Hochschule Deggendorf Dr. Peter Jüttner	
Vorlesung: Grundlagen der Informatik	WS 2012
Übung 14	Termin 15.1.13

Wiederholung - Musterlösung

1. Addieren Sie die Binärzahlen 101100101011 und 101110110111

```
 101100101011
+101110110111
  11  111111
1011011100010
```

2. Wandeln Sie die folgenden Festkommazahlen im Dezimalsystem in die entsprechende Darstellung im Binärsystem um.

   a.) 3,25

   3 entspricht binär 11 und 0,25 entspricht 0,01, d.h. $3{,}25_{10} = 11{,}01_2$

   b.) 4,75

   4 entspricht binär 100, 0,75 = 0,5 + 0,25, d.h. $4{,}75_{10} = 100{,}11_2$

   c.) 1,5

   1 entspricht binär 1, 0,5 entspricht binär 0,1, d.h. $1{,}5_{10} = 1{,}1_2$

3. Wandeln Sie die folgenden, binär dargestellten Festkommazahlen in die entsprechende Dezimaldarstellung um.

   a.) 1,01

   $1{,}01_2 = 1_{10} + 0{,}25_{10} = 1{,}25_{10}$

   b.) 101,1101

   $101{,}1101_2 = 5_{10} + 0{,}5_{10} + 0{,}25_{10} + 0{,}0625_{10} = 5{,}8125_{10}$

   c.) 10,0101

   $10{,}0101_2 = 2_{10} + 0{,}25_{10} + 0{,}0625_{10} = 2{,}3125_{10}$

4. Schreiben Sie ein C-Programm, das feststellt, wie viele 1er und 0er die Binärdarstellung einer Zahl vom Datentyp unsigned short (Datenlänge 2 Byte) hat. Die Zahl soll interaktiv eingelesen werden.

   Lösungsweg:
   Einen Zähler für Einsen definieren und mit 0 initialisieren, Die Zahl so lange durch zwei dividieren und bei Rest 1 den Einsenzähler um 1 erhöhen, bis 0 erreicht ist. Die Anzahl der Nullen ergibt sich aus 16-Anzahl der Einsen.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{ unsigned short s;
  unsigned short hilf;
  int zaehlereins = 0;
  printf("Bitte Zahl eingeben\n");
  scanf("%hu", &s);
  hilf = s;
  while (s != 0)
  { if ((s%2)==1)
       zaehlereins++;
    s = s/2;
  };
  printf("Die Zahl %hu enthaelt in Binaerdarstellung %d 1er und %d
0er\n",hilf,zaehlereins,16-zaehlereins);

  system("PAUSE");
  return 0;
}
```

   Alternative:
   Die Zahl wir bitweise mit einer Maske „verundet“, die alle Bits bis auf eins ausblendet. Ist die Zahl nach der Maskierung ungleich 0, wird die Anzahl der 1er erhöht.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{ short s;
  unsigned short maske = 0x8000;
  int zaehler = 0;
  printf("Bitte Zahl eingeben\n");
  scanf("%d", &s);
  for (int i=0; i<16; i++)
  { if ((s & maske) != 0)
      zaehler++;
    maske = maske >> 1;
  };
  printf("Die Zahl %d enthaelt in Binaerdarstellung %d 1er und %d
0er\n",s,zaehler,16-zaehler);
```

```
  system("PAUSE");
  return 0;
}
```

5. Was wird bei folgenden Programmteilen ausgegeben?
(Hinweis: die Formatangabe %h... bewirkt, dass der auszugebende Ausdruck als short interpretiert wird, z.B. “%hd“ Ausgabe als 2Byte Ganzzahl

a.)
```
short c;
c = 0x7FFF;
printf(“%hd“, c<<1);
```

Ausgabe ist -2, da 0x7FFF = 0111111111111111_2. Durch den Linksshift um 1 entsteht 1111111111111110_2. Interpretiert im 2er-Komplement ergibt dies -2. (Herleitung: $-1_{10} = 1111111111111111_2$, 1 abgezogen ergibt -2_{10} bzw. 1111111111111110_2.
Alternative Herleitung: $-(1111111111111110_2)$ ergibt im 2er-Komplement 0000000000000001 (Invertieren), dann 1 addieren, somit erhält man $0000000000000020_2 = 2_{10}$. Also ist das Negaitve von 1111111111111110_2 gleich 2_{10}, somit ist die Zahl -2_{10}.)

