COMP20121 The Implementation and Power of Computer Languages

‘Power’ Part

http://www.cs.man.ac.uk/~petera/2121/index.html .

Peter Aczel

room: CS2.52, tel: 56155

email: petera@cs.man.ac.uk

Department of Computer Science, University of Manchester

COMP20121, ‘power’ part: section 1

lectures 2,3: Finite automata for regular languages

LECTURE FOUR

- Some properties of Regular Languages
- Start of section 2: Context-free Grammars

Properties of regular languages

Proposition

- *Every finite language is regular.*

Properties of regular languages

Proposition

- *Every finite language is regular.*
- *If L is a regular language then so are L^n (for all $n \in \mathbb{N}$), L^*, $\overline{L}$ and L^R.*

Properties of regular languages

Proposition

- *Every finite language is regular.*
- *If L is a regular language then so are L^n (for all $n \in \mathbb{N}$), L^*, $\overline{L}$ and L^R.*
- *If L and L' are regular languages then so are $L \cup L'$, $L \cap L'$ and $L \cdot L'$.*

Proofs—I

We first show that every finite language is regular:

Proofs—I

We first show that every finite language is regular:

If $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ is a finite language then $\alpha_1|\alpha_2|\cdots|\alpha_n$ is a pattern for the language.

Proofs—II

If L is regular then so are L^n and L^*:

Proofs—II

If L is regular then so is L^n:

For if p is a pattern for L then $?$ is a pattern for L^n,

Proofs—II

If L is regular then so is L^n:

For if p is a pattern for L then $\underbrace{pp\cdots p}_{n}$ is a pattern for L^n, and

Proofs—II

If L is regular then so is L^n:

For if p is a pattern for L then $\underbrace{pp\cdots p}_{n}$ is a pattern for L^n, and $?$ is one for L^*.

Proofs—II

If L is regular then so is L^n:

For if p is a pattern for L then $\underbrace{pp\cdots p}_{n}$ is a pattern for L^n, and $p*$ is one for L^*.

Proofs—II

If L is regular then so is its complement, $\overline{L}$.

This follows from Assessed Exercise 1(b) and Theorem 1.5.

Proofs—II

If L is regular then so is L^R.

This is an exercise.

Proofs—III

If L and L' are regular languages then so is $L \cup L'$:

Proofs—III

If L and L' are regular languages then so is $L \cup L'$:

If p is a pattern for L and p' one for L' then $?$ is a pattern for $L \cup L'$.

Proofs—III

If L and L' are regular languages then so is $L \cup L'$:

If p is a pattern for L and p' one for L' then $p|p'$ is a pattern for $L \cup L'$.

Proofs—III

If L and L' are regular languages then so is $L \cdot L'$:

Proofs—III

If L and L' are regular languages then so is $L \cdot L'$:

If p is a pattern for L and p' one for L' then $?$ is a pattern for $L \cdot L'$.

Proofs—III

If L and L' are regular languages then so is $L \cdot L'$:

If p is a pattern for L and p' one for L' then pp' is a pattern for $L \cdot L'$.

Proofs—III

If L and L' are regular languages then so is $L \cap L'$.

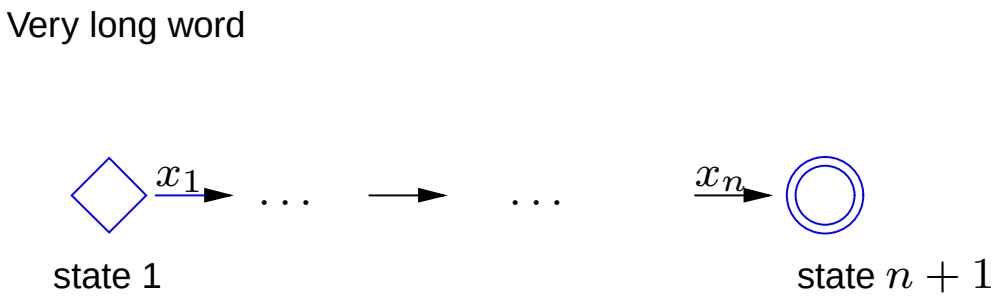
This is again an exercise.

Long words in regular languages

Assume that a DFA has n states. Then every word of length at least n will go through (at least) one state twice.

Long words in regular languages

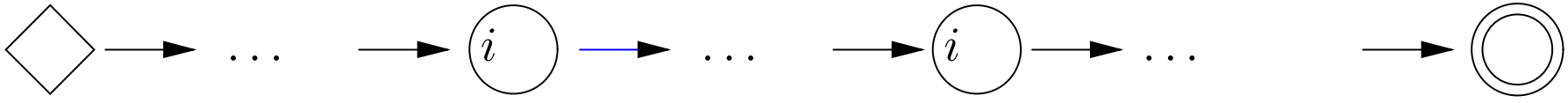
Assume that a DFA has n states. Then every word of length at least n will go through (at least) one state twice.



Long words in regular languages

Assume that a DFA has n states. Then every word of length at least n will go through (at least) one state twice.

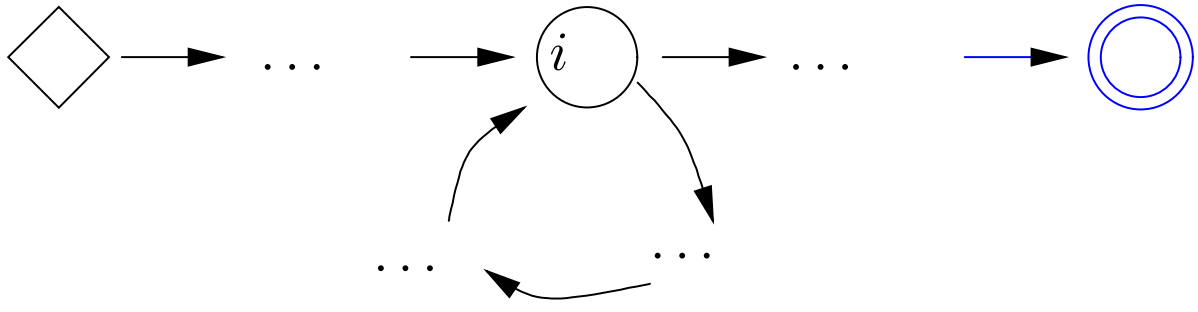
Length of word $\geq n$, where n is the number of states



Long words in regular languages

Assume that a DFA has n states. Then every word of length at least n will go through (at least) one state twice.

Length of word $\geq n$, where n is the number of states



The Pumping Lemma

Lemma (The Pumping Lemma)

Let L be an infinite regular language. Then there exists a natural number $n > 0$ such that every word α of L consisting of at least n characters contains a 'pumping section' in the following sense.

