

# Informatik I: Einführung in die Programmierung

## 18. Automaten, Zustandsmodellierung mit abstrakten und generischen Klassen

Albert-Ludwigs-Universität Freiburg



**UNI  
FREIBURG**

Prof. Dr. Peter Thiemann

24. Januar 2023

- Motivierendes Beispiel
- Formale Grundlagen
- Verhalten eines DEAs
- Teilwort-Erkennung

Deterministische  
endliche  
Automaten

Motivierendes  
Beispiel  
Formale  
Grundlagen  
Verhalten eines  
DEAs  
Teilwort-Erkennung

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

Vor kurzem fand ich auf dem Dachboden eines Kollegen einen Würfel und einen Brief.



*Lieber Finder dieses Briefes,  
der Würfel ist gefallen:  
5-1-5-6 ergibt  $x$ ,  
5-4-2-5 ergibt  $y$ .*

*Bei N 48° 00,  $x$ ' O 7° 50,  $y$ ' in  
15 Meter Tiefe wirst du  
einen sagenhaften Schatz  
finden, der das  
Bernsteinzimmer wie eine  
Studentenbude aussehen  
läßt. Ich wünsche Dir viel  
Glück bei deiner Suche !*

*Emil Nebel  
Freiburg, 1980*

Deterministische  
endliche  
Automaten

Motivierendes  
Beispiel

Formale  
Grundlagen

Verhalten eines  
DEAs

Teilwort-Erkennung

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

- In dem Würfel gibt es einen Mechanismus, der die **Abfolge** der nach oben gerichteten Würfelseiten **erkennt**.
- Nachdem die richtige Folge „gewürfelt“ wurde, klopft von innen ein Hämmerchen die Koordinaten.
- Wie erkennt der Würfel solche Folgen von Ereignissen?
- Hierfür ist ein **endlicher Automat** geeignet.
- Ein endliche Automat ist ein sehr einfaches und eingeschränktes **Berechnungsmodell**, das aber oft adäquat ist.

Deterministische  
endliche  
Automaten

Motivierendes  
Beispiel

Formale  
Grundlagen

Verhalten eines  
DEAs

Teilwort-Erkennung

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

- Ein **Alphabet**  $\Sigma$  ist eine endliche, nicht-leere Menge von Symbolen.
- Ein **Wort** über einem Alphabet  $\Sigma$  ist eine Folge von Zeichen aus  $\Sigma$ .
- $\Sigma^*$  ist die Menge aller Worte über  $\Sigma$ .
- Eine (formale) **Sprache**  $L \subseteq \Sigma^*$  ist eine beliebige (endliche oder unendliche) Menge von Worten.

## Beispiel

- Alphabet  $\Sigma = \{1, 2, 3, 4, 5, 6\}$ , die Würfelseiten.
- Beispielworte: 5156, 4611, usw.

Deterministische  
endliche  
Automaten

Motivierendes  
Beispiel

Formale  
Grundlagen

Verhalten eines  
DEAs

Teilwort-Erkennung

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

Ein **deterministischer Automat** ist ein Quintupel  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ . Dabei ist

- $Q$  eine Menge von **Zuständen**,
- $\Sigma$  das **Eingabealphabet**,
- $\delta : Q \times \Sigma \rightarrow Q$  die **Übergangsfunktion**,
- $q_0 \in Q$  der **Anfangszustand**,
- $F \subseteq Q$  die Menge der (akzeptierenden) **Endzustände**.

## Bemerkung

Falls  $Q$  endlich ist, so ist  $A$  ein **deterministischer endlicher Automat** (DEA).

