Modul-Aufgabe 4

Ansprechpartner:

Wolfgang Bauer – *WolfgangM.Bauer@th-nuernberg.de*
Jürgen Bäsig – *Juergen.Baesig@th-nuernberg.de*

Beschreibung:
Wegen andauernder Beschwerden der Studierenden über die Qualität des Kaffees in der Mensateria wird beschlossen, einen hochwertigen Kaffee - 100% Arabica, 0% Robusta - zum Preis von 4,-Euro pro Becher (150 ml) anzubieten. Hierzu soll ein Gourmet-Kaffee-Automat entwickelt werden, wobei Sie als Student/in der Elektrotechnik und Informationstechnik die Aufgabe erhalten, einen Zustandsautomaten für eine korrekte Münzannahme und einen ordnungsgemäßen Ablauf beim Ausschenken des Kaffees zu entwerfen und in der Programmiersprache C zu programmieren.

Die grafische Schnittstelle wird mittels einer statischen Bibliothek bereitgestellt. Diese kommuniziert anhand einer C-Schnittstelle, die in der **automat.h** vorgeben ist, mit dem zu erstellenden C- Programm. Die Schnittstelle beinhaltet folgende Funktionen:

- **automat_reset()**: Setzt den Zustandsautomat zurück in den Anfangszustand
- **automat_transition()**: Führt einen Zustandsübergang gemäß des übergebenen Eingangsvektor durch
- **automat_output()**: Gib den Ausgangsvektor des aktuellen Zustands zurück

Folgende Randbedingungen sind zu berücksichtigen:

- Alle Münzen außer 1-Euro und 2-Euro Münzen werden von einem vorgegebenen Münzerkennungssystem automatisch in den Geldauswurf umgeleitet. Für eine korrekte Münze wird das Signal **muenze** kurzzeitig auf *TRUE* gesetzt, wobei das Signal **muenz_wert** für eine 1-Euro Münze *FALSE* und für eine 2-Euro Münze *TRUE* ist.
- Es gibt kein Wechselgeld und keine Überzahlung. Falls zu viel gezahlt wird, soll die jeweilige Münze durch das kurzeitige Setzen des Signals **muenz_rueck** auf *TRUE* in den Geldauswurf umgeleitet werden.
- Das Ausgangssignal **guthaben** beträgt das aktuelle Guthaben des Kunden. Wenn der Betrag von 4,- Euro bezahlt wurde, soll die Kaffeeausgabe durch das Signal **kaffee_los** gestartet werden. Das Signal **kaffee_los** wird dabei kurzzeitig auf *TRUE* gesetzt.
- Es ist zu berücksichtigen, dass **kaffee_los** nur gesetzt werden kann, wenn über das Eingangssignal **becher** sichergestellt ist, dass ein Becher bereit steht.
- Erst nach Wegnahme des gefüllten Bechers ist es möglich, einen neuen Becher Kaffee zu bezahlen.
- Eine LED-Beleuchtung wird durch das Ausgabesignal display wie folgt angesteuert:
  - *FALSE*: Bitte 4,- Euro einwerfen!
  - *TRUE*: Bitte Becher unterstellen!
- Mittels **display_string** kann dem Benutzer eine Meldung angezeigt werden.

Aufgaben:

a) Entwerfen Sie einen Moore-Automaten. Im Initialisierungszustand – A – tragen alle Ausgangssignale den Signalzustand *FALSE*. Für den Eingangs- und Ausgangsvektor gelte:

EV = **becher**, **muenze**, **muenz_wert**
AV = **display**, **muenz_rueck**, **kaffee_los** [, **guthaben**, **display_string**]

