Prof. Dr. Thomas Breuel
Dipl.-Inf. Patrick Michel (AG ST)
Dipl.-Ing. Ludwig Schmidt-Hackenberg

TU Kaiserslautern
Fachbereich Informatik
AG Bildverstehen und Mustererkennung

Lösungshinweise/-vorschläge zum Übungsblatt 3: Software-Entwicklung 1 (WS 2010/11)

Die Hinweise und Vorschläge in diesem Dokument sollen der Lösungsfindung dienen und erheben demnach weder Anspruch auf Vollständigkeit noch Korrektheit. Sollten Sie Fehler finden würden wir uns freuen wenn Sie uns diese mitteilen. (Kontaktdaten finden Sie auf unserer Webpräsenz.)

Aufgabe 1 Syntaxdiagramme (Präsenzaufgabe)

a) Entscheiden Sie, ob folgende Sätze zur eben definierten Sprache gehören (Leerzeichen dürfen Sie ignorieren, diese dienen nur der besseren Darstellung):
In dieser Aufgabe geht es rein um die Syntax:

1. inkorrekt: Hole und Lege nehmen erst den Greifarm, dann den Stapel.
2. korrekt
3. inkorrekt: Wiederhole bekommt einen Stapel als ersten Parameter, keinen Block.
4. korrekt
5. korrekt

b) Besitzen die in a) definierten Sätze eine sinnvolle Bedeutung (Semantik)? Diskutieren Sie.
Hier geht es um die Bedeutung der Programme. Nur syntaktisch korrekte Programme haben eine Bedeutung. Viele Programme machen nicht in jeder Situation Sinn, sie haben eine *Vorbedingung*.

1. Ohne korrekte Syntax gibt es auch keine Semantik.
2. Vorbedingung: Um nicht endlos zu laufen muss Stapel D leer sein, wenn Stapel D nicht leer ist darf R keine Karte enthalten, Bedeutung: Das Programm tut entweder nichts oder terminiert nie...
3. Ohne korrekte Syntax gibt es auch keine Semantik.
4. Vorbedingung: In L muss eine Karte sein, in R darf keine Karte sein, auf B muss eine Karte liegen, Bedeutung: Das Programm terminiert nicht...
5. Vorbedingung: Auf Stapel C müssen mindestens zwei Karten liegen, die Arme müssen beide leer sein, Bedeutung: Das Programm nimmt die ersten beiden Karten von Stapel C und legt die zweite, falls sie größer als die erste ist, auf Stapel B.

Aufgabe 2 Roboter-Programme in Haskell (Einreichaufgabe)

a) Führen Sie Ihre Roboter-Programme von Blatt 2 aus und testen Sie sie mit verschiedenen Eingaben. Versuchen Sie häufige Aktionssequenzen, wie in 2b von Blatt 2 motiviert, wiederzuverwenden:

```
aufgabe1c =
     Wiederhole A ( Hole L A :> Lege L C )
  :> Hole L C :> Lege L A
  :> Wiederhole C ( Hole L C :> Lege L B )
  :> Hole L A :> Lege L C
```

```
aufgabe2a =
     Hole L A
  :> Wiederhole A ( Hole R A :> Vergleiche Vertausche Nichts :> Lege R C )
  :> Lege L B

groesste_A_B =
     Hole L A
  :> Wiederhole A ( Hole R A :> Vergleiche Vertausche Nichts :> Lege R C )
  :> Lege L B
  :> Wiederhole C ( Hole L C :> Lege L A )

aufgabe2b = groesste_A_B

aufgabe2c =
     Wiederhole A groesste_A_B
  :> Wiederhole B ( Hole R B :> Lege R C )
  :> Wiederhole C ( Hole R C :> Lege R A )

von_B_nach_A = Wiederhole B ( Hole R B :> Lege R A )
von_C_nach_A = Wiederhole C ( Hole R C :> Lege R A )

schleife =
     Wiederhole A ( Hole R A :> Vergleiche
       ( Lege L B :> Lege R C :> von_C_nach_A :> Hole L A )
       ( Lege R C )
     )

hauptprogramm =
     Hole L A :> schleife :> von_C_nach_A :> von_B_nach_A :> Lege L B

aufgabe2d = hauptprogramm
```

b) Der Haskell-Roboter kann einen weiteren Stapel *D* verwenden. Schreiben Sie ein Roboterprogramm, das die Karten vom Anfangsstapel gleichmäßig auf drei Stapel aufteilt. Wie sieht es ohne den Zusatzstapel *D* aus?

