Aufgabe 3 (Syntax und Semantik): (6 + 3 + 3 = 12 Punkte)

a) Die Menge der syntaktisch korrekten **SASP** Programme wird durch die Grammatik $G_2 = (\{A, B, S_2\}, \{\texttt{.}, \texttt{:-}, \texttt{p}, \texttt{q}, \texttt{r}, \texttt{s}\}, P_2, S_2)$ definiert, wobei P_2 genau die folgenden Produktionsregeln enthält:

$$\begin{aligned} S_2 &\to A\,\texttt{.} \\ S_2 &\to A\,\texttt{.}\ S_2 \\ A &\to B \\ A &\to B\ \texttt{:-}\ B \\ B &\to \texttt{p} \\ B &\to \texttt{q} \\ B &\to \texttt{r} \\ B &\to \texttt{s} \end{aligned}$$

Die Semantik $\mathcal{W}(\mathcal{P})$ eines syntaktisch korrekten **SASP** Programms $\mathcal{P}$ ist wie folgt definiert, wobei $\mathcal{P}'$ ebenfalls ein syntaktisch korrektes **SASP** Programm ist und $x, y \in \{\texttt{p}, \texttt{q}, \texttt{r}, \texttt{s}\}$:

$$\begin{aligned} \mathcal{W}(x\,\texttt{.}) &= \{x\} \\ \mathcal{W}(x\ \texttt{:-}\ y\,\texttt{.}) &= \varnothing \\ \mathcal{W}(\mathcal{P}'\ x\,\texttt{.}) &= \mathcal{W}(\mathcal{P}') \cup \{x\} \\ \mathcal{W}(\mathcal{P}'\ x\ \texttt{:-}\ y\,\texttt{.}) &= \begin{cases} \mathcal{W}(\mathcal{P}') \cup \{x\} & \textit{falls } y \in \mathcal{W}(\mathcal{P}') \\ \mathcal{W}(\mathcal{P}') & \textit{sonst} \end{cases} \end{aligned}$$

Für alle **SASP** Programme $\mathcal{P}$ gilt also $\mathcal{W}(\mathcal{P}) \subseteq \{\texttt{p}, \texttt{q}, \texttt{r}, \texttt{s}\}$.

Geben Sie für die folgenden Ausdrücke an, ob es sich um ein syntaktisch korrektes **SASP** Programm handelt und welche Semantik es hat.

i) `t. s :- t.` ii) `s. p :- r. r. q :- r.` iii) `r` iv) `p. q. q :- r. r. s :- q.`

b) Begründen oder widerlegen Sie: Zwei Ausdrücke einer Sprache mit unterschiedlicher Semantik haben auch immer eine unterschiedliche Syntax.

c) Begründen oder widerlegen Sie: Zwei Ausdrücke einer Sprache mit unterschiedlicher Syntax haben auch immer eine unterschiedliche Semantik.

Lösung:

a) i) Das Programm ist syntaktisch nicht korrekt und hat daher keine Semantik (`t` ist kein Terminalsymbol der Grammatik G_2).

ii) Das Programm ist syntaktisch korrekt und seine Semantik ist $\{\texttt{s}, \texttt{r}, \texttt{q}\}$.

iii) Das Programm ist syntaktisch nicht korrekt und hat daher keine Semantik (der `.` fehlt am Ende).

iv) Das Programm ist syntaktisch korrekt und seine Semantik ist $\{\texttt{p}, \texttt{q}, \texttt{r}, \texttt{s}\}$.

b) Die Aussage ist wahr. Jedem syntaktisch korrekten Ausdruck wird genau eine Semantik zugeordnet. Wären die Ausdrücke syntaktisch gleich, würden sie also auch die gleiche Semantik zugeordnet bekommen.

c) Die Aussage ist falsch. Beispielsweise haben die **SASP** Programme `p.` und `p. p.` die gleiche Semantik $\{\texttt{p}\}$, unterscheiden sich jedoch in ihrer Syntax.

