Syntactical Analysis
using Context-Free Grammars (CFG)

Before we dive into CFG...

Before we dive into CFG...

- We want to **recognize** the source code as a structured object

Before we dive into CFG...

- We want to **recognize** the source code as a structured object
- As a DFA (or NFA) recognizes a regular language, a Push Down Automaton (PDA) recognizes a non-regular language

Before we dive into CFG...

- We want to **recognize** the source code as a structured object
- As a DFA (or NFA) recognizes a regular language, a Push Down Automaton (PDA) recognizes a non-regular language
- To describe a push down automaton, it can be convenient to do it through a CFG, as it is more convenient to describe a DFA with a very elegant and simple RE (instead of state transition, etc)

Context-Free Grammar (CFG)

A CFG is a 4-tuple (T, N, S, P) such that

Context-Free Grammar (CFG)

A CFG is a 4-tuple (T, N, S, P) such that

- T is a finite set of terminal symbols (constants or tokenized strings described by regular expressions)

Context-Free Grammar (CFG)

A CFG is a 4-tuple (T, N, S, P) such that

- T is a finite set of terminal symbols (constants or tokenized strings described by regular expressions)
- N is a finite set of non-terminal symbols

Context-Free Grammar (CFG)

A CFG is a 4-tuple (T, N, S, P) such that

- T is a finite set of terminal symbols (constants or tokenized strings described by regular expressions)
- N is a finite set of non-terminal symbols
- $S \in N$ is a start non-terminal symbol

Context-Free Grammar (CFG)

A CFG is a 4-tuple (T, N, S, P) such that

- T is a finite set of terminal symbols (constants or tokenized strings described by regular expressions)
- N is a finite set of non-terminal symbols
- $S \in N$ is a start non-terminal symbol
- P is a finite set of production rules, a relation from N to $(T \cup N)^*$. The symbol * denotes the Kleene star (ε is also allowed!)

Context-Free Grammar (CFG)

A CFG is a 4-tuple (T, N, S, P) such that

- T is a finite set of terminal symbols (constants or tokenized strings described by regular expressions)
- N is a finite set of non-terminal symbols
- $S \in N$ is a start non-terminal symbol
- P is a finite set of production rules, a relation from N to $(T \cup N)^*$. The symbol * denotes the Kleene star (ε is also allowed!)
- $p \in P$ has the notation $N \mapsto \alpha$, where $\alpha \in (T \cup N)^*$. This means, in the left hand side of the production rule there is only one non-terminal (which makes it context-free), and on the right hand a sequence of terminal and non-terminal symbols

CFG Example

Consider the following REs:

CFG Example

Consider the following REs:

- $op = +|-|/|*$
- $int = [0-9]^{+}$
- $opar = ($
- $cpar =)$

CFG Example

Consider the following REs:

- $op = +|-|/|*$
- $int = [0-9]^{+}$
- $opar = ($
- $cpar =)$

Consider the following production rules

CFG Example

Consider the following REs:

- $op = +|-|/|*$
- $int = [0-9]^+$
- $opar = ($
- $cpar =)$

Consider the following production rules

- $Start \mapsto Expr$
- $Expr \mapsto Expr\ op\ Expr$
- $Expr \mapsto int$
- $Expr \mapsto opar\ Expr\ cpar$

Using the CFG to generate "a sentence"

the (informal) algorithm

Using the CFG to generate "a sentence"

the (informal) algorithm

- Begin with the start symbol (Start in the previous CFG)

Using the CFG to generate "a sentence"

the (informal) algorithm

- Begin with the start symbol (Start in the previous CFG)
- Loop until non-terminal symbols appear

Using the CFG to generate "a sentence"

the (informal) algorithm

- Begin with the start symbol (Start in the previous CFG)
- Loop until non-terminal symbols appear
  - Choose a non-terminal from the current Right Hand Side (RHS) of the production rule

Using the CFG to generate "a sentence"

the (informal) algorithm

- Begin with the start symbol (Start in the previous CFG)
- Loop until non-terminal symbols appear
  - Choose a non-terminal from the current Right Hand Side (RHS) of the production rule
  - Choose a production rule from the previously selected non-terminal symbol

Using the CFG to generate "a sentence"

the (informal) algorithm

- Begin with the start symbol (Start in the previous CFG)
- Loop until non-terminal symbols appear
  - Choose a non-terminal from the current Right Hand Side (RHS) of the production rule
  - Choose a production rule from the previously selected non-terminal symbol
  - Replace the chosen non-terminal symbol by substituting Left Hand Side (LHS) non-terminal symbol for the RHS sequence in the chosen production rule