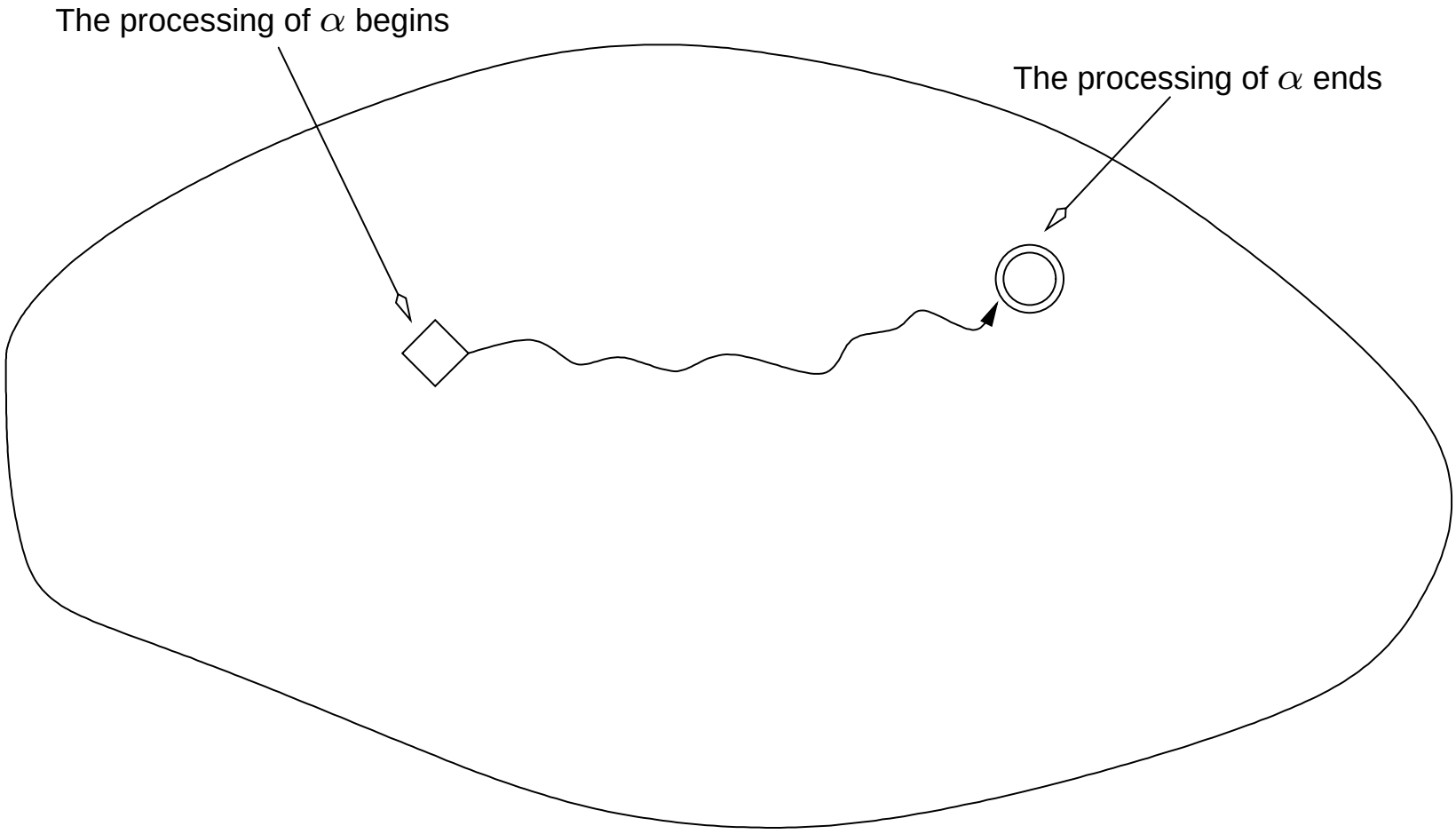
The Pumping Lemma

The word α can be split up into three parts, say λ, μ, and ν such that

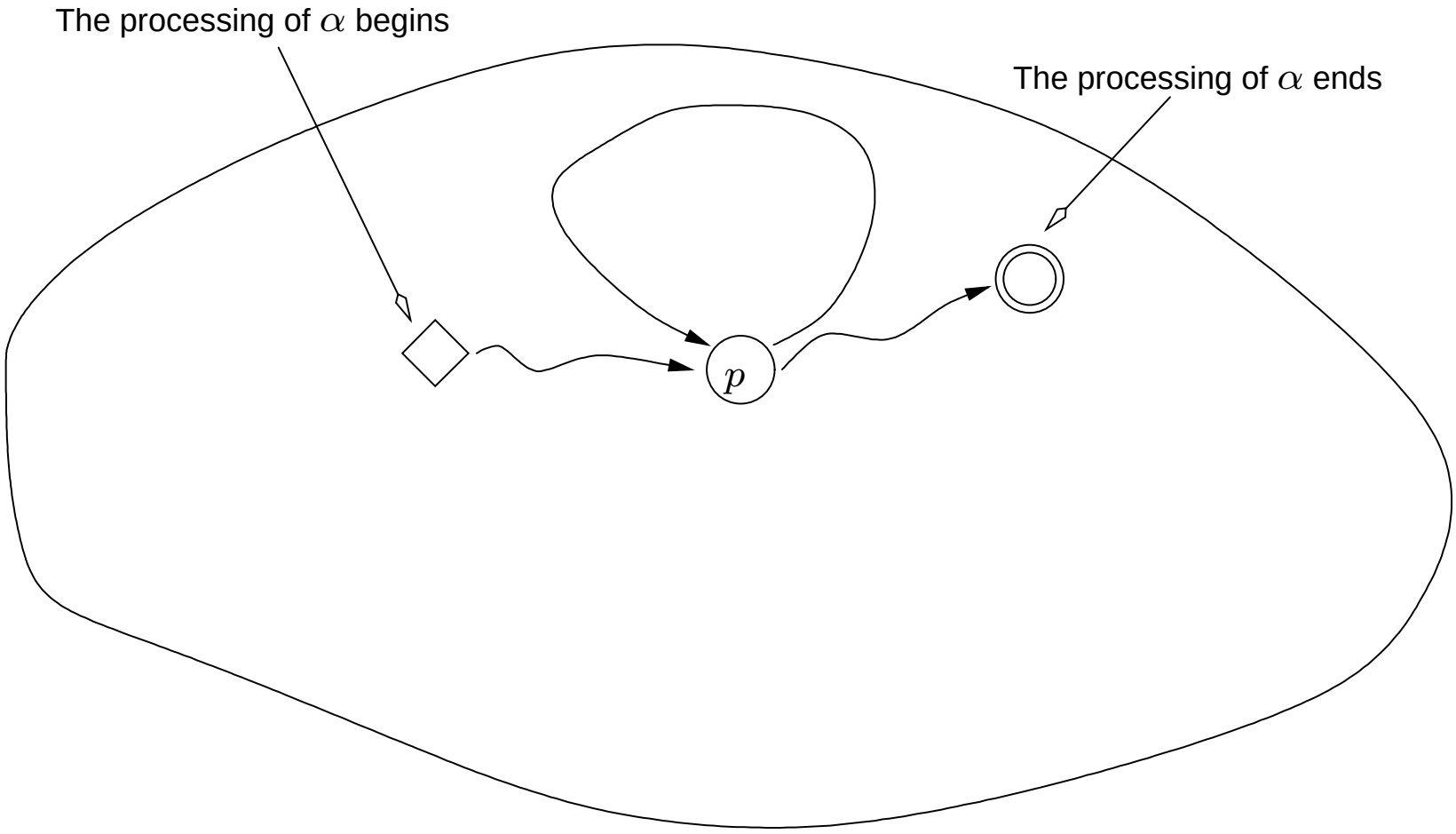
- $\alpha = \lambda\mu\nu$ *and* $\mu \neq \epsilon$;
- *the length of $\lambda\mu$ is at most n*

and so that all words of the form $\lambda\mu^k\nu$, for $k \in \mathbb{N}$, belong to L.