Aufgabe 5 (Formale Sprachen und Grammatiken): (5 + 2 + 2 = 9 Punkte)

Gegeben sei die folgende Sprache:

$L_2 = \{w \in \{a,b\}^* \mid$ auf jedes b folgen direkt mindestens drei a oder auf jedes a folgt direkt mindestens ein $b\}$

Die folgenden Wörter sind beispielsweise in der Sprache enthalten:

$$b \qquad abbbbbabb \qquad aaaabaaabaaa \qquad \varepsilon$$

Folgende Wörter sind nicht Bestandteil der Sprache:

$$aba \qquad bbbbbbbbba \qquad bababa \qquad abbaba$$

a) Geben Sie eine kontextfreie Grammatik an, welche die Sprache L_2 erzeugt.

b) Geben Sie eine Grammatik in EBNF an, die L_2 definiert. Ihre Grammatik darf nur aus einer Regel bestehen und diese Regel darf nicht rekursiv sein (d. h. das Nichtterminalsymbol auf der linken Seite darf rechts nicht auftreten).

Um die Lesbarkeit zu erhöhen, dürfen Sie Anführungszeichen um Terminalsymbole weglassen.

c) Geben Sie ein Syntaxdiagramm ohne Nichtterminalsymbole an, das die Sprache L_2 definiert.

Lösung:

a) Die kontextfreie Grammatik $G_4 = (\{S_4, A, B, X, Y\}, \{a, b\}, P_4, S_4)$ erzeugt die Sprache L_2, wobei P_4 genau die folgenden Produktionsregeln enthält:

$$\begin{aligned}
S_4 &\to AX \\
S_4 &\to BY \\
X &\to baaaAX \\
X &\to \varepsilon \\
Y &\to abBY \\
Y &\to \varepsilon \\
A &\to aA \\
A &\to \varepsilon \\
B &\to bB \\
B &\to \varepsilon
\end{aligned}$$

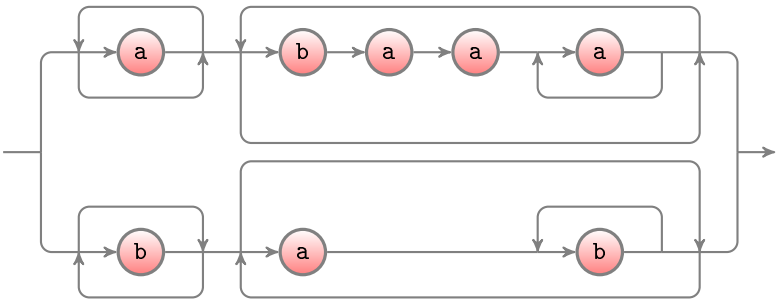
- Mit den ersten beiden Regeln entscheidet man, ob im Wort auf jedes b mindestens drei a folgen, oder ob auf jedes a mindestens ein b folgt.
- Mit X kann man nur Wörter erzeugen, in denen auf jedes b mindestens drei a folgen.
- Mit Y kann man nur Wörter erzeugen, in denen auf jedes a mindestens ein b folgt.
- Mit A und B kann man jeweils kein, ein oder mehrere a bzw. b erzeugen.

b) Die folgende Grammatik in EBNF mit nur einer nicht-rekursiven Regel definiert L_2.

$$S_4 = (\underbrace{\{a\}\{baaa\{a\}\}}_{(1)} \mid \underbrace{\{b\}\{ab\{b\}\}}_{(2)})$$

Diese Konstruktion ist analog zu der Grammatik aus Teilaufgabe a). Teil (1) entspricht dem Nichtterminalwort AX. Teil (2) entspricht dem Nichtterminalwort BY.

c) Das folgende Syntaxdiagramm definiert die Sprache L_2:



Aufgabe 7 (Zweierkomplement): (5 + 8 = 13 Punkte)

a) Welche Zahlen repräsentieren die folgenden Bitfolgen im 10-Bit Zweierkomplement?

