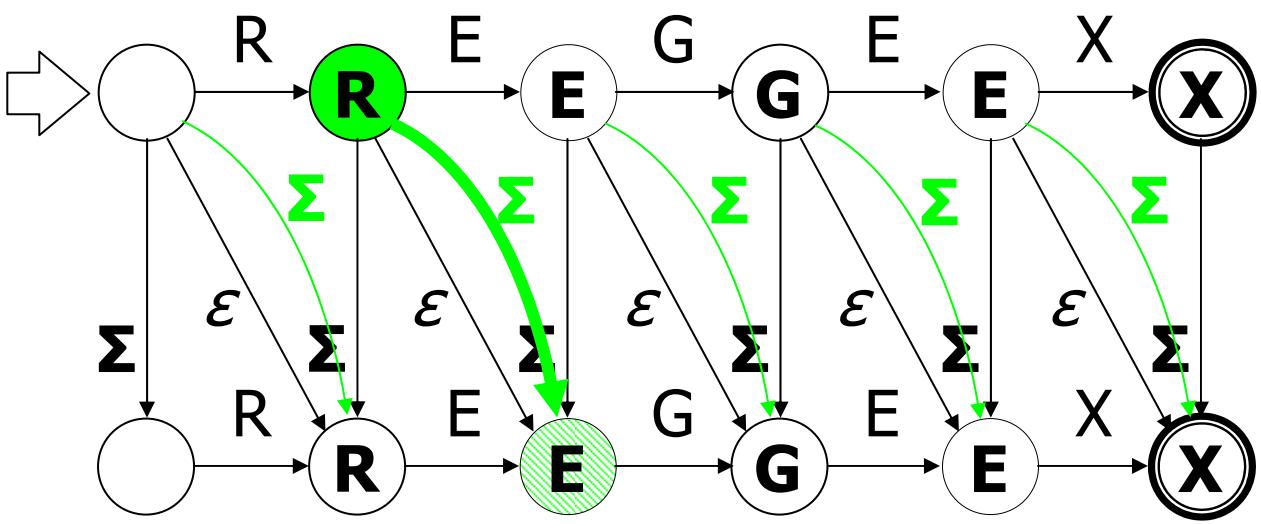


Approximate search

$$R_1' = \underbrace{T[R_1] \,\&\, B[c]}_{\text{no errors}} \mid \underbrace{R_0}_{\text{del}} \mid \underbrace{T[R_0']}_{\text{ins}} \mid \underbrace{T[R_0]}_{\text{subst}}$$

- Similar to exact matching except...
- ... we don't care about the character read

RAGEX



Error-free regions

- Approximate search

$R_1' = T[R_1] \;\&\; B[c] \mid R_0 \mid T[R_0'] \mid T[R_0]$

- With no-error regions

$R_1' = \underbrace{T[R_1] \;\&\; B[c]}_{\text{no errors}} \mid \underbrace{R_0 \;\&\; I}_{\text{del}} \mid \underbrace{T_e[R_0']}_{\text{ins}} \mid \underbrace{T_e[R_0]}_{\text{subst}}$

Error-free regions

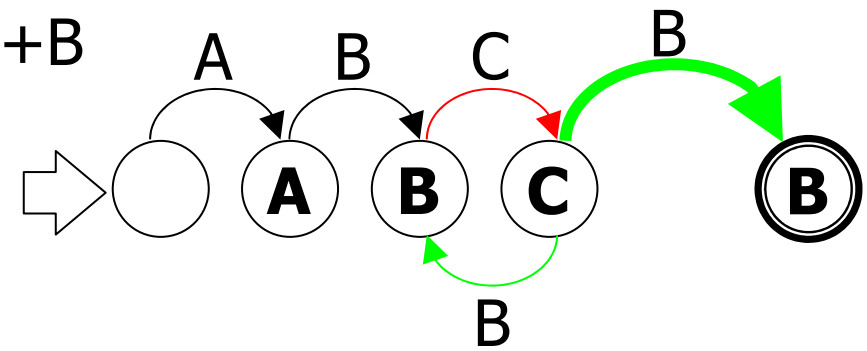
- Deletion from text

	A(B**C**)+B	A<(B**C**)+>B	A<B**C**>+B
A**X**BCBCB	match	match	match
AB**X**CBCB	match	no match	no mat
ABCBC**X**B	match	match	match
ABC**X**BCB	match	no match	match