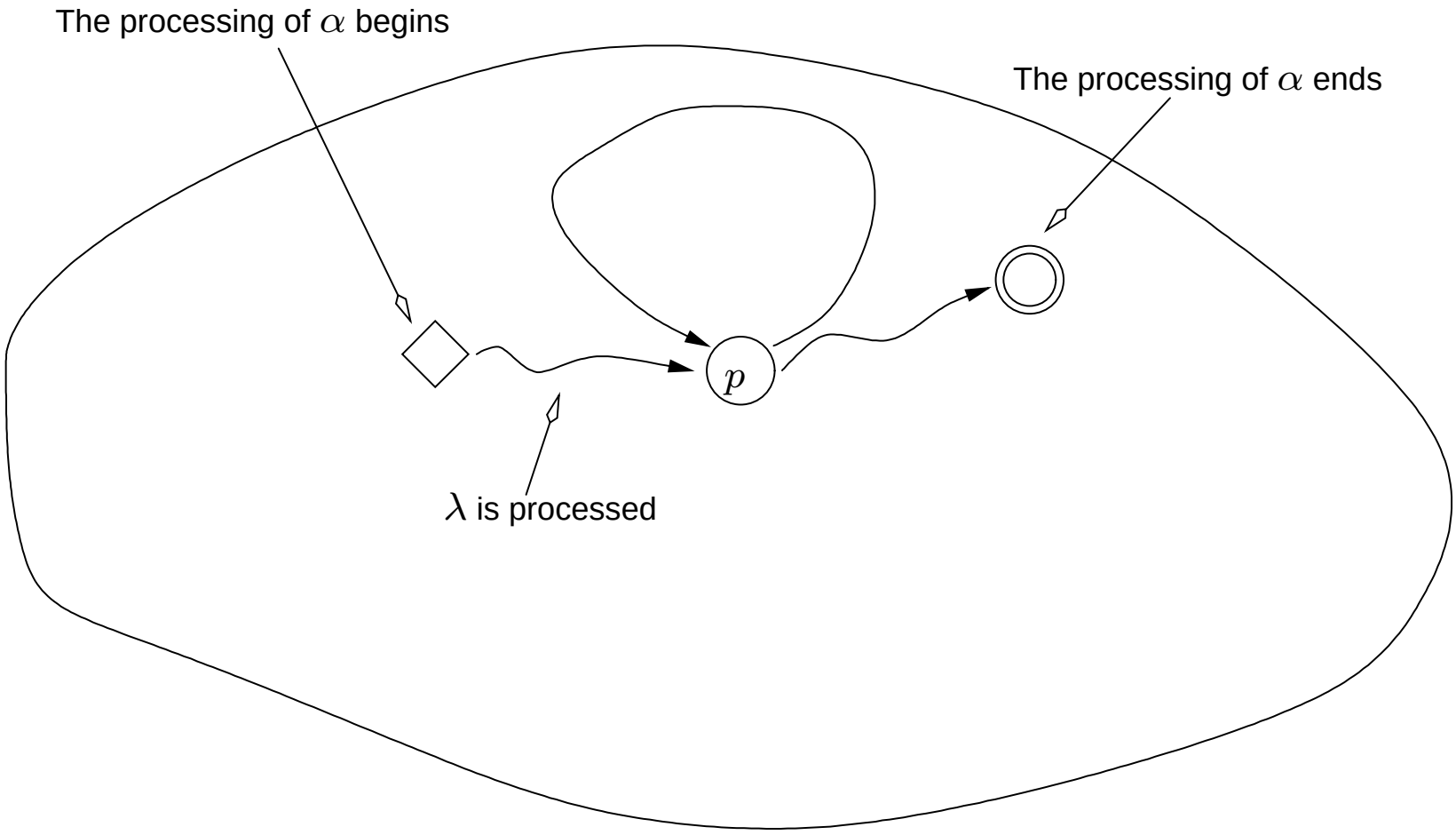
The Pumping Lemma–II



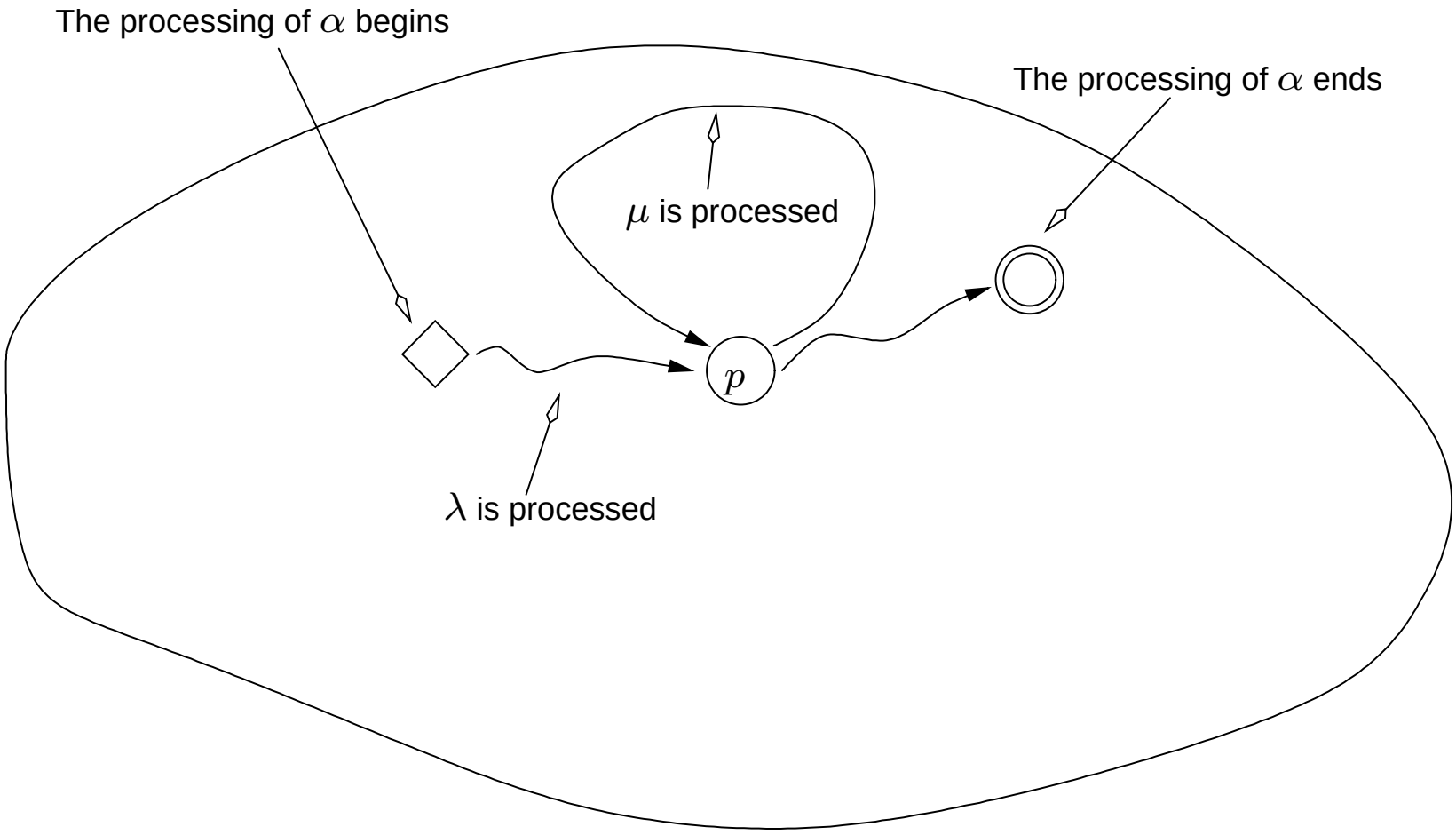
The Pumping Lemma–II



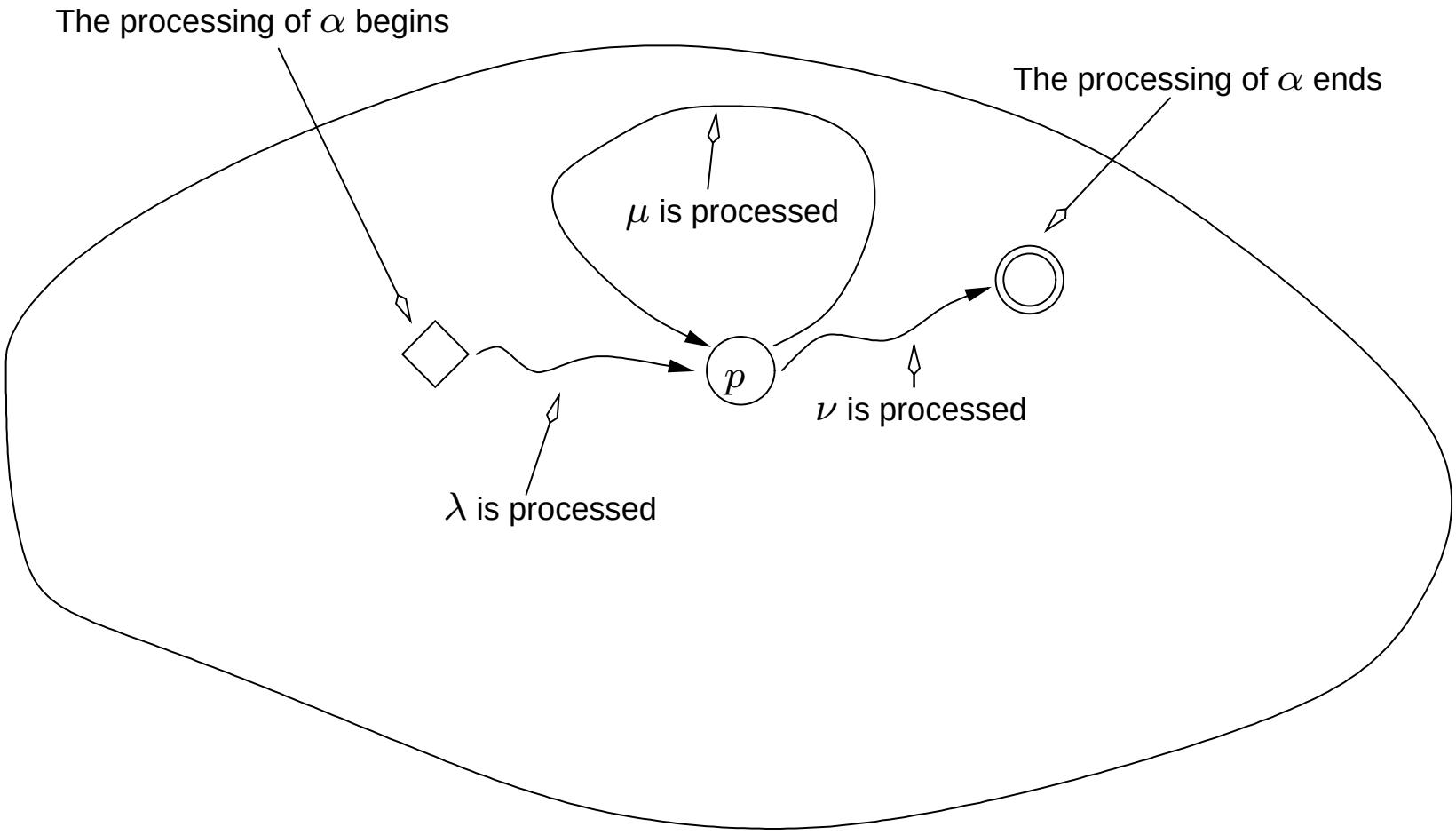
The Pumping Lemma–II



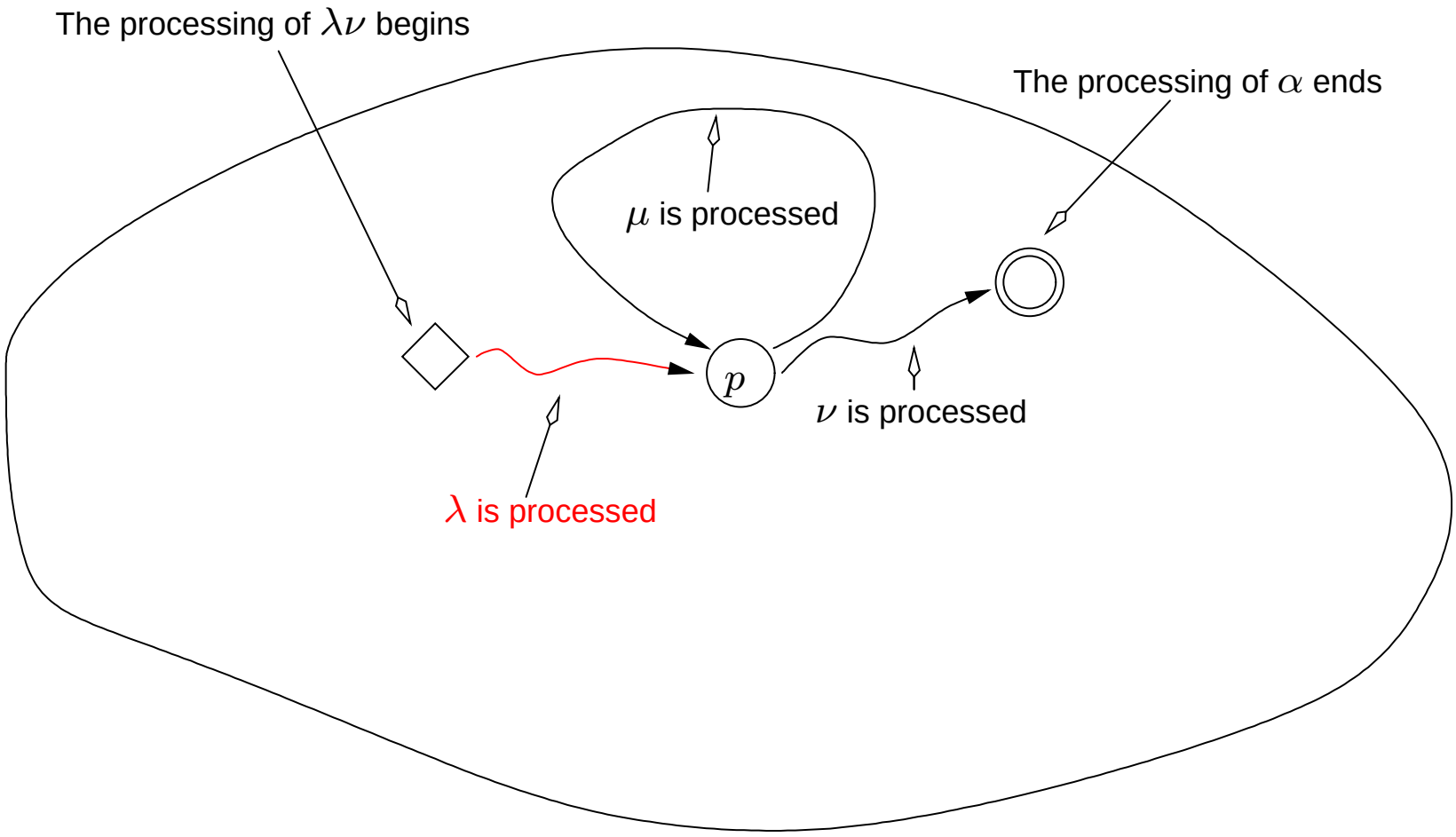
The Pumping Lemma–II



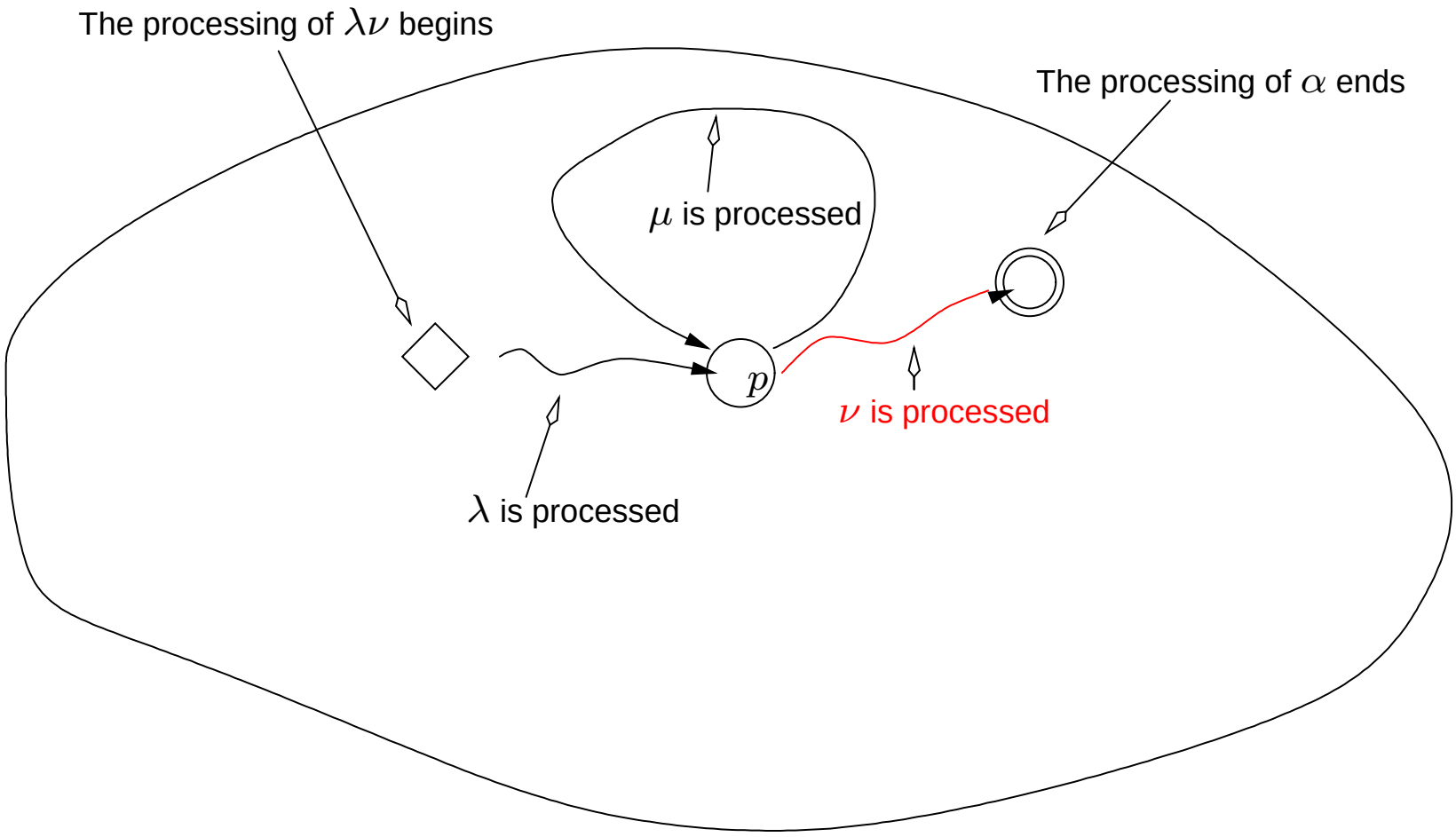
The Pumping Lemma–II



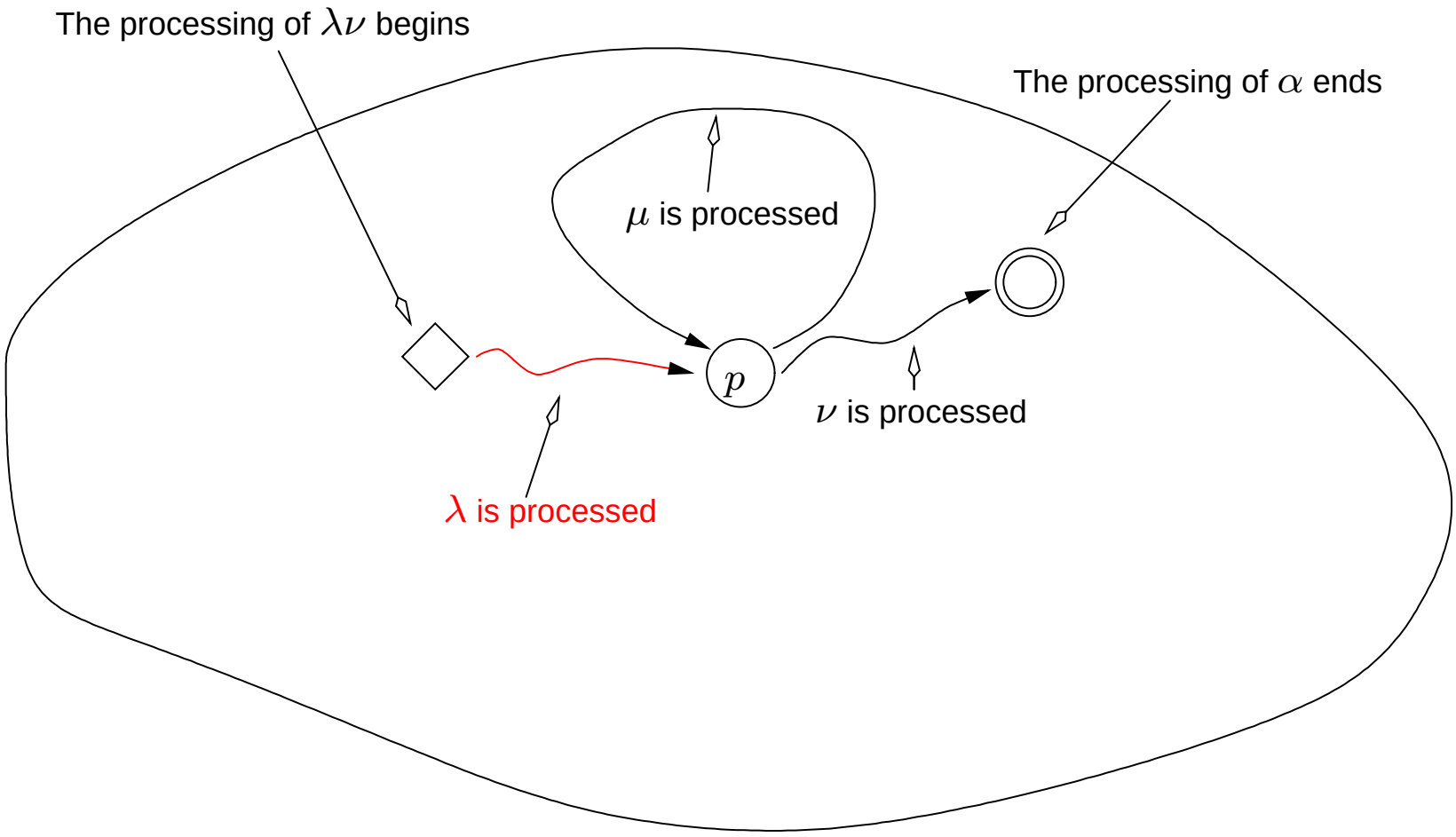
Processing $\lambda\nu$



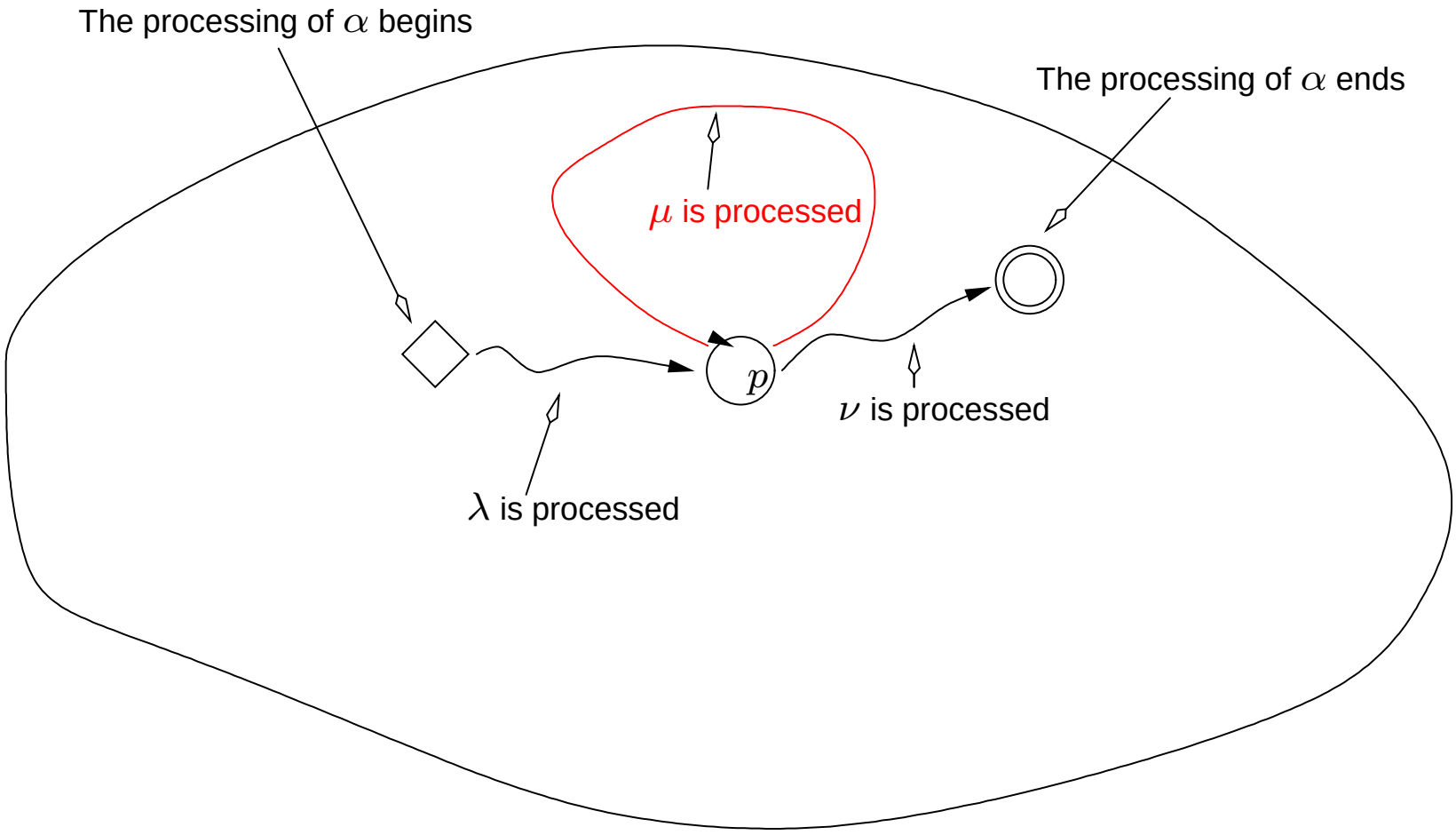
Processing $\lambda\nu$



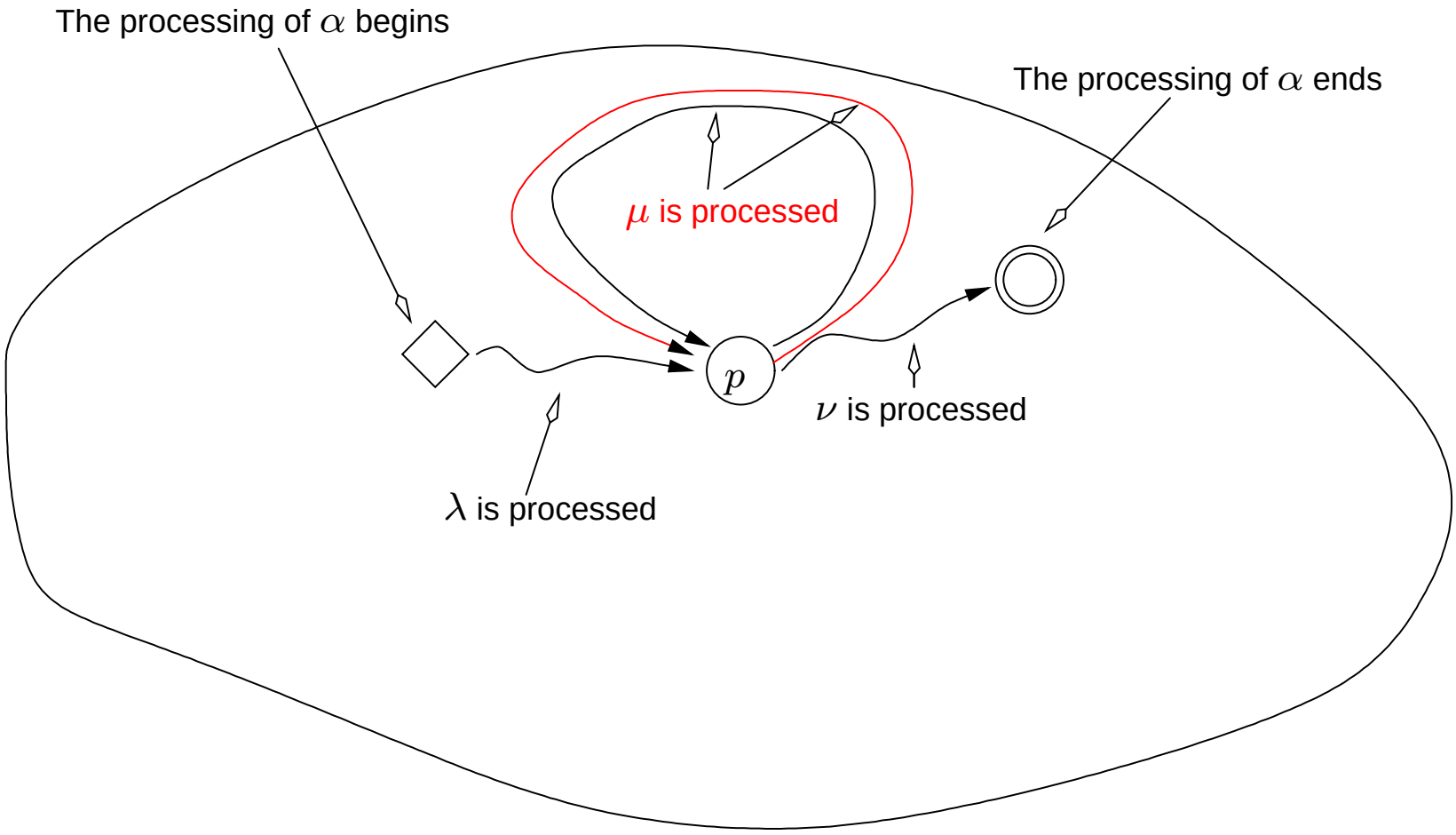
Processing $\lambda\mu^2\nu$



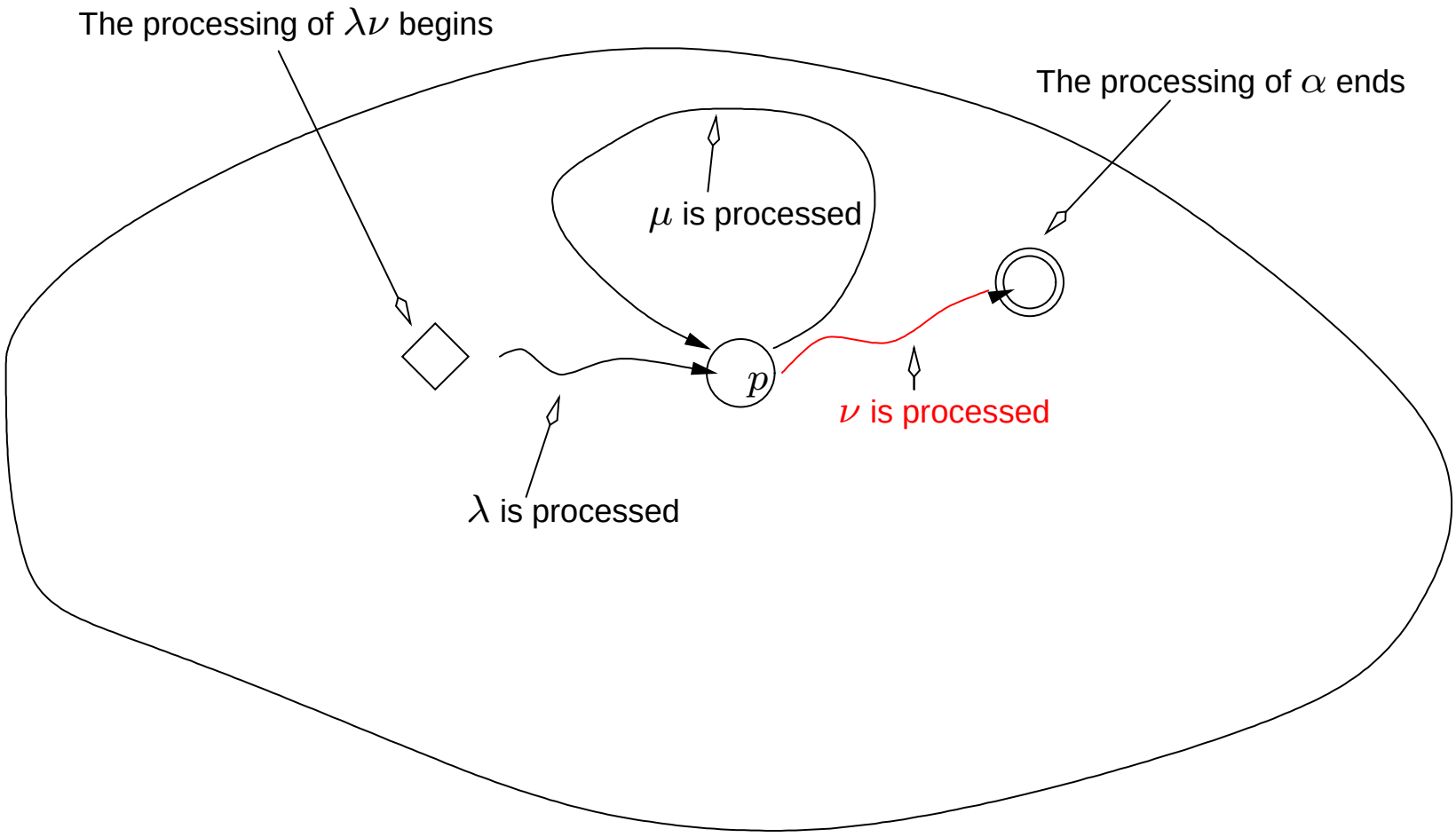
Processing $\lambda\mu^2\nu$



Processing $\lambda\mu^2\nu$



Processing $\lambda\mu^2\nu$



Pumping Lemma: Example

The language $L = \{0^i 1^i \mid i \in \mathbb{N}\}$ is not regular.

- Assume n is as in the Pumping Lemma.

Pumping Lemma: Example

The language $L = \{0^i 1^i \mid i \in \mathbb{N}\}$ is not regular.

- Assume n is as in the Pumping Lemma.
- Consider $0^n 1^n$; assume λ, μ, ν as in the Pumping Lemma.

Pumping Lemma: Example

- Since the length of $\lambda\mu$ is at most n and $\mu \neq \epsilon$, we know that

$$\overbrace{\underbrace{00\cdots}_{\lambda}\underbrace{\cdots}_{\mu}00}^{n}\overbrace{1\cdots1}^{n}.$$

Pumping Lemma: Example

- Since the length of $\lambda\mu$ is at most n and $\mu \neq \epsilon$, we know that

$$\overbrace{\underbrace{00\cdots}_{\lambda}\underbrace{\cdots 00}_{\mu}}^{n}\overbrace{1\cdots 1}^{n}.$$

- But then $\lambda\nu = \lambda\mu^0\nu$ is not in L:

$$\overbrace{\underbrace{00\cdots 0}_{\lambda}}^{<n}\overbrace{1\cdots 1}^{n}.$$

Section 1 summary

- Regular languages are describable via patterns (regular expressions).

Section 1 summary

- Regular languages are describable via patterns (regular expressions).
- Regular languages are precisely the languages accepted by finite deterministic automata and

Section 1 summary

- Regular languages are describable via patterns (regular expressions).
- Regular languages are precisely the languages accepted by finite deterministic automata and
- these are precisely the languages accepted by finite non-deterministic automata.

Section 1 summary