Using the CFG to generate "a sentence"

the (informal) algorithm

- Begin with the start symbol (Start in the previous CFG)
- Loop until non-terminal symbols appear
  - Choose a non-terminal from the current Right Hand Side (RHS) of the production rule
  - Choose a production rule from the previously selected non-terminal symbol
  - Replace the chosen non-terminal symbol by substituting Left Hand Side (LHS) non-terminal symbol for the RHS sequence in the chosen production rule
- Generate the necessary strings from the token REs

Using the CFG to generate "a sentence"

the (informal) algorithm

- Begin with the start symbol (Start in the previous CFG)
- Loop until non-terminal symbols appear
  - Choose a non-terminal from the current Right Hand Side (RHS) of the production rule
  - Choose a production rule from the previously selected non-terminal symbol
  - Replace the chosen non-terminal symbol by substituting Left Hand Side (LHS) non-terminal symbol for the RHS sequence in the chosen production rule
- Generate the necessary strings from the token REs

The obtained string belongs the CFG language!

Using the CFG to generate "a sentence"

the (informal) algorithm

- Begin with the start symbol (Start in the previous CFG)
- Loop until non-terminal symbols appear
  - Choose a non-terminal from the current Right Hand Side (RHS) of the production rule
  - Choose a production rule from the previously selected non-terminal symbol
  - Replace the chosen non-terminal symbol by substituting Left Hand Side (LHS) non-terminal symbol for the RHS sequence in the chosen production rule
- Generate the necessary strings from the token REs

The obtained string belongs the CFG language!

- Important note: different choices produce different "sentences"

Example CFG, generating a string of the CFG

- $op = +|-|/|*$
- $int = [0-9]^+$
- $opar = ($
- $cpar =)$

1. $Start \mapsto Expr$
2. $Expr \mapsto Expr\ op\ Expr$
3. $Expr \mapsto int$
4. $Expr \mapsto opar\ Expr\ cpar$

Example CFG, generating a string of the CFG

Let's derive...

- $op = +|-|/|*$
- $int = [0-9]^+$
- $opar = ($
- $cpar =)$

1. $Start \mapsto Expr$
2. $Expr \mapsto Expr\ op\ Expr$
3. $Expr \mapsto int$
4. $Expr \mapsto opar\ Expr\ cpar$

Example CFG, generating a string of the CFG

Let's derive...

- $Start$

- $op = +|-|/|*$
- $int = [0-9]^+$
- $opar = ($
- $cpar =)$

1. $Start \mapsto Expr$
2. $Expr \mapsto Expr\ op\ Expr$
3. $Expr \mapsto int$
4. $Expr \mapsto opar\ Expr\ cpar$

Example CFG, generating a string of the CFG

- $op = +|-|/|*$
- $int = [0-9]^+$
- $opar = ($
- $cpar =)$

Let's derive...

- *Start*
- *Expr* (1)

1. $Start \mapsto Expr$
2. $Expr \mapsto Expr\ op\ Expr$
3. $Expr \mapsto int$
4. $Expr \mapsto opar\ Expr\ cpar$

Example CFG, generating a string of the CFG

- $op = +|-|/|*$
- $int = [0-9]^+$
- $opar = ($
- $cpar =)$

Let's derive...

- *Start*
- *Expr* (1)
- *Expr op Expr* (2)

1. $Start \mapsto Expr$
2. $Expr \mapsto Expr\ op\ Expr$
3. $Expr \mapsto int$
4. $Expr \mapsto opar\ Expr\ cpar$

Example CFG, generating a string of the CFG

- $op = +|-|/|*$
- $int = [0-9]^+$
- $opar = ($
- $cpar =)$

Let's derive...

- *Start*
- *Expr* (1)
- *Expr op Expr* (2)
- *int op Expr* (3)

1. $Start \mapsto Expr$
2. $Expr \mapsto Expr\ op\ Expr$
3. $Expr \mapsto int$
4. $Expr \mapsto opar\ Expr\ cpar$

Example CFG, generating a string of the CFG

- $op = +|-|/|*$
- $int = [0-9]^+$
- $opar = ($
- $cpar =)$

Let's derive...

- *Start*
- *Expr* (1)
- *Expr op Expr* (2)
- *int op Expr* (3)
- *int op opar Expr cpar* (4)

1. $Start \mapsto Expr$
2. $Expr \mapsto Expr\ op\ Expr$
3. $Expr \mapsto int$
4. $Expr \mapsto opar\ Expr\ cpar$

Example CFG, generating a string of the CFG

- $op = +| - |/|*$
- $int = [0 - 9]^+$
- $opar = ($
- $cpar =)$

1. $Start \mapsto Expr$
2. $Expr \mapsto Expr\ op\ Expr$
3. $Expr \mapsto int$
4. $Expr \mapsto opar\ Expr\ cpar$

Let's derive...

