Institut für Automatik
Proff. M.Morari, W.Schaufelberger, A.Glattfelder
Oliver Kaiser, Urs Baumann, Tobias Geyer

Abt. IIIB, 5./6. Sem.
Nov. 2001
Rev. 4.2

Einführung in Stateflow

Stateflow (SF) ist ein Zusatz zu Matlab/Simulink, der es erlaubt, «Finite-State Machines» darzustellen und zu simulieren. Stateflow benötigt Matlab 5.1 und Simulink 2.1 und existiert nur für die Plattformen Windows und UNIX.

Zustandsautomaten («Finite-State Machines») sind ein wertvolles Werkzeug, um Sequenzsteuerungen zu modellieren. Nach der Modellierung als Zustandsautomaten mussten die Steuerungen aber meist noch mit einer gängigen Programmiersprache implementiert werden. Stateflow erlaubt es jedoch, die grafisch eingegebenen Diagramme direkt in ausführbaren Code umzusetzen.

Diese Einführung ist folgendermassen strukturiert: Nach einigen einleitenden Worten und etwas Theorie wird als Beispiel schrittweise die Steuerung für einen Getränkeautomaten entwickelt.

1 Starten von Stateflow

Um mit Stateflow zu arbeiten, müssen Sie zuerst Matlab starten. Tippen Sie anschliessend `stateflow` ins Command Window, worauf eine Bibliothek mit einem leeren SF-Block, einem Knopf zum Starten der Demos und einem Schalter erscheint. Die Demos sind durchaus empfehlenswert, um einen ersten Eindruck zu erhalten.

Ein Stateflow-System kann nicht für sich allein existieren, sondern ist immer Teil eines Simulink-Modells. Öffnen Sie daher ein leeres Simulink-Modell und ziehen Sie anschliessend aus der SF-Bibliothek das SF-System «Untitled» hinüber.

Das Erstellen eines SF-Systems geschieht auf zwei Ebenen. Einerseits werden mit dem «Editor» die graphischen Elemente «gezeichnet», andererseits werden mit dem «Explorer» die nichtgraphischen Elemente wie Events und Data definiert.

2 Der Editor

Der Editor dient, wie gesagt, der Definition der graphischen Elemente, der Zustände und Transitionen (s. Abb. 1), und wird durch Doppelklicken auf den SF-Block gestartet. Das Beispiel in Abb. 1 hat einen Makro-Zustand `Funktionsgenerator`. Darin sind zwei parallele Zustände, `RechteckGenerator` und `Einzelpuls`, verschachtelt. Die Parallelität (AND) wird durch die gestrichelte Linie visualisiert. Der Zustand `Einzelpuls` besteht aus drei Unterzuständen. Alle Transitionen sind von Bedingungen abhängig («guarded transitions»). Die Bedingungen stehen in eckigen Klammern bei den Transitionen. Die Definition von «Actions» und «Activities» ist beispielsweise im Zustand `Funktionsgenerator.Einzelpuls.Ausgeben` ersichtlich.

- Ein Zustand wird eingefügt, indem man auf das gewünschte Symbol am linken Rand klickt und dieses dann im Diagramm einsetzt.
- Durch Ziehen an den Ecken kann die Grösse der Zustände verändert werden, wodurch eine Verschachtelung der Zustände möglich wird.
- Transitionen können direkt, d.h. ohne Auswahl am linken Rand gezeichnet werden.
- Die Beschriftung der Zustände und Transitionen erfolgt indem man auf die entsprechenden Fragezeichen klickt.
- Ganze Gruppen von Zuständen können verschoben werden, indem man sie vor dem Verschieben mit der Maus selektiert.