0101110100 1010000101 1101001011 0111001100 1010010010

b) Die zwei folgenden Java-Ausdrücke werten jeweils zu `true` aus. Geben Sie dafür jeweils eine Begründung.

1) `2_147_483_000 + 648 == -(-2_147_483_648)`
2) `2_000_000_000 + 1_000_000_000 + 1_294_967_296 == 0`

Hinweise:

- In Java dürfen in einer Zahl beliebig viele `_` zwischen den Ziffern auftreten. Dies verändert den Wert der Zahl nicht, sondern es wird nur verwendet, um die Lesbarkeit der Zahl zu erhöhen. `2_000_000_000` ist also nur eine andere Schreibweise für die Zahl 2000000000.

Lösung:

a)

01110111001101011001010000000000
00111011100110101100101000000000
00000000000000000000000000000001011001011010000010111100000000

Bitfolge	10-Bit Zweierkomplement
0101110100	+372
1010000101	-379
1101001011	-181
0111001100	+460
1010010010	-366

b) 1) Der Wert auf der linken Seite der Gleichung ist `2_147_483_648`, also 2^{31}. Diese Zahl ist jedoch zu groß, um im 32-Bit Zweierkomplement dargestellt zu werden. Es tritt ein Überlauf ein, sodass die tatsächlich berechnete Zahl `-2_147_483_648` ist. Bei der Berechnung von `-(-2_147_483_648)` auf der rechten Seite wird der entsprechende Bitvektor $10\ldots0$ zunächst invertiert (was $01\ldots1$ ergibt), und dann noch 1 auf diesen Bitvektor addiert (was den Bitvektor $10\ldots0$ ergibt). Der entstandene Bitvektor ist genau derselbe, von dem wir ursprünglich ausgegangen sind ($10\ldots0$). Die Berechnung `-(-2_147_483_648)` ergibt also den Wert `-2_147_483_648`. Somit sind beide Seiten gleich.

2) Die Berechnung von `2_000_000_000 + 1_000_000_000` hat als Ergebnis `-1_294_967_296` ($= 3\,000\,000\,000 - 2^{32}$), da das Ergebnis zu groß wäre, um mit 32 Stellen dargestellt zu werden und somit ein Überlauf eintritt. Das zusätzliche Bit wird ignoriert und das restliche Ergebnis wird als Zahl im Zweierkomplement interpretiert. Bei dieser konkreten Berechnung ergibt sich eine Binärzahl, bei der das vorderste Bit eine 1 ist, weshalb das Ergebnis der Berechnung eine negative Zahl ist. Nun ist `-1_294_967_296 + 1_294_967_296 == 0` natürlich `true`.

Aufgabe 9 (Casting): (16 Punkte)

Bestimmen Sie den Typ und das Ergebnis der folgenden Java-Ausdrücke und begründen Sie Ihre Antwort. Sollte der Ausdruck nicht typkorrekt sein, begründen Sie, worin der Fehler besteht.
Dabei seien die Variablen `x`, `y` und `z` wie folgt deklariert: `int x = 1000000000; int y = 121; int z = 126;`

a) `2000000000 + x`

b) `2000000000 + 'x'`

c) `2000000000 + "x"`

d) `(byte) (3 * z) == 'z' || false`

e) `(int) 2147483648L * z`

f) `(byte) 256 * 3f`

g) `"x" + y - z`

h) `y != 'y' ? 1.0 : 'z'`

Lösung:

`int x = 1000000000; int y = 121; int z = 126;`

a) `2000000000 + x`

Der Ausdruck liefert `-1294967296` ($= 3000000000 - 2^{32}$) vom Typ `int`. Es werden zwei `int`-Werte addiert, was zu einem Überlauf mit dem genannten Ergebnis führt.