- Matching a word to a pattern is a recursive process, while checking acceptance with a deterministic automaton is straightforward.

Section 1 summary

- Matching a word to a pattern is a recursive process, while checking acceptance with a deterministic automaton is straightforward.
- The Pumping Lemma says that infinite regular languages contain strings where a ‘pumping section’ can be repeated arbitrarily often.

Section 1 summary

- Regular languages are not powerful enough for the parsing of programming languages.

LECTURE FOUR (ctd)

Section 2: Context Free Grammars

Context-free grammars

Definition *A* **context-free grammar** *is given by the following:*

- *an alphabet* Σ *of* terminal symbols*,* *also called the* object alphabet*;*

Context-free grammars

Definition *A* **context-free grammar** *is given by the following:*

- *an alphabet* Σ *of* terminal symbols, *also called the* object alphabet*;*
- *an alphabet* N *of* non-terminal symbols, *the elements of which are also referred to as* auxiliary characters, placeholders *or, in some books,* variables, *where* $N \cap \Sigma = \emptyset$*;*

Context-free grammars

- *a special non-terminal symbol* $S \in N$ *called the* start symbol*;*

Context-free grammars

- *a special non-terminal symbol* $S \in N$ *called the* start symbol*;*
- *a finite set of* production rules*, i.e. strings of the form* $R \to \Gamma$ *where* $R \in N$ *is a non-terminal symbol and* $\Gamma \in (\Sigma \cup N)^*$ *is an arbitrary string of terminal and non-terminal symbols. The rule can be read as*

  'R can be replaced by Γ*'.*

Context-free grammars: example

$$\Sigma = \{(,\ ),\ +,\ -,\ \times,\ /,\ 0,\ 1,\ 2,\ \ldots\}$$

$$N = \{S, B\}$$

Context-free grammars: example

$$\Sigma = \{(,\ ),\ +,\ -,\ \times,\ /,\ 0,\ 1,\ 2,\ \ldots\}$$

$$N = \{S, B\}$$

$$B \to 0 \qquad S \to B \qquad S \to (S)$$

$$B \to 1 \qquad S \to S + S \qquad S \to S - S$$

$$B \to 2 \qquad S \to S \times S \qquad S \to S/S$$

$$\vdots$$

Context-free grammars: example

$$\Sigma = \{(,\ ),\ +,\ -,\ \times,\ /,\ 0,\ 1,\ 2,\ \ldots\}$$

$$N = \{S, B\}$$

Or, shorter:

$$B \rightarrow 0 \,|\, 1 \,|\, 2 \,|\, \ldots$$

$$S \rightarrow B \,|\, (S) \,|\, S + S \,|\, S - S$$

$$|\, S \times S \,|\, S/S$$

Generating strings: example

Generating strings:

Generating strings: example

Generating strings:

S

Generating strings: example

Generating strings:

$$S \Rightarrow S \times S$$

Generating strings: example

Generating strings:

$$S \Rightarrow S \times S \Rightarrow S \times (S)$$

Generating strings: example

Generating strings:

$$S \Rightarrow S \times S \Rightarrow S \times (S) \Rightarrow B \times (S)$$

Generating strings: example

Generating strings:

$$S \Rightarrow S \times S \Rightarrow S \times (S) \Rightarrow B \times (S)$$

$$\Rightarrow B \times (S + S)$$

Generating strings: example

Generating strings:

$$S \Rightarrow S \times S \Rightarrow S \times (S) \Rightarrow B \times (S)$$

$$\Rightarrow B \times (S + S) \Rightarrow B \times (B + S)$$

Generating strings: example

Generating strings:

$$S \Rightarrow S \times S \Rightarrow S \times (S) \Rightarrow B \times (S)$$

$$\Rightarrow B \times (S + S) \Rightarrow B \times (B + S) \Rightarrow$$

$$2 \times (B + S)$$

Generating strings: example

Generating strings:

$$S \Rightarrow S \times S \Rightarrow S \times (S) \Rightarrow B \times (S)$$

$$\Rightarrow B \times (S + S) \Rightarrow B \times (B + S) \Rightarrow$$

$$2 \times (B + S) \Rightarrow 2 \times (3 + S)$$

Generating strings: example

Generating strings:

$$S \Rightarrow S \times S \Rightarrow S \times (S) \Rightarrow B \times (S)$$

$$\Rightarrow B \times (S + S) \Rightarrow B \times (B + S)$$

$$\Rightarrow 2 \times (B + S) \Rightarrow 2 \times (3 + S) \Rightarrow$$

$$2 \times (3 + B)$$

Generating strings: example

Generating strings:

$$S \Rightarrow S \times S \Rightarrow S \times (S) \Rightarrow B \times (S)$$

$$\Rightarrow B \times (S + S) \Rightarrow B \times (B + S)$$

$$\Rightarrow 2 \times (B + S) \Rightarrow 2 \times (3 + S) \Rightarrow$$

$$2 \times (3 + B) \Rightarrow 2 \times (3 + 4)$$

Generating strings: Parse tree

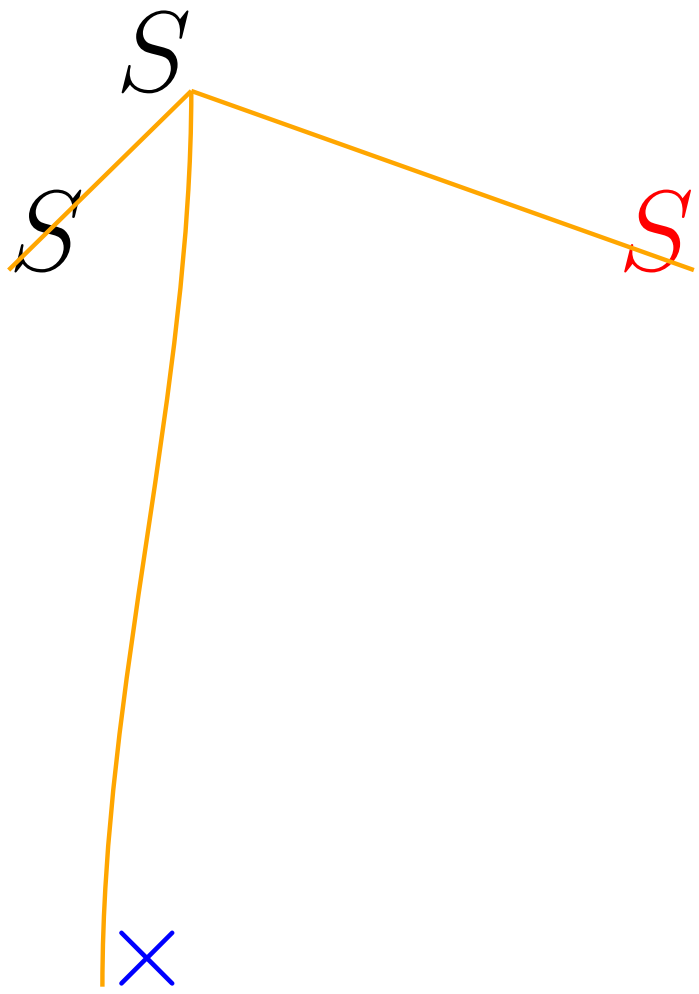
$$S \Rightarrow \cdots \Rightarrow 2 \times (3 + 4)$$