Da für eine unbekannte Anzahl an Karten eine Schleife benötigt wird, die beim Roboter immer dazu führen wird, dass ein Stapel nachher leer ist, ist es ohne einen Extra Stapel nicht lösbar (für sowohl *endliche*, als auch *ohne Fehler terminierende* Programme).

```
-- Mit extra Stapel (insofern durch drei teilbar)
threeD =
     Wiederhole A ( Hole L A :> Lege L B
                 :> Hole L A :> Lege L C
                 :> Hole L A :> Lege L D )

-- Mit copy & paste (insofern n bekannt, hier n = 24)
threeCP =
     Hole L A :> Lege L B :> Hole L A :> Lege L C
  :> Hole L A :> Lege L B :> Hole L A :> Lege L C
  :> Hole L A :> Lege L B :> Hole L A :> Lege L C
  :> Hole L A :> Lege L B :> Hole L A :> Lege L C
  :> Hole L A :> Lege L B :> Hole L A :> Lege L C
  :> Hole L A :> Lege L B :> Hole L A :> Lege L C
  :> Hole L A :> Lege L B :> Hole L A :> Lege L C
  :> Hole L A :> Lege L B :> Hole L A :> Lege L C

-- Mit Haskell Sprachmitteln (insofern n bekannt, hier n = 24)
threeHO =
     foldl1 (:>) $ replicate 8 $ Hole L A :> Lege L B :> Hole L A :> Lege L C
```

```
-- Korrekt und fehlerfrei terminierende Loesung, allerdings mit unendlich
-- grossen Programmen. Vielen Dank an die Gruppe:
--   Thierry Entringer
--   Patrick Maricato
--   Franky Frantzen
--   Joachim Krenciszek

-- legt das Minimum von A auf B, der restliche Stabel landet wieder auf A
minAtoB = Hole L A :> Wiederhole A
  (Hole R A :> Vergleiche Nichts Vertausche :> Lege R C) :>
    Lege L B :> Wiederhole C (Hole L C :> Lege L A)

-- sortiert Stapel A absteigend mit hilfe von minAtoB
sortA = Wiederhole A minAtoB :> Wiederhole B (Hole L B :> Lege L A)

-- verteilt die Karten im Verhaeltnis 1:2 auf die Stapel B und C
startup = Wiederhole A
  (Hole L A :> Lege L B :> Hole L A :> Lege L C :> Hole L A :> Lege L C)

-- befoerdert Stapel B geflipped auf Stabel A, und die obere Haelfte von C
-- geflipped dann darauf
rekursion1 = Wiederhole B
  (Hole L B :> Lege L A :> rekursion1 :> Hole L C :> Lege L A)

-- legt die obere Haelfte von A auf B
rekursion2 = Hole R A :> Vergleiche
  (Lege R B :> rekursion2) (Lege R A :> Lege L B)

-- kombiniert alle Hilfsfunktionen um die Karten gleichmaessig auf die drei
-- Stapel zu verteilen ohne Stapel D zu benutzen
-- zum Verstaendnis ist es hilfreich die Funktion prog schrittweise auszufuehren
prog = sortA :> startup :> rekursion1 :> Hole L A :> rekursion2
```

