AtoCC – didaktischer Ort und erste Erfahrungen

Michael Hielscher, Christian Wagenknecht

Fachbereich Informatik
Hochschule Zittau/Görlitz (FH)
Brückenstraße 1
D-02826 Görlitz
mail@atocc.de
c.wagenknecht@hs-zigr.de

Abstract: AtoCC (http://www.atocc.de) ist eine modular aufgebaute Lehr-/ Lernumgebung für ausgewählte Inhalte der theoretischen Informatik (formale Sprachen und Automaten) und Grundlagen des Compilerbaus. Sie unterstützt Schülerinnen und Schüler bei der Anwendung von Theorie-Kenntnissen in einem Projektbezogenen Umfeld (Compilerkonstruktion). Das inzwischen etablierte Werkzeug AutoEdit [HW05, Wi06] wurde dafür mit weiteren Softwarekomponenten angereichert, um ein kompaktes System für den Unterricht zu schaffen, das durch abgestimmte Handhabung den Blick auf die eigentlichen Lerninhalte richtet.

1 Einleitung

Sowohl an Universitäten und Hochschulen als auch an Gymnasien[1] werden Kerninhalte der theoretischen Informatik gelehrt. Die im hessischen Lehrplan festgelegten Inhalte kommen einem Theorie-Grundkurs eines Informatik-Studiums recht nahe, vgl. [Sc92].

Begriffe und Methoden der theoretischen Informatik sind sehr abstrakt und stellen gegenüber eher ingenieurwissenschaftlichen Arbeitsformen anderer Teilgebiete der Informatik eine besondere didaktische Herausforderung dar. Von daher ist die Entwicklung geeigneter Lehr/Lernumgebungen besonders motiviert. So gibt es zur Unterstützung der Behandlung von Algorithmen entsprechende Visualisierungssoftware, wie [RAK06] und [Rö05], die ggf. durch Parametervariationen einen what-if-Arbeitsstil ermöglicht. Im Allg. ist Software dieser Art jedoch auf einen eng begrenzten Schwerpunkt beschränkt und bietet keinen Raum für Konstruktion neuen Wissens und Kontext-Erweiterung. Außerdem ist die auf das Lernen gerichtete Aktivität bei Visualisierungen relativ gering.

[1] Themen, wie formale Sprachen, formale Grammatiken, Automatentheorie und Übersetzerbau, sind beispielsweise im hessischen Lehrplan der Jahrgangsstufe 13 im Pflichtteil von Grund- und Leistungskurs „Konzepte und Anwendungen der Theoretischen Informatik“ [LIH] und auch in den Lehrplänen anderer Bundesländer zu finden.

Im Bereich der Automatentheorie findet man eine Vielzahl von Applets und Simulationstools, wie JFLAP [RF06] oder Kara [RNH04]. Eine Abgrenzung von AtoCC gegenüber dem bekannten Werkzeug JFLAP findet sich in [HW05].

Automatentheorie wird im Übersetzerbau angewendet. Neben dem fachsystematischen Aspekt ist dies aus didaktischer Sicht (Verstehen abstrakter Inhalte durch Anwendung/Instanziierung) von besonderer Bedeutung. Hierfür werden gern einfache Scanner und Parser für überschaubare (LL(1)-)Sprachen in einer bereits bekannten Programmiersprache implementiert. Im Informatiklehrplan des Saarlands wird beispielsweise auf das Verfahren des rekursiven Abstiegs zur Syntaxanalyse Bezug genommen [LISL].

