Technische Universität München
Fakultät für Informatik
M. Bauer, D. Golias, S. Riesner,
Dr. M. Schneider

WS 2002/2003
Lösungsvorschläge zu Blatt 7
28. November 2002

Übungen zu Einführung in die Informatik I

Aufgabe 24 Prüfpolynome und CRC-Berechnung (Lösungsvorschlag)

a) Gegeben sei das Generator Polynom $G_1(x) = x^{16} + x^{12} + x^5 + 1$.
Bestimmen Sie die Prüfsumme der Nachricht 100010000011111 mittels $G_1(x)$ nach dem CRC-Verfahren.

```
1000100000111110000000000000000
10001000000100001
-----------------
0000000000101110100000000000
          10001000000100001
          -----------------
            11001000010000100
            10001000000100001
            -----------------
             10000000101001010
             10001000000100001
             -----------------
                 1000101101010110
                 ==============
```

b) Ein Empfänger erhält eine mit $G_2(x) = x^3 + x + 1$ gesicherte CRC-Nachricht 1101110111010.
Überprüfen Sie, ob ein Übertragungsfehler eingetreten ist.

Ist kein Übertragungsfehler aufgetreten, so müsste restfreie Division der gesicherten Nachricht mit dem Generatorpolynom möglich sein. Dies ist hier jedoch nicht der Fall, so dass offensichtlich Fehler aufgetreten sind:

```
1101110111010
1011
----
 1101
 1011
 ----
  1101
  1011
  ----
   1100
   1011
   ----
```

```
1111
1011
----
 1001
 1011
 ----
   1010
   1011
   ----
      110
      ===
```

c) Gegeben sei das Generatorpolynom $G_3(x) = x^2 + x + 1$.
Geben sie mind. drei unterschiedliche Nachrichten an, welche durch eine CRC-Sicherung mittels $G_3(x)$ nicht unterschieden, somit ggf. nicht als Fehler erkannt werden können. Welche verallgemeinerte Regel gilt hierbei ?

Nicht unterschieden werden können z.B.: (111, 1110,11100).

Dies liegt daran, dass alle Nachrichten die sich gegenüber der Codierung des Generator Polynoms nur durch nachfolgende 0-er unterscheiden stets restfrei teilbar sind. Einen weiteren Spezialfall bildet jede Nachricht die ausschließlich aus 0-ern besteht.

Aufgabe 25 (H) CRC-Berechnung mit OCaml (Lösungsvorschlag)

Aufbauend auf das manuelle Berechungsverfahren und den bisher gelernten Techniken, soll in dieser Aufgabe Schritt für Schritt eine Funktion *generate_crc* entwickelt werden.
Eingabe sollen die zu schützende Nachricht sowie das verwendete Generatorpolynom (jeweils als Liste von 0-en und 1-en, z.B. [1;0;1]) sein, Ausgabe die entsprechend gesicherte Nachricht.

a) Hilfsfunktionen:

```
# let head_of_list lst = match lst with
        | []       -> failwith "Empty List"
        | hd::tail -> hd
;;

# let tail_of_list lst = match lst with
        | []       -> []
        | hd::tail -> tail
;;
```

b) Schreiben Sie eine boolsche Funktion *length_compare*, mit dem Ziel zwei geg. Eingabe-Listen l1 und l2 derart zu vergleichen, dass **true** zurückgegeben wird, falls l1 wenigstens soviele Elemente besitzt wie l2. Andernfalls soll **false** zurückgegeben werden.

```
# let rec length_compare n p = match n with
        | [] when p == [] -> true
        | []              -> false
        | _               -> length_compare
                                 (tail_of_list n) (tail_of_list p)
;;
```

c) Sei g die Anzahl der Elemente einer Eingabe-Liste. Schreiben Sie eine Funktion *generate_0s*, welche eine Liste mit (g-1) 0-en ausgibt.

```
# let rec generate_0s p = match p with
        | []       -> []
        | hd::[]   -> []
        | hd::tail -> [0] @ (generate_0s tail)
;;
```

d) Schreiben Sie eine Funktion *remove_leading_0s*, welche eine Liste ausgibt, die sich von eine Eingabeliste derart unterscheidet, dass alle ggf. führenden 0-en entfernt wurden.

```
# let rec remove_leading_0s n = match n with
        | []       -> []
        | hd::tail when hd == 0
                   -> remove_leading_0s tail
        | _        -> n
;;
```