- *Start*
- *Expr* (1)
- *Expr op Expr* (2)
- *int op Expr* (3)
- *int op opar Expr cpar* (4)
- *int op opar Expr op Expr cpar* (2)

Example CFG, generating a string of the CFG

- $op = +|-|/|*$
- $int = [0-9]^+$
- $opar = ($
- $cpar =)$

1. $Start \mapsto Expr$
2. $Expr \mapsto Expr\ op\ Expr$
3. $Expr \mapsto int$
4. $Expr \mapsto opar\ Expr\ cpar$

Let's derive...

- *Start*
- *Expr* (1)
- *Expr op Expr* (2)
- *int op Expr* (3)
- *int op opar Expr cpar* (4)
- *int op opar Expr op Expr cpar* (2)
- *int op opar int op Expr cpar* (3)

Example CFG, generating a string of the CFG

- $op = +|-|/|*$
- $int = [0-9]^+$
- $opar = ($
- $cpar =)$

1. $Start \mapsto Expr$
2. $Expr \mapsto Expr\ op\ Expr$
3. $Expr \mapsto int$
4. $Expr \mapsto opar\ Expr\ cpar$

Let's derive...

- *Start*
- *Expr* (1)
- *Expr op Expr* (2)
- *int op Expr* (3)
- *int op opar Expr cpar* (4)
- *int op opar Expr op Expr cpar* (2)
- *int op opar int op Expr cpar* (3)
- *int op opar int op int cpar* (3)

Example CFG, generating a string of the CFG

- $op = +|-|/|*$
- $int = [0-9]^+$
- $opar = ($
- $cpar =)$

1. $Start \mapsto Expr$
2. $Expr \mapsto Expr\ op\ Expr$
3. $Expr \mapsto int$
4. $Expr \mapsto opar\ Expr\ cpar$

Let's derive...

- *Start*
- *Expr* (1)
- *Expr op Expr* (2)
- *int op Expr* (3)
- *int op opar Expr cpar* (4)
- *int op opar Expr op Expr cpar* (2)
- *int op opar int op Expr cpar* (3)
- *int op opar int op int cpar* (3)
- 10 / (7 − 5)

Example CFG, generating a string of the CFG

- $op = +|-|/|*$
- $int = [0-9]^+$
- $opar = ($
- $cpar =)$

1. $Start \mapsto Expr$
2. $Expr \mapsto Expr\ op\ Expr$
3. $Expr \mapsto int$
4. $Expr \mapsto opar\ Expr\ cpar$

Let's derive...

- *Start*
- *Expr* (1)
- *Expr op Expr* (2)
- *int op Expr* (3)
- *int op opar Expr cpar* (4)
- *int op opar Expr op Expr cpar* (2)
- *int op opar int op Expr cpar* (3)
- *int op opar int op int cpar* (3)
- 10 / (7 − 5)

Please, derive one yourselves

Constructing a CFG

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

Constructing a CFG

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>López</td>
    <td>80</td>
  </tr>
</table>
```

Constructing a CFG

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>López</td>
    <td>80</td>
  </tr>
</table>
```

Produces (close-enough):

Name	**Lastname**	**Score**
Eve	Jackson-Renard	94
Vladimir	López	80

Constructing a CFG (cont.)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

Constructing a CFG (cont.)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

REs:

- table attributes (class, border, etc.)

Constructing a CFG (cont.)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

REs:

- $attributes = style|class|border|\ldots$

Constructing a CFG (cont.)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

REs:

- $attributes = style|class|border|\ldots$
- values of the attributes

Constructing a CFG (cont.)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

REs:

- $attributes = style|class|border|\ldots$
- $values = \backslash"[\hat{}\ \backslash"]^* \backslash"$

Constructing a CFG (cont.)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

REs:

- $attributes = style|class|border|\ldots$
- $values = \backslash"[\hat{}\ \backslash"]^* \backslash"$
- data

Constructing a CFG (cont.)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

REs:

- $attributes = style|class|border|\ldots$
- $values = \backslash"[\hat{}\ \backslash"]^*\ \backslash"$
- $data = [a-zA-Z0-9]^*\ldots$

Constructing a CFG (cont.++)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

Constructing a CFG (cont.++)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

Production rules:

Constructing a CFG (cont.++)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

Production rules:

1. $S \mapsto < table\ A1 > S2 < /table >$

CONSTRUCTING A CFG (CONT.++)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

Production rules:

1. $S \mapsto < table\ A1 > S2 < /table >$
2. $S2 \mapsto < tr > S3 < /tr >$

CONSTRUCTING A CFG (CONT.++)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

Production rules:

1. $S \mapsto < table\ A1 > S2 < /table >$
2. $S2 \mapsto < tr > S3 < /tr >$
3. $S2 \mapsto S2\ S2$

Constructing a CFG (cont.++)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

Production rules:

1. $S \mapsto < table\ A1 > S2 < /table >$
2. $S2 \mapsto < tr > S3 < /tr >$
3. $S2 \mapsto S2\ S2$
4. $S3 \mapsto < th > S4 < /th >$

Constructing a CFG (cont.++)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

Production rules:

1. $S \mapsto < table\ A1 > S2 < /table >$
2. $S2 \mapsto < tr > S3 < /tr >$
3. $S2 \mapsto S2\ S2$
4. $S3 \mapsto < th > S4 < /th >$
5. $S3 \mapsto < td > S4 < /td >$

Constructing a CFG (cont.++)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

Production rules:

1. $S \mapsto < table\ A1 > S2 < /table >$
2. $S2 \mapsto < tr > S3 < /tr >$
3. $S2 \mapsto S2\ S2$
4. $S3 \mapsto < th > S4 < /th >$
5. $S3 \mapsto < td > S4 < /td >$
6. $S3 \mapsto S3\ S3$

Constructing a CFG (cont.++)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

Production rules:

1. $S \mapsto < table\ A1 > S2 < /table >$
2. $S2 \mapsto < tr > S3 < /tr >$
3. $S2 \mapsto S2\ S2$
4. $S3 \mapsto < th > S4 < /th >$
5. $S3 \mapsto < td > S4 < /td >$
6. $S3 \mapsto S3\ S3$
7. $S4 \mapsto data$

CONSTRUCTING A CFG (CONT.++)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

Production rules:

1. $S \mapsto < table\ A1 > S2 < /table >$
2. $S2 \mapsto < tr > S3 < /tr >$
3. $S2 \mapsto S2\ S2$
4. $S3 \mapsto < th > S4 < /th >$
5. $S3 \mapsto < td > S4 < /td >$
6. $S3 \mapsto S3\ S3$
7. $S4 \mapsto data$
8. $S4 \mapsto S2$ //(assume no table def needed)

CONSTRUCTING A CFG (CONT.++)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

Production rules:

1. $S \mapsto < table\ A1 > S2 < /table >$
2. $S2 \mapsto < tr > S3 < /tr >$
3. $S2 \mapsto S2\ S2$
4. $S3 \mapsto < th > S4 < /th >$
5. $S3 \mapsto < td > S4 < /td >$
6. $S3 \mapsto S3\ S3$
7. $S4 \mapsto data$
8. $S4 \mapsto S2$ //(assume no table def needed)
9. $A1 \mapsto attributes = values$

CONSTRUCTING A CFG (CONT.++)

Let's assume we want to describe the non-regular language for a for HTML tables (looks like this):