Auf der Lösung der Studenten aufbauend hier eine von der Laufzeit viel kürzere Version, die nur die Essenz der Lösungsidee benutzt:

```
-- Mit unendlich grossem Programm, dass aber nicht unendlich abgearbeitet wird
threeInf =
      Hole L A :> Hole R A :> Vergleiche Nichts Vertausche :> Lege L C
   :> Wiederhole A ( Hole L A :> Vergleiche
        ( Lege R B :> Hole R C :> Vergleiche Nichts Vertausche :> Lege L C )
        ( Lege L B ) )
   :> Hole L C :> Lege L A
   :> Wiederhole B ( Hole L B :> Lege L A )
   :> split where

  split =
      Hole L A :> Lege L B
   :> Hole L A :> Vergleiche
        ( Lege L C :> Lege R A )
        ( Lege L C :> Hole L A :> Lege L B :> split :> Hole L B :> Lege L A )
```

Der erste Teil der Lösung bestimmt die beiden kleinsten Karten im Stapel, legt die kleinste *unter* den Kartenstapel und behält die andere in der Hand. Nun kann man mit einer Haskell-rekursiven `split` Funktion den Stapel zwei zu eins aufteilen und hat die kleinste Karte als Abbruchbedingung der Rekursion. Da die Rekursion so oft durchlaufen wurde wie die gewünschten Stapel am Ende hoch sind, kann der doppelt so hohe Stapel nun beim rekursiven Aufstieg in zwei Hälften geteilt werden.

Auch diese Lösung benutzt ein unendlich großes Roboterprogramm und die Vergleiche Operation um die unendliche Abarbeitung abzubrechen.

c) Schreiben Sie ein Roboterprogramm, das den Kartenstapel so aufteilt, dass alle Karten mit ungradem Wert auf einem Stapel liegen, alle mit gradem auf einem anderen.

Im wesentlichen sortiert man einfach und spaltet den Stapel dann auf. Das setzt allerdings das Wissen voraus, dass die Karten fortlaufend von eins anfangend nummeriert sind und setzt auch eine gerade Anzahl voraus.

```
evenOdd =
     Wiederhole A groesste_A_B
  :> Wiederhole B ( Hole L B :> Lege L A :> Hole L B :> Lege L C )
```

d) *(freiwillige Zusatzaufgabe)*

Schreiben und testen Sie ein Roboterprogramm, das den Anfangsstapel mit Hilfe des Insertionsort Algorithmus sortiert und anschließend genauso zurücklässt wie in Aufgabe 2c von Blatt 2 verlangt. Sie tauschen also die Implementierung aus, erhalten aber die Funktionalität.

In der Aufgabe wird davon gesprochen, dass wir Selectionsort schon implementiert haben. Genau genommen ist dies allerdings der Bubblesort gewesen; Für Selectionsort muss man das Primitiv aus der Zusatzaufgabe bemühen und braucht dann auch einen extra Stapel.

```
-- Um nicht immer neue von_X_nach_Y Funktionen zu schreiben:
x `nach` y = Wiederhole x (Hole L x :> Lege L y)

insert =
     Hole L A                          -- Karte zum Einsortieren
  :> Wiederhole C (                    -- Stapel zum Einsortieren
          Hole R C                     -- Vergleichskarte
       :> Vergleiche ( Lege R B ) (    -- einfach weglegen
                Lege L B :> Lege R B   -- Stelle gefunden, also Karten
             :> (C `nach` B)           --   weglegen und Rest schaufeln
             :> Hole L B               -- damit der Arm nicht leer ist
          )
     )                                 -- wenn C leer ist, ist Karte noch
  :> Lege L B                          --   in der Hand, deshalb zuruecklegen
  :> (B `nach` C)                      -- jetzt noch aufraeumen, zurueck nach C

insertionsort = Wiederhole A insert :> (C `nach` A)
```

Aufgabe 3 Kontextfreie Grammatiken (Präsenzaufgabe)

a) Welches ist der kürzeste Satz der Sprache L? Geben Sie alle Sätze der Sprache (mit Ableitung) an, die bis zu fünf Buchstaben besitzen.
Kürzeste: bcd, bis zu fünf: {bcd, abcd, aabcd, bcbcd}
Ableitungen: $V \Rightarrow Ud \Rightarrow bSd \Rightarrow bcd$, usw.

