10.6 ENDLICHE AUTOMATEN

EINFÜHRUNG

Will man untersuchen, welche Probleme ein Computer grundsätzlich lösen kann und welches sein Grenzen sind, so muss man zuerst exakt definieren, was man unter einer Rechenmaschine versteht. De berühmte Mathematiker und Informatiker Alan Turing veröffentlichte bereits 1936 eine Untersuchung z diesem Thema, lange bevor es überhaupt einen programmierbaren Digitalrechner gab. Die nach ih benannte **Turingmaschine** durchläuft programmgesteuert und auf Grund von Eingabewerten, die sie vo einem Band liest, schrittweise einzelne Zustände und schreibt dabei Ausgabewerte auf das Band. Dies grundsätzliche Vorstellung über die Funktionsweise des Computers ist auch heute noch gültig, denn jede Prozessor ist eigentlich eine Turingmaschine, die im Takt einer Clock Zustand um Zustand durchläuf Besser an die Praxis angepasst sind allerdings Zustandsautomaten, die sich mit Zustandsgraphe modellieren lassen. Man nennt sie **Endliche Automaten**.

PROGRAMMIERKONZEPTE:*Turingmaschine, Endlicher Automat, Mealy-Automat, Automatengraph, Formale Sprache*

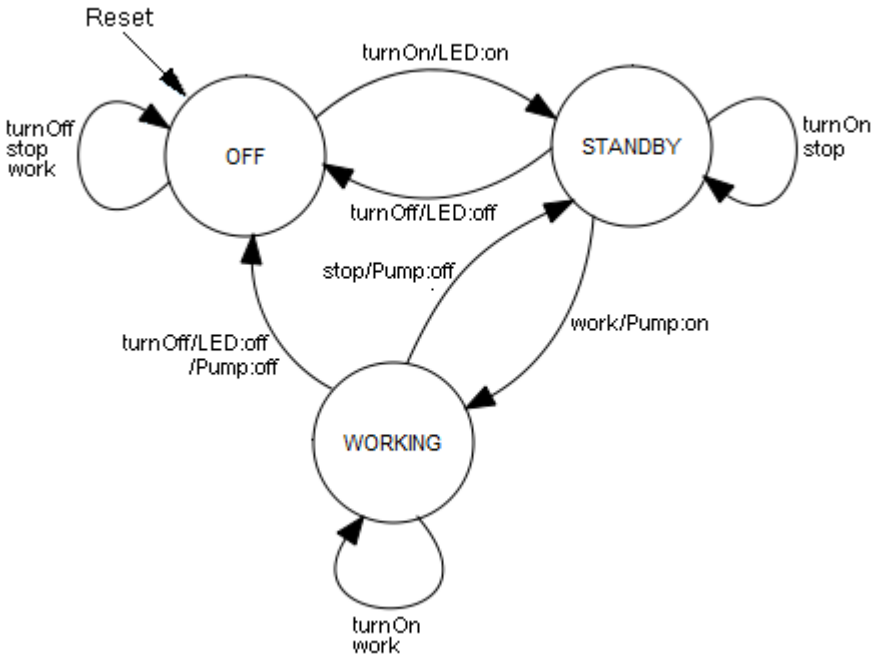
DIE ESPRESSO-MASCHINE ALS MEALY-AUTOMAT

Im täglichen Leben machst du Bekanntschaft mit vielen Geräten und Maschinen, die du als Automate auffassen kannst. Dazu gehören Getränkeautomaten, Waschautomaten, Geldautomaten, usw. A Ingenieur und Informatiker entwickelst du einen solchen Automaten mit der klaren Vorstellung, dass diese ausgehend von einem aktuellen **Zustand** schrittweise in einen **Nachfolgezustand** übergeht, der vo Sensordaten und der Betätigung von Tasten und Schaltern abhängt. Dies nennst du die **Eingaben** de Automaten. Bei jedem **Übergang** betätigt der Automat bestimmte Aktoren, wie Motoren, Pumpen, Lampe usw. Dies nennst du die **Ausgaben** des Automaten.

Du entwickelst hier einen Espresso-Automaten, der 3 Zustände besitzt: Er kann ausgeschaltet (OFF betriebsbereit (STANDBY) und am Kaffeepumpen (WORKING) sein. Zur Bedienung stehen 4 Drucktaste für die Funktionen *Einschalten* (turnOn), *Ausschalten* (turnOff), *Kaffeepumpe einschalten* (work *Kaffeepumpe ausschalten* (stop) zur Verfügung.

