Modellierung als Automat

Der Automat bildet bei uns nicht das ganze Projekt ab, da dies viel zu komplex wäre und auch nicht im Sinne der Aufgabenstellung. Deshalb bilden wir nur einen Teil ab. Hier: Die Kontrolle ob eine eingegebener Code überhaupt richtig sein kann. Die Codes haben nämlich einen bestimmten Aufbau und sind nicht komplett zufallsgeneriert. Zuerst kommen zwar 4 willkürliche Ziffern, doch dann folgt die Quersumme aus diesen 4 Ziffern (sollte diese Quersumme zweistellig sein, dann bilden wir wieder die Quersumme aus ihr). Unser Automat prüft ob wirklich 5 Ziffern eingegeben wurden und gleicht dann die Quersumme mit der fünften Ziffer ab. Wenn alles stimmt erreichen wir einen Endzustand, sonst nicht.

Um den optimalen Typ für den Automaten zu finden gab es folgenden Überlegungen:

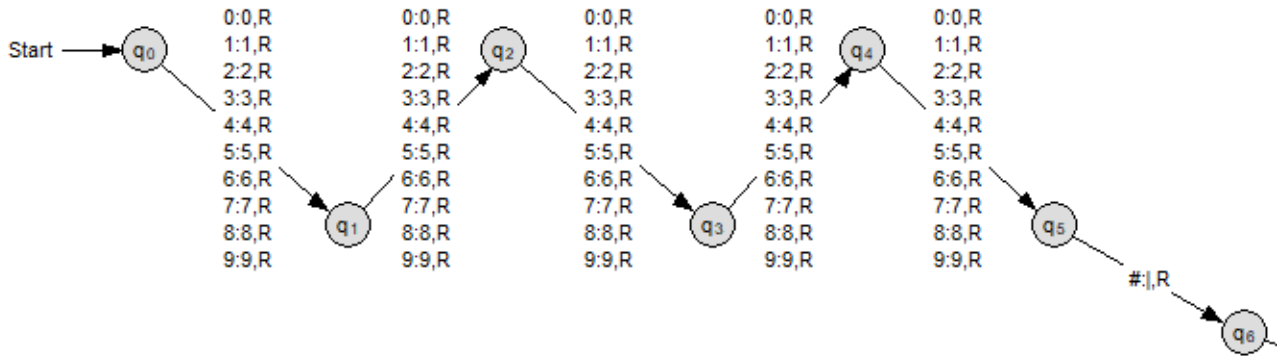
- Der Automat muss nicht mit mehreren Möglichkeiten arbeiten und soll mit den gleichen Eingaben immer auf ein und das Selbe Ergebnis kommen. Also fallen alle *nicht-deterministische* Typen weg.
- Außerdem muss der Automat den Code irgendwie speichern können um damit zu arbeiten. Damit dabei nicht extrem viele Zustände benötigt werden, bleiben nur noch der *Kellerautomat* und die *Turingmaschine*
- Zuletzt müssen wir Zahlen addieren, was effektiv nicht mit einem Kellerautomaten machbar ist. Also bleibt noch die *Turingmaschine*.

Dadurch, dass wir die *Turingmaschine* benutzen müssen, können wir aber die Zahlen nicht „live“ einlesen, da die Turingmaschine schon direkt die komplette Eingabe benötigt um dann damit zu arbeiten. Das heißt, wir werden die physische Eingabe des Codes später separat machen müssen und dann beispielsweise wenn die „Enter“-Taste gedrückt wird einfach die letzten 5 Ziffern nehmen und sie dem Automaten zur Verfügung stellen.

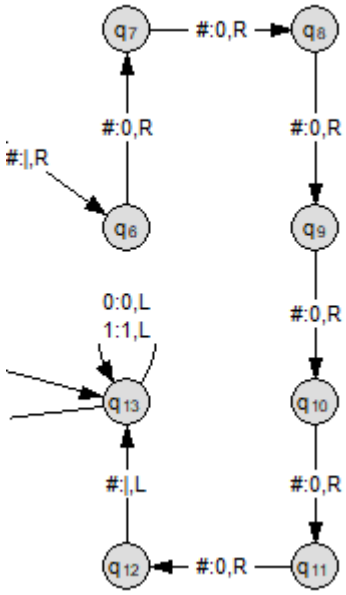
Jetzt sieht der grobe Ablauf des Automaten so aus:

1. Wir prüfen erst ob da wirklich 5 Ziffern stehen.
2. Die Quersumme der ersten 4 Stellen wird gebildet.
3. Die Quersumme wird mit der 5ten Stelle verglichen.

