Qlue-ls, a powerful *SPARQL* language server

brining modern langague tooling to SPARQL

Ioannis Nezis

University Freiburg

2025-10-08

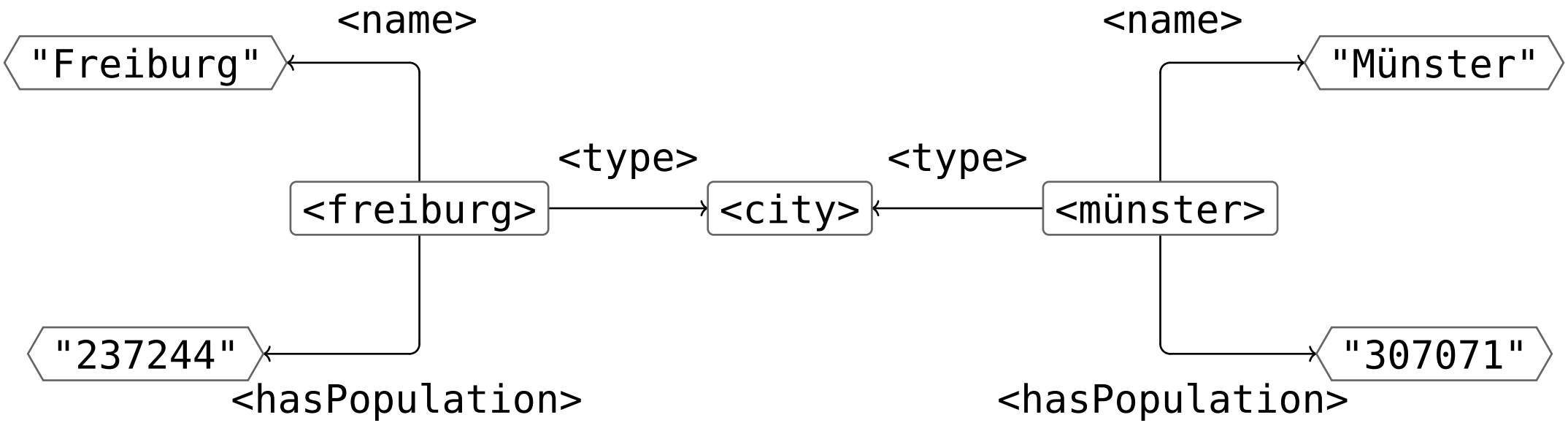
Outline

1. Background 2
   1.1 RDF 3
   1.2 SPARQL 4
2. The Problem 8
   2.1 Language Support for SPARQL 9
   2.2 Reusablility 10
3. The Solution 11
   3.1 Language server protocol 12
   3.2 Building a Parse tree 14
   3.3 Implementing each capability 41
4. Evaluation 46

1. Background

1.1 RDF

- **R**esource **D**escription **F**ramework to describe directed graphs



- used in wikidata, UniProt, DBLP, ...
- interoperable by design

1.2 SPARQL

- **S**PARQL **P**rotocol and **R**DF **Q**uery **L**anguage
- standard query language for RDF
- retrieve or change data

1.2 SPARQL

- **S**PARQL **P**rotocol and **R**DF **Q**uery **L**anguage
- standard query language for RDF
- retrieve or change data

Example: List all cities with their names.

```
1  SELECT * WHERE {
2      ?city  <type>  <city>  .
3      ?city  <name>  ?name
4  }
```

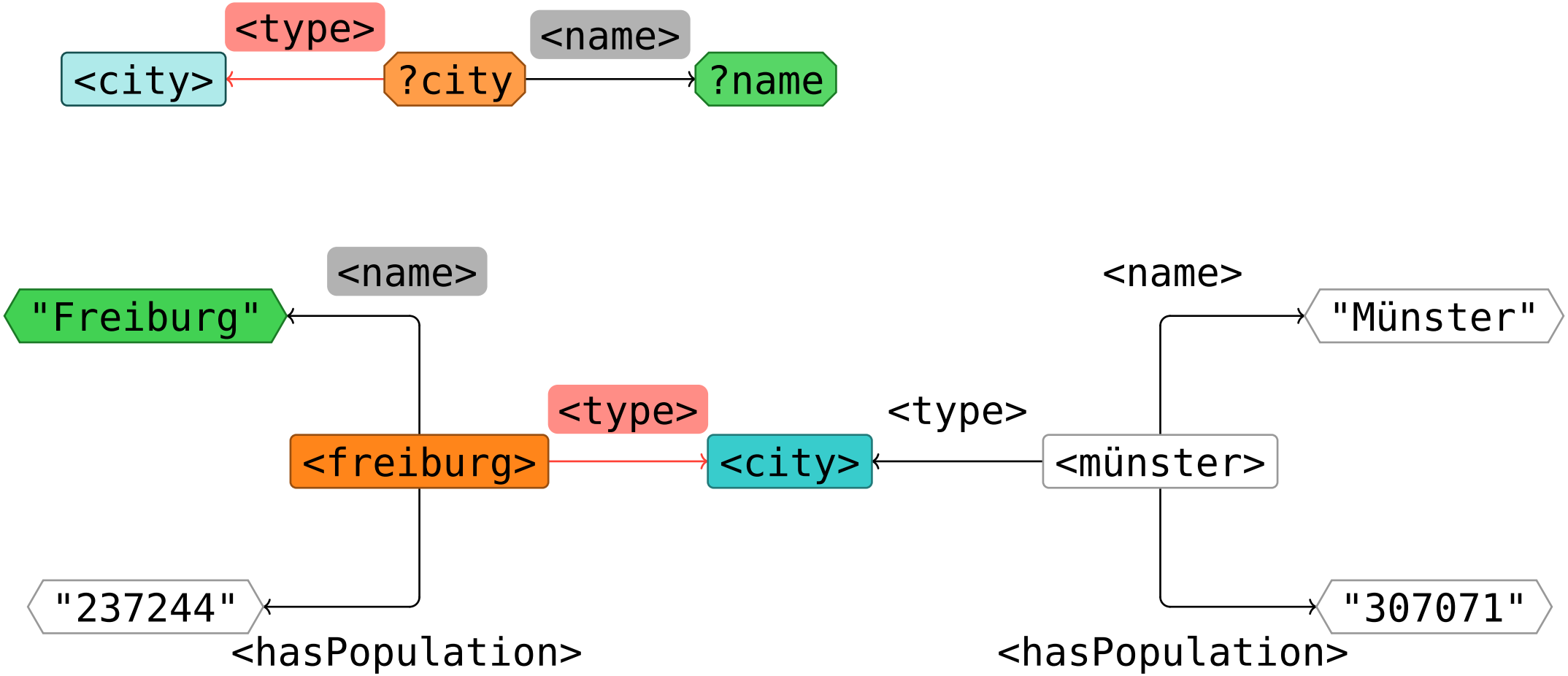
1.2 SPARQL

- **S**PARQL **P**rotocol and **R**DF **Q**uery **L**anguage
- standard query language for RDF
- retrieve or change data

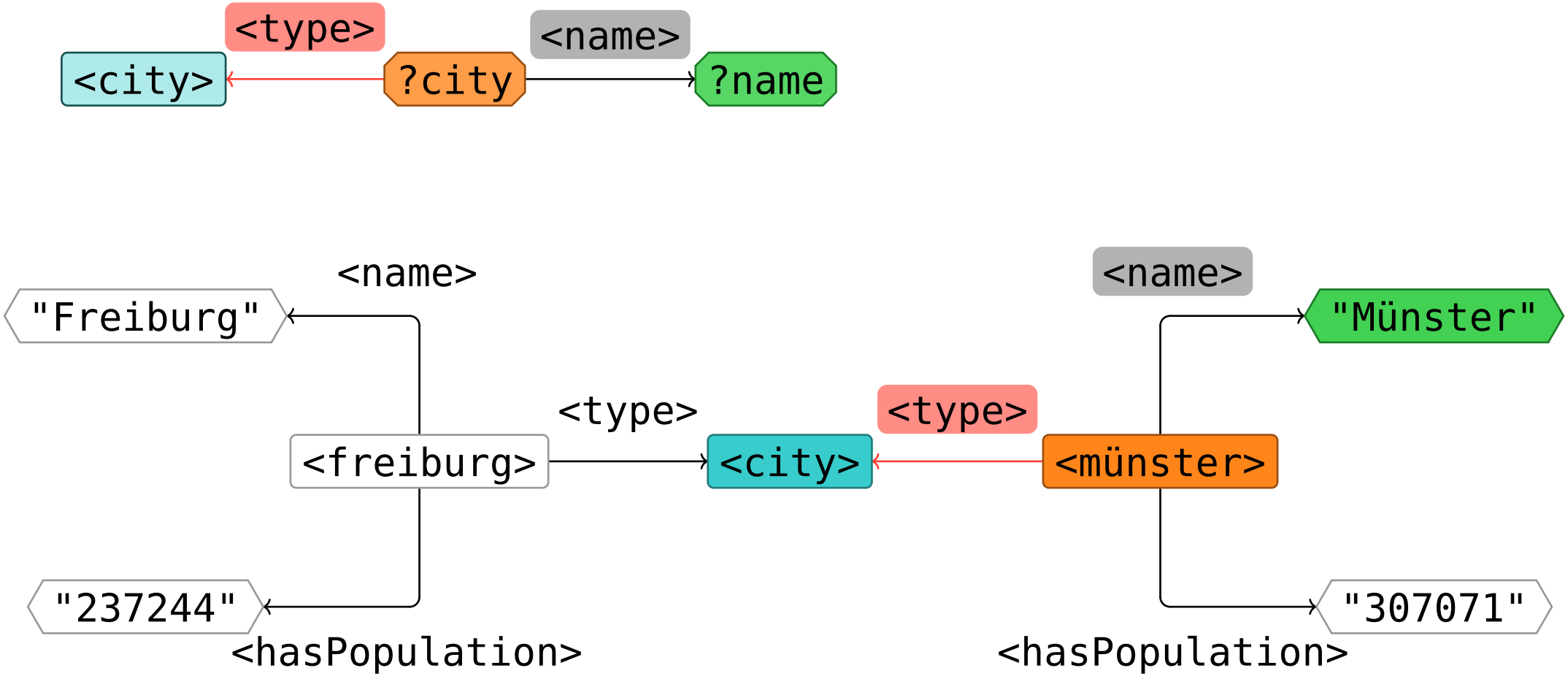
Example: List all cities with their names.

```
1  SELECT * WHERE {
2      ?city  <type>  <city>  .
3      ?city  <name>  ?name
4  }
```

1.2 SPARQL


<type>
<name>
<city>
?city
?name
<name>
"Freiburg"
<type>
<type>
<name>
"Münster"
<freiburg>
<city>
<münster>
"237244"
"307071"
<hasPopulation>
<hasPopulation>

1.2 SPARQL


<type>
<name>
<city>
?city
?name
<name>
<name>
"Freiburg"
"Münster"
<type>
<type>
<freiburg>
<city>
<münster>
"237244"
"307071"
<hasPopulation>
<hasPopulation>

1.2 SPARQL

Result:

```
1  SELECT * WHERE {
2      ?city  <type>  <city>  .
3      ?city  <name>  ?name
4  }
```

?city	?name
<freiburg>	“Freiburg”
<münster>	“Münster”

2. The Problem

2.1 Language Support for SPARQL

Language support is a vague term.

I define language support for as follows:

2.1 Language Support for SPARQL

Language support is a vague term.
I define language support for as follows:

completion

The **completion** capability suggests completions at a given position in the input.

2.1 Language Support for SPARQL

Language support is a vague term.

I define language support for as follows:

- completion
- formatting

The **formatting** capability transforms the input into a standard form.

2.1 Language Support for SPARQL

Language support is a vague term.
I define language support for as follows:

completion	formatting	diagnostics

Given a *SPARQL* query, the **diagnostics** capability reports issues in the query, each tied to a specific range in the document. Each issue has one of four severity levels: Error, Warning, Info, or Hint.

2.1 Language Support for SPARQL

Language support is a vague term.
I define language support for as follows:

completion	formatting	diagnostics	code actions

The **code action** capability suggests edits to improve or modify the query. These edits can be simple or complex. Code actions typically automate common tasks — for example, adding a language filter.

2.1 Language Support for SPARQL

Language support is a vague term.

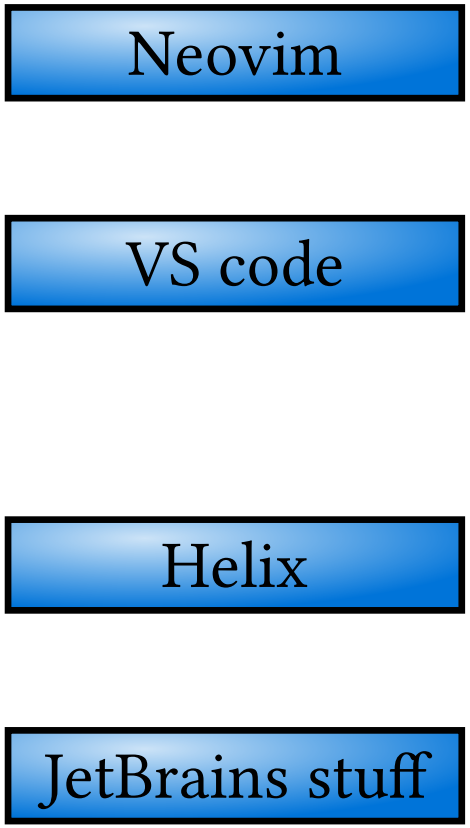
I define language support for as follows:

completion	formatting	diagnostics	code actions	hover

Given a *SPARQL* query and a cursor position, the **hover** capability shows textual information about the element under the cursor.

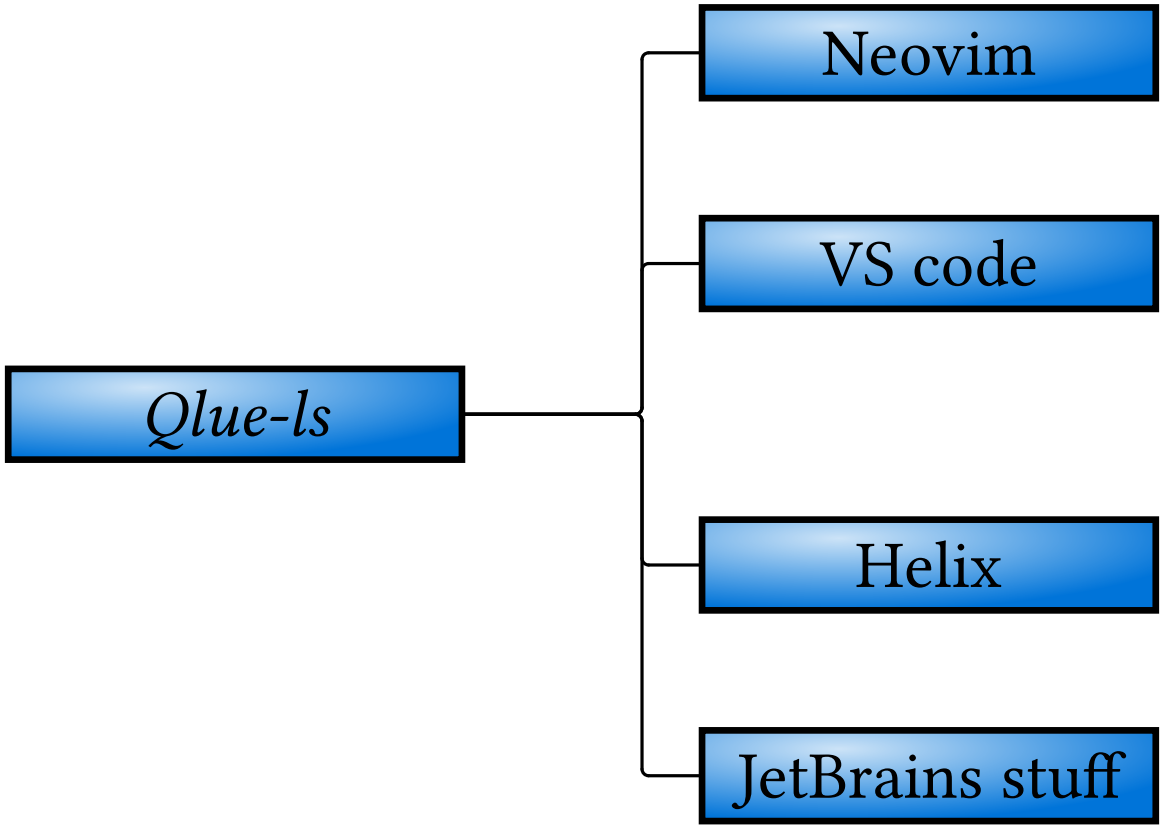
2.2 Reusablility

- I want a reusable componet
- Able to plug into many editors: Neovim, VSCode, Helix, ...



