

Endliche Automaten als Modellierungswerkzeug

Implementierung endlicher Automaten

Anwendung endlicher Automaten

Sie haben den Aufbau und die Funktionsweise eines deterministischen endlichen Automaten kennengelernt und können inzwischen auch eigene Automatenmodelle in Form von Zustandsgraphen modellieren. Auch kennen Sie den Zusammenhang zwischen regulären Grammatiken und endlichen Automaten.

Diese Modelle haben nicht nur eine theoretische Relevanz. Sie sind häufig Grundlage für die Entwicklung von Programmen, bei denen eine bestimmte Art von Eingaben erforderlich ist (vgl. Abschnitt „Formale Sprachen im Alltag“). Im Folgenden wird es darum gehen, gegebene Automaten in Snap! zu implementieren. Die Implementierung in einer textbasierten Programmiersprache erfolgt analog.

Beispiel 1: Plausibilitätsprüfungen bei E-Mail-Adressen

Sie haben möglicherweise beim Ausfüllen von Online-Formularen schon einmal erlebt, dass dort Plausibilitätsprüfungen von Eingaben erfolgen. Dies fällt immer dann auf, wenn man gebeten wird, die gemachten Eingaben noch einmal zu überprüfen, da sie nicht zu den erlaubten Zeichenketten gehören. Der durch den Zustandsgraphen in Abbildung 1 definierte endliche Automat modelliert eine mögliche Sprache von E-Mail-Adressen. Dabei steht z für ein beliebiges Zeichen ungleich $@$.

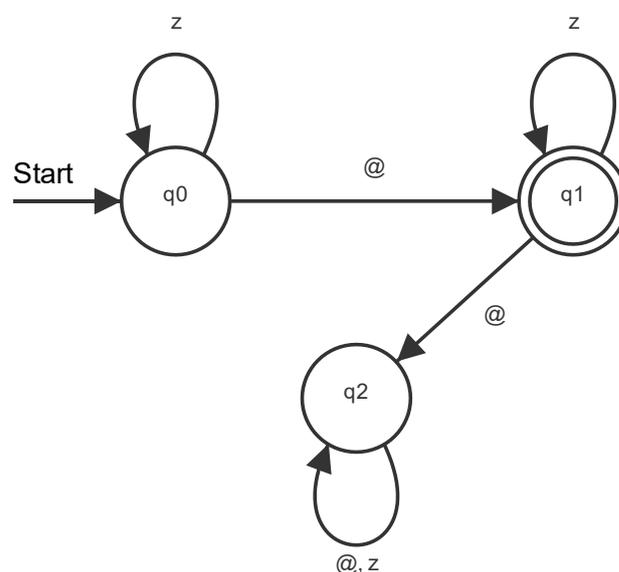


Abbildung 1: DEA mit Eingabealphabet $\Sigma=\{z,@\}$ zur Beschreibung von Email-Adressen

- Überprüfen Sie schrittweise, ob die Eingaben `test@blabla` und `hallo@welt@evtl` vom endlichen Automaten aus Abbildung 1 akzeptiert werden.
- Beschreiben Sie die durch den Automaten definierte Sprache mit eigenen Worten.

- c) Ein Mensch sieht einer Zeichenkette sofort an, ob sie ein @ enthält oder nicht. Erläutern Sie, warum ein Algorithmus dies nicht sofort „sehen“ kann, sondern auf Modellen wie beispielsweise dem endlichen Automaten aus Abbildung 1 basiert.
- d) Die gegebene Modellierung ist sicherlich nicht optimal: Geben Sie ein Beispiel einer Zeichenfolge an, die garantiert keine Email-Adresse darstellt, vom Automaten aber akzeptiert wird.