Zur konzeptionellen vs. informationstechnischen Behandlung des Übersetzerbaus ist ein Top-Down-Vorgehen angezeigt. Dies entspricht genau dem, was man unter automatisierter Compiler-Entwicklung (kurz: compiler construction – das CC in AtoCC) versteht. Statt ein Übersetzungsprogramm in einer bestimmten Programmiersprache zu schreiben, werden aus der Grammatik-Definition (sprachliche) Beschreibungen für einen Übersetzer-Generator gewonnen. Die hierfür in der Berufspraxis verwendeten Werkzeuge Lex und Yacc[2] sind für didaktische Zwecke ungeeignet, da sie gegenüber den zugrunde liegenden Automaten-Modellen einen Paradigmenbruch hinnehmen und auf Performance angelegtes technisches Beiwerk erfordern. Eine Ableitung der Compiler-Beschreibung aus einem vorher entwickelten Automatenmodell ist in Verbindung mit diesen Werkzeugen nicht möglich. AtoCC unterstützt die Entwicklungskette vom Automat zum Compiler in einem modular strukturierten System mit vereinheitlichter Bedienung.

Die im Compilerbau auftretenden Prozesse werden gern durch T-Diagramme beschrieben und visualisiert. Schon auf Papier eignen sich T-Diagramme hervorragend, um Konzepte wie Bootstrapping, mehrstufige und Cross-Compiler zu beschreiben. Allerdings finden Papierdarstellungen keine Fortsetzung in der tatsächlichen Compilerentwicklung. Diesen Medienbruch (Papier – Softwaresystem) überwindet die TDiag-Komponente von AtoCC. Dabei wird nicht nur die grafische Darstellung nach Baustein-artigem Zusammensetzen einzelner T-Diagramme von der Software übernommen, sondern darüber hinaus findet eine Prüfung der Passfähigkeit benachbarter Bausteine statt. Auf diese Weise entstehen also stets korrekte Übersetzungsketten. Hiermit wird selbstgesteuerter Wissenserwerb durch zielgerichtetes Explorieren / Konstruieren tutoriell begleitet.

2 AtoCC – „Vom Automaten zum Compiler Compiler“

Bisherige Beobachtungen zeigen, dass die Verwendung von AtoCC in der Hand von Lehrenden und Schüler/innen eine sehr kurze Einarbeitungszeit erfordert. Dies ist auf die einheitliche Benutzungsoberfläche aller Systemkomponenten zurückzuführen und war von Beginn an ein Entwicklungsziel. Integrierte AtoCC-Assistenten mit einfachen Einführungsbeispielen tragen zusätzlich dazu bei.

[2] Lex ist als Scannergenerator und Yacc als Parsergenerator auch unter den Weiterentwicklungen Flex und Bison bekannt, s. auch http://dinosaur.compilertools.net/.

Im Gegensatz zu zersplitterten Systemarchitekturen stellt die auf einander abgestimmte Weiterverwendbarkeit von Aufgabenlösungen (z.B.: Generierung eines Compilers aus einem Automaten oder Generierung eines T-Diagramms für einen erstellen Compiler) in zuständigen System-Komponenten einen wichtigen konzeptionellen Schwerpunkt von AtoCC dar. Das Zusammenspiel aller Komponenten im Hinblick auf einen einzigen programmiersprachlichen Träger[3] beschleunigt die „Produktherstellung“ (Compiler) und reduziert den in anderen Systemen beklagten Ballast. AtoCC sollte aber keineswegs als eine einfache, thematisch abgestimmte Tool-Sammlung verstanden werden. Durch vielseitige Beziehungen zwischen den Komponenten entsteht ein komplexes System, s. Abb. 1. und [HW06].