b) `2000000000 + 'x'`

Der Ausdruck liefert `2000000120` vom Typ `int`. Der `char`-Wert `'x'` wird zunächst zum `int`-Wert `120` konvertiert. Anschließend werden zwei `int`-Werte addiert, was dem genannten Ergebnis führt.

c) `2000000000 + "x"`

Der Ausdruck liefert `"2000000000x"` vom Typ `String`. Der `int`-Wert `2000000000` wird zunächst zum `String`-Wert `"2000000000"` konvertiert. Anschließend werden zwei `String`-Werte konkateniert, was zum genannten Ergebnis führt.

d) `(byte) (3 * z) == 'z' || false`

Der Ausdruck liefert `true` vom Typ `boolean`. Der Ausdruck `3 * z` wird zunächst zum `int`-Wert `378` ausgewertet. Anschließend wird dieser `int`-Wert in den `byte`-Wert `122` ($= 378 - 2^8$) konvertiert, da bei der Konvertierung ein Überlauf mit dem genannten Ergebnis stattfindet. Anschließend wird dieser `byte`-Wert für den Vergleich in den `int`-Wert `122` konvertiert. Nun wird der `char`-Wert `'z'` in den `int`-Wert `122` konvertiert. Anschließend werden die beiden `int`-Werte `122` und `122` miteinander verglichen. Da sie gleich sind, wird der `boolean`-Wert `true` erzeugt. Zuletzt werden die beiden `boolean`-Werte `true` und `false` zu `true` verknüpft. **378 = 256+64+32+16+8+2 101111010 => 01111010 => 2+8+16+32+64 = 122**

e) `(int) 2147483648L * z`

Der Ausdruck liefert `0` vom Typ `int`. Zunächst wird der `long`-Wert `2147483648` in den `int`-Wert `-2147483648` konvertiert, da bei der Konvertierung ein Überlauf mit dem genannten Ergebnis stattfindet. Anschließend liefert die Multiplikation mit dem `int`-Wert `126` das genannte Ergebnis. Wäre `z` ungerade, dann wäre das Ergebnis `-2147483648`.

f) `(byte) 256 * 3f`

Der Ausdruck liefert `0` vom Typ `float`. Zunächst wird der `int`-Wert `256` in den `byte`-Wert `0` konvertiert, da bei der Konvertierung ein Überlauf mit dem genannten Ergebnis stattfindet. Anschließend liefert die Multiplikation mit dem `float`-Wert `3` das genannte Ergebnis. **256 = 100000000 => 00000000 = 0**

g) `"x" + y - z`

Der Ausdruck liefert einen Fehler, da zunächst `"x" + y` zu einem `String`-Wert ausgewertet würde und der Operator `-` nicht für `String`-Werte definiert ist.

h) `y != 'y' ? 1.0 : 'z'`

Der Ausdruck liefert `122.0` vom Typ `double`. Zunächst wird der Ausdruck `y != 'y'` ausgewertet. Dazu wird der `char`-Wert `'y'` zum `int`-Wert `121` konvertiert. Der anschließende Vergleich mit dem `int`-Wert `y` gibt den `boolean`-Wert `false` zurück. Somit gibt der `?:`-Operator seinen `false`-Wert zurück, also `'z'`. Jedoch müssen beim `?:`-Operator beide Rückgabewerte vom selben Typ sein. Da der `true`-Wert vom Typ `double` ist wird also auch der `false`-Wert `'z'` als `double`-Wert zurückgegeben.

Aufgabe 10 (Intro, Deck 0 und Deck 1): (Codescape)

Schließen Sie das Intro und das Tutorial zum Spiel Codescape ab und lösen Sie die Missionen von Deck 0 und Deck 1.
Ihre Lösung für die Codescape Missionen wird nur dann für die Zulassung gezählt, wenn sie Ihre Lösung vor der einheitlichen Codescape Deadline am Samstag, den 22.01.2022, um 23:59 Uhr abschicken.

Lösung: