ENCODING/COMPRESSION

MISSISSIPPI^RIVER

Mississippi^River

- Character Frequency

M	1
I	5
S	4
P	2
^	1
R	1
V	1
E	1

Frequency Node

- A Node interface into a binary tree. The node has two features:

1. Symbol: Alphanumeric character
2. Frequency: Frequency of the Symbol in the sample documents.

```
class Node
{
public:
	virtual float freq() = 0;
	virtual string symbol() = 0;
};
```

Branch Node

- A specialized Node with two sub-nodes:

Left:

A child node for containing a symbol less than the parent node.

Right:

A child node for containing a symbol greater than the parent node.

```
class Branch : public Node
        Branch(Node*, Node*);
        float frequency();
        string symbol();
        Node* left()  { return _Left; }
        Node* right()  { return _Right; }
        Node* _Left;
        Node* _Right;
```

Leaf Node

- A leaf is a specialization of a Node. It has two attributes:
- Symbol: The alphanumeric symbol
- Frequency: The frequency of the Symbol

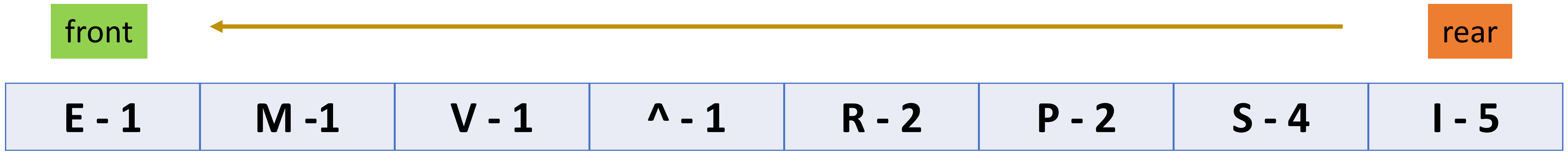
```
class Leaf : public Node
        Leaf(string&, float f);
        float frequency() { return _freq; }
        string symbol() { return _symbol; }
        float _freq;
        string _symbol;
```

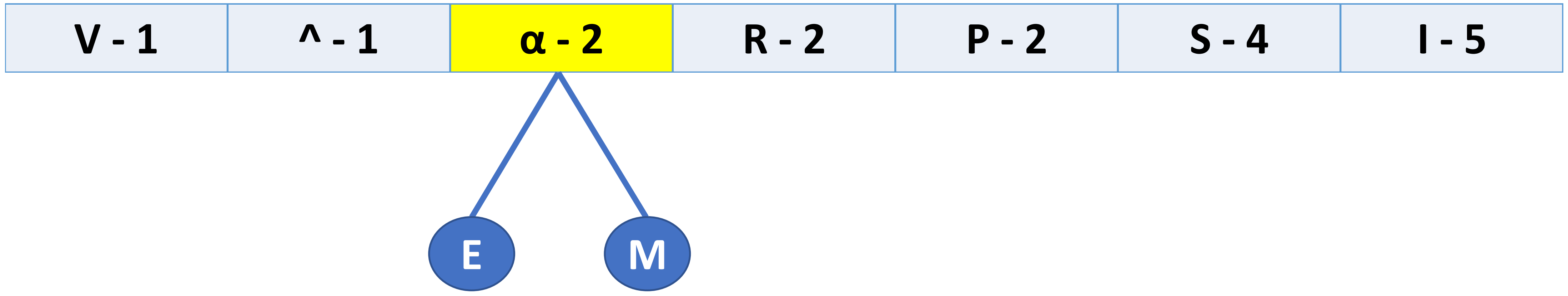
Input: Frequency Queue of Character frequencies

```
Algorithm PriorityQueueTree( FQ )
    while FQ.size() > 1 do
        QLeft = FQ.front()
        FQ.pop()
        QRight = FQ.front()
        FQ.pop()
        node = new Branch(QLeft,QRight)
        Insert node into FQ
    return FQ
```

PriorityQueue

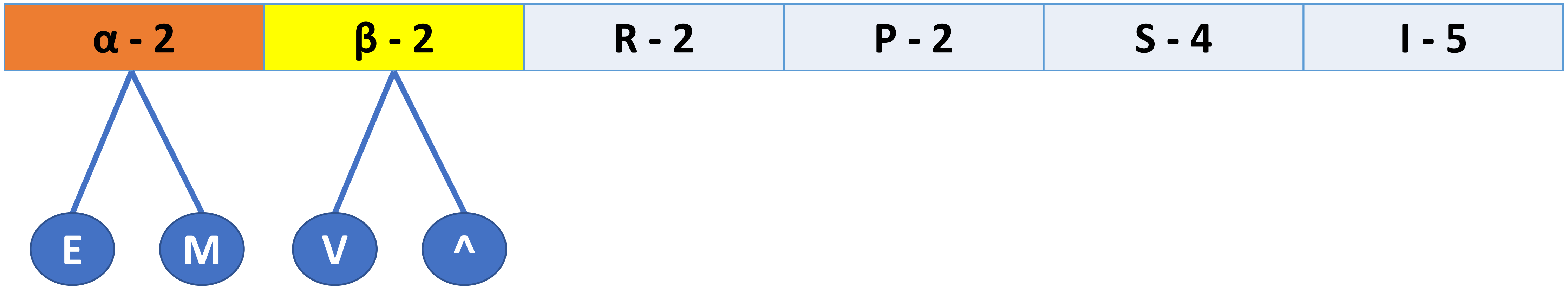
```
priority_queue<Node*,int,
        [](Lhs,Rhs) { return lhs->frequency < rhs->frequency; } > // lambda
```





