

Exercise 01

01 Eigenschaften einer Gramatik

Besitzen die folgenden Grammatiken die LL(1)-Eigenschaft? Begründen Sie Ihre Antwort in textueller Form (warum LL(1), warum nicht LL(1)). Sollte es mindestens eine LL(1)-Verletzungen geben, geben Sie bitte alle an. Geben Sie zudem für jede Grammatik, welche die LL(1)-Eigenschaft nicht besitzt, die theoretisch minimale Anzahl an Vorgriffssymbolen an, die für das Parsen der jeweiligen Grammatik notwendig wäre.

Task a

```
1 Temperature = ["-"] ( Celsius | Kelvin ) .
2 Celsius = { number } "°C" .
3 Kelvin = { number } "K" .
```

Conflicts:

- $\text{first}(\text{Celsius}) \cap \text{first}(\text{Kelvin}) = \{\text{number}\} \Rightarrow \text{LL}(1) \text{ conflict}$
 - Both productions begin with the same token (number). This is a LL(1) conflict.
 - It is not even LL(k) because there can be infinite (number) token before the distinction between Celsius and Kelvin happens.

Task b

```
1 Locals = "class" ident "{" { VarDecl | FuncDecl } "}" .
2 VarDecl = "int" ident { ", " ident } ";" .
3 FuncDecl = "fn" "int" ident "(" ")" ";" .
```

$$\text{first}(\text{FuncDecl}) \cap \text{first}(\text{VarDecl}) = \{\text{"int"}\} \cap \{\text{"fn"}\} = \emptyset$$

$$\begin{aligned} & (\text{follow}(\text{VarDecl}) \cup \text{follow}(\text{FuncDecl})) \cap (\text{first}(\text{FuncDecl}) \cup \text{first}(\text{VarDecl})) \\ &= \{\text{"int"}, \text{"fn"}\} \cap \{\text{"}"\} = \emptyset \end{aligned}$$

- No LL(1) conflict
- No left recursion
- Grammar is LL(1)

Task c

```
1 Var = ident ":" Type .
2 Type = ( Primitive | FuncType ) .
3 Primitive = "i32" | "f64" .
4 FuncType = Type "->" Primitive .
```

- There is left recursion. Which means that the grammar is not LL(1). Consider the following snippet. The first element of FuncType evaluates to Type, which then can evaluate to FuncType => left recursion

```
1 FuncType = Type "->" Primitive .
```

```
2 Type = ( Primitive | FuncType ) .
```

- First(Type) can also not be computed because of the left recursion. So checking the intersection of First(Type) and First(FuncType) is not possible.
- The amount of lookahead symbols does not matter in this case. The grammar is not LL(k) for any k.

02 Beseitigung von LL(1)-Konflikten mittels Faktorisierung

Beseitigen Sie in folgenden Grammatiken die enthaltenen LL(1)-Konflikte mittels Einsetzen und Faktorisierung. Geben Sie dazu eine umgeformte, ggf. vereinfachte EBNF Grammatik an.

Task a

```
1 StructDecl = ident "=" "{" { Value | NullableValue } "}" .
2 Value = ident ":" Type .
3 NullableValue = ident ":" "?" Type .
4 Type = "i32" | "i64" | "string" .
```

```
1 StructDecl = ident "=" "{" { NullableValue } "}" .
2 NullableValue = ident ":" ["?"] Type .
3 Type = "i32" | "i64" | "string" .
```

Task b

```
1 Args = FixArgs ";" VarArgs .
2 FixArgs = ident ":" Type { ";" ident ":" Type } .
3 VarArgs = "... " ident .
4 Type = "i32" | "string" .
```

```
1 Args = FixArgs VarArgs .
2 FixArgs = ident ":" Type ";" { ident ":" Type ";" } .
3 VarArgs = "... " ident .
4 Type = "i32" | "string" .
```

03 Beseitigung von Linksrekursion

Beseitigen Sie in folgender Grammatik die enthaltenen LL(1)-Konflikte. Geben Sie dazu eine umgeformte EBNF Grammatik ohne Rekursion an. Beseitigen Sie mögliche Linksrekursionen durch Umformung in eine Iteration (nicht durch Umformung in Rechtsrekursion).

```
1 Fields = Type Assigns .
2 Assigns = ident "=" Expr .
3 Expr = ident | number | Expr ( "==" | "!=" ) ( ident | number ) .
4 Type = ident | Type "->" ident .
```

First lets look at the Type Production:

```
1 Type = ident | Type "->" ident .
```

This can be rewritten as:

```
1 Type = ident { "->" ident } .
```

Second we need to address the Expr Production:

```
1 Expr = ident | number | Expr ( "==" | "!=" ) ( ident | number ) .
```

This can be rewritten as:

```
1 Expr = ( ident | number ) { ( "==" | "!=" ) ( ident | number ) } .
```

04 LL(4) Grammatik

Geben Sie eine einfache Grammatik mit mindestens 3 Produktionen an, welche die LL(4)-Eigenschaft, jedoch nicht die LL(3)-Eigenschaft besitzt.

```
1 A = B | C
2 B = "one" "two" "three" "B"
3 C = "one" "two" "three" "C"
```