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th>Lastname</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson-Renard</td>
    <td>94</td>
  </tr>
  <tr>
    <td>Vladimir</td>
    <td>Lopez</td>
    <td>80</td>
  </tr>
</table>
```

Production rules:

1. $S \mapsto < table\ A1 > S2 < /table >$
2. $S2 \mapsto < tr > S3 < /tr >$
3. $S2 \mapsto S2\ S2$
4. $S3 \mapsto < th > S4 < /th >$
5. $S3 \mapsto < td > S4 < /td >$
6. $S3 \mapsto S3\ S3$
7. $S4 \mapsto data$
8. $S4 \mapsto S2$ //(assume no table def needed)
9. $A1 \mapsto attributes = values$
10. $A1 \mapsto A1\ A1$

Exercises – Describing languages through CFG

Please, describe the grammar for the language:

Exercises – Describing languages through CFG

Please, describe the grammar for the language:

- Balanced parenthesis (at least one pair), e.g,(()())((()))(())

Exercises – Describing languages through CFG

Please, describe the grammar for the language:

- Balanced parenthesis (at least one pair), e.g,(()())((()))(())
  - $P \mapsto PP$
  - $P \mapsto (P)$
  - $P \mapsto ()$ //E-Z, right?

Exercises – Describing languages through CFG

Please, describe the grammar for the language:

- Balanced parenthesis (at least one pair), e.g,(()())((()))(())
  - $P \mapsto PP$
  - $P \mapsto (P)$
  - $P \mapsto ()$ //E-Z, right?
- The non-regular language $\{a^m b^n a^m : m \geq 1, n \geq 0\}$, e.g. aaabaaa, abbba, aabbaa, aaaa

Exercises – Describing languages through CFG

Please, describe the grammar for the language:

- Balanced parenthesis (at least one pair), e.g,(()())((()))(())
  - $P \mapsto PP$
  - $P \mapsto (P)$
  - $P \mapsto ()$ //E-Z, right?
- The non-regular language $\{a^m b^n a^m : m \geq 1, n \geq 0\}$, e.g. aaabaaa, abbba, aabbaa, aaaa
  - $S \mapsto aSa|aS_1a$ // Note the "OR"(|) notation
  - $S_1 \mapsto b|bS_1|\varepsilon$ //E-Zer, right?

Exercises – Describing languages through CFG

Please, describe the grammar for the language:

- Balanced parenthesis (at least one pair), e.g,(()())((()))(())
  - $P \mapsto PP$
  - $P \mapsto (P)$
  - $P \mapsto ()$ //E-Z, right?
- The non-regular language $\{a^m b^n a^m : m \geq 1, n \geq 0\}$, e.g. aaabaaa, abbba, aabbaa, aaaa
  - $S \mapsto aSa|aS_1a$ // Note the "OR"(|) notation
  - $S_1 \mapsto b|bS_1|\varepsilon$ //E-Zer, right?
- Last one, balanced brackets [] and (), including ε quite simple for you now…

Exercises – Describing languages through CFG

Please, describe the grammar for the language:

- Balanced parenthesis (at least one pair), e.g,(()())((()))(())
  - $P \mapsto PP$
  - $P \mapsto (P)$
  - $P \mapsto ()$ //E-Z, right?
- The non-regular language $\{a^m b^n a^m : m \geq 1, n \geq 0\}$, e.g. aaabaaa, abbba, aabbaa, aaaa
  - $S \mapsto aSa|aS_1a$ // Note the "OR"(|) notation
  - $S_1 \mapsto b|bS_1|\varepsilon$ //E-Zer, right?
- Last one, balanced brackets [] and (), including ε quite simple for you now...
  - $S \mapsto SS|(S)|[S]|()|[]|\varepsilon$

Grammars...

Every regular grammar is a CFG

GRAMMARS...

Every regular grammar is a CFG

- The grammar generated by: $(a|b)^*$ can be generated by $S \mapsto aS|bS|\varepsilon$

Grammars...

Every regular grammar is a CFG

- The grammar generated by: $(a|b)^*$ can be generated by $S \mapsto aS|bS|\varepsilon$

Not every CFG is a regular grammar (canonical balanced parenthesis example)

Grammars...

Every regular grammar is a CFG

- The grammar generated by: $(a|b)^*$ can be generated by $S \mapsto aS|bS|\varepsilon$

Not every CFG is a regular grammar (canonical balanced parenthesis example)

A hierarchical power of grammars, and corresponding automata

Grammars...

Every regular grammar is a CFG

- The grammar generated by: $(a|b)^*$ can be generated by $S \mapsto aS|bS|\varepsilon$

Not every CFG is a regular grammar (canonical balanced parenthesis example)

A hierarchical power of grammars, and corresponding automata

- Regular Grammars & Finite State Automata
- Context-Free Grammars & Push Down Automata
- Context-Sensitive Grammars & Turing Machines
- I mean, just in case you forgot :)

You are here ↓

What we know

You are here ↓

What we know

- How to describe context-free languages using CFGs + REs

You are here ↓

What we know

- How to describe context-free languages using CFGs + REs
- The generative vs. the recognition approach of grammars and automata

You are here ↓

What we know

- How to describe context-free languages using CFGs + REs
- The generative vs. the recognition approach of grammars and automata

What we do not know (yet)

You are here ↓

What we know

- How to describe context-free languages using CFGs + REs
- The generative vs. the recognition approach of grammars and automata

What we do not know (yet)

- How to obtain the structure to analyze the code (which was the final goal) :(

You are here ↓

What we know

- How to describe context-free languages using CFGs + REs
- The generative vs. the recognition approach of grammars and automata

What we do not know (yet)

- How to obtain the structure to analyze the code (which was the final goal) :(
- To obtain such structure, we focus on the grammar derivation, grammar ambiguities are harmful, and we will see how to correct them

You are here ↓

What we know

- How to describe context-free languages using CFGs + REs
- The generative vs. the recognition approach of grammars and automata

What we do not know (yet)

- How to obtain the structure to analyze the code (which was the final goal) :(
- To obtain such structure, we focus on the grammar derivation, grammar ambiguities are harmful, and we will see how to correct them
- But, do not worry, next Tuesday(?) you'll know...