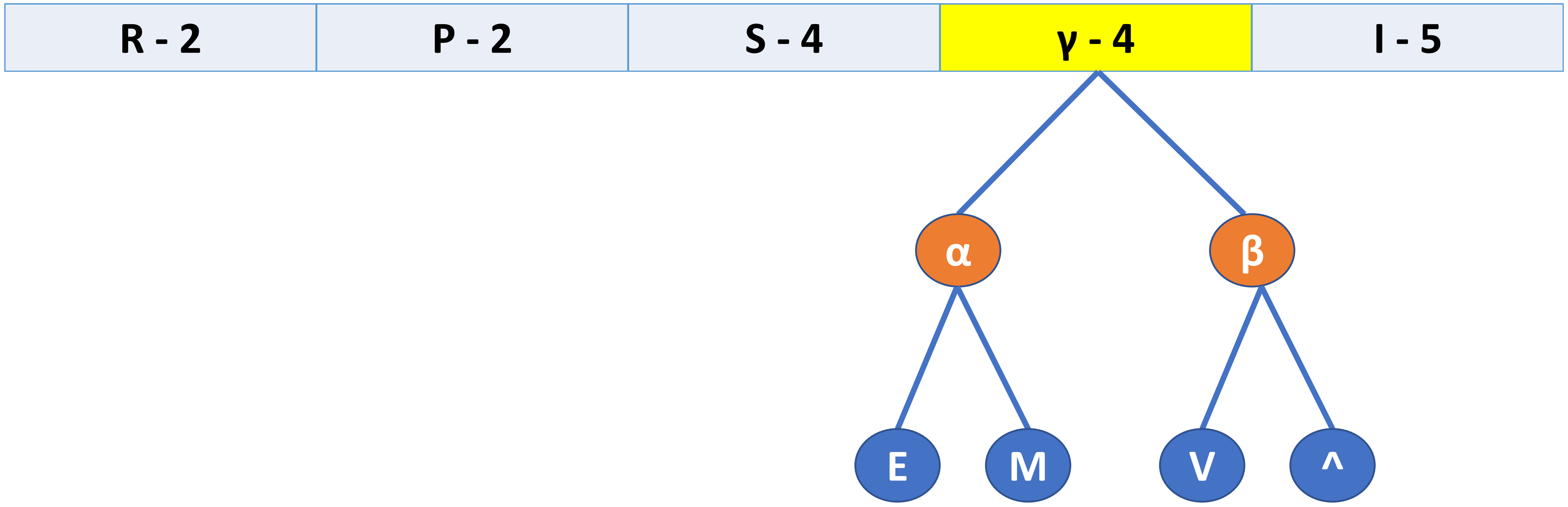
🟠 Branch

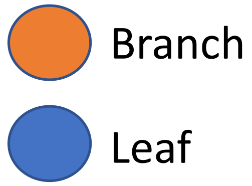
🔵 Leaf

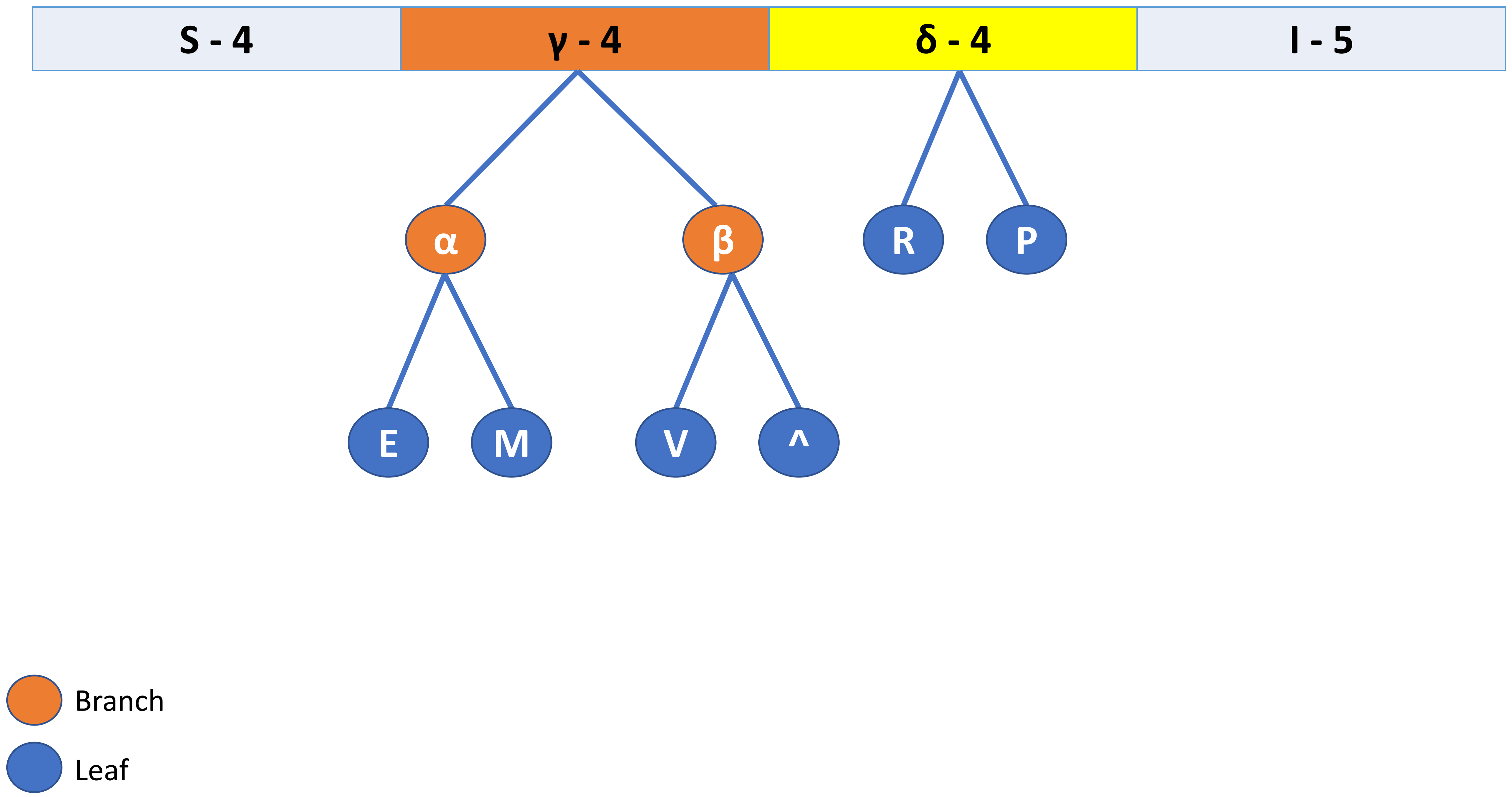


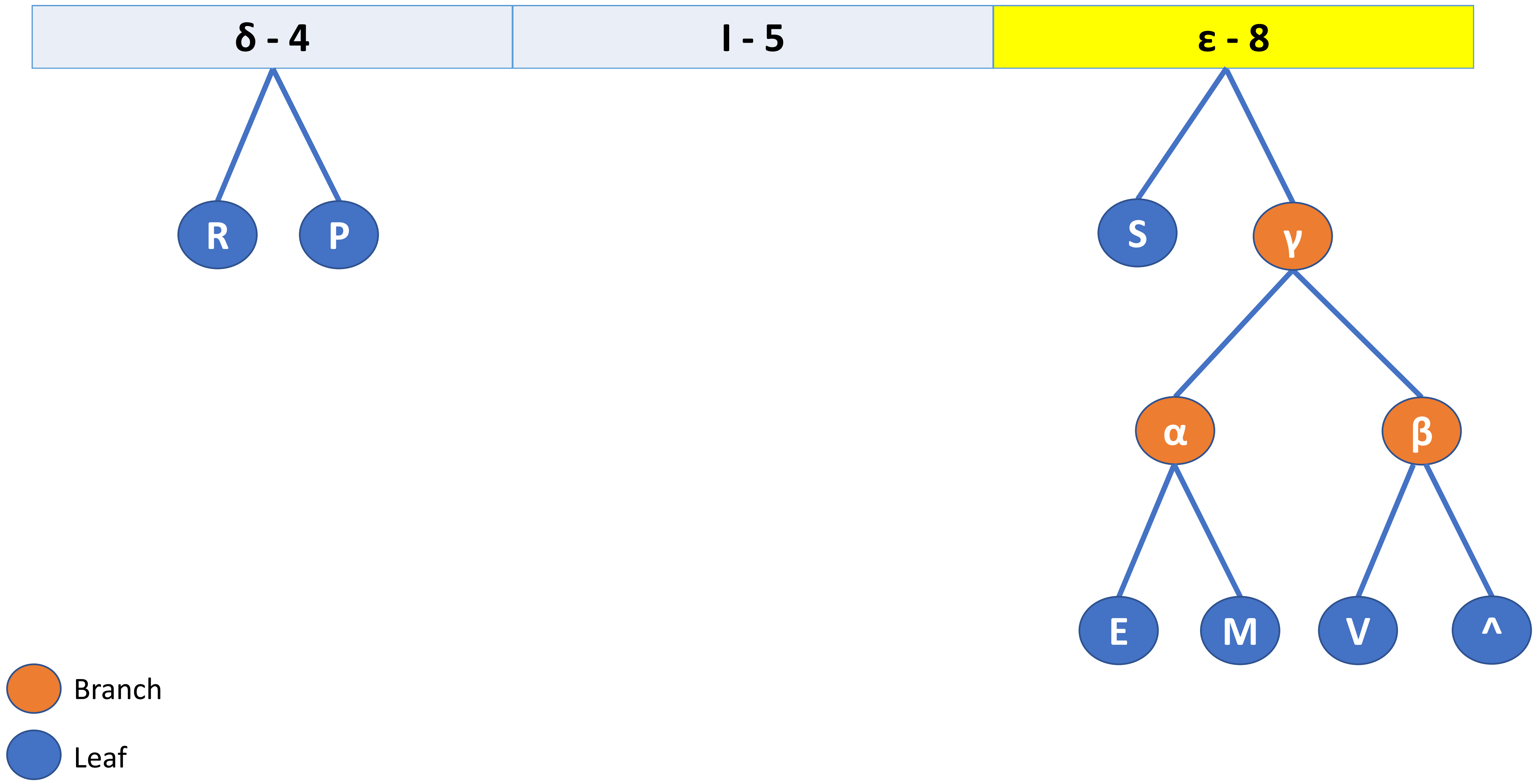