Du kannst zwar die Funktionsweise des Espresso-Automaten in Worten beschreiben. Viel anschaulicher is es aber, den **Automatengraph** zu zeichnen. Dabei stellst du die **Zustände** mit einem **Kreis** und di **Übergänge** als **Übergangspfeile** dar, die du mit den **Eingaben/Ausgaben** anschreibst. Zudem ist e wichtig festzulegen, in welchem **Anfangszustand** der Automat ist, wenn du ihn ans Netz anschliesst. Da i jedem Zustand irgendeine Taste gedrückt werden kann, müssen bei jedem Zustand alle mögliche Eingaben vorkommen. Falls keine Aktion erfolgt, wird die Ausgabe weggelassen.

Automatengraph:



Du kannst das Verhalten auch in einer Tabelle festhalten, in der du zu jedem Zustand s und jeder Eingabe den Nachfolgezustand s' angibst. Mit einem Stern bezeichnest du den Anfangszustand.

Übergangstabelle:

t = s =	**OFF(*)**	**STANDBY**	**WORKING**
turnOff	OFF	OFF	OFF
turnOn	STANDBY	STANDBY	WORKING
stop	OFF	STANDBY	STANDBY
work	OFF	STANDBY	WORKING

Mathematisch ausgedrückt kannst du sagen, dass der Nachfolgezustand s' eine Funktion des aktuelle Zustands *s* und der Eingabe t ist: *s' = F(s, t)*. Du nennst F die **Übergangsfunktion**.

Die Ausgaben, die zu jedem Zustand und einer Eingabe gehören, kannst du ebenfalls tabellarisc festhalten:

Ausgabetabelle:

t = s =	**OFF(*)**	**STANDBY**	**WORKING**
turnOff	-	LED off	LED off, Pump off
turnOn	LED on	-	-
stop	-	-	Pump off
work	-	Pump on	-

Mathematisch kannst du auch hier sagen, dass die Ausgabe g eine Funktion des aktuellen Zustands s un der Eingabe ist: *g = G(s, t).* Du nennst G die **Ausgabefunktion**.

MEMO

Die Zustände (mit Auszeichnung des Anfangszustandes), die Eingabe- und Ausgabewerte, sowie di Übergangs- aus Ausgabefunktion bilden zusammen einen sogenannten **Mealy-Automaten**.

IMPLEMENTIERUNG DES ESPRESSO-AUTOMATEN MIT STRINGS

Der Auslöser für den Übergang von einem Zustand zum nächsten soll ein Tastendruck sein. Di betreffende Taste legt den Eingabewert fest, und zwar werden die 4 Cursor-Tasten verwendet. Di Implementierung ist typisch: In einer endlosen Ereignisschleife (event loop) wartet das Programm m getEntry() auf eine Tastatureingabe. Mit dem Rückgabewert wird der aktuelle Zustand gemäss de Übergangstabelle geändert und die Ausgaben gemäss der Ausgabetabelle gemacht.

```
from gconsole import *

def getEntry():
    keyCode = getKeyCodeWait()
    if keyCode == 38: # up
        return "stop"
    if keyCode == 40: # down
        return "work"
    if keyCode == 37: # left
        return "turnOff"
    if keyCode == 39: # right
        return "turnOn"
    return ""

state = "OFF"  # Start state
makeConsole()
while True:
    gprintln("State: " + state)
    entry = getEntry()
    if entry == "turnOff":
       if state == "STANDBY":
            state = "OFF"
            gprintln("LED off")
       if state == "WORKING":
            state = "OFF"
            gprintln("LED and pump off")
    elif entry == "turnOn":
        if state == "OFF":
            state = "STANDBY"
            gprintln("LED enabled")
    elif entry == "stop":
        if state == "WORKING":
            state = "STANDBY"
            gprintln("Pumpe off")
    elif entry == "work":
        if state == "STANDBY":
            state = "WORKING"
            gprintln("Pumpe enabled")
```

Programmcode markieren (Ctrl+C kopieren, Ctrl+V einfügen)

MEMO

In der Eventloop werden nur diejenigen Events behandelt, die zu einem Zustandswechsel führen oder di eine Ausgabe erzeugen.

Mit makeConsole() erzeugst du ein einfaches Eingabe-/Ausgabefenster, das die Eingabe von einzelne Tastaturzeichen (ohne nachfolgendes <return>) ermöglicht. Der Aufruf getKeyCodeWait() wartet, bis ein Taste gedrückt wurde und gibt ihren Code zurück. Die Dokumentation zum Modus gconsole findest du i TigerJython-Menü unter Hilfe/APLU Dokumentationen.