e) Gegeben seinen zwei Eingabe-Listen l1 und l2. Sei g die Anzahl der Elemente der Liste l2. Entwickeln Sie eine Funktion *xor_lists*, welche die ersten g Elemente der beiden Listen mit xor vergleicht und das Ergebnis zurückgibt. Es darf davon ausgegangen werden, dass *length_compare* für l1 und l2 erfüllt ist.

```
# let rec xor_lists n p = match p with
        | []  -> []
        | hd::tail when hd == (head_of_list n)
             -> 0::(xor_lists (tail_of_list n) tail)
        | hd::tail
             -> 1::(xor_lists (tail_of_list n) tail)
;;
```

f) Entwickeln Sie eine Funktion *rest_of_list1*, welche zwei Listen l1 und l2 als Eingabe erwartet, und aus l1 soviele führende Elemente entfernt, wie l2 Elemente besitzt. Wiederum erfüllen l1 und l2 die *length_compare* Bedingung.

```
# let rec rest_of_list1 n p = match p with
        | []       -> n
        | hd::tail -> rest_of_list1 (tail_of_list n) tail
;;
```

g) Entwerfen Sie nun die Funktion *generate_crc* auf der Basis des manuellen Verfahrens (vgl. vorhergehende Aufgabe), sowie der bereits entwickelten Funktionen.

Der Übersichtlichkeit wegen soll davon ausgegangen werden, dass das Generator Polynom keine führenden 0-en enthält. Andernfalls müsste bei jeder Verwendung die Funktion *remove_leading_0s* aufgerufen werden.

Aus gleichem Grund werden zwei (nichtrekursive) Hilfs-Funktionen eingeführt:

```
# let xored_msg n p =
      remove_leading_0s (xor_lists (remove_leading_0s n) p)
;;

# let notxored_msg n p =
      rest_of_list1 (remove_leading_0s n) p
;;
```

Sowie eine eingebettete Funktion, welche die Kernaufgabe übernimmt:

```
# let rec generate_crc_rec n p =
      if (length_compare (remove_leading_0s n) p)
      then  generate_crc_rec ((xored_msg n p) @ (notxored_msg n p)) p
      else  n
;;
```

Die eigentliche Funktion:

```
# let generate_crc n p
      = generate_crc_rec (n @ (generate_0s p)) p
;;
```

Hier ist zu beachten: Beim erstmaligen Aufruf der eingebetteten Funktion wird die Nachricht n um die entsprechende Zahl an Suffix-0-en erweitert.

Das Problem dieser Lösung besteht darin, dass die Ergebnisse der *generate_crc_rec* zu kurz sein können, da im Verlauf der Berechnung alle führenden 0-en abgeschnitten werden. So wird für

```
# let n = [1;1;0;0;1;1;0;1;1;1;0;0;0;0];;
# let p = [1;0;0;1;1];;
```

[1;0;0] statt [0;1;0;0] ausgegeben. Abhilfe schafft die Einführung einer weiteren Funktion *prefix_0s*, welche vor der Rückgabe *auffüllen* hilft:

```
# let rec prefix_0s n p = match n with
    | []       -> generate_0s p
    | hd::tail -> prefix_0s tail (tail_of_list p)
;;
```

Voraussetzung ist, dass n nicht laenger als p ist. Dabei wird rekursiv solange das führende Zeichen aus n und p entfernt, bis n leer ist. Mit der *generate_0s* Funktion werden nun soviele 0s wie noetig generiert (Eins weniger als p jetzt noch lang ist).

Damit verändert sich die *generate_crc_rec* zu :

```
# let rec generate_crc_rec n p =
      if (length_compare (remove_leading_0s n) p)
      then  generate_crc_rec ((xored_msg n p) @ (notxored_msg n p)) p
      else  (prefix_0s n p) @ n
;;
```

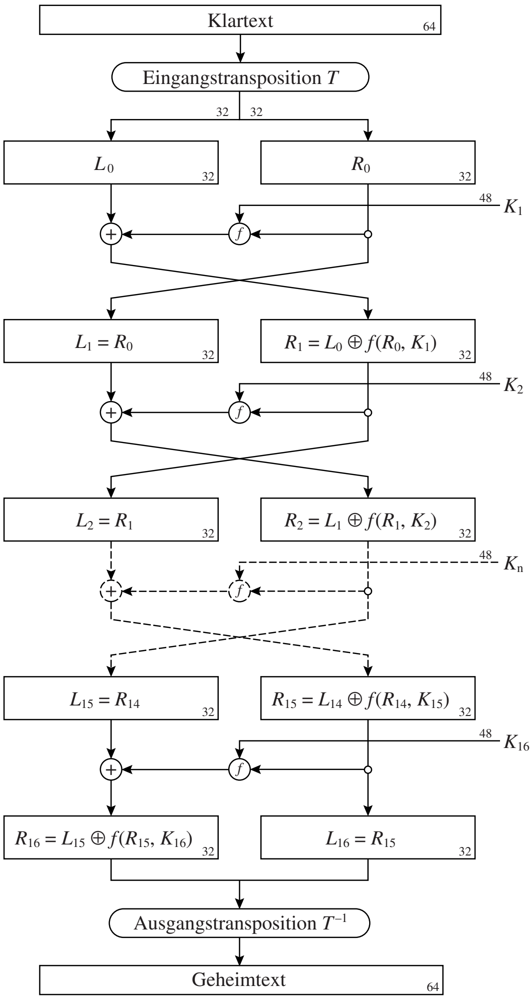
Aufgabe 26 DES Verfahren (Lösungsvorschlag)

