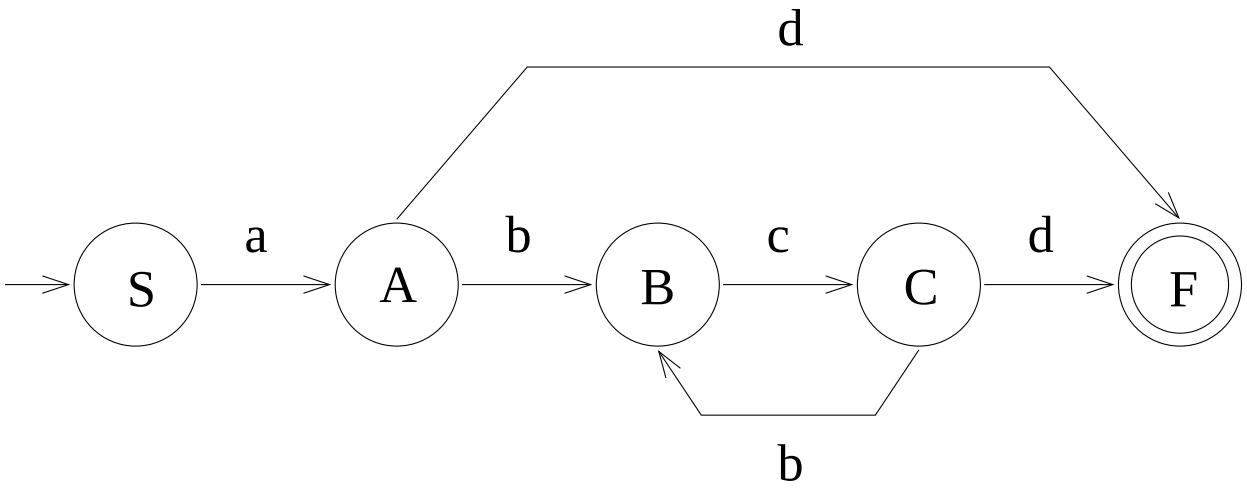
Technische Universität München
Fakultät für Informatik
Prof. Dr. A. Knoll

WS 2005/2006
11. Februar 2006

Lösungsvorschläge der Zwischenklausur zu Einführung in die Informatik I

Aufgabe 1 Endliche Automaten und reguläre Ausdrücke (Lösungsvorschlag)

a) Endlicher, deterministischer Automat:



b) Regulärer Ausdruck:

$$a\left(d \mid \left((bc)^{+} d\right)\right)$$

Aufgabe 2 Dreifachbaum (Lösungsvorschlag)

a)

```
type 'a tripleTree = Empty | Node of ('a * 'a tripleTree * '
   a tripleTree * 'a tripleTree);;
```

b)

```
let rec count tree = match tree with
    Empty -> 0
  | Node(_, a, b, c) -> 1 + (count a) + (count b) + (count c
    )
;;

let rec insert elem tree = match tree with
    Empty -> Node(elem, Empty, Empty, Empty)
```

```
  | Node(n, a, b, c) when ((count a) <= (count b) && (count
     a) <= (count c)) -> Node(n, insert elem a, b, c)
  | Node(n, a, b, c) when ((count b) <= (count a) && (count
     b) <= (count c)) -> Node(n, a, insert elem b, c)
  | Node(n, a, b, c) when ((count c) <= (count a) && (count
     c) <= (count b)) -> Node(n, a, b, insert elem c)
  | _ -> failwith "bad input"
;;

let test = insert 1 Empty;;
let test = insert 2 test;;
let test = insert 3 test;;
let test = insert 4 test;;
let test = insert 5 test;;
let test = insert 6 test;;
```

Aufgabe 3 Sortierte Menge in Java (Lösungsvorschlag)

Es müssen nur die Methoden `add` und `union` implementiert werden. Anbei wird ein vollständiges Programm plus die zugehörige Testumgebung gezeigt um das Verständnis zu erhöhen.

Hinweis: In der Methode `union` kann die erste While-Schleife auch weggelassen werden, da `add` keine doppelten Elemente einfügt.

```
public class SortedSet {

    private SetElem first;


    public SortedSet() {
        first = null;
    }

    public void add(Point p) {
        // adds a new SetElem with content p regarding the sortition
        if (first == null) {
            first = new SetElem(p);
        } else if (first.getContent().compareTo(p) > 0) {
            // insert element at first position
            SetElem newElem = new SetElem(p);
            newElem.setNext(first);
            first = newElem;
        } else {
            // simple linear search
            SetElem tmp = first;
            while (tmp.getNext() != null
                    && tmp.getNext().getContent().compareTo(p) <= 0) {
                tmp = tmp.getNext();
            }
            if (tmp.getContent().compareTo(p) == 0) {
                // p is already in the set
```

```
            return;
        }
        // p must been insert between tmp and tmp.getNext()
        SetElem newElem = new SetElem(p);
        newElem.setNext(tmp.getNext());
        tmp.setNext(newElem);
    }
}

public SortedSet union(SortedSet s) {
    SetElem e1 = this.first;
    SetElem e2 = s.first;

    SortedSet result = new SortedSet();

    while(e1 != null && e2 != null) {
        if (e1.getContent().compareTo(e2.getContent()) < 0) {
            // e2 < e1
            result.add(e2.getContent());
            e2 = e2.getNext();
        } else {
            // e2 >= e1
            result.add(e1.getContent());
            e1 = e1.getNext();
        }
    }
    // one of e1 or e2 is null
    while(e1 != null) {
        result.add(e1.getContent());
        e1 = e1.getNext();
    }
    while(e2 != null) {
        result.add(e2.getContent());
        e2 = e2.getNext();
    }
    return result;
}

//only used for testing
public static void printSet(String a, SortedSet s) {
    System.out.print("Content␣" + a + ":");
    SetElem tmp = s.first;
    while(tmp != null) {
        System.out.print("␣" + tmp.getContent().toString());
        tmp = tmp.getNext();
    }
    System.out.println();
}

public static void main(String[] argv) {
    SortedSet set1 = new SortedSet();
```

```
        set1.add(new Point(10));
        set1.add(new Point(3));
        set1.add(new Point(23));
        set1.add(new Point(15));
        set1.add(new Point(7));
        set1.add(new Point(33));
        set1.add(new Point(8));
        printSet("set1", set1);
        SortedSet set2 = new SortedSet();
        set2.add(new Point(43));
        set2.add(new Point(13));
        set2.add(new Point(2));
        set2.add(new Point(9));
        printSet("set2", set2);
        SortedSet set3 = set1.union(set2);
        printSet("set1", set1);
        printSet("set2", set2);
        printSet("set3", set3);
    }
}

public class SetElem {

    private Point content;
    private SetElem next;

    public SetElem(Point content) {
        this.content = content;
        this.next = null;
    }

    public void setNext(SetElem s) {
        this.next = s;
    }

    public SetElem getNext() {
        return this.next;
    }

    public Point getContent() {
        return this.content;
    }
}

public class Point implements Comparable<Point> {

    private Integer x;

    public Point(Integer x) {
        this.x = x;
    }
```

```
    public Integer getValue() {
        return x;
    }

    public int compareTo(Point p) {
        return this.getValue().compareTo(p.getValue());
    }

    public String toString() {
        return getValue().toString();
    }
}
```

Aufgabe 4 Korrektheit (Lösungsvorschlag)

Der Beweis teilt sich in zwei Teile auf: Zuerst wird bewiesen, dass das Resultat der Funktion $addFirst(x,ls)$ gleich der Länge der Liste ls ist, anschließend beweisen wir die Behauptung $length(subseq(ls)) = 2^{length(ls)}$.

- Z.z.:$length(addFirst(x,ls)) = length(ls)$.

  Induktionsanfang: Sei $ls = []$, dann gilt die Behauptung, den $length(addFirst(x,[])) = 0 = length([])$

  Induktionsschluß: Sei $ls' = l :: rest$ und die Behauptung gelte für die Liste $rest$. Dann gilt

$$\begin{aligned} length(addFirst(x, l :: rest)) &= length((x :: l) :: addFirst(x, rest)) \\ &= 1 + length(addFirst(x, rest)) \\ &= 1 + length(rest) \\ &= length(ls') \end{aligned}$$

- Z.z.:$length(subseq(ls)) = 2^{length(ls)}$.

  Induktionsanfang: Sei $ls = []$, dann gilt die Behauptung, denn

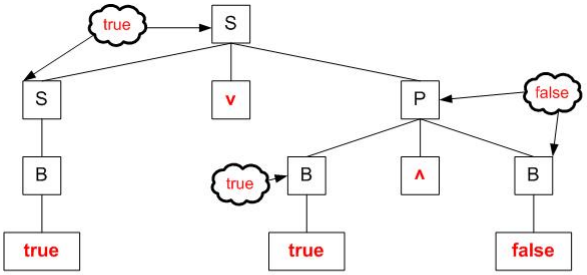
$$\begin{aligned} length(subseq([])) &= length([[]]) \\ &= 2^0 \\ &= 2^{length([])} \end{aligned}$$

  Induktionsschluß: Sei $ls' = x :: ls$ und die Behauptung gelte für ls. Dann folgt:

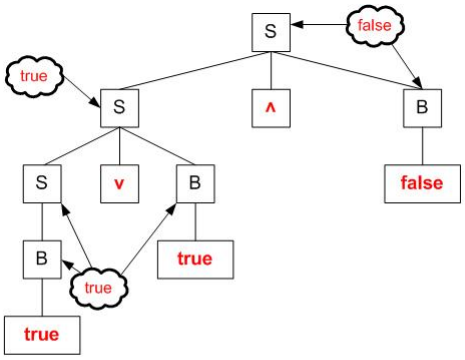
$$\begin{aligned} length(subseq(x :: ls) &= length(subseq(ls) @ (addFirst(x, subseq(ls)))) \\ &= length(subseq(ls)) + length(addFirst(x, subseq(ls))) \\ &= 2^{length(ls)} + length(subseq(ls)) \\ &= 2^{length(ls)} + 2^{length(ls)} \\ &= 2^{length(ls)+1} \\ &= 2^{length(ls')} \end{aligned}$$

Aufgabe 5 Boolsche Terme, Kontextfreie Grammatiken, Kellerautomaten (Lösungsvorschlag)

a) Ableitungsbaum für G_1:



Ableitungsbaum für G_2:



Auswertung: siehe Grafik
Beide Grammatiken beschreiben die Sprache der korrekten boolschen Terme, jedoch unterscheiden sie sich in der Auswertungsreihenfolge. G_2 wertet die Terme strikt von links aus, während G_1 neben der Auswertung von links nach rechts noch die Bindung der Operatoren berücksichtigt ($\wedge$ bindet stärker als $\vee$).

b) Kellerautomat $(Q, q_0, \Gamma, Z_0, F, \delta)$:
$Q = \{q_0, q_{bool}, q_{left}, q_{right}, q_{op}, q_f\}$
$\Gamma = \{L, \perp\}$
$Z_0 = \perp$
$F = \{q_f\}$
δ:

$$
\begin{aligned}
(q_0, b, \perp) &\rightarrow (q_{bool}, \perp) \\
(q_{op}, b, x) &\rightarrow (q_{bool}, x) \\
(q_{left}, b, x) &\rightarrow (q_{bool}, x) \\
(q_{bool}, o, x) &\rightarrow (q_{op}, x) \\
(q_{right}, o, x) &\rightarrow (q_{op}, x)
\end{aligned}
$$

$$
\begin{aligned}
(q_0,(,\bot) &\rightarrow (q_{left},L)\\
(q_{op},(,x) &\rightarrow (q_{left},xL)\\
(q_{left},(,L) &\rightarrow (q_{left},LL)\\
(q_{bool},),L) &\rightarrow (q_{right},\varepsilon)\\
(q_{right},),L) &\rightarrow (q_{right},\varepsilon)\\
(q_{bool},\varepsilon,\bot) &\rightarrow (q_f,\bot)\\
(q_{right},\varepsilon,\bot) &\rightarrow (q_f,\bot)
\end{aligned}
$$

mit
$b \in \{false, true\}$, $x \in \Gamma$, $o \in \{\wedge, \vee\}$

c) Zusätzliche Produktionsregeln: P_1:

$$
\begin{aligned}
P &\rightarrow P \wedge \neg(T) \mid B \wedge \neg(T) \mid \neg(T)\\
B &\rightarrow \neg\, true \mid \neg\, false
\end{aligned}
$$

Aufgabe 6 Programmierung (Lösungsvorschlag)

```
let rec nthRow2Col n matrix = match matrix with
        |[] -> []
        |el :: rest -> (nthRow2Col n rest) @ [nth n el]
;;

let transpose matrix =
        let rec transposeN n matrix =
                if (n > length matrix) then []
                else (transposeN (n+1) matrix) @ [(nthRow2Col n matrix)]
        in transposeN 1 matrix
```