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

SS 2003
Lösungsvorschläge zu Blatt 7
30. Mai 2003

Übungen zur Vorlesung Einführung in die Informatik IV

Aufgabe 28 Entscheidbarkeit

(**a)** Das Prädikat $r(x,y) = p(x,y) \wedge q(x,y)$ ist entscheidbar.

Beweis: Seien P und Q Entscheidungsverfahren für die Prädikate $p(x,y)$ bzw. $q(x,y)$. Entscheidungsverfahren für $r(x,y)$ für ein gegebenes Tupel (x,y):

- Wende P und Q jeweils auf (x,y) an. Beide Verfahren terminieren.
- Wenn $p(x,y) =$ **true** und $q(x,y) =$ **true**, gebe **true** aus; ansonsten **false**.

(**b)** Die transitive Hülle eines entscheidbaren Prädikates $p(x,y)$ ist i.A. nicht entscheidbar.

Beweis (durch Gegenbeispiel): Wir betrachten die Folgekonfigurationsrelation $p(x,y)$ auf Turingmaschinen, d.h. $p_{TM}(x,y)$ gdw in einer gegebenen Turingmaschine TM y eine Folgekonfiguration einer Konfiguration x ist. Offenbar ist $p_{TM}(x,y)$ für alle Turingmaschinen TM entscheidbar. Die transitive Hülle $p^*_{TM}(x,y)$ enthält genau die Paare (x,y) von Konfigurationen, für die es eine Berechnung gibt, die von x zu y führt. Insbesondere gilt das auch für terminale (bzw. akzeptierende) Konfigurationen y. Nun kann man jede Turingmaschine so modifizieren, dass sie vor dem Halten den Bandinhalt komplett löscht. Die Menge der Eingabewörter, für die die Maschine hält, bleibt gleich. Damit gibt es genau eine Haltekonfiguration h für die Maschine. Die Maschine hält angesetzt auf ein Eingabewort w genau dann, wenn $p^*_{TM}(x_0,h)$ gilt, wobei x_0 die Anfangskonfiguration für das gegebene Wort w sei. Wenn nun $p^*_{TM}(x,y)$ entscheidbar wäre, wäre das Halteproblem für Turingmaschinen entscheidbar. (Anderes Beispiel: Übergangsrelation $\rightarrow$ einer Chomsky-0-Grammatik.)

(**c)** Entscheidungsverfahren für Akzeptanzproblem einer nichtdeterministischen Turingmaschine TM mit akzeptierenden Berechnungen der Länge $\leq f(|w|)$ für ein Eingabewort w:

- Man betrachte den Konfigurationsbaum K von TM angesetzt auf w bis zur Tiefe $f(|w|)$. Da der Baum K endlich verzweigend ist, ist K endlich.
- Wenn K eine terminale bzw. akzeptierende Konfiguration enthält, gebe **true** aus, ansonsten **false**.

Das Verfahren ist $O(c^{p(f(|w|))})$-zeitbeschränkt, wobei p ein Polynom ist und c der maximale Grad des Nichtdeterminismus von TM (c ergibt sich direkt aus der Übergangsrelation δ).

Aufgabe 29 Turingmaschine, Nichtdeterminismus vs. Determinismus

(a) Grundidee für die Transformation einer nichtdeterministischen Turingmaschine TM in eine deterministische Turingmaschine ist die Betrachtung des Konfigurationsbaumes K von TM für ein

Eingabewort w. D.h. die deterministische Maschine TM' zählt alle Konfigurationen der Maschine TM angesetzt auf w auf und akzeptiert, falls eine akzeptierende Konfiguration für TM gefunden wird.

Da der Konfigurationsbaum i.A. unendlich ist, muss ein *faires* Verfahren angewandt werden, um alle Konfigurationen zu erwischen. Beim im folgenden entwickelten Breitensuchverfahren wird der Konfigurationsbaum schichtenweise abgearbeitet, alle Konfigurationen einer Schicht werden parallel gehalten und nacheinander überprüft, bevor die nächste Schicht entwickelt wird.

Zunächst muss dazu eine Möglichkeit gefunden werden, Mengen von Konfigurationen (l,a,s,r) für die Maschine $TM = (T,S,\delta,s_0,S_Z)$ auf dem Band von TM' zu speichern. Eine Konfiguration (l,a,s,r) von TM lässt sich z.B. auf dem Band von TM' kodieren als endliches Wort $l \circ \kappa(a,s) \circ r$, wobei $\kappa(a,s) : T \cup \{\#\} \times S \to T'$ eine injektive Abbildung ist (T' sei das Alphabet von TM'). Damit lässt sich jede Konfiguration eindeutig identifizieren. Mengen von Konfiguration trennt man einfach durch ein Sonderzeichen ab, z.B. durch $\vee$.

Nachdem man zu Beginn die Anfangskonfiguration von TM auf das Band geschrieben hat, geschieht die Simulation von TM durch die Iteration des folgenden Verfahrens:

(1) Man nimmt die auf dem Band am weitesten links stehende Konfiguration k_1 (falls vorhanden).

(2) Man sucht die Kopfposition von TM, das heisst die Stelle, wo $\kappa(a,s)$ steht. Falls TM k_1 akzeptieren würde (d.h. wenn $s \in S_Z$), akzeptiert man.

(3) Andernfalls hängt man alle l Folgekonfigurationen von k_1 ganz rechts auf dem Band an, durch $\vee$ voneinander und von den anderen Konfigurationen getrennt. Dazu kopiert man k_1 l-mal und simuliert jeweils einen Schritt der Maschine TM.

(4) Anschliessend geht man wieder nach links zurück und löscht die Konfiguration k_1.

(b) (H) Zur deterministischen Simulation der angegeben nichtdeterministischen Turingmaschine TM könnten wir im Prinzip das in (a) beschriebene Verfahren verwenden, was allerdings sehr aufwändig wäre. Stattdessen bieten wir hier eine einfachere Lösung. Wir nutzen dafür aus, dass alle Konfigurationsbäume für TM endlich sind. Damit entfällt das Problem der Fairness und wir können effizienter vorgehen.

(1) Für alle deterministischen Übergänge simulieren wir direkt schrittweise TM, indem wir die links stehende Konfiguration modifizieren anstatt sie nach rechts zu kopieren und dann links zu löschen.

(2) Im Fall einer terminalen Konfiguration, die nicht akzeptierend ist, wird diese gelöscht.

(3) Für den einzigen Fall von Nichtdeterminismus:

Ausführung von $((s_0,*),(s_0,a,\gg))$ oder $((s_0,*),(s_0,b,\gg))$

schreiben wir lediglich die aus dem zweiten Übergang resultierende Konfiguration rechts auf das Band.

(1) Damit erhalten wir für alle deterministischen Übergänge $((s,t),(s',t',\beta))$ in TM ein Tupel

$((r_1,\kappa(s,t),(r_1,\kappa(s',t'),\beta))$ in TM'.

(2) Für alle Paare (s,t), die nicht im Vorbereich von δ vorkommen, fügen wir ein:

$((r_1,\kappa(s,t)),(r_2,\kappa(s,t),\ll))$ (Löschen einer terminalen nichtakzeptierenden Konfiguration)

r_2-Löschunterprogramm (gehe ganz nach links, dann lösche bis zum ersten $\vee$ rechts):

$((r_2,t),(r_2,t),\ll))$ für $t \neq \#$

$((r_2,\#),(r_3,\#),\gg))$,

$((r_3,t),(r_3,\#),\gg))$, für $t \neq \vee$

$((r_3,\vee),(r_0,\#),\gg))$ (anschliessend zum Anfangszustand)

(3) Behandlung der nichtdetermistischen Übergänge: $((s_0,*),(s_0,a,\gg))$, $((s_0,*),(s_0,b,\gg))$

Vorgehen (nicht im Detail ausgeführt):

- Zunächst kopieren der Konfiguration nach ganz rechts
- Simulation des Übergangs $((s_0,*),(s_0,b,\gg))$ rechts
- Zuücklaufen nach links
- Simulation des Übergangs $((s_0,*),(s_0,a,\gg))$ links

Aufgabe 30 **Rekursive Mengen, rekursive Aufzählbarkeit**

(a) I. A. ist nicht jede Teilmenge einer rekursiven Menge rekursiv aufzählbar.

Gegenbeispiel: Sei L eine Sprache, die nicht rekursiv aufzälbar ist, und Σ das L zugrunde liegende Alphabet. Offenbar ist Σ^*, die Menge *aller* Wörter über Σ, rekursiv und eine Obermenge von L.

(b) (i) Für zwei rekursiv aufzählbare Mengen M_1 und M_2 ist die Menge $M_a = M_1 \cup M_2$ rekursiv aufzählbar.

Beweis: Sei M eine gegebene Grundmenge mit $M_1 \cup M_2 \subseteq M$. Eine rekursive Aufzählung $e_a : \mathbb{N} \to M$ für M_a (angenommen $0 \notin \mathbb{N}$) lässt sich aus den beiden (nach Annahme existierenden) rekursiven Aufzählungen $e_1 : \mathbb{N} \to M$ für M_1 und $e_2 : \mathbb{N} \to M$ für M_2 wie folgt definieren:

$$e_a(i) = \begin{cases} e_2(i/2) & \text{falls } i \text{ gerade} \\ e_1((i+1)/2) & \text{falls } i \text{ ungerade} \end{cases}$$

(ii) Für zwei rekursiv aufzählbare Mengen M_1 und M_2 ist die Menge $M_b = \{x : x \in M_1 \text{ gdw } x \in M_2\}$ i.A. nicht rekursiv aufzählbar.

Beweis (durch Gegenbeispiel): Wir wählen $M_1 = \emptyset$ und M_2 eine beliebige nicht rekursive Menge. Nach Definition gilt

$$M_b = \{x : x \in M_1 \cap M_2 \text{ oder } x \notin M_1 \cup M_2\}.$$

Da $M_1 = \emptyset$, gilt

$$M_b = \{x : x \in \emptyset \cap M_2 \text{ oder } x \notin \emptyset \cup M_2\} = \{x : x \notin M_2\}.$$

Damit kann M_b nicht rekursiv aufzählbar sein, da ansonsten M_b rekursiv wäre, im Widerspruch zur Annahme.

(c) Die transitive Hülle $H(M)$ einer rekursiv aufzählbare Menge M von Paaren (x,y) ist rekursiv aufzählbar.

Beweis: Sei $e : \mathbb{N} \to M$ eine Aufzählung von M (wobei M die Menge *aller* Paare (x,y) sei derart, dass x und y in irgendeinem Tupel von M vorkommen).

Wir definieren zunächst eine Relation $r \subseteq \mathbb{N} \times M$ wie folgt: $(i,(x,y)) \in r$ gdw (x,y) in der transitiven Hülle von $\bigcup_{j\leq i}\{e(j)\}$ ist. Da jedes $\bigcup_{j\leq i}\{e(j)\}$ endlich und nichtleer ist, ist auch seine transitive Hülle endlich und nichtleer.

Es sei e_0 eine Aufzählung aller Paare in M. Für alle $S \subseteq M$ bezeichnen wir mit μS dasjenige Paar aus S, das als erstes in der Aufzählung e_0 vorkommt.

Nun betrachten wir die wie folgt induktiv definierte Funktion $e^* : \mathbb{N} \to M$:

- $e^*(1) = e(1)$
- $e^*(i) = \mu\left(H\left(\bigcup_{j\leq i}\{e(j)\}\right) \setminus \{e^*(j) : j < i\}\right)$

(Bemerkung.: Die Definition setzt die Unendlichkeit von M voraus. Für endliche Mengen gilt die rekursive Aufzählbarkeit der transitiven Hülle aber trivialweise.)

e^* ist eine Aufzählung für die transitive Hülle $H(M)$.

Aufgabe 31 **Komplexität: Notationen**

(a) Es ergibt sich folgende aufsteigende Kette:

$$4\log n =_O 3\log n =_O 4\log(n^3) <_O \sqrt{4n}+4\log n <_O \sqrt{n}+4n\log n <_O \frac{n}{6}(\log n)^2 <_O$$
$$3n^2+4n\log n <_O n^{\log 6} <_O n^{10}-n^6+5n^3 <_O 2^n <_O 3^n =_O 3^{n-3} <_O n^{\frac{n}{2}}$$

(b) Zur Klärung der Frage, ob $f(n) \in O(g(n)), f(n) \in \Theta(g(n)), f(n) \in \Omega(g(n))$ gelten, gehen wir folgendermaßen vor: Man berechne den Grenzwert

$$\lim_{n\to\infty} \frac{f(n)}{g(n)};$$

Ist dieser Grenzwert

- gleich Null, dann gilt $f(n) \in O(g(n))$
- ist er eine Konstante, dann gilt $f(n) \in \Theta(g(n))$
- ist er „∞“,dann gilt $f(n) \in \Omega(g(n))$.

Somit folgt:

(i) $f(n) \in \Theta(g(n))$, denn

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{3^n}{3^{n-1}} = 3$$

(ii) $f(n) \in \Theta(g(n))$, denn

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n+10}{n+\sqrt{n}} = 1$$

(iii) $f(n) \in \Omega(g(n))$, denn

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\frac{n^2}{\log^2 n}}{n^{\frac{3}{2}}} = \infty$$

(iv) $f(n) \in O(g(n))$, denn

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\ln(n)}{n^k} = 0$$

Aufgabe 32 (P) Nichtdeterministische Turing-Maschine

Die vorgestellte Lösung ist verhältnismäßig umfangreich, jedoch wird ein Großteil der benötigten Klassen als Vorlage im WWW zur Verfügung gestellt. Im folgenden sind daher nur die nicht vorgegebenen Teile der Klassen `TMConfiguration` und `TuringMachine` aufgeführt.

Teilaufgabe a)

```
/**
 * Represents a configuration of the Turing Machine,
 * i.e. a tuple of tape content, current state and head position
 */
public class TMConfiguration {
    private TMTape tape;
    private TMState state;
    private int position;

    public TMConfiguration(TMTape t, TMState s, int pos) {
        tape = t;
        state = s;
        position = pos;
    }

    // Formatted output of configuration,
    // may be used to identify equivalent configurations
    public String toString() {
        return "<" + tape.toString() + "," + state + "," + position + ">";
    }

    // Standard getter/setter methods
    public TMState getState() { return state; }
    public TMTape getTape() { return tape; }
    public void setState(TMState state) { this.state = state; }
    public void setTape(TMTape tape) { this.tape = tape; }
    public int getPosition() { return position; }
    public void setPosition(int i) { position = i; }
}
```

Teilaufgabe b)

```
/**
 * Implements a nondeterministic Turing Machine (TM) with a single, left-bounded tape
 */
public class TuringMachine {
    ...
    // Tabular representation of nondeterministic transition matrix
    private Set[][] transitionMatrix;
    ...

    // Create a new instance for a TM with maximal number of states and symbols
    public TuringMachine(int maxStates, int maxSymbols) {
        ...
        transitionMatrix = new Set[maxStates][maxSymbols];
    }

    ...

    // Add a new transition to the TM,
    // all referenced states & symbols have to exist already!
    public void addTransition(String fromState, String inputSymbol,
                              String toState, String outputSymbol, int direction) {
        ...
        // Create result set and add transition for nondeterministic relation
        Set tmp = transitionMatrix[origin.getId()][input.getId()];
        if (tmp == null) {
            tmp = new HashSet();
            transitionMatrix[origin.getId()][input.getId()] = tmp;
        }
        tmp.add(result);
    }

    // Execute TM after previous initialisation, providing trace output if desired.
    // Returns true if TM reached final state or false otherwise
    public boolean execute(boolean doTrace) {
        List configurations = new Vector();     // Breadth-first expansion
        Set  oldConfigurations = new HashSet(); // Store expanded configurations
        boolean running = true;
        while (running) {
            if (doTrace) printConfig();
            running = step(configurations, oldConfigurations);
        }
        if (doTrace) printConfig();
        return currentState.isTerminal();
    }

    // Perform a singe step of the TM,
    // returning true if the TM may continue or false otherwise
    // Expects a list of configurations to expand and a set of
    // already expanded configurations
```

```
    private boolean step(List configurations, Set oldConfigurations) {
        TMConfiguration config = null;
        TMSymbol symbol = tape.read(currentPosition);
        // Determine possible transitions
        Set results = transitionMatrix[currentState.getId()][symbol.getId()];
        if (results != null) {
            // Create a new configuration for every possible transition
            Iterator resultsIt = results.iterator();
            while (resultsIt.hasNext()) {
                // Duplicate current tape and its content
                TMTape newTape = (TMTape) this.getTape().clone();
                // Retrieve result and execute transition locally
                TMResult result = (TMResult) resultsIt.next();
                TMState newState = result.getState();
                newTape.write(currentPosition, result.getSymbol());
                int newPosition = currentPosition;
                switch (result.getHeadDirection()) {
                    case LEFT: newPosition--; break;
                    case RIGHT: newPosition++; break;
                    case HOLD: ; // Do nothing
                }
                TMConfiguration newConfig =
                    new TMConfiguration(newTape, newState, newPosition);
                // Check if current configuration has been reached already
                if (!oldConfigurations.contains(newConfig.toString())) {
                    System.out.println("*** Adding configuration " + newConfig);
                    oldConfigurations.add(newConfig.toString());
                    // Append configuration to current list (breadth-first expansion)
                    configurations.add(newConfig);
                }
            }
        }
        // Retrieve first configuration if there are still configurations to try
        try {
            config = (TMConfiguration) configurations.remove(0);
        } catch (IndexOutOfBoundsException ex) {
            System.out.println("*** No more configurations to try");
            return false;
        }
        // Set new configuration of TM, indicating if execution may continue
        tape = config.getTape();
        currentState = config.getState();
        currentPosition = config.getPosition();
        return !currentState.isTerminal();
    }
    ...
}
```