2.2 Reusablility

- I want a reusable componet
- Able to plug into many editors: Neovim, VSCode, Helix, ...



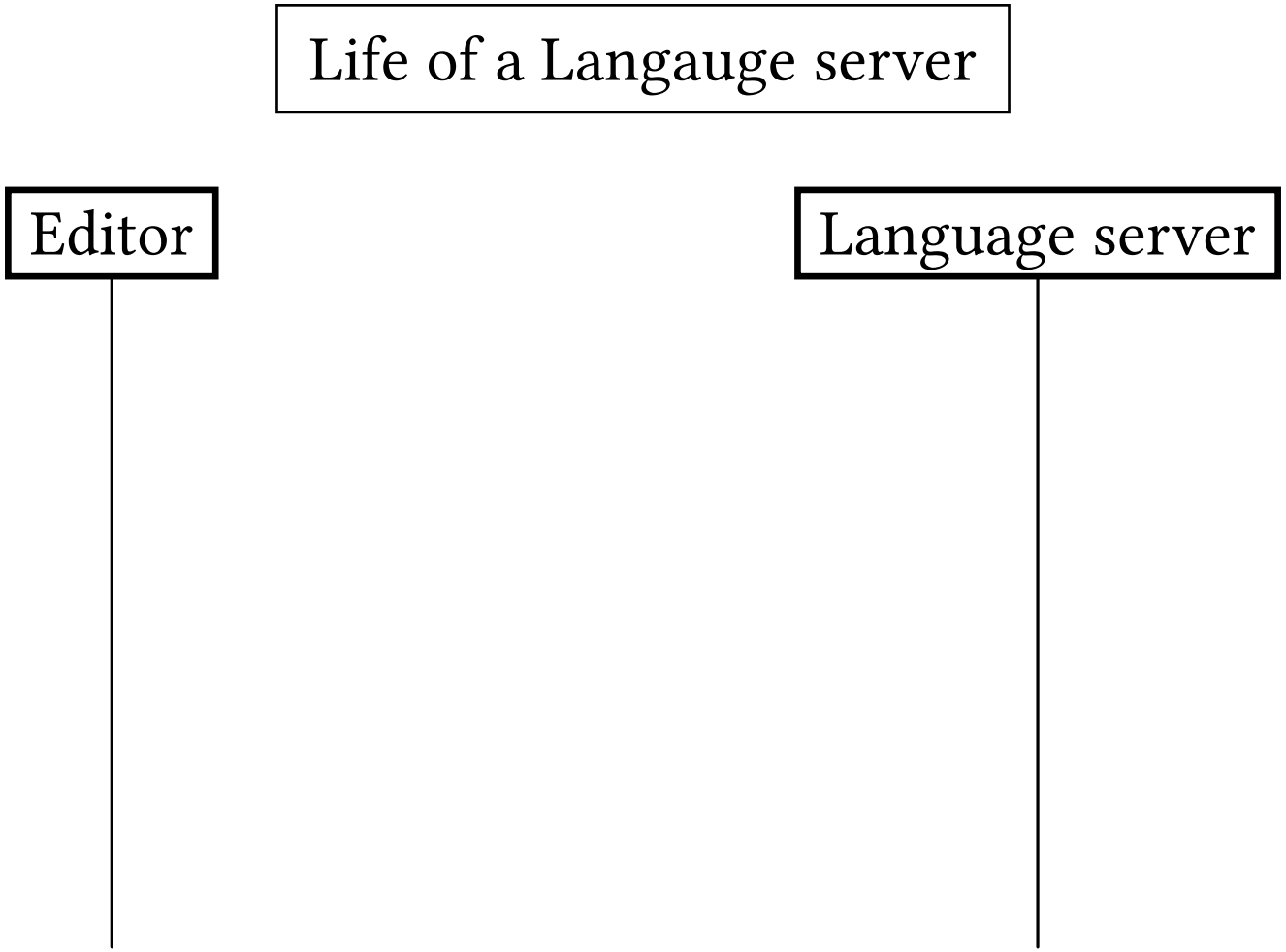
3. The Solution

3.1 Language server protocol

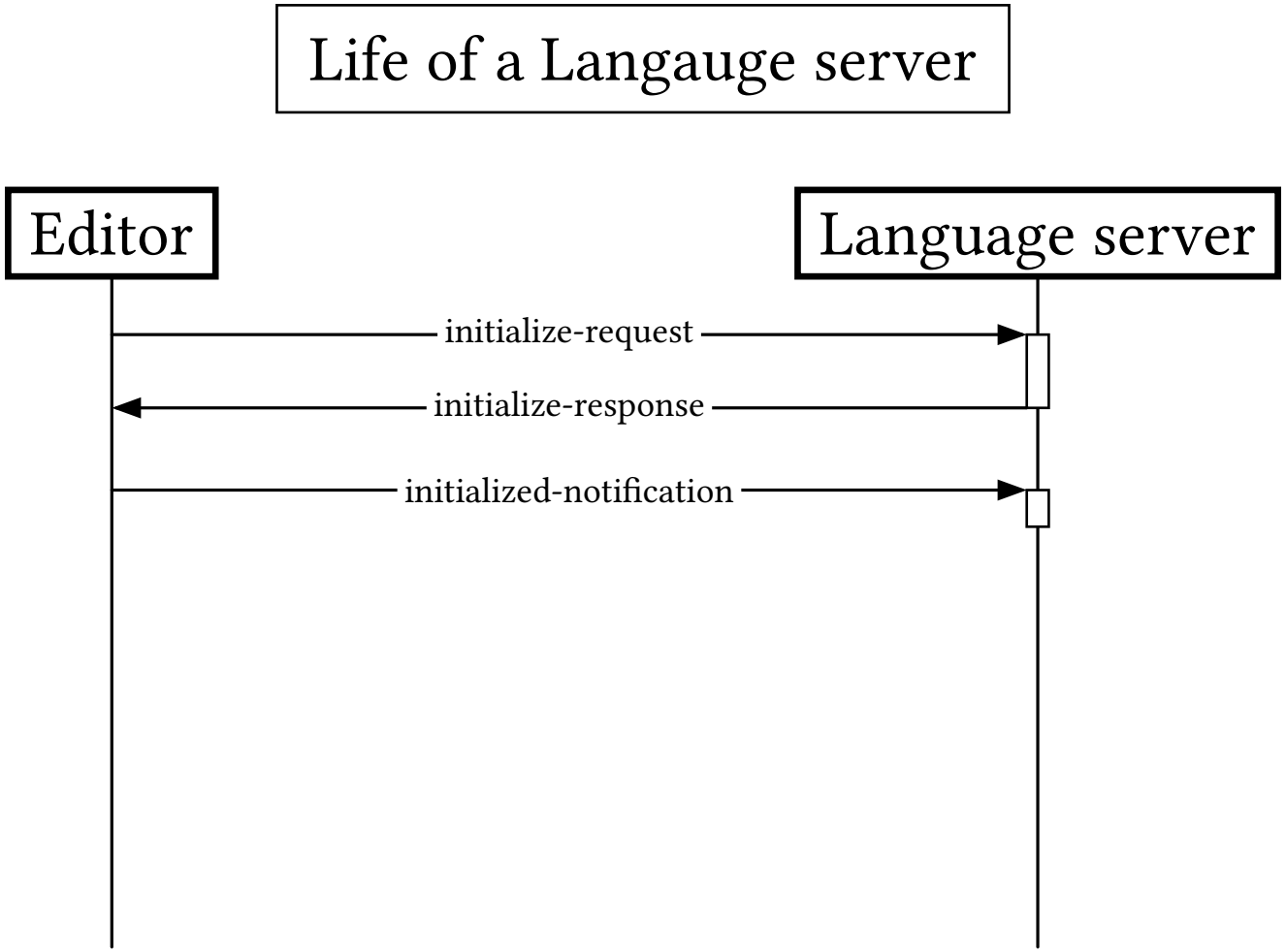
The **Language server protocol** specifies the communication between a a development tool e.g. a editor and a language server.