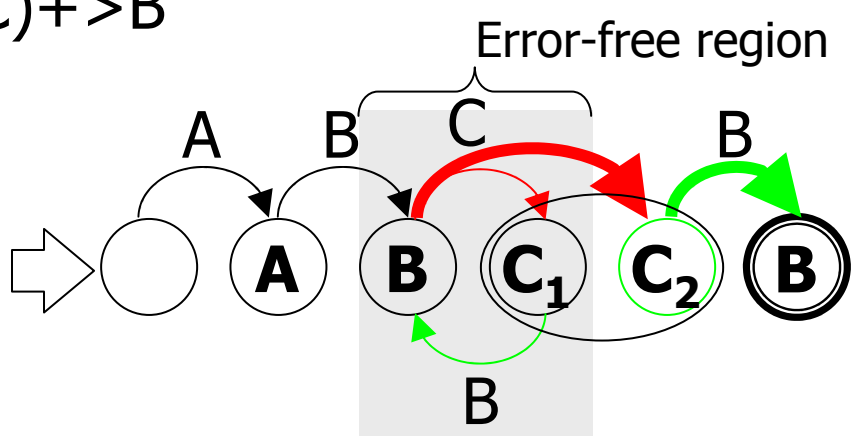
- Make two copies of 'C'
  - Those characters that error-free regions can end with must be duplicated