b.)
```
short c;
c = 0x7FFF ;
printf(“%hu“, c<<1);
```

Ausgabe ist 65534, da 0x7FFF = 0111111111111111_2. Durch den Linksshift um 1 entsteht 1111111111111110_2. Die Zahl wird vorzeichenlos interpretiert, d.h. der Wert ist $(2^{16}-1) -1 = 65534$

c.)
```
short c;
c = -1 ;
printf(“%hu“, c<<1);
```

Ausgabe ist 65534, da -1 = 1111111111111111_2. Durch den Linksshift wird eine 0 rechts nachgezogen, die Binärdarstellung ist jetzt 1111111111111110_2. Vorzeichenlos interpretiert (Formatangabe “%hu“) ergibt 65534

d.)
```
short c ;
c = -1 ;
printf(“%hd“, c<<1) ;
```

Ausgabe ist -2, da -1 = 1111111111111111_2. Durch den Linksshift wird eine 0 rechts nachgezogen, die Binärdarstellung ist 1111111111111110_2. Vorzeichenbehaftet interpretiert (Formatangabe “%hd“) ergibt -2

e.)
```
short c ;
c = -1 ;
printf(“%hd“, c>>1) ;
```

Ausgabe ist -1, da -1 = 1111111111111111_2. Durch den Rechtsshift wird eine 1 links nachgezogen, die Binärdarstellung bleibt 1111111111111111_2. Vorzeichenbehaftet interpretiert (Formatangabe "%hd") ergibt -1

Zeichencodierungen

6. Was wird in folgenden Programmteilen ausgegeben?
   Hinweis: Nutzen Sie die ASCII-Tabelle hinten

   a.) char c;
   c = 'x';
   printf("%c\n",c);

   Ausgabe ist x, da die Variable als Buchstabe ausgegeben wird.

   b.) char c;
   c = 'x';
   printf("%d\n",c);

   Ausgabe ist 120, da die Variable als Zahl (ASCII Wert) ausgegeben wird.

   c.) unsigned char c;
   c = 'X' + 'Y';
   printf("%u\n", c);

   Ausgabe ist -79, da 'X' + 'Y' der Addition der ASCII-Wert (88, 89) entspricht. 88+89 = 177. Die Binärdarstellung von 177 ist 10110001_2. Dies entspricht, interpretiert als 1 Byte Zahl im Zweierkomplement -79

   d.) char c;
   c = '1' + '2';
   printf("%c\n", c);
   Ausgabe ist c, da '1' + '2' der Addition der ASCII-Wert der Ziffern 1 und 2 (49, 50) entspricht. 49 + 50 = 99. 99 ist der ASCII Code des Buchstaben c.

   e.) char c;
   c = 'a';
   c++;
   printf("%c\n",c);

   Ausgabe ist b, da c++ den Wert der Variablen c um 1 erhöht. Die ASCII werte der Buchstaben sind aufsteigend.

7. Schreiben Sie eine C-Funktion, die einen Buchstaben (char) als Parameter hat und den ASCII_Wert als Integer zurückgibt.

```
int ascii_wert(char c)
 {
```

```
    return c;
}
```

Aussagenlogik

8. Stellen Sie die Wahrheitstabelle folgender aussagenlogischer Formeln auf

a.) $(A \vee B) \wedge C$

A	B	C	$(A \vee B) \wedge C$
F	F	F	F
F	F	T	F
F	T	T	T
T	T	T	T
T	T	F	F
T	F	F	F
T	F	T	T
F	T	F	F

b.) $(A \Rightarrow B) \wedge C$

A	B	C	$(A \Rightarrow B) \wedge C$
F	F	F	F
F	F	T	T
F	T	T	T
T	T	T	T
T	T	F	F
T	F	F	F
T	F	T	F
F	T	F	F

c.) $(A \wedge (B \Rightarrow C))$

A	B	C	$(A \wedge (B \Rightarrow C))$
F	F	F	F
F	F	T	F
F	T	T	F
T	T	T	T
T	T	F	F
T	F	F	T
T	F	T	T
F	T	F	F