b) Überprüfen Sie, ob Ihre Ableitungen aus Aufgabenteil a *Linksableitungen* sind. Falls nicht, geben Sie zu den entsprechenden Sätzen auch eine Linksableitung an. Können Sie für einen der Sätze auch eine andere Linksableitung angeben?
Nein, geht nicht.

c) Ist die Grammatik Γ eindeutig? Wenn nicht, geben Sie ein Gegenbeispiel an.
Nein. Gegenbeispiel (zwei Linksableitungen für bcbcbcd):
$V \Rightarrow Ud \Rightarrow UUd \Rightarrow bSUd \Rightarrow bcUD \Rightarrow bcUUd \Rightarrow bcbSUd \Rightarrow bcbcUd \Rightarrow bcbcbSd \Rightarrow bcbcbcd$
$V \Rightarrow Ud \Rightarrow UUd \Rightarrow UUUd \Rightarrow bSUUD \Rightarrow bcUUd \Rightarrow bcbSUd \Rightarrow bcbcUd \Rightarrow bcbcbSd \Rightarrow bcbcbcd$
Es reicht aber auch, zwei verschiedene Ableitungsbäume für UUUd zu erstellen, das ist schließlich auch der springende Punkt.

d) Geben Sie eine kontextfreie Grammatik an, welche genau alle Datumsangaben zwischen dem 1.1.2000 und dem 31.12.3000 in diesem Format erzeugen kann. Gehen Sie vereinfachend davon aus, dass alle Monate 31 Tage haben. Sie können ein beliebiges Alphabet Ihrer Wahl zu Grunde legen und abkürzende Schreibweisen für Mengen verwenden.

Γ=(N, T, Π, S)

N={S, T, M, J}

T={0, 1, . . . , 9}

$$\Pi = \left\{ \begin{array}{rcl} S & \rightarrow & JMT \\ J & \rightarrow & 2000 \mid \cdots \mid 3000 \\ M & \rightarrow & 01 \mid \cdots \mid 12 \\ T & \rightarrow & 01 \mid \cdots \mid 31 \end{array} \right\}$$

Aufgabe 4 Syntax formaler Sprachen (Einreichaufgabe)

a) Geben Sie eine kontextfreie Grammatik an, welche genau alle Datumsangaben zwischen dem 1.1.2010 und dem 31.12.2030 erzeugen kann und dabei die unterschiedliche Anzahl an Tagen der Monate berücksichtigt. Schaltjahre sollen nicht betrachtet werden, d.h. der Monat Februar wird mit konstant 28 Tagen angenommen. Für diese Aufgabe sollen sie **keine** abkürzenden Schreibweisen mehr verwenden!

Γ=(N, T, Π, S)

N={S, C, D, M, N, U, V, T, J}

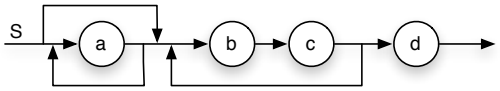
T={0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

$$\Pi = \left\{ \begin{array}{lcl} S & \rightarrow & JMT \mid J02U \mid JNV \\ C & \rightarrow & 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \\ D & \rightarrow & 0 \mid C \mid 9 \\ M & \rightarrow & 01 \mid 03 \mid 05 \mid 07 \mid 08 \mid 10 \mid 12 \\ N & \rightarrow & 04 \mid 06 \mid 09 \mid 11 \\ U & \rightarrow & 0C \mid 09 \mid 1D \mid 20 \mid 2C \\ V & \rightarrow & U \mid 29 \mid 30 \\ T & \rightarrow & V \mid 31 \\ J & \rightarrow & 201D \mid 202D \mid 2030 \end{array} \right\}$$

