FLACI – Eine Lernumgebung für theoretische Informatik

Michael Hielscher & Christian Wagenknecht

Sprachen ermöglichen nicht nur die Kommunikation zwischen Menschen, sondern auch mit und zwischen Maschinen. Dazu benötigt es Regeln, wie sie sich mit formalen Grammatiken oder abstrakten Automaten aus dem Werkzeugkasten der theoretischen Informatik (kurz: TI) beschreiben lassen. Dieser Beitrag gibt Anregungen zur Umsetzung ausgewählter Inhalte der TI im Informatikunterricht.

1 Einführung

Praktische, technische und theoretische Informatik prägen die Informatik als Wissenschaftsdisziplin. Für die praktische und technische Informatik gibt es überzeugende Alltagsbezüge, die sich motivierend auf bestimmte Themen des Informatikunterrichts auswirken: Die Entwicklung einer kleinen Datenbank für die private Musiksammlung und die Programmierung eines Roboters sind Beispiele.

Für Inhalte der TI gibt es derartige Verknüpfungen mit Alltagsphänomenen nicht oder es handelt sich um Fragestellungen, deren Attraktivität bereits gute TI-Kenntnisse voraussetzt. Die mit TI einhergehenden abstrakten Begriffe und mathematischen Beschreibungsmethoden wirken nicht nur für Schülerinnen und Schüler meist abstoßend und werden deshalb im Schulunterricht wenn möglich weggelassen. Damit fehlt aber eine wichtige Perspektive auf die Informatik als Wissenschaftsdisziplin der automatisierten und strukturierten Informationsverarbeitung.

Die Kommunikation zwischen Mensch und Computer und zwischen Computern erfolgt immer über einen klar definierten Informationsaustausch, der nach entsprechenden Regeln stattfindet. Eine Programmiersprache ist ein gutes Beispiel dafür. Werden Informationen strukturiert gespeichert, bedarf es einer formalen Definition des Aufbaus – konkret – welche Daten werden in welcher Reihenfolge in einem Datenformat oder im Arbeitsspeicher abgelegt, um diese später auch wieder lesen zu können (Datenmodellierung). Das einfache Eingabe-Verarbeitung-Ausgabe-Modell (EVA) bildet die Grundlage für die Informationsverarbeitung im Computer. Doch wie kann der Computer eine Eingabe lesen und verarbeiten?

Die TI beschäftigt sich mit den absoluten und praktischen Grenzen algorithmischen Problemlösens (Berechenbarkeitstheorie, Komplexitätstheorie und Algorithmik). Insbesondere durch den Bedarf einer theoretischen Fundierung der Verarbeitungsprozesse von Programmen (Compilerbau, praktische Informatik) wurden die Theorie formaler Sprachen und die Automatentheorie motiviert. Für die Disziplin Informatik sind Sprachen und Automaten fundamental. Deshalb stellen sie auch einen der fünf zentralen Inhaltsbereiche der GI Bildungsstandards Informatik für die allgemeinbildende Schule und finden sich bereits in den Lehrplänen der Sekundarstufe II in mehreren Bundesländern.

2 Syntaxanalyse

Die Verarbeitung einer Eingabe – eines Wortes oder Quelltextes – beginnt mit dessen syntaktischer Analyse. Wie kann man feststellen, ob eine vorgelegte Zeichenkette Wort einer zuvor definierten Sprache ist? Dabei spielen semantische Aspekte zunächst keine Rolle: Die TI beschäftigt sich mit formalen Sprachen, der Compilerbau (praktische Informatik; Compiler = Sprachübersetzer) befasst sich mit Programmiersprachen. Die einfache Regel lautet: formale Sprache + Semantik = Programmiersprache. Jede Compilation beginnt mit der syntaktischen Analyse (Parsing) des Eingabewortes (Kasten 1).

Gegeben ist das Alphabet $A = \{a,b,c\}$. L sei die Menge aller Wörter über A, die mit b beginnen und deren Längen (Anzahl aller Zeichen des Wortes) genau 3 betragen. Es soll ein Parse-Verfahren angegeben werden, das für jede Zeichenkette x über A feststellt, ob x zu L gehört oder nicht.
Die Syntaxanalyse formaler Sprachen kann sehr einfach sein: Wir vergleichen jedes der neun möglichen Wörter aus $L = \{baa, bab, bac, bba, bbb, bbc, bca, bcb, bcc\}$ mit x. Nach der ersten Übereinstimmung wissen wir, dass x syntaktisch korrekt ist und zu L gehört. Hat man hingegen alle Wörter aus L erfolglos geprüft, ist klar, dass x nicht zu L gehört.