ENUMERATIONEN ALS ZUSTANDS- UND EVENTBEZEICHNER

Da der Automat mit bestimmten Zuständen und bestimmten Eingabe- und Ausgabewerten arbeitet, ist e sinnvoll, dafür eine spezielle Datenstruktur einzuführen. Viele Programmiersprachen kennen dafür eine speziellen **Aufzählungstyp** (**enumeration**). In der Standardsyntax von Python fehlt dieser Datentyp leide er wurde aber in TigerJython mit dem zusätzlichen Schlüsselwort ***enum()*** hinzugefügt. Bei der Definitic der Aufzählungswerte verwendet man Strings. Diese müssen sich an die erlaubte Namensgebung fi Variablen halten.

```
from gconsole import *

def getEvent():
    keyCode = getKeyCodeWait()
    if keyCode == 38: # up
        return Events.stop
    if keyCode == 40: # down
        return Events.work
    if keyCode == 37: # left
        return Events.turnOff
    if keyCode == 39: # right
        return Events.turnOn
    return None

State = enum("OFF", "STANDBY", "WORKING")
state = State.OFF
Events = enum("turnOn", "turnOff", "stop", "work")
makeConsole()
while True:
    gprintln("State: " + str(state))
    event = getEvent()
    if event == Events.turnOn:
        if state == State.OFF:
            state = State.STANDBY
    elif event == Events.turnOff:
            state = State.OFF
    elif event == Events.work:
        if state == State.STANDBY:
            state = State.WORKING
    elif event == Events.stop:
        if state == State.WORKING:
            state = State.STANDBY
```

Programmcode markieren (Ctrl+C kopieren, Ctrl+V einfügen)

MEMO

Es ist Geschmackssache, ob man den zusätzlichen Datentyp *enum* verwenden will. Die Programm werden dadurch nicht kürzer, hingegen übersichtlicher und sicherer, da nur im *enum* definier Aufzählungswerte vorkommen dürfen.

MAUSGESTEUERTE IMPLEMENTIERUNG DES ESPRESSO-AUTOMATEN

Mit nur wenig zusätzlichem Aufwand kannst du mit der GameGrid-Bibliothek den Espresso-Automat grafisch simulieren, wodurch das Programm stark an Anschaulichkeit gewinnt und das Programmieren mehr Spass macht. Statt der Tastatur werden die 4 Eingaben durch Mausklicks auf simulierte Druckknöpfe ausgelöst und die Ausgaben der LED und der Pumpe unmittelbar mit Spritebildern sichtbar gemacht. An Stelle der Eventloop tritt hier der Callback *pressEvent()*, der immer dann aufgerufen wird, wenn mit der Maus auf das Bild geklickt wird. Da du als *GameGrid* ein Gitter mit 7 x 11 Zellen verwendet, kannst du die Klicks auf die Druckknöpfe mit Gitterkoordinaten erfassen.

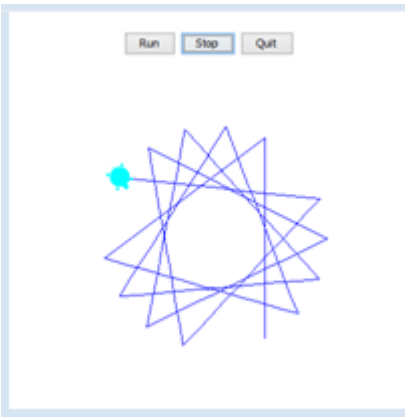
```
from gamegrid import *

def pressEvent(e):
    global state
    loc = toLocationInGrid(e.getX(), e.getY())
    if loc == Location(1, 2): # off
        state = State.OFF
        led.show(0)
        coffee.hide()
    elif loc == Location(2, 2): # on
        if state == State.OFF:
            state = State.STANDBY
            led.show(1)
    elif loc == Location(4, 2): # stop
        if state == State.WORKING:
            state = State.STANDBY
            coffee.hide()
    elif loc == Location(5, 2): # work
        if state == State.STANDBY:
            state = State.WORKING
            coffee.show()
    setTitle("State: " + str(state))
    refresh()

State = enum("OFF", "STANDBY", "WORKING")
state = State.OFF
makeGameGrid(7, 11, 50, None, "sprites/espresso.png", False,
             mousePressed = pressEvent)
show()
setTitle("State: " + str(state))
led = Actor("sprites/lightout.gif", 2)
addActor(led, Location(3, 3))
coffee = Actor("sprites/coffee.png")
addActor(coffee, Location(3, 6))
coffee.hide()
refresh()
```

Programmcode markieren (Ctrl+C kopieren, Ctrl+V einfügen)