Implementierung der Plausibilitätsprüfung

Wahrscheinlich haben Sie bei der Bearbeitung obiger Aufgabe bemerkt, dass der Automat aus Abbildung 1 die Sprache aller Zeichenfolgen beschreibt, die genau ein @-Zeichen enthalten. Es gibt viele Möglichkeiten, ein Programm zu implementieren, welches überprüft, ob eine Zeichenfolge diese Eigenschaft besitzt. Mögliche Implementierungen wären zum Beispiel:

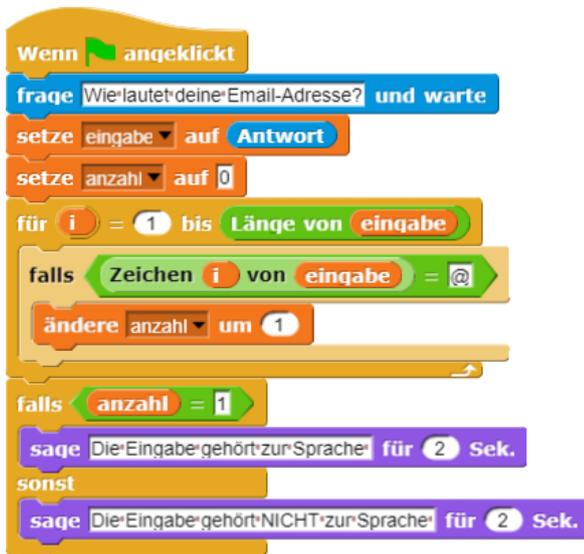


Abbildung 2: Die Sprache besteht aus allen Zeichenfolgen, die genau einmal das Zeichen @ enthalten.

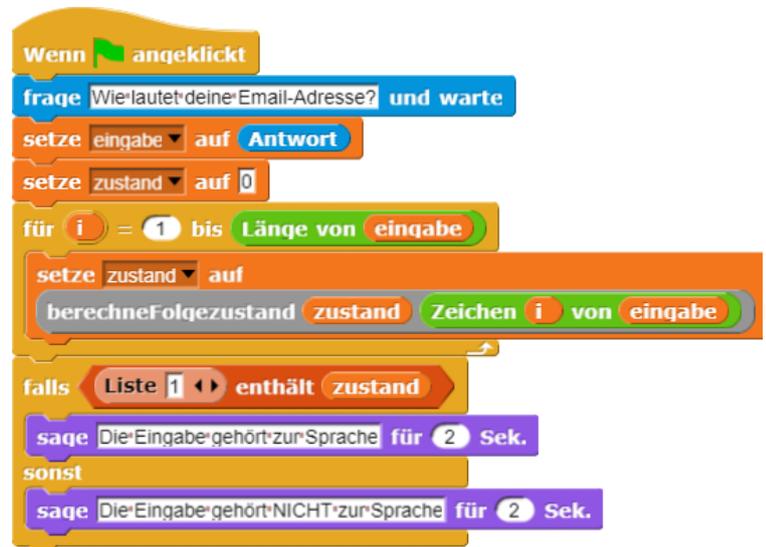


Abbildung 3: Prüfung auf die gleiche Sprache wie in Abbildung 2, diesmal unter Verwendung des endlichen Automaten aus Abbildung 1.

Aufgaben

- a) Beschreiben Sie Gemeinsamkeiten und Unterschiede der beiden Implementierungen.
- b) In Abbildung 3 wird eine Implementierung unter Verwendung des endlichen Automaten aus Abbildung 1 dargestellt. In diesem Automaten bezeichnet q_1 den einzigen Endzustand. Geben Sie an, wo dies im Quelltext in Abbildung 3 ablesbar ist, und wie das Programm geändert werden müsste, falls zusätzlich auch q_2 ein Endzustand wäre.
- c) Begründen Sie: Alle Programme, die überprüfen, ob eine Zeichenfolge genau ein @-Zeichen enthält, müssen folgende Eigenschaften besitzen:
 - Die Zeichenkette muss (evtl. vollständig) durchlaufen werden (d.h., man benötigt eine Schleifenstruktur bei der Implementierung).
 - Man benötigt eine Variable, deren Wert in jedem Schritt abhängig vom aktuell eingelesenen Zeichen angepasst wird.
 - Abhängig vom Endwert der Variablen gehört eine Eingabe zur Sprache oder nicht.

