13
Automaten und formale Sprachen

Jörn Loviscach

Versionsstand: 2. Januar 2011, 14:43

Die nummerierten Felder sind absichtlich leer, zum Ausfüllen in der Vorlesung.
Videos dazu: http://www.youtube.com/joernloviscach

1 Endliche Automaten

Viele Systeme lassen sich mit Hilfe von Zuständen [states] und Übergängen [transitions] zwischen den Übergängen beschreiben. Typischerweise gibt man für jeden Übergang auch die Bedingung [condition] oder das Ereignis [event] an, wann er zu nehmen ist. Das geht bei der Ampel

[1]

ebenso wie beim Handy.

[2]

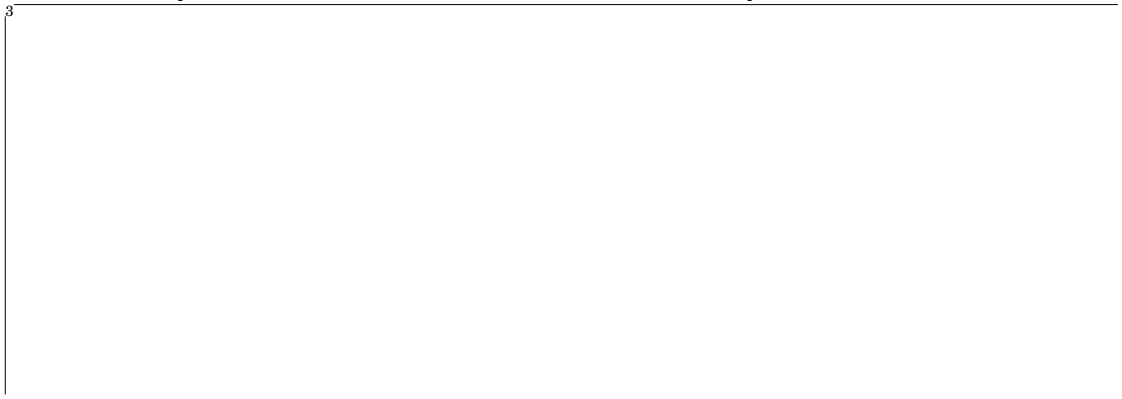
Klarer Anwendungsfall: Steuerungstechnik.

Solche Modelle haben typischerweise nur endlich viele Zustände. Außerdem enthalten die Bedingungen meist keinen Zufall, so dass immer eindeutig bestimmt (determiniert) ist, welchen Weg das System nehmen muss. Beide Eigenschaften

zusammen machen einen deterministischen endlichen Zustandsautomaten aus [deterministic finite state machine, DFA: deterministic finite automaton].

Eine Art Zustandsautomaten darzustellen und zu programmieren, sind Übergangstabellen [state transition tables]. Ein Typ davon listet alle Zustände auf der x-Achse und auf der y-Achse. Auf jedem Kreuzungspunkt steht, unter welcher Bedingung – wenn überhaupt – ein Übergang vom Zustand auf der x-Achse zu dem auf der y-Achse stattfindet. Das Modell des Handys sieht damit so aus:

[3]