🟠 Branch

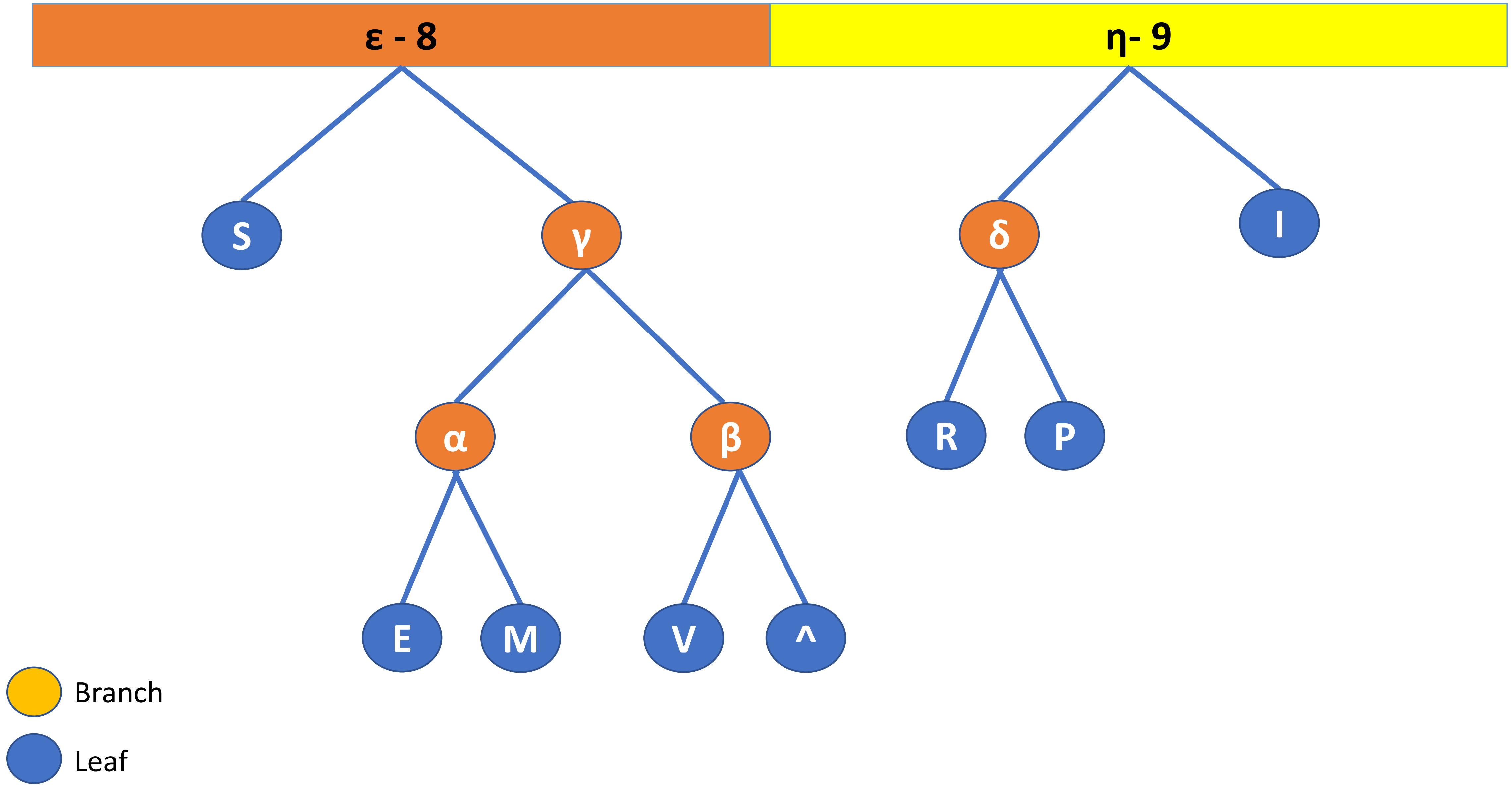
🔵 Leaf


R - 2
P - 2
S - 4
γ - 4
I - 5
α
β
E
M
V
^

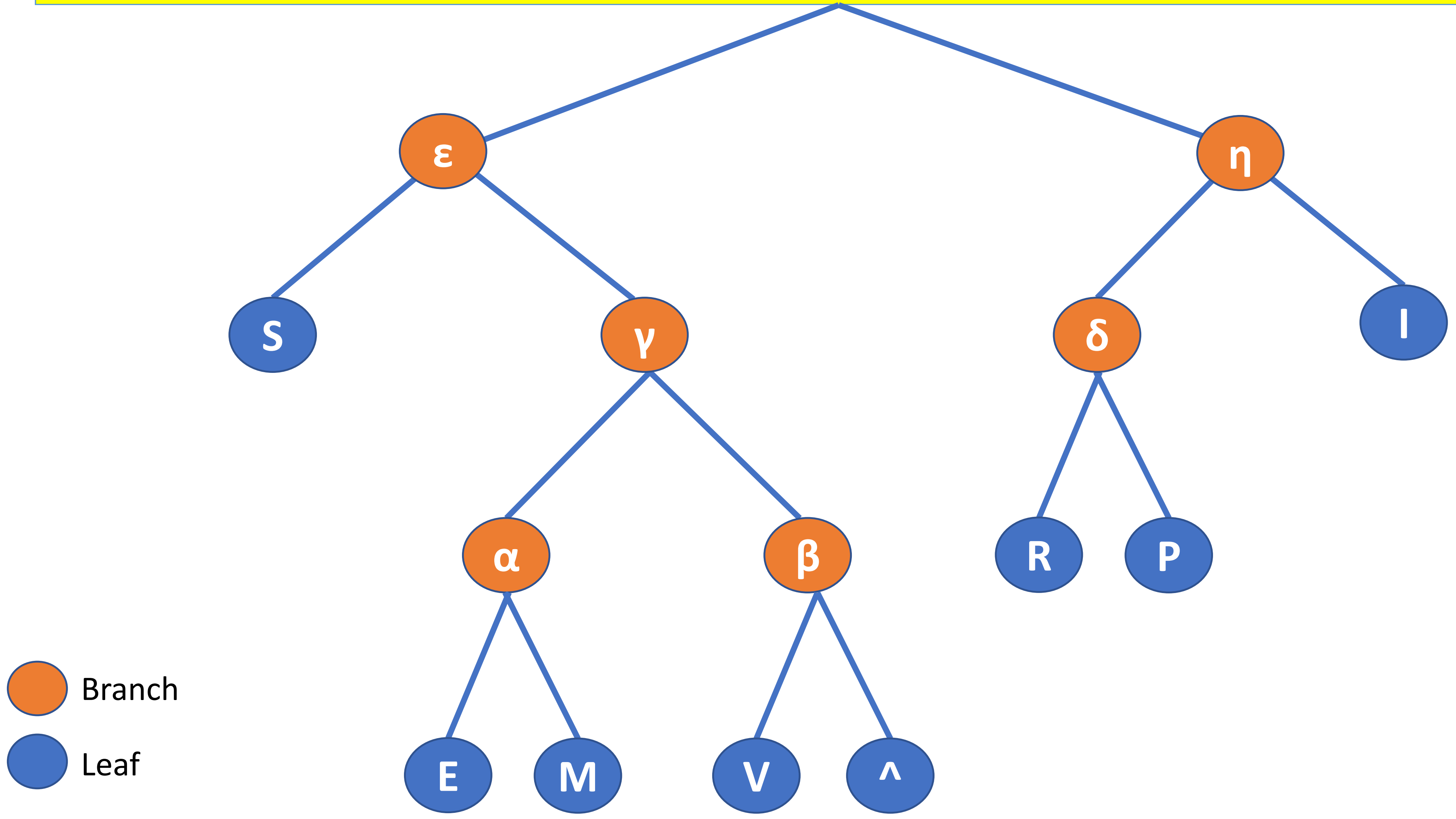

Branch
Leaf

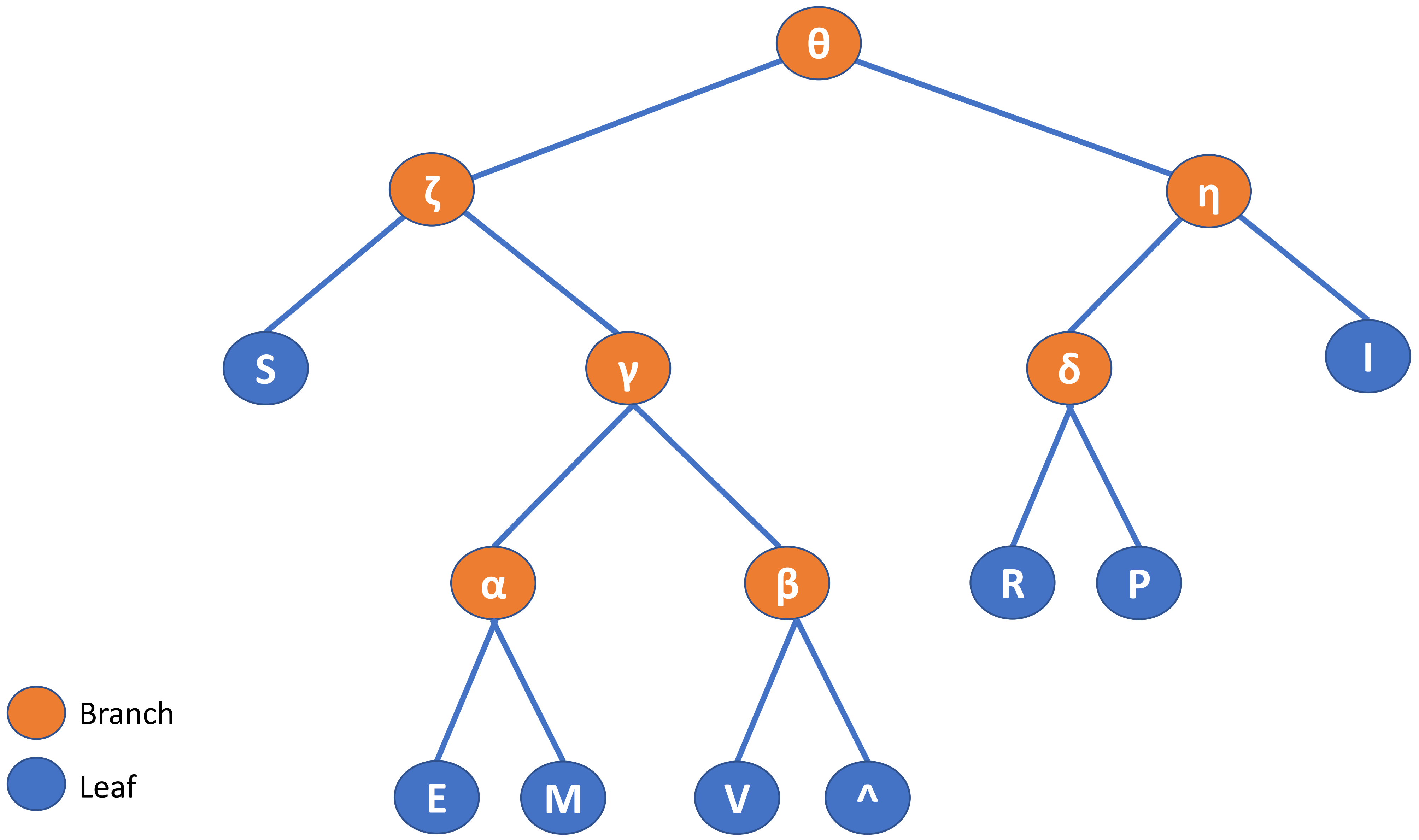

