Konstruktion von Kellerautomaten

Die Fachdidaktik Informatik hat sich in den letzten Jahren etwas stiefmütterlich um die theoretische Informatik gekümmert. Andere Themen standen im Fokus wie z. B. Modellierung, E-Learning und Internet. Die Bedeutung der theoretischen Informatik für den Informatikunterricht in der Schule ist aber fachdidaktisch unumstritten. So gibt Baumann in [Bau] als relevante Gründe für die Behandlung von Theorie unter anderem an: Beständigkeit durch Theorie, allgemeinbildende Wirkung der Theorie, Theorie als Voraussetzung erfolgreicher Praxis, Theorie als Gegengift zur Hackermentalität, Theorie als Garant des Wissenschaftscharakters der Informatik. Bei den fundamentalen Ideen der Informatik nach Schwill ([Sch]) wird man bezüglich Aspekten der theoretischen Informatik auch reichlich fündig.

Die unterrichtliche Auseinandersetzung mit Inhalten der theoretischen Informatik kann aber nur dann überzeugend und erfolgreich sein, wenn eine schulgeeignete didaktische Gestaltung vorgenommen wird, die sich von universitären Zugängen deutlich abhebt. Der Grad der Formalisierung muss reduziert werden, die Beispiele müssen anwendungsnäher sein und falls möglich sollen praktische Anwendungen der Theorie behandelt werden. Es reicht also nicht aus, sich an den Zugängen und Beispielen zu orientieren, wie sie in der fachwissenschaftlichen Literatur bzw. in der ersten Phase der Lehrerausbildung vermittelt werden. Die didaktische Gestaltung der theoretischen Informatik für die Schule liegt bisher aber nur in Ansätzen vor. Im Folgenden wird der Themenbereich kontextfreie Sprachen und Kellerautomaten didaktisch aufbereitet.

Endliche Automaten und Kellerautomaten

Die zustandsorientierte Modellierung von Automaten hat mittlerweile schon Eingang in den Informatikunterricht der Sekundarstufe I gehalten (siehe [Hub]). Altersgemäß lässt sich das Wirkprinzip eines endlichen Automaten bei der Modellierung realer Automaten (z. B. Getränkeautomat, Digitaluhr) erfahren. In der Sekundarstufe II kann eine Vertiefung stattfinden, wobei dann sicherlich Nichtdeterminismus und Grenzen endlicher Automaten behandelt werden. Als praktische Anwendung kommen neben der Syntaxanalyse regulärer Sprachen (z. B. Turtlegrafik-Interpreter nach Modrow [Mod]) insbesondere reguläre Ausdrücke in Frage.

Endliche Automaten sind nicht in der Lage die Sprache $L = \{a^n b^n \mid n \in \mathbb{N}\}$ zu erkennen. Die Ursache besteht in der begrenzten Speicherfähigkeit aufgrund der endlichen Anzahl von Zuständen. Lassen wir unendlich viele Zustände zu, arbeiten wir also mit einem unendlichen Automaten, so kann diese Sprache erkannt werden. Solche unendliche Automaten sucht man in Theoriebüchern vergeblich, stattdessen wird das Konzept des Kellerautomaten eingeführt. Der Vorteil liegt auf der Hand. Beim endlichen Automaten sind Steuer- und Speichereinheit untrennbar verbunden, was bei der Erweiterung auf unendliche Automaten ganz unpraktisch wird. Beim Kellerautomaten werden hingegen Steuer- und Speichereinheit getrennt ausgeführt. Die Steuereinheit bleibt endlich, lediglich die Speichereinheit muss unbegrenzt sein.

Doch wie funktioniert der Speicher? Bei wahlfreiem Zugriff erhält man eine Turingmaschine, bei festem Zugriff den Kellerautomaten. Fester Zugriff bedeutet, dass man die zu lesende oder schreibende Speicherzelle nicht auswählen kann. Sie wird implizit durch die Folge der Lese- und Schreiboperationen bestimmt und funktioniert nach dem Kellerprinzip: last-in first-out. Was zuletzt in den Speicher geschrieben wurde wird als nächstes gelesen.

Kellerautomaten in der Fachwissenschaft

In der fachwissenschaftlichen Literatur werden Kellerautomaten meist so definiert, dass sie nicht nur ein einzelnes Zeichen sondern gleich ein ganzes Wort im Keller speichern können (z. B. [Ast] S. 300 und [Weg] S. 182). Diese Art der Definition zielt von vorneherein auf das wichtigste Ergebnis nämlich auf die Äquivalenz von Kellerautomaten und kontextfreien Grammatiken ab. So schreibt Wegener in [Weg]: „Unsere Untersuchung von Kellerautomaten ist nur dadurch motiviert, dass wir ein Maschinenmodell gesucht haben, das genau die kontextfreien Sprachen akzeptiert." Außer dem Beweis dieser Äquivalenz wird oft noch die Äquivalenz des Akzeptanzverhaltens beim Akzeptieren mit Endzuständen bzw. beim Akzeptieren mit leerem Keller für nichtdeterministische Kellerautomaten nachgewiesen. Die Auswahl an Beispielen für Kellerautomaten fällt in der Regel dürftig aus. Angegeben werden Kellerautomaten für die Sprachen $L_1 = \{a^n b^n \mid n \in \mathbb{N}\}$, $L_2 = \{xx^r \mid x \in \{0, 1\}^*,\ x^r \text{ ist das zu x inverse Wort}\}$ und $L_3 = \{xcx^r \mid x \in \{0, 1\}^*\}$ als Beispiel für einen deterministischen Kellerautomaten. Einige Fachbücher vertiefen die Betrachtung deterministischer Kellerautomaten im Hinblick auf LR(k)-Grammatiken und ihre Anwendung zum Parsen von Programmiersprachen.

Fachdidaktische Bewertung

Die Fokussierung des Themas Kellerautomaten auf die Äquivalenz zu kontextfreien Grammatiken ist für die Schule kein geeigneter Ansatz. Der eigentliche Äquivalenzbeweis liegt außerhalb der schulischen Reichweite, da er die Greibach-Normalform und Induktionsbeweise erfordert. An einem Beispiel kann lediglich plausibel gemacht werden, wie ein akzeptierender Kellerautomat zu einer kontextfreien Grammatik prinzipiell konstruiert werden kann. Die Definition eines Kellerautomaten muss am Vorwissen der Schülerinnen und Schüler anknüpfen, d. h. dass die vom abstrakten Datentyp Keller bekannten Operationen push, pop und nop als Speicheroperationen des Kellerautomaten zur Verfügung stehen müssen. Insbesondere sollte die push-Operation nur ein Zeichen auf dem Keller ablegen, um so auch das Wirkprinzip der zeichenverarbeitenden Maschinen zu betonen. Das Ablegen von Wörtern kann unter Hinzunahme von Zwischenzuständen durch zeichenweises pushen einzelner Zeichen mit ε-Übergängen realisiert werden.

Akzeptieren durch Endzustände ist den Schülerinnen und Schülern von endlichen Automaten her bekannt und kann problemlos auf nichtdeterministische und deterministische Kellerautomaten übertragen werden. Im Hinblick auf Anwendungen ist allerdings Akzeptieren durch leeren Keller vorzuziehen, denn typischerweise treten dabei geklammerte Ausdrücke auf. Öffnende Klammern werden auf dem Keller abgelegt, eine schließende Klammer führt zum Entfernen der zugehörigen öffnenden Klammer vom Keller. Nach Abarbeitung des geklammerten Ausdrucks muss der Keller leer sein. Die Akzeptanzvariante leerer Keller ist somit viel evidenter als Akzeptieren durch Endzustände.

Da bei deterministischen Kellerautomaten Akzeptieren durch leeren Keller schwächer als Akzeptieren durch Endzustände ist, müssen bei manchen Beispielen Endzustände benutzt werden. Die sogenannte Präfixeigenschaft (vgl. [Ast] S. 320) liefert ein einfaches Kriterium, um bei deterministisch kontextfreien Sprachen entscheiden zu können, ob sie von einem Kellerautomaten mit der Akzeptanzvariante leerer Keller erkannt werden können oder nicht. Eine Sprache L hat die Präfixeigenschaft, wenn für alle Wörter $w \in L$ gilt: ist x ein echtes Präfix von w so gehört x nicht zu L. Die beiden später im Text behandelten Beispiele sind Sprachen ohne die Präfixeigenschaft.

Gänzlich unbefriedigend ist die geringe Anzahl von Beispielen für kontextfreie Sprachen, die in der Fachliteratur angegeben werden. Die Sprache L_1 wird standardmäßig für die Einführung des Modells des Kellerautomaten benutzt, L_2 und L_3 braucht man für die Unterscheidung zwischen determinstsichen und nicht deterministischen Kellerautomaten. Einige weitere Beispiele findet man in

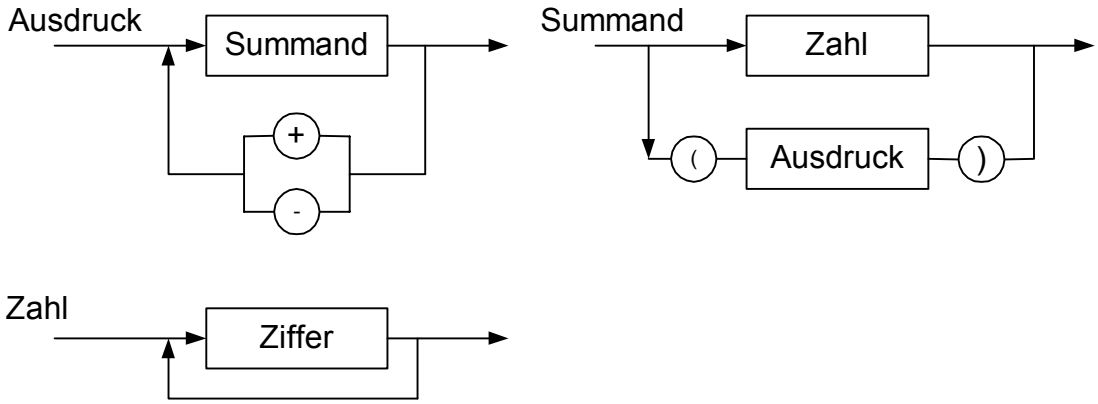
den Übungsaufgaben der jeweiligen Kapiteln der Fachliteratur. Aber auch diese sind meist abstrakt, d. h. es wird kein konkreter Anwendungsbezug angegeben, die Produktionen der zu analysierenden Grammatiken beschränken sich auf formale Aspekte, inhaltliche bleiben aussen vor. Im Sinne wissenschaftspropädetischen Arbeitens sollten solche abstrakten Beispiele auch in der Schule vorkommen. Die einseitige Beschänkung auf solche Beispiele wird aber den schulischen Anforderungen nicht gerecht. Die Schülerinnen und Schüler haben ausgiebig Erfahrungen im Umgang mit kontextfreien Sprachen (z. B. Programmiersprache, HTML, SQL) gesammelt und sollten nun auch einen Zusammenhang zwischen Theorie und Praxis herstellen können. Anderenfalls bleibt die Theorie leer und abgehoben, die Frage nach der Bedeutung der Theorie unbeantwortet.

Kellerautomaten in Schulbüchern

In nennenswertem Umfang werden Kellerautomaten in [Bau], [Mod], [Bur] und [Röh] behandelt. Typischerweise werden Kellerautomaten mit Hilfe eines Programms simuliert. Dieses wird selbst geschrieben oder man benutzt ein fertiges Simulationsprogramm (z. B. [Pol]). Damit lässt sich das Konzept von Kellerautomaten schülernah erarbeiten. Die Anwendung in der Praxis besteht meist in der Entwicklung eines Parsers oder Interpreters für arithmetische Terme oder logoähnliche Grafikbefehle. Die Modellierung von Kellerautomaten findet aber im Unterschied zu endlichen Automaten nur selten statt. Warum das so ist wird klar, wenn man versucht, einen Kellerautomaten für eine realtitätsnahe kontextfreie Sprache zu entwickeln.

Modellierung von Kellerautomaten

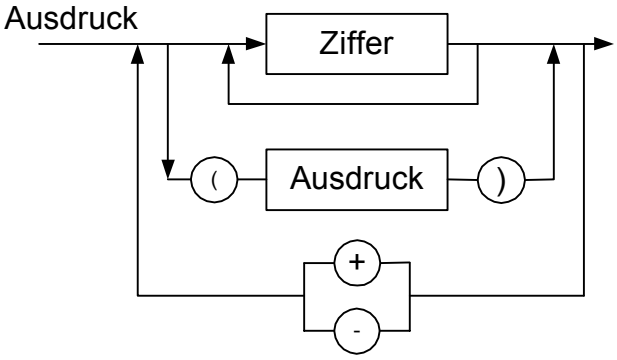
Betrachten wir dazu die Sprache der teilweise geklammerten Summanden, welche durch die drei folgenden Syntaxdiagramme definiert wird:



Beispielsweise ist 23+12-(18+15)-7 ein gültiger Ausdruck dieser Sprache. Die Sprache selbst ist vergleichsweise einfach, die Syntaxdiagramme sind für Schülerinnen und Schüler leicht lesbar. Man vermutet daher, dass die Modellierung eines Kellerautomaten für diese Sprache auch ohne größeren Aufwand möglich ist. Das Gegenteil ist der Fall! Es erweist sich als äußerst schwierig, einen Kellerautomaten für diese einfache Sprache zu konstruieren. Die Theorie lässt einem im Stich. In der Fachliteratur werden keine Kellerautomaten entwickelt, sondern nach unterschiedlichen Strategien Parser, die sich lediglich eines Kellerspeichers als Hilfsmittel bedienen.

Will man wirklich den Kellerautomaten als eigenständiges Konzept im Unterricht behandeln, so müssen Kellerautomaten auch konstruiert werden und dazu bedarf es offenbar einer angemessenen Konstruktionsmethode. Der Autor schlägt folgende Methode vor.

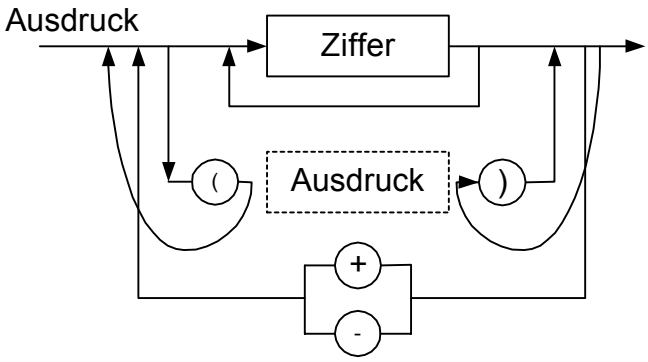
Zunächst fasst man die einzelnen Syntaxdiagramme durch Einsetzen von Teildiagrammen in das oberste Syntaxdiagramm zu einem einzigen Syntaxdiagramm zusammen. Im Beispiel entsteht dabei dieses Syntaxdiagramm:



Ein wesentlicher Gesichtspunkt für die Zusammenfassung in ein Syntaxdiagramm ist, dass es leichter sein muss aus einem Syntaxdiagramm das Zustandsdiagramm des erkennenden Kellerautomaten zu entwickeln als aus drei einzelnen Syntaxdiagrammen, denn das Zustandsdiagramm des Kellerautomaten ist vom Syntaxdiagramm abhängig.

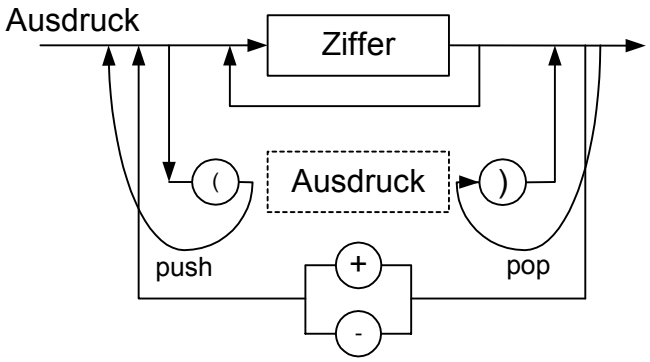
Die Variable Ziffer im Syntaxdiagramm könnte man durch das Syntaxdiagramm einer Fallunterscheidung für die zehn Ziffern 0 bis 9 ersetzen. Da dies die Problemstellung durch die Aufblähung eher erschwert als vereinfacht behalten wir die Variable Ziffer bei. Die Schülerinnen und Schüler haben genügend Erfahrungen im Umgang mit endlichen Automaten, um die Variable Ziffer adäquat in einem Diagramm modellieren zu können.

Echte Schwierigkeiten macht hingegen der geklammerte Ausdruck, denn hier kommt im Syntaxdiagramm Rekursion vor. Sicherlich muss der Kellerautomat die öffnenden Klammern auf dem Kellerspeicher ablegen und bei einer schließenden Klammer eine öffnende Klammer vom Kellerspeicher entfernen, aber wie soll er mit dem rekursiven Aufruf umgehen? Das sieht schwierig aus, ist aber einfach zu bewältigen. Nach der öffnenden Klammer gehen wir nicht zur inneren Variablen Ausdruck sondern zur äußeren und nach dem äußeren Ausdruck geht es einfach vor die schließenden Klammer zurück. Das sieht dann so aus:

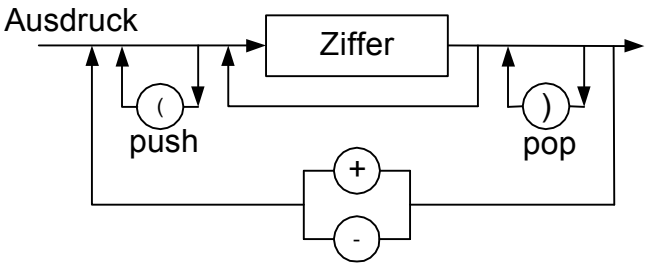


Durch diese Umkonstruktion wird die gestrichelt gezeichnete innere Variable Ausdruck entbehrlich und somit die Rekursion beseitigt. Das dadurch entstehende Syntaxdiagramm ist aber nicht mehr äquivalent zu dem der Ausgangssprache. Während das ursprüngliche Syntaxdiagramm noch die paarweise Verwendung von öffnender und schließender Klammer erzwingt, ist dies im transformierten Syntaxdiagramm nicht mehr der Fall. In diesem wäre z. B. auch der Ausdruck (((74) zulässig. Zur Beseitigung dieses Effekts setzen wir den Kellerspeicher ein, d. h. beim Lesen einer öffnenden Klammer wird diese auf dem Kellerspeicher abgelegt (push), beim Lesen eine schließenden Klammer muss eine öffnende vom Kellerspeicher heruntergenommen werden (pop). Wir bringen

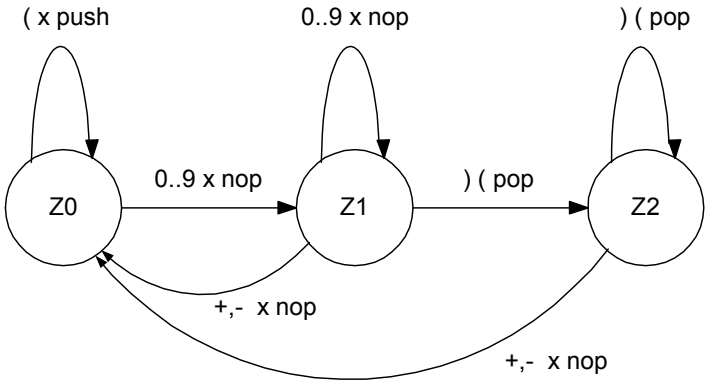
dies dadurch zum Ausdruck, dass wir das Syntaxdiagramm bei den Klammern um die durchzuführenden Operationen ergänzen und nennen das ganze dann ein attributiertes Syntaxdiagramm.



Wir haben durch Einsatz eines Kellerspeichers die Rekursion beseitigt, in eine Iteration transformiert und somit nebenbei ein Stück beziehungshaltige Informatik betrieben. Doch ganz fertig sind wir noch nicht, das Syntaxdiagramm muss noch etwas vereinfacht werden. Die push- und pop-Schleifen lassen sich wie folgt prägnanter zeichnen, wodurch das Syntaxdiagramm eine einfache und klare Struktur bekommt:



Man sieht jetzt drei sequentiell angeordnete Schleifen für push, Ziffer und pop. Die Sequenz selbst ist Bestandteil einer äußeren Schleife. Das so transformierte Syntaxdigramm lässt sich jetzt ohne besondere Probleme in das Zustandsdiagramm eines Kellerautomaten umsetzen. Im Startzustand Z_0 kann der Automat öffnende Klammern auf dem Kellerspeicher ablegen. Kommt eine Ziffer wechselt er in den Zustand Z_1, in dem er weitere Ziffern erkennt. Folgt in der Eingabe eine schließende Klammer wechselt der Automat in den Zustand Z_2, in dem er weitere schließende Klammern akzeptiert. Sowohl von Z_1 als auch von Z_2 erfolgt ein Zustandsübergang nach Z_0, wenn eines der beiden Rechenzeichen + oder – auftritt. Das Zustandsdiagramm des Kellerautomaten sieht also wie folgt aus:



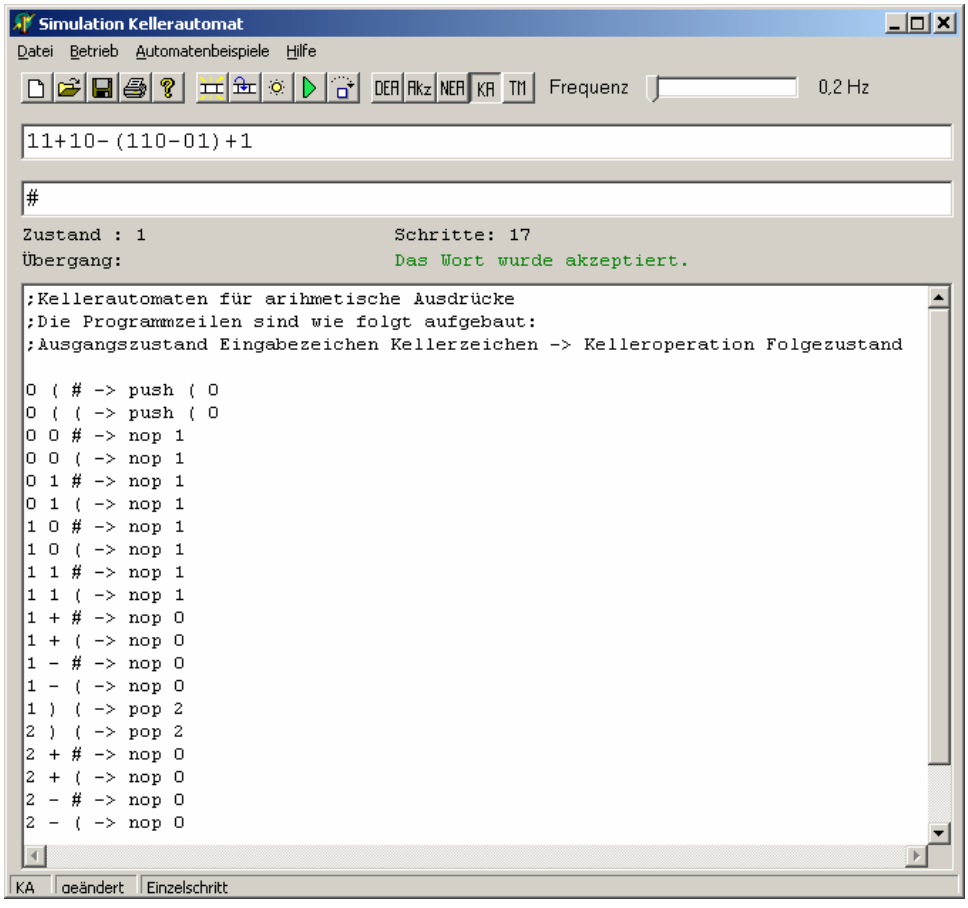
Die Zustandsübergänge werden mit drei Angaben beschriftet: Eingabezeichen, Kellerzeichen und Kelleroperation push, nop oder pop. x steht für ein x-beliebiges Kellerzeichen, 0..9 steht abkürzend für eine der zehn Ziffern. Der Zustandsübergang (x push bedeutet also, dass beim Lesen einer Klammer und einem x-beliebigem Zeichen auf dem Keller die öffnende Klammer gepusht wird. Der entstandene Kellerautomat ist deterministisch. Zusätzliche Gedanken muss man sich zu seinem Akzeptanzverhalten machen. Die Akzeptanzvariante leerer Keller reicht nicht aus, denn beispielsweise bei der Eingabe 23+ bleibt der Keller leer, obwohl der Ausdruck nicht zulässig ist. Die Zustände Z_1 und Z_2 sind die gültigen Endzustände, wobei zusätzlich ein leerer Keller gefordert wird. Formal lässt sich die zusätzliche Akzeptanzbedingung leerer Keller durch Einführung eines neuen Endzustandes beseitigen, aber diese Feinheit muss nicht unbedingt erarbeitet werden.

Letztlich kommt man bei der im Beispiel behandelten Sprache nicht mit der Akzeptanzvariante leerer Keller aus, weil die Sprache nicht die Präfixeigenschaft hat (vgl. oben). Im gültigen Ausdruck 23+12-(18+15)-7 ist das Präfix 23+12 schon ein gültiger Ausdruck ist.

Simulation des Kellerautomaten

Im Unterricht sollte der Kellerautomat simuliert werden. Da gibt es etwas praktisches zu tun, was die Motivation fördert und zudem kann mit der Simulation der entworfene Kellerautomat geprüft werden. Für die Simulation gibt es verschiedene Ansätze. Man kann die Schüler in der ihnen bekannten Programmiersprache ein Simulationsprogramm schreiben lassen, ein fertiges Simulationsprogramm einsetzen oder eine Simulationsumgebung (z. B. JFLAP) verwenden.

Ein Simulationsprogramm, mit dem man alle Automatenmodelle simulieren kann, steht auf dem Hessischen Bildungsserver zur Verfügung (vgl. [Pol]). In diesem Simulationsprogramm sieht der Kellerautomat wie im Bild aus:



Die Zustandsübergänge des Kellerautomaten werden in der Form:

Ausgangszustand Eingabezeichen Kellerzeichen -> Kelleroperation Folgezustand

aufgeschrieben. Die Zustände Z_0, Z_1 und Z_2 sind durch die Ziffern 0, 1 und 2 und der leere Keller durch das Zeichen # dargestellt. Zur Vereinfachung wurden im Bild nur die Ziffern 0 und 1 zugelassen.

In [Röh] wird gezeigt, wie man schulisch relevante Konzepte und Anwendungen der theoretische Informatik mit Hilfe der Programmiersprache Prolog auf dem Rechner bearbeiten kann. Dazu gehören Grammatiken, formale Sprachen, endliche Automaten, Kellerautomaten und Turingmaschinen.

Am Beispiels des obigen Kellerautomaten sehen wir uns die Modellierung mit Prolog an. Mittels Fakten wird das Zustandsdiagramm in der Wissensbasis erfasst. Für die Zustände nehmen wir

```
anfangszustand(z0).
endzustand(z1).
endzustand(z2).
```

und für die Übergänge das Prädikat
uebergang(Zustand, Eingabezeichen, Kellerelement, Kelleroperation, Folgezustand)

```
uebergang(z0, '(', _, push('('), z0).
uebergang(z0, Zeichen, _, nop, z1):-
  ziffer(Zeichen).
uebergang(z1, ')', '(', pop, z2).
uebergang(z1, Zeichen, _, nop, z1):-
  ziffer(Zeichen).
uebergang(z1, Zeichen, _, nop, z0):-
  member(Zeichen, [+, -]).
uebergang(z2, ')', '(', pop, z2).
uebergang(z2, Zeichen, _, nop, z0):-
  member(Zeichen, [+, -]).

ziffer(Zeichen):-
  member(Zeichen, ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']).
```

Die grundsätzliche Arbeitsweise des Kellerautomaten wird beschrieben durch:

```
% akzeptiere(+Wort)
akzeptiere(Wort):-
  anfangszustand(Zustand),
  atom_chars(Wort, Eingabeliste),
  kellerautomat(Zustand, Eingabeliste, [#]).

% kellern(+Kelleroperation, +AlterKeller, -NeuerKeller)
kellern(push(Element), Keller, [Element|Keller]).
kellern(pop, [_Element|Keller], Keller).
kellern(nop, Keller, Keller).

% kellerautomat(+Zustand, +Eingabeliste, +Keller)
kellerautomat(Zustand, [Eingabe|Rest], [Top|Keller]):-
  uebergang(Zustand, Eingabe, Top, KellerOp, NeuerZustand),
  kellern(KellerOp, [Top|Keller], NeuerKeller),
  write(Zustand), tab(2),
  write([Eingabe|Rest]), tab(2),
  write(Top), write(' -> '),
  write(NeuerKeller), tab(2),
  write(NeuerZustand), nl,
```

```
  kellerautomat(NeuerZustand, Rest, NeuerKeller).
kellerautomat(EndZustand, [], [#]):-
  endzustand(EndZustand).

test:-
  akzeptiere('23+12-(18+15)-7').
```

Mit der am Beispiel dargestellen Methode der Entrekursivierung von Struktogrammen durch Attributierung mit push/pop-Operationen lassen sich für alle schulisch relevanten Anwendungsbeispiele Kellerautomaten konstruieren. Ist die zu erkennende Sprache durch eine Grammatik gegeben, so kann diese durch Syntaxdiagramme dargestellt werden. Untergeordnete Syntaxdiagramme setzt man in übergeordnete Syntaxdiagramme, welche sich auf die untergeordneten Syntaxdiagramme beziehen. Tritt dabei eine Rekursion auf, wird diese durch push/pop-Attributierung beseitigt. Dieser Prozess wird fortgesetzt bis ein Gesamt-Syntaxdiagramm entstanden ist, dass nur noch Terminale enthält. Zum Schluss wird das erhaltene Gesamt-Syntaxdiagramm systematisch in ein Zustandsdiagramm transformiert. Dabei werden aus den Terminalen des Gesamt-Syntaxdiagramms Zustandsübergänge im Zustandsdiagramm und aus den die Terminale verbindenden Strecken die Zustände.

Weiteres Beispiel (könnte auch weggelassen werden)

Die Methode ist nicht nur für linear rekursive Syntaxdigramme, sonder auch für nichtlinear rekursive Syntaxdiagramme anwendbar. Wir betrachten dazu als weiteres Beispiel die Sprache mit folgender Grammatik:

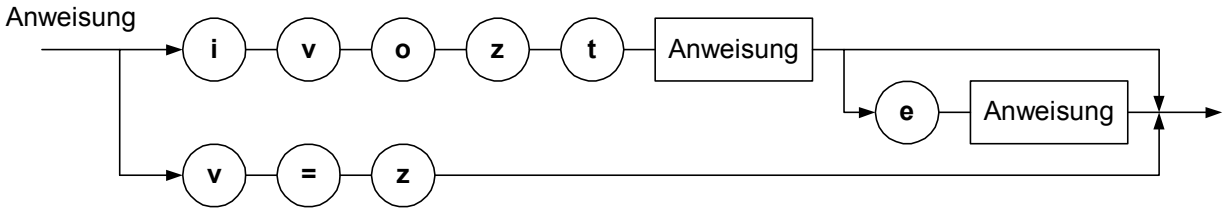
```
Anweisung → if-Anweisung | Wertzuweisung
if-Anweisung → if Bedingung then Anweisung [else Anweisung]
Wertzuweisung → Variable = Zahl
Bedingung → Variable Vergleichsoperator Zahl
Vergleichsoperator → < | <= | = | => | > | <>
Variable → Buchstabe | Buchstabe Variable
Zahl → Ziffer | Ziffer Zahl
Buchstabe → a | ... | z
Ziffer → 0 | ... | 9
```

Da keine Klammern vorkommen, könnte man meinen, diese Sprache sei regulär. Dem ist aber wegen der beliebig tief möglichen Schachtelung von if-Anweisungen nicht so. Werden nur vollständige if-then-else-Anweisungen geschachtelt, so muss die Anzahl der in der Eingabe vorkommende else-Terminale gleich der Anzahl der then-Terminale sein. Das entspricht von den Anforderungen her einem korrekt „geklammerten“ Ausdruck und macht deutlich, dass die Sprache nicht regulär ist. Kommen auch unvollständige if-then-Anweisungen vor, so muss die Bedingung abgeschwächt werden: die Anzahl der else-Teminale muss kleiner gleich der Anzahl der then-Terminale sein und zwar in allen Präfixen der Eingabe.

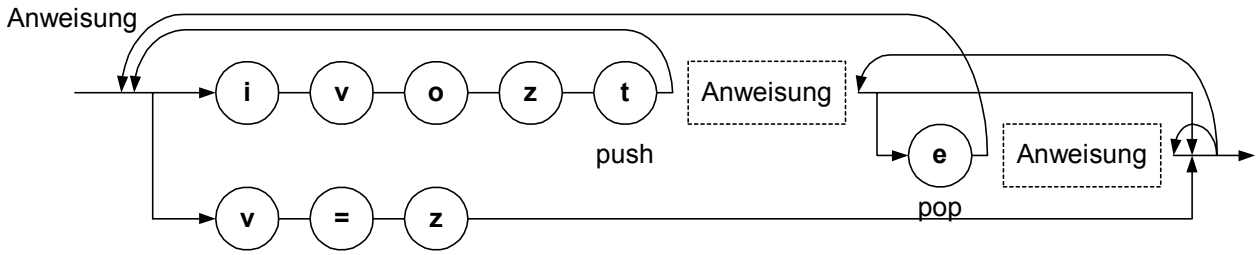
Für die Konstruktion des Kellerautomaten führen wir einige Vereinfachungen durch, die in einem Parser/Interpreter-Projekt von einem vorgeschalteten Scanner-Modul erledigt würden.

```
Anweisung → if-Anweisung | Wertzuweisung
if-Anweisung → i Bedingung t Anweisung [e Anweisung]
Wertzuweisung → Variable = Zahl
Bedingung → Variable Vergleichsoperator Zahl
Vergleichsoperator → o
Variable → v
Zahl → z
```

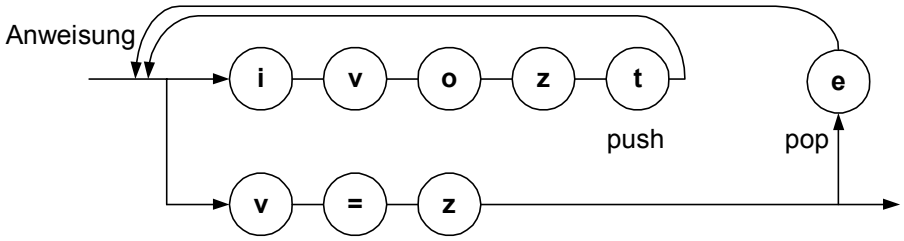
Das Gesamt-Syntaxdiagramm sieht wie folgt aus:



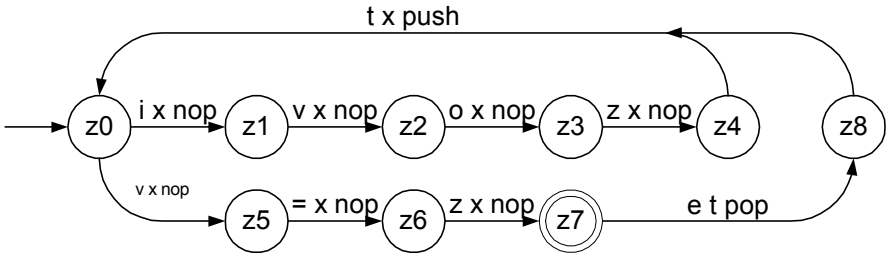
Da zweimal Rekursion auftritt handelt es sich um eine nichtlineare Rekursion. Wir entfernen beide Rekursionen durch Verzweigung an den Anfang und das Ende des Gesamt-Syntaxdiagramms und Attributierung mittels push und pop.



Das sich ergebende Syntaxdiagramm wird abschließend vereinfacht.



Daraus ergibt sich dieser Kellerautomat:



Die Sprache hat nicht die Präfixeigenschaft. Zum Beispiel gehört das Wort ivoztv=zev=z (if Bedingung then Wertzuweisung else Wertzuweisung) zur Sprache, aber auch das Präfix ivoztv=z (if Bedingung then Wertzuweisung) gehört auch schon dazu. Da die Anzahl der *else* kleiner als die Anzahl der *then* sein kann, funktioniert die Akzeptanzvariante leerer Keller nicht. Akzeptiert wird daher mit dem Endzustand Z_7.

Zusammenfassung

Die theoretische Informatik spielt in den Lehrplänen der Bundesländern eine wichtige Rolle. Die Fachwissenschaft gibt die relevanten Begriffe, Konzepte und Verfahren vor. Für die Auseinandersetzung mit der theoretischen Informatik in der Schule müssen allerdings die in der Ausbildung und der fachwissenschaftlichen Literatur angebotenen Zugänge kritisch auf schulische Verwendbarkeit geprüft werden. Eine direkte Übernahme der angebotenen Zugänge und Beispiele wird dem allgemeinbildenden Ansatz der Schulinformatik in der Regel nicht gerecht.

Am Beispiel des Kellerautomaten wird die Kluft zwischen fachwissenschaftlicher und schulischer Intention deutlich. Für die erforderliche Anwendung der Theorie in der Praxis, also die Konstruktion von Kellerautomaten für kontextfreie Sprachen, sind die angebotenen Zugänge wenig hilfreich. Die vom Autor vorgeschlagene Methode bietet einen schülergerechten Zugang, bei dem Vorkenntnisse sinnhaft vernetzt werden. Es können Kellerautomaten für anwendungsbezogene kontextfreie Sprachen konstruiert werden, wobei der Zusammenhang zwischen Rekursion und Kellerspeicher evident wird.

Literatur

[Ast] A. Asteroth, C. Baier: Theoretische Informatik. Pearson Studium, 2002.
[Bau] R. Baumann: Didaktik der Informatik, Ernst-Klett-Verlag, 2. Auflage, 1996.
[Bur] J. Burkert: Algorithmen und Datenstrukturen, Schroedel-Verlag, 1988.
[Hub] P. Hubwieser: Didaktik der Informatik, Springer-Verlag, 2000.
[Mod] E. Modrow: Automaten, Schaltwerke, Sprachen, Dümmler-Verlag, 1986.
[Pol] J. Poloczek: Simulationsprogramm für Automaten http://lernarchiv.bildung.hessen.de/archiv/sek_ii/informatik/13.1 zuletzt abgerufen am 29.10.06
[Röh] G. Röhner: Informatik mit Prolog, HeLP, 2. Auflage, 2002.
[Sch] S. Schubert, A. Schwill: Didaktik der Informatik, Spektrum Akademischer Verlag, 2004.
[Weg] I. Wegener: Theoretische Informatik. B. G. Teubner Stuttgart, 1993.