- published by Microsoft in 2017
- JSON-RPC based
- inter procress communication

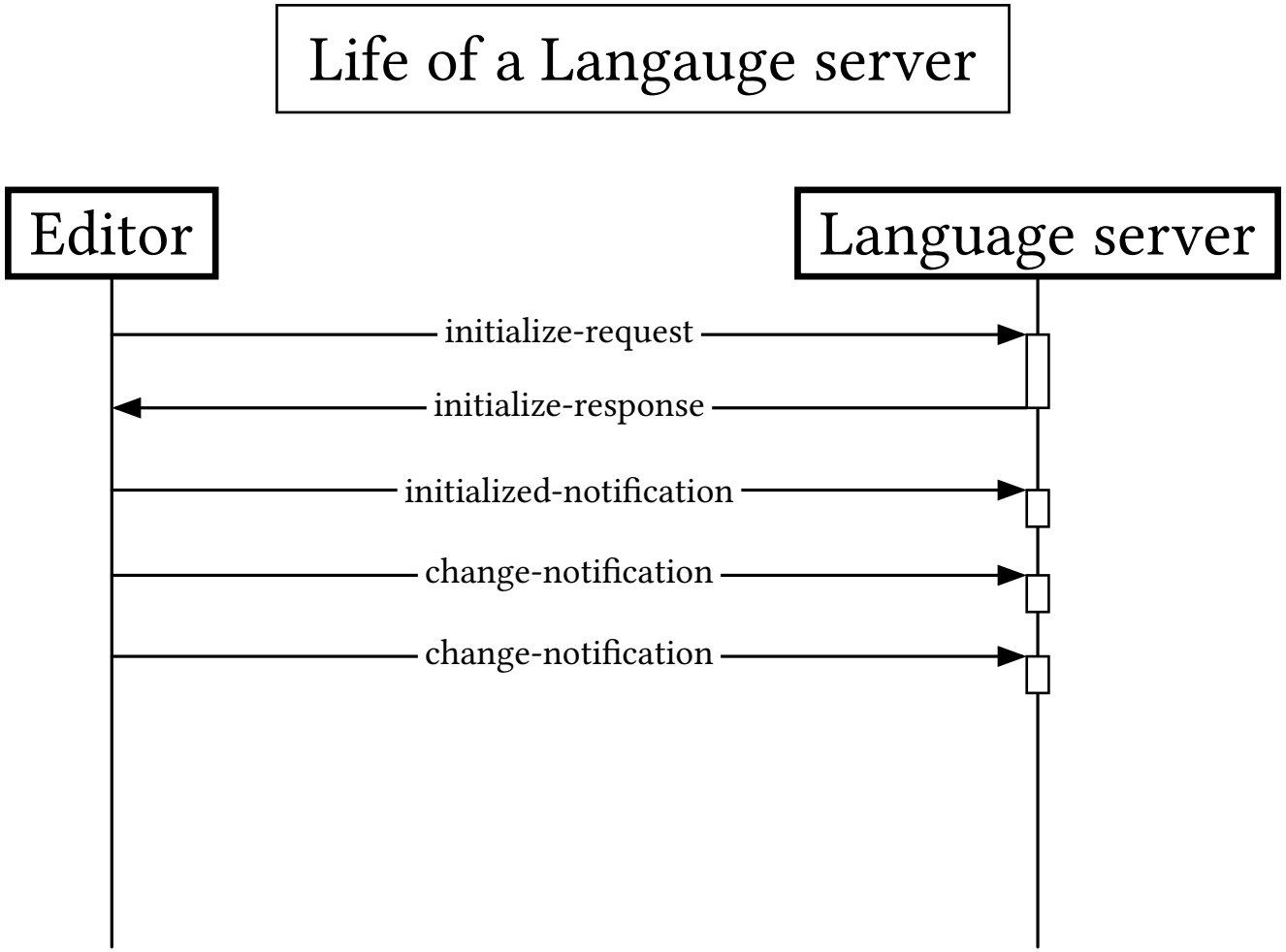
3.1 Language server protocol



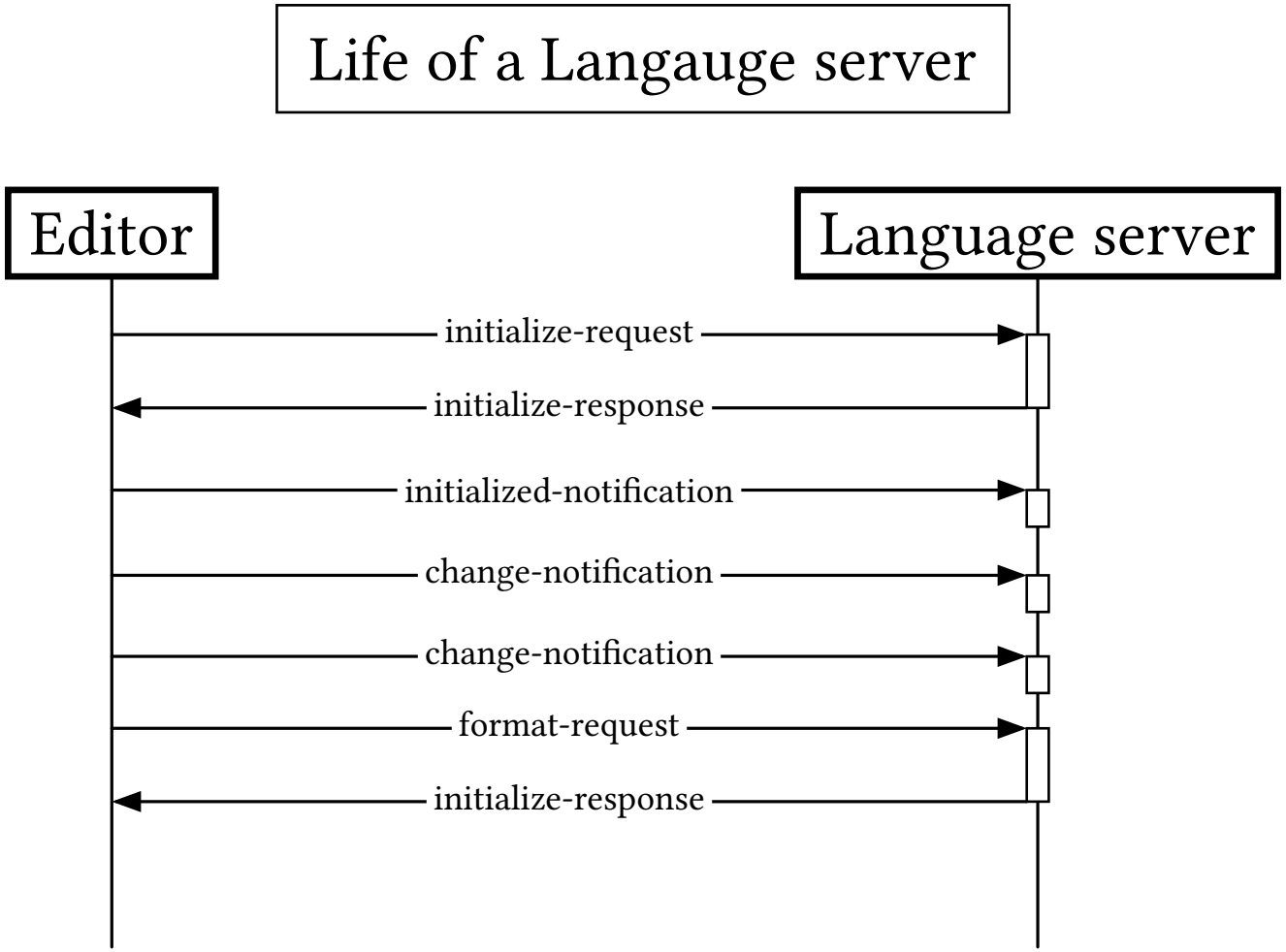
3.1 Language server protocol



3.1 Language server protocol



3.1 Language server protocol

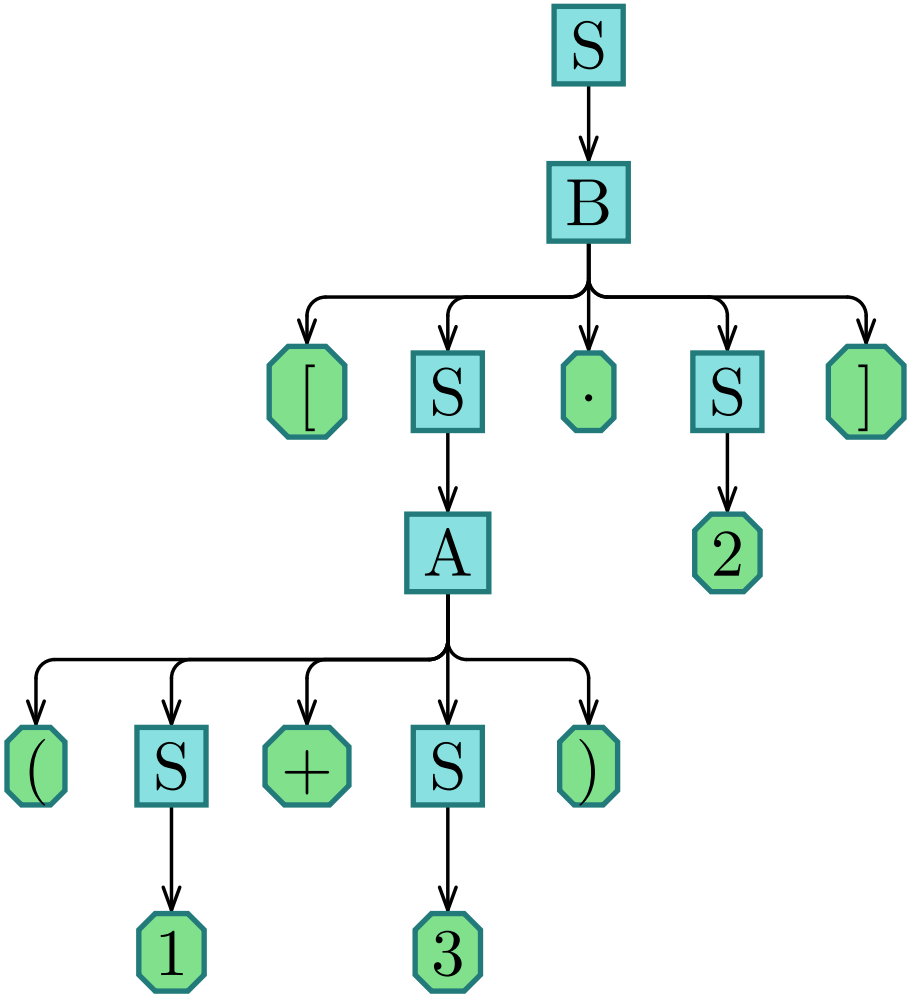


3.2 Building a Parse tree

- internal representation of the syntacitc structure

Example:

$$[(1 + 3) \cdot 2]$$



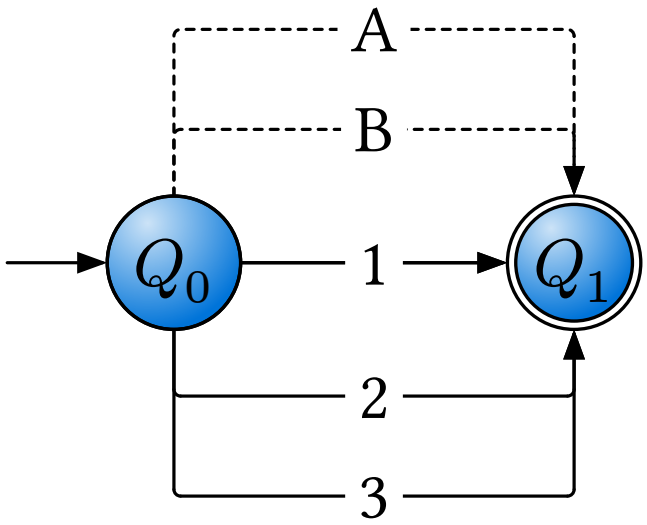
3.2 Building a Parse tree

- context free grammar to describe the syntax
- **recursive decent parser** to build a parse tree
  - parsing procedure for each nonterminal

```
1  S -> A | B | 1 | 2 | 3
2  A -> ( S + S )
3  B -> [ S • S ]
```

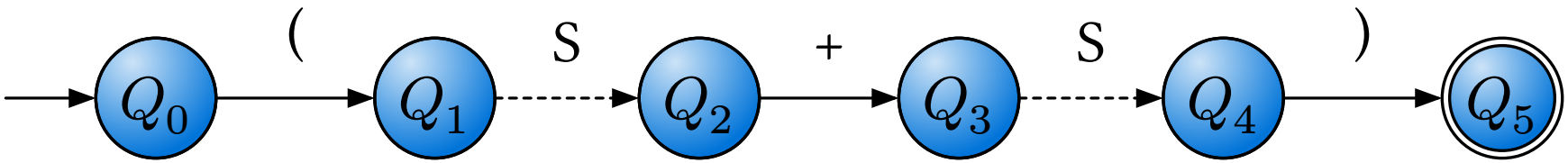
$$\text{FIRST}(S) = \{(, [, 1, 2, 3\}$$
$$\text{FIRST}(A) = \{(\}$$
$$\text{FIRST}(B) = \{[\}$$

3.2 Building a Parse tree



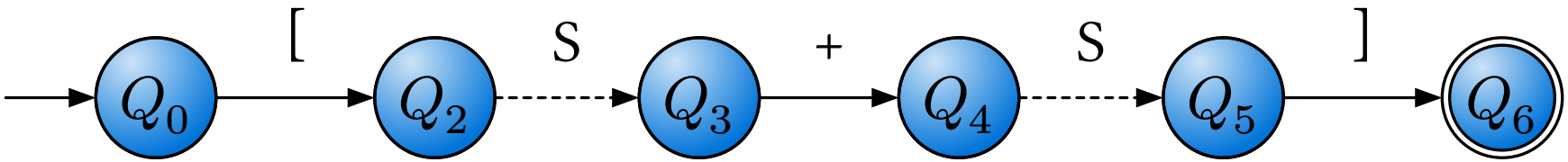

```
S -> A | B | 1 | 2 | 3
```

3.2 Building a Parse tree




```
A -> ( S + S )
```

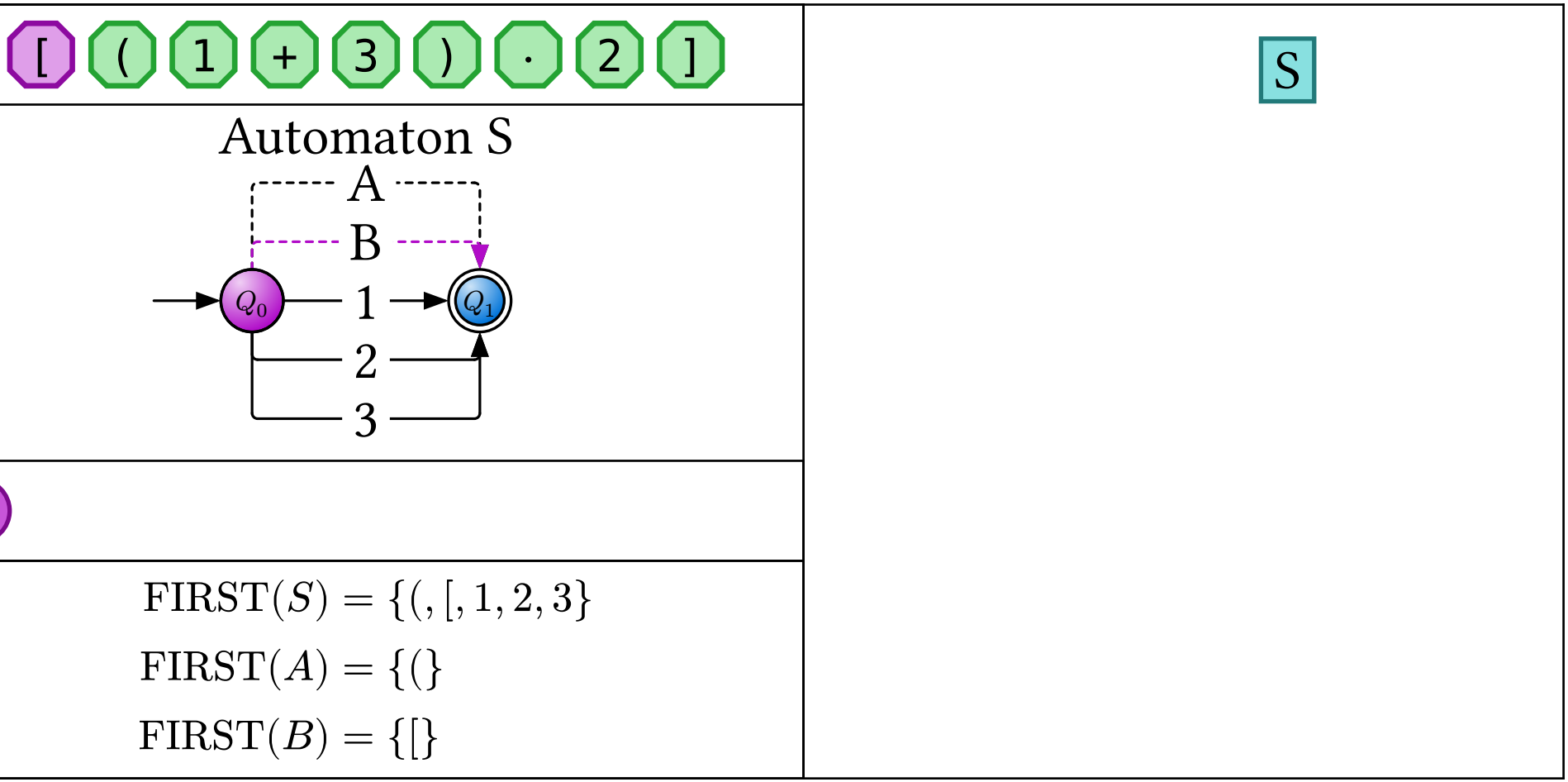
3.2 Building a Parse tree



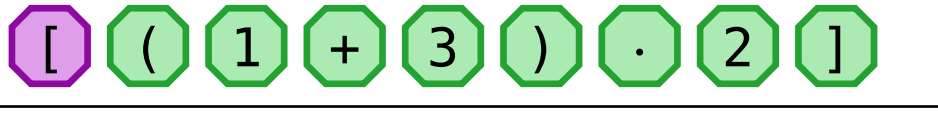

```
B -> [ S + S ]
```

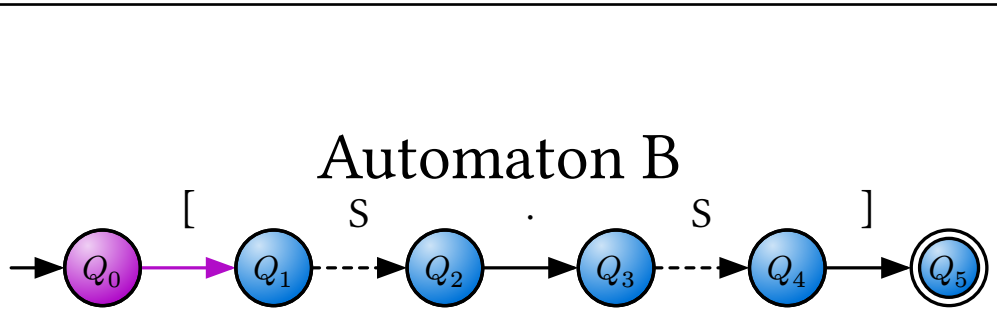
3.2 Building a Parse tree

3.2.1 Example: $[(1 + 3) \cdot 2]$

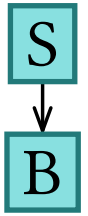


3.2 Building a Parse tree

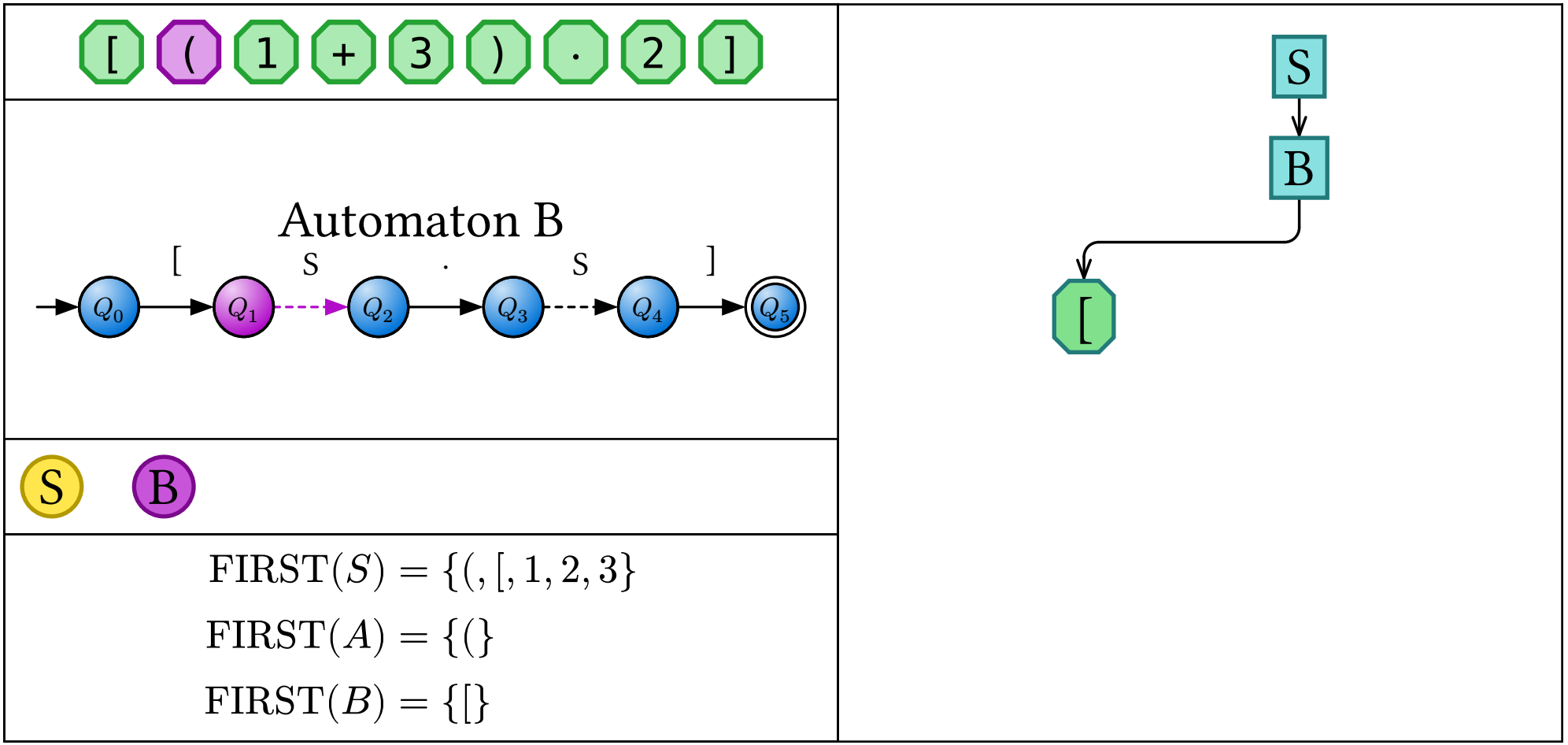




$$\text{FIRST}(S) = \{(, [, 1, 2, 3\}$$
$$\text{FIRST}(A) = \{(\}$$
$$\text{FIRST}(B) = \{[\}$$