S - 4
γ - 4
δ - 4
I - 5
α
β
R
P
E
M
V
^
Branch
Leaf

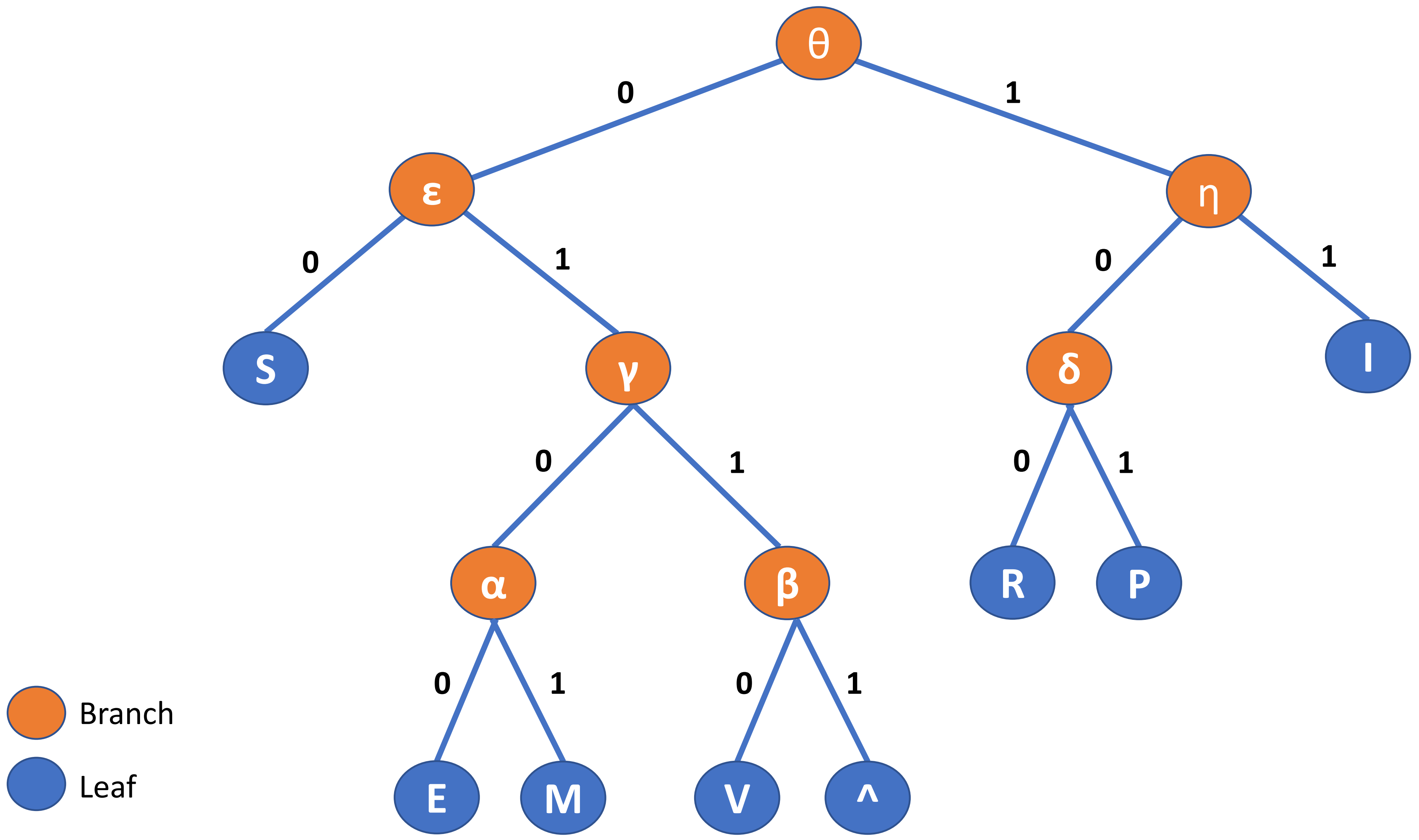

δ - 4
I - 5
ε - 8
R
P
S
γ
α
β
E
M
V
^
Branch
Leaf


ε - 8
η- 9
S
γ
δ
I
α
β
R
P