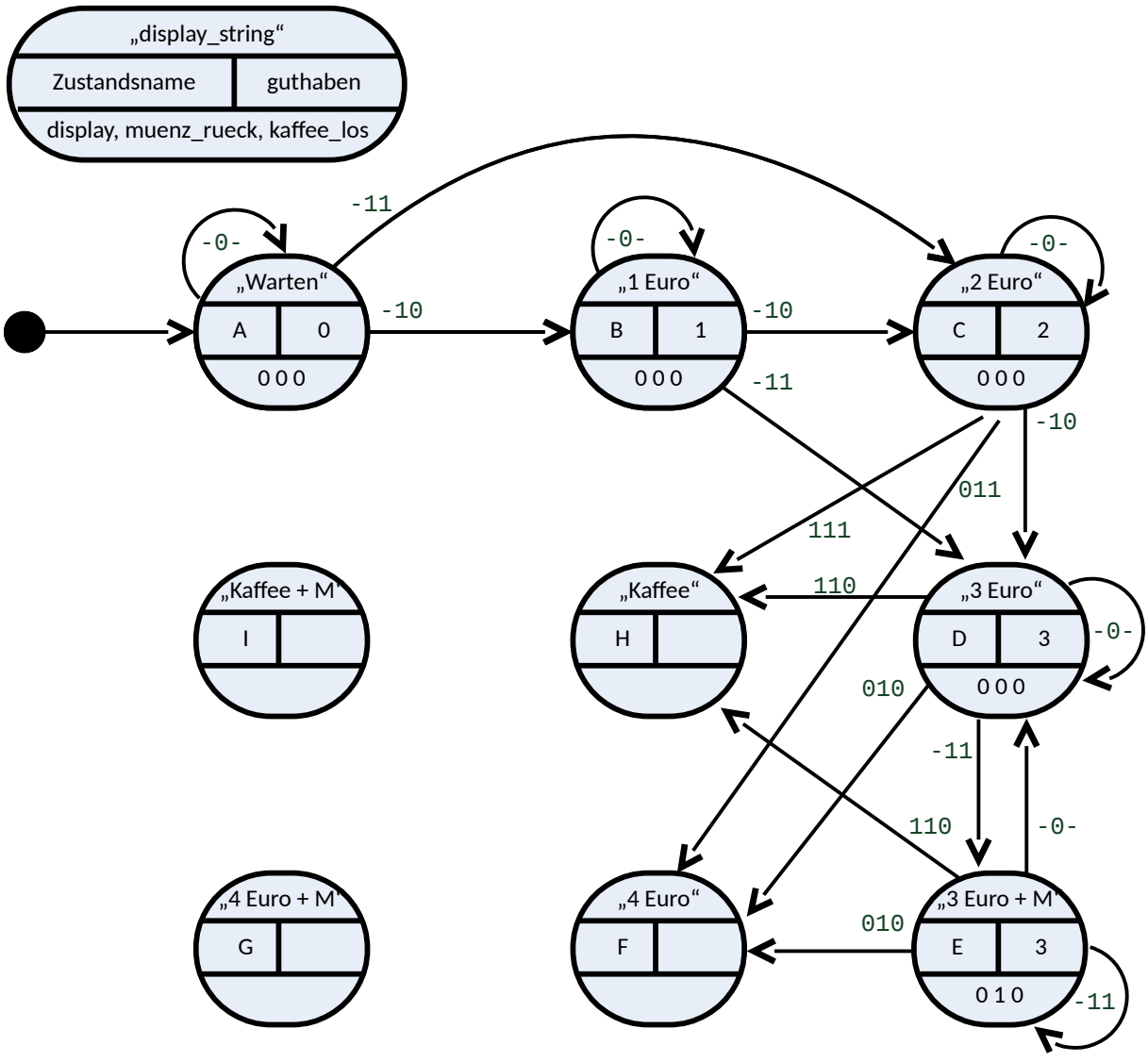
Es wird empfohlen den Automaten mit dem Programm AutomatenSim zu simulieren.

b) Implementieren Sie den Automaten in der Programmiersprache C. Berücksichtigen Sie die Deklarationen in der **automat.h** Datei.

c) Führen Sie einen exemplarischen Test durch. Bestimmen Sie die Anzahl der Eingaben, damit bei dem Automaten alle Zustände und Übergänge mindestens einmal durchlaufen werden.

Ergänzen Sie den folgenden angefangenen Zustandsautomaten:

Legende:



Beschreibung der verschiedenen Laufzeitumgebungen zur Modul-Aufgabe 4

Dieser Abschnitt beschreibt die notwendigen Schritte, um die in Modul-Aufgabe 4 benötigte Laufzeitumgebung aufzusetzen. Um den Aufwand möglichst gering zu halten, werden verschiedene Pakete zur Verfügung gestellt.

Übersicht

Paketname	Betriebssystem	Systemvorraussetzungen	
kaffee-automat-win-codeblocks.zip	Windows	*IDE:*	Code::Blocks (Version >= 16.01)
		Compiler:	mingw-gcc
kaffee-automat-all-console.zip	Windows Linux MacOS	*Compiler:*	mingw-gcc / visual-c++ / gcc / clang
		Tools:	mingw32-make / make (optional)
kaffee-automat-all-source.zip	Windows Linux MacOS	*Compiler:*	mingw-gcc / gcc / clang
		Tools:	mingw32-make / make / xcode
		Libraries:	Qt (Version 4 oder 5)

kaffee-automat-win-codeblocks

Inhalt:

- Leere Funktionsrümpfe: automat.c
- Schnittstellenbeschreibung: automat.h
- Code::Blocks-Projektdatei: kaffee_automat.cbp
- Statische Bibliothek mit der grafischen Benutzeroberfläche: lib\libkaffee_automat.a
- Laufzeitbibliotheken: bin\Debug\{Qt5Core, Qt5Gui, libgcc_s_dw2-1.dll, ...}