- d) Mithilfe des Blocks **berechneFolgezustand** wird der in Abbildung 1 gegebene Automat als Programm umgesetzt und dazu der Folgezustand abhängig vom aktuellen Zustand und vom aktuellen Zeichen berechnet. In Abbildung 4 wird ein erster Ansatz einer Implementierung dargestellt. Beschreiben Sie, wie das Programm erweitert werden muss, damit für alle Kombinationen von aktuellem Zustand und aktuellem Zeichen der Folgezustand berechnet wird. Testen Sie im Anschluss das Programm `Email-Automat.xml` in Snap! und vergleichen Sie die Implementierung mit Ihren Überlegungen.



Abbildung 4: Implementierung der Berechnung des Folgezustands

Beispiel 2: Gültige Raumnummern

In vielen öffentlichen Gebäuden werden Räume durch bestimmte Kombinationen von Buchstaben und Ziffern bezeichnet. Der durch den Zustandsgraphen in Abbildung 5 definierte endliche Automat modelliert gültige Raumnummern einer Beispielschule.

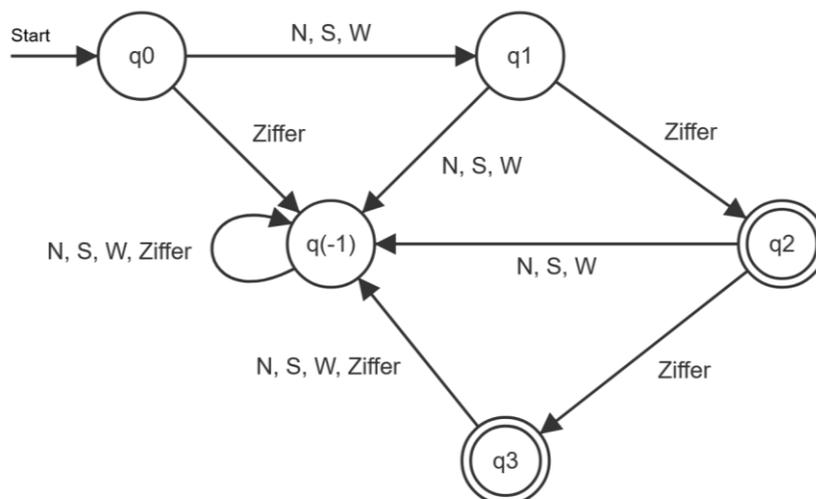


Abbildung 5: DEA mit Eingabealphabet $\Sigma = \{N, S, W, Ziffer\}$ zur Beschreibung von Raumnummern einer fiktiven Schule

- Überprüfen Sie schrittweise, ob die Eingaben S42 und N1 vom Automaten akzeptiert werden.
- Implementieren Sie ein Programm, welches mithilfe des Automaten für eine einzugebende Zeichenfolge entscheidet, ob sie zu den durch den Automaten definierten gültigen Raumnummern gehört oder nicht. Sie können als Vorlage die Datei Grundgeruest_Automat.xml verwenden.

Allgemeines Vorgehen

Je nach zu untersuchender Sprache kann es ganz unterschiedliche Implementierungen zur Erkennung einer Sprache geben. Weiß man z.B. wie in Beispiel 2, dass gültige Raumnummern aus zwei oder drei Zeichen bestehen, von denen das erste eines der drei Zeichen Z, S oder W ist gefolgt von einer oder zwei Ziffern, so kann eine Implementierung in Snap! wie folgt aussehen:

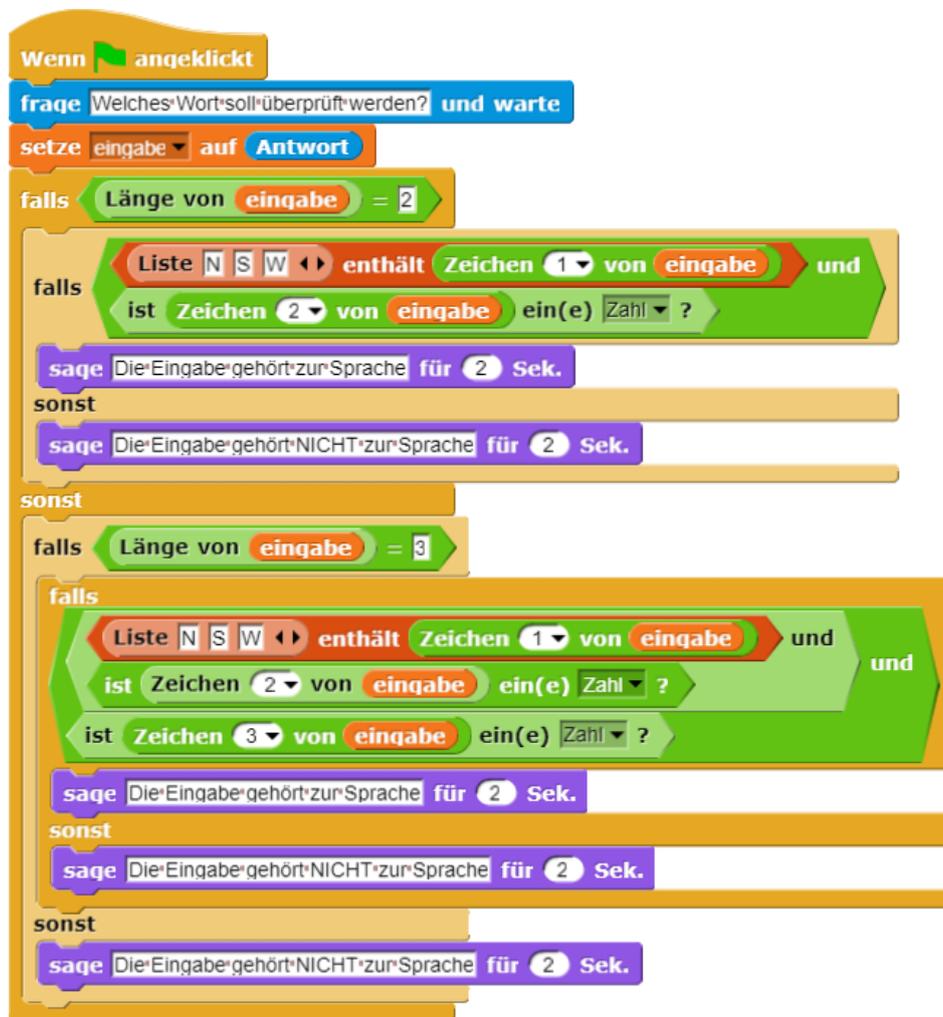


Abbildung 6: Programm zur Erkennung gültiger Raumnummern aus Beispiel 2

Liegt dagegen bereits eine Modellierung der Sprache durch einen DEA vor, so kann diese auch „rezeptartig“ in eine Implementierung überführt werden.

Möchte man mithilfe eines DEA ein Programm zur Erkennung einer regulären Sprache implementieren, ist das Vorgehen dabei immer ähnlich:

- (1) Die einzelnen Zustände werden durch eine ganzzahlige Variable `zustand` kodiert. Diese hat zu Beginn den Wert `zustand=0`.
- (2) Die Eingabe wird mithilfe einer Schleife vollständig durchlaufen.
- (3) Abhängig vom aktuellen Zustand und dem aktuellen Eingabezeichen wird dabei jeweils der Folgezustand berechnet. Hierfür verwendet man eine geschachtelte Verzweigung.
- (4) Ist das Eingabewort vollständig durchlaufen, wird es als zur Sprache zugehörig akzeptiert, falls die Zustandsnummer `zustand` für einen Endzustand steht.

Aufgrund dieser allgemeinen Eigenschaften kann beim „rezeptartigen Vorgehen“ stets die gleiche Vorlage, bei Snap! beispielsweise die Datei `Grundgeruest_Automat.xml`, verwendet werden. Unabhängig vom Automaten sind insbesondere die ersten beiden Aspekte (1) und (2) immer gleich. Die Methode (3) muss jeweils abhängig vom Automaten angepasst werden. In (4) muss jeweils die Liste der Endzustände aktualisiert werden.