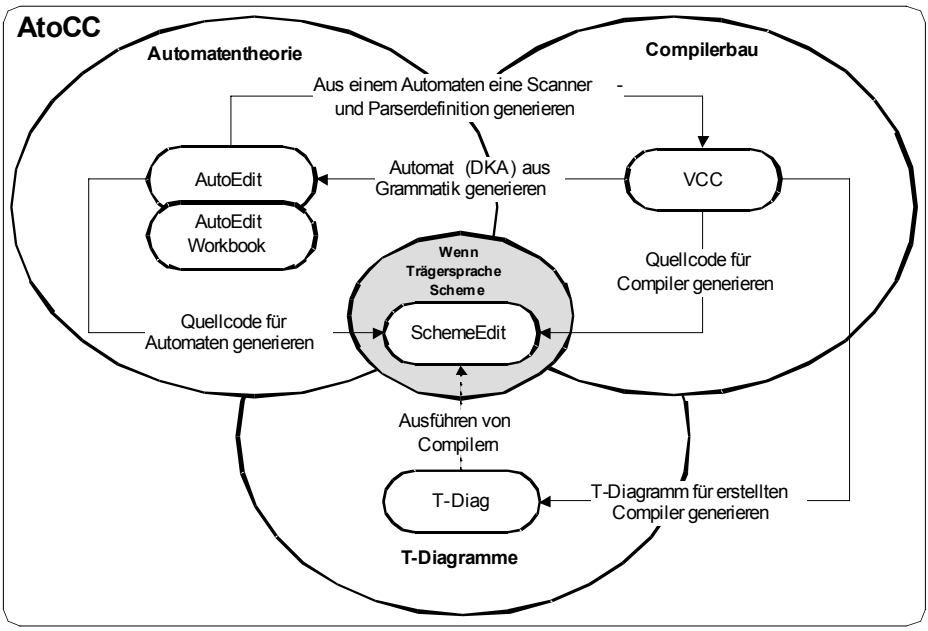

Abbildung 1: AtoCC-Komponenten und ihre Wechselwirkungen

Folgende Grundaufgaben können mit AtoCC bearbeitet werden:

- Darstellung, Simulation und Transformation von Automaten (deterministische und nichtdeterministische endliche Automaten – DEA, NEA; deterministische und nichtdeterministische Kellerautomaten – DKA, NKA; Turingmaschinen; Mealy- und Moore-Automaten)
- Automatisierte Konstruktion von Compilern (Scanner, Parser, Codegenerator)
- Konstruktion von T-Diagrammen zur konzeptionellen Planung von Übersetzungsprozessen und deren Visualisierung (Modell-getriebener Ansatz)

[3] Hierfür können in AtoCC derzeit die Programmiersprachen Scheme, Java und C# .NET gewählt werden.

Die dazu im Allg. im Verbund eingesetzten Module werden im Folgenden zusammen mit einer kurzen didaktischen Ortsbestimmung vorgestellt.

2.1 AutoEdit und AutoEdit Workbook

An der Hochschule Zittau/Görlitz wird im Fach „Theoretische Informatik“ bereits seit Jahren AutoEdit als Darstellungs-, Simulations- und Transformationswerkzeug für Automaten in Seminaraufgaben eingesetzt. Durch viele Vorschläge von Anwendern im Schul- und Hochschulbereich konnte die didaktische Qualität und der Leistungsumfang des Werkzeugs maßgeblich gesteigert werden. Mit dem Lehrbuch [Wa06] wird AutoEdit auch erstmals in entsprechenden Unterrichtsinhalten verankert.

AutoEdit Workbook, als Erweiterung zu AutoEdit, bietet Übungsaufgaben zur selbstständigen Bearbeitung und anschließenden automatischen Bewertung an. Dieses Konzept für Übungsaufgaben ist den Projekten SchemeGrader [Wa05] und Exorciser [TLN02] entlehnt, bei denen ebenfalls eine direkte Überprüfung einer Aufgabenlösung erfolgt. Eine Webdatenbank stellt diese Übungsaufgaben allen Lehrenden und Lernenden zur Verfügung[4]. Die jeweils aktuelle Aufgabenbasis kann von Lehrer/innen mit Hilfe eines integrierten Assistenten jederzeit erweitert werden. Auch Schüler/innen können für gute Ideen durch deren Eintrag ins Workbook gewürdigt werden. Die Aufgabenstellungen können mit Hinweisen (Notizzettel), Musterlösungen und bestimmten Vorgaben (Hinweise, Teillösungen, die zu vervollständigen sind) versehen werden.