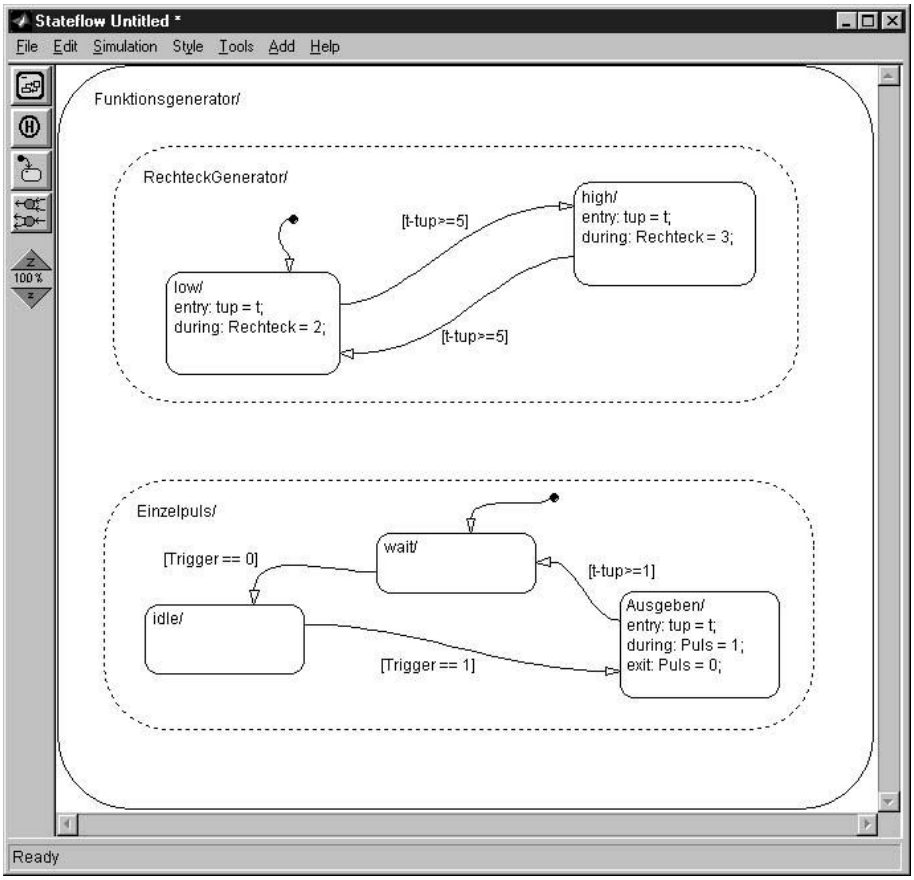

Abbildung 1: Der Stateflow-Editor.

- Durch Drücken der rechten Maustaste erscheint ein Menu mit weiteren Editor-Befehlen.

Hinweis: Nur die Eingabe der in Abb. 1 gezeigten graphischen Elemente führt noch nicht zu einer funktionierenden Zustandsmaschine. Die Elemente `tup`, `Rechteck` und `Puls` müssen zusätzlich noch mit dem Explorer definiert werden.

3 Der Explorer

Indem beim Editor das Menu-Item `Tools.Explore` gewählt wird, erscheint ein Fenster mit dem Stateflow-Explorer (s. Abb. 2). Er dient der Definition von «Events» und «Data».

In Abb. 2 sind auf der linken Seite die Zustände gemäss ihrer Verschachtelung gezeichnet. Durch Doppelklicken auf den Namen können die verschiedenen Zustände expandiert werden. Wenn man einen Zustand einmal anklickt, wird der Text invertiert und rechts erscheinen die dazugehörigen Events und Daten. Mit dem Menu `Add` können weitere Events und Daten hinzugefügt werden.

Bei der Definition von Events und Daten müssen die Sichtbarkeitsregeln (Scope) beachtet werden. Jeder Event oder Data gehört zu dem Zustand («is parented by»), in dem er definiert wurde. Die Sichtbarkeit beschränkt sich auf diesen und die darunterliegenden Zustände. Beim Beispiel in Abb. 1 wurde die Variable `tup` zweimal lokal definiert, einmal unter dem Zustand `RechteckGenerator` und einmal unter dem Zustand `Einzelpuls`, weshalb die beiden Versionen voneinander unabhängig sind. Wäre `tup` unter dem Zustand `Funktionsgenerator` definiert worden, gäbe es zwischen `RechteckGenerator` und `Einzelpuls` ungewollte Seiteneffekte.