Error-free regions

A(BC)+B



A<(BC)+>B



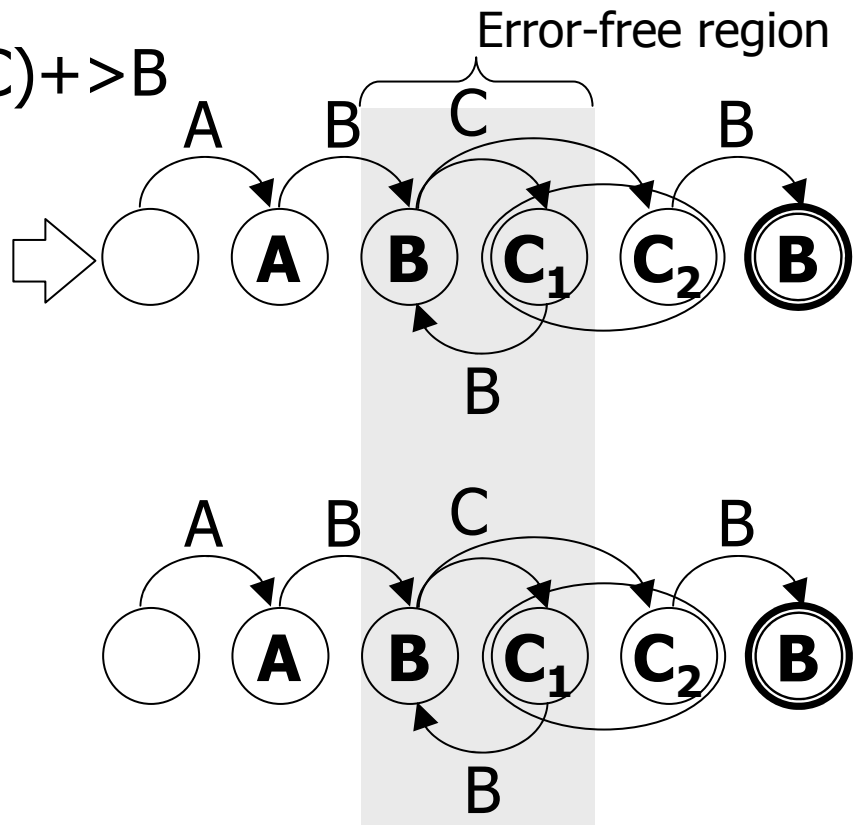
Error-free regions

$R_1' = T[R_1]\ \&\ B[c]\ |\ R_0 \quad |\ T[R_0']\ |\ T[R_0]$

$R_1' = T[R_1]\ \&\ B[c]\ |\ \ldots$

no errors

A<(BC)+>B



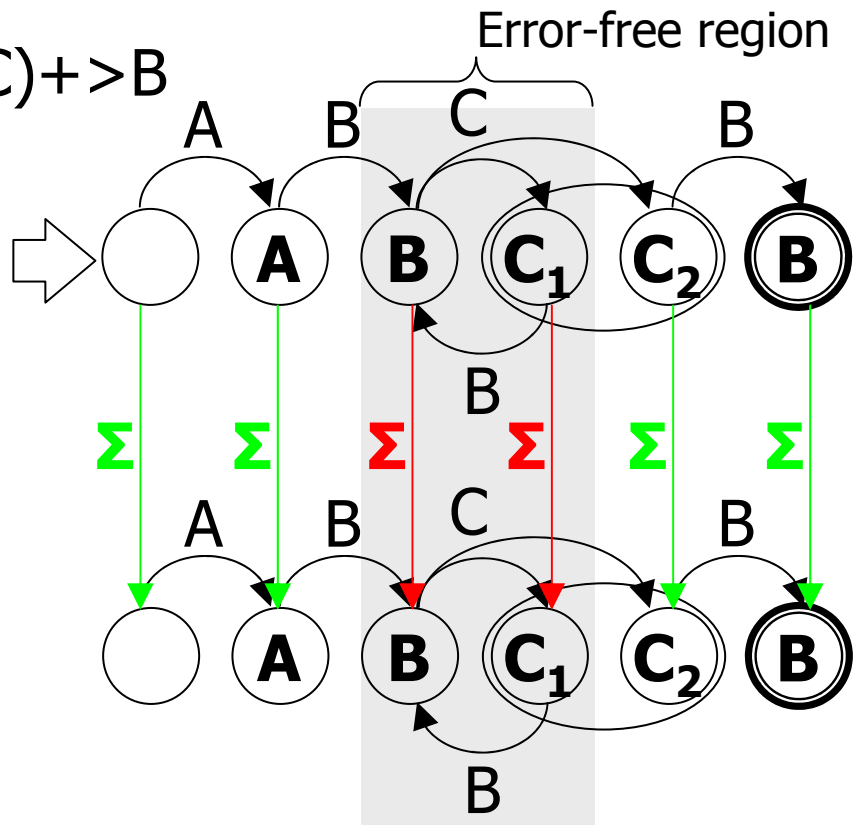
Error-free regions

$R_1' = T[R_1]\ \&\ B[c] \mid R_0 \mid T[R_0'] \mid T[R_0]$

$R_1' = \underline{T[R_1]\ \&\ B[c]} \mid \underline{???}$

no errors del

A<(BC)+>B