Im Detail sieht das Ganze so aus:
Um zu Prüfen, ob wirklich genau 5 Ziffern eingegeben wurden, gehen wir einfach 5 Übergänge lang nach rechts, unter der Bedingung, dass da eine der Ziffern 0-9 steht und erwarten dann am Ende ein „#“ (unser Band-Vorbelegungszeichen). Wird dieses „#“ da gefunden schreiben wir stattdessen ein „|“ als Trennzeichen für später.



Nun soll die Quersumme aus den ersten 4 Ziffern gebildet werden. Wir haben uns dazu entschieden, die Quersumme zuerst in binär zu bilden, da wir uns damit auch schon mehr im Informatik-Unterricht beschäftigt hatten und es uns trivialer erschien. Damit wir das Ergebnis in binär speichern können bereiten wir hier erst einmal einen 6-stelligen Bereich vor (6 Stellen, da die maximale Quersumme 9+9+9+9=36 ist und da $36_{10} = 100100_2$ Wir benötigen also maximal 6 Stellen)

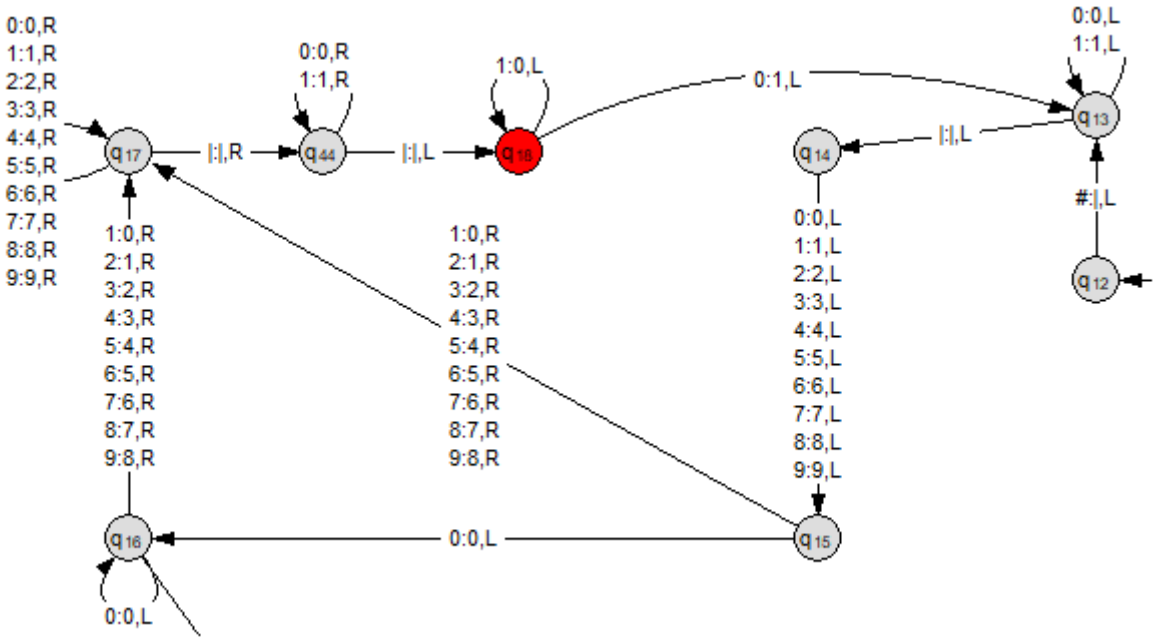


Um einen kleinen Überblick zu geben, hier ein Bild von einem beispielhaften Band der Turingmaschine bisher:

#	1	2	3	4	1	|	0	0	0	0	0	0	|	#	#

Jetzt beginnt die Bildung der Quersumme. Dafür wird die Leseposition auf dem Band bis zum Trennzeichen zwischen Code und Summe bewegt. Dann wird die fünfte Ziffer übersprungen, da sie für die Bildung der Quersumme egal ist.