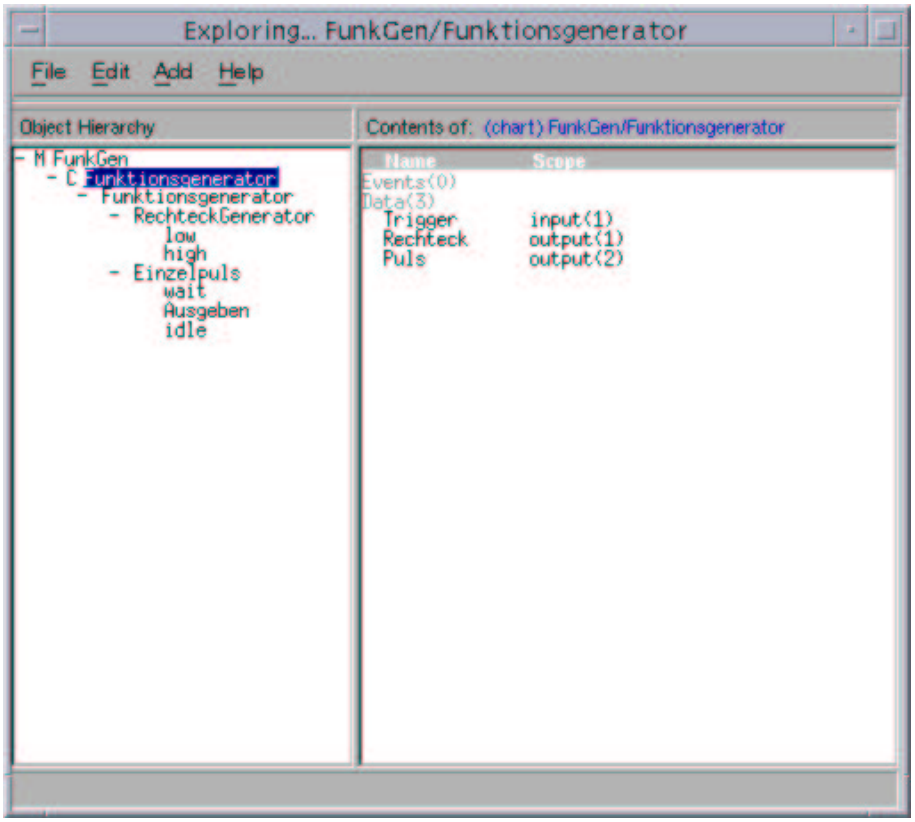

Abbildung 2: Der Stateflow-Explorer.

4 Animation der Zustandsdiagramme

Wenn man ein Simulink-Modell, das einen SF-Block enthält, simuliert, so besteht die Möglichkeit, die Zustandsübergänge des SF-Blocks zu visualiseren. Gehen Sie dazu wie folgt vor:

1. Wählen Sie das Menu-Item `Tools.Debug` und stellen Sie `Animation` auf `enabled`.

2. Wählen Sie das Menu-Item `Tools.Open Simulation Target...`, klicken Sie auf `Coder Options...` und aktivieren Sie `Enable Debugging/Animation`.

Wenn Sie das System jetzt simulieren, wird der aktive Zustand hervorgehoben. Falls Sie nichts erkennen können, weil alles zu schnell abläuft, müssen Sie bei den Simulationsparametern des Simulink-Modells eine kleinere Schrittweite wählen.

5 Simulieren von Zustandsdiagrammen

Durch Drücken von `Simulation.Start` laufen die folgenden Schritte ab:

- Aus dem SF-Diagramm wird zuerst C-Code generiert.

- Der C-Code wird zu einer von Simulink ausführbaren S-Function compiliert.

- Die eigentliche Simulation wird gestartet.

Dieser Vorgang kann einige Zeit beanspruchen (Command Window beachten).

6 Beispiel: Der Getränkeautomat

Im folgenden wird eine einfache Steuerung eines Getränkeautomaten entwickelt. Diese besteht aus zwei parallelen Prozessen: Einem Kühlprozess, der die Kühlung des Automaten periodisch ein- und ausschaltet und einem Prozess, der die Kommunikation mit dem Kunden übernimmt (FrontEnd-Prozess).

Das Durcharbeiten dieses Tutorials benötigt ca. 30 Minuten und macht Sie vertraut mit den wichtigsten SF Elementen. Tippen Sie nun `stateflow` ins Command Window von Matlab. Es wird ein Simulink Modell geöffnet, das nur einen Stateflow Block `Untitled` enthält. Speichern Sie diesen Block bspw. unter dem Dateinamen `Automat` ab.

