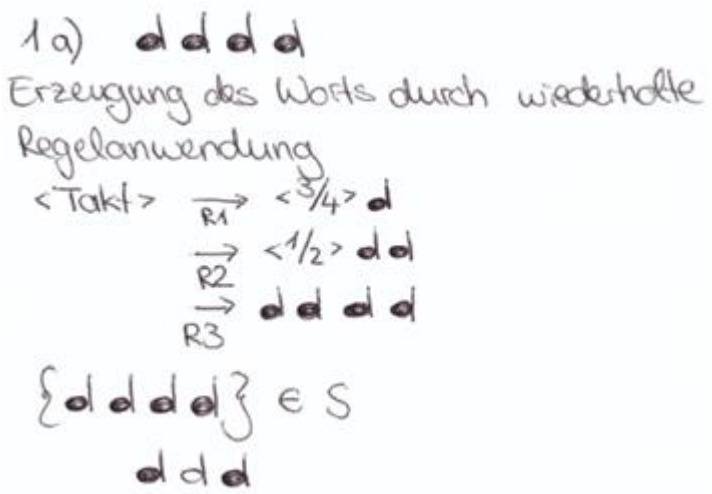
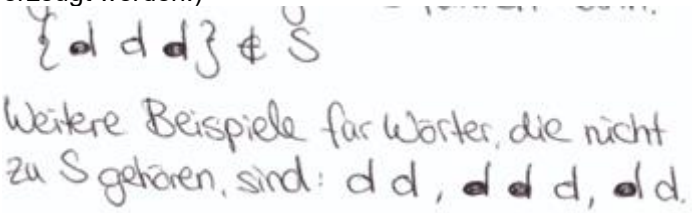
Informatik Abitur Bayern 2019 / III - Lösung

Autor:
Frauendorfer

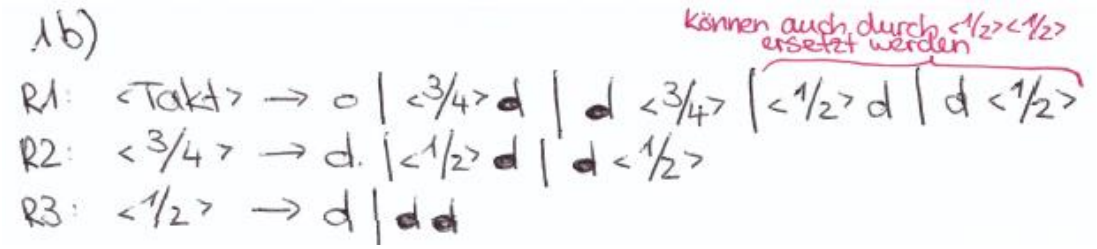
1a 3



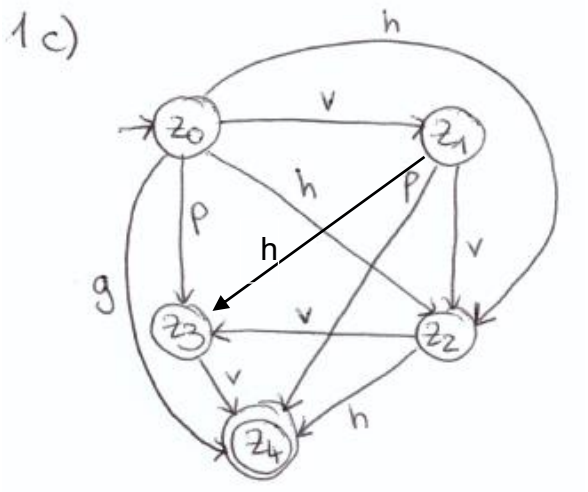
Das Wort kann durch wiederholte Regelanwendung nicht erzeugt werden, da nach Anwendung der 1. Produktionsregel (R1) keine weitere Regel zu dem gewünschten Ergebnis führt. (Ausführlicher: Die ersten beiden Regeln können nur am Ende eine Viertel Note einfügen. Die dritte erzeugt immer zwei in Folge und die gegebene Folge enthält weder eine ¼ Note am Ende, noch zwei ¼-Noten hintereinander, kann also nicht durch die Regeln erzeugt werden.)



1b 3



1c 4



Anmerkung: Zusätzlich kann noch ein Fehlerzustand eingeführt werden.