Generating strings: Parse tree

$$S \Rightarrow \cdots \Rightarrow 2 \times (3 + 4)$$
$$S$$

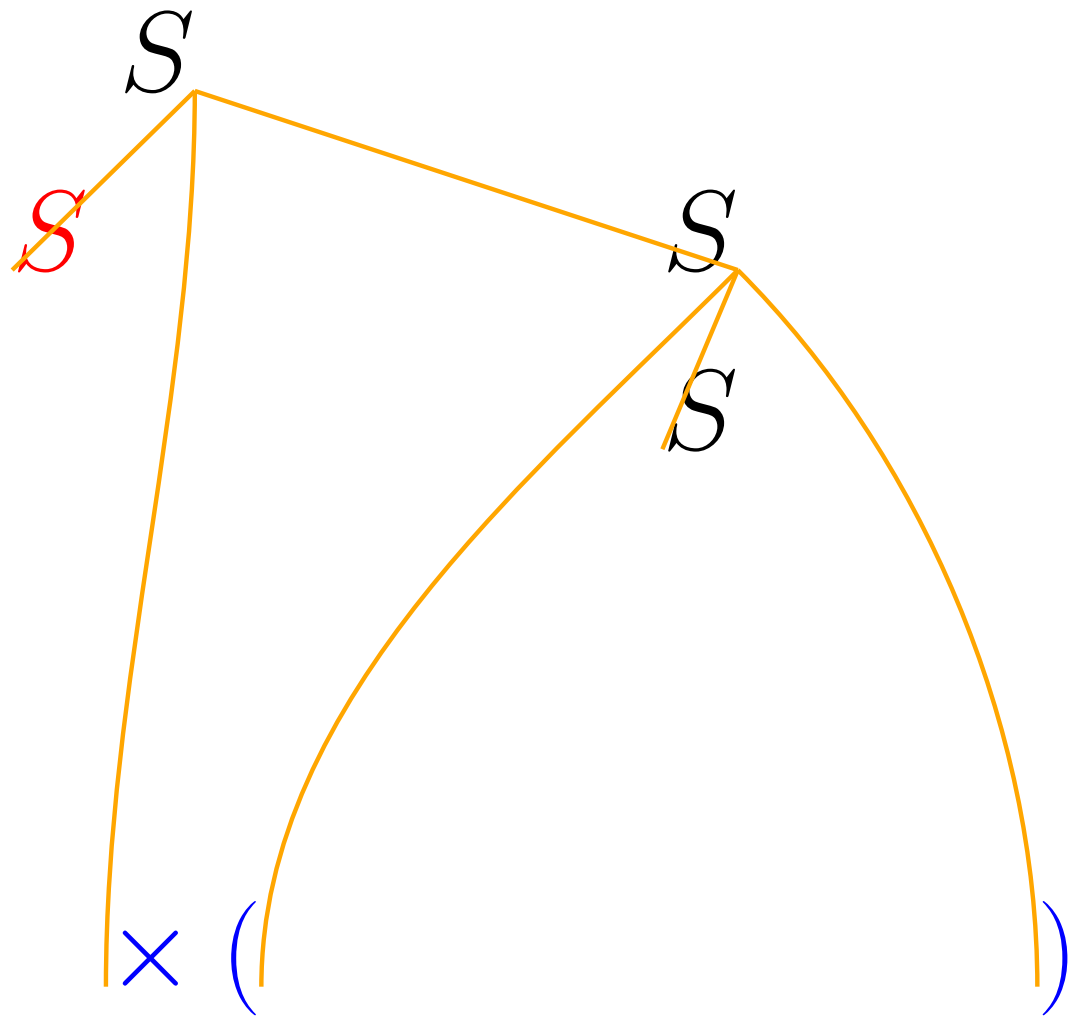
Generating strings: Parse tree

$$S \Rightarrow \cdots \Rightarrow 2 \times (3 + 4)$$



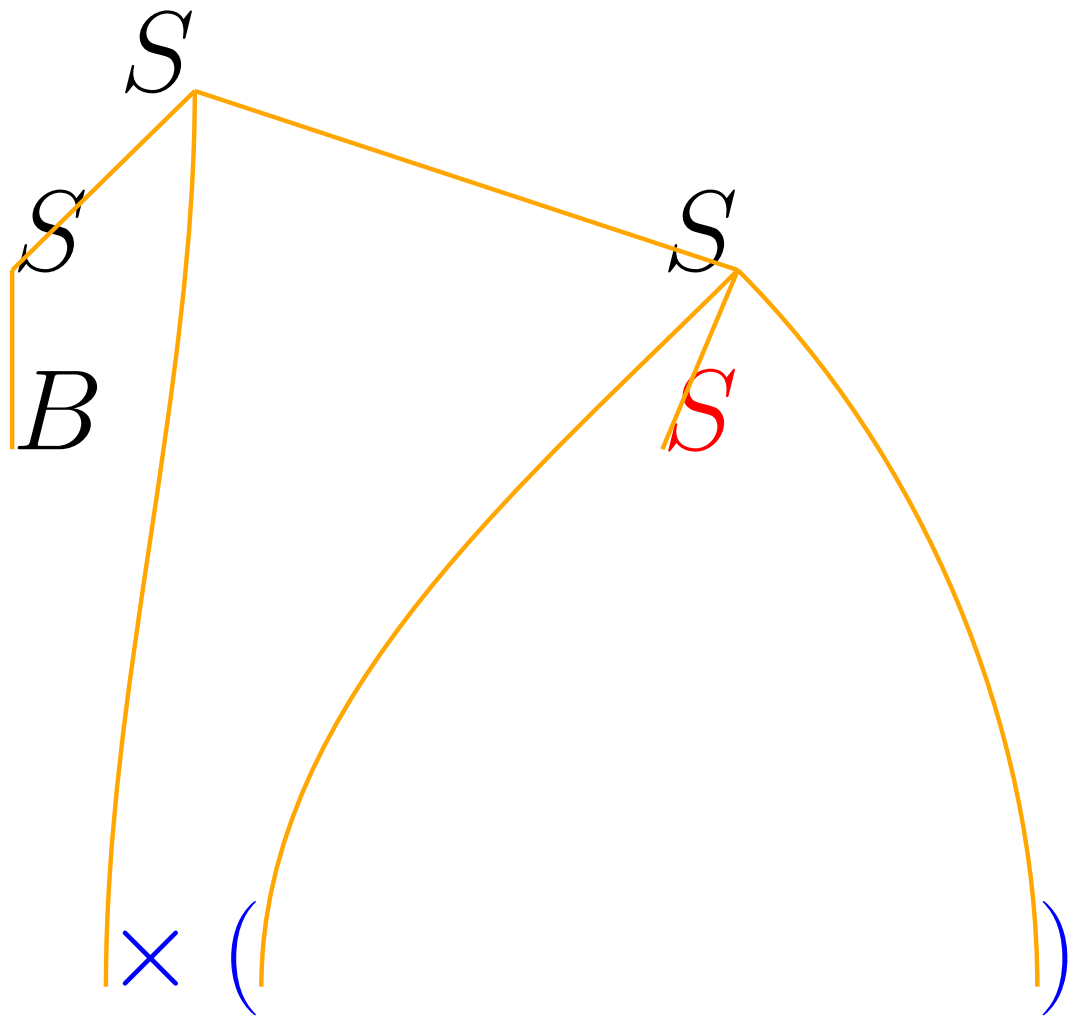
Generating strings: Parse tree

$$S \Rightarrow \cdots \Rightarrow 2 \times (3 + 4)$$



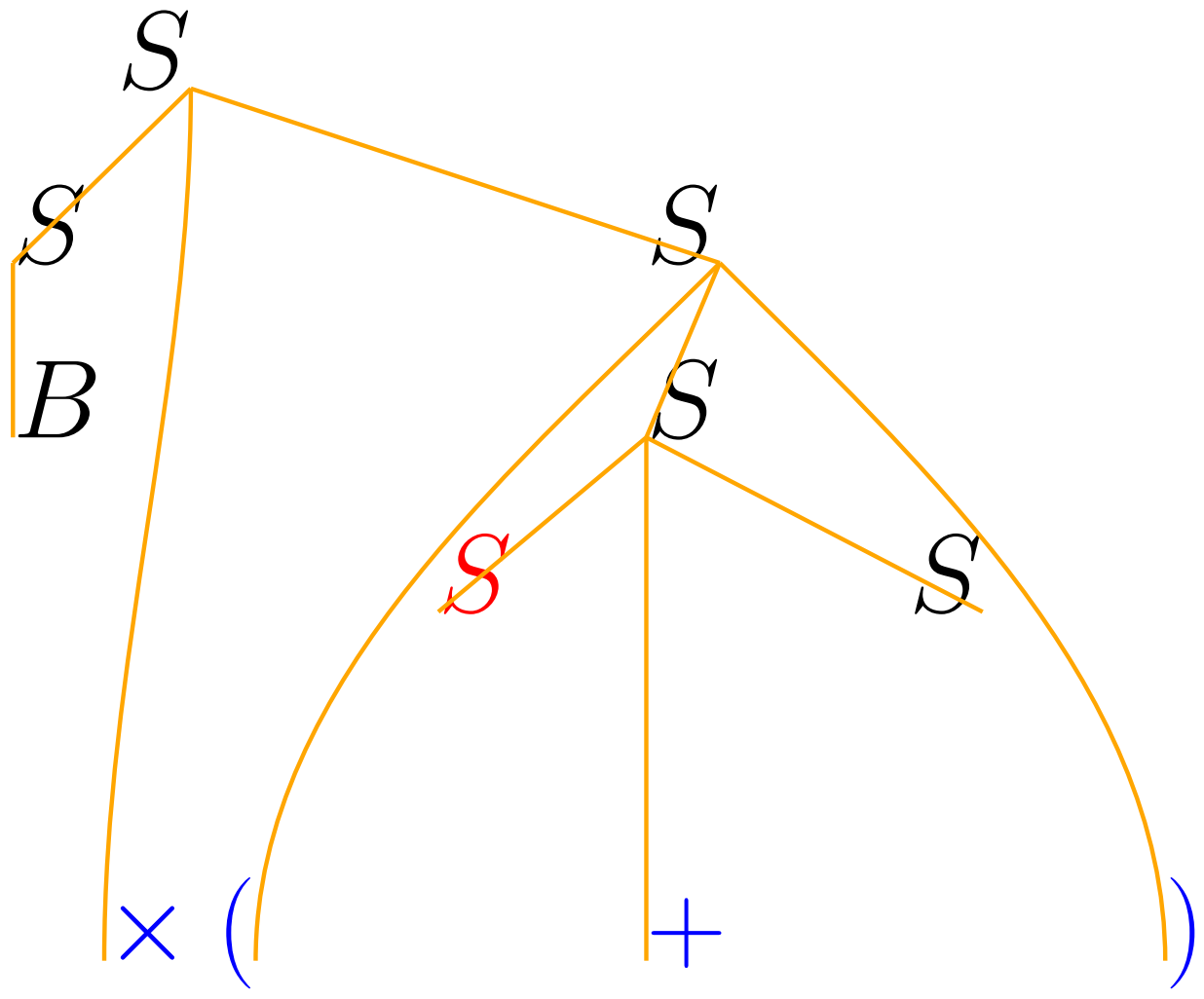
Generating strings: Parse tree

$$S \Rightarrow \cdots \Rightarrow 2 \times (3 + 4)$$



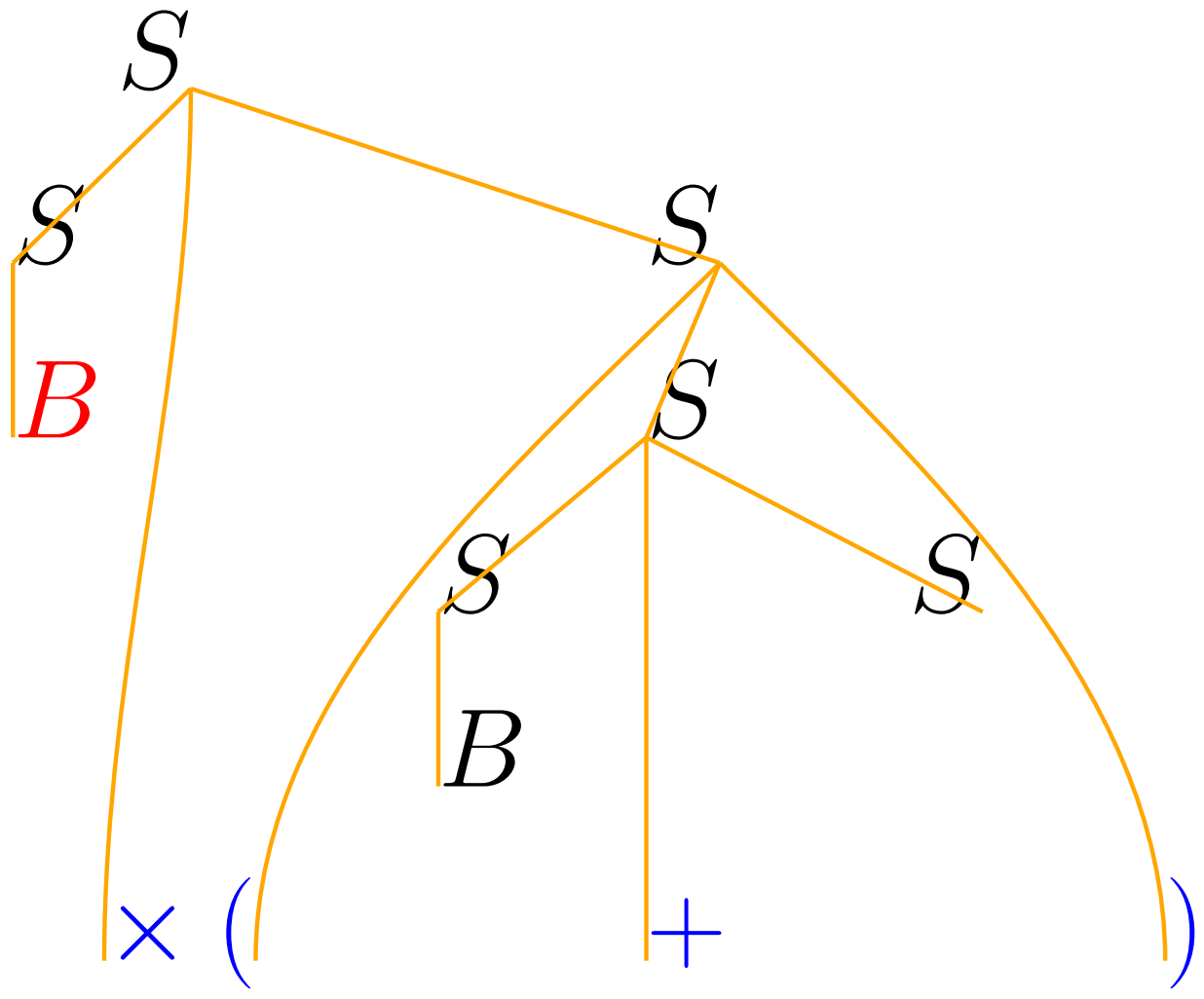
Generating strings: Parse tree

$$S \Rightarrow \cdots \Rightarrow 2 \times (3 + 4)$$



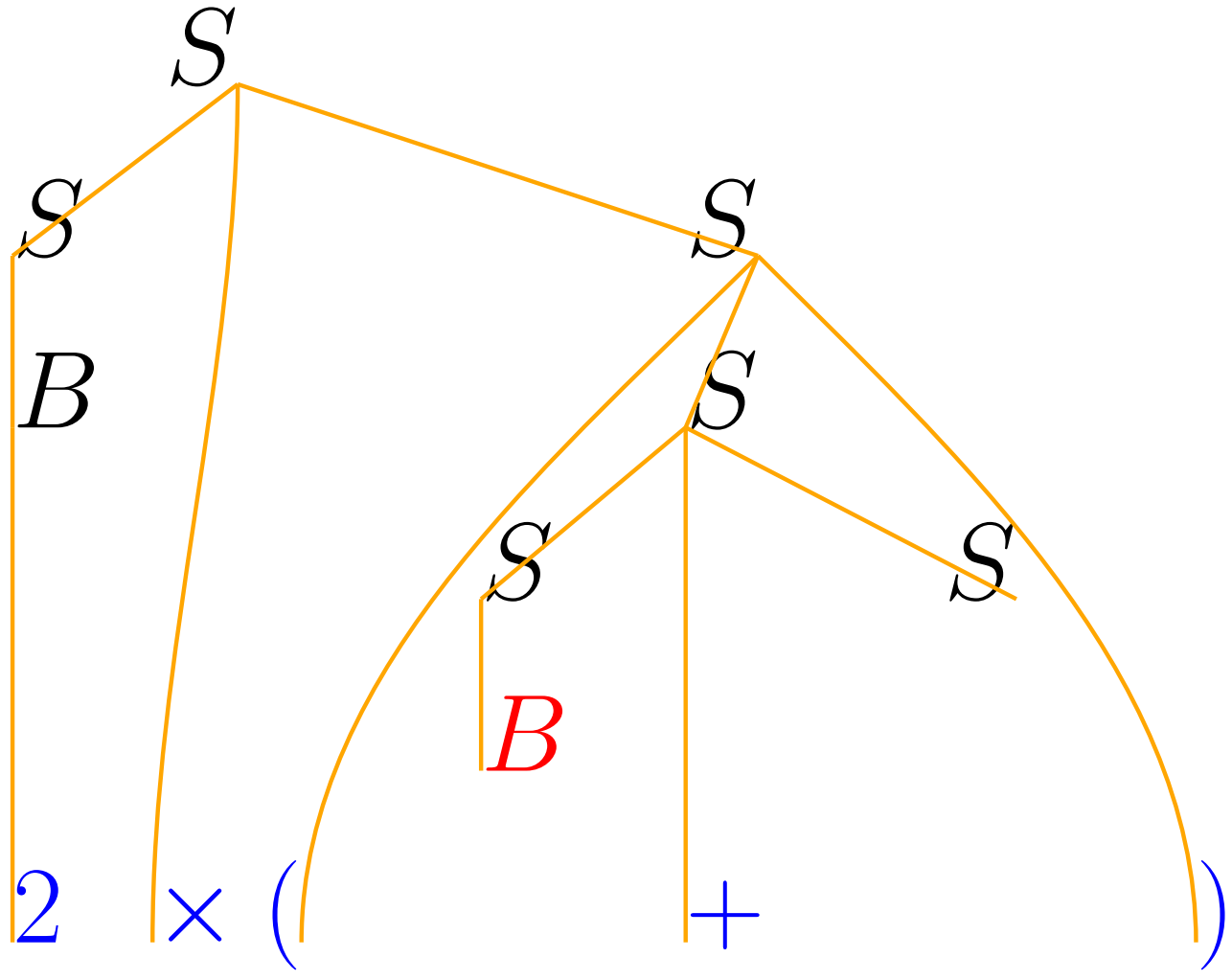
Generating strings: Parse tree

$$S \Rightarrow \cdots \Rightarrow 2 \times (3 + 4)$$



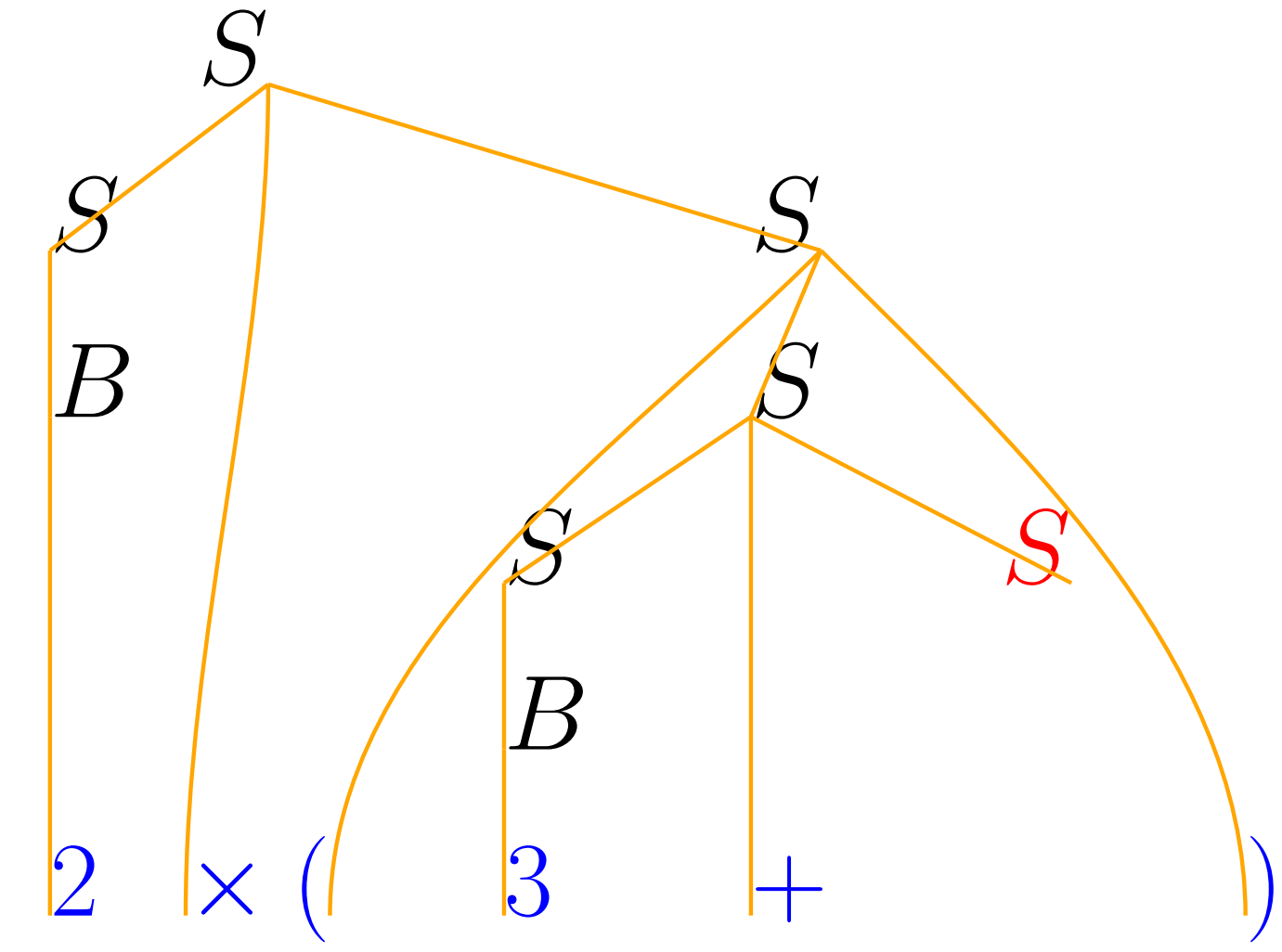
Generating strings: Parse tree

$$S \Rightarrow \cdots \Rightarrow 2 \times (3 + 4)$$



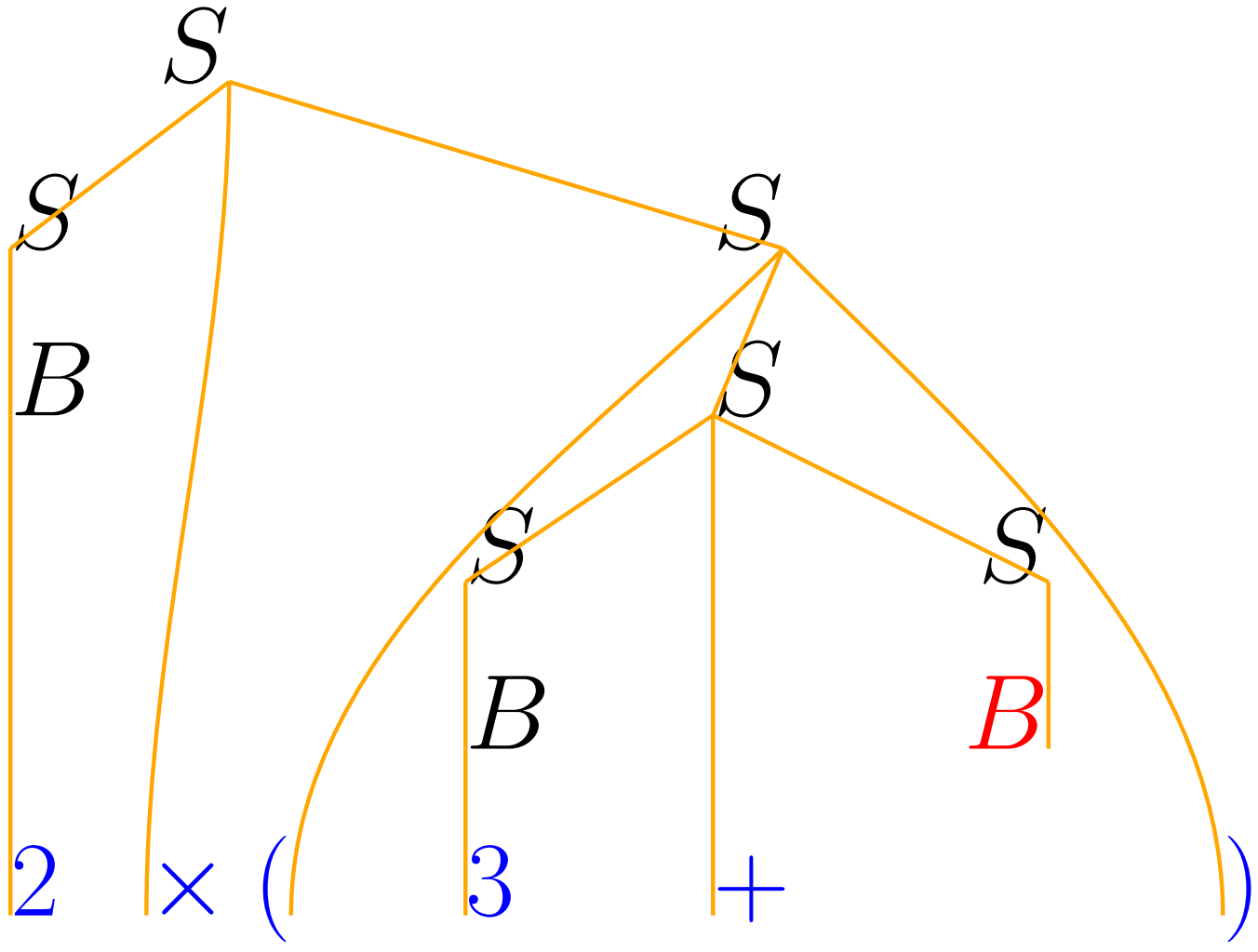
Generating strings: Parse tree

$$S \Rightarrow \cdots \Rightarrow 2 \times (3 + 4)$$



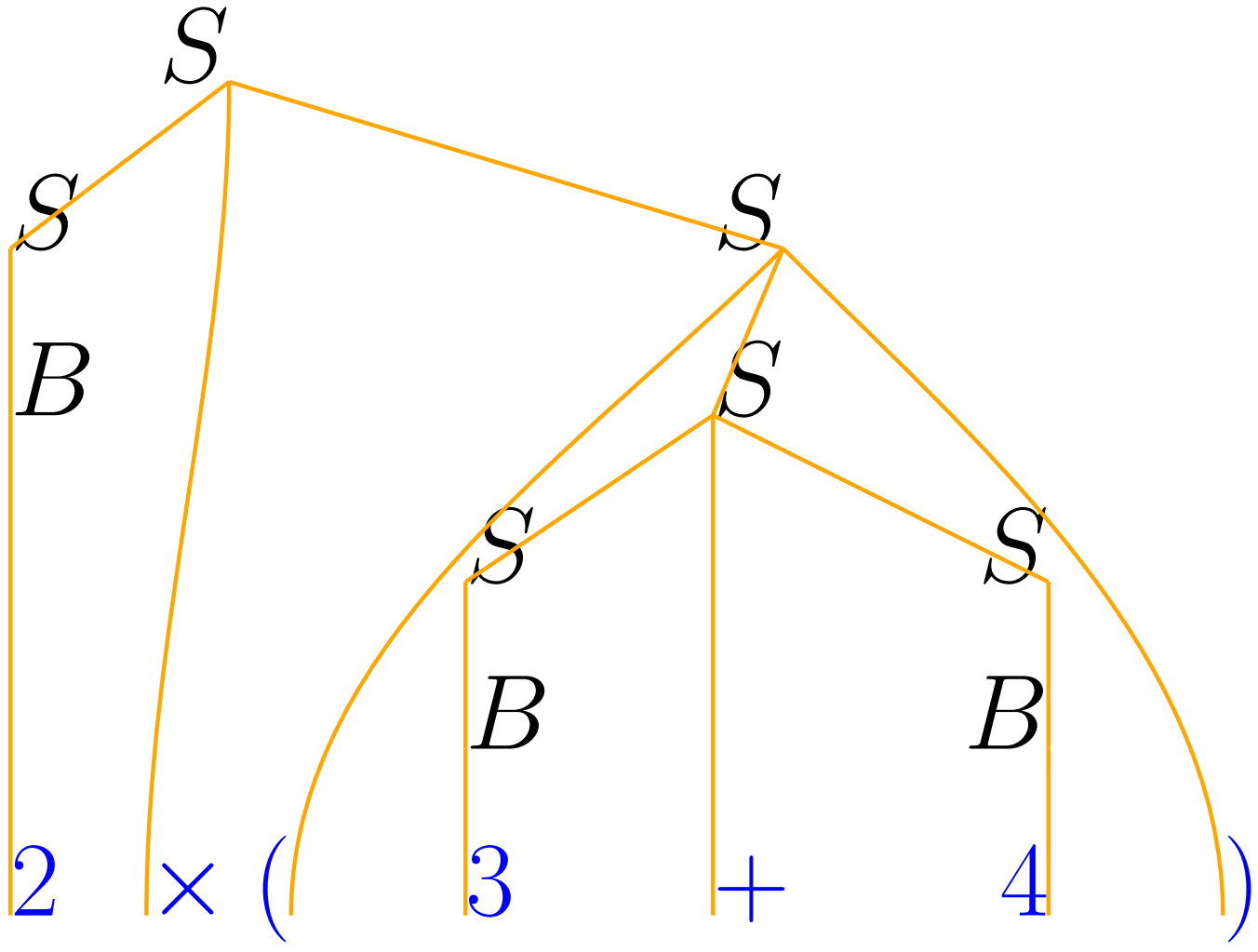
Generating strings: Parse tree

$$S \Rightarrow \cdots \Rightarrow 2 \times (3 + 4)$$



Generating strings: Parse tree

$$S \Rightarrow \cdots \Rightarrow 2 \times (3 + 4)$$



Generating strings: formally

Definition *A string* $\Gamma \in (\Sigma \cup N)^*$ **is generated by a grammar** G *if there is a sequence of strings*

$$S = \Gamma_0 \Rightarrow \Gamma_1 \Rightarrow \cdots \Rightarrow \Gamma_n = \Gamma$$

such that each step $\Gamma_i \Rightarrow \Gamma_{i+1}$ *is obtained by the application of one of* G*'s production rules to a non-terminal occurring in* Γ_i *as follows.*

Generating strings: formally

Let $A \in N$ occur in Γ_i and assume that

there is a production rule

$$A \to \Delta.$$

Then Γ_{i+1} is obtained from Γ_i by replacing one occurrence of A in Γ_i by the string Δ.