6.1 Erstellen von parallelen Prozessen

Öffnen Sie das Stateflow Diagramm durch Doppelklicken auf den Block. Unser Automat soll einerseits einen Steuerprozess zur Interaktion mit dem Kunden und andererseits einen Kühlprozess enthalten. Beide sollen parallel ablaufen. Erzeugen Sie zwei Zustände und beschriften Sie diese wie in Abb. 3 dargestellt. Der «/» am Namensende ist optional. Parallelisieren Sie die Zustände, indem Sie ausserhalb von diesen mit der rechten Maustaste in das Editor Window klicken und `Decomposition.Parallel(AND)` wählen. Daraufhin erscheinen die Zustandsgrenzen gestrichelt.

Abbildung 3: Die parallelen Prozesse des Automaten.

6.2 Der Kühler-Prozess

Zunächst soll das Zustandsdiagramm für den Kühler erstellt werden. Zeichnen Sie dafür zwei neue Zustände innerhalb des Zustandes `Kuehler/`. Fügen Sie anschliessend die in Abb. 4 angegebenen Transitionen ein. Die Transition mit einem Punkt am Anfang ist die «default Transition», die angibt, in welchen Zustand der Automat beim Starten des Diagramms springt. Sie wird durch den drittobersten Button auf der linken Seite des Editorfensters erzeugt. Die übrigen Transitionen werden direkt, d.h. ohne Auswahl auf der linken Seite erzeugt. Beschriften Sie die Zustände und Transitionen wie in Abb. 4.

Die Aktionen in den Zuständen lassen sich je nach deren gewünschtem Ausführzeitpunkt einteilen durch die Label `entry`, `during`, `on` *eventname* und `exit`. Wird kein Label gesetzt, wird `entry` angenommen, d.h. die entsprechenden Aktionen werden ausgeführt sobald der Zustand aktiv wird. In unserem Beispiel wird bspw. die Variable `Kuehlen` zu Beginn auf 0 gesetzt und beim Verlassen von `Warten/` auf 1.

Der Wechsel von einem Zustand in den nächsten kann mit «Events» und «Conditions» kontrolliert werden:

- Will man, dass beim Auftreten eines Events ein Zustandsübergang stattfindet, schreibt man den Eventnamen in den gewünschten Zustandsübergang.
- Will man den Zustandsübergang an eine logische Bedingung knüpfen, wird die Bedingung in eckigen Klammern in den gewünschten Zustandsübergang geschrieben, wie dies in Abb. 4 zu sehen ist.

Mit der Variablen `t` steht die Simulationszeit des Simulink Modells in Stateflow zur Verfügung.

Der Prozess `Kuehler/` bleibt während 10 Sekunden im Zustand `Warten/` (`Kuehlen = 0`) und schaltet danach den Kühler 5 Sekunden lang ein, d.h. der Zustand `Kuehlen/` ist aktiv.

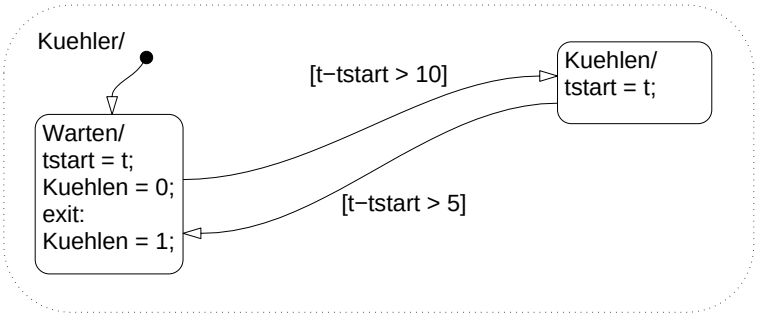

Abbildung 4: Der Prozess `Kuehler/` des Automaten.

6.3 Der FrontEnd-Prozess

Erstellen Sie nun die Transitionen und den Zustand `Warten/` im Zustand `FrontEnd/` wie in Abb. 5 gezeigt. Die «Junction» (kleiner Kreis) wird mit dem viertobersten Button auf der linken Seite des Editor Windows erzeugt.

Der Zustand `Warten/` wartet auf die Ereignisse `Muenzeeingeworfen` und `Geldzurueck`. Tritt eines dieser Ereignisse auf, wird die im Automaten vorhandene Geldsumme aufdatiert. Die Initialisierung von Variablen geschieht über den Explorer (vgl. weiter unten). Sobald die Summe im Automaten grösser als der Kaufpreis für ein Getränk ist, kann durch Erzeugen eines Ereignisses `Getraenk1` resp. `Getraenk2` ein Getränk herausgelassen werden. Dargestellt wird dies durch die Variable `Getraenkausgabe`.