Deterministische  
endliche  
Automaten

Motivierendes  
Beispiel

Formale  
Grundlagen

Verhalten eines  
DEAs

Teilwort-Erkennung

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

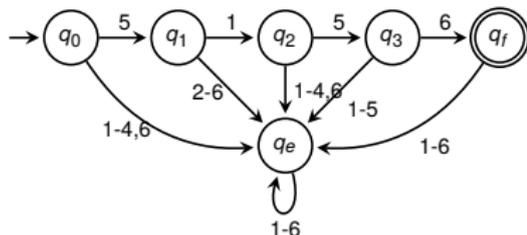
Zusammenfassung &  
Ausblick

Die Übergangsfunktion eines DEA wird entweder durch eine **Übergangstabelle** oder durch ein **Übergangsdiagramm** (Transitionsdiagramm) angegeben.

**Beispiel: Übergangsfunktion zur Erkennung von 5156**

( $q_e$  bezeichnet einen Fehlerzustand und  $F = \{q_f\}$ ).

	1	2	3	4	5	6
$q_0$	$q_e$	$q_e$	$q_e$	$q_e$	$q_1$	$q_e$
$q_1$	$q_2$	$q_e$	$q_e$	$q_e$	$q_e$	$q_e$
$q_2$	$q_e$	$q_e$	$q_e$	$q_e$	$q_3$	$q_e$
$q_3$	$q_e$	$q_e$	$q_e$	$q_e$	$q_e$	$q_f$
$q_f$	$q_e$	$q_e$	$q_e$	$q_e$	$q_e$	$q_e$
$q_e$						



**Beachte:** Im Übergangsdiagramm wird der **absorbierende Fehlerzustand**  $q_e$  mit allen Übergängen dorthin in der Regel nicht angegeben.

Deterministische  
endliche  
Automaten

Motivierendes  
Beispiel

Formale  
Grundlagen

Verhalten eines  
DEAs

Teilwort-Erkennung

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

- Anfänglich befindet sich der Automat im **Startzustand**  $q_0$ .
- Der Automat erhält ein Wort  $w = a_1 a_2 \dots a_n$  über  $\Sigma$  als **Eingabe** (darf auch leer sein, d.h.  $n \geq 0$ ).
- Der Automat liest im Zustand  $q_i$  das **Eingabesymbol**  $a_{i+1}$  und wechselt in den **Folgezustand**  $q_{i+1} = \delta(q_i, a_{i+1})$ .
- Das macht der Automat, so lange Eingabezeichen gelesen werden können, das heißt für  $i \in \{0, 1, \dots, n-1\}$ .
- Am Ende der Eingabe befindet sich der Automat in einem Zustand  $q_n$ . Das Eingabewort  $w$  wird genau dann **akzeptiert**, wenn  $q_n \in F$  ein **Endzustand** ist.
- Die **Sprache von A**,  $\mathcal{L}(A)$ , ist die Menge aller von A akzeptierten Worte.

Deterministische  
endliche  
Automaten

Motivierendes  
Beispiel  
Formale  
Grundlagen

Verhalten eines  
DEAs

Teilwort-Erkennung

Transduktoren

Modellierung  
von  
Automaten

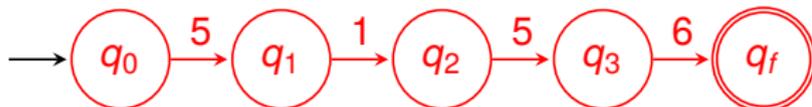
Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

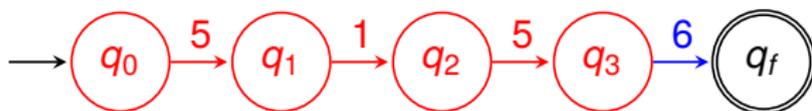
Eingabe: 5156 5156 156 156 56 56 6 6

Eingabe akzeptiert



Eingabe: 515156 515156 15156 15156 5156 5156 156 156  
156

Kein Übergang von  $q_3$  aus möglich! Eingabe nicht **akzeptiert**.