d.) $(A \Leftrightarrow B) \Rightarrow C$

A	B	C	$(A \Leftrightarrow B)$

			⇒C
F	F	F	F
F	F	T	T
F	T	T	T
T	T	T	T
T	T	F	F
T	F	F	T
T	F	T	T
F	T	F	T

9. Stellen Sie fest, ob die folgenden aussagenlogischen Formeln Tautologien sind. Begründen Sie Ihre Aussage:

   a.) $(A \wedge B) \Rightarrow (B \wedge A)$

   Es liegt eine Tautologie vor, da auf beiden Seiten der Implikation der gleiche Wert steht und es gilt T⇒T und F⇒F

   b.) $(A \Rightarrow B) \vee (B \Rightarrow A)$

   Es liegt eine Tautologie vor da immer mindestens eine der beiden Seiten der Oder-Verknüpfung wahr wird.

10. Stellen Sie fest ob folgende Aussage Tautologien sind. Benutzen Sie dazu keine Wahrheitstabelle der kompletten Aussage. Begründen Sie Ihre Aussage

    a.) $(A \wedge B) \vee C \Rightarrow \neg ((\neg A \vee \neg B) \wedge \neg C)$

    Es liegt eine Tautologie vor, da auf der rechten Seite der Implikation die doppelt negierte Formel der rechten Seite steht. Die doppelte Negation liefert die ursprüngliche Formel, d.h. auf beiden Seiten der Implikation steht für alle Belegungen der gleiche Wert.

    b.) $((A \vee B) \wedge (C \vee D)) \Leftrightarrow ((\neg C \wedge \neg D) \wedge (A \vee \neg\neg B))$

    Es liegt keine Tautologie vor, da jede Belegung auf beiden Seiten der Äquivalenz zu unterschiedliche Ergebnissen führt (((¬C∧¬D) ist die Negation von (C∨D)), somit wird die Äquivalenz immer falsch.

11. Schreiben Sie ein C-Programm, das drei Wahrheitswerte A, B und C einliest, mit folgenden aussagenlogischen Operatoren verknüpft und das Ergebnis ausgibt:

    a.) $(A \Rightarrow B) \Rightarrow \neg C$

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
  int a=2, b=2, c=2, ergebnis = 2;
```

```
  while ((a != 0) && (a != 1))
  {
    printf("Bitte Wert fuer A eingeben (0 fuer Falsch, 1 fuer Wahr):\n");
    scanf("%d",&a);
    if ((a != 0) && (a != 1))
      printf("Falsche Eingabe - Nochmal !\n");
  }
  while ((b != 0) && (b != 1))
  {
    printf("Bitte Wert fuer B eingeben (0 fuer Falsch, 1 fuer Wahr):\n");
    scanf("%d",&b);
    if ((b != 0) && (b != 1))
      printf("Falsche Eingabe - Nochmal !\n");
  }
  while ((c != 0) && (c != 1))
  {
    printf("Bitte Wert fuer C eingeben (0 fuer Falsch, 1 fuer Wahr):\n");
    scanf("%d",&c);
    if ((c != 0) && (c != 1))
      printf("Falsche Eingabe - Nochmal !\n");
  }

  if (a)
  { /* a wahr */
    if (b)
    { /* b wahr */
      ergebnis = !c;
    }
    else /* b falsch */
    { /* b falsch */
      ergebnis = 1;
    }
  }
  else
  { /* a falsch */
    ergebnis = !c;
  }
  if (ergebnis)
    printf("Ausdruck wahr\n");
  else printf("Ausdruck falsch\n");

  system("PAUSE");
  return 0;
}
```

b.) $(A \Leftrightarrow B) \Rightarrow \neg(A \Rightarrow C)$

```
... Einlesen der Werte für a, b, c, wie oben ...

  if (((a) && (b)) || ((!a) && (!b)))
  { /* a <=> b wahr */
```

```
    ergebnis = a && !c;
  }
  else ergebnis = 1;

... Ausgabe ergebnis wie oben ...