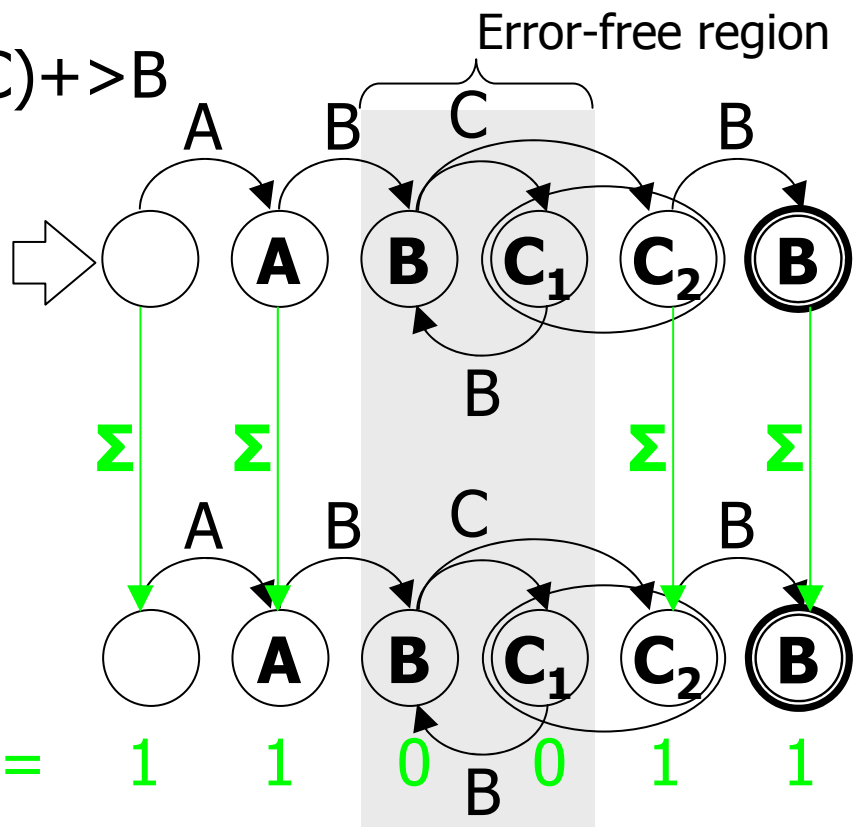
Error-free regions

$R_1' = T[R_1]\ \&\ B[c]\ |\ R_0\ \ |\ T[R_0']\ |\ T[R_0]$

$R_1' = \underline{T[R_1]\ \&\ B[c]}\ |\ \underline{R_0\ \&\ I}\ |\ \ldots$

no errors del

A<(BC)+>B



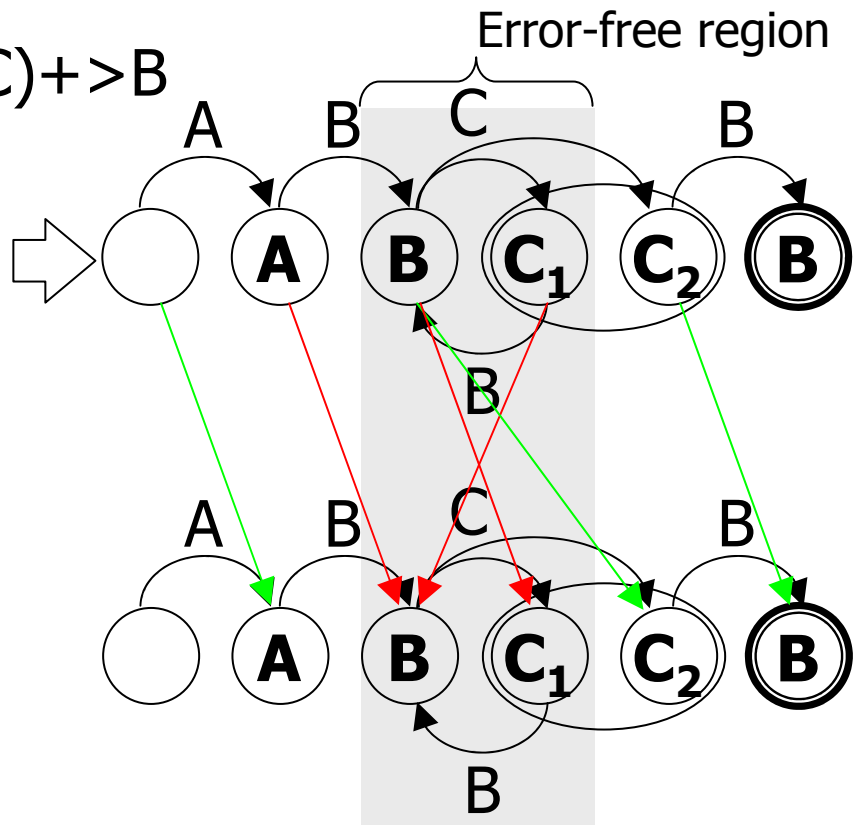
Error-free regions

$R_1' = T[R_1]\ \&\ B[c] \mid R_0 \mid T[R_0'] \mid T[R_0]$

$R_1' = \underline{T[R_1]\ \&\ B[c]} \mid \underline{R_0\ \&\ I} \mid \underline{???}$

