Compilers
Lab IV

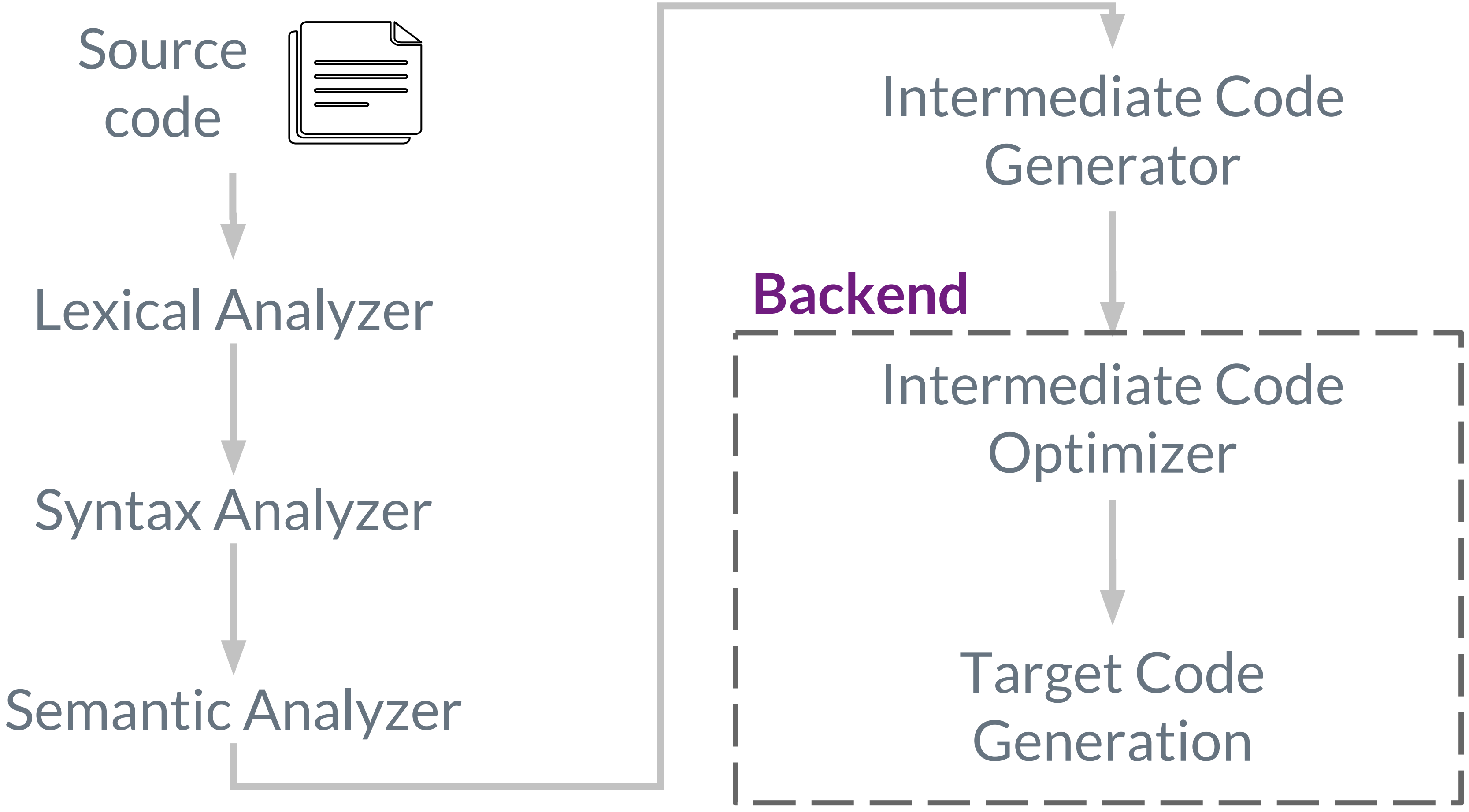
Mo.Agamia

Plan

- Overview
- Grammar
- Left recursion elimination & factoring
- ANTLR grammar

1.
Overview

Compiler phases



Source code

Lexical Analyzer

Syntax Analyzer

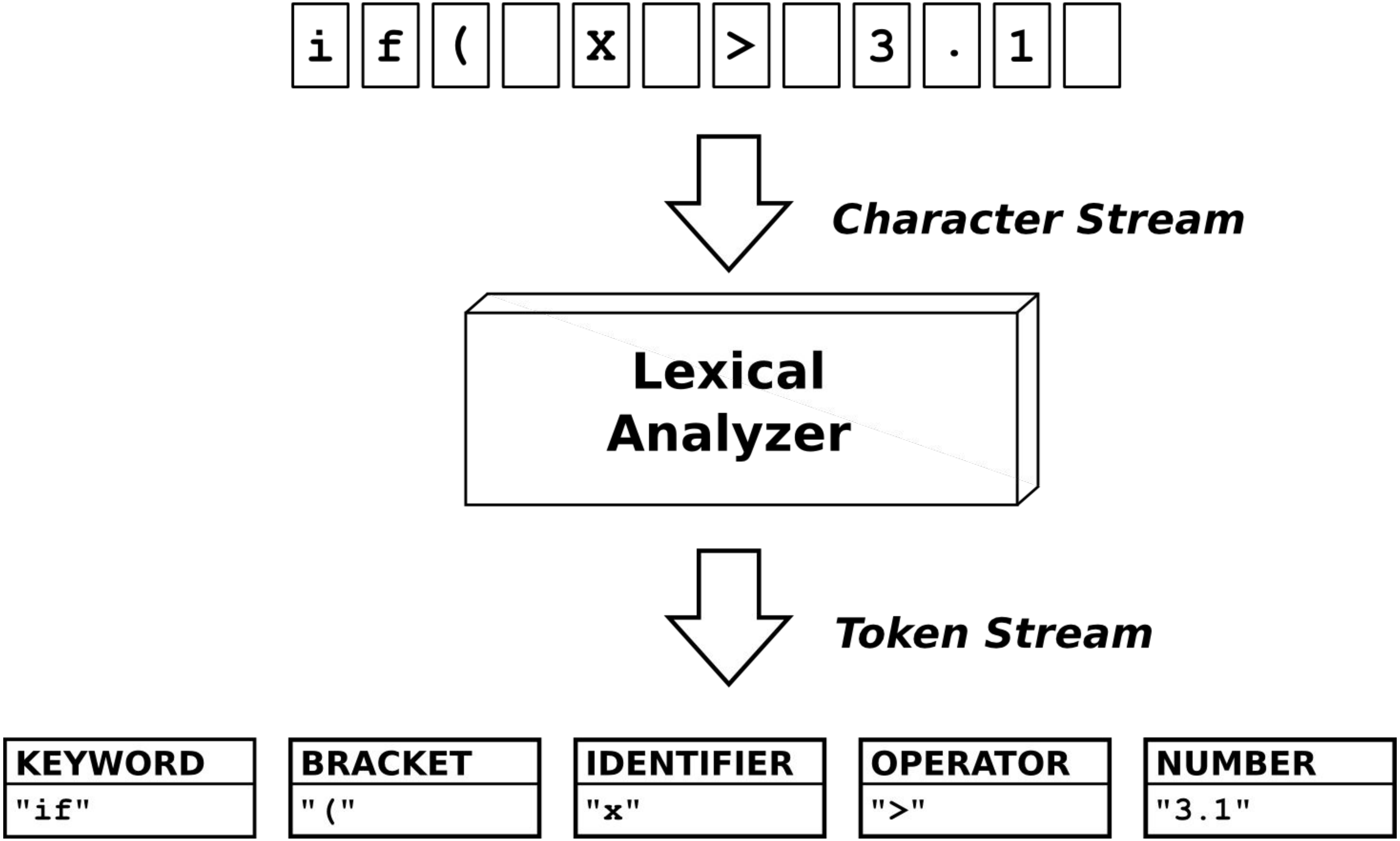
Semantic Analyzer

Intermediate Code Generator

Backend

Intermediate Code Optimizer

Target Code Generation



i f (x > 3 . 1

Character Stream

Lexical Analyzer

Token Stream

KEYWORD	BRACKET	IDENTIFIER	OPERATOR	NUMBER
"if"	"("	"x"	">"	"3.1"