2.2 VCC – Visual Compiler Compiler

VCC hilft bei einer Projekt-basierten Vermittlung ausgewählter Konzepte des Compilerbaus. Dabei steht weniger die Effizienz des Produkts, sondern vielmehr der didaktische Zugang im Vordergrund: Schüler/innen können mit VCC einen Compiler nach dem Baukastenprinzip visuell erstellen. Definierte Bausteine, wie die für die Tokens des Scanners, dienen als Grundlage für die spätere Konstruktion des Parsers. Das von AutoEdit bekannte Prinzip Software-geleiteter Arbeitsschritte wird auch hier eingesetzt.

Die visuelle Konstruktionstechnik vermeidet syntaktische und Schreibfehler weitestgehend. Da VCC intern auf einer angepassten Version von YACC beruht, ist sowohl inhaltlich (Tokendefinitionen, S-Attribut-Grammatiken) als auch technisch (Kellerautomat) ein mit Lex und Yacc vergleichbares Vorgehen garantiert. VCC vereint Scanner-, Parser- und Codegenerator in einem einzigen Werkzeug und legt erstellte Compilerdefinitionen in einem vielseitig verwendbaren XML-Format ab.

2.3 TDiag – T-Diagramme

T-Diagramme sind geeignet, um Compilerprozesse zu planen (Entwurfsperspektive) und zu visualisieren (Verifikationsperspektive). Aus den vier Grundbausteinen Compiler,

[4] Der Aufgabenautor kann ggf. festlegen, dass die jeweilige Aufgabe nur über ein Passwort zugänglich ist.

Interpreter, Ein-/Ausgabe und Programm lassen sich nahezu alle anfallenden Prozesse modellieren[5].

TDiag bietet ein einfaches Interface, um diese Diagramme aus einzelnen Bausteinen zu konstruieren. Dem Lernenden wird durch entsprechende Einfärbungen der grafischen Bausteine angezeigt, ob und wie einzelne Komponenten zusammengesetzt werden können. Auf diese Weise entstehen stets korrekte Prozessmodelle.

Um komplexe Kompilationsprozesse abzuarbeiten werden Stapelverarbeitungsdateien[6] verwendet. In TDiag kann ein entworfenes Diagramm mit einem einzigen Klick in eine Batch-Datei, die dann automatisch abgearbeitet wird, übersetzt werden. Die Schüler/innen können somit direkt überprüfen, inwieweit das entwickelte Diagramm dem gewünschten Prozess entspricht. Um dies zu ermöglichen, bietet jeder Baustein zusätzliche Runtime-Attribute, die im Diagramm nicht direkt sichtbar sind.

3 Ein kommentiertes Unterrichtsbeispiel

Das folgende Beispiel wurde einer realen Unterrichtssituation entnommen. Es handelt sich keineswegs um ein Einführungsbeispiel, sondern ist von mittlerem Schwierigkeitsgrad und illustriert die medien- und paradigmenbruchfreie Entwicklung eines Compilers innerhalb von AtoCC beginnend beim Automatenmodell.

Aufgabenstellung:

> Entwickeln Sie einen DEA[7], der alle Wörter akzeptiert, die eine durch 4 teilbare[8], natürliche Zahl repräsentieren.
>
> "182342340" wird akzeptiert, da 40 durch 4 teilbar ist.
>
> "234523173" wird nicht akzeptiert, da 73 nicht durch 4 teilbar ist.
>
> Wählen Sie zunächst ein passendes Eingabealphabet Σ.
>
> Um die Funktionsweise Ihres Automaten zu überprüfen, generieren Sie 10 Zufallswörter über Σ unter „Mehrere Eingaben prüfen“ (auf der Simulationsseite).

