Prof. Dr. J. Giesl
D. Cloerkes, S. Dollase, N. Lommen, D. Meier, F. Meyer

Tutoraufgabe 1 (Überblickswissen):

a) Was passiert beim Kompilieren und Ausführen eines Java-Programms?

b) Was ist der Unterschied zwischen der Variable `x`, dem `char`-Wert `'x'` und dem `String`-Wert `"x"`?

c) Woran erkennt man bei einer im Zweierkomplement dargestellten Zahl, ob diese positiv oder negativ ist?

Lösung:

a) Beim Kompilieren (`javac`) wird zunächst der Java-Code (`.java`-Dateien) in Java-Bytecode (`.class`-Dateien) übersetzt. Hierbei werden, falls vorhanden, bereits Syntax-, Typ- und einige Semantik-Fehler gefunden. Anschließend kann der Java-Bytecode mithilfe der Java Virtual Machine (`java`) ausgeführt werden. Diese sorgt dafür, dass der plattformunabhängige Java-Bytecode auf der aktuell vorhandenen Hardware korrekt ausgeführt wird.

Läuft ein Java-Programm für eine längere Zeitspanne ohne Unterbrechung, so sorgt der Just-In-Time-Compiler (JIT-Compiler), welcher ein Teil der Java Virtual Machine ist, dafür, dass die häufig ausgeführten Teile des Programms auf der aktuellen Hardware performanter ausgeführt werden.

b) Eine Variable wird bei der Auswertung durch den ihr zugewiesenen Wert ersetzt. Der Name der Variable ist hingegen irrelevant, solange sie an allen Stellen gleich benannt wird.

Der `char`-Wert `'x'` stellt nur ein einzelnes Zeichen dar. Ein `char`-Wert kann auch immer nur ein einzelnes Zeichen darstellen. So produziert der Ausdruck `'xy'` beispielsweise einen Syntaxfehler, da versucht wird, mehr als ein Zeichen in einem `char`-Wert zu speichern.

Der `String`-Wert `"x"` stellt ebenfalls ein einzelnes Zeichen dar, jedoch könnte ein `String` beliebig viele Zeichen enthalten. Der Ausdruck `"xy"` stellt also ebenfalls einen gültigen Ausdruck dar und führt nicht zu einem Syntaxfehler.

Der Unterschied zwischen `char` und `String` ist auch im Typsystem fest verankert. So ist es weder möglich, einer `char`-Variablen einen `String`-Wert zuzuweisen, noch ist dies umgekehrt möglich. Um einer `String`-Variablen einen `char`-Wert zuzuweisen, kann jedoch beispielsweise der `char`-Wert zunächst mit dem leeren `String` verkettet werden, sodass der `char`-Wert in einen `String`-Wert konvertiert wird: `String text = 'x' + "";`

c) Im Zweierkomplement ist das erste Bit das Vorzeichenbit. Ist es `0`, so ist die Zahl positiv (oder 0). Ist es `1`, so ist die Zahl negativ.

Tutoraufgabe 2 (Syntax und Semantik):

a) Die Menge der syntaktisch korrekten einfachen arithmetischen Ausdrücke (**EAA**) wird durch die Grammatik $G_1 = (\{S_1\}, \{(,), ;, \mathsf{plus}, \mathsf{s}, \mathcal{O}\}, P_1, S_1)$ definiert, wobei P_1 genau die folgenden Produktionsregeln enthält:

$$\begin{array}{lcl} S_1 & \to & \mathcal{O} \\ S_1 & \to & \mathsf{s}(S_1) \\ S_1 & \to & \mathsf{plus}(S_1; S_1) \end{array}$$

Die Semantik $\mathcal{W}(\mathcal{A})$ eines syntaktisch korrekten **EAA**s $\mathcal{A}$ ist wie folgt definiert, wobei x und y ebenfalls syntaktisch korrekte **EAA**s sind:

$$\begin{array}{rcl} \mathcal{W}(\mathcal{O}) & = & 0 \\ \mathcal{W}(\mathsf{s}(x)) & = & \mathcal{W}(x) + 1 \\ \mathcal{W}(\mathsf{plus}(x; y)) & = & \mathcal{W}(x) + \mathcal{W}(y) \end{array}$$

Für alle **EAA**s $\mathcal{A}$ gilt also $\mathcal{W}(\mathcal{A}) \in \mathbb{N}$.

Geben Sie für die folgenden drei Ausdrücke an, ob es sich um einen syntaktisch korrekten **EAA** handelt und welche Semantik er hat.

i) $\mathsf{plus}(\mathsf{s}(\mathcal{O}); \mathcal{O})$ ii) $\mathsf{plus}(\mathcal{O}; \mathsf{s}(\mathcal{O}); \mathcal{O})$ iii) $\mathsf{plus}(\mathsf{plus}(\mathcal{O}; \mathsf{s}(\mathcal{O})); \mathsf{s}(\mathsf{plus}(\mathcal{O}; \mathcal{O})))$

b) Begründen oder widerlegen Sie: Zwei Ausdrücke mit gleicher Syntax haben auch die gleiche Semantik.

c) Begründen oder widerlegen Sie: Ein syntaktisch korrektes Programm ist auch semantisch korrekt.