Compiler phases

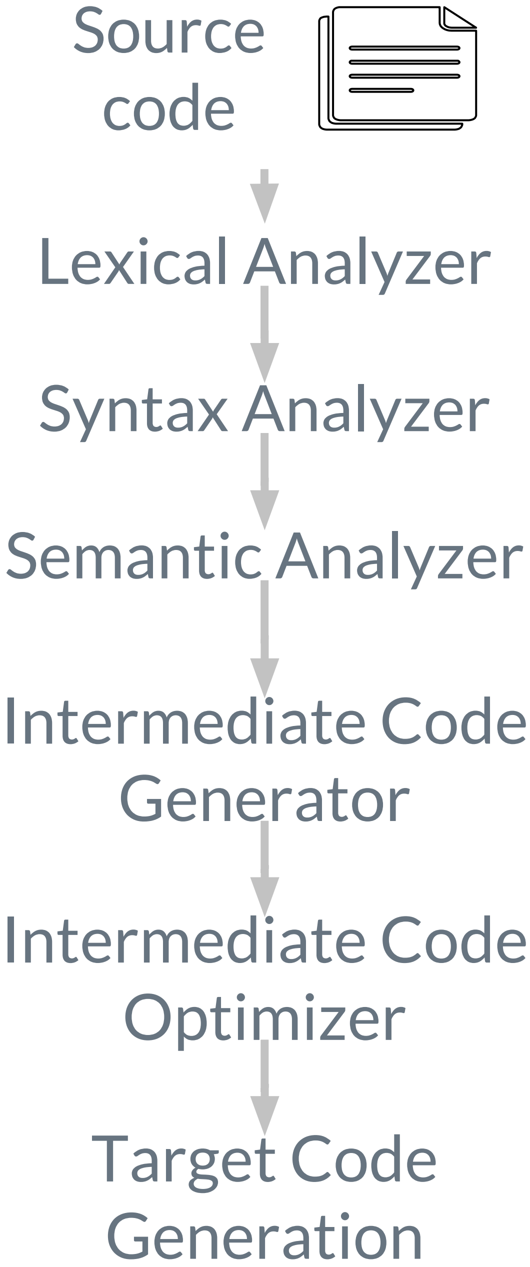
- x = a + b * c //statement
- Id = id + id * id
- Parse tree
- Verify parse tree semantically

S → id = E

E → E+T/T

T → T*F/F

F → id



Source code

Lexical Analyzer

Syntax Analyzer

Semantic Analyzer

Intermediate Code Generator

Intermediate Code Optimizer

Target Code Generation

2.
Grammar

Grammar is a set of production rules that describe all possible strings in a given formal language.

Grammar

Issues with grammars:

- Epsilon
- Ambiguity
- Left recursion
  - Immediate
  - Indirect
- Left factoring
  - Immediate
  - Indirect
- Cycles
- Unit productions

Grammar

G = (V, T, P, S)

- ▷ V is variables
- ▷ T is terminals
- ▷ P is productions
- ▷ S is start symbol

Example :

$E \rightarrow$ E + E |

E * E |

id

$G \rightarrow$

Ambiguous | Unambiguous

$G \rightarrow$

Left | Right Recursive

$G \rightarrow$

Deterministic | Non-Deter.

Grammar

Example :

id + id * id

E → E + E |

E * E |

id

E → E + T | T

T → T * F | F

F → id

Grammar

$$A \rightarrow A\ a \mid b \qquad \Longleftrightarrow \qquad \begin{array}{c} A \rightarrow b\ A' \\ A' \rightarrow a\ A' \mid \varepsilon \end{array}$$

Example :

$E \rightarrow \quad E\ a \mid b$

$b\ a^*$

Example :

$E \rightarrow \quad a\ E \mid b$

$a^*\ b$

Grammar

Example : a d

E → a b |

a c |

a d

E → a E’

E’ → b | c | d

2.
Left recursion elimination & factoring

Left recursion elimination

Example :

- E -> E a | b

- S -> A a | B b | c

  A -> S c | B f | b d

  B -> B e | f

Left recursion elimination

Example :

- E -> E a | b

- S -> A a | B b | c

  A -> S c | B f | b d

  B -> B e | f

- E -> b E’