Im folgenden soll der interne Ablauf des `FrontEnd/` Prozesses genauer erläutert werden. Beim Starten der Simulation wird über die «default Transition» der `Warten/` Zustand aktiviert. Es werden dann dauernd die Bedingungen für die Transitionen aus dem Zustand `Warten/` überprüft, und zwar ausgehend von 12 Uhr im Uhrzeigersinn. Dadurch ergibt sich folgende Reihenfolge:

1. Ist `Summe >= Preis`? Wenn ja, ist das Ereignis `Getraenk1` oder `Getraenk2` eingetreten?

2. Ist das Event `Muenzeeingeworfen` eingetreten?

3. Ist das Event `Geldzurueck` eingetreten?

Ist der erste Punkt erfüllt, wird der Zustand `Warten/` verlassen (nachdem allfällige `exit` Statements ausgeführt worden sind). Danach wird `Summe -= Preis` gesetzt (C-Syntax) und der Variablen `Getraenkausgabe` der Wert 1 resp. 2 zugewiesen. Anschliessend wird in den Zustand `Warten/` gewechselt. Ist der zweite resp. dritte Punkt erfüllt, wird `Summe++` bzw. `Summe = 0` gesetzt und zwar bevor der Zustand `Warten/` verlassen wird (und allfällige `exit` Statements ausgeführt worden sind). Danach wird wiederum in den Zustamd `Warten/` gewechselt.

Zur Syntax seien folgende Punkte erwähnt.

1. Transitionstexte in geschweiften Klammern {} werden direkt, nachdem eine Transition wahr geworden ist, ausgeführt. Sie werden auch ausgeführt, wenn nur der erste Teil einer Transition wahr ist. Wäre bspw. in Abb. 5 der Term `Summe -= Preis` in geschweiften Klammern, würde er ausgeführt, sobald `Summe >= Preis` unabhängig davon, ob das Event `Getraenk1` oder `Getraenk2` eingetreten ist.

2. Transitionstexte, die mit einem / beginnen, werden erst nach dem Verlassen eines Zustandes ausgeführt.

3. Die Junction, d.h. der Kreis in Abbildung Abb. 5 ist kein Zustand. Folglich verweilt der Prozess auch nicht an dieser Stelle.

4. Mit drei Punkten am Ende einer Zeile kann der Transitionstext auf mehrere Zeilen verteilt werden.

5. Die Zuweisungssyntax orientiert sich an der Sprache C.

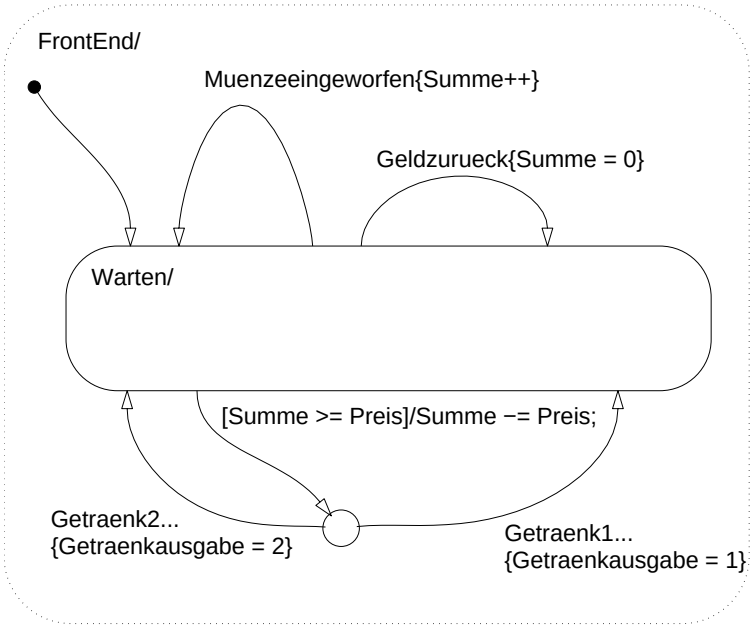

Abbildung 5: Der FrontEnd Prozess des Automaten.