b) Geben Sie eine kontextfreie Grammatik an, die der Sprachdefinition mit Syntaxdiagrammen aus Aufgabe 1 entspricht.
Man kann das Diagramm 1:1 in eine Grammatik übersetzen, indem man alle Konstrukte übersetzt. Man darf das Viertupel der Grammatik nicht vergessen und muss natürlich auch die Menge der Terminale und Nichtterminale angeben. Es gibt fünf Nichtterminale namens `Programm`, `Stapel`, `Greifarm`, `Aktion` und `Block`, wobei `Programm` das Startsymbol ist. Da es auch keine Schleifen im Diagramm gibt, ergibt jeder "Durchlauf" durch das Diagramm eines Nichtterminals genau eine Grammatikregel für dieses Nichtterminal. Terminalsymbole sind sowohl Wörter wie `Lege` und `Hole`, als auch Zeichen wie '(', ':>' oder 'B'.

c) Übersetzen Sie Grammatik Γ aus Aufgabe 3 in ein Syntaxdiagramm.
Auch hier gilt, dass eine Übersetzung der Grundkonstrukte schon zielführend ist. Eine Grammatikregel kann einfach in ein Diagramm aus Sequenzen übersetzt werden. Mehrere Regeln für ein Nichtterminal fügt man als Alternativen in ein Diagramm mit dem Namen des Nichtterminals zusammen.

d) Versuchen Sie, ein möglichst minimales Syntaxdiagramm zu erstellen, das ebenfalls die Sprache L beschreibt.
Wenn man die Regeln der Grammatik 1:1 übersetzt erhält man mehrere Diagramme in denen insbesondere noch alle Nichtterminale enthalten sind. Mit den Grundformen lässt sich die Sprache L ($a^*(bc)^+d$) sehr knapp darstellen:



e) Entscheiden Sie, ob der Satz `aaaabcbcbcbcbcd` zur Sprache L gehört und geben Sie ggf. eine Ableitung an.
Der Satz gehört zur Sprache.

f) Geben Sie eine *eindeutige* kontextfreie Grammatik an, welche die Sprache L beschreibt.
Man muss nur die Regel $U \rightarrow UU$ durch $U \rightarrow UbS$ oder $U \rightarrow bSU$ ersetzen, da die Regel dann nur noch links- bzw. rechtsrekursiv ist.

Aufgabe 5 Sprachen und Grammatiken (Einreichaufgabe)

Im Folgenden betrachten wir eine Familie von Grammatiken $\Gamma_i = (N, T, \Pi_i, Q)$, mit Nicht-Terminalsymbolen $N = \{Q, F, H, L\}$ und Terminalsymbolen $T = \{k, q, z, b\}$, die sich also nur in ihren Produktionen Π_i unterscheiden:

a) Wie viele verschiedene Sprachen werden durch die obigen Grammatiken definiert? Begründen Sie!
Sieben verschiedene Sprachen:

1. Ohne jegliche Produktionen (Γ_1) oder niemals terminierenden Ableitungen (Γ_6, Γ_9) kann man keinen Satz ableiten.
2. Γ_2 definiert *nicht* die leere Sprache, sondern die Sprache, welche nur ϵ enthält.
3. Γ_4 definiert ganz normal eine Sprache (die auch von keiner anderen erzeugt wird).
4. Γ_3 erzeugt nur k und es stört auch nicht, dass sich nicht terminierende Ableitungen bilden lassen.
5. Es ist auch egal, dass manche Ableitungen in eine Sackgasse führen. Γ_5 produziert ein oder mehr qs.
6. Γ_7 und Γ_8 definieren die gleiche Sprache mit jeweils leicht anderen Regeln und umbenannten Nichtterminalsymbolen.
7. Γ_{10} erzeugt nur b, alles andere führt nur zu Ableitungen, die sich nicht mehr beenden lassen.

b) Welche Sprachen werden definiert?