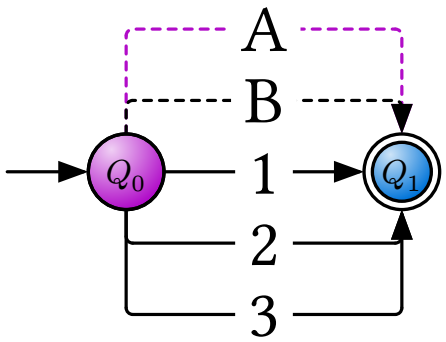


3.2 Building a Parse tree



3.2 Building a Parse tree

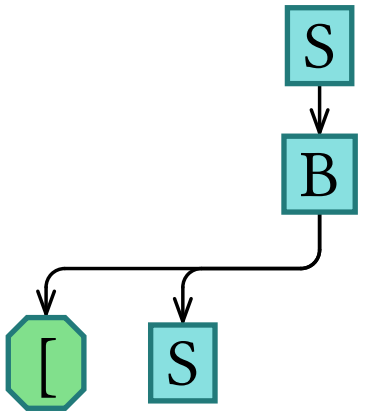
Automaton S



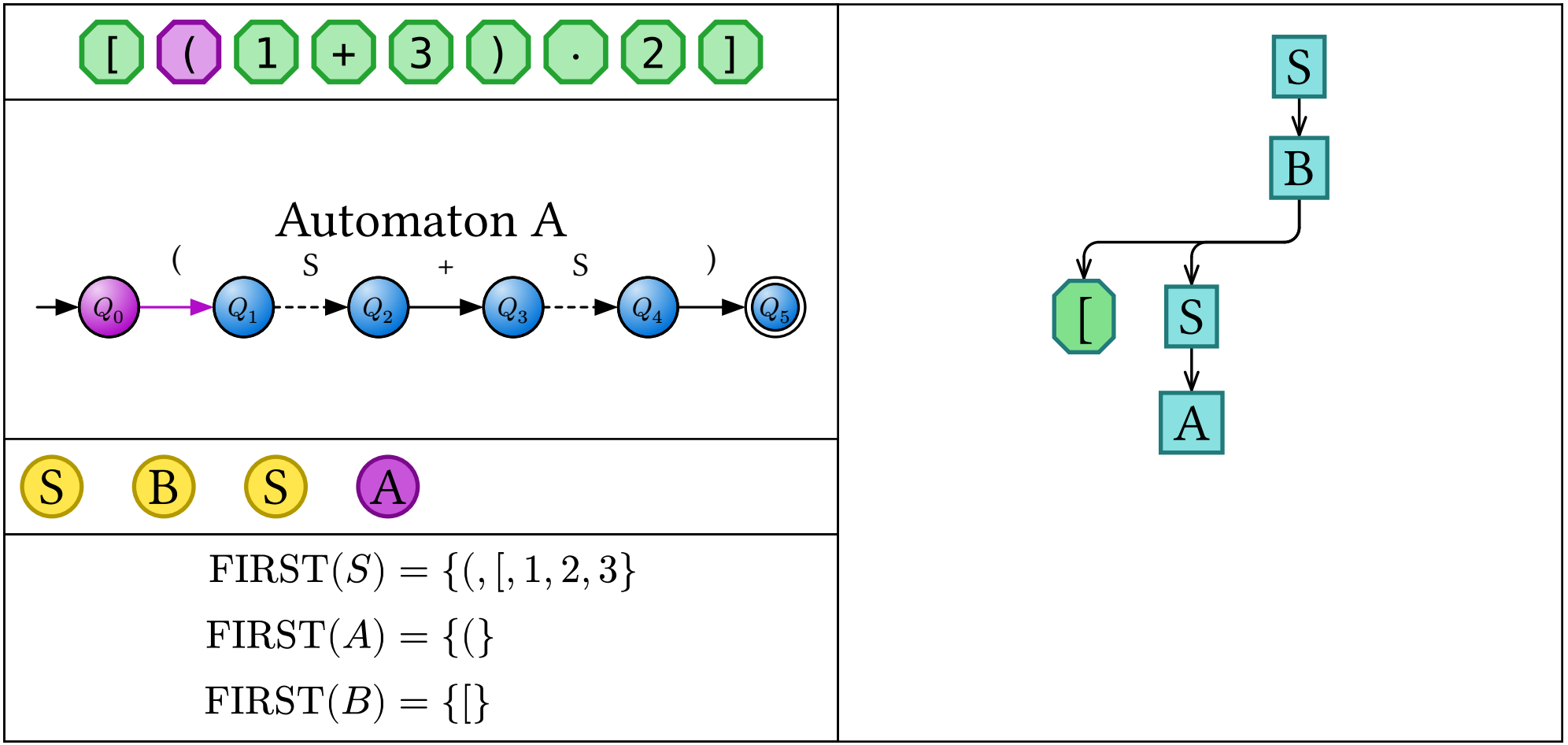
$$\text{FIRST}(S) = \{(, [, 1, 2, 3\}$$

$$\text{FIRST}(A) = \{(\}$$

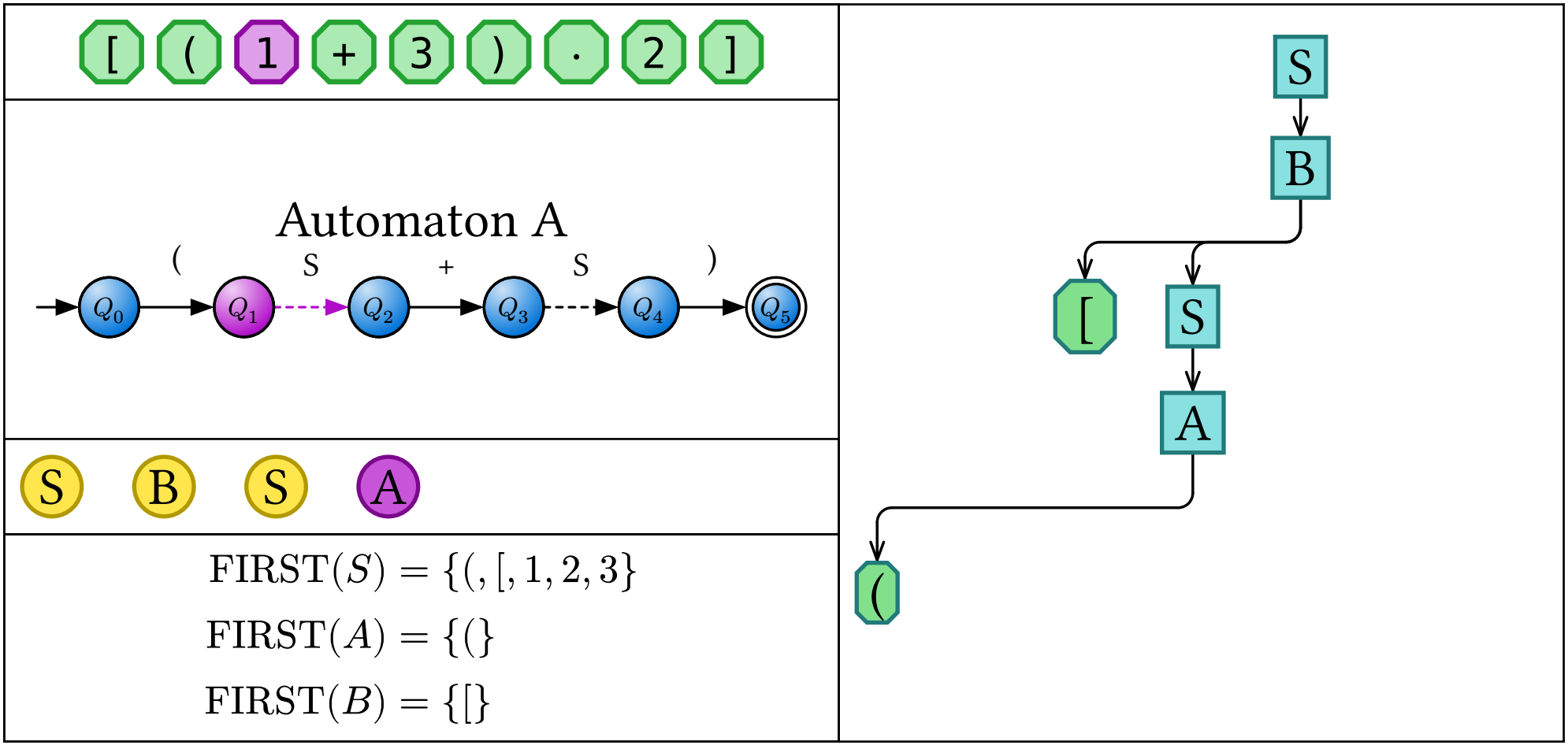
$$\text{FIRST}(B) = \{[\}$$



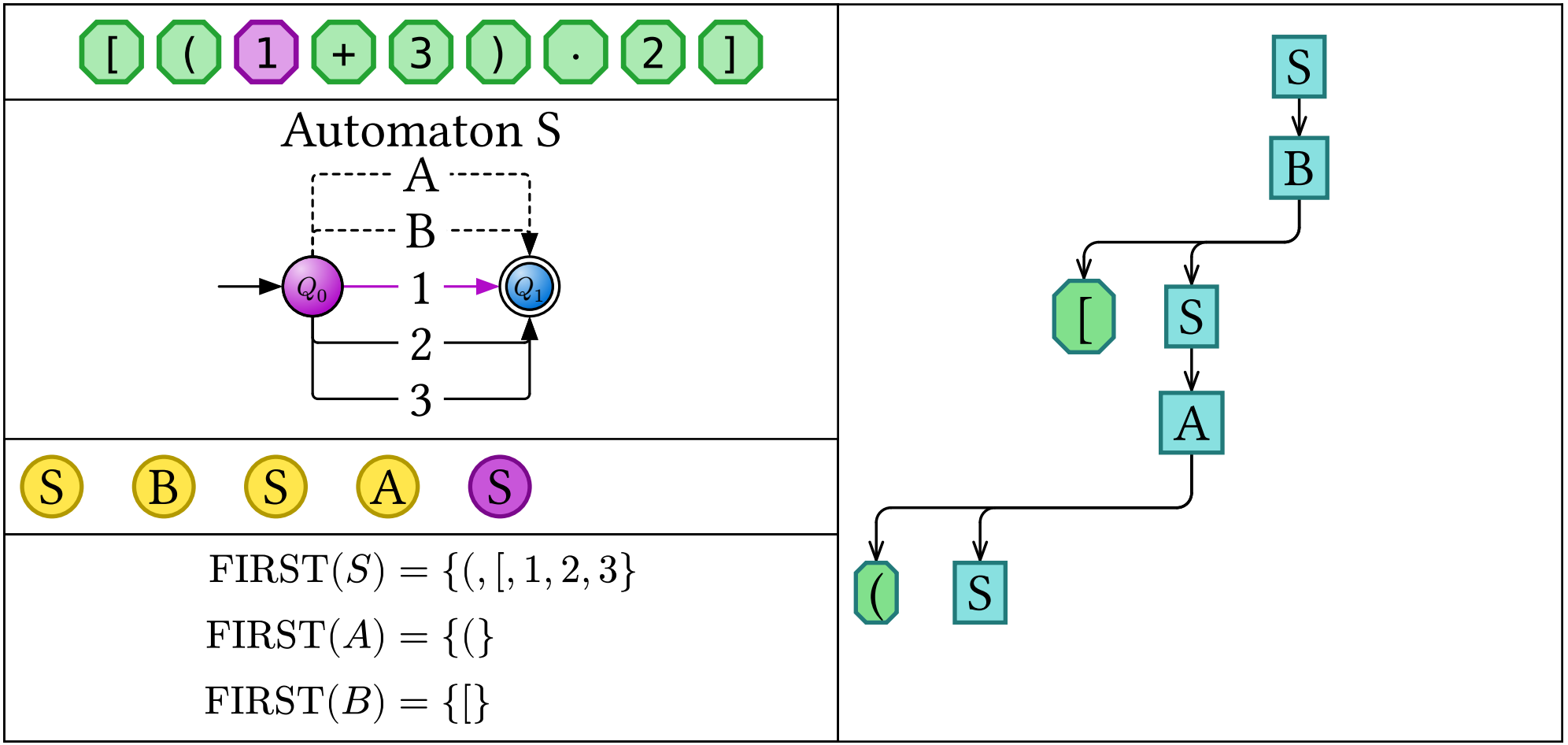
3.2 Building a Parse tree



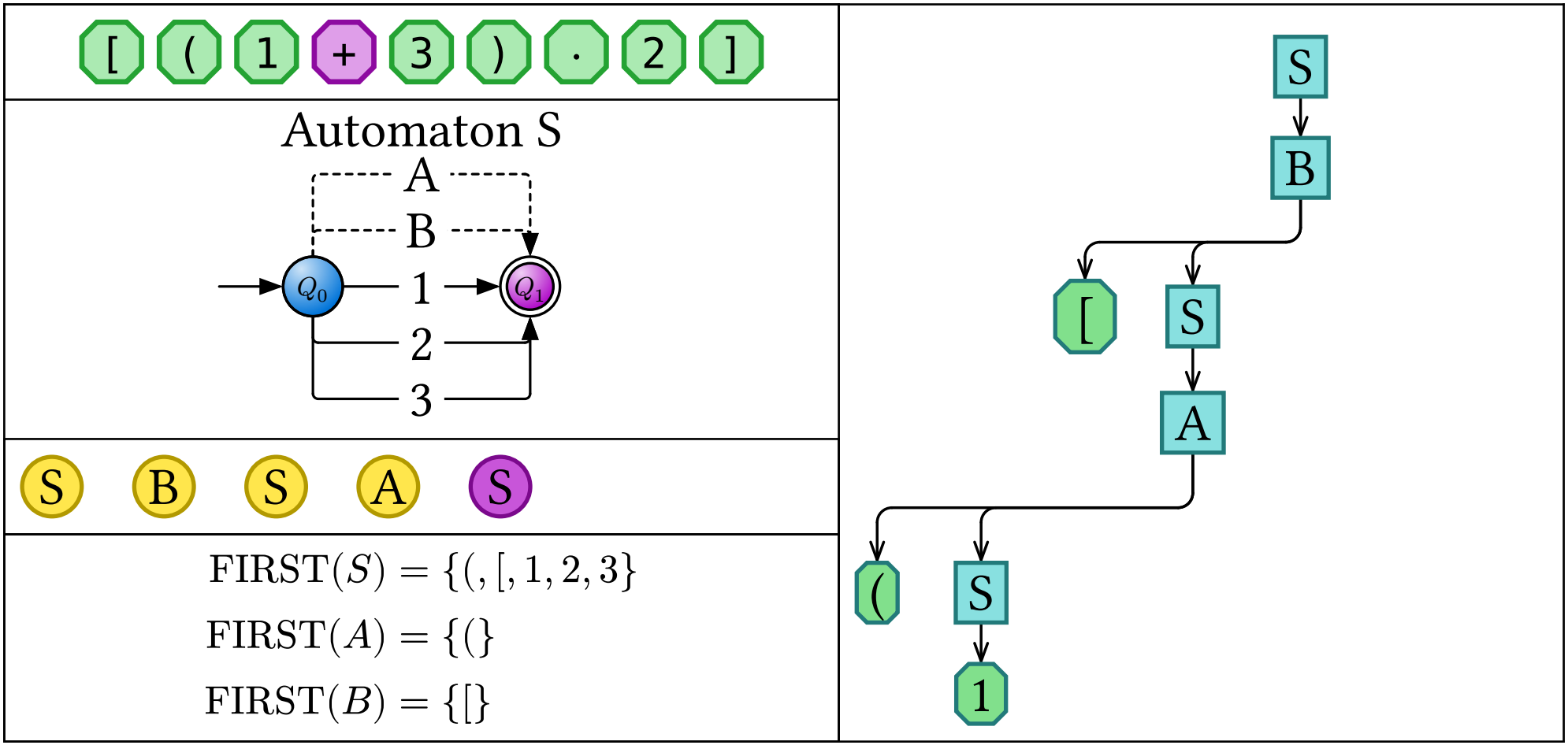
3.2 Building a Parse tree