6.4 Definition von Variablen und Ereignissen

Zur Deklaration der verwendeten Variablen (Data) und Ereignisse (Events) wählen Sie `Tools.Explore`. In der Spalte «Object Hierarchy» können Sie sich durch die hierarchisch geordneten Zustände des Getränkeautomaten klicken. Gehen Sie auf die Stufe `Chart`. Fügen Sie mit `Add.Event` folgende Events hinzu und wählen Sie als `Scope` jeweils `Input from Simulink`.

1. `Muenzeeingeworfen`
2. `Geldzurueck`
3. `Getraenk1`
4. `Getraenk2`
5. `Takt`

Damit haben Sie 5 Ereignisse definiert, die aus Simulink in das Stateflow Diagramm eingespiesen werden können. Der entsprechende Eingangsport wird auf der Oberseite des Simulink Blocks `Automat` ersichtlich.

Das Event `Takt` wird gebraucht, um das Stateflow Diagramm regelmässig zu aktivieren. Ein solcher Takt ist nur nötig, falls man externe Events (Input from Simulink) definiert hat, ansonsten wird die Taktrate des Simulink Modells übernommen.

Als nächstes werden die Daten, welche im ganzen Stateflow Diagramm Gültigkeit haben, deklariert. Fügen Sie mit `Add.Data` folgende Variablen (auf Stufe `Chart`) hinzu, und wählen Sie als `Scope` jeweils `Output to Simulink`.

1. `Summe`

2. `Getraenkausgabe`

3. `Kuehlen`

Auf der rechten Seite des Simulink Blocks erscheinen drei Ausgangsports. Alle Daten und Events, welche mit dem Simulink Modell kommunizieren, müssen auf der Stufe Chart deklariert werden. Wollte man z.B. die Variable `Kuehlen` nicht in Simulink sichtbar machen, könnte man sie auch lokal im Zustand `Kuehler/` deklarieren.

Zum Schluss müssen noch die lokalen Variablen der Prozesse `FrontEnd/` und `Kuehler/` deklariert werden. Gehen Sie auf Stufe `FrontEnd`, wählen Sie `Add.Data` und geben Sie den Namen `Preis` ein. Der `Scope` ist `Constant` und der `Initial Value` 3. Deklarieren Sie nun noch die Variable `tstart` auf Stufe `Kuehler` mit dem `Scope Local`.

6.5 Einbindung des Zustandsautomaten in Simulink

Zum Schluss ist das Stateflow Diagramm noch in das Simulink Modell zu integrieren. Geben Sie dazu `simulink` in das Matlab Command Window ein, um die Simulink Library zu öffnen. Stellen

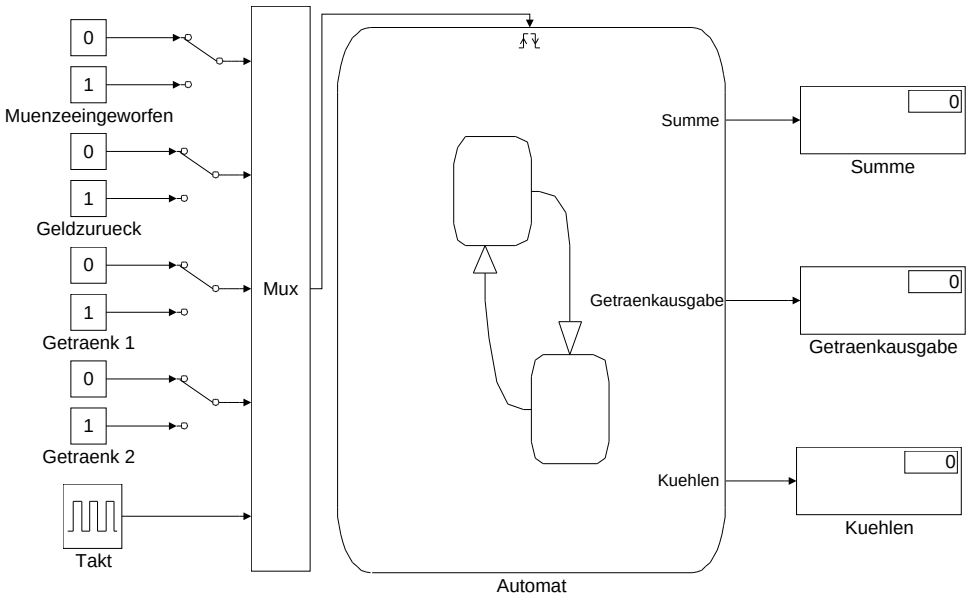

Abbildung 6: Das Simulink Modell des Getränkeautomaten mit integrierter Stateflow Steuerung.