Deterministische  
endliche  
Automaten

Motivierendes  
Beispiel  
Formale  
Grundlagen

Verhalten eines  
DEAs

Teilwort-Erkennung

Transduktoren

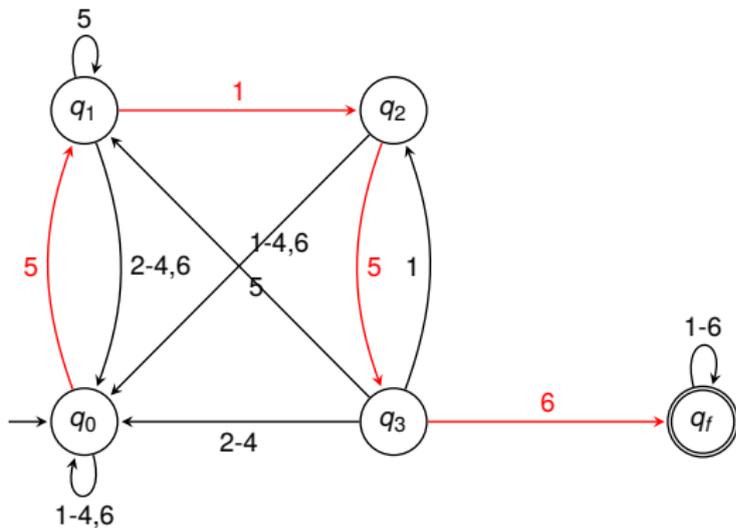
Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

Der Würfel soll letztlich alle Folgen akzeptieren, die 5156 als **Teilwort** enthalten, z.B. auch 55156, oder 5155156, oder 515156 oder ...5156...



Deterministische  
endliche  
Automaten

Motivierendes  
Beispiel  
Formale  
Grundlagen

Verhalten eines  
DEAs

Teilwort-Erkennung

Transduktoren

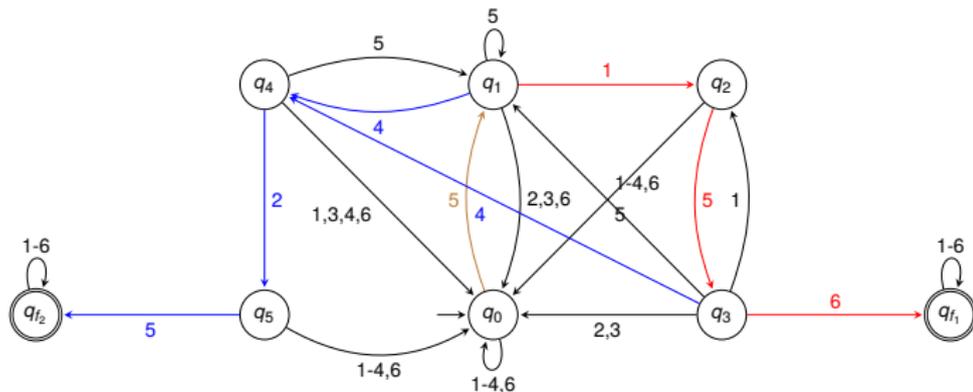
Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

Zusätzlich muss der Würfel noch 5425 als Teilstring erkennen!  
Der Automat ändert sich wie folgt:



Deterministische  
endliche  
Automaten

Motivierendes  
Beispiel  
Formale  
Grundlagen  
Verhalten eines  
DEAs

Teilwort-Erkennung

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

- Moore-Automat
- Umsetzung
- Python-Skript für Beispiel

Deterministische  
endliche  
Automaten

Transduktoren

Moore-Automat  
Umsetzung  
Python-Skript für  
Beispiel

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

# Nach der Akzeptanz ist vor der Akzeptanz!



- Der bisher konstruierte Automat akzeptiert alle Wörter, die 5156 oder 5425 als Teilwort enthalten.
- Eigentlich ist eine Maschine gesucht, die „ewig“ läuft und die jeweils nach einem akzeptierten Teilwort eine **Ausgabe** macht.
- Eine solche Maschine heißt **Transduktor**, das ist ein Automat, der auch Ausgaben macht.
- Transduktoren werden oft verwendet um das Verhalten **eingebetteter Systeme** zu beschreiben.

Deterministische  
endliche  
Automaten

Transduktoren

Moore-Automat  
Umsetzung  
Python-Skript für  
Beispiel

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

Ein **Moore-Automat** (nach Edward F. Moore) ist ein endlicher Automat, der in jedem Zustand ein Zeichen ausgeben kann.

Es ist ein 6-Tupel  $A = \langle Q, \Sigma, \Lambda, \delta, \lambda, q_0 \rangle$ , wobei

- $Q$  ist eine endliche Zustandsmenge,
- $\Sigma$  ist das Eingabealphabet,
- $\Lambda$  ist das **Ausgabealphabet**,
- $\delta : Q \times \Sigma \rightarrow Q$  ist die Übergangsfunktion,
- $\lambda : Q \rightarrow \Lambda$  ist die **Ausgabefunktion**.
- $q_0$  ist der Startzustand.

Kommt der Automat in einen Zustand  $q$ , dann gibt er das Zeichen  $\lambda(q)$  aus. Oft werden diese Ausgabezeichen als Aktionen verstanden (oder sind Eingaben für andere Automaten).

Deterministische  
endliche  
Automaten

Transduktoren

Moore-Automat

Umsetzung

Python-Skript für  
Beispiel

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

- Im **Startzustand**  $q_0$  gibt der Automat  $\lambda(q_0)$  aus.
- Der Automat erhält ein Wort  $w = a_1 a_2 \dots$  über  $\Sigma$  als **Eingabe**.
- Der Automat liest im Zustand  $q_i$  das **Eingabesymbol**  $a_{i+1}$ , wechselt in den **Folgezustand**  $q_{i+1} = \delta(q_i, a_{i+1})$  und gibt dann  $\lambda(q_{i+1})$  aus.
- Das macht der Automat, so lange wie Eingabezeichen verfügbar sind. (Ggf. mit offenem Ende.)
- Dabei wird das Eingabewort  $w$  in das Ausgabewort  $\lambda(q_0)\lambda(q_1)\dots$  **übersetzt**. Daher auch der Name Transduktor.

Deterministische  
endliche  
Automaten

Transduktoren

Moore-Automat

Umsetzung

Python-Skript für  
Beispiel

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

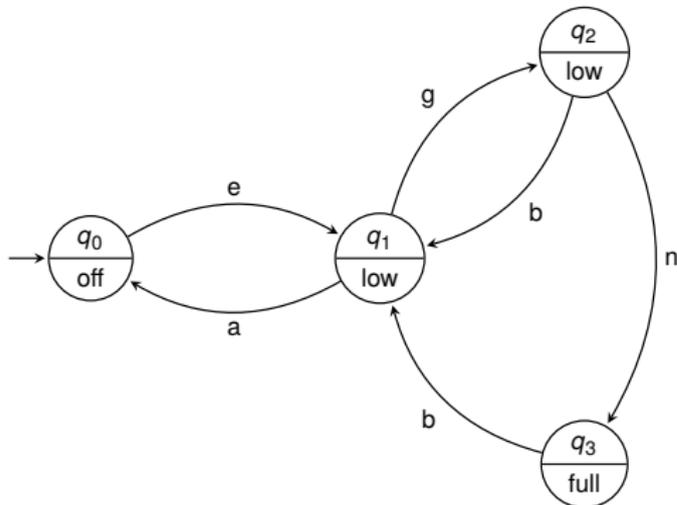
Welt &  
Modell

Zusammenfassung &  
Ausblick

# Beispiel: Ein hypothetische Motorsteuerung



$\Sigma = \{e, a, g, b, n\}$ , wobei  $e$  für „ein“,  $a$  für „aus“,  $g$  für „Gas geben“,  $b$  für „bremsen“,  $n$  für „nicht drehende Räder“ steht.  
 $\Lambda = \{\text{off}, \text{low}, \text{full}\}$ .



Deterministische  
endliche  
Automaten

Transduktoren

Moore-Automat

Umsetzung

Python-Skript für  
Beispiel

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick



- Wie sieht die **Implementierung** eines solchen abstrakten Automaten aus?
- Das Innere des Würfels:



- Batterien (4×AA-Akkus, also 4,5-6 Volt),
- Servomotor,
- pyboard (mit einem ARM-5 Prozessor, Beschleunigungsmesser, usw.), auf dem Micropython läuft

Deterministische  
endliche  
Automaten

Transduktoren

Moore-Automat

**Umsetzung**

Python-Skript für  
Beispiel

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

- `side_up()`:  
Bestimmt mit Hilfe des Beschleunigungsmessers, welche Seite oben liegt. Bei unklaren Werten wartet die Funktion, bis eine stabile Lage eingetreten ist.
- `new_input()`:  
Erzeugt ein neues Eingabesymbol (Zahl zwischen 1 und 6), wenn der Würfel 500 Millisekunden stabil lag.
- `next_state(state, input)`:  
Die Übergangsfunktion.
- `output_symbol(state)`:  
Berechnet das zum Zustand gehörige Ausgabesymbol.
- `automaton()`:  
Die Endlosschleife zur Ausführung des Automaten.
- `code_knock(code)`:  
Klopft entsprechend dem angeforderten Code.

Deterministische  
endliche  
Automaten

Transduktoren

Moore-Automat

Umsetzung

Python-Skript für  
Beispiel

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

# Der Seitenerkenner mittels Beschleunigungssensor



Die Erdbeschleunigung von 1g entspricht einem Messwert von rund 20.

## Seitenerkenner

```
thres = 12
def side_up():
    while True:
        x = acc.x(); y = acc.y(); z = acc.z()
        if x > thres: return 5 #x up
        if x < -thres: return 2 #x down
        if y > thres: return 6 #y up
        if y < -thres: return 1 #y down
        if z > thres: return 3 #z up
        if z < -thres: return 4 #z down
        # no stable situation yet
```

Deterministische  
endliche  
Automaten

Transduktoren

Moore-Automat

Umsetzung

Python-Skript für  
Beispiel

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

## Symbolerzeugung

```
def new_input():  
    while True:  
        curr = side_up()  
        new = curr  
        start = pyb.millis()  
        while (curr == new and  
               pyb.elapsed_millis(start) <= 500):  
            new = side_up()  
        if curr == new:  
            return curr
```

Erzeugt ca. alle 0,5 Sekunden ein neues Eingabesymbol.  
Nicht nur, wenn die Seite gewechselt wird. Daher muss der  
Automat etwas anders aussehen!

Deterministische  
endliche  
Automaten

Transduktoren

Moore-Automat

Umsetzung

Python-Skript für  
Beispiel

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

## Übergangsfunktion

```
def next_state(state, input):  
    if state == 0: # initial state  
        if input == 5: return 1  
        return 0  
    elif state == 1: # '5' read  
        if input == 5: return 1  
        if input == 1: return 2  
        if input == 4: return 4  
        return 0  
    elif state == 2: # '51' read  
        if input == 1: return 2 # repetition!  
        if input == 5: return 3  
        return 0  
    elif ...
```

Beachte: Jeder Zustand hat eine Schleife für das Zeichen, das dafür notwendig war, in den Zustand zu kommen.

Deterministische  
endliche  
Automaten

Transduktoren

Moore-Automat

Umsetzung

Python-Skript für  
Beispiel

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

## Der Automat & die Ausgabefunktion

```
def automaton():
    state = 0
    while True:
        if sw(): return # if switch is pressed, exit
        state = next_state(state, new_input())
        code_knock(output_symbol(state))

def output_symbol(state):
    if state == 10:
        return "north"
    elif state == 11:
        return "east"
    else:
        return None
```

Deterministische  
endliche  
Automaten

Transduktoren

Moore-Automat  
Umsetzung  
Python-Skript für  
Beispiel

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

# 3 Modellierung von Automaten



Deterministische  
endliche  
Automaten

Transduktoren

**Modellierung  
von  
Automaten**

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

- Im Vergleich zu Standard-Python hat Micropython einige Einschränkungen.
- Die Modellierung der Zustände durch Zahlen ist kompakt und effizient, aber fehleranfällig.
- Besserer Ansatz: Verwende Aufzählungstypen.

Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

```
from enum import Enum, auto
```

```
class State(Enum):  
    INIT = auto()  
    AFTER_5 = auto()  
    AFTER_51 = auto()  
    AFTER_515 = auto()  
    AFTER_5156 = auto()  
    AFTER_54 = auto()  
    AFTER_542 = auto()  
    AFTER_5425 = auto()
```

- Bessere Dokumentation für die Zustände.
- `enum.auto()` erzeugt automatisch einen neuen internen Code, sodass die Zustände durchnummeriert werden.

Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

S = State

```
def next_state(q: State, i: int) -> State:
    match (q, i):
        case (S.INIT,      5): return S.AFTER_5
        case (S.AFTER_5,   1): return S.AFTER_51
        case (S.AFTER_5,   4): return S.AFTER_54
        case (S.AFTER_5,   5): return S.AFTER_5
        case (S.AFTER_51,  5): return S.AFTER_515
        case (S.AFTER_515, 6): return S.AFTER_5156
        case (S.AFTER_515, 1): return S.AFTER_51
        case (S.AFTER_515, 4): return S.AFTER_54
        case (S.AFTER_54,  2): return S.AFTER_542
        case (S.AFTER_542, 5): return S.AFTER_5425
    return S.INIT
```

- Hier ist ein Fehler!

Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick



- Falls die Zustände selbst noch weitere Information beinhalten, können sie als “richtige” Objekte betrachtet werden.
- Die Zustandsfunktion wird nun als Methode implementiert.
- So ist auch die Menge der Zustände einfach erweiterbar.
- Wegen des Speicherbedarfs allerdings für eingebettete Systeme weniger geeignet.
- Das Zustandsmuster ist ein sogenanntes **design pattern**.
- Siehe *Design Patterns: Elements of Reusable Object-Oriented Software* von der “Gang of Four” (Gamma, Helm, Johnson, Vlissides).

Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick



```
# base class for all states
@dataclass
class State:
    def next(self, input: int) -> 'State':
        return self
    def output(self) -> str:
        return ""
```

- Die Basisklasse `State` dient nur als Superklasse, von der konkrete Zustandsklassen erben.
- Sie soll nicht instanziiert werden, d.h. `State()` soll nicht verwendet werden.
- Auch die Methoden sollen eigentlich nicht verwendet werden, sondern nur anzeigen, was in den Subklassen implementiert werden soll.
- Eine solche Klasse heißt **abstrakte Klasse** und die Methoden **abstrakte Methoden**.

Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick



```
@dataclass
class S_Init(State):
    def next(self, input: int) -> State:
        match input:
            case 5:
                return S_After([5])
        return self
```

- Subklasse von State.
- Methode `next()` wird überschrieben.
- Methode `output()` wird aus der Superklasse übernommen (keine Ausgabe).

Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

```
@dataclass
class S_After(State):
    prefix: list[int]
    def next(self, input: int) -> State:
        self.prefix = self.prefix + [input]
        match self.prefix:
            case ([5, 1] | [5, 1, 5] | [5, 1, 5, 6] |
                  [5, 4] | [5, 4, 2] | [5, 4, 2, 5]):
                return self
        match self.prefix[-1:]:
            case [5]:
                return S_After(self.prefix[-1:])
        match self.prefix[-2:]:
            case [5, 1] | [5, 4]:
                return S_After(self.prefix[-2:])
        return S_Init()
```

Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

```
def output(self) -> str:
    match self.prefix:
        case [5, 1, 5, 6]:
            return "north"
        case [5, 4, 2, 5]:
            return "east"
        case _:
            return ""
```

- Die Klasse `S_After` verwaltet das bisher gelesene Präfix der Eingabe im Feld `prefix`.
- Zustandsübergang durch Änderung des Präfixes (soweit möglich).

Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

## ■ Modellierung mit abstrakter, generischer Klasse

Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

**Abstrakte  
und  
generische  
Klassen**

Modellierung mit  
abstrakter,  
generischer Klasse

Welt &  
Modell

Zusammenfassung &  
Ausblick



Die Basisklasse `State` hat einige Nachteile, die dazu führen, dass der Code schlecht wiederverwendbar ist.

## Zu konkret

- Die Klasse sollte abstrakt sein, aber wir konnten das nicht ausdrücken.
- D.h., direkte Instanzen von `State` sind nicht erwünscht und jede Subklasse muss die Methoden `next` und `output` überschreiben.

## Zu spezifisch

- Die Klasse legt die Typen von Eingabe und Ausgabe bereits fest, obwohl diese Typen den Code nicht beeinflussen.
- Sie muss quasi kopiert werden um einen Automatenzustand in einem anderen Automaten zu verwenden.

Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Modellierung mit  
abstrakter,  
generischer Klasse

Welt &  
Modell

Zusammenfassung &  
Ausblick



## Abstrakte Klasse

- Eine **abstrakte Klasse** besitzt keine eigenen Instanzen, sondern dient nur als Muster für Subklassen.
- Sie kann **abstrakte Methoden** definieren. Jede Subklasse muss alle abstrakten Methoden implementieren.

## Generische Klasse

- Eine **generische Klasse** besitzt einen oder mehrere **Typparameter**, angezeigt durch **Typvariablen**.
- Die Typparameter können als Typen von Feldern sowie Parametern und Ergebnissen von Methoden verwendet werden.
- Bei Verwendung einer generischen Klasse müssen die Typparameter in eckigen Klammern angegeben werden.
- Beispiel: `list` ist generische Klasse mit einem Parameter; Verwendung als `list[int]`, `list[str]` usw.

Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Modellierung mit  
abstrakter,  
generischer Klasse

Welt &  
Modell

Zusammenfassung &  
Ausblick



```
from abc import ABC, abstractmethod
@dataclass
class State(ABC):
    def next(self, input: int) -> 'State':
        return self
    @abstractmethod
    def output(self) -> str:
        ...
```

## Um State zur abstrakten Klasse zu machen...

- Zur Subklasse von `abc.ABC` (`ABC` = abstract base class) machen.
- Abstrakte Methoden mit `@abstractmethod` dekorieren.
- Als Rumpf einer abstrakten Methode dient die Ellipse `...`, da unter Umständen kein sinnvoller Code möglich ist.

Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Modellierung mit  
abstrakter,  
generischer Klasse

Welt &  
Modell

Zusammenfassung &  
Ausblick

```
from abc import ABC, abstractmethod
from typing import Generic, TypeVar
INP = TypeVar ('INP')
OUT = TypeVar ('OUT')
@dataclass
class State(Generic[INP,OUT], ABC):
    def next(self, input: INP) -> 'State':
        return self
    @abstractmethod
    def output(self) -> OUT:
        ...
```

Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Modellierung mit  
abstrakter,  
generischer Klasse

Welt &  
Modell

Zusammenfassung &  
Ausblick

- Definiere Typvariablen zur Spezifikation von Typparametern:

```
INP = TypeVar ('INP')
```

```
OUT = TypeVar ('OUT')
```

Der deklarierte Name muss jeweils mit dem Argumentstring übereinstimmen.

