Exercise 01

01 Grundbegriffe

Task a

Wie lang (in Terminalsymbolen) ist der kürzeste Satz in MicroJava, der mindestens ein Tilde (“~”) enthält? Geben Sie ein Beispiel für einen Satz dieser Länge an. Der Satz muss nur syntaktisch korrekt (entspricht der Grammatik) sein, nicht semantisch (hält alle Kontextbedingungen der Sprache ein). Bei- spielsweise ist x = x + 1, ohne x zuvor zu definieren, eine syntaktisch korrekte Zuweisung, semantisch korrekt ist sie aber nicht.

```java
program MyProgram {
  void main() {
    x[~1]++;
  }
}
```

The statement above contains the following tokens:

- “program” (literal, terminal)
- MyProgram (ident, terminal)
- “{“ (literal, terminal)
- “void” (literal, terminal)
- main (ident, terminal)
- “(“ (literal, terminal)
- “)” (literal, terminal)
- “{“ (literal, terminal)
- x (iden, terminal)
- “[“ (literal, terminal)
- “ “ (literall, terminal)
- 1 (number, terminal)
- “]” (literal, terminal)
- “++” (literal, terminal)
- “;” (literal, terminal)
- “}” (literal, terminal)
- “}” (literal, terminal)

To sum it up the statement has a ~ and has a terminal symbol count of 17.

Task b

Sind die Nonterminalsymbole Statement und ActPars rekursiv? Wenn ja, geben Sie alle Rekursionen- arten an (Transitivität: direkt, indirekt; Orientierung: links, rechts, zentral). Beispiele für Rekursionsarten sind direkt zentralrekursiv, indirekt linksrekursiv, . . . Pro NT kann es mehrere Rekursionsarten geben. Geben Sie bei indirekten Rekursionen einen möglichen Pfad an (Beispiel: Factor ist indirekt zentralre- kursiv über Factor -> Expr -> Term -> Factor).

Statement
The relevant Productions for the Statement Symbol are:

```
1  Statement = Designator ( Assignop Expr | ActPars | "++" | "--" ) ";"
2  | "if" "(" Condition ")" Statement [ "else" Statement ]
3  | "while" "(" Condition ")" Statement
4  | "break" ";"
5  | "return" [ Expr ] ";"
6  | "read" "(" Designator ")" ";"
7  | "print" "(" Expr [ "," number ] ")" ";"
8  | Block
9  | ";".
10
11 Block = "{" { Statement } "}".
```

We have the following Recursions:

- line 2: Statement is directly central recursive
- line 3: Statement is directly right recursive
- line 8: Block is indirectly central recursive

ActPars

The relevant Productions for the ActPars Symbol are:

```
1  ActPars = "(" [ Expr { "," Expr } ] ")".
2
3  Expr = [ "–" ] Term { Addop Term }.
4
5  Term = Factor { Mulop Factor }.
6
7  Factor = Designator [ ActPars ]
8  | number
9  | charConst
10 | "new" ident [ "[" Expr "]" ]
11 | "(" Expr ")".
```

We have the following Recursion:

- line 1: Factor is indirectly central recursive

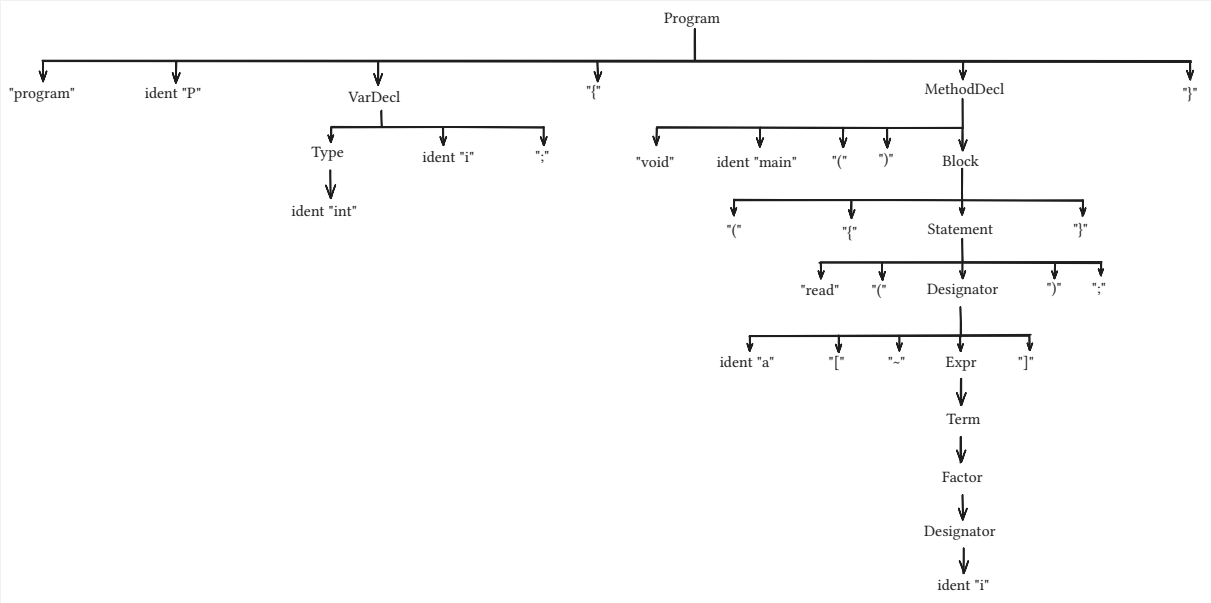
Task c

Wie sieht der konkrete Syntaxbaum für folgenden Satz aus? (Achtung, kann groß werden)

```
1 program P int i; { void main(){ read(a[~i]); }}
```
Java

- Geben Sie für Symbole der Terminalklassen ident und charConst im Baum auch die zugehörigen Werte an. Beispiel: ident (“main”)
- Sie dürfen den Syntaxbaum auch handschriftlich erstellen und abfotografieren. Achten Sie dabei bitte darauf, dass die Diagramme sauber gezeichnet und gut lesbar sind.
- Wenn Sie das Diagramm digital zeichnen wollen, dann bieten sich Zeichentools wie beispielsweise draw.io, LibreOffice Draw oder auch Microsoft PowerPoint an

Es gibt nur einen konkreten Syntaxbaum.



Task d

Welche terminalen Anfänge und Nachfolger haben die Regeln MethodDecl, AssignOp und CondTerm? Bitte die finalen Antworten komplett ausschreiben (i.e., jeweils ein Set an TS)

First Sets

- MethodDecl: { “void”, Type, “}” } = **{ “void”, ident, “}” }**
- AsignOp: **{ “=” , “+=” , “-=” , “*=” , “/=” , “%=” }**
- CondTerm: { CondFact } = { Expr } = { Term, ‘-’ }
  = { Factor, ‘-’ } = { Designator, number, charConst, “new”, “(“, “-” }
  = **{ iden, number, charConst, “new”, “(“, “-” }**

Follow Sets

- MethodDecl: { “}”, “void”, Follow(Type) } = **{ “}”, “void”, ident }**
- AsignOp: { Expr } = { “-”, Term } = { “-”, Factor }
  = { “-”, Designator, number, charConst, “new”, “(“, “-” }
  = { “-”, ident, number, charConst, “new”, “(“, “-” }
  = **{ “-”, ident, number, charConst, “new”, “(“ }**
- CondTerm: { “||”, Follow(Condition) } = **{ “||”, “)” }**

02 Konstruktion einer Grammatik

Geben Sie eine EBNF-Grammatik an, die folgende Beschreibung abdeckt. Es handelt sich dabei um eine fiktive Schachnotation. Bilden Sie dabei beliebig viele Non-Terminalklassen, mindestens jedoch eine für jedes fett geschriebene Wort. Ein Spiel setzt sich aus mindestens einem Zug zusammen, wobei Züge durch Beistriche (“,”) getrennt sind. Nach dem letzten Zug steht ein Strichpunkt (“;”). Ein Zug beginnt mit einer Figur, gefolgt von einer Position (Startposition). Darauf folgt ein Bindestrich (“-”), gefolgt von einer weiteren Position (Zielposition). Sollte durch den Zug eine Figur geschlagen worden sein, folgt ein “x”. Unabhängig davon, ob eine Figur geschlagen wurde, folgt gegebenenfalls die Information, ob der Gegner Schach (+) oder Schach-Matt (#) steht. Ein Spieler kann nicht gleichzeitig Schach und Schach-Matt stehen. Beispielzüge: Ka1-a2, Qf5-h7x#, Pb2-c3x, Rh1-h6+, Bb5-d7x+ Eine Figur ist entweder ein König (“K”), eine

Königin (“Q”), ein Turm (“R”), ein Läufer (“B”), ein Springer (“N”), oder ein Bauer (“P”). Eine Position ist eine Spalte, gefolgt von einer Zeile. Eine Spalte ist ein Buchstabe zwischen a und h (a, b, c, d, e, f, g, h). Eine Zeile ist eine Zahl zwischen 1 und 8 (1, 2, 3, 4, 5, 6, 7, 8).

```ebnf
Game = Turn { "," Turn} ";".
Turn = Character Position "-" Position ["x"] ["+" | "#"].
Character = “K” | “Q” | “R” | “B” | “N” | “P”.
Position = Column Row.
Column = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h".
Row = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8".
```

03 Beseitigung von Linksrekursionen

1. Wo befinden sich in der folgenden Grammatik Linksrekursionen? Geben Sie die Produktionen an.
2. Wie könnte man die Linksrekursionen entfernen? Geben Sie eine umgeformte EBNF Grammatik ohne Linksrekursionen an. Ersetzten Sie die Linksrekursionen dabei durch Iterationen.

```ebnf
Object = ident ":" "{" Props "}" ";".
Props = ident ":" Value | Props "," ident ":" Value .
Value = "’" text "’" | Value "+" "’" text "’" | ε .
```

Beispielsätze:

```
alice : { name: ’Alice’, greeting: ’welcome’ + ’alice’ };
bob: { name: ’Bob’, greeting: ’hello’ + ’to’ + ’bob’ };
test: { noValue: , emptyValue: ’empty’, plusValue: + ’plus’ };
```

Task 1

- line 2: Props is left recursive
- line 3: Value is left recursive

Task 2

```ebnf
Props = ident ":" Value { "," ident ":" Value }.
Value = (ε | "’" text "’") {"+" "’" text "’"}.
```