  system("PAUSE");
  return 0;
}
```

12. Überführen Sie die folgende Aussagenlogische Formel in die konjunktive Normalform:

a.) $(A \Leftrightarrow B) \Rightarrow \neg C$

Vorgehensweise: Aufstellen der Wahrheitstabelle), alle Kombinationen, die False liefern werden „verundet“ (mit und verknüpft). Die entsprechenden Parameter werden „verodert“ (mit oder verknüpft) in negierter Form, falls Wert = False. Danach wird der gesamte Ausdruck negiert.

A	**B**	**C**	**$(A \Leftrightarrow B) \Rightarrow \neg C$**
F	F	F	T
F	F	T	**F**
F	T	T	T
T	T	T	**F**
T	T	F	T
T	F	F	T
T	F	T	T
F	T	F	T

KNF:
$\neg ((\neg A \wedge \neg B \wedge C) \vee (A \wedge B \wedge C)) = \neg (\neg A \wedge \neg B \wedge C) \wedge \neg (A \wedge B \wedge C) =$
$(A \vee B \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg C))$

b.) $(A \wedge B) \Leftrightarrow (B \Rightarrow \neg C)$

A	**B**	**C**	**$(A \wedge B) \Leftrightarrow (B \Rightarrow \neg C)$**
F	F	F	**F**
F	F	T	**F**
F	T	T	T
T	T	T	**F**
T	T	F	T
T	F	F	**F**
T	F	T	**F**

F	T	F	**F**

KNF:
$\neg ((\neg A \wedge \neg B \wedge \neg C) \vee (\neg A \wedge \neg B \wedge C) \vee (A \wedge B \wedge C) \vee (A \wedge \neg B \wedge \neg C) \vee$
$(A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge \neg C)) =$
$\neg(\neg A \wedge \neg B \wedge \neg C) \wedge \neg(\neg A \wedge \neg B \wedge C) \wedge \neg(A \wedge B \wedge C) \wedge \neg(A \wedge \neg B \wedge \neg C) \wedge$
$\neg(A \wedge \neg B \wedge C) \wedge \neg(\neg A \wedge B \wedge \neg C) =$
$(A \vee B \vee C) \wedge (A \vee B \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee B \vee C) \wedge$