Für Beispiel 2 erhält man so die folgende Implementierung:

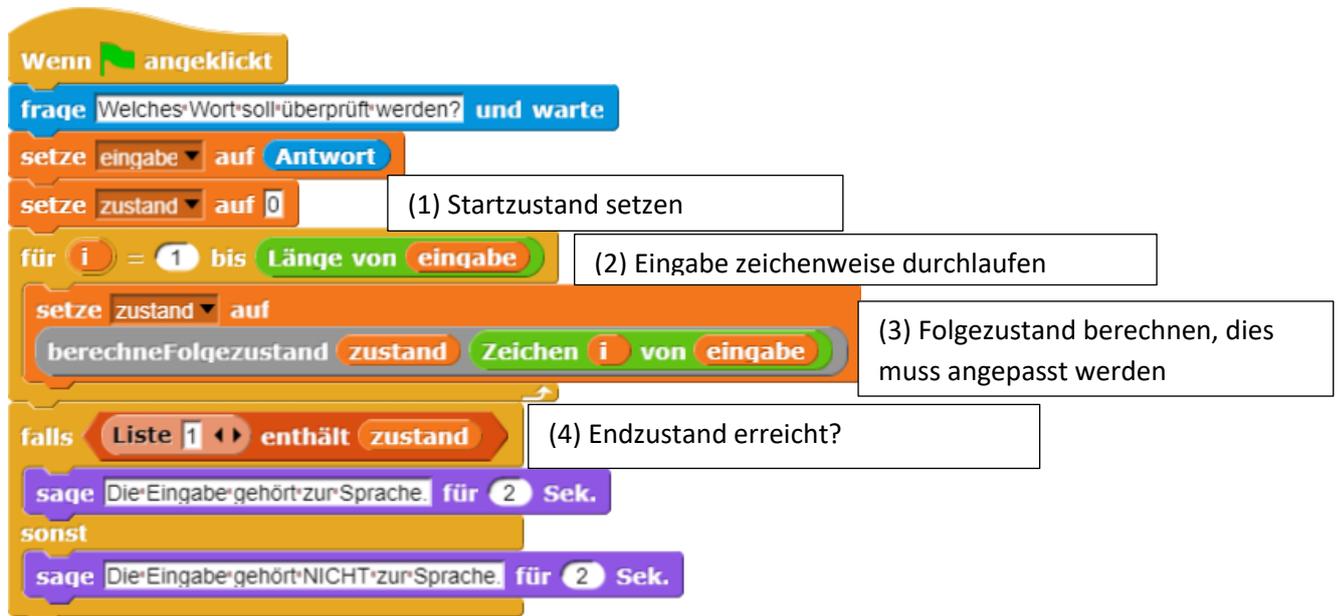


Abbildung 7: Vorlage `Grundgeruest_Automat.xml` zur Implementierung eines endlichen Automaten

mit der Implementierung von `berechneFolgezustand` definiert durch:

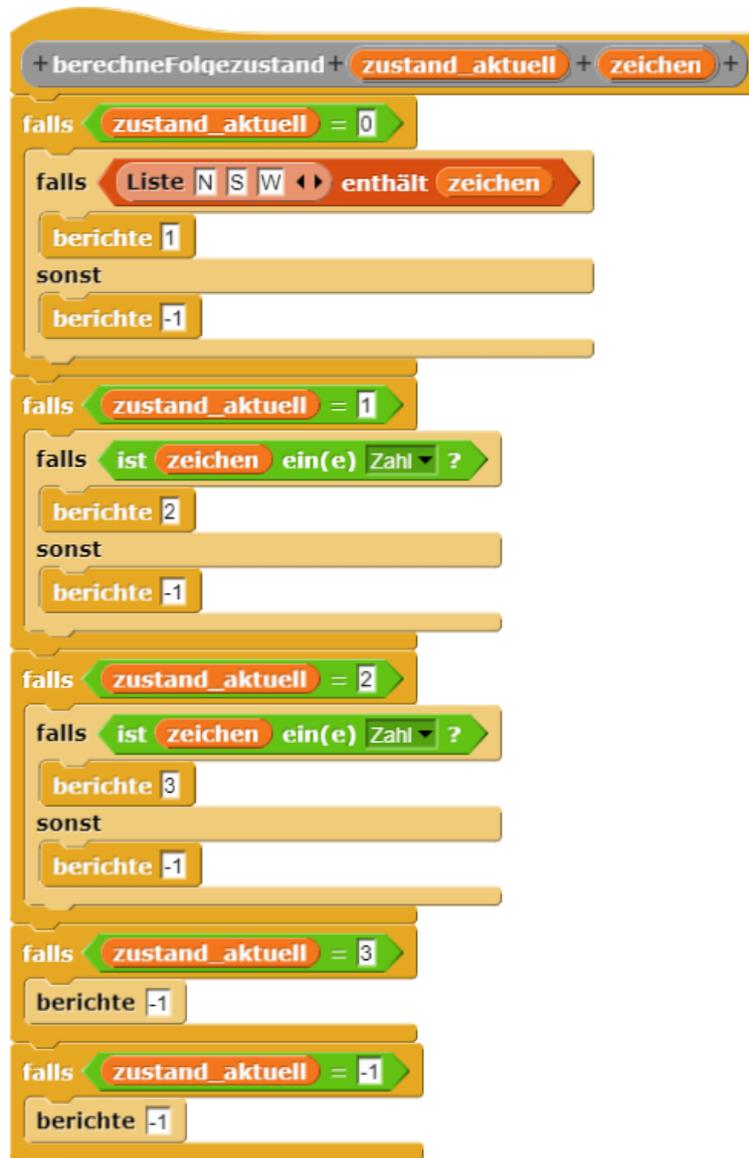


Abbildung 8: Implementierung zu Beispiel 2, dabei steht der Zustand -1 für einen Fehlerzustand

Die Rolle des Eingabealphabetes am Beispiel eines PLZ-Erkenners

Genauso wie Email-Adressen muss man in Online-Formularen manchmal auch die eigene Postleitzahl eingeben. Auch hier erfolgt häufig zunächst eine Plausibilitätsprüfung, die eine Eingabe dahingehend überprüft, ob es sich überhaupt um eine PLZ handeln kann. Deutsche Postleitzahlen bestehen aus fünf Ziffern. Der durch den Zustandsgraphen in Abbildung 9 definierte endliche Automat modelliert die Sprache der Postleitzahlen.

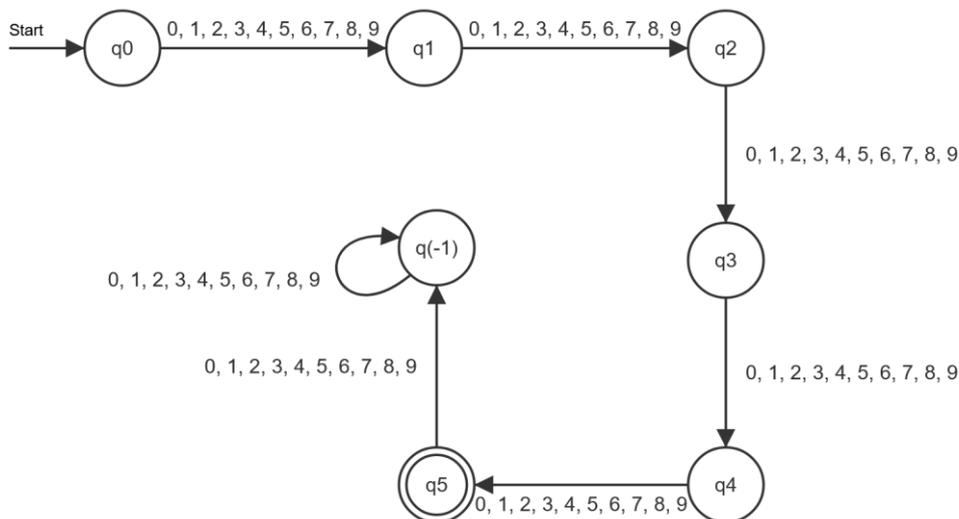


Abbildung 9: DEA mit Eingabealphabet $\Sigma=\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ zur Beschreibung deutscher Postleitzahlen