1d *Hinsichtlich der Übergänge genügt es, wenn Sie beispielhaft die vom Startzustand ausgehenden Übergänge implementieren.* 7

```
public class AUTOMAT
{
	private int zustand;

	public boolean istViervierteltakt(String eingabe)
	{
		zustand = 0;
		for (int i = 0; i<eingabe.laenge(); i=i+1)
		{
			zustandWechseln(eingabe.zeichenAn(i));
		}
		if (zustand == 4)
		{
			return true;
		}
		else
		{
			Ausgabe("Dies ist kein valider Viervierteltakt!");
			return false;
		}


	}

	private void zustandWechseln(char zeichen)
	{
		switch(zustand)
		{
			case 0:
				switch(zeichen)
				// dies geht natürlich auch mit mehreren if-
				Abfragen anstelle des zweiten switch(case)
				{
					case 'v': zustand = 1; break;
					case 'h': zustand = 2; break;
					case 'p': zustand = 3; break;
					case 'g': zustand = 4; break;
				}
				break;
				...
}	}	}	}
```

2a 4

n	z	m	Kommentar
243		0	Startwerte
	3		Wegen 3>0 ist die Bedingung wahr.
		3	
24			Wiederholung
	4		Wegen 4>3 ist die Bedingung wahr.
		4	
2			Wiederholung
	2		Wegen 2<4 ist die Bedingung falsch.
0			

Bei dem gegebenen Algorithmus wird jede Ziffer einer Zahl von der Einer-Stelle aus betrachtet und der Rückgabewert gibt die betragsmäßig größte Ziffer der Zahl n an.

2b *Bemerkung: Die Lösung nutzt die symbolische Adressierung der Speicherzellen symbolische Sprungmarken. Alternativ kann auch mit den in der Aufgabenstellung genannten Adressen gearbeitet werden.* 6

```
1  loadi 0       // lädt 0 in den Akkumulator
2  store m       // speichert m=0
3  load n        // lädt n, da in dieser Speicherzelle bereits der
                      Wert von n vorhanden ist
4  jle 15        // springt zu Befehl 15, falls n<=0
5  modi 10       // berechnet im Akkumulator n mod 10
6  store z       // n mod 10 wird in z gespeichert
7  sub m         // z – m wird berechnet, um z>=m herauszufinden
8  jlt 11        // springt zu 11, falls z<m
9  load z        // Wert von z wird geladen
10 store m       // Wert von z wird in m gespeichert
11 load n        // lädt n in den Akkumulator
12 divi 10       // teilt n durch 10 (n/10)
13 store n       // speichert das Ergebnis in m
14 jmp 3         // spring zu 3, um die Wiederholung auszuführen
15 load m        // gibt m im Akkumulator zurück
16 end
```

3a In diesem Fall muss der Zugriff auf gemeinsame Daten synchronisiert, das heißt zeitlich abgestimmt, werden. 2
Ein mögliches Konzept dafür wäre das **Monitorkonzept**.
Erklärung (nicht gefordert): Dabei ist ein Monitor ein Überwachungsmechanismus, der einem Objekt (Sitzplatz!) zugeordnet ist und bei Eintritt in den Programmteil (Methode *buchen*) explizit angefragt wird und entweder belegt wird (wenn er frei ist) bzw. auf Freigabe wartet. Er sorgt dafür, dass die Ausführung eines Programmteils nur von einem Thread und nicht von mehreren gleichzeitig erfolgen kann. Das Objekt wird beim Eintritt eines Threads in den Programmteil als belegt markiert. Alle anderen Threads, die auch den geschützten Programmteil ausführen wollen, müssen warten, bis das Objekt freigegeben wird.

3b Es kann zu einer Verklemmung kommen, wenn folgende Situation eintritt: 4
Thread A führt den Aufruf *nachbar1.buchenmit(nachbar2)* durch. Kurze Zeit später führt Thread B den Aufruf *nachbar2.buchenMit(nachbar1)* durch, obwohl Thread A noch nicht bei dem Befehl *nachbar.buchen()* ist. Da nun beide Threads nicht in der Lage sind *nachbar.buchen()* aufzurufen, da sie auf den jeweils anderen warten, entsteht eine sogenannte Verklemmung (zyklische Wartesituation von Prozessen) und keiner der Threads kann seinen Auftrag ausführen.

4 Für n=4 wird der Vergleich „a[i] gleich b[j]“ bei Algorithmus I im jedem Fall 4*4 = 16 mal ausgeführt, was im Allgemeinen in Abhängigkeit von n höchstens n^2 Vergleiche bedeutet, was dem Graphen in Abbildung C entspricht, da dieser ein quadratisches Laufzeitverhalten zeigt. 7
Beim Algorithmus II muss der Vergleich „a[i] gleich b[j]“ im ungünstigsten Fall 7 mal durchgeführt werden, da dieser die Sortierung gewinnbringend nutzen kann. Dadurch müssen allgemein höchstens 2*n-1 Vergleiche verwendet werden, was bedeutet, dass hier ein lineares Laufzeitverhalten vorliegt, welches in Abbildung B gezeigt wird.
Algorithmus I zeigt hier das ungünstigere Laufzeitverhalten.