Technische Universität München
Fakultät für Informatik
Dr. M. Schneider, Dr. M. Pizka,
Dr. A. Vilbig, Dr. R. Letz,
G. Bauer

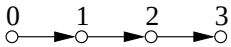
SS 2003
Lösungsvorschläge zu Blatt 1
28. April 2003

Übungen zur Vorlesung Einführung in die Informatik IV

Aufgabe 1 Relationen

Gegeben sei die folgende binäre (zweistellige) Relation über der Grundmenge **N** der natürlichen Zahlen, die "Nachfolger-Relation" $R' = \{(n, n+1) : n \in \mathbf{N}\}$. Wir betrachten im Folgenden die endliche Einschränkung R dieser Relation auf die Grundmenge $M = \{0, \ldots, 3\}$

(a) Stellen Sie die Relation mit Hilfe eines Pfeildiagramms und einer Adjazenzmatrix dar.



	0	1	2	3
0	**0**	**L**	**0**	**0**
1	**0**	**0**	**L**	**0**
2	**0**	**0**	**0**	**L**
3	**0**	**0**	**0**	**0**

(b) Stellen Sie die folgenden Relationen mit Hilfe von Adjazenzmatrizen dar und geben Sie jeweils an, um welche Relation es sich handelt.

Geben Sie an, welche dieser Relationen reflexiv, symmetrisch, antisymmetrisch, asymmetrisch, transitiv, irreflexiv, linkseindeutig, rechtseindeutig, linkstotal, rechtstotal, partielle Ordnungen, lineare Ordnungen, Äquivalenzrelationen, partielle und totale Funktionen sind.

(i) die konverse (inverse) Relation R^T: die "Vorgänger-Relation",
$R^T = \{(n+1, n) : n, n+1 \in M\}$

	0	1	2	3
0	**0**	**0**	**0**	**0**
1	**L**	**0**	**0**	**0**
2	**0**	**L**	**0**	**0**
3	**0**	**0**	**L**	**0**

(ii) das Relationenprodukt $R \circ R$: "Nachfolger des Nachfolgers",
$R \circ R = \{(n, n+2) : n, n+2 \in M\}$

	0	1	2	3
0	**0**	**0**	**L**	**0**
1	**0**	**0**	**0**	**L**
2	**0**	**0**	**0**	**0**
3	**0**	**0**	**0**	**0**

(iii) die transitive Hülle R^+: die "Kleiner"-Relation,
$R^+ = \{(n,m) : n < m \wedge n,m \in M\}$

	0	1	2	3
0	**0**	**L**	**L**	**L**
1	**0**	**0**	**L**	**L**
2	**0**	**0**	**0**	**L**
3	**0**	**0**	**0**	**0**

(iv) reflexive und transitive Hülle R^*: die "Kleiner-Gleich"-Relation,
$R^* = \{(n,m) : n \leq m \wedge n,m \in M\}$

	0	1	2	3
0	**L**	**L**	**L**	**L**
1	**0**	**L**	**L**	**L**
2	**0**	**0**	**L**	**L**
3	**0**	**0**	**0**	**L**

(v) die symmetrische Hülle R^{sym}: "Abstand 1"-Relation,
$R^{\text{sym}} = \{(n,n+1) : n,n+1 \in M\} \cup \{(n+1,n) : n,n+1 \in M\}$

	0	1	2	3
0	**0**	**L**	**0**	**0**
1	**L**	**0**	**L**	**0**
2	**0**	**L**	**0**	**L**
3	**0**	**0**	**L**	**0**

(vi) die symmetrische transitive reflexive Hülle $R^{\otimes}$: die vollständige Relation,
$R^{\otimes} = L_M$

	0	1	2	3
0	**L**	**L**	**L**	**L**
1	**L**	**L**	**L**	**L**
2	**L**	**L**	**L**	**L**
3	**L**	**L**	**L**	**L**

Wir verwenden die folgenden Notationen für einige spezielle Relationen:

- $O_M = \emptyset$ für die *Nullrelation*
- $L_M = M \times M$ für die *vollständige Relation*
- $I_M = \{(x,y) \in M \times M : x = y\}$ für die *Identitätsrelation*

Eine Relation heisst

- *reflexiv,* falls $I_M \subseteq R$
- *symmetrisch,* falls $R = R^T$
- *antisymmetrisch,* falls $R \cap R^T \subseteq I_M$
- *asymmetrisch,* falls $R \cap R^T = O_M$
- *transitiv,* falls $R \circ R \subseteq R$
- *irreflexiv,* fall $I_M \cap R = O_M$
- *linkseindeutig (injektiv),* falls $R \circ R^T \subseteq I_M$
- *rechtseindeutig (eindeutig),* falls $R^T \circ R \subseteq I_M$
- *linkstotal (total),* falls $I_M \subseteq R \circ R^T$

- *rechtstotal (surjektiv)*, falls $I_M \subseteq R^T \circ R$

Eine Relation ist eine

- *partielle Ordnung*, wenn sie transitiv, reflexiv und antisymmetrisch ist,
- *lineare Ordnung*, wenn sie transitiv, reflexiv und antisymmetrisch ist, und $R \cup R^T = L_M$
- *Äquivalenzrelation*, transitiv, reflexiv und symmetrisch ist,
- *partielle Funktion*, wenn sie (rechts)eindeutig ist,
- *totale Funktion*, wenn sie (rechts)eindeutig und (links)total ist.

	R	R^T	$R \circ R$	R^+	R^*	R^{sym}	$R^{\otimes}$
reflexiv					√		√
symmetrisch						√	√
antisymmetrisch	√	√	√	√	√		
asymmetrisch	√	√	√	√			
transitiv			√	√	√		√
irreflexiv	√	√	√	√		√	
linkseindeutig	√	√	√				
rechtseindeutig	√	√	√				
linkstotal					√	√	√
rechtstotal					√	√	√
partielle Ordnung					√		
lineare Ordnung					√		
Äquivalenzrelation							√
partielle Funktion	√	√	√				
totale Funktion							

(vii) **(H)** das Komplement der transitiven Hülle $(R^+)^-$: die "Grösser-Gleich"-Relation, $(R^+)^- = \{(n,m) : n \geq m \wedge n,m \in M\}$

	0	1	2	3
0	**L**	**0**	**0**	**0**
1	**L**	**L**	**0**	**0**
2	**L**	**L**	**L**	**0**
3	**L**	**L**	**L**	**L**

(viii) **(H)** das Komplement der reflexiven und transitiven Hülle $(R^*)^-$: die "Grösser"-Relation, $(R^*)^- = \{(n,m) : n > m \wedge n,m \in M\}$

	0	1	2	3
0	**0**	**0**	**0**	**0**
1	**L**	**0**	**0**	**0**
2	**L**	**L**	**0**	**0**
3	**L**	**L**	**L**	**0**

(ix) **(H)** den Durchschnitt $R^* \cap (R^*)^T$: die Identitätsrelation $R^* \cap (R^*)^T = I_M$

	0	1	2	3
0	**L**	**0**	**0**	**0**
1	**0**	**L**	**0**	**0**
2	**0**	**0**	**L**	**0**
3	**0**	**0**	**0**	**L**

(x) **(H)** die Vereinigung $R^+ \cup (R^+)^T$: die "Ungleich"-Relation
$R^+ \cup (R^+)^T = \{(n,m) : n \neq m \wedge n,m \in M\}$

	0	1	2	3
0	**0**	**L**	**L**	**L**
1	**L**	**0**	**L**	**L**
2	**L**	**L**	**0**	**L**
3	**L**	**L**	**L**	**0**

	$(R^+)^-$	$(R^*)^-$	$R^* \cap (R^*)^T$	$R^+ \cup (R^+)^T$
reflexiv	√		√	
symmetrisch			√	√
antisymmetrisch	√	√	√	
asymmetrisch		√		
transitiv	√	√	√	
irreflexiv		√		√
linkseindeutig			√	
rechtseindeutig			√	
linkstotal	√		√	√
rechtstotal	√		√	√
partielle Ordnung	√		√	
lineare Ordnung	√			
Äquivalenzrelation			√	
partielle Funktion			√	
totale Funktion			√	

Bemerkung: Alle diese Relationen lassen sich in natürlicher Weise auf die Grundmenge der natürlichen Zahlen fortsetzen.

Aufgabe 2 Textersetzungssysteme, Markovalgorithmen

Gegebenes Textersetzungssystem T für Eingaben der Form $w \circ \mathtt{L}$, wobei w Wort über $\{\mathtt{0},\mathtt{1}\}$:

(1) `R0` → `1R`
(2) `R` → `L`
(3) `1L` → `0L`
(4) `10L` → `0R0`
(5) `0L` → `L0`

(a) Für Eingaben der genannten Form terminiert jede Berechnung.

Beweis: Da alle Regeln die Wortlänge erhalten, können aus einem Eingabewort der Länge k nur Wörter der Länge k erzeugt werden. Die Terminierung ist damit gezeigt, wenn in keiner Berechnungssequenz dasselbe Wort mehr als einmal vorkommt. Am besten kann dies gewährleisten werden, wenn wir eine lineare Ordnung $\prec$ auf der Menge aller Wörter angeben, so dass jede Regelanwendung ein kleineres Wort produziert. Wir setzen $\mathtt{L} \prec \mathtt{0} \prec \mathtt{1} \prec \mathtt{R}$ für alle Buchstaben und allgemein $w' \prec w$ wenn w' lexikographisch vor w nach der gegebenen Buchstabenordnung. Jede Regelanwendung produziert ein kleineres Wort und da es nur endlich viele Wörter einer Länge k gibt, muss damit jede Berechnungssequenz terminieren. q.e.d.

(b) Die Ausgabe ist für alle Eingaben der genannten Form determiniert.

Beweis: Wir zeigen, dass alle Berechnungssequenzen für ein Eingabewort $w \circ \mathtt{L}$ der Form $w \circ \mathtt{L}$ mit w Wort über $\{\mathtt{0},\mathtt{1}\}$ im Wort

$(*)$ $\mathtt{L}\underbrace{\mathtt{0}\cdots\mathtt{0}}_{k \text{ mal}}$

terminieren, wobei k die Länge von w ist. Zunächst folgt aus (a), dass alle Berechnungssequenzen terminieren. Nun zeigen wir, dass alle aus Eingabewörtern der gegebenen Form ableitbaren Wörter, die nicht die Form $(*)$ haben, nichtterminal sind, woraus sich dann die Determiniertheit ergibt.

(i) Zunächst folgt aus der Struktur der Eingabe und den Regeln, dass in jedem ableitbaren Wort genau einmal L oder R vorkommt und 1 nicht rechts von L oder R vorkommt.

(ii) Ein Wort, dass R enthält, ist nichtterminal (wegen Regel (2)).

(iii) Damit müssen nur noch Wörter der Form $w = u \circ \mathtt{L} \circ v$ betrachtet werden, wobei u Wort über $\{\mathtt{0},\mathtt{1}\}$ und v Wort über $\{\mathtt{0}\}$. Aus den Regeln (3) und (4) folgt, dass in einem terminalen w keine 1 in u vorkommt, und daraus mit Regel (5), dass auch 0 nicht in u vorkommt. Damit bleibt für ein terminales Wort nur noch die Form $(*)$. q.e.d.

(c) Menge aller Wörter über dem Alphabet {0,1,L,R}, für die eine Berechnungssequenz in T existiert, in deren terminalem Wort (mindestens einmal) 1 vorkommt: Alle Wörter, der Form $u \circ \mathtt{1} \circ v$ mit u beliebig und v über $\{\mathtt{0},\mathtt{1}\}$, d.h. alle Wörter, die eine 1 enthalten und rechts davon kein L oder R, denn solch eine 1 muss/kann niemals beseitigt werden.

T unter der Markovanwendungsstrategie.

(d) Berechnungssequenz für das Eingabewort 111L:

$\mathtt{111L} \to_3 \mathtt{110L} \to_4 \mathtt{10R0} \to_1 \mathtt{101R} \to_2 \mathtt{101L} \to_3 \mathtt{100L} \to_5 \mathtt{10L0} \to_4 \mathtt{0R00} \to_1 \mathtt{01R0} \to_1 \mathtt{011R} \to_2 \mathtt{011L} \to_3 \mathtt{010L} \to_4 \mathtt{00R0} \to_1 \mathtt{001R} \to_2 \mathtt{001L} \to_3 \mathtt{000L} \to_5 \mathtt{00L0} \to_5 \mathtt{0L00} \to_5 \mathtt{L000}$

(e) Länge der Berechnungssequenz: Unter der Markovanwendungsstrategie muss Regel (4) vor (5) angewendet werden. Damit muss beim Herunterzählen eines Wortes der Form $u \circ \mathtt{10L0}\cdots\mathtt{0}$ gezwungenermassen zum Wort $u \circ \mathtt{011}\cdots\mathtt{1L}$ übergegangen werden, also zum nächstkleineren Binärwert. Die Länge der Berechnungssequenz für ein Eingabewort $w \circ \mathtt{L}$ mit w über $\{\mathtt{0},\mathtt{1}\}$ ist damit $\geq$ dem Wert von w als Binärwort gelesen (d.h. z.B. 111L hat eine Sequenz von ≥ 7 Schritten) und ist damit im worst-case exponentiell in der Eingabelänge.

(f) Berechnung der Unärzahl aus einer Binärzahl:

Grundidee: Jede Anwendung der Regel (3) oder (4) wirkt so, dass der Binärwert schliesslich um 1 abnimmt, was wir durch * notieren. Alle * werden am Wortanfang gespeichert (Regeln (A) und (B)). Am Ende werden alle 0 gelöscht (Regel (C)) und schliesslich jedes * durch 1 ersetzt und über L hinweg nach rechts verschoben (Regel (D)).

(A)	0*	→	*0
(B)	1*	→	*1
(1)	R0	→	1R
(2)	R	→	L
(3')	1L	→	*0L
(4')	10L	→	*0R0
(5)	0L	→	L0
(C)	*L0	→	*L
(D)	*L	→	L1

Aufgabe 3 (P) Reguläre Sprache, Verarbeitung strukturierter Informationen

Die Struktur einer zu verarbeitenden EMail-Nachricht folgt einer regulären Sprache, deren Grammatik zu Beginn der Klasse `SimpleMailer` unter Teilaufgabe b) angegeben ist. Das Ziel der Aufgabe ist jedoch zunächst nur eine Hinführung auf die Thematik der formalen Sprachen, so dass die Implementierung des Parsers eher ad-hoc über zahlreiche verschachtelte while-Schleifen erfolgt. Zu einem späteren Zeitpunkt ist eine Fortführung der Aufgabe mit expliziter Grammatik und methodischer, automatenbasierter Lösung geplant.

Teilaufgabe a)

```
/*
 * Class Message represents a simplified email message with single sender,
 * multiple recipients and carbon copies, single subject and body.
 */

import java.util.*;

public class Message {
    private String sender;
    private Set recipients;
    private Set carbonCopies;
    private String subject;
    private String body;

    public Message() {
        sender = new String();
        recipients = new HashSet();
        carbonCopies = new HashSet();
        subject = new String();
        body = new String();
    }

    public String toString() {
        String result = new String();
        result += "from: " + getSender() + "\n";
        result += "to: ";
        Iterator rec = getRecipients().iterator();
        while (rec.hasNext()) {
            result += (String) rec.next();
            if (rec.hasNext()) result += ", ";
        }
        result += "\n";
        result += "cc: ";
        Iterator cc = getCarbonCopies().iterator();
        while (cc.hasNext()) {
            result += (String) cc.next();
            if (cc.hasNext()) result += ", ";
        }
        result += "\n";
        result += "subject: " + getSubject() + "\n";
        result += "body: " + getBody() + "\n";
        return result;
```

```
    }

    // Standard getter/setter methods
    public String getBody() {return body;}
    public Set getCarbonCopies() {return carbonCopies;}
    public Set getRecipients() {return recipients;}
    public String getSender() {return sender;}
    public String getSubject() {return subject;}
    public void setBody(String string) {body = string;}
    public void setCarbonCopies(Set set) {carbonCopies = set;}
    public void setRecipients(Set set) {recipients = set;}
    public void setSender(String string) {sender = string;}
    public void setSubject(String string) {subject = string;}
}
```

Teilaufgabe b)

```
/*
 * Somewhat naive implementation of a parser for an
 * email message defined by the following grammar:
 * <message>    ::= <sender> <recipient>+ <carboncopy>* <subject> <body>
 * <sender>     ::= "from:" <address>
 * <recipient>  ::= "to:" <address>
 * <carboncopy> ::= "cc:" <address>
 * <subject>    ::= "subject:" <string>+
 * <body>       ::= "body:" <string>*
 * <address>    ::= <id>'@'<id>
 * <id>         ::= ['a'..'z','A'..'Z','0'..'9','.','-','_']+
 * <string>     ::= ['a'..'z','A'..'Z','0'..'9','ä'..'Ü','.',',',,'_']+
 *
 * (assuming that <id> and <string> do not contain key tokens)
 */

import java.util.*;
import java.io.*;

public class SimpleMailer {
    private Set keys;
    private Message msg;

    public SimpleMailer() {
        keys = new HashSet();
        keys.add("from:");
        keys.add("to:");
        keys.add("cc:");
        keys.add("subject:");
        keys.add("body:");
    }

    public static void main(String[] args) {
```

```
    if (args.length != 1) {
        System.out.println("Usage: java SimpleMailer <message file>");
        System.exit(0);
    }
    SimpleMailer mailer = new SimpleMailer();
    mailer.processMessage(args[0]);
}

private void processMessage(String fileName) {
    String buffer = new String();
    try {
        BufferedReader reader = new BufferedReader(new FileReader(fileName));
        while (reader.ready()) {
            buffer += reader.readLine() + " ";
        }
        reader.close();
    }
    catch (IOException ex) {
        ex.printStackTrace();
        System.exit(1);
    }
    parseMessage(buffer);
    System.out.println("*** Message ***\n" + msg);
}

private void parseMessage(String str) {
    msg = new Message();
    StringTokenizer tokenizer = new StringTokenizer(str);
    String token = tokenizer.nextToken();
    // Process single sender
    if (token.equals("from:")) {
        String tmp = new String();
        token = tokenizer.nextToken();
        while (tokenizer.hasMoreTokens() && !keys.contains(token)) {
            tmp += token + " ";
            token = tokenizer.nextToken();
        }
        processSender(tmp);
    } else {
        System.out.println("-> Expected sender!");
    }
    // Process multiple recipients
    if (token.equals("to:")) {
        while (token.equals("to:")) {
            String tmp = new String();
            token = tokenizer.nextToken();
            while (tokenizer.hasMoreTokens() && !keys.contains(token)) {
                tmp += token + " ";
                token = tokenizer.nextToken();
            }
            processRecipient(tmp);
```

```
            }
        } else {
            System.out.println("-> Expected recipient!");
        }
        // Process multiple carbon copies if present
        while (token.equals("cc:")) {
            String tmp = new String();
            token = tokenizer.nextToken();
            while (tokenizer.hasMoreTokens() && !keys.contains(token)) {
                tmp += token + " ";
                token = tokenizer.nextToken();
            }
            processCarbonCopy(tmp);
        }
        // Process single subject
        if (token.equals("subject:")) {
            String tmp = new String();
            token = tokenizer.nextToken();
            while (tokenizer.hasMoreTokens() && !keys.contains(token)) {
                tmp += token + " ";
                token = tokenizer.nextToken();
            }
            if (tmp.length() > 0)
                msg.setSubject(tmp);
            else
                System.out.println("-> Empty subject!");
        } else {
            System.out.println("-> Expected subject!");
        }
        // Process single body
        if (token.equals("body:")) {
            String tmp = new String();
            while (tokenizer.hasMoreTokens()) {
                token = tokenizer.nextToken();
                tmp += token + " ";
            }
            msg.setBody(tmp);
        } else {
            System.out.println("-> Expected body!");
        }
    }

    private void processSender(String str) {
        StringTokenizer tokenizer = new StringTokenizer(str);
        if (tokenizer.countTokens() != 1)
            System.out.println("-> Not exactly one sender address!");
        if (tokenizer.hasMoreTokens()) {
            String adr = tokenizer.nextToken();
            checkAddress(adr);
            msg.setSender(adr);
        }
```

```
    }

    private void processRecipient(String str) {
        StringTokenizer tokenizer = new StringTokenizer(str);
        if (tokenizer.countTokens() != 1)
            System.out.println("-> Not exactly one recipient address!");
        if (tokenizer.hasMoreTokens()) {
            String adr = tokenizer.nextToken();
            checkAddress(adr);
            msg.getRecipients().add(adr);
        }
    }

    private void processCarbonCopy(String str) {
        StringTokenizer tokenizer = new StringTokenizer(str);
        if (tokenizer.countTokens() != 1)
            System.out.println("-> Not exactly one carbon copy address!");
        if (tokenizer.hasMoreTokens()) {
            String adr = tokenizer.nextToken();
            checkAddress(adr);
            msg.getCarbonCopies().add(adr);
        }
    }

    private boolean checkAddress(String adr) {
        StringTokenizer tokenizer = new StringTokenizer(adr, "@");
        if (tokenizer.countTokens() != 2) {
            System.out.println("-> Invalid address: \"" + adr + "\"!");
            return false;
        }
        return true;
    }
}
```