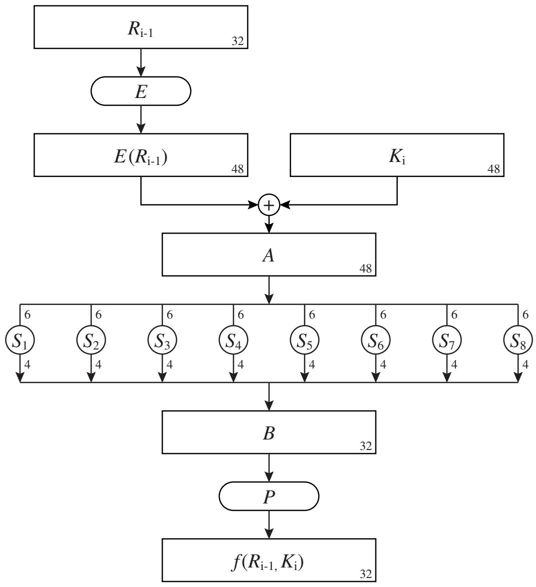
Beim DES-Verfahren wird ein Nachrichtenblock von 64 Bit Länge mit Hilfe eines Schlüssels von 56 Bit Länge chiffriert. Nach einer Eingangstransposition T wird der Klartext aufgeteilt in zwei Hälften L_0 und R_0 von je 32 Bit Länge. Darauf folgen 16 ‘Runden’ mit

$$L_i = R_{i-1} \quad \text{und} \quad R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \qquad i = 1, \dots, 16$$

wobei die Schlüssel K_i Teilmengen der Bits des ursprünglichen Schlüssels sind.

a) Zeigen Sie, daß die Dechiffrierung $DES_K^{-1}(M)$ einer Chiffrierung $DES_K(M)$ mit den Schlüsseln K_i in umgekehrter Reihenfolge entspricht.





Beweis: Die einzelnen Runden des Chirierens können durch die involutorischen Abbildungen

$$\begin{array}{llr} h_i : (R, L) \mapsto & (R, L \oplus f(R, K_i)) & (f - Funktion) \\ g : (R, L) \mapsto & (L, R) & (Vertauschen) \end{array}$$

beschrieben werden. Bei g ist die Involution offensichtlich, bei h_i folgt sie aus der Beziehung

$$L \oplus f(R, K_i) \oplus f(R, K_i) = L$$

Das Chiffrieren führt also insgesamt auf die Abbildung

$$DES = T^{-1} \circ h_{16} \circ g \circ h_{15} \circ g \circ \cdots \circ h_2 \circ g \circ h_1 \circ T$$

(bei der letzten Runde wird nicht mehr vertauscht). Beim Dechiffrieren wird lediglich die Reihenfolge der Teilschlüssel umgekehrt:

$$DES^{-1} = T^{-1} \circ h_1 \circ g \circ h_2 \circ g \circ \cdots \circ h_{15} \circ g \circ h_{16} \circ T$$

Die Komposition von DES und DES^{-1} ergibt wegen der Involution der einzelnen Abbildungen die identische Abbildung.

b) Zeigen Sie, daß gilt: $DES_{\overline{K}}(\overline{M}) = \overline{DES_K(M)}$ wobei $\overline{x}$ das bitweise Komplement von x bezeichnet.

Auch bei dieser Aufgabe hilft es, das ganze Problem schrittweise zu betrachten. Die Eingangstransposition T ist kryptologisch irrelevant, da sie am Ende wieder aufgehoben wird. Die invertierte Nachricht $\overline{M}$ wird aufgeteilt in zwei Teile $\overline{L_0}$ und $\overline{R_0}$, wobei letzteres als Eingabe für die erste Anwendung der f-Funktion dient.