$(\neg A \vee B \vee \neg C) \wedge (A \vee \neg B \vee C)$

13. Überführen Sie die folgende Aussagenlogische Formel in die disjunktive Normalform:

Vorgehensweise: Aufstellen der Wahrheitstabelle), alle Kombinationen, die True liefern werden „verundet“ (mit und verknüpft). Die entsprechenden Parameter werden „verodert“ (mit oder verknüpft) in negierter Form, falls Wert = False.
.

a.) $(A \Rightarrow B) \Rightarrow (\neg A \Rightarrow C)$

A	**B**	**C**	**$(A \Rightarrow B) \Rightarrow (\neg A \Rightarrow C)$**
F	F	F	F
F	F	T	**T**
F	T	T	**T**
T	T	T	**T**
T	T	F	**T**
T	F	F	**T**
T	F	T	**T**
F	T	F	F

DNF:
$(\neg A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge C) \vee (A \wedge B \wedge C) \vee (A \wedge B \wedge \neg C)$
$\vee (A \wedge \neg B \wedge \neg C) \vee (A \wedge \neg B \wedge C))$

b.) $(A \vee \neg B) \Rightarrow (C \Leftrightarrow \neg A)$

A	**B**	**C**	**$(A \vee \neg B) \Rightarrow (C \Leftrightarrow \neg A)$**
F	F	F	F
F	F	T	**T**
F	T	T	**T**
T	T	T	F
T	T	F	**T**
T	F	F	**T**
T	F	T	F
F	T	F	**T**

DNF:
$(\neg A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge C) \vee (A \wedge B \wedge \neg C) \vee (A \wedge \neg B \wedge \neg C)$
$\vee (\neg A \wedge B \wedge \neg C)$

Prädikatenlogik

14. Formulieren Sie folgende Aussagen in prädikatenlogischer Form, Benutzen Sie dabei ein- und zwei-stellige Prädikate wie können fliegen(x), ist kleiner als(x,y),

    a.) Nicht alle Vögel können fliegen

    M Menge aller Vögel
    $\neg (\forall x \in M$ (kann fliegen(x)))

    b.) Alle Vögel haben Federn