$$\begin{aligned}
\mathrm{L}(\Gamma_1) = \mathrm{L}(\Gamma_6) = \mathrm{L}(\Gamma_9) &= \{\} \\
\mathrm{L}(\Gamma_2) &= \{\epsilon\} \\
\mathrm{L}(\Gamma_3) &= \{k\} \\
\mathrm{L}(\Gamma_4) &= \{(kz)^n qb^{n+1} \mid n \geq 0\} \\
\mathrm{L}(\Gamma_5) &= \{q^n \mid n > 0\} \\
\mathrm{L}(\Gamma_7) = \mathrm{L}(\Gamma_8) &= \{(kq^{n_i}z)^m b \mid n_i \geq 0, m > 0, i = 1, \ldots m\} \\
\mathrm{L}(\Gamma_{10}) &= \{b\}
\end{aligned}$$

c) Geben Sie eine kontextfreie Grammatik an, welche die Vereinigung all dieser Sprachen definiert.

$$\Gamma_{11} = (N \cup \{W\}, T, \Pi_{11}, Q) \qquad \Pi_{11} = \Pi_7 \cup \left\{ \begin{array}{lcl} Q & \to & \epsilon \\ Q & \to & k \\ Q & \to & b \end{array} \right\} \cup \left\{ \begin{array}{lcl} Q & \to & L \\ L & \to & kzLb \\ L & \to & qb \end{array} \right\} \cup \left\{ \begin{array}{lcl} Q & \to & W \\ W & \to & q \\ W & \to & Wq \end{array} \right\}$$

d) Überlegen Sie sich, wie Sie zeigen könnten, dass zwei Grammatiken dieselbe Sprache definieren. Geben Sie zwei verschiedene Grammatiken Ihrer Wahl an, und beweisen Sie, dass sie dieselbe Sprache definieren.
Ist die Sprache L bekannt kann man zum Beispiel für beide Grammatiken getrennt beweisen, dass sie in der Tat diese Sprache definieren. Dies wiederum kann z.B. durch vollständige Induktion erfolgen, indem sowohl $\mathrm{L} \subset \mathrm{L}(\Gamma)$ als auch $\mathrm{L}(\Gamma) \subset \mathrm{L}$ gezeigt werden, d.h. jeder Satz kann aus L abgeleitet werden und jeder abgeleitete Satz ist in L enthalten.

Ein anderer Ansatz ist zu zeigen, dass die Mengen der Terminalsymbole gleich und die Nichtterminalsymbole und Grammatikregeln zueinander isomorph sind. Ein sehr einfaches Beispiel ist $\Gamma_1 = (N_1, T, \Pi_1, S)$, $\Gamma_2 = (N_2, T, \Pi_2, S)$ mit:

$$N_1 = \{S, A\} \qquad N_2 = \{S, B\} \qquad T = \{a, b\}$$

$$\Pi_1 = \left\{ \begin{array}{lcl} S & \to & aSb \\ S & \to & A \\ A & \to & a \\ A & \to & bAa \end{array} \right\} \qquad \Pi_2 = \left\{ \begin{array}{lcl} S & \to & aSb \\ S & \to & B \\ B & \to & a \\ B & \to & bBa \end{array} \right\}$$

Die gesuchte Isomorphie bildet S auf S und A auf B ab und wird entsprechend in trivialer Weise auf die Mengen der Produktionen erweitert.

Die Äquivalenz von Grammatiken endlicher Sprachen ist natürlich trivial.

Das beschriebene Verfahren ist natürlich immer noch unnötig restriktiv. Es ist zum Beispiel auch kein Problem, wenn die Grammatiken in der Menge der Terminalsymbole noch weitere Symbole haben, die aber in der gemeinsam erzeugten Sprache gar nicht auftauchen. Dasselbe gilt für Nichtterminalsymbole und Grammatikregeln, so dass auch diese beiden Mengen jeweils nicht notwendig eine Isomorphie bilden müssen. Die Grammatiken müssen jeweils in den "relevanten Teilen" irgendwie aufeinander passen. Man spricht in diesem Kontext auch von *reduzierten* Grammatiken, die teilweise automatisch aus Grammatiken erstellt werden können. Sie enthalten keine Terminale, Nichtterminale oder Produktionen mehr die "unnötig" sind. Insbesondere kann man dann auch nicht-rekursive Ableitungsregeln eleminieren.