Sie nun das Modell zusammen wie in Abb. 6 dargestellt. Die Displays für die Ausgänge sind unter `Sinks` zu finden. Der `Mux` ist unter `Signals & Systems` und die Schalter unter `Nonlinear` abgelegt. Schliesslich finden Sie die 0 und 1 Blöcke (`Constant`) sowie den `Takt` Block (`Pulse Generator`) unter `Sources`. Die Parametereinstellungen der Blöcke können durch Doppelklicken auf diese verändert werden. Öffnen Sie den Block `Takt` und ändern Sie `Period (secs)` zu 0.1. Die Namen von einzelnen Blöcken können ausgeschaltet werden, indem man den Block selektiert und `Format.Hide Name` wählt.

Zum Schluss legen Sie die Simulationsparameter über das Simulink Window `Simulation.Parameters...` fest. Stellen Sie die `Stop time` auf 1000000. Vergessen Sie nicht, Ihren Automaten von Zeit zu Zeit abzuspeichern. Über `Simulation.Start` kann nun der Getränkeautomat gestartet werden. Durch Doppelklicken auf die Schalter im Simulink Modell können Events, wie bspw. der Einwurf einer Münze oder die Aufforderung für die Ausgabe eines Getränkes, generiert werden. Der Zustand des Automaten und die Getränkeausgabe können auf den entsprechenden Displays abgelesen werden.

7 «Inner Transitions»

Eine «Inner Transition» ist eine Transition, die den Ausgangszustand nicht verlässt. Dieses Konzept ist sehr mächtig und kann Stateflow-Diagramme stark vereinfachen. Insbesondere beim Versuch A15 (Verkehrskreuzung) leistet es wertvolle Dienste bei der Handsteuerung. Beim Chügelimat gelangt es nicht zum Einsatz.

Betrachten Sie das Diagramm in Abb. 7, das ohne «Inner Transition» auskommt. Nach der Aktivierung des Diagramms wird abhängig von den Bedingungen `C_one` und `C_two` in die Zustände `A1` bis `A3` verzweigt. Anschliessend wird auf das Auftreten des Events `E_one` gewartet und abhängig von den Bedingungen `C_one` und `C_two` zwischen den Zuständen gewechselt.

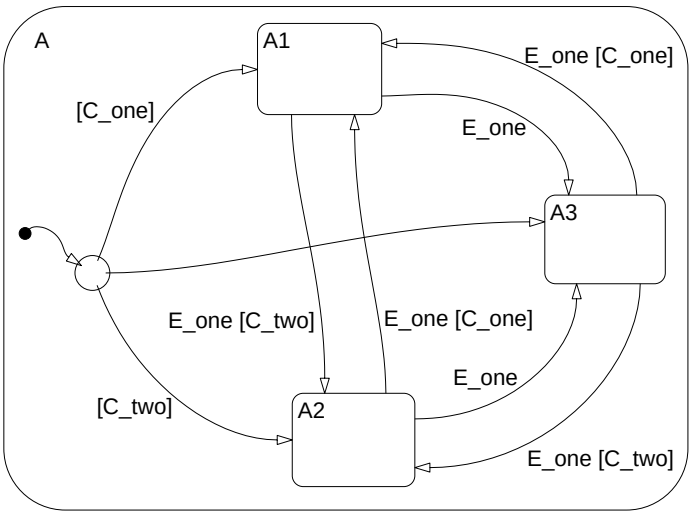

Abbildung 7: Stateflow-Diagramm ohne «Inner Transition»

Die gleiche Funktionalität besitzt auch das Diagramm in Abb. 8. Auch hier wird nach der Aktivierung des Diagramms abhängig von den Bedingungen `C_one` und `C_two` in die Zustände `A1` bis `A3` verzweigt. Wenn anschliessend das Event `E_one` auftritt, gelangt man über die am Rand des Zustandes `A` beginnende «Inner Transition» wieder zur Junction, wo abhängig von `C_one` und `C_two` verzweigt wird. Unabhängig vom Ausgangszustand `A1` bis `A3` wird immer wieder die gleiche Transition durchlaufen.

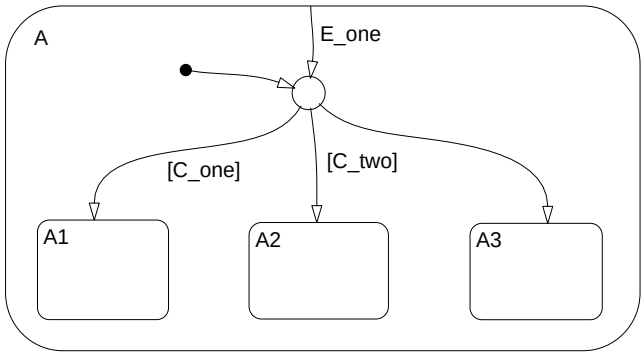

Abbildung 8: Stateflow-Diagramm mit «Inner Transition»

Wichtig ist dabei folgendes: befindet sich das System beispielsweise im Zustand `A3` und das Event `E_one` tritt auf während `C_one` und `C_two` beide falsch sind, dann *bleibt* das System im Zustand `A3`. Es geht nicht etwa aus dem Zustand hinaus und kehrt in diesen zurück. Insbesondere werden allfällige `entry`- und `exit`-Actions von `A3` nicht ausgeführt.

8 Syntax

In diesem Kapitel werden die wichtigsten Syntaxelemente zusammenfassend aufgelistet.

8.1 Zustände

Zustandsname/	Name des Zustandes. Dieser muss zuoberst im Zustand stehen. Der Schrägstrich ist optional.
`entry:`	Die nach dem Schlüsselwort `entry` stehenden Aktionen werden beim Eintritt in den Zustand ausgeführt.
`during:`	Die nach dem Schlüsselwort `during` stehenden Aktionen werden während des Aufenthaltes im Zustand ausgeführt.
`exit:`	Die nach dem Schlüsselwort `exit` stehenden Aktionen werden beim Verlassen des Zustand ausgeführt.

Darüber hinaus können in Zuständen Events gesetzt und abgefragt werden:

eventname	Setzt das Event *eventname*. Hinweis: Wird das Event nicht im selben Moment abgefragt, geht es 'verloren'. D.h. Events werden nicht auf einem Stack abgelegt und dann bei der Abfrage vom Stack wieder entfernt!
`on` *eventname*:	Tritt das Event *eventname* ein, so werden die nach `on` *eventname* aufgeführten Aktionen ausgeführt.

8.2 Transitionen

Transitionen werden durchlaufen, falls bestimmte Bedingungen erfüllt oder bestimmte Ereignisse eingetreten sind. Davon abgesehen können auch Aktionen ausgeführt und Ereignisse ausgelöst werden:

`[cond]`	Die Transition wird erst durchlaufen, wenn die Bedingung `[cond]` erfüllt ist. Solch eine Bedingung lautet beispielsweise `[varname == 5]`. Hier wird die Transition erst durchlaufen, wenn die Variable `varname` gleich 5 ist. Anmerkung: Bedingungen stehen immer in eckigen Klammern.
`[cond]{action}`	Sollen, nachdem die Bedingung `[cond]` erfüllt wurde, eine odere mehrere Aktionen `action` ausgeführt werden, so lassen sich diese in geschweiften Klammern an die Bedingung anfügen. Anmerkung: Eine Aktion ist beispielsweise die Zuweisung eines Skalares zu einer Variablen: `varname = 5` setzt die Variable `varname` auf 5.
`/action`	Bei Durchlaufen der Transition wird die Aktion `action` ausgeführt.
`/event1`	Bei Durchlaufen der Transition wird das Ereignis `event1` ausgelöst. Hinweis: Wird das Event nicht im selben Moment abgefragt, geht es 'verloren'. D.h. Events werden nicht auf einem Stack abgelegt und dann bei der Abfrage vom Stack wieder entfernt!
`event1`	Die Transition wird erst durchlaufen, wenn das Ereignis `event1` eintritt.

8.3 Operatoren

Die Syntax orientiert sich an der Programmiersprache C:

`=`	Zuweisung.
`==`	Test auf Gleichheit.
`!=`	Test auf Ungleichheit.
`~=`	Test auf Ungleichheit.
`a -= b`	a wird auf $a - b$ gesetzt.
`a++`	a wird um 1 erhöht.
`&&`	Modulo-Operator.
`&`	Logisches AND.
`|`	Logisches OR.