E
M
V
^
Branch
Leaf

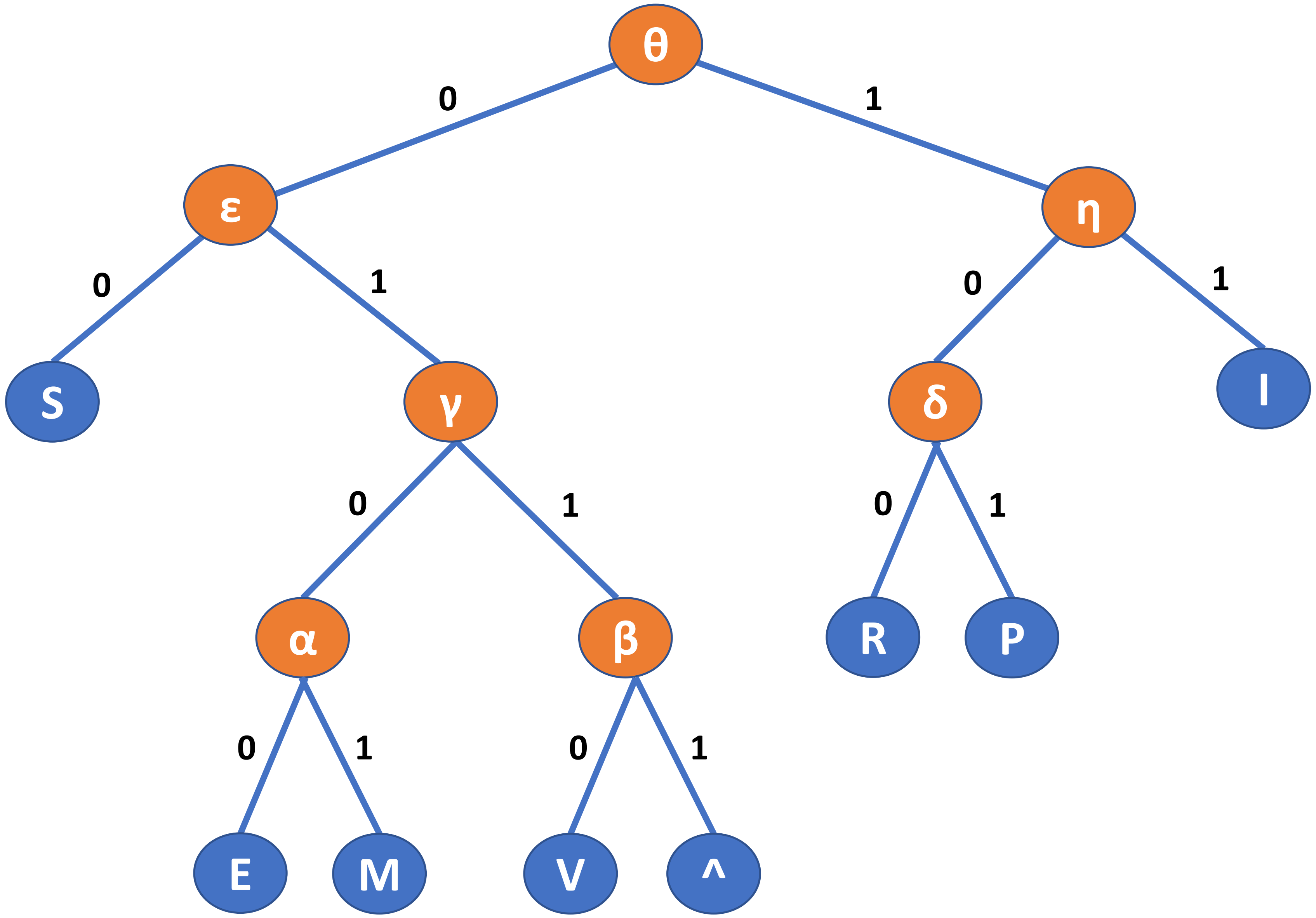
θ- 17




θ
ζ
η
S
γ
δ
I
α
β
R
P
E
M
V
^
Branch
Leaf


θ
0
1
ε
η
0
1
0
1
S
γ
δ
I
0
1
0
1
α
β
R
P
0
1
0
1
E
M
V
^
Branch
Leaf

E	0100
I	11
M	0101
P	101
R	100
S	00
V	0110
^	0111



Input: Single Node Tree, String S

Output: Encoded Pair for each symbol

```
Algorithm Encode( T, S )
    for each Node in Tree:
        If Node is a Branch:
            Encode Node.Left, S += "0"
            Encode Node.Right, S += "1"
        If Node is a Leaf:
            Node.pair = encoded string + symbol
```