    M Menge aller Vögel
    $\forall x \in M$(hat Federn(x))

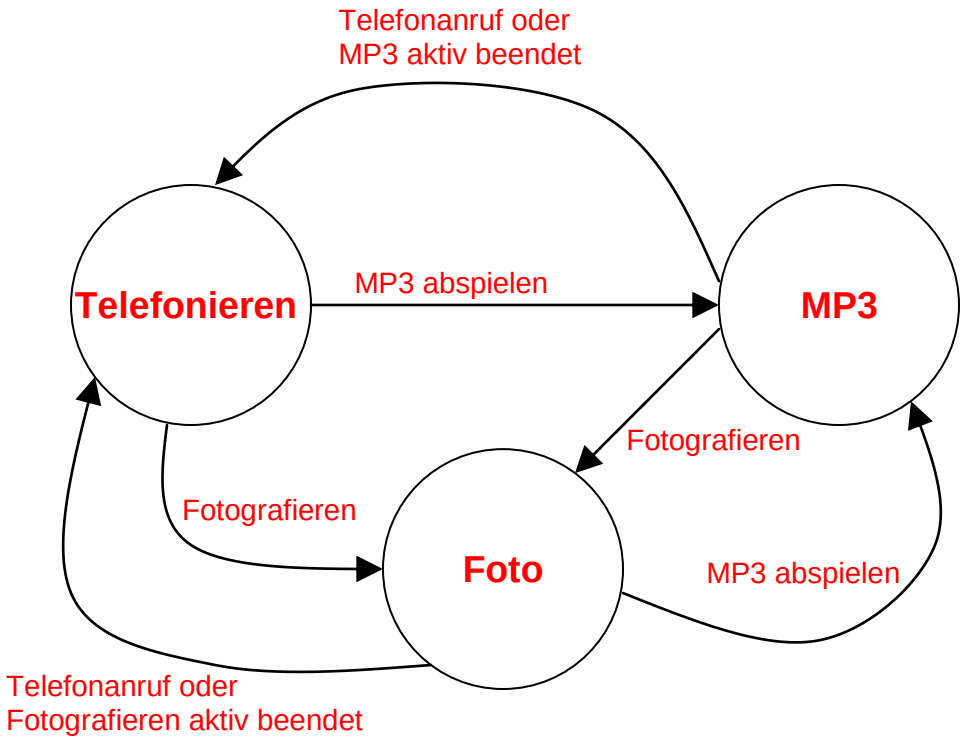
    c.) Nicht alle Zahlen sind kleiner als 100

    M Menge aller (reellen) Zahlen
    $\neg (\forall x \in R$ (x<100))

Endliche Automaten

15. Ein Handy kann entweder telefonieren, als MP3-Spieler fungieren oder Fotografieren. Alle Funktionen werden per Menü am Display über Tasten aktiviert und wieder beendet.
    Sowohl das Abspielen von MP3 als auch das Fotografieren können zusätzlich jederzeit von einem ankommenden Telefonanruf automatisch unterbrochen und beendet werden. Geben Sie eine Grafische Darstellung des Automaten mit den entsprechenden Zuständen, Ereignissen und Aktionen an.

    Lösung: Vereinfachend wird angenommen, dass Telefonieren einen Zustand bezeichnet, bei der gerade ein Anruf aktiv ist oder das Handy auf einen Anruf (ankommend oder abgehend wartet)



Syntaxbeschreibungen

16. Gegeben sei die folgende Grammatik (N, T, P, S) mit
    - Nichtterminalsymbolen N := {A, B, S}
    - Terminalsymbole T := { +, * x}
    - Produktionen := { S ::= A | S + A, A ::= B | A + B, B ::= x | x * B }
    - Startsymbol S := {S}

    a. Geben Sie den kürzesten Text (Satz), der mittels dieser Grammatik erzeigt werden kann, und seine Ableitung an.

    Der kürzeste Text ist x. Dies ergibt sich aus S ::= A, A::= B, B ::= x

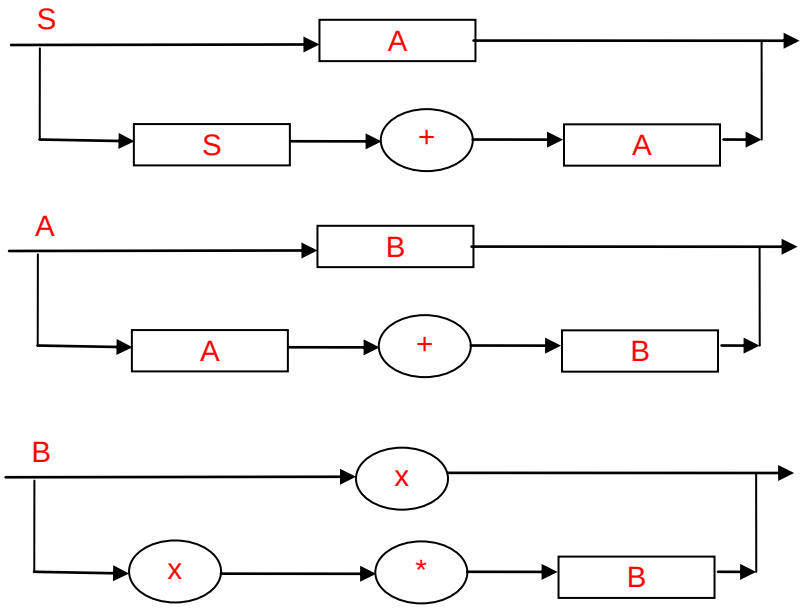
    b. Geben Sie einen Text und seine Ableitung an, der genau ein + und ein * enthält.

    Text x*x+x, Ableitung S ::= A, A ::= S+A, S ::= B, B::= x, A ::= B, B ::= x * B, B ::= x

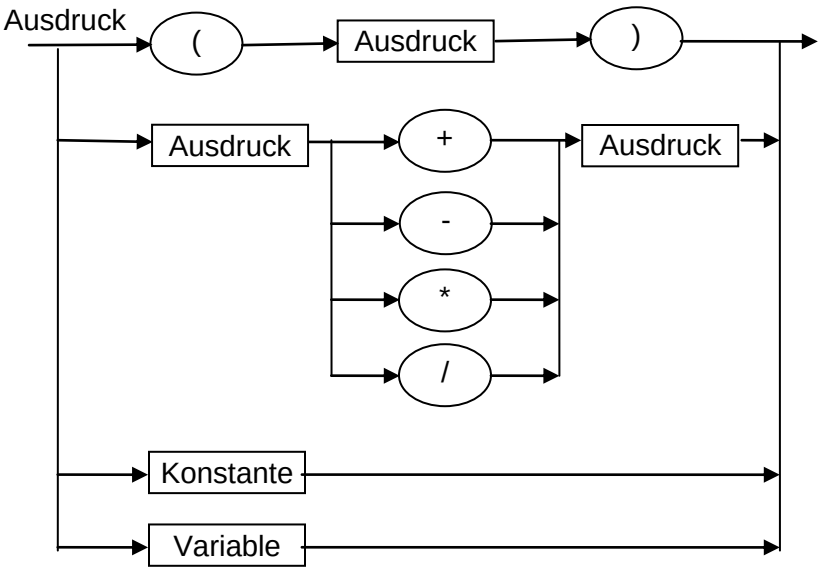