Kasten 1. Beispiel Parsing

Die Lernumgebung FLACI (https://flaci.com, Hielscher & Wagenknecht, 2019) stellt eine kleine Experimentierumgebung bereit, um sowohl mit den in Kasten 1 verwendeten Begriffen als auch mit einfachen Sprachen über einem selbst gewählten Alphabet zu explorieren und bei der Begriffsbildung zu unterstützen.

2.1 Reguläre Ausdrücke

Es ist offensichtlich, dass die Analysemethode durch Direktvergleich, wie in Kasten 1, nur für endliche Sprachen anwendbar ist. Es gibt eine Reihe von Anwendungsmöglichkeiten für endliche Sprachen, wie etwa das Durchmustern einer Konfigurationsdatei. Im Allgemeinen sind formale Sprachen aber unendliche Mengen. Verzichtet man in Kasten 1 auf die Längenbeschränkung für die Wörter, ergibt sich die unendliche Sprache

L = {*b, ba, bb, bc, baa, bab, bac, bba, bbb, bbc, bca, bcb, bcc, baaa,...* }, die wir nicht komplett aufschreiben können.
Hier hilft dann doch eine Alltagserfahrung weiter: Jedes Wort aus *L* muss in eine Schablone passen. Das erste Zeichen ist ein *b* und danach folgen beliebig viele weitere Zeichen aus dem Alphabet *A*. Schablonen zur Syntaxanalyse (einfacher Sprachen) heißen reguläre Ausdrücke (Abb. 1).

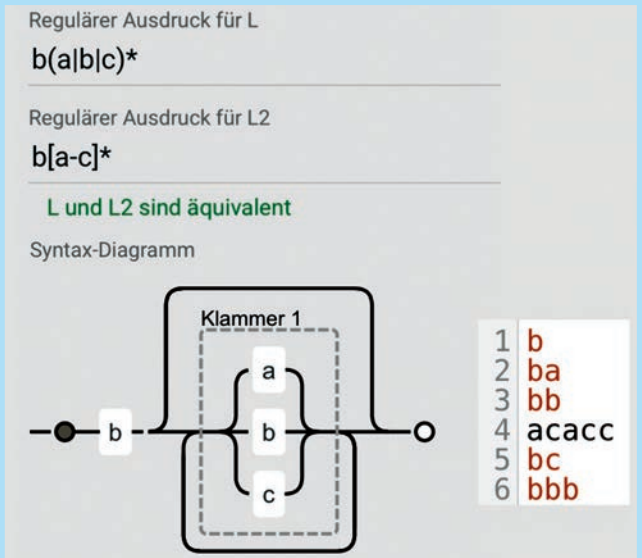

Abb. 1. Reguläre Ausdrücke in FLACI

Reguläre Ausdrücke spielen auch in der Praxis eine große Rolle. Auf der Kommandozeile genau wie in den meisten Programmierumgebungen und einigen Texteditoren lassen sich reguläre Ausdrücke zum Suchen und Ersetzen von Zeichenketten verwenden. Um zum Beispiel in einer Textdatei alle Zeilen zu finden, die mit einer dreistelligen Zahl beginnen, kann ein entsprechender `grep`-Befehl verwendet werden:

```
grep '^[0-9][0-9][0-9]' Eingabe.txt
```

FLACI hält interaktives Lernmaterial für reguläre Ausdrücke und einen Experimentalbereich bereit. Syntaxdiagramme werden automatisch generiert und verschiedene Ausdrücke auf Äquivalenz überprüft. Einige Beispiele zur Eingabevalidierung von Telefon- und ISBN-Nummern, Postleitzahlen oder Email-Adressen stehen für die praktische Bedeutung regulärer Ausdrücke.

2.2 Formale Grammatiken

Reguläre Ausdrücke sind deskriptive Beschreibungsmittel für formale Sprachen: Solche Ausdrücke beschreiben die Gestalt aller zulässigen Wörter ohne ein spezielles Prüfverfahren (Anwendung der Schablone) dafür anzugeben.
Eine weitere (sogar noch leistungsfähigere) Beschreibungsform syntaktisch korrekter Wörter sind formale Grammatiken. Kernelemente sind Regeln, deren schrittweise Anwendungen die Herstellung/Ableitung je eines syntaktisch korrekten Wortes garantieren.
Die Deskriptivität dieser Methode resultiert aus der Verwendung rekursiver Regeln, die die Unendlichkeit der Sprache sicherstellen (Kasten 2).