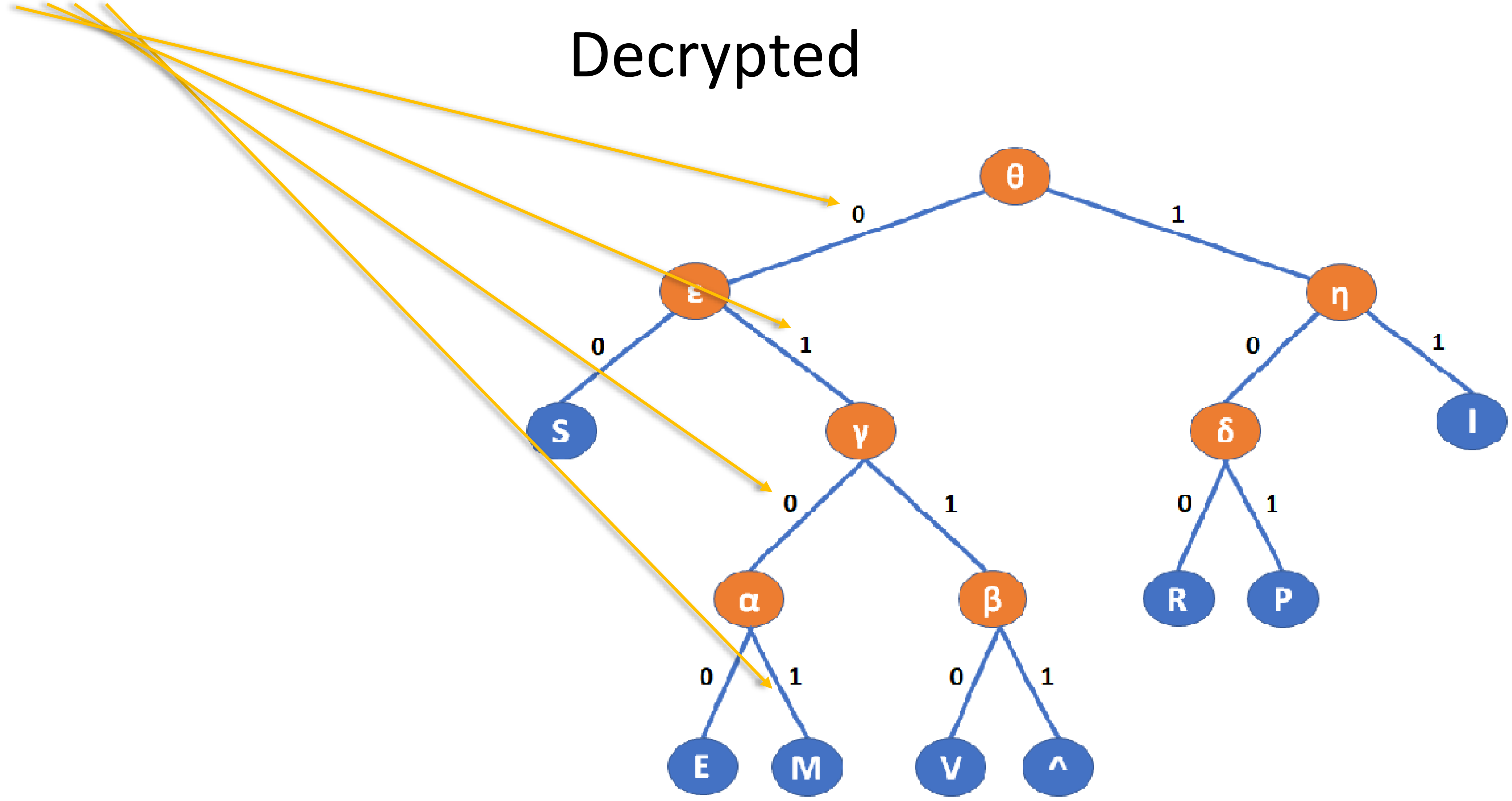
Encrypted

M	I	S	S	I	S	S	I	P	P	I	^	R	I	V	E	R
0101	11	00	00	11	00	00	11	101	101	11	0111	100	11	0110	0100	100

E	0100
I	11
M	0101
P	101
R	100
S	00
V	0110
^	0111

0101110000110000111011011101111001101100100100

Decrypted


θ
0
1
ε
η
0
1
0
1
S
γ
δ
I
0
1
0
1
α
β
R
P
0
1
0
1
E
M
V
^