  E’ -> a E’ | ϵ

- S -> A a | B b | c

  A -> B b c A’ | c c A ‘ |

  B f A’ | b d

  A’ -> a c A’ | ϵ

  B -> f B’

  B’ -> e B’ | ϵ

Left factoring elimination

Example :

▷ A -> A c | A a d | b d | ϵ

▷ S -> a S S b S | a S a S b |
a b b | b

Left factoring elimination

Example :

- ▷ A -> A c | A a d | b d | ϵ

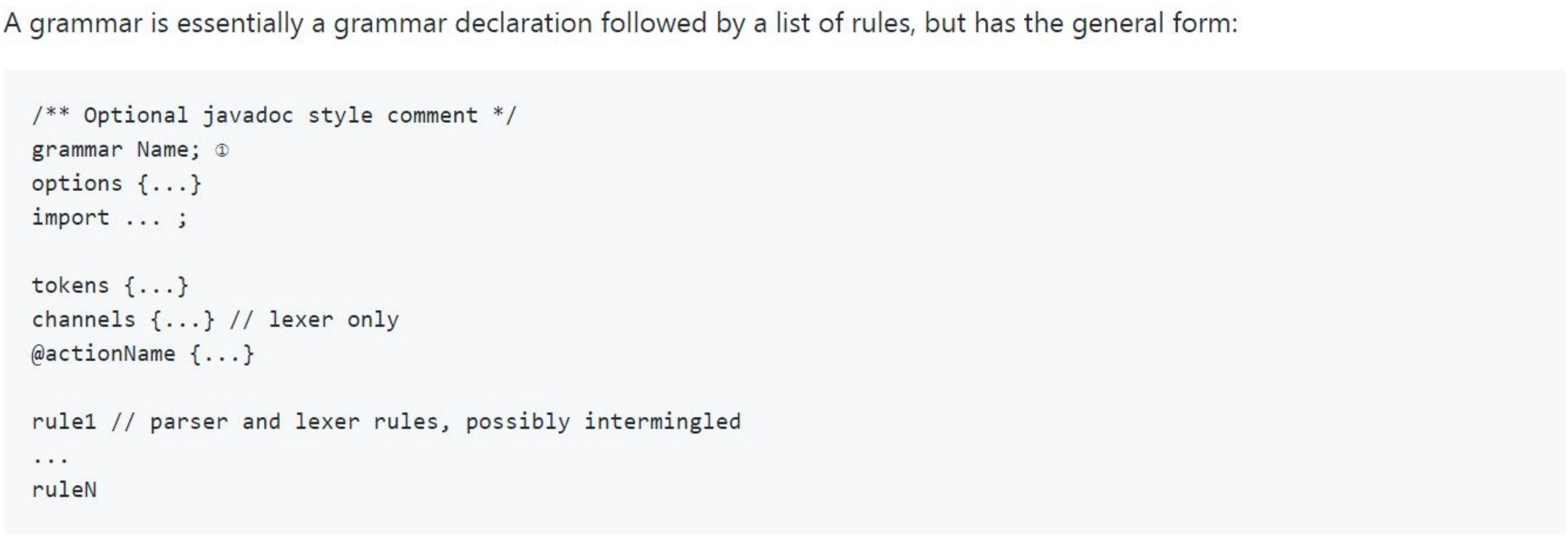
- ▷ A -> A A' | b d | ϵ
  A' -> c | a d

- ▷ S -> a S S b S | a S a S b |
  a b b | b

- ▷ S -> a S’ | b
  S’ -> S S” | bb
  S” -> S b S | a S b

1.
ANTLR Grammar

Structure



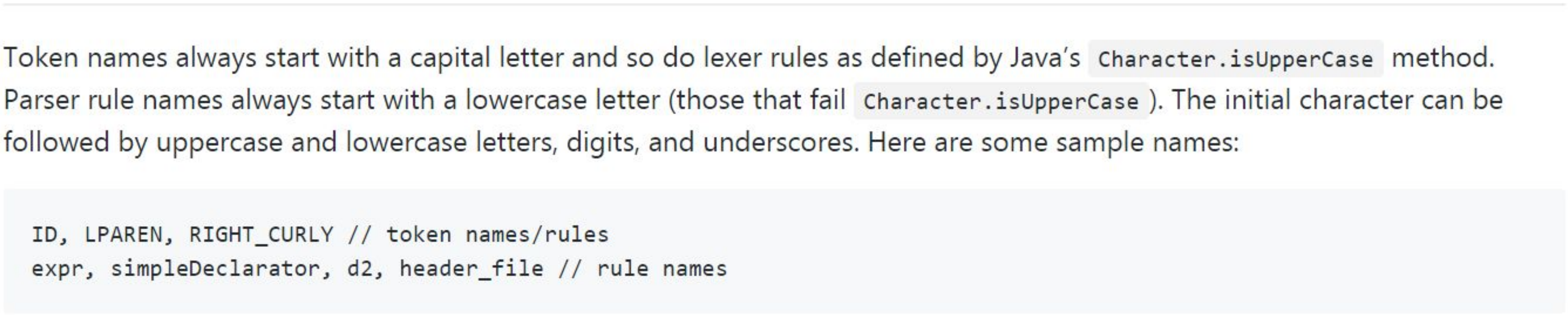
A grammar is essentially a grammar declaration followed by a list of rules, but has the general form:

```
/** Optional javadoc style comment */
grammar Name; ①
options {...}
import ... ;

tokens {...}
channels {...} // lexer only
@actionName {...}

rule1 // parser and lexer rules, possibly intermingled
...
ruleN
```