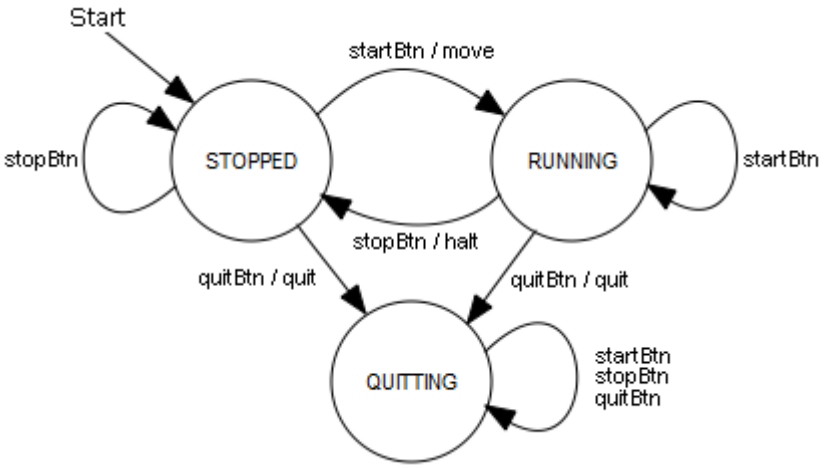
MEMO

Eine Simulation gewinnt durch eine grafische Benutzeroberfläche viel an Anschaulichkeit und Attraktivität.

DENKEN IN ZUSTÄNDEN BEI GRAFISCHEN BENUTZEROBERFLÄCHEN



Auf den ersten Blick scheinen Mealy-Automaten eine eher theoretische Angelegenheit zu sein. Dies ist aber keineswegs so. Vielmehr muss man bei modernen, eventgesteuerten Programmen mit einer grafischen Benutzeroberfläche **immer in Zuständen denken**. Als Beispiel schreibst du ein Turtle-Programm, das über 3 Buttons gesteuert wird: Der *Startbutton* setzt die Turtle in Bewegung, der *Stopbutton* hält die Bewegung an und der *Quitbutton* beendet das Programm. Um das Programm richtig zu implementieren, musst du den Automatengraph im Kopf haben:



Wie du weisst, dürfen in Callbacks von GUI-Events keine Animationen und nur kurz dauernder Cod ausgeführt werden, da der Bildschirm nur am Ende der Funktion neu gerendert wird. Darum schaltest du i den Callbacks der Buttonklicks nur den Zustand um und führst die Bewegung der Turtle im Hauptteil de Programms aus.

Mehr zu diesem Problem erfährst du im Anhang 4: Parallelverarbeitung

```
from javax.swing import JButton
from gturtle import *

def buttonCallback(evt):
    global state
    source = evt.getSource()
    if source == runBtn:
        state = State.RUNNING
        setTitle("State: RUNNING")
    if source == stopBtn:
        state = State.STOPPED
        setTitle("State: STOPPED")
    if source == quitBtn:
        state = State.QUITTING
        setTitle("State: QUITTING")

State = enum("STOPPED", "RUNNING", "QUITTING")
state = State.STOPPED

runBtn = JButton("Run", actionPerformed = buttonCallback)
stopBtn = JButton("Stop", actionPerformed = buttonCallback)
quitBtn = JButton("Quit", actionPerformed = buttonCallback)
makeTurtle()
```

```
setTitle("State: STOPPED")
back(100)

pg = getPlayground()
pg.add(runBtn)
pg.add(stopBtn)
pg.add(quitBtn)
pg.validate()

while state != State.QUITTING and not isDisposed():
    if state == State.RUNNING:
          forward(200).left(127)
dispose()
```

Programmcode markieren (Ctrl+C kopieren, Ctrl+V einfügen)

MEMO

Diese Programmstruktur ist für eventgesteuerte Programme typisch und du solltest sie dir gut merken.

Du musst das Package *JButton* importieren, um die Buttons zu verwenden. Mit *add()* fügst du sie in da Turtlefenster (den Playground). Damit sie sichtbar werden, musst du das Turtlefenster mit *validate()* ne rendern.

AUFGABEN

1. Ein Parkscheinautomat akzeptiert nur 1 € und 2 € Geldstücke, die einzeln hintereinander eingeworfe werden. Sobald der Automat mindestens die Parkgebühr erhalten hat, gibt er den Parkschein und da Restgeld aus. Die Parkgebühr betrage 3 €.

   Der Automat durchlaufe ausgehend vom Startzustand S0 die Zustände S1 oder S2, je nachdem ob 1 oder 2 € eingeworfen werden. Seine Ausgabewerte sind - (nichts), K (Karte) oder K,R (Karte un Rückgeld).