Betrachten wir die f-Funktion genauer: $\overline{R_{i-1}}$ wird zuerst expandiert durch Bitverdopplungen; dabei bleibt die Invertiertheit sicher erhalten, damit erhalten wir

$$E(\overline{R_{i-1}}) = \overline{E(R_{i-1})}$$

. Dies wird per $\oplus$ (Aditiion modulo 2) mit dem invertierten Schlüssel $\overline{K_i}$ verknüpft; die Schlüssel K_i sind bestimmte Bitstellen des Eingangsschlüssels K, d.h aus $\overline{k}$ folgt $\overline{K_i}$).

Aus der Eigenschaft

$$a \oplus b = \overline{a} \oplus \overline{b}$$

erhalten wir damit als Eingang für die "S-Funktionen" die gleiche Eingabe wie im nicht invertierten Fall:

$$A = \overline{E(R_{i-1})} \oplus \overline{K_i} = E(R_{i-1}) \oplus K_i$$

und damit auch die gleiche Ausgabe B, welche nun wieder permutiert wird und zurückgegeben.

Wir haben also

$$f(\overline{R_{i-1}}, \overline{K_i}) = f(R_{i-1}, K_i)$$

und durch die Addition (modulo 2) mit L_{i-1} ergibt sich

$$\overline{L_{i-1}} \oplus f(R_{i-1}, K_i) = \overline{R_i}$$

und $\overline{R_{i-1}}$ wird einfach kopiert, also $\overline{L_i}$. Zusammengenommen gilt also, daß nach jeder Runde die Ausgaben wieder invertiert sind – da das letztendliche Ergebnis (bis auf die Ausgangstransposition T^{-1}) genau die Ausgabe der letzen Runde ist, also $\overline{R_{16}} \circ \overline{L_{16}}$, haben wir also q.e.d.

$$DES_{\overline{K}}(\overline{M}) = \overline{DES_K(M)}$$

Aufgabe 27 Zeichensequenzen (Lösungsvorschlag)

a) Die Aufgabe demonstriert u.a., das eine vollständige und eindeutige Beschreibung eines Algorithmus in natürlicher Sprache sehr schwierig ist. Eine mögliche (unvollständige) Beschreibung eines Algorithmus könnte wie folgt lauten:

Eine Lösungsidee ist, am Beginn des Worts, den man an dem Symbol „<" erkennt, ein zusätzliches Zeichen einzuführen. Initial steht dieses Zeichen für „gerade". Mit diesem Zeichen wandern wir von links nach rechts. Bei Überschreiten eines Striches von links nach rechts, ersetzen wir unser zusätzliches Zeichen durch ein Zeichen für „ungerade" und umgekehrt. Das zusätzliche Zeichen gibt somit stets an, ob die Anzahl der bereits verarbeiteten Striche gerade oder ungerade war. Bei Erreichen des Wortendes ist abschließend das Zeichen für „gerade" durch *gerade* bzw. „ungerade" durch *ungerade* zu ersetzen.

b) Ein Textersetzungssystem, das die umgangssprachlich formulierte Lösungsidee aus der Tailaufgabe a) realisiert, wäre beispielsweise:

- Variante 1

$$\begin{aligned} \text{Zeichenvorrat } V = \; & \{\; |, <, >, G, U, gerade, ungerade\} \\ \text{Regelmenge } T = \; & \{\; < \to G, \\ & G| \to U, \\ & U| \to G, \\ & G> \to gerade, \\ & U> \to ungerade\} \end{aligned}$$

  Dieses Textersetzungssystem ist deterministisch, determiniert und terminierend.

- Variante 2
  Würde man noch folgende Regeln hinzunehmen, so wäre das Textersetzungssystem nicht mehr deterministisch, aber nach wie vor determiniert und terminierend.

$$\begin{aligned} G|> &\to ungerade \\ U|> &\to gerade \end{aligned}$$

- Variante 3 (elegante Lösung ohne zusätzliche Zeichen)

$$\begin{aligned} \text{Zeichenvorrat } V = \; & \{\; |, <, >, gerade, ungerade\} \\ \text{Regelmenge } T = \; & \{<|| \to <, \\ & <|> \to ungerade, \\ & <> \to gerade\} \end{aligned}$$

c) Bei Variante 2 ergäbe sich folgende Ableitung (Ersetzungsstelle ist jeweils unterstrichen):

$$\underline{<}|||||||> \longrightarrow \underline{G|}||||||> \longrightarrow \underline{U|}|||||> \longrightarrow \underline{G|}||||> \longrightarrow \underline{U|}|||> \longrightarrow \underline{G|}||> \longrightarrow$$

$$\begin{tikzcd} \underline{U|}|> \arrow[r] & \underline{G|}> \arrow[r] \arrow[dr] & \underline{U>} \arrow[r] & ungerade \\ & & ungerade & \end{tikzcd}$$