Eine mögliche Schülerlösung, die mit AutoEdit erarbeitet wurde, zeigt Abb. 2. Vom Schüler wird allerdings die vollständige Tupel-Definition des Automaten gefordert.

Der Entwurfsprozess des Automaten wird vom AtoCC-Programm in bestimmten Arbeitsschritten gesteuert („weiter“-Knopf – unten rechts): Hiernach beginnt die Definition eines Automaten stets mit der Festlegung des Eingabealphabets usw.

---

[5] Eine Ausnahme bildet der Compiler Compiler, für den bislang kein entsprechender Baustein definiert wurde.
[6] Unter UNIX/Linux verwendet man Shell-Scripts um Folgen von Befehlen automatisiert abarbeiten zu lassen. Vergleichbar erfüllen MS-DOS-Batchdateien unter Microsoft Betriebsystemen diese Aufgabe.
[7] Deterministischer endlicher Automat
[8] Hinweis: Es gilt, dass alle mehrstelligen Zahlen durch 4 teilbar sind, wenn die Zahl aus den letzten beiden Stellen durch 4 teilbar ist.

Nach Fertigstellung des Automatenentwurfs verwenden die Schüler den Simulationsmodus: Neben der freien Eingabe eines Eingabewortes und dessen automatischer bzw. Einzelschritt-Verarbeitung (visualisierte Durchwanderung der betreffenden Knoten des Graphen) hat sich die Verarbeitung generierbarer Zufallswörter wählbarer Länge als sehr nützlich erwiesen. Auf diese Weise kann die Verifikation des entwickelten Automaten wesentlich befördert werden.

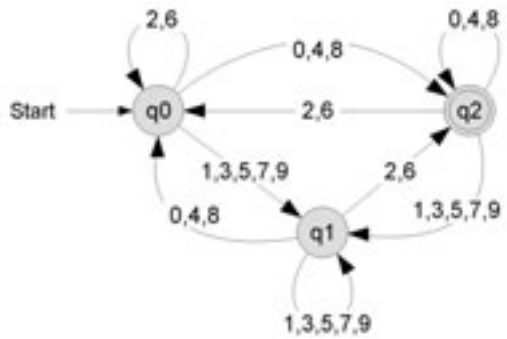

Abbildung 2: DEA für alle durch 4 teilbare Zahlen

Handelt es sich um eine Aufgabe aus dem Workbook, so kann eine Musterlösung hinterlegt werden. Diese dient der Selbstkontrolle der erarbeiteten Lösung durch die Schüler. Ein Notizzettel kann Lösungshinweise zur Aufgabe bereitstellen. Für schwierigere Aufgaben werden Lösungsrudimente in Gestalt einer mehr oder weniger ausgearbeiteten Vorgabelösung zur Verfügung gestellt. Auf diese Weise ist eine Binnendifferenzierung erreichbar. Auch für Hausaufgaben ist dies eine wertvolle Hilfe.

Bei der Darstellung der vollständigen Automatendefinition wird auch eine zugehörige formale Grammatik angegeben. Dies bietet eine zusätzliche Lernmöglichkeit.

```
G = (N,T,P,s)
G = ({q0,q2,q1},{0,1,2,3,4,5,6,7,8,9},P,q0)
P = {
  q0 -> 0q2 | 0 | 4q2 | 4 | 8q2 | 8 | 1q1 | 3q1 | 5q1 | 7q1 | 9q1 | 2q0 | 6q0
  q2 -> 0q2 | 0 | 4q2 | 4 | 8q2 | 8 | 1q1 | 3q1 | 5q1 | 7q1 | 9q1 | 2q0 | 6q0
  q1 -> 0q0 | 4q0 | 8q0 | 1q1 | 3q1 | 5q1 | 7q1 | 9q1 | 2q2 | 2 | 6q2 | 6
}
```

Der betrachtete Automat kann von AutoEdit automatisch in eine Scanner- und eine Parserdefinition für VCC transformiert werden