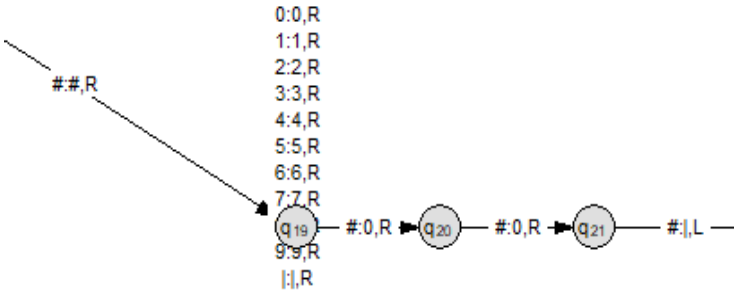
Anschließend wird von der folgenden Zahl 1 subtrahiert. Wenn dort eine 0 steht, gehen wir weiter zur nächsten Zahl usw. Nachdem die Ziffer um den Wert 1 dekrementiert wurde, wird die Leseposition nach ganz rechts zum äußeren Trennzeichen bewegt. Dann bewegen wir uns nach links durch die Binärzahl durch und erhöhen sie dabei um 1. Schlussendlich wird die Leseposition wieder ans mittlere Trennzeichen bewegt. Dieser Prozess wird so lange wiederholt, bis die Stellen 1 bis 4 im Code zu 0 geworden sind.
Hier der Ausschnitt aus dem tatsächlichen Automaten:



Das beispielhafte Band mit der Eingabe „12341“ sähe jetzt so aus:

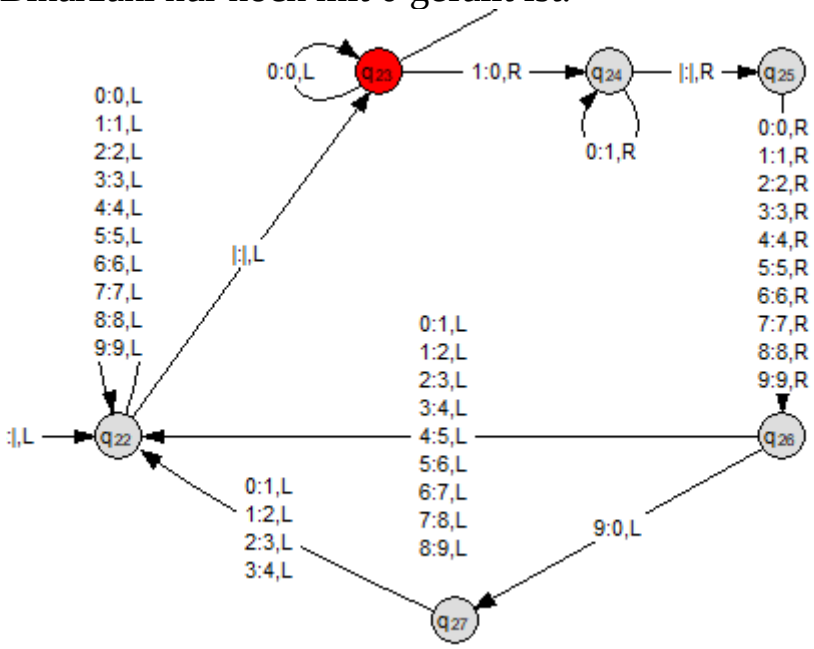
#	0	0	0	0	1	|	0	0	1	0	1	0	|	#	#

Um sicherzustellen, dass die erhaltenen Quersumme einstellig ist, müssen wir sie jetzt wieder zurück ins Dezimalsystem konvertieren. Dafür bewegen wir den „Cursor“ erst mal ganz nach rechts um dort Platz für die zu bauende Dezimalzahl zu schaffen.



#	0	0	0	0	1	|	0	0	1	0	1	0	|	0	0	|	#

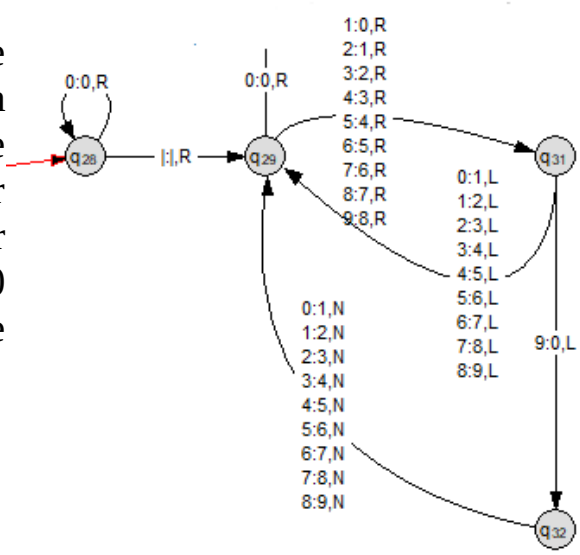
Für die tatsächliche Umrechnung wird die Leseposition bis zum Trennzeichen zwischen Binär- und Dezimalsumme bewegt. Dann subtrahieren wir immer 1 von der Binärzahl und addieren im Anschluss 1 zur Dezimalzahl, bis der Bereich der Binärzahl nur noch mit 0 gefüllt ist.



Auf dem Band sieht man jetzt im dritten Zahlenbereich die 10 als Quersumme von 1234:

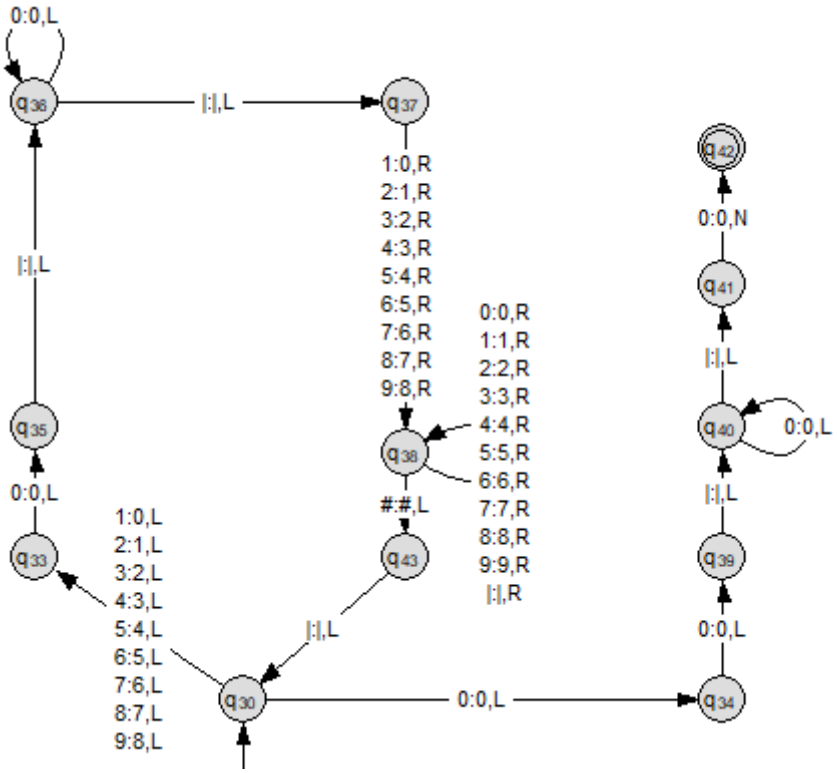
#	0	0	0	0	1	|	0	0	0	0	0	0	|	1	0	|	#

Der letzte Schritt ist hier jetzt nochmal die Quersumme aus der eventuell zweistelligen Quersumme zu bilden. Dafür wird, solange die Zehnerstelle nicht 0 ist, immer 1 von ihr subtrahiert und anschließend 1 zur Einerstelle addiert. Sobald die Zehnerstelle 0 ist, haben wir in der Einerstelle die fertige Prüfziffer.



#	0	0	0	0	1	|	0	0	0	0	0	0	|	0	1	|	#

Zu guter Letzt erfolgt der Vergleich der beiden Prüfziffern. Hierfür bewegt sich der „Cursor“ immer zwischen den beiden hin und her und subtrahiert abwechselnd von ihnen immer je 1. Sollte die eingegebene Prüfziffer zu früh 0 erreichen, bricht der Automat ab → der Code kann gar nicht richtig sein. Wenn die ausgerechnete Prüfziffer 0 erreicht, wird als allerletztes überprüft ob auch die eingegebene Prüfziffer im gleichen Schritt 0 geworden ist. Ist das der Fall, kann der Code stimmen und der Automat erreicht einen Endzustand.



In unserem Beispiel mit der Eingabe 12341 ist die 1 die richtige Prüfziffer, weshalb auf dem Band zur gleichen Zeit vorne und hinten Nullen entstehen:

#	0	0	0	0	0	|	0	0	0	0	0	0	|	0	0	|	#

Testphase: Der Automat funktioniert, erkennt also theoretisch richtige Code als richtig und theoretisch falsche Codes als falsch. Achtung: Ab einer bestimmten Komplexität braucht der Automat halt länger als 1000 Schritte, da das halt komplex ist. Da AutoEdit ~~ein Stück Shit ist~~ ab 1000 Schritten abbricht zu berechnen, funktioniert die Simulation hier nur bis zu einem bestimmten Punkt.

Die Implementierung auf dem Arduino(*später Zielseite mit Hyperlink einfügen*) besitzt aber die Kapazitäten alle Möglichen Codes zu testen.

Gesamter Automat (leicht anders ausgerichtet als auf den Bildern oben):