Erzeugung und Benutzung:
Die beiliegende Projektdatei ist so konfiguriert, dass mit dem Debug-Target eine ausführbare Datei erstellt und gestartet werden kann. Dabei wird Datei `kaffee_automat.exe` im Ordner `bin\Debug` erstellt. Zur Aufführung der Datei werden bereits in diesem Ordner befindlichen Dateien benötigt.

kaffee-automat-all-console.zip

Inhalt:

- Leere Funktionsrümpfe: automat.c
- Schnittstellenbeschreibung: automat.h
- Konsolenbasierte Oberfläche: view.h, view.c, io.h, io.c, checker.h, checker.c, s_transaction_list.h, main.c
- Erzeugungsbeschreibung für gcc: Makefile

Erzeugung und Benutzung:
Die Version der Laufzeitumgebung hängt ausschließlich von Standard-C Bibliotheken an. Zur Erzeugung des aufführbaren Programmes werden alle in diesem Packet vorhandenen C-Dateien kompiliert und gelinkt. Dazu kann die gewohnte C-Programmierumgebung genutzt werden. Bei der Erzeugung und Ausführung des Programms in der Konsole unter Linux ergeben sich folgende Befehle:

```
cd Verzeichnisname
gcc automat.c view.c io.c checker.c main.c -o console_automat
./console_automat
```

Alternativ kann die aufführbare Datei mit Hilfe des mitgelieferten `Makefile` erzeugt werden. In dieser sind alle für die Erstellung notwendigen Befehle und Abhängigkeiten beschrieben. Folgende Schritte müssen in einer Eingabeaufforderung (cmd.exe) ausgeführt werden. Hier muss sichergestellt sein, dass die Umgebungsvariable PATH so eingerichtet ist, dass die Programme `make` bzw. `mingw32-make` und `gcc` gefunden werden. Unter Windows befindet sich dazu meist im *MinGW*-Installationsverzeichnis eine Skriptdatei (z.B. `mingwvars.bat`), mittels der die Umgebungsvariablen entsprechend gesetzt werden können.

```
cd Verzeichnisname
make -f Makefile
./console_automat
```

kaffee-automat-all-source.zip

Inhalt:

- Leere Funktionsrümpfe: app/automat.c
- Schnittstellenbeschreibung: app/automat.h
- Quellen für die grafische Benutzeroberfläche: lib/*
- Erzeugungsbeschreibungen: kaffee_automat.pro, lib/lib.pro, app/app.pro

Erzeugung und Benutzung:
Zur Erstellung der grafischen Benutzeroberfläche muss die C++-Klassenbibliothek Qt in der Version 4 oder 5 installiert sein. Diese beinhaltet zusätzlich das Programm `qmake`, mit dem aus qt-Projektdateien Makefiles erzeugt werden können. Unter MacOs werden mit Hilfe von qmake Xcode-Projektdateien erstellt. Mittels der Projektdatei `kaffee_automat.pro` werden

zunächst die statische Bibliothek für die grafische Schnittstelle und anschließend die ausführbare Datei erzeugt. Folgende Schritte müssen in einer Eingabeaufforderung (`cmd.exe` oder `/bin/bash`), die entsprechend eingerichtet ist, ausgeführt werden:

Windows:

```
X:                              (Wechsel auf das benötigte Laufwerk)
cd Verzeichnisname              (Wechsel in das Quellverzeichnis)
qmake -r                        (Erzeugung des Makefiles)
mingw32-make.exe -f Makefile (Kompilieren und Linken der Quelldateien)
app\automat.exe                 (Ausführen des Programms)
```

Linux:

```
cd Verzeichnisname              (Wechsel in das Quellverzeichnis)
qmake -r                        (Erzeugung des Makefiles)
make -f Makefile                (Kompilieren und Linken der Quelldateien)
app/automat                     (Ausführen des Programms)
```

MacOS:
Da unter MacOS von qmake anstelle von Makefiles Xcode-Projektdateien erstellt werden, gibt es keinen automatisierten Mechanismus, der zuerst die statisch Bibliothek und anschließend die Zustandsbeschreibung kompiliert. In dieser Beschreibung werden die beiden Kompiliervorgänge nacheinander durchgeführt. Anstatt des hier beschriebenen `xcodebuild`-Aufrufs (dazu sind die Xcode-Commandline-Tools notwendig) ist es ebenfalls möglich die Xcode-Projekte mit Hilfe der grafischen Oberfläche zu übersetzen.

```
cd Verzeichnisname              (Wechsel in das Quellverzeichnis)
qmake -r                        (Erzeugung der Xcode-Projektdateien)
cd lib                          (Wechsel in das lib-Quellverzeichnis)
xcodebuild                      (Erzeugung der statischen Bibliothek)
cd ../app                       (Wechsel in das app-Quellverzeichnis)
xcodebuild                      (Erzeugung des Applikationspakets)
open build/Release/automat.app  (Starten der Applikation)
```