Ein arithmetischer Ausdruck sei eine Zeichenkette, aus den Terminalen (T, Alphabetzeichen) Plus, Minus, Klammer auf, Klammer zu und den Ziffern 1 bis 9. Jede öffnende Klammer muss auch eine schließende Klammer besitzen, um einen syntaktisch korrekten Ausdruck zu bilden. Die Eingabe **(1+(3−4)−(1))** gehört zur Sprache der arithmetischen Ausdrücke, **(8+(7+(2** und **(1−))2+)3)** hingegen nicht. Für den Aufbau dieser Ausdrücke lassen sich Regeln (auch Produktionen P genannt) angeben, bei denen Nichtterminale (N) als Platzhalter fungieren.

Eine vollständige formale Grammatik *G* = (*N, T, P, s*) kann für die arithmetischen Ausdrücke wie folgt angegeben werden:

N = { Ausdruck }
T = { **+, −, (,), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9** }
P = {
Ausdruck → Ausdruck + Ausdruck
Ausdruck → Ausdruck − Ausdruck
Ausdruck → **(** Ausdruck **)**
Ausdruck → **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**
}
s = Ausdruck

Die Analyse des Wortes **8−(2+3)** beginnt beim Startsymbol *s* und ergibt den Ableitungsbaum in Abbildung 2.
Die Eingabe **(8+(7+(2** scheitert hingegen.

Kasten 2. Eine Grammatik für arithmetische Ausdrücke

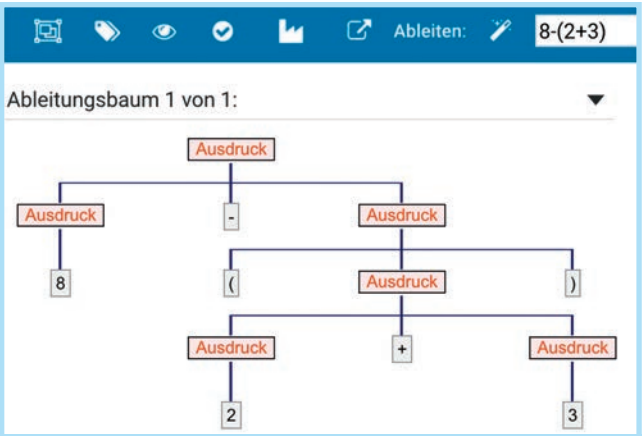

Abb. 2. Interaktive Ableitung eines Eingabewortes in FLACI

Die rekursive Anwendung der Regeln führt zur Ableitung eines Eingabeworts. Nur wenn diese gelingt, kann die syntaktische Korrektheit der Eingabe nachgewiesen werden. Der Ableitungsprozess lässt sich in FLACI schrittweise visualisieren (Abb. 2).

2.3 Abstrakte Automaten

Näher am maschinellen Verarbeitungsmodell bei der Eingabewortanalyse sind die Zustandsmodelle abstrakter Automaten. Die Arbeitsweise der Automaten wird durch eine Überführungsfunktion, gern als Zustandsgraph dargestellt, beschrieben. Sie ist häufig einfacher nachzuvollziehen als die rekursiven Regeln formaler Grammatiken. Mit der Überführungsfunktion wird festgelegt, welche Zustandsübergänge unter bestimmten

Bedingungen stattfinden können. Die Übergangsfolge verbraucht das Eingabewort und entscheidet am Ende dessen Akzeptanz, d.h. dessen syntaktische Korrektheit (Kasten 3).

Das Eingabewort wird taktweise verarbeitet. Der Automat startet auf dem ersten Zeichen des Wortes im Startzustand q_0. Mit dem Lesen des ersten b wechselt er in den Endzustand q_2, den er für alle weiteren Zeichen des Wortes nicht mehr verlässt. Für a und c erfolgt ein Wechsel in q_1 mit dem Unterschied, dass q_1 wie eine Falle wirkt: keinem Alphabetzeichen gelingt es, diesen Zustand wieder zu verlassen. Da q_1 kein Endzustand ist, wird das betrachtete Wort abgewiesen (Abb. 3).