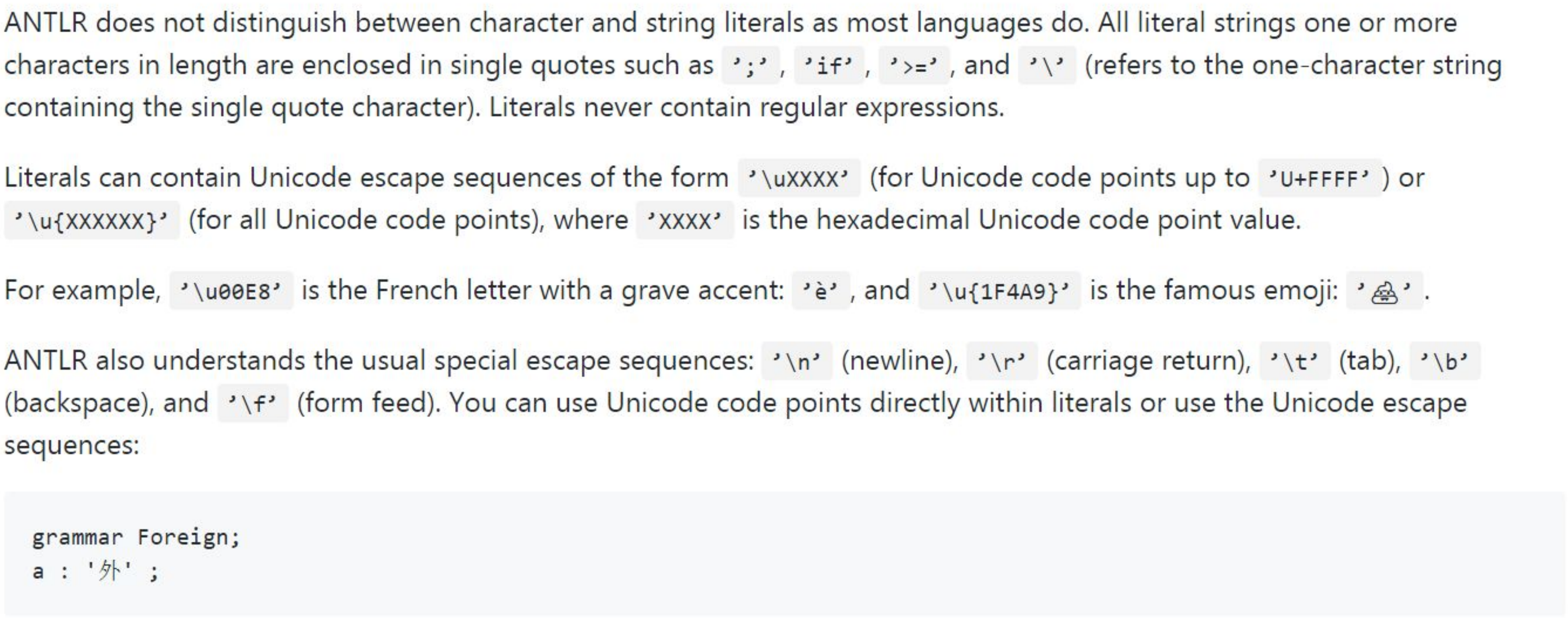
Identifiers



Token names always start with a capital letter and so do lexer rules as defined by Java's `Character.isUpperCase` method. Parser rule names always start with a lowercase letter (those that fail `Character.isUpperCase`). The initial character can be followed by uppercase and lowercase letters, digits, and underscores. Here are some sample names:

```
ID, LPAREN, RIGHT_CURLY // token names/rules
expr, simpleDeclarator, d2, header_file // rule names
```

Literals



ANTLR does not distinguish between character and string literals as most languages do. All literal strings one or more characters in length are enclosed in single quotes such as `’;’`, `’if’`, `’>=’`, and `’\’` (refers to the one-character string containing the single quote character). Literals never contain regular expressions.

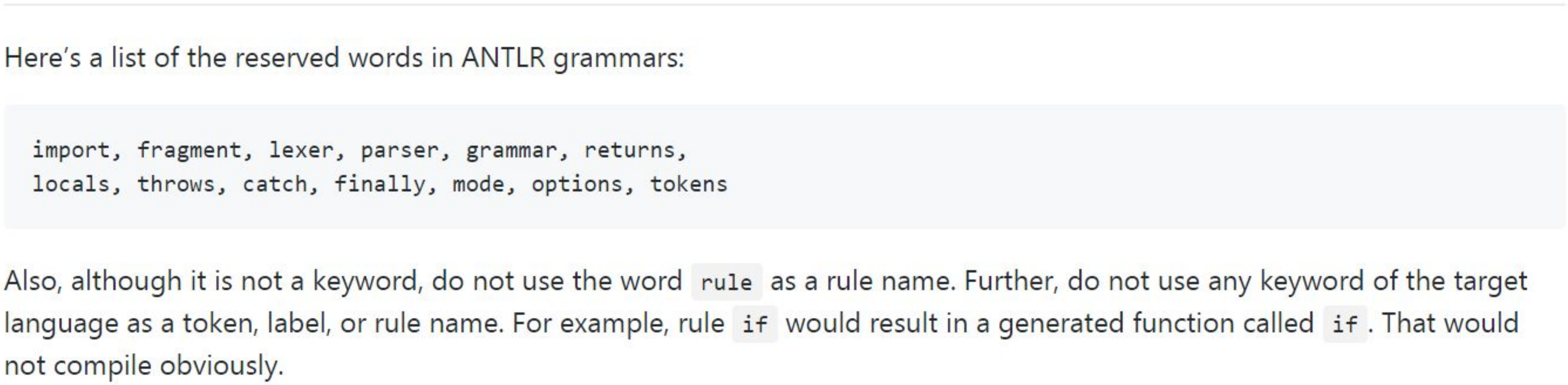
Literals can contain Unicode escape sequences of the form `’\uXXXX’` (for Unicode code points up to `’U+FFFF’`) or `’\u{XXXXXX}’` (for all Unicode code points), where `’XXXX’` is the hexadecimal Unicode code point value.

For example, `’\u00E8’` is the French letter with a grave accent: `’è’`, and `’\u{1F4A9}’` is the famous emoji: `’💩’`.

ANTLR also understands the usual special escape sequences: `’\n’` (newline), `’\r’` (carriage return), `’\t’` (tab), `’\b’` (backspace), and `’\f’` (form feed). You can use Unicode code points directly within literals or use the Unicode escape sequences:

```
grammar Foreign;
a : '外' ;
```

Keywords



Here's a list of the reserved words in ANTLR grammars:

```
import, fragment, lexer, parser, grammar, returns,
locals, throws, catch, finally, mode, options, tokens
```

Also, although it is not a keyword, do not use the word `rule` as a rule name. Further, do not use any keyword of the target language as a token, label, or rule name. For example, rule `if` would result in a generated function called `if`. That would not compile obviously.

Thanks!

Any questions?

You can find me at:

@piazza

mohammed.agamia@guc.edu.eg