Generating strings: formally

If $A \longrightarrow \Delta$ is a production rule, so that

$$A \in N, \Delta \in (\Sigma \cup N)^*,$$

and Γ_i is $\cdots A \cdots$ then we get

$$\Gamma_i \Rightarrow \Gamma_{i+1}$$

where Γ_{i+1} is $\cdots \Delta \cdots$.

Generated Language

Definition 11 *The* **language generated by a grammar** G *is the set of all strings* $\alpha \in \Sigma^*$ *which are generated by* G*.*

Languages generated by context-free grammars are called **context-free***.*

Regular languages are context-free

Proposition: *Every regular language is generated by a context-free grammar.*

Regular languages are context-free

Proposition: *Every regular language is generated by a context-free grammar.*

We build a grammar from the automaton $(Q, q_\bullet, F, \delta)$.

Regular languages are context-free

Proposition: *Every regular language is generated by a context-free grammar.*

We build a grammar from the automaton $(Q, q_\bullet, F, \delta)$. We put $N = Q$, $S = q_\bullet$

Regular languages are context-free

Proposition: *Every regular language is generated by a context-free grammar.*

We build a grammar from the automaton $(Q, q_\bullet, F, \delta)$. We put $N = Q$, $S = q_\bullet$ and have a rule

$q \longrightarrow xq'$ whenever $q \stackrel{x}{\longrightarrow} q'$ and a rule $q \longrightarrow \epsilon$ whenever $q \in F$.

Regular languages are context-free

Then

$$q_\bullet \xrightarrow{x_1} q_1 \xrightarrow{x_2} \cdots \xrightarrow{x_n} q_n \in F$$

if and only if

Regular languages are context-free

Then

$$q_\bullet \xrightarrow{x_1} q_1 \xrightarrow{x_2} \cdots \xrightarrow{x_n} q_n \in F$$

if and only if

$$S = q_\bullet \Rightarrow x_1 q_1 \Rightarrow x_1 x_2 q_2 \Rightarrow \cdots$$