Bei der Implementierung spielt das Eingabealphabet eine besondere Rolle. Zur Modellierung einer Sprache wird als Eingabealphabet nur die Menge aller wirklich relevanten Zeichen betrachtet, in diesem Fall zum Beispiel nur Ziffern von 0 bis 9. Bei der Implementierung ist bei der Plausibilitätsprüfung dagegen wichtig, auch zu überprüfen, ob in einer Eingabe nicht möglicherweise auch Zeichen enthalten sind, die gar nicht zum Eingabealphabet gehören. In diesem Fall ist eine Eingabe sicher ungültig bzw. unplausibel. Für eine Implementierung kann es daher unter Umständen sinnvoll sein, das Eingabealphabet um einen Stellvertreter aller ungültigen Zeichen zu erweitern und im Übergangsgraphen des DEA entsprechende Übergänge in einen Fehlerzustand zu ergänzen. Ein mögliches Beispiel ist in Abbildung 10 dargestellt. Dort steht b für ein beliebiges Zeichen, welches keine Ziffer ist.

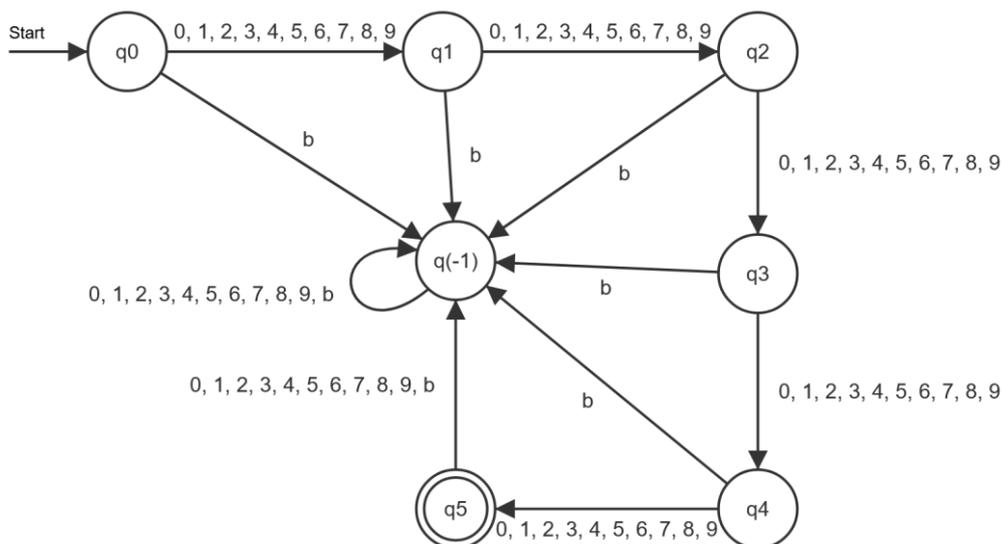


Abbildung 10: DEA mit Eingabealphabet $\Sigma=\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, b\}$ zur Beschreibung deutscher Postleitzahlen

Aufgaben

- a) Implementieren Sie zwei verschiedene Programme, die jeweils erkennen, ob eine Zeichenfolge zur Sprache der deutschen Postleitzahlen gehört oder nicht. Gehen Sie bei einer Version „rezeptartig“ wie auf Seite 5 beschrieben vor. Entwickeln Sie für eine weitere Version einen Algorithmus unter Verwendung konkreter Eigenschaften von Postleitzahlen (wie beispielsweise ihre Länge,...). Vergleichen Sie die unterschiedlichen Implementierungen.
- b) Diskutieren Sie Vor- und Nachteile beim „rezeptartigen“ Umsetzen eines endlichen Automaten als Programm.
- c) Im Folgenden werden weitere Beispiele für reguläre Sprachen im Alltag aufgelistet. Modellieren Sie diese jeweils durch einen DEA. Wählen Sie anschließend zwei reguläre Sprachen aus und implementieren Sie jeweils ein Programm zur Erkennung der zugehörigen regulären Sprache:
 - Twitter-Nutzernamen
 - Hashtags
 - Kfz-Kennzeichen
 - Kleidergrößen
 - Datumsangaben
 - ...

Dieses Werk ist lizenziert unter einer [Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz](https://creativecommons.org/licenses/by-nc-sa/4.0/). Sie erlaubt Bearbeitungen und Weiterverteilung des Werks unter Nennung meines Namens und unter gleichen Bedingungen, jedoch keinerlei kommerzielle Nutzung.