- Erben von `typing.Generic` mit der gewünschten Zahl von Typparametern, hier zwei.

```
class State(Generic[INP,OUT], ABC):
```

- Verwenden der Typparameter in einer Methodensignatur.

```
def output(self) -> OUT:
```

Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Modellierung mit  
abstrakter,  
generischer Klasse

Welt &  
Modell

Zusammenfassung &  
Ausblick



```
@dataclass
class S_Init(State[int, str]):
    def next(self, input: int) -> State[int, str]:
        match input:
            case 5:
                return S_After([5])
        return self
    def output(self) -> str:
        return ""
```

- Erbt von `State[int, str]`. Dadurch wird `INP = int` und `OUT = str` eingesetzt.
- Methodensignaturen werden entsprechend eingesetzt.
- Der restliche Code der Implementierung ändert sich nicht.

Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Modellierung mit  
abstrakter,  
generischer Klasse

Welt &  
Modell

Zusammenfassung &  
Ausblick

Deterministische  
endliche  
Automaten

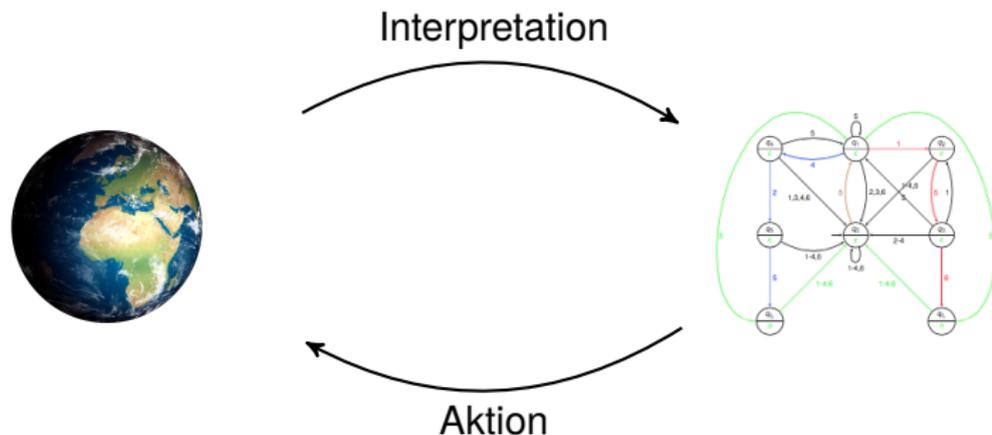
Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

**Welt &  
Modell**

Zusammenfassung &  
Ausblick



Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick

Vor dem Einsatz **formaler Modelle** (wie Moore-Automaten) müssen die Messwerte/Eingaben **interpretiert** und in **Symbole** umgesetzt werden. Die **Interpretation** und das **Modell** beeinflussen sich dabei gegenseitig (Beispiel: Würfelseitenerkennung und Automat).

⇒ Siehe Vorlesung Embedded Systems.

# 6 Zusammenfassung & Ausblick



**UNI  
FREIBURG**

Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

**Zusammen-  
fassung &  
Ausblick**

- Eine **formale Sprache** ist eine Menge von Wörtern.
- Automaten sind ein einfaches **Berechnungsmodell**.
- Sie können Sprachen erkennen (Akzeptoren) und übersetzen (Transduktoren).
- **Deterministische endliche Automaten (DEAs)** und **Transduktoren** (Moore-Automaten) werden zur Beschreibung von eingebetteten Systemen verwendet.
- Einfache Implementierung.
- Abstrakte und generische Klassen helfen bei der Strukturierung.

Deterministische  
endliche  
Automaten

Transduktoren

Modellierung  
von  
Automaten

Abstrakte  
und  
generische  
Klassen

Welt &  
Modell

Zusammenfassung &  
Ausblick