no errors del ins

A<(BC)+>B



ε - transitions

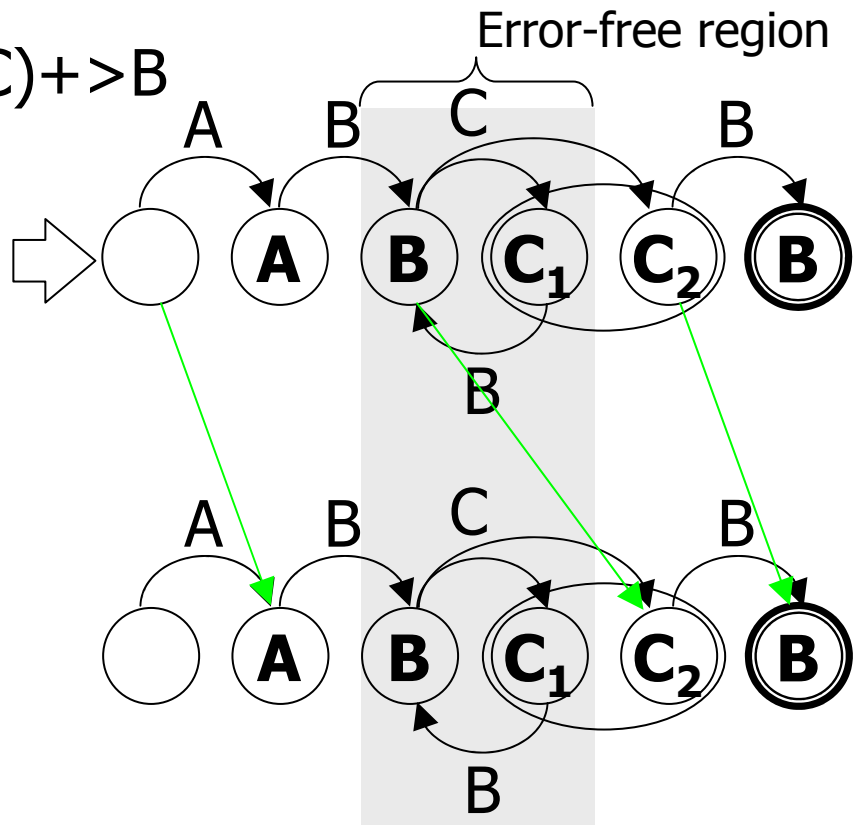
Error-free regions

$R_1' = T[R_1]\ \&\ B[c] \mid R_0 \mid T[R_0'] \mid T[R_0]$

$R_1' = \underline{T[R_1]\ \&\ B[c]} \mid \underline{R_0\ \&\ I} \mid \underline{T_e[R_0']} \mid \ldots$

no errors del ins

A<(BC)+>B



ε - transitions

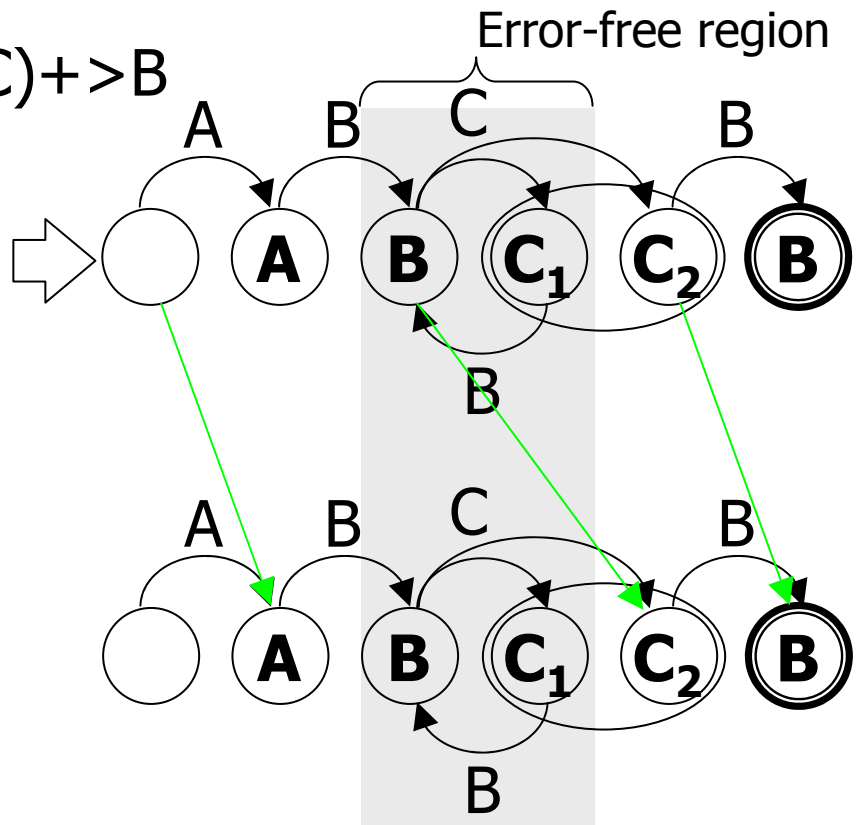
Error-free regions

$R_1' = T[R_1] \,\&\, B[c] \mid R_0 \mid T[R_0'] \mid T[R_0]$

$R_1' = \underline{T[R_1] \,\&\, B[c]} \mid \underline{R_0 \,\&\, I} \mid \underline{T_e[R_0']} \mid \underline{T_e[R_0]}$

no errors del ins subst

A<(BC)+>B



Σ - transitions