$$\Rightarrow x_1 x_2 \cdots x_n q_n \Rightarrow x_1 x_2 \cdots x_n$$

Grammars for regular languages

Proposition

A language is regular if and only if it is generated by a context-free grammar which is subject to the restriction that all production rules have one of the forms

$$R \to xR' \text{ or } R \to x \text{ or } R \to \epsilon$$

where $R, R' \in N$ *and* $x \in \Sigma$.

Grammars for regular languages

- ‘If’: see previous Proposition.
- ‘Only if’: Build an automaton from the grammar:

Grammars for regular languages

- 'If': see previous Proposition.
- 'Only if': Build an automaton from the grammar:
  - $Q = N + \{Z\}$, $q_\bullet = S$;

Grammars for regular languages

- 'If': see previous Proposition.
- 'Only if': Build an automaton from the grammar:
  - $Q = N + \{Z\}$, $q_\bullet = S$;
  - $R \in F$ if $R \rightarrow \epsilon$ or $R = Z$;

Grammars for regular languages

- 'If': see previous Proposition.
- 'Only if': Build an automaton from the grammar:
  - $Q = N + \{Z\}$, $q_\bullet = S$;
  - $R \in F$ if $R \longrightarrow \epsilon$ or $R = Z$;
  - $R \xrightarrow{x} R'$ if $R \longrightarrow xR'$,
    $R \xrightarrow{x} Z$ if $R \longrightarrow x$.

Grammars for regular languages

An accepting path, labelled with $x_1 \cdots x_n$, through the automaton is either $q_\bullet \xrightarrow{x_1} R_1 \xrightarrow{x_2} \cdots \xrightarrow{x_n} R_n \in F$ or else is $q_\bullet \xrightarrow{x_1} R_1 \xrightarrow{x_2} \cdots \xrightarrow{x_{n-1}} R_{n-1} \xrightarrow{x_n} Z \in F$ and these correspond in the grammar to either $S \Rightarrow x_1 R_1 \Rightarrow x_1 x_2 R_2 \Rightarrow \cdots \Rightarrow x_1 \cdots x_n R_n \Rightarrow x_1 \cdots x_n$ or else $S \Rightarrow x_1 R_1 \Rightarrow x_1 x_2 R_2 \Rightarrow \cdots \Rightarrow x_1 \cdots x_{n-1} R_{n-1} \Rightarrow x_1 \cdots x_n$