

Übungen zu Grundlagen der Softwarezuverlässigkeit

Abgabe bis zum 8.12.2004

Aufgabe 4.1 DMMs

Um auf einem Server eingehende Anfragen bis zur Verarbeitung zwischenspeichern, werden u.a. sogenannte 'Message Queues' benutzt. Speziell werden in dieser Aufgabe hierfür Nachrichtenpuffer betrachtet, welche folgender Spezifikation genügen:

- Der Puffer kann maximal n Nachrichten zwischenspeichern.
- Jede Nachricht besitzt eine Priorität $p \in \{1, \dots, m\}$.
- Folgende Befehle werden von dem Puffer erkannt:
 - p : (mit $p \in \{1, \dots, m\}$.) Ist der Puffer nicht voll, so wird eine Nachricht mit Priorität p gespeichert und mit *ok* der Empfang bestätigt; anderenfalls bleibt der Zustand des Puffers unverändert und es wird die Meldung *fail* zurückgegeben.
 - get*: Ist der Puffer nicht leer, so wird nicht-deterministisch eine Nachricht mit höchster Priorität aus dem Puffer entfernt und mit *ok* kommentiert. Beinhaltet der Puffer hingegen keine Nachricht, so soll wiederum *fail* zurückgegeben werden.
 - reset*: Der Puffer wird geleert, d.h. alle u.U. gespeicherten Nachrichten werden gelöscht. Es wird *ok* zurückgegeben.
- Zu Beginn sei der Nachrichtenpuffer leer.

(a) Zur Vereinfachung betrachten wir nur den Fall $n = 2, m = 2$. Geben Sie hierfür eine deterministische Mealy-Maschine an. Kürzen Sie die Befehle durch die Anfangsbuchstaben jeweils ab.

Ist die so erhaltene DMM minimal (vgl. S.139/140) ? Falls nicht, minimieren Sie sie. Spielt für diese minimale DMM die Priorität einer Nachricht noch eine Rolle?

(b) Die Spezifikation wird dahingehend abändert, dass *get* statt *ok* die Priorität der betreffenden Nachricht ausgibt.

Geben Sie für diese erweiterte Spezifikation eine DMM M an.

Um mögliche Implementierungen auf Konformität zu testen, soll entsprechend der Vorlesung eine Eulertour durch den M zu Grunde liegenden Graphen G konstruiert werden. Erweitern Sie G hierfür notfalls um entsprechende Kanten, das heißt, balancieren Sie den Graphen mit dem Algorithmus von S. 158. Geben Sie dann ebenfalls unter Verwendung der Algorithmen aus der Vorlesung (S.154/155) eine Eulertour durch den erweiterten Graphen an.

- (c) Aus Sicherheitsgründen wird der Befehl *reset* deaktiviert. Um den Puffer aus einem beliebigem Zustand heraus in einen eindeutig bestimmten Zustand zu bringen, wird nun also eine Homing-Sequence benötigt. Warum ist diese in diesem Fall sehr einfach zu finden?

Wie viele Schritte benötigt hingegen die Konstruktion mittels des Ungewißheitsbaums im schlechtesten Fall? (vgl. S.163ff)

- (d) Können Sie eine Sequenz σ von Eingaben angeben, so dass die Ausgaben von M bzgl. σ für jeden Zustand von M unterschiedlich ist? Das heißt, können Sie eine 'globale' UIO-Sequenz für alle Zustände angeben?

Gibt es für einzelne Zustände auch kürzere UIO-Sequenzen? Verwenden Sie das Verfahren von S.172ff.

- (e) Erweitern Sie den Graphen entsprechend S.179 um 'Pseudokanten', das heißt um Kanten, welche das Ausführen einer UIO-Sequenz repräsentieren, und berechnen Sie eine Eulertour für den so erweiterten Graphen, nachdem Sie diesen - soweit nötig - balanciert haben.

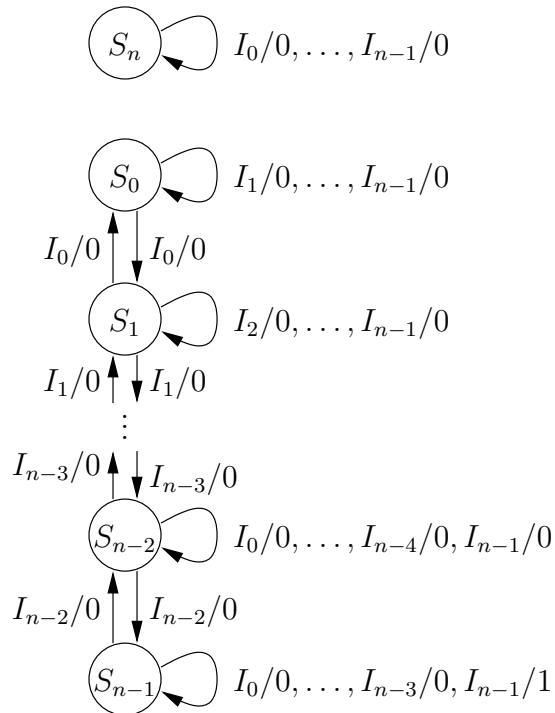
Verwenden Sie diese Eulertour, um formal eine Eingabe-/Ausgabefolge anzugeben, mit welcher Sie testen können, ob eine Implementierung alle Zustände und Transitionen der Implementierung aufweist.