Lösung:

a) i) Der Ausdruck ist syntaktisch korrekt und seine Semantik ist 1.
ii) Der Ausdruck ist syntaktisch nicht korrekt und er hat daher keine Semantik.
iii) Der Ausdruck ist syntaktisch korrekt und seine Semantik ist 2.

b) Die Aussage ist falsch. Betrachten wir den (nicht einfachen) arithmetischen Ausdruck `1 / 2`. In der Sprache Java wird dieser Ausdruck zu 0 ausgewertet, da Java bei der Division zweier ganzer Zahlen die Ganzzahldivision ohne Rest verwendet. In der Sprache Prolog wird dieser Ausdruck hingegen zu 0.5 ausgewertet, da Prolog für den Operator / grundsätzlich Gleitkommadivision (oder Fließkommadivision) verwendet. Dieses Gegenbeispiel widerlegt die Aussage.

c) Die Aussage ist falsch. Ein Programm ist semantisch korrekt, wenn es genau die Anforderungen erfüllt, für die es entwickelt wurde. Lautet die Anforderung, dass das Programm die ersten eintausend Primzahlen ausgeben soll, so ist ein syntaktisch korrektes Programm, welches stattdessen "`Hello world!`" ausgibt, nicht semantisch korrekt. Dieses Gegenbeispiel widerlegt die Aussage.

Tutoraufgabe 4 (Formale Sprachen und Grammatiken):

Gegeben sei die folgende Sprache:

$$L_1 = \{w \in \{a, b\}^* \mid \text{ auf ein } a \text{ folgt nie ein } b \text{ oder auf ein } b \text{ folgt nie ein } a\}$$

Die folgenden Wörter sind beispielsweise in der Sprache enthalten:

$$aaab \qquad bbaa \qquad aa \qquad \varepsilon$$

Folgende Wörter sind nicht Bestandteil der Sprache:

$$bab \qquad abba \qquad baba$$

a) Geben Sie eine kontextfreie Grammatik an, welche die Sprache L_1 erzeugt.

b) Geben Sie eine Grammatik in EBNF an, die L_1 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_1 definiert.

Lösung:

a) Die kontextfreie Grammatik $G_3 = (\{S_3, A, B\}, \{a, b\}, P_3, S_3)$ erzeugt die Sprache L_1, wobei P_3 genau die folgenden Produktionsregeln enthält:

$$\begin{aligned} S_3 &\to AB \\ S_3 &\to BA \\ A &\to aA \\ A &\to \varepsilon \\ B &\to bB \\ B &\to \varepsilon \end{aligned}$$

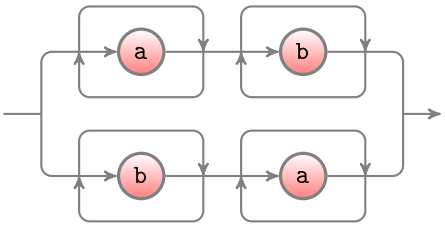
Mit dem Nonterminal S_3 wählt man, ob man zuerst a oder b Symbole erzeugen will. Das Nonterminal A erzeugt Wörter bestehend aus beliebig vielen a Symbolen, während B Wörter aus beliebig vielen b Symbolen erzeugt.

b) Die folgende Grammatik in EBNF mit nur einer nicht-rekursiven Regel definiert L_1.

$$S_3 = (\underbrace{\{a\}\{b\}}_{(1)} \mid \underbrace{\{b\}\{a\}}_{(2)})$$

Diese Konstruktion ist analog zu der Grammatik aus Teilaufgabe a). In Teil (1) wird ein Wort erzeugt, bei dem beliebig viele a Symbole vor beliebig vielen b Symbolen stehen. In Teil (2) wird ein Wort erzeugt, bei dem beliebig viele b Symbole vor beliebig vielen a Symbolen stehen.

c) Das folgende Syntaxdiagramm definiert die Sprache L_1:



Tutoraufgabe 6 (Zweierkomplement):

a) Sei x eine ganze Zahl. Wie unterscheiden sich die Zweierkomplement-Darstellungen von x und $-x$?

b) Erklären Sie im Detail, wie die beiden Ausgaben des folgenden Programms berechnet werden.

```
public class Test {
    public static void main(String[] args) {
        int zahl = -2147483648;

        System.out.println(zahl + 1);
        System.out.println(zahl - 1);
    }
}
```

Hinweis: $-2^{31} = -2147483648$

c) Welche Zahlen repräsentieren die folgenden Bitfolgen im 5-Bit Zweierkomplement?

00010 10111 11011 01101 10000

Lösung:

a) Ausgehend von der Zweierkomplement-Darstellung von x erreicht man durch die folgenden beiden Schritte die Zweierkomplement-Darstellung von $-x$:

a) vertausche alle 0en und 1en

b) addiere 1

Mit diesen beiden Schritten ist auch die Rückrichtung ($-x$ zu x) möglich.