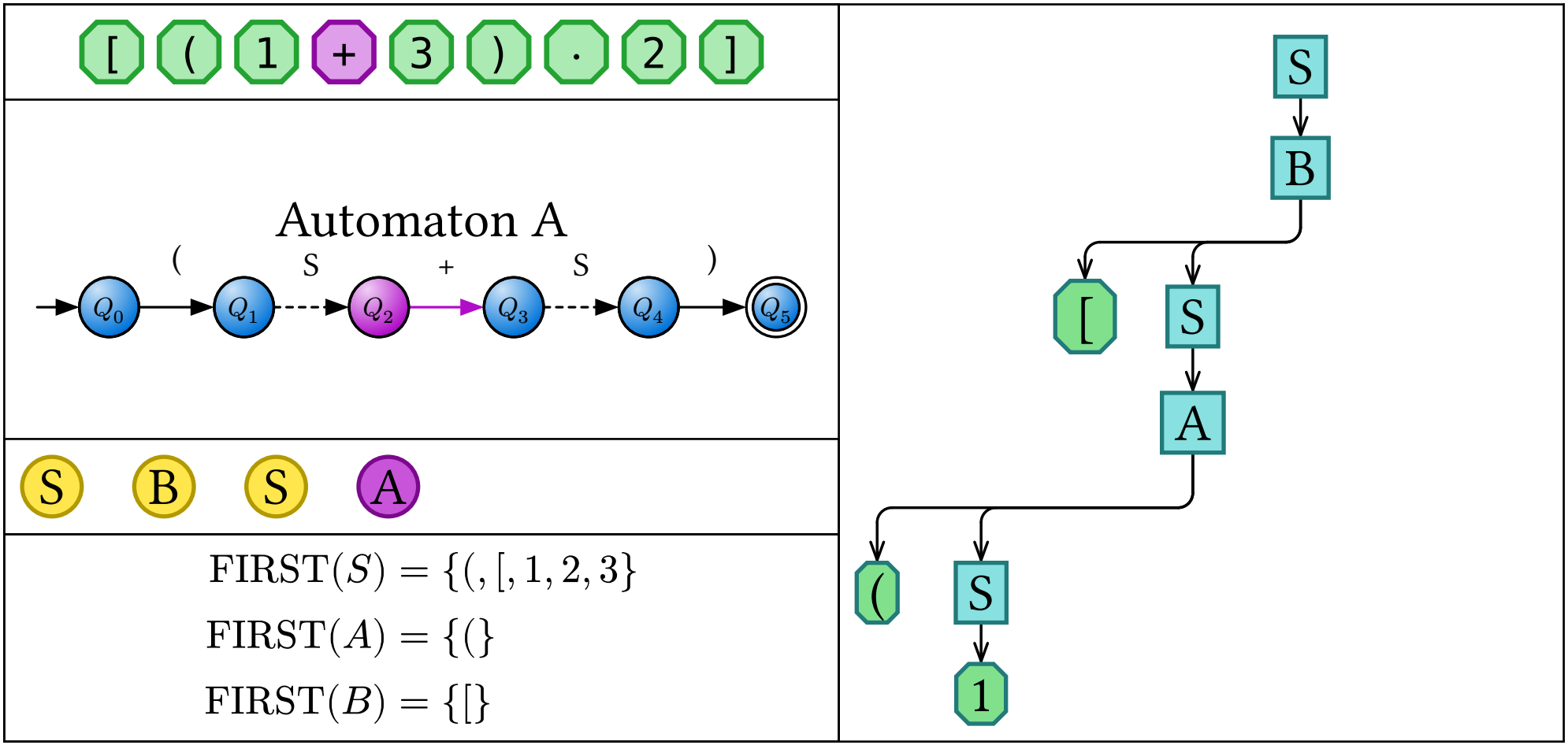
3.2 Building a Parse tree



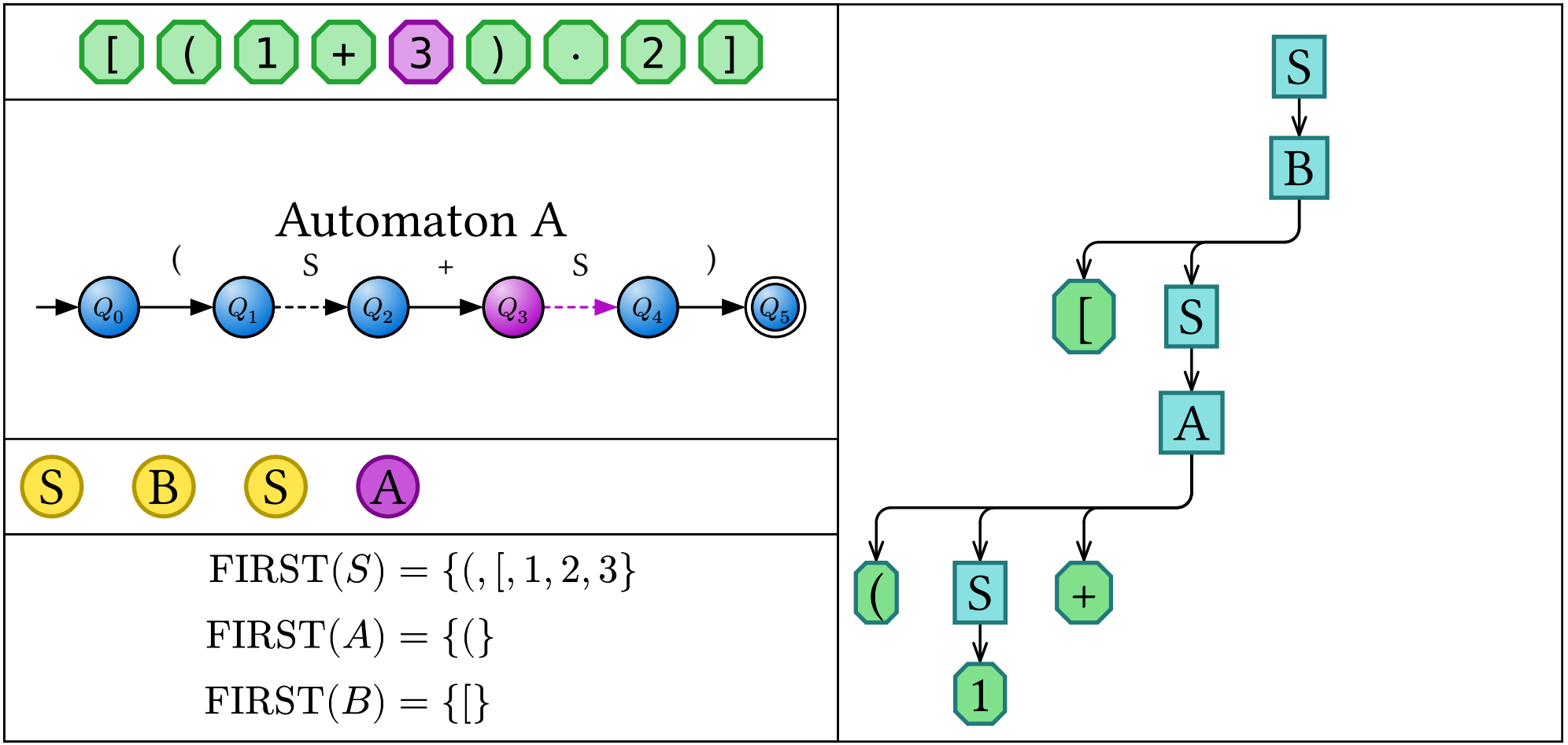
3.2 Building a Parse tree



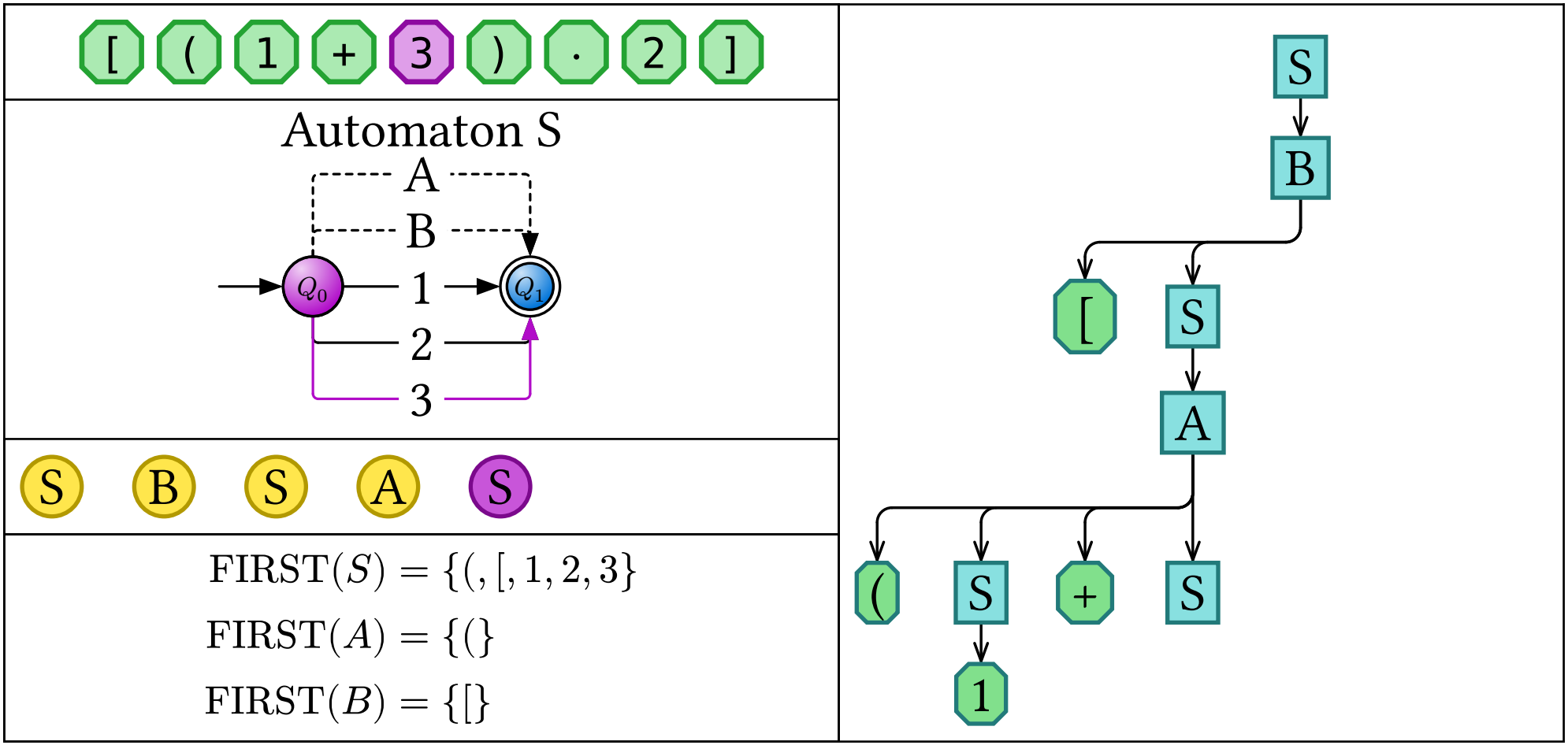
3.2 Building a Parse tree



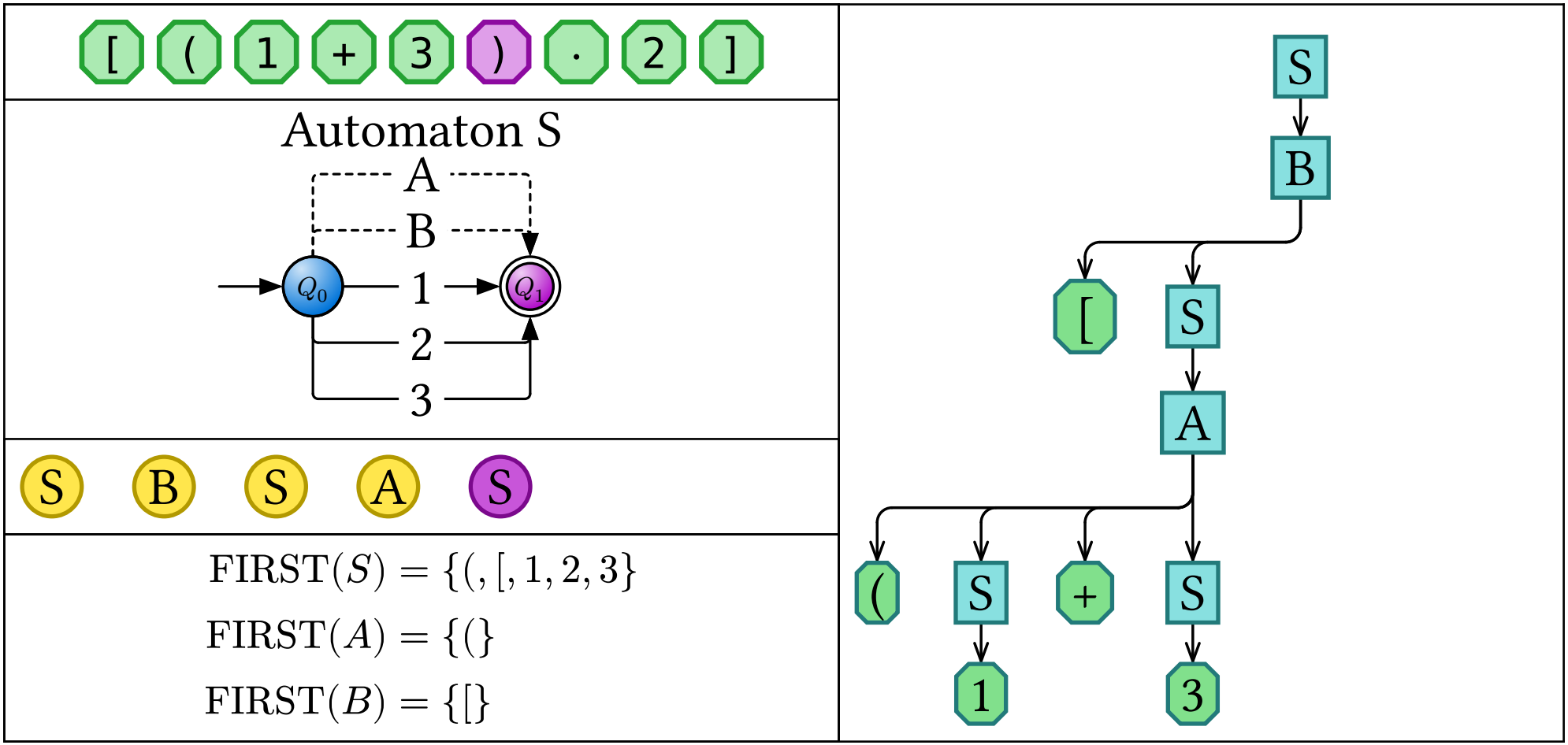
3.2 Building a Parse tree



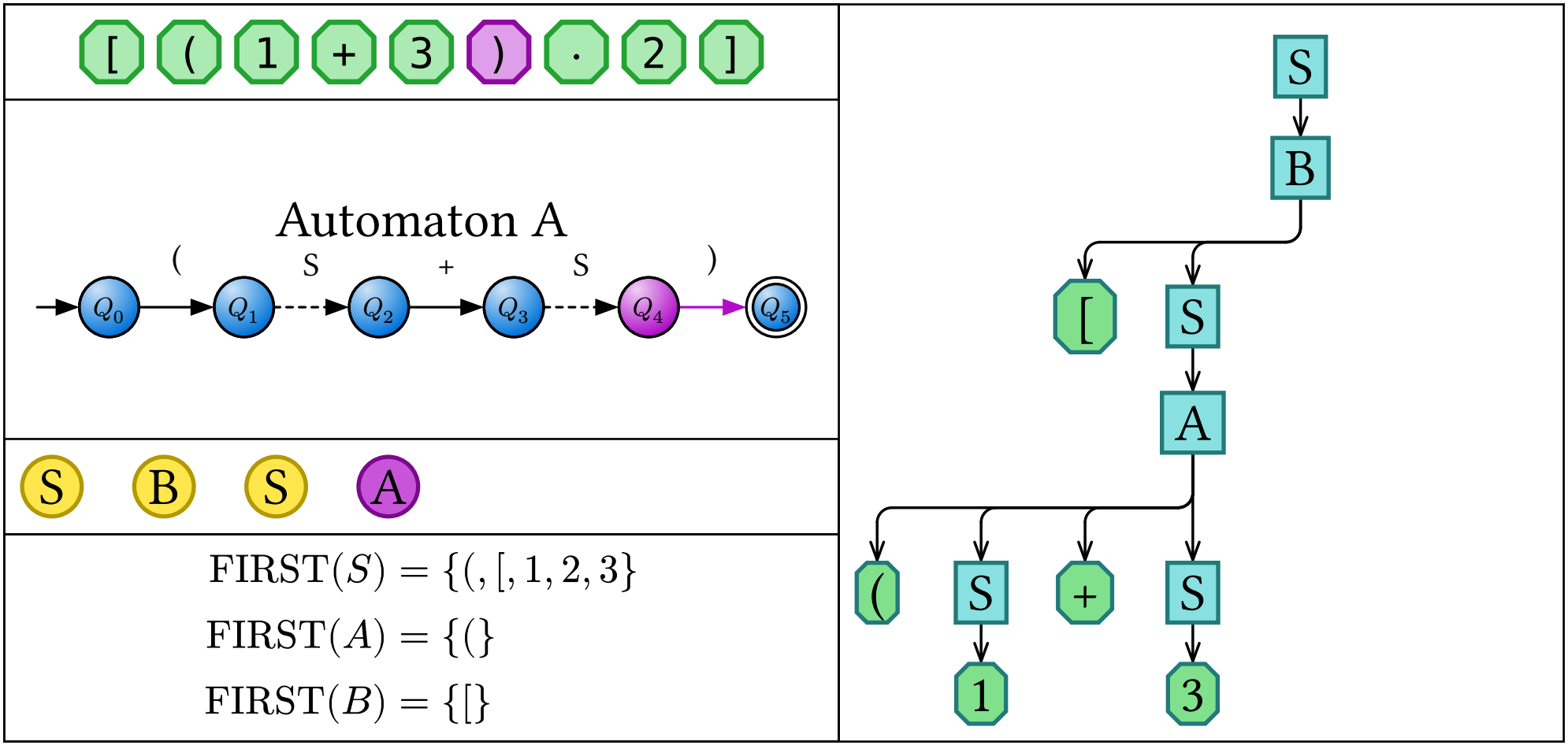
3.2 Building a Parse tree



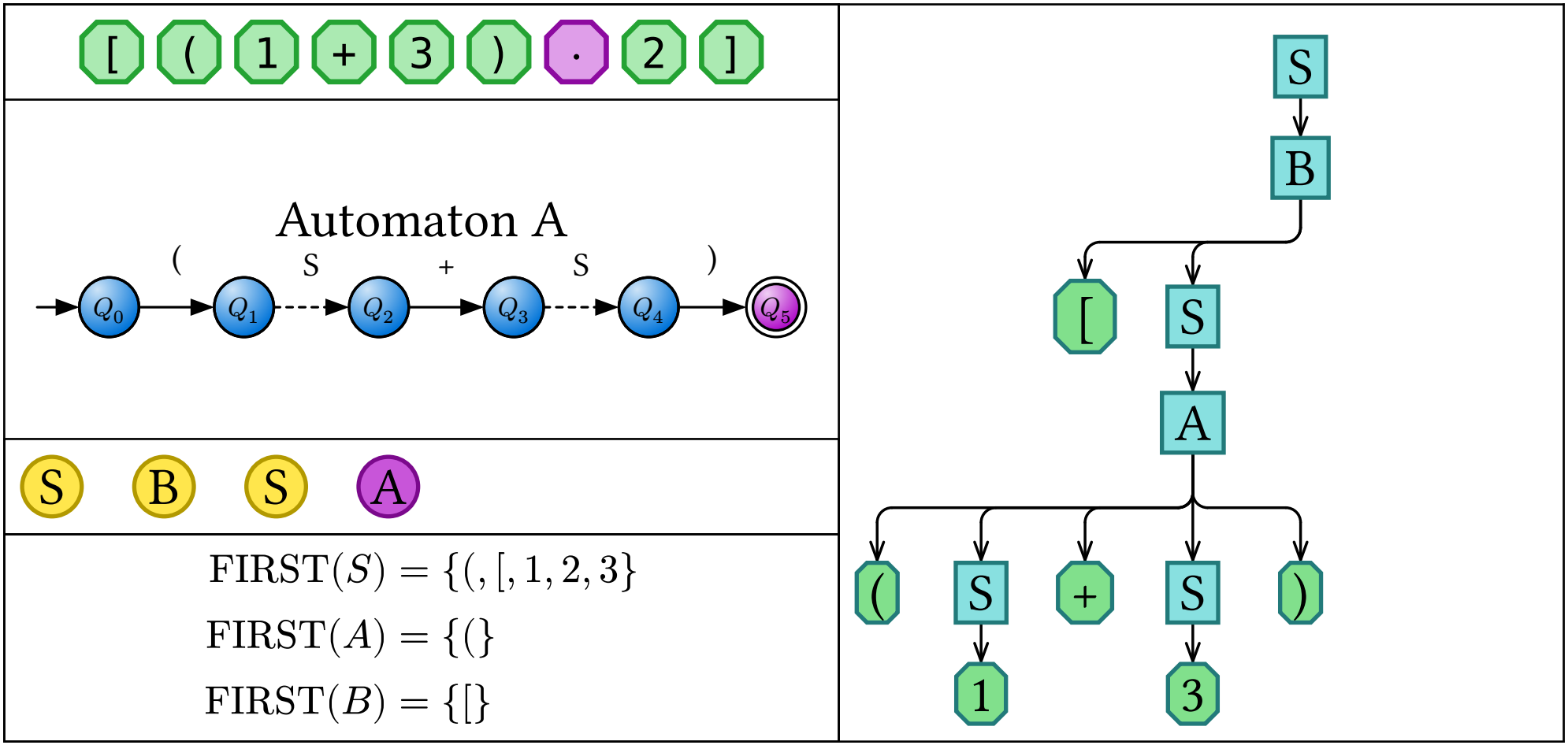
3.2 Building a Parse tree



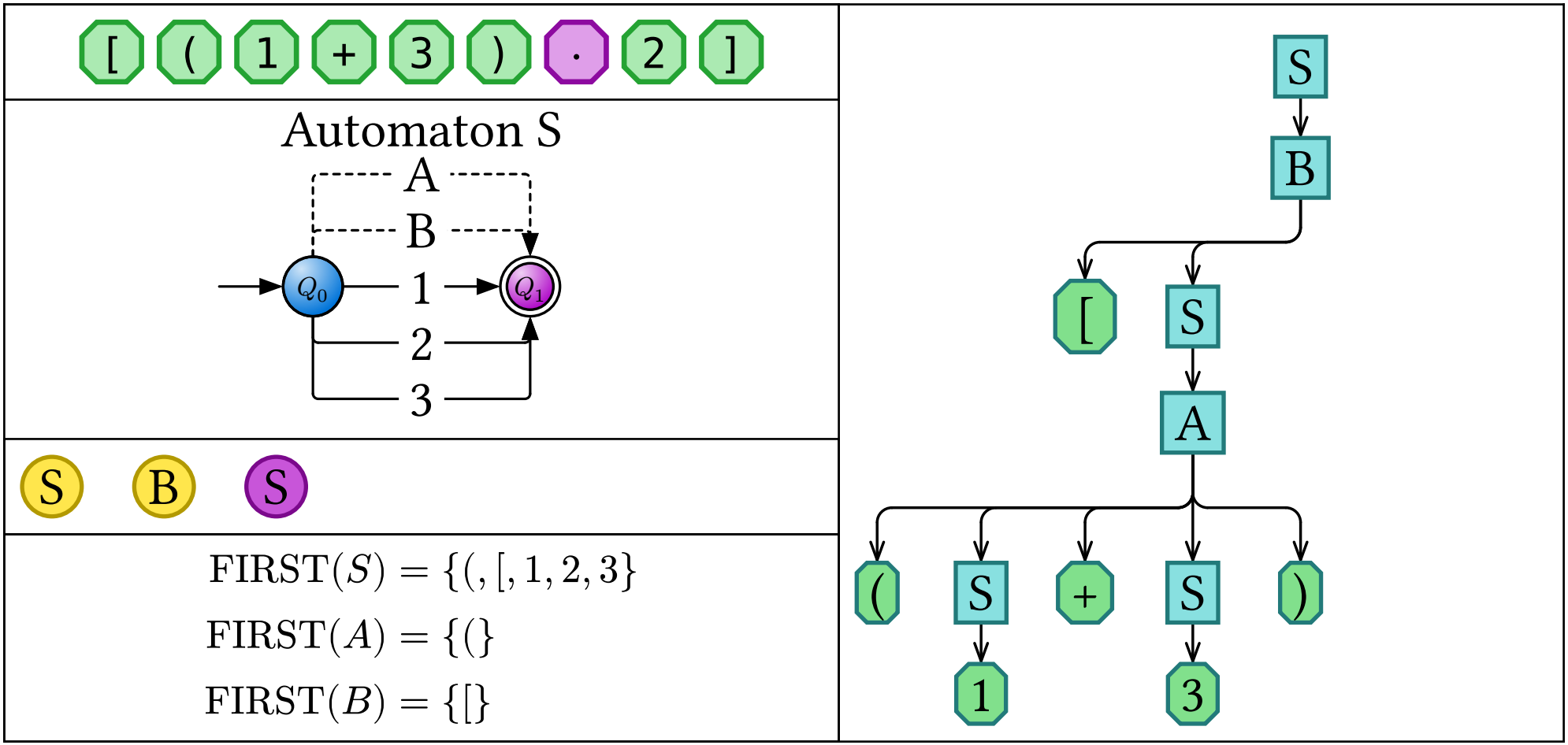
3.2 Building a Parse tree



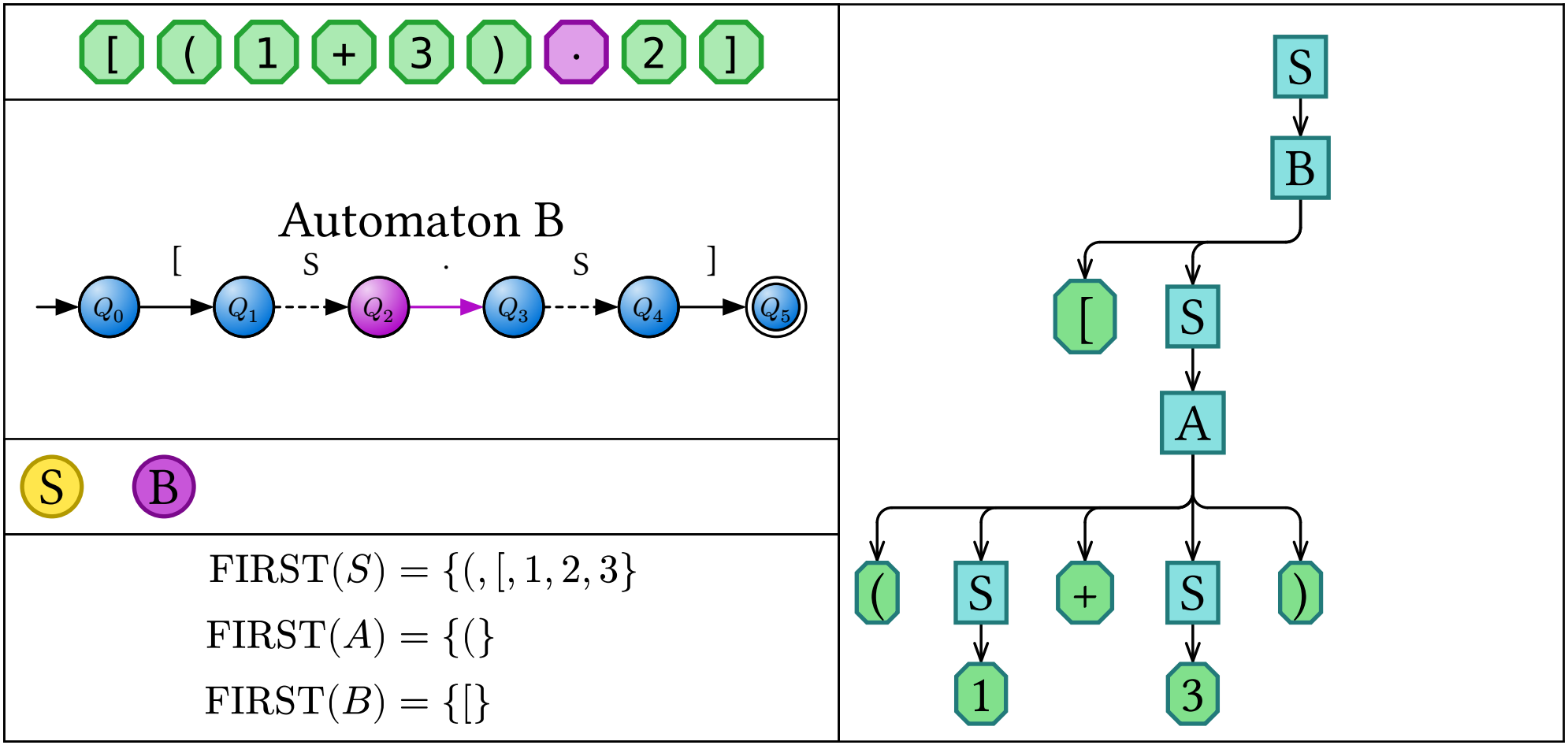
3.2 Building a Parse tree



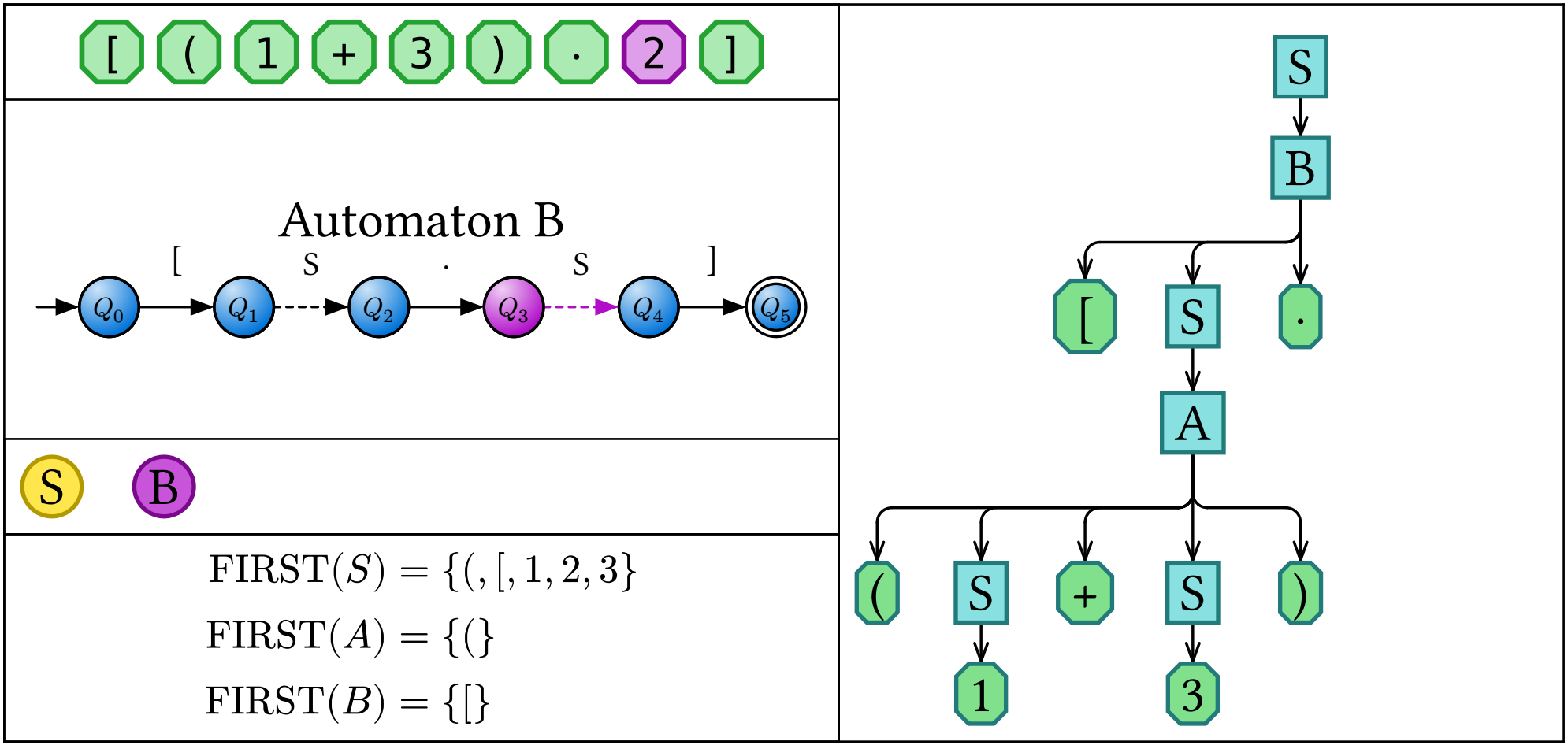
3.2 Building a Parse tree



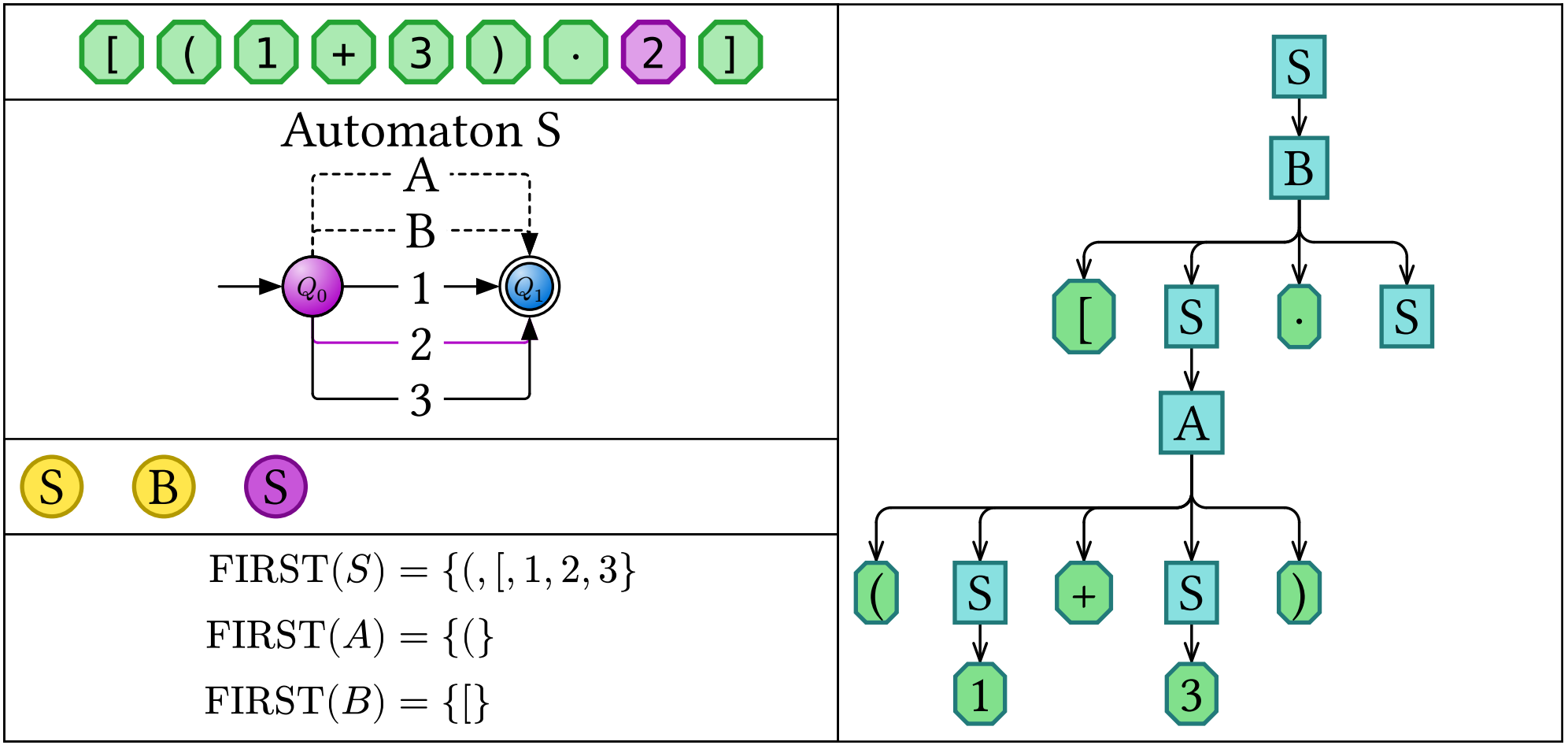
3.2 Building a Parse tree



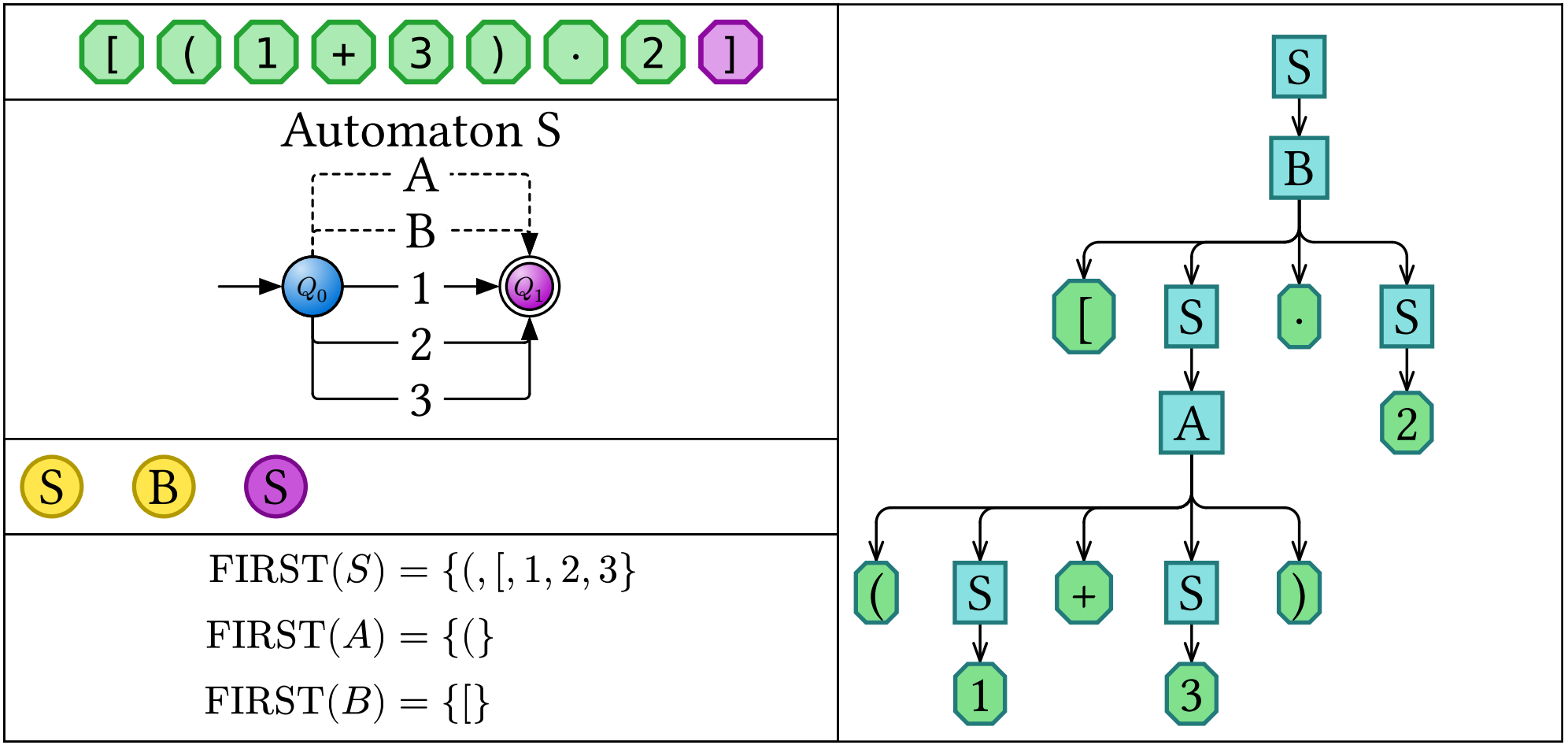
3.2 Building a Parse tree



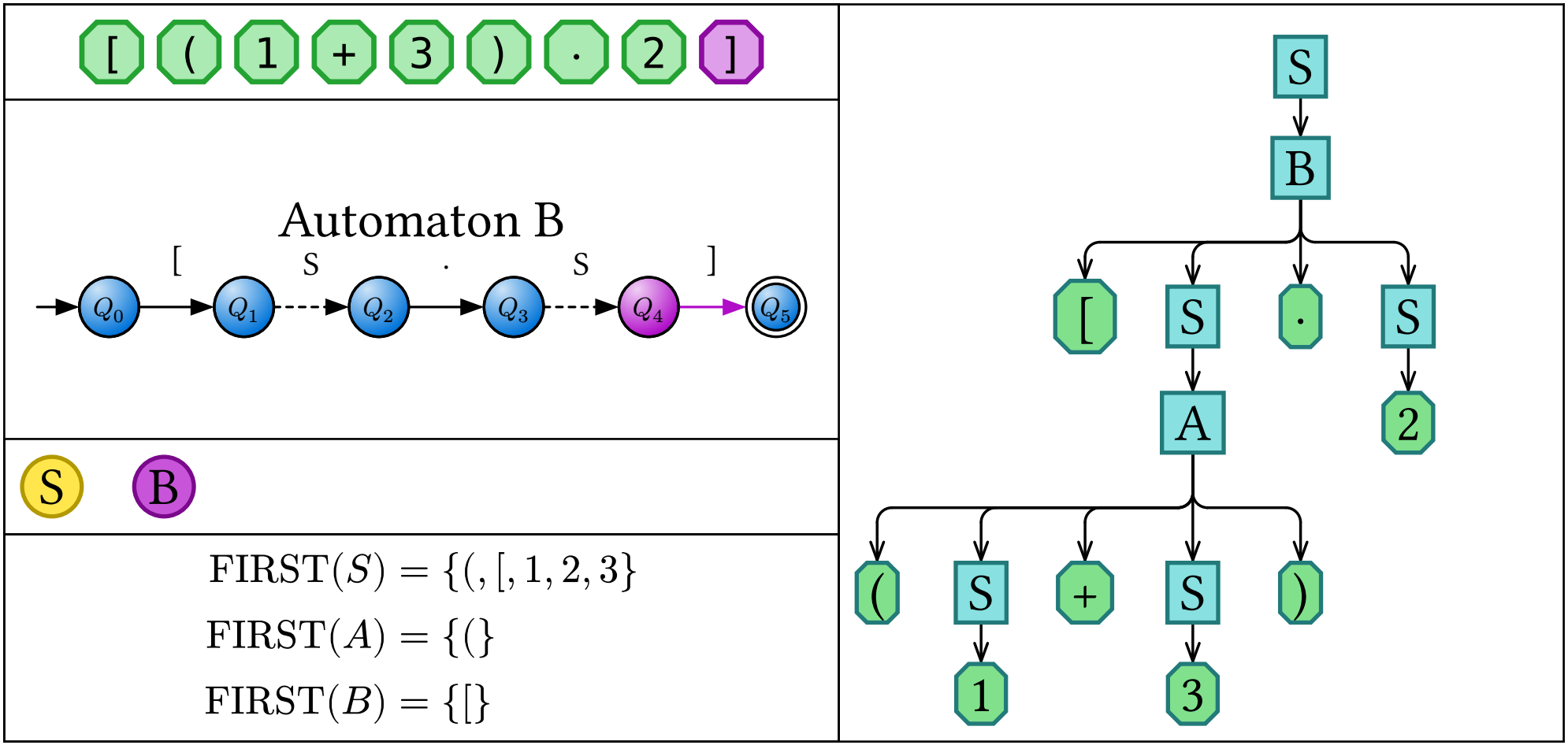
3.2 Building a Parse tree



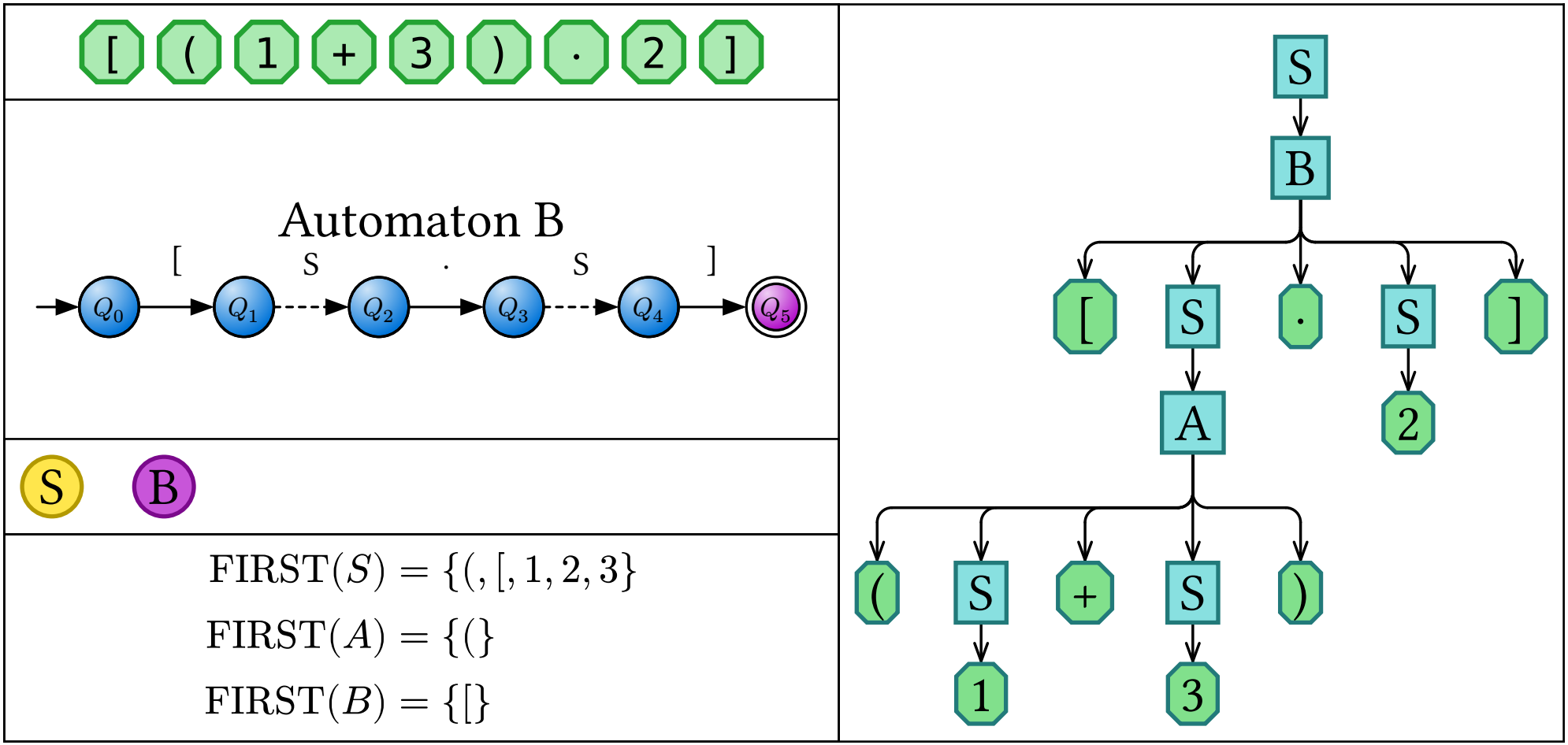
3.2 Building a Parse tree



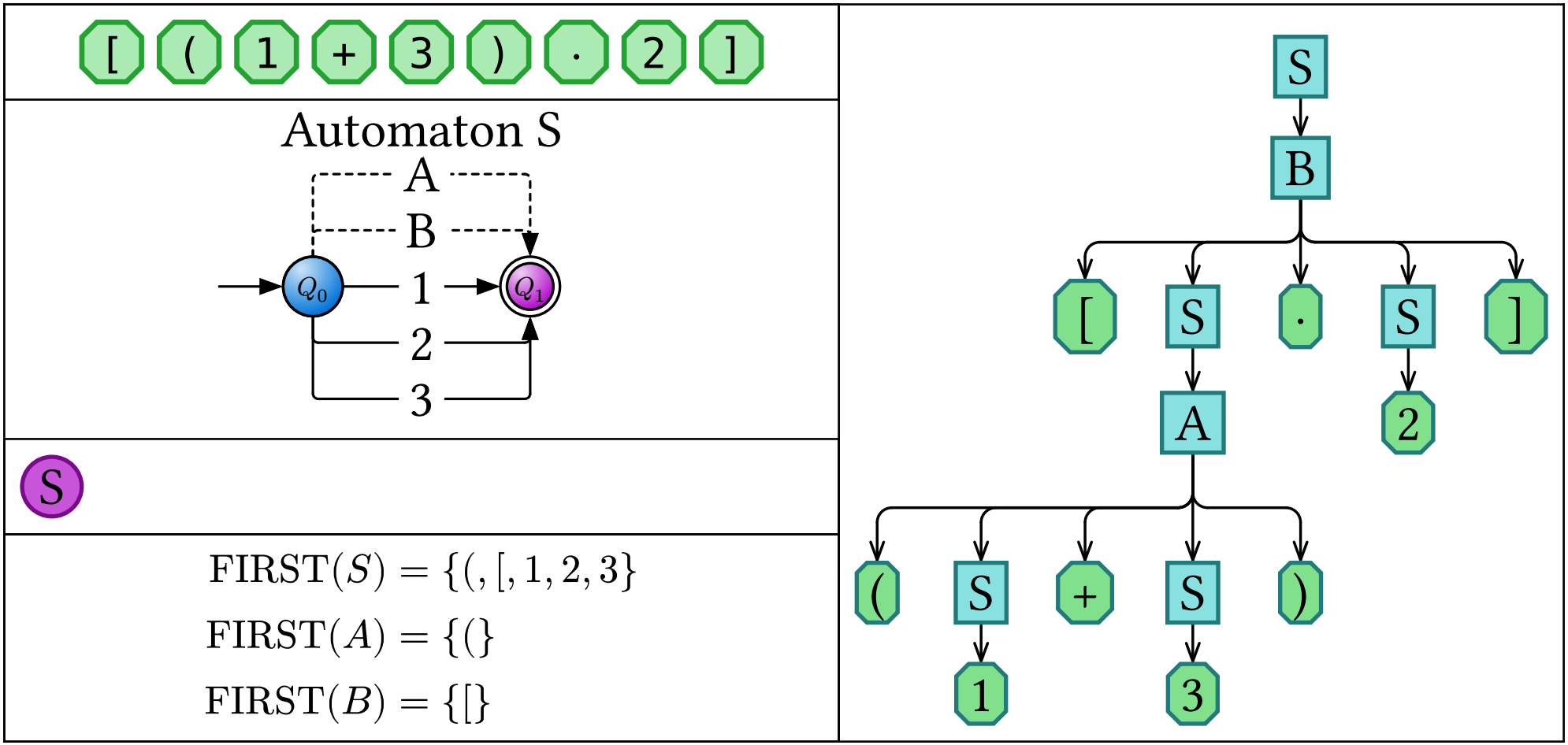
3.2 Building a Parse tree



3.2 Building a Parse tree

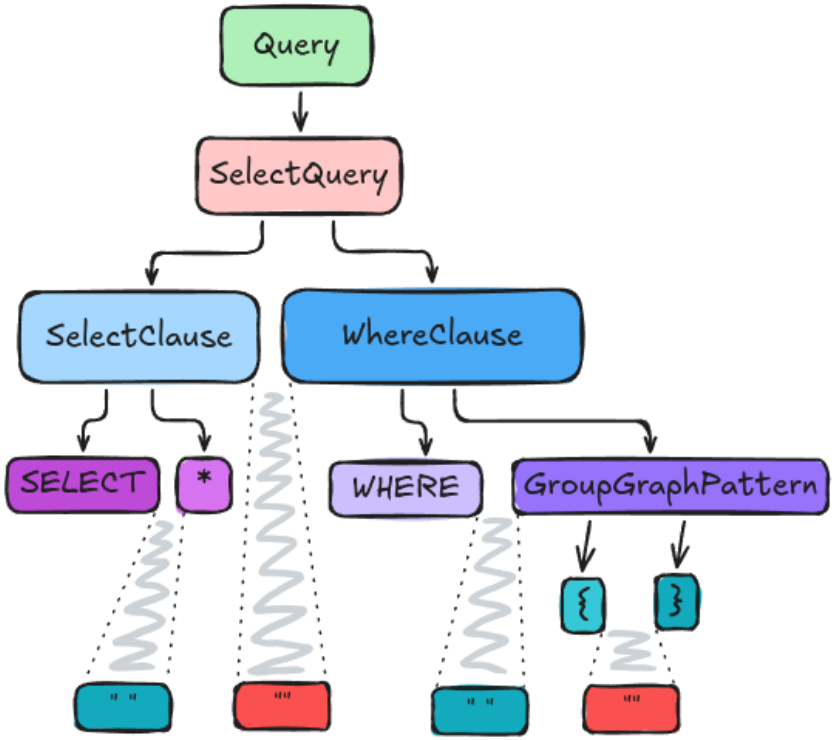


3.2 Building a Parse tree



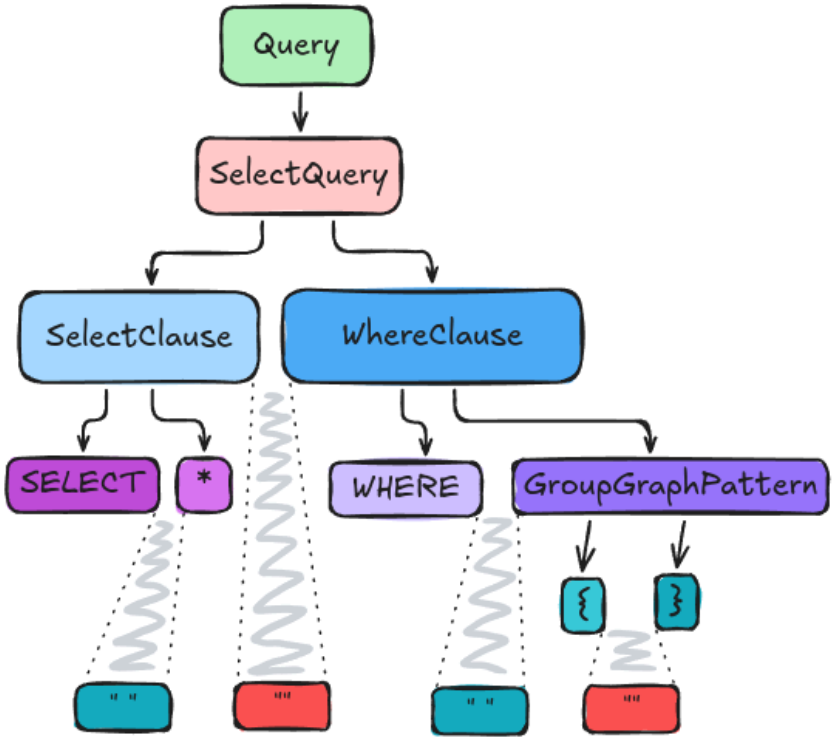
3.3 Implementing each capability

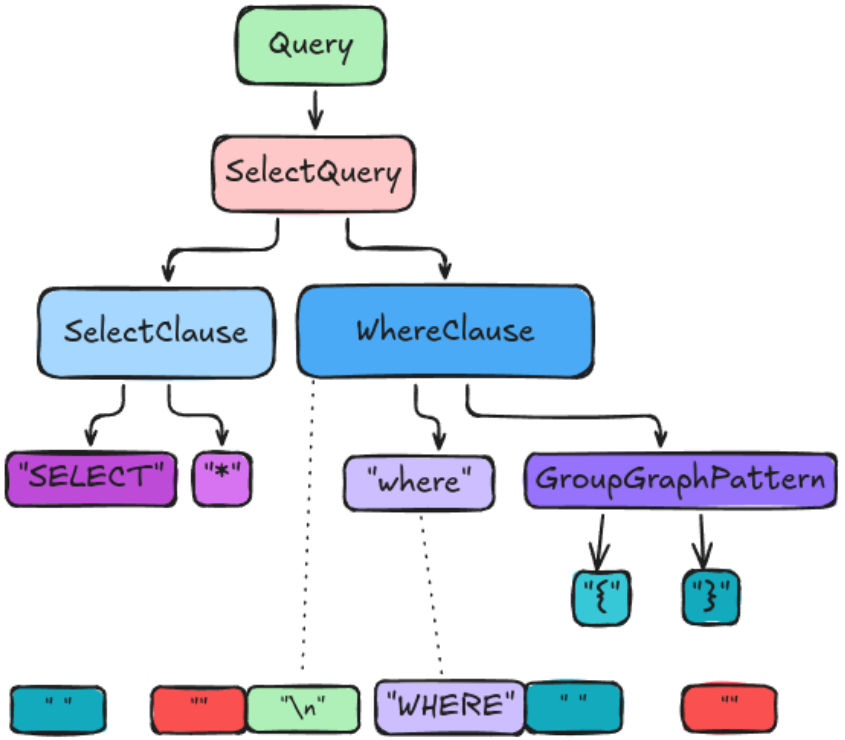
3.3.1 Formatting



3.3 Implementing each capability

3.3.1 Formatting





3.3 Implementing each capability

3.3.2 Completion

3.3.3 Diagnostics

- unused prefix declaration
- Uncompressed IRI
- Ungrouped select variable
- Invalid Projection Variable
- Same Subject

3.3 Implementing each capability