- (f) Aus der Vorlesung ist bekannt, dass das Testen mit UIO-Sequenzen zwar in der Praxis häufig angewendet wird, jedoch nicht vollständig ist.

Geben Sie daher einen vollständigen Satz von separierenden Sequenzen für das Verfahren von S.180ff an.

Aufgabe 4.2 Homing-Sequences

Betrachten Sie folgende, parametrisierte DMM M_n ($n > 1$):



Die Maschine besitzt somit die Zustände $Z = \{S_0, \dots, S_n\}$, das Eingabealphabet $\Sigma = \{I_0, \dots, I_{n-1}\}$ und das Ausgabealphabet $\Omega = \{0, 1\}$. Beachten Sie, dass einzig im Zustand S_{n-1} mittels der Eingabe I_{n-1} als Ausgabe 1 erhalten werden kann. In allen anderen Zuständen ist für jede Eingabe die Ausgabe 0.

- Im Gegensatz zur Vorlesung ist der M_n zu Grunde liegende Graph nicht stark zusammenhängend. Zeigen Sie, dass dennoch eine Homing-Sequence existieren muss.
- Zu einer gegebenen Eingabesequenz $\sigma = \sigma_1 \dots \sigma_l$ definieren wir eine Folge von Vektoren wie folgt:

- Es wird mit der 'Identität' $v^{(0)} := (S_0, \dots, S_{n-1})$ begonnen.
- Für $\sigma_i = I_k$ mit $k < n - 1$ entsteht $v^{(i)}$ aus $v^{(i-1)}$ durch Vertauschen der k -ten mit der $k + 1$ -ten Komponente:

$$v^{(i)} = (v_0^{(i-1)}, \dots, v_{k-1}^{(i-1)}, v_{k+1}^{(i-1)}, v_k^{(i-1)}, v_{k+2}^{(i-1)}, \dots, v_{n-1}^{(i-1)}).$$

Gilt $\sigma_i = I_{n-1}$, so wird $v^{(i)} = v^{(i-1)}$ gesetzt.

- Zeigen Sie, beginnt M in dem Zustand $v_i^{(l)}$ für $i < n$, so befindet sich M nach Einspielen von σ in dem Zustand S_i . Weiterhin sind alle Komponenten der Vektoren $v^{(k)}$ für jedes k paarweise verschieden.
- Wir schreiben die Zeilenvektoren $v^{(l)}$ für eine feste Eingabesequenz σ untereinander und bilden so die Matrix V_σ . σ heißt *gut*, falls jeder Zustand S_j für $j < n$ mindestens einmal in der letzten Spalte von V_σ auftritt. Zeigen Sie:
Ist σ nicht *gut*, d.h. existiert ein Zustand S_k mit $k < n$, welcher nie in der letzten Spalte von V_σ auftritt, dann kann dieser Zustand S_k nicht von S_n durch die Ausgabe bei Einspielung von σ unterschieden werden.

- iii) Zeigen Sie nun, dass jede Homing-Sequence für M_n mindestens die Länge $\frac{n}{2}(n+1)$ besitzt.

Aufgabe 4.3 Adaptive Homing-Sequences

Die aus der Vorlesung bekannte Definition für Homing-Sequences verlangt eine Sequenz σ , welche unabhängig von den Ausgaben ist, welche während des Ausführens von σ beobachtet werden. Das heißt, σ wird stur abgespielt und erst am Ende wird auf Grund der Ausgabe bestimmt, in welchem Zustand sich der Automat befindet.

Eine bessere Strategie ist natürlich, während des Einspielens von σ bereits auf Grund der bisher beobachteten Ausgabe die nächste Eingabe zu wählen. Damit wird σ zu einer Abbildung von der bereits beobachteten Ausgabe und der damit verbleibenden Ungewissheit.

Als Länge einer adaptiven Homing-Sequence σ definieren wir die maximale Anzahl von Eingaben entsprechend σ , welche zur eindeutigen Bestimmung des aktuellen Zustands benötigt werden.

- (a) Machen Sie sich klar, dass das Anwenden einer adaptiven Homing-Sequence immer höchstens soviele Schritte benötigt wie das Anwenden einer gewöhnlichen Homing-Sequence.
- (b) Zeigen Sie, dass für eine minimale reduzierte (d.h. es existieren keine vom Startzustand unerreichbaren Zustände) DMM M mit n Zuständen stets gilt:
Jede Menge von $n - k$ Zuständen von M für $0 \leq k \leq n - 2$ besitzt ein Paar von Zuständen, welches eine separierende Sequenz der Länge $\leq k + 1$ besitzt.
- (c) Zeigen Sie mittels dem letzten Resultat, dass eine reduzierte minimale DMM immer eine adaptive Homing-Sequence der Länge $\leq \frac{n}{2}(n - 1)$ besitzt.
- (d) Was ist die kürzeste adaptive Homing-Sequence für die DMMs M_n aus Aufgabe 4.2?