In der folgenden Tabelle finden Sie alle Binärzahlen mit drei Ziffern. Man erkennt das Muster, nach dem das genannte Verfahren funktioniert.

3	011
2	010
1	001
0	000
-1	111
-2	110
-3	101
-4	100

b) Im Folgenden werden Binärzahlen mit einem `Z` markiert, wenn die Zahl im Zweierkomplement verstanden werden muss. Die Zahl 1111`Z` ist also als -1 zu verstehen, während 1111 für die Zahl 15 steht.

Der Datentyp `int` benutzt 32 Bit. Die Darstellung der Zahl -2147483648 im Zweierkomplement ist:

```
10000000000000000000000000000000 Z (31 Nullen)
```

Das Ergebnis der Addition `zahl + 1` berechnet sich wie folgt:

```
10000000000000000000000000000000 Z
00000000000000000000000000000001 Z
--------------------------------
10000000000000000000000000000001 Z
```

Auch hier gibt die führende 1 an, dass die dargestellte Zahl negativ ist. Den Dezimalwert der dargestellten Zahl erhält man durch Invertieren und Addieren von 1:

```
01111111111111111111111111111110
00000000000000000000000000000001
--------------------------------
01111111111111111111111111111111
```

Dies steht für 2147483647. Mit der Vorzeicheninformation von oben ergibt sich -2147483647.

Berechnet man `zahl - 1`, berechnet sich das Ergebnis durch die Addition mit -1. Die Zahl -1 ist im Zweierkomplement dargestellt durch:

```
11111111111111111111111111111111 Z
```

Die Addition -2147483648 + (-1) ergibt demzufolge:

```
10000000000000000000000000000000 Z
11111111111111111111111111111111 Z
--------------------------------
01111111111111111111111111111111 Z
```

Das Ergebnis ist also nicht negativ (erkennbar durch die führende 0) und entspricht der Dezimalzahl +2147483647. Dieses Ergebnis wird auch durch das Java-Programm ausgegeben.

c)

Bitfolge	5-Bit Zweierkomplement
00010	2
10111	-9
11011	-5
01101	13
10000	-16

Tutoraufgabe 8 (Casting):

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 = 1; int y = 2; int z = 3;`

a) `false && true`

b) `10 / 3`

c) `10 / 3.`

d) `x == y ? x > y : y < z`

e) `(byte) (127 + 1)`

f) `'x' + y + z`

g) `x + y + "z"`

h) `1 || 0`

Lösung:

```
int x = 1; int y = 2; int z = 3;
```

a) `false && true`
Der Ausdruck liefert den Wert `false` vom Typ `boolean`, da die logische Und-Verknüpfung zweier `boolean` Werte hier ganz normal ausgeführt werden kann.

b) `10 / 3`
Der Ausdruck liefert den `int`-Wert `3`, da bei der Division zweier `int`-Werte in Java Ganzzahldivision ohne Rest verwendet wird.

c) `10 / 3.`
Der Ausdruck liefert den `double`-Wert `3.3333333333333335`, da der `int`-Wert `10` für die `double`-Division erst zu `double` konvertiert wird.

d) `x == y ? x > y : y < z`
Der Ausdruck liefert den `boolean`-Wert `true`, da zuerst der `boolean`-Vergleich `x == y` zu `false` und anschließend `y < z` zu `true` ausgewertet wird. Der Typ von `x > y` ist ebenfalls `boolean`, weshalb kein Fehler auftritt.

e) `(byte) (127 + 1)`
Der Ausdruck liefert das Ergebnis `-128`, da zuerst die `int`-Addition durchgeführt wird und das Ergebnis `+128` anschließend in den `byte`-Datentypen konvertiert wird. Dieser Datentyp kann diesen Wert allerdings nicht darstellen. Daher werden nur die letzten 8 Bit berücksichtigt (alle zusätzlichen Bits werden also „abgeschnitten“; dies wird auch „Overflow“ oder auf Deutsch Überlauf genannt).

f) `'x' + y + z`
Durch die Auswertung von links nach rechts wird zuerst `'x' + y` ausgewertet. Dafür wird das Zeichen `'x'` zuerst in die `int`-Zahl 120 konvertiert. Dies ergibt also `120 + 2 = 122`. Dieser Wert wird dann mit der `int`-Zahl `3` addiert, und somit wird der Gesamtausdruck zu `125` vom Typ `int` ausgewertet.

g) `x + y + "z"`
Durch die Auswertung von links nach rechts wird zuerst `x + y` zur `int`-Zahl `3` ausgewertet. Dieser Wert wird dann mit dem `String` `"z"` verkettet, und somit wird der Gesamtausdruck zu `"3z"` vom Typ `String` ausgewertet.

h) `1 || 0`
Der Ausdruck liefert einen Fehler, da `1` und `0` vom Typ `int` sind und damit die `boolean`-Verknüpfung nicht möglich ist.