Example: Uncompressed IRI

```
1  SELECT * WHERE {
2    <http://www.wikidata.org/entity/Q5>
```

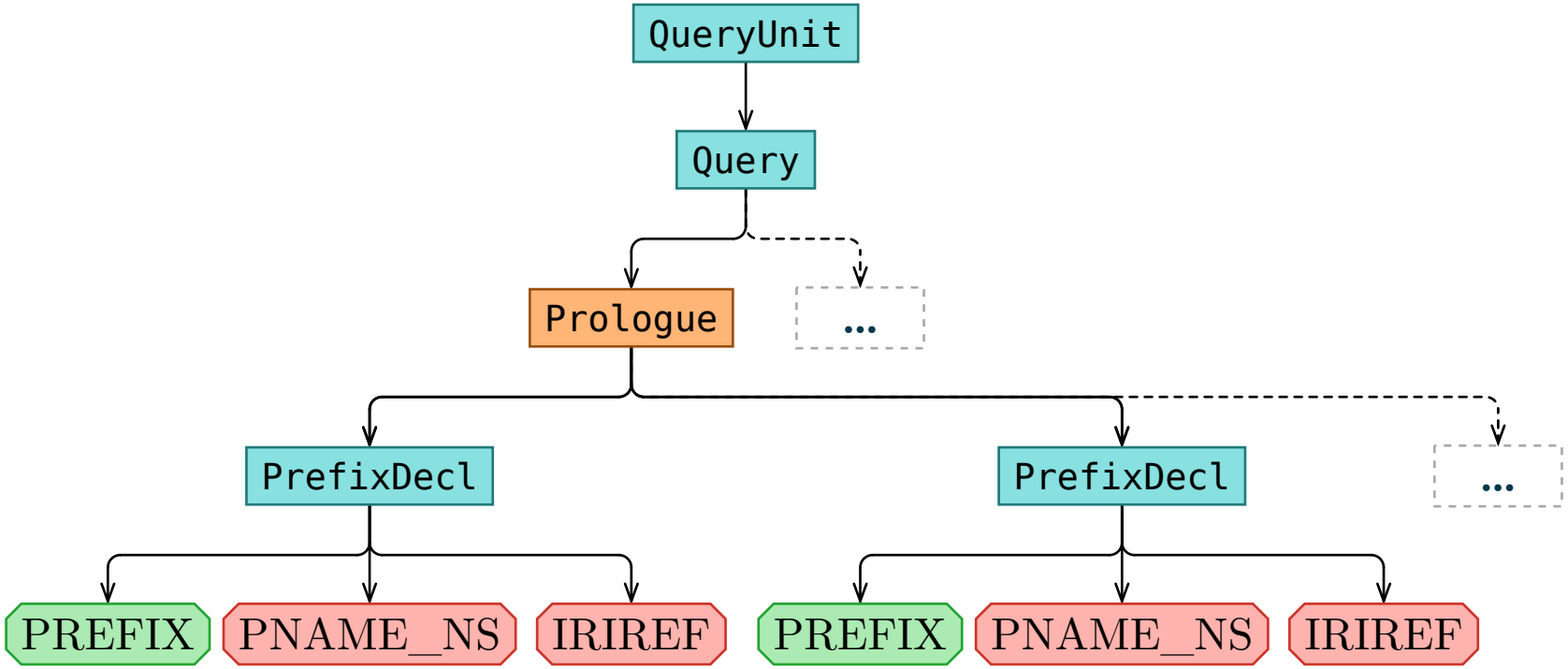
query.rq 1 of 2 problems

```
You might want to shorten this Uri
http://www.wikidata.org/entity/Q5 -> wd:Q5 (uncompacted-uri)
```

```
3  }
```

1. find declared & known prefixes
2. find uncompressed IRI
3. compare both sets

3.3 Implementing each capability



3.3 Implementing each capability

3.3.4 Code-actions

3.3.5 Hover

4. Evaluation