c. Geben Sie eine Ableitung für folgenden Satz an: x+x+x+x+x*x

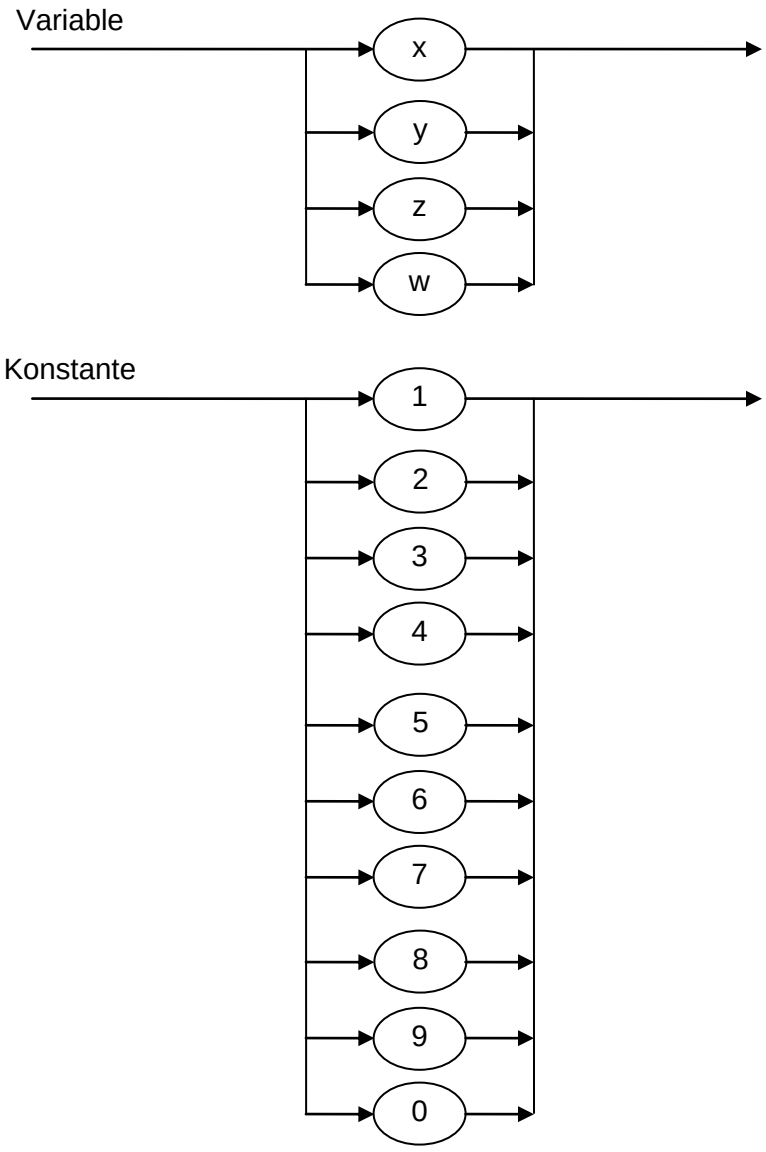
S ::= S + A (mehrfach angewendet ergibt S + A + A + A + A, S wird zu A durch S ::= A, dadurch entsteht A+A+A+A+A, Die ersten vier A werden durch A ::= B und B ::= x zu x+x+x+x+ das letzte A wird durch A ::= B, B::= x*B und B::= x zum gewünschten Ergebnis.

d. Zeichen Sie die Syntaxdiagramme, die den oben angegebenen Produktionen entsprechen.



17. Gegeben sind die folgenden Syntaxdiagramme für einen Ausdruck:


Ausdruck
(
Ausdruck
)
Ausdruck
+
Ausdruck
-
*
/
Konstante
Variable



a. Geben Sie die den Syntaxdiagrammen entsprechende Grammtik (N, T, P, S) an (Startsymbol ist Ausdruck)

N = { Ausdruck, Variable, Konstante }
T = { (,), *, /, +, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, x, y, z, w }
P = { Ausdruck ::= (Ausdruck) | Konstante | Variable |
Ausdruck + Ausdruck | Ausdruck * Ausdruck |
Ausdruck / Ausdruck | Ausdruck – Ausdruck
Konstante ::= 1|2|3|4|5|6|7|8|9|0
Variable ::= x|y|z|w }
S = Ausdruck

b. Stellen Sie fest, ob und ggf. wie sich die folgenden Zeichenketten als Ausdruck ableiten lassen:

- x+5

  x+5 läßt sich ableiten aus Ausdruck ::= Ausdruck + Ausdruck, Ausdruck ::= Varianble, Ausdruck ::= Konstante, Variable ::= x, Konstante ::= 5

- (x+y)*7

  Analog zu oben läßt sich aus Ausdruck ::= (Ausdruck) und Ausdruck ::= Ausdruck * Ausdruck und weiteren Ableitungen zu Konstanten bzw. Variablen die Zeichenkette herleiten

- (a+7)*21

  Diese Zeichenkette läßt sich nicht herleiten, da a kein Symbol der Grammatik ist.