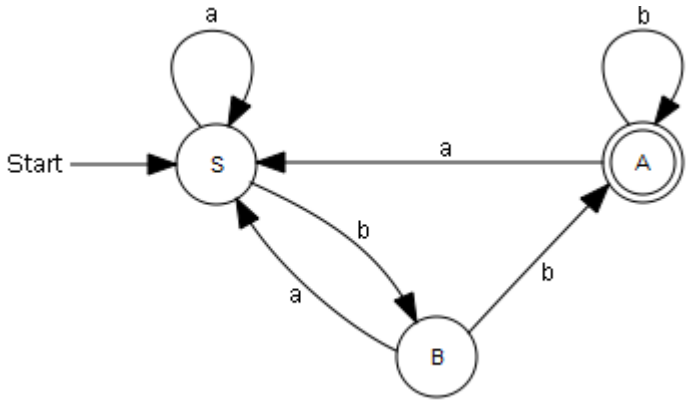
   a. Erstelle die Übergangs- und Ausgabetabellen
   b. Zeichne den Automatengraphen
   c. Erstelle ein Programm mit GConsole, welches das Drücken der Zahlentaste 1 als Einwurf von 1 und das Drücken der Zahlentaste 2 als Einwurf von 2 € interpretiert und den Folgezustand, sowi die Ausgabewerte in das Konsolenfenster ausschreibt.

ZUSATZSTOFF

AKZEPTOR FÜR REGULÄRE SPRACHEN

Eine formale Sprache besteht aus einem Alphabet von Zeichen und einem Regelsystem, mit dem ma eindeutig entscheiden kann, ob eine bestimmte Zeichensequenz mit Zeichen aus diesem Alphabet zu Sprache gehört. Kann man das Regelsystem mit einem Automaten realisieren, so spricht man von eine **regulären formalen Sprache**.

Du betrachtest als Beispiel eine sehr einfache Sprache mit einem Alphabet, das nur aus den Buchstaben und b besteht. Das Regelsystem kannst du als Spezialfall eines Mealy-Automaten auffassen, der kein Ausgabewerte erzeugt. Dabei liest der Automat ausgehend von einem Startzustand Zeichen um Zeiche und geht entsprechend des gelesenen Zeichens in einen Nachfolgezustand über. Befindet er sich nac dem Lesen des letzten Zeichens in einem der vorgegebenen Endzustände, so gehört das Wort zu Sprache. Du betrachtest den folgenden Automatengraphen (S: Startzustand, A: Endzustand):



In der Implementierung wird der Zustandswechsel durch Drücken der Buchstabentasten a oder ausgelöst. Danach schreibt dein Programm den aktuellen Zustand und das bisher eingegebene Wort aus.

```
from gconsole import *

def getKeyEvent():
    global word
    keyCode = getKeyCodeWait(True)
    if keyCode == KeyEvent.VK_A:
        return Events.a
    if keyCode == KeyEvent.VK_B:
        return Events.b
    return None

State = enum("S", "A", "B")
state = State.S
Events = enum("a", "b")
makeConsole()
word = ""
gprintln("State: " + str(state))
while True:
    entry = getKeyEvent()
    if entry == Events.a:
        if state == State.A:
            state = State.S
        elif state == State.B:
            state = State.S
        word += "a"
        gprint("Word: " + word + " -> ")
        gprintln("State: " + str(state))
    elif entry == Events.b:
        if state == State.S:
            state = State.B
        elif state == State.B:
            state = State.A
        word += "b"
        gprint("Word: " + word + " -> ")
        gprintln("State: " + str(state))
```

Programmcode markieren (Ctrl+C kopieren, Ctrl+V einfügen)

MEMO

Ein Akzeptor prüft, ob ein Wort zu einer Sprache gehört. Er ist ein Spezialfall eines Mealy-Automaten ohn Ausgabewerte. Das Wort gehört dann zur Sprache, wenn man ausgehend vom Startzustand S nach de Lesen aller Buchstaben bei einem Endzustand A ankommt. Beispielsweise gehört *abbabb* zur Sprach hingegen *baabaa* nicht.

AUFGABEN

1. Ein Lachautomat soll nur die Wörter *ha.* oder *haha.* oder *hahaha.* usw. (letztes Zeichen ein Punk akzeptieren. Zeichne den Automatengraph und implementiere ihn.
   Anleitung: Du kannst einen Fehlerzustand E einführen, von dem man bei beliebiger Eingabe nicht mel wegkommt.