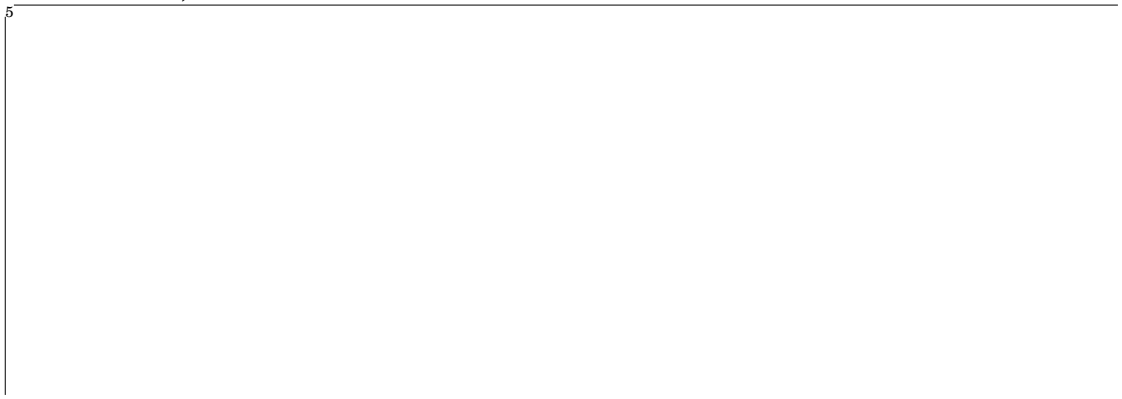


Die Norm UML (Unified Modeling Language) kennt diverse Typen an Diagrammen, darunter das Zustandsdiagramm [state diagram]. Das ist die heute übliche Art, Zustandsautomaten aufzuzeichnen. Hier einige der vielen in UML standardisierten Elemente:

[4]

Ein UML-Zustandsdiagramm erlaubt auch verschachtelte und/oder parallele Abläufe. (Diese könnte man im Prinzip auch mit gewöhnlichen Zustandsautomaten modellieren – wenn man ein Gestrüpp an Hilfszuständen und -übergängen in Kauf nimmt.)

[5]



MATLAB® *Stateflow* kann das praktisch genau so umsetzen. Demo. Achtung: Nicht mit MATLAB® *Simulink* verwechseln! Das ist eine Diagrammsprache für Datenfluss. Dort stehen die Knoten nicht für Zustände, sondern erzeugen,

verarbeiten oder empfangen Signale. Allerdings können Stateflow und Simulink auch zusammenarbeiten: Das Simulink-Modell schickt Ereignisse an Stateflow; Stateflow stellt Parameter des Simulink-Modells ein.

2 Formale Sprachen

Bei Sprachen in der Informatik wie in der Linguistik kann man mindestens zwei Ebenen der Untersuchung unterscheiden:

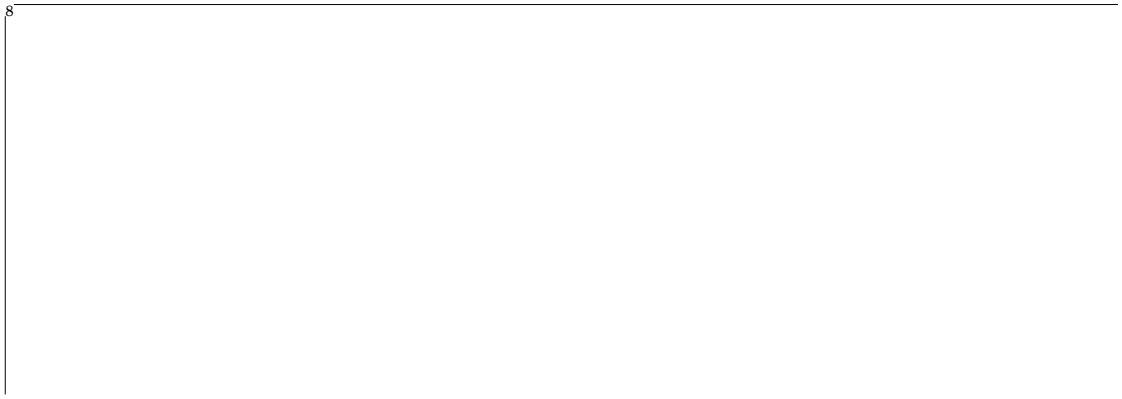
[6]

Eine formale Sprache ist anders als eine natürliche Sprache eine Sprache mit einer strengen mathematischen Definition – allerdings zunächst nur von Syntax und Grammatik; die Semantik wird oft ausgelagert. Einige Beispiele für formale Sprachen:

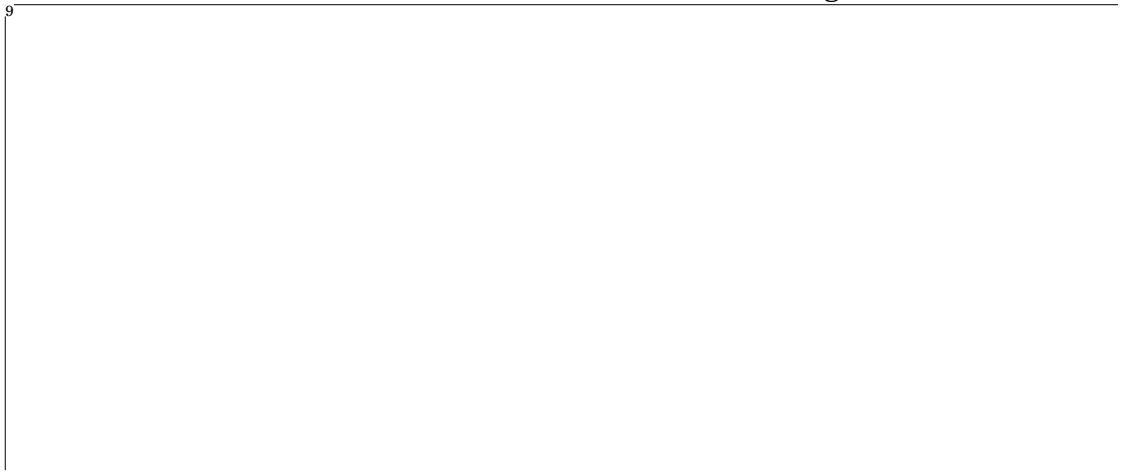
[7]

Streng mathematisch ist eine formale Sprache die Menge aller erlaubten endlichen Zeichenketten aus einem endlichen Alphabet. Bei den interessanten Sprachen sind unendlich viele Zeichenketten erlaubt. (Für welche der Beispiele eben gilt das?) Diese Sprachen kann man dann nicht mehr durch Aufschreiben aller erlaubten Zeichenketten beschreiben. Vielmehr muss man Regeln angeben, wie im Latein-Sprachlehrbuch.

Die Regeln für kompliziertere Sprachen lassen sich mit verschiedenen Methoden aufschreiben. Zum Beispiel mit einem Syntaxdiagramm wie hier für die Regel, wie man Gleitkomma-Zahlen bilden darf:

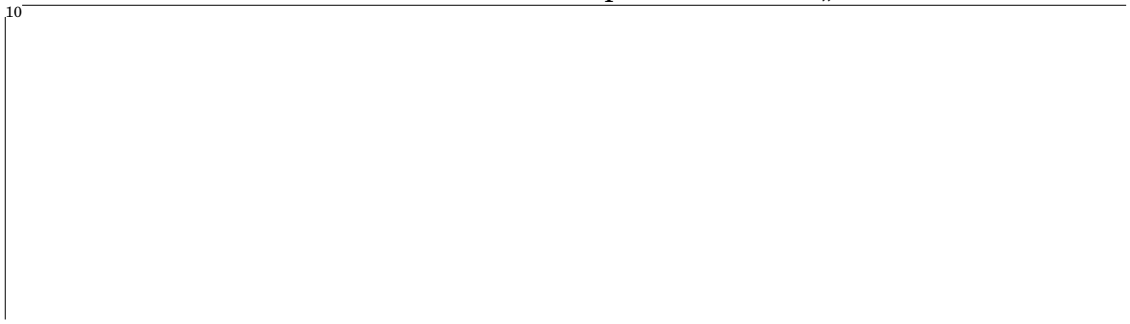
[8]


Vorsicht: Worin besteht der Unterschied zum Diagramm eines Zustandsautomaten? Man kann aber einen Zustandsautomaten aufzeichen, der genau dann vom Start bis zu einem „akzeptierenden“ Zustand durchlaufen wird, wenn man die Zeichen einer Gleitkomma-Zahl und nichts anderes als Ereignisse einfüttert:

[9]


Sprachen, die auf diese Weise in endliche deterministische Zustandsautomaten übersetzt werden können, heißen regulär. Dieser Zustandsautomat kann dann auch direkt zum Übersetzen oder Ausführen der jeweiligen Sprache verwendet werden.

Die aktuell wichtigsten formalen Sprachen sind HTML (Hypertext Markup Language) und die Sprachen auf Basis von XML (Extensible Markup Language). Wesentlicher Bestandteil von allen diesen Sprachen ist das „Element“:

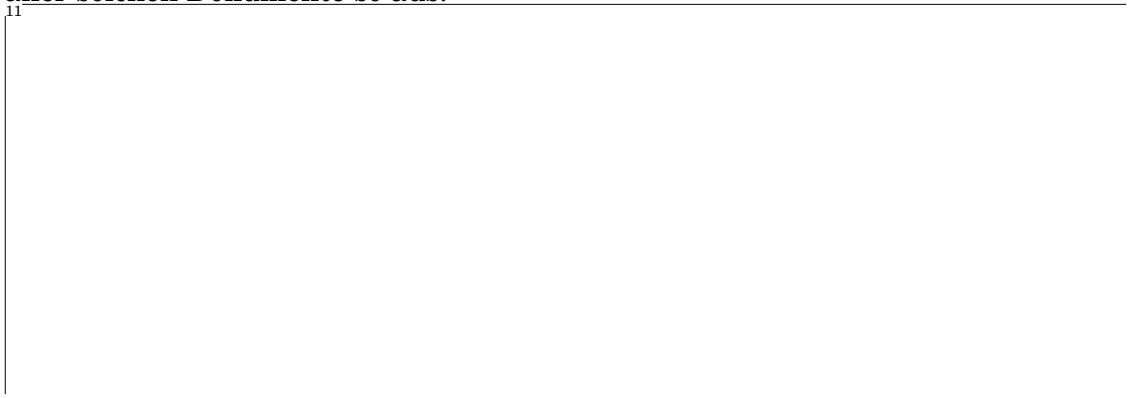
[10]


Diese werden verschachtelt, um stufenweise immer größere Konstrukte zu bilden. Dabei ist in der jeweiligen Sprachdefinition (wahlweise „Document Type Definition“ oder „Schema“) festgelegt, welche Tags mit welchen Attributen es gibt und wie sie verschachtelt werden können bzw. müssen.

So kann das in HTML aussehen:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                        "http://www.w3.org/TR/html4/loose.dtd">
<html>
 <head>
  <title>Eine geniale Seite</title>
 </head>
 <body>
  <h1>Eine &Uuml;berschrift</h1>
  <p>blubb blubb blubb blubb blubb blubb</p>
  <p>blubb blubb blubb blubb blubb blubb</p>
  <h1>Noch eine &Uuml;berschrift</h1>
  <p>blubb blubb blubb blubb blubb blubb</p>
 </body>
</html>
```

Demo: XML in Dokumentdateien von OpenOffice.org. Abstrakt sieht die Struktur aller solchen Dokumente so aus:[11]



Welche der klassischen Datenstrukturen ist das? Demo: Ansicht im Microsoft Internet Explorer.

XML-basierte Sprachen sind sehr speicherplatzfressend (Deshalb das Zip in OpenOffice.org!) und oft zu raffiniert. Deshalb verwendet man für kleinere Aufgaben oft schlankere Sprachen, insbesondere JSON (JavaScript Object Notation):

```
{
  "Titel": "Info-Vorlesung",
  "Datum": "2011-01-06",
  "Uhrzeit": "14:00",
  "Location":
  {
    "Adresse": "Wilhelm-Bertelsmann-Str. 10",
    "Ort": "Bielefeld",
    "PLZ": "33602"
  },
  "Teilnehmer":
  [
    {
    "Nachname": "Loviscach",
      "Vorname": "Jörn"
```

```
    },
    {
      "Nachname": "blabla",
      "Vorname": "blubb"
    }
  ]
}
```

Demo mit den Entwicklertools von Microsoft Internet Explorer, in eine HTML-Seite eingebettet mit `<script type="text/javascript">` und `</script>`.