Kasten 3. Beschreibung der taktweisen Verarbeitung der Konfigurationenfolge *b a b c*

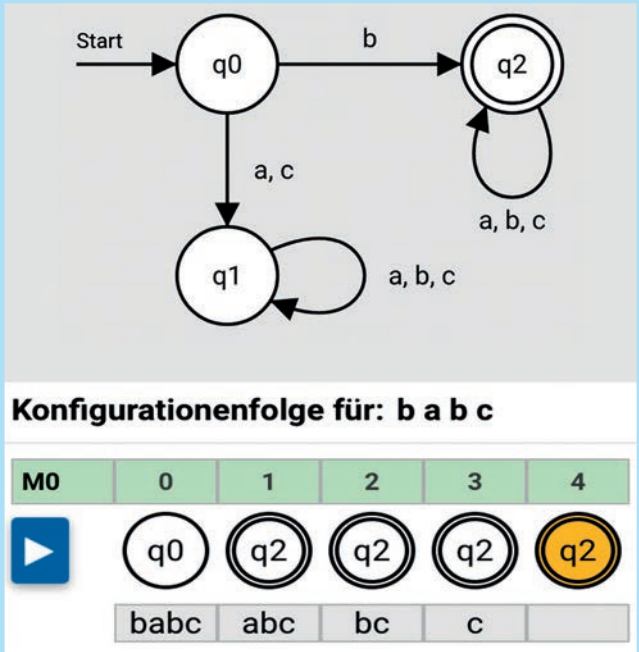

Abb. 3. Zustandsautomat in FLACI

Der hier vorgestellte Automatentyp (DEA = deterministischer endlicher Automat) arbeitet in der beschriebenen Art und Weise. Es gibt weitere Automatenmodelle, die sich in der Wirkungsweise und im Akzeptanzverhalten wesentlich unterscheiden, z.B. Kellerautomaten (DKA und NKA). Alle genannten Automatentypen, aber auch Moore- und Mealy-Maschinen (Automaten mit Ausgabe) sowie Turing-Maschinen, werden durch FLACI unterstützt. Ihre Arbeitsweise kann schrittweise simuliert werden.

2.4 Äquivalente Beschreibungsmittel für Sprachen

Sowohl reguläre Ausdrücke, formale Grammatiken als auch abstrakte Automaten beschreiben Sprachen so, dass Computer diese Definitionen verarbeiten und zur Eingabeprüfung verwenden können. Unterschiedliche Beschreibungsmittel können durch Transformationen ineinander überführt werden, wobei die drei genannten Formen unterschiedlich mächtig sind und jeweils nicht alle Sprachklassen beschreiben können. Am Beispiel der arithmetischen Ausdrücke lässt sich zeigen, dass kein äquivalenter regulärer Ausdruck existiert, sondern ein Kellerautomat nötig ist, um sich die geöffneten und zugehörigen geschlossenen Klammern bei der Abarbeitung „merken“ zu können. Die Lernumgebung FLACI unterstützt durch vielfältige, automatische Transformationen auf Knopfdruck (Abb. 4).

3 Unterrichtsbeispiel Sprachübersetzer

Die in den vorherigen Abschnitten erarbeiteten Grundlagen lassen sich im Unterricht attraktiv mit Basisinhalten des Compilerbaus verknüpfen und verzahnt mit praktischer Anwendung vermitteln und anwenden (vgl. Hielscher & Wagenknecht, 2009). Die Lernumgebung FLACI (formal languages, compiler, and interpreter) unterstützt sowohl bei der Vermittlung der Theorie als auch bei deren praktischer Anwendung. FLACI folgt damit einer langen Tradition in der Informatikdidaktik, durch digitale Werkzeuge das Lernen und Lehren aktiv zu unterstützen.
In einem phänomenorientierten Informatikunterricht kann die Programmiersprache ins Zentrum gestellt und der Frage nachgegangen werden, wie Computer Eingaben verarbeiten können. Dazu wird der Wunsch nach einem eigenen Compiler für eine frei erfundene formale Sprache (Mini-Programmiersprache) als Ziel einer Unterrichtseinheit formuliert. Im Folgenden werden die einzelnen Problemfelder in einer repräsentativen Modellierungsaufgabe identifiziert, deren Betrachtung motiviert und in einer lehrplangerechten Vertiefungsstufe ausgeführt. Dabei wird der Einsatz von FLACI an einem konkreten Beispiel illustriert.

3.1 Compilation und Interpretation von Sprachen

Programme, also Wörter einer Sprache L_1, werden vor ihrer Interpretation gewöhnlich in eine Sprache L_2 übertragen. Dies geschieht mit dem Ziel, aus einer menschenlesbaren Form eine verarbeitungsnähere (maschinennähere) Notation herzustellen. Beispielsweise wird ein Java-Programm in ein entsprechendes Bytecode-Programm übersetzt. Für eine solche (mit-

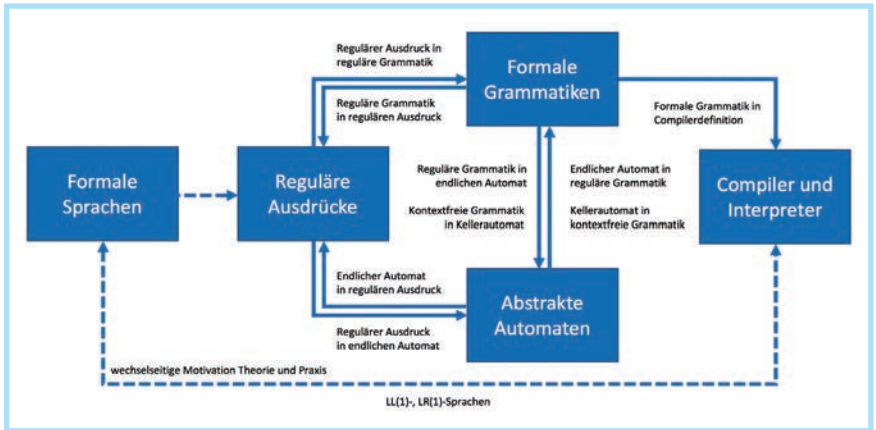

Abb. 4. Transformationen und Beziehungen innerhalb der Lernumgebung FLACI

unter mehrstufige) Übersetzung benötigt man einen $L_1 \rightarrow L_2$ Compiler. Im Unterschied zur Interpretation entsteht bei der Compilation stets ein neues Wort/Programm, das nach bestimmten Regeln aus dem verarbeiteten Wort gebildet wird. Der Interpreter führt hingegen in Abhängigkeit von der Eingabe verschiedene Aktionen unmittelbar aus.

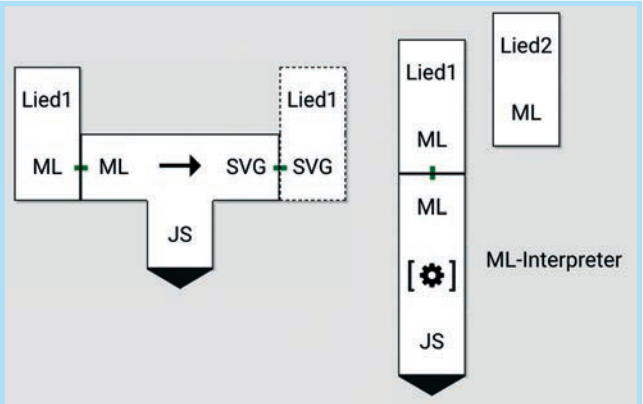

Abb. 5. Musiksprache ML → SVG Compiler (links) und ML-Interpreter

In Abbildung 5 werden T-Diagramme zur Veranschaulichung und Modellierung von Übersetzungs- und Interpretationsabläufen verwendet. Der Name „T-Diagramm" ergibt sich aus der Form. Im oberen waagerechten Teil des T's wird die $L_1 \rightarrow L_2$ Übersetzung konkretisiert, im unteren senkrechten Teil ist die Sprache angegeben, in der der Compiler geschrieben ist. Handelt es sich dabei um JavaScript, kurz: JS, ist das betrachtete Programm lauffähig. Die niedrigste Abstraktionsebene für den verwendeten realen Computer ist also JS. Auch der Interpreter (rechts in der Abb. 5) läuft auf unserer JS-Maschine.
Im Beispiel wird eine frei erfundene Musiksprache ML in eine grafische Darstellung in SVG übersetzt (Notenzeile) bzw. mit einem ML-Interpreter in hörbare Töne umgewandelt. Die didaktische Reduktion auf kleine, überschaubare Sprachen wie ML liegt nahe. Audio-visuelle Zielsprachen sind zudem für Lernende motivierender als etwa Maschinencode.
Durch Anwendung eines *ML → SVG* Compilers auf ein Lied, das als Wort der Musiksprache *ML* vorliegt, entsteht ein entsprechendes Notenblatt (Grafik). *SVG* ist dafür ein sehr geeignetes Datenformat und kann im Browser dargestellt werden.

Für das Analysewort Hänschenklein in *ML*:
G0-4 E0-4 E0-2 F0-4 D0-4 D0-2
C0-4 D0-4 E0-4 F0-4 G0-4 G0-4 G0-2
G0-4 E0-4 E0-2 F0-4 D0-4 D0-2
C0-4 E0-4 G0-4 G0-4 C0-2

soll zum Beispiel als Ausgabe eine Notenzeile in *SVG* erzeugt werden (Abb. 6):

Abb. 6. Notenzeile

3.2 Formale Sprachbeschreibung

Natürlich müssen zunächst sowohl L_1 als auch L_2 formal definiert werden. Dies geschieht unter der Feststellung, dass Quellsprache L_1 und Zielsprache L_2 abzählbar unendliche Mengen sind. Mit einer Auflistung „aller" möglichen Musikstücke ist es also genauso wenig getan, wie mit der Auflistung aller Notenblätter. Die Zielsprache *SVG* ist bereits hinreichend in entsprechenden Dokumentationen formal beschrieben und die grundlegenden Elemente von *SVG* an Beispielen im Unterricht erarbeitet. Wir können uns somit ganz auf die formale Beschreibung unserer eigenen Sprache *ML* konzentrieren. Für L_1 = *ML* geben wir beispielsweise die folgende Grammatik G_{ML} an (Kasten 4):

$G_{ML} = (N, T, P, s)$

N = { Duration, Key, KeyName, Octave, Note, Notes, Pause, Song }

T = { **16**, **32**, **–**, **C**, **D**, **E**, **F**, **G**, **H**, **A**, **0**, **1**, **2**, **3**, **4**, **8**, **P** },

P = {

Song	→	Notes
Notes	→	Note | Note Notes
Note	→	Key - Duration | Pause - Duration
Key	→	KeyName Octave
KeyName	→	**C** | **D** | **E** | **F** | **G** | **H** | **A**
Octave	→	**0** | **1** | **2** | **3**
Duration	→	**1** | **2** | **4** | **8** | **16** | **32**
Pause	→	**P**

}

s = Song

Kasten 4. Formale Beschreibung der Musiksprache *ML*

In Abbildung 7 ist ein Syntaxdiagramm in FLACI gezeigt, welches auf Wunsch eingeblendet wird und die Grammatik grafisch zu editieren erlaubt.

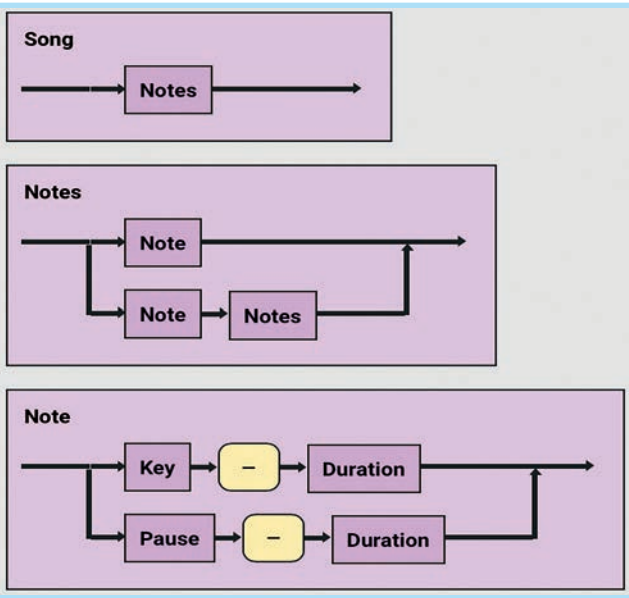

Abb. 7. Syntaxdiagramme für *ML* (Ausschnitt)

Statt per Hand einen abstrakten Automaten für die Sprache ML zu konstruieren, kann aus der bereits vorhandenen ML-Grammatik ein äquivalenter Kellerautomat durch Transformation auf Knopfdruck gewonnen werden. Es genügt damit anhand überschaubarer Beispiele abstrakte Automaten und ihre Arbeitsweise, sowie Automatentypen und deren Mächtigkeit einzuführen. Für komplexere Sprachen wie die Notensprache ML genügt anschließend die Betrachtung von Transformationsergebnissen. Dies relativiert insbesondere auch die praktische Bedeutungslosigkeit minimalistischer „ab“-Sprachbeispiele, die bei der Einführung abstrakten Automaten im Unterricht häufig verwendet werden, um den Konstruktionsaufwand überschaubar zu halten. Auch Sprachen wie ML lassen sich so mit einem Automaten abbilden und dessen Verarbeitung einer Eingabe mit FLACI taktweise visualisieren.

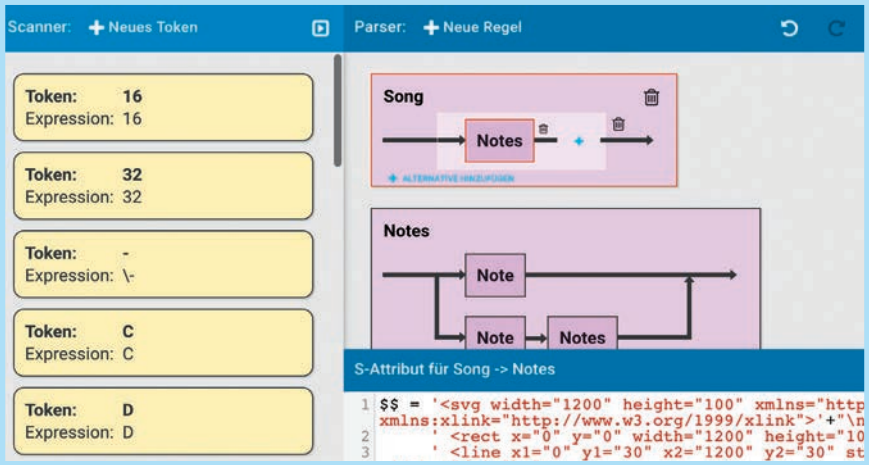

Abb. 9. Visuelle Compilerdefinition (TDL) in FLACI

3.3 Übersetzerbau

Nachdem die Sprachen *ML* mit verschiedenen Werkzeugen beschrieben und deren Verarbeitung betrachtet wurde, kehren wir zu dem in Abbildung 5 modellierten Übersetzungsprozess zurück. Statt einen Compiler oder Interpreter per Hand zu programmieren, entscheiden wir uns für einen generativen Ansatz mit Hilfe eines Compiler-Generators (Compiler Compiler). Dieser kann aus einer formalen Compilerdefinition automatisiert einen lauffähigen Compiler erzeugen. Das T-Diagramm wird durch die entsprechenden Vorverarbeitungsschritte erweitert (Abb. 8).

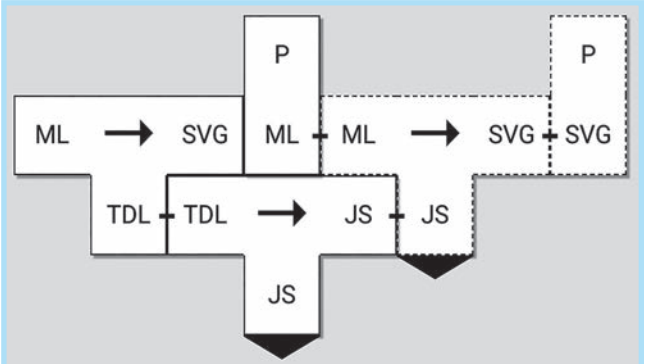

Abb. 8. Mehrteiliger Verarbeitungsprozess als Komposition von T-Diagrammen. Die gestrichelten Elemente werden als Ausgabe des jeweils vorherigen Schritts erzeugt.

Die Beschreibung eines Compilers oder Interpreters erfolgt in FLACI in der visuellen Transformation Description Language (TDL). Die TDL kombiniert die bereits erarbeiteten Beschreibungsmittel reguläre Ausdrücke und formale Grammatik zur Definition eines Scanners und Parser. Durch eine weitere, automatisierte Transformation der ML-Grammatik kann eine entsprechende Compilerdefinition in TDL gewonnen werden, die anschließend nur noch ergänzt werden muss. Dazu können gewünschte Ausgaben oder Aktionen bei den jeweiligen Parserregeln (S-Attribute) in Form von einfachen JavaScript-Befehle hinterlegt werden (Abb. 9).

Jetzt kann der mehrstufige Übersetzungsprozess auf Knopfdruck für beliebige Eingaben angestoßen und die Ausgabe betrachtet bzw. angehört werden (Abb. 10). Damit ist der gesamte Modellierungs- und Entwicklungsprozess eines einfachen Sprachübersetzers abgeschlossen. Treten Fehler auf, können die beteiligten Compiler (mit FLACI grafisch) separiert und individuell evaluiert werden. FLACI bietet für den gesamten Prozess entsprechende Unterstützungswerkzeuge an, das Unterrichtsbeispiel könnte grundsätzlich aber auch mit anderen bekannten Werkzeugen aus dem Compilerbau umgesetzt werden.

Eine ausführliche Beschreibung dieses Unterrichtsansatzes findet sich im Lehr- und Arbeitsbuch (Wagenknecht & Hielscher, 2014).

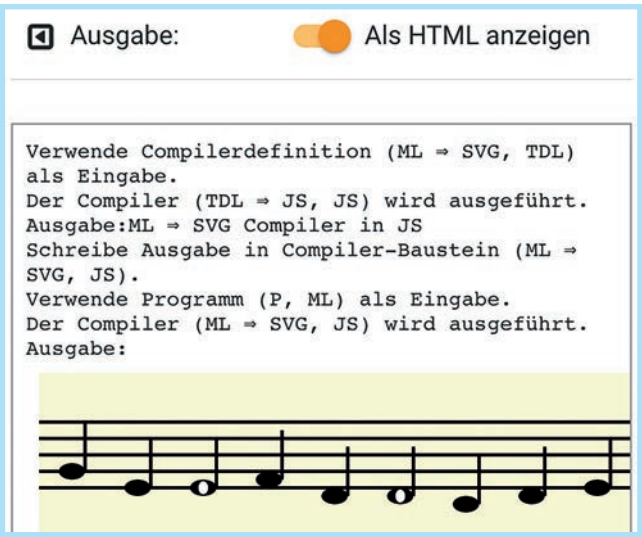

Abb. 10. Ausgabe eines Übersetzungsprozesses in FLACI

4 Fazit und Ausblick

Ähnlich wie im Chemieunterricht mit Atommodellen gearbeitet wird und Moleküle und Bindungen veranschaulicht werden, gibt es in der Informatik ebenso theoretische Modelle. Formalen Sprachen und Automaten als unliebsames Thema des Informatik-

unterrichts sollte deshalb nicht einfach ausgelassen werden. Es gilt attraktive Unterrichtskonzepte zu finden, die sowohl genügend Raum für die Einführung und Anwendung der Modelle der theoretischen Informatik bieten, als auch mit ganz praktischen Alltagsbezügen verknüpft werden können. In der Schweiz lernen die Schülerinnen und Schüler beispielsweise bereits in der 6. Klasse einen einfachen Zustandsautomaten am Beispiel eines Getränkeautomaten kennen (im Lehrmittel Connected Band 2 des Lehrmittelverlags Zürich).

Die Notwendigkeit der Vermittlung abstrakter Begriffe und zugehöriger Methoden führt zum Bedarf einer in den didaktischen Prozess integrierbaren Arbeitsumgebung. FLACI ist eine konsequente Weiterentwicklung des Vorgängersystems AtoCC (vgl. Wagenknecht & Hielscher, 2014) und ist abwärtskompatibel. Ein Umstieg bietet viele Vorteile. Als kostenlose, webbasierte Lösung stellt FLACI nur minimale Anforderungen an die Infrastruktur in der Schule und zuhause. FLACI ist quasi überall, zu jeder Zeit und in der aktuellsten Version verfügbar. Vielfältige Exportmöglichkeiten (z. B. PNG, SVG, LaTeX) unterstützen die Lehrperson zudem bei der Produktion von Unterrichtsmaterialien in diesem Themenbereich.

Obwohl mit dem Sprachübersetzer für eine selbstdefinierte Musiksprache ML ein konkretes Beispiel für die Umsetzung von TI-Inhalten im Informatikunterricht mit FLACI skizziert wurde, obliegt es den Lehrpersonen in den Schulen und Hochschulen stufengerechte Unterrichtsszenarien unter Zuhilfenahme von FLACI zu entwickeln. Als Autoren des beschriebenen Unterrichtskonzepts und Werkzeugs freuen wir uns auf Rückmeldungen aus der Praxis.

Literatur

Hielscher, M., Wagenknecht, C. (2019). FLACI – Eine Lernumgebung für theoretische Informatik. In A. Pasternak (Hg.), *Informatik für alle*. Proceedings der 18. GI-Fachtagung Informatik und Schule, INFOS 2019, Dortmund. Bonn: LNI, 211–220.

Hielscher, M.; Wagenknecht, C. (2009). Teaching language theory and automata: a compiler generation-oriented approach using ATOCC. *ACC Journal*, 15(1), 6–14.

Wagenknecht, C.; Hielscher, M. (2014): *Formale Sprachen, abstrakte Automaten und Compiler*. 2. Auflage. Wiesbaden: Springer.

Dr. Michael Hielscher, michael.hielscher@phsz.ch, forscht im Bereich digitaler Lehr- und Lernumgebungen und ist Dozent für Informatik in der Aus- und Weiterbildung von Lehrpersonen an der Pädagogischen Hochschule Schwyz, Schweiz.

Prof. Dr. Christian Wagenknecht, c.wagenknecht@hszg.de, beschäftigt sich seit ca. 40 Jahren mit fach- und mediendidaktischen Aspekten der Informatik, Lehr-/Lernumgebungen für theoretische Informatik, Programmierparadigmen und der Entwicklung von Web-Anwendungen. Er lehrt an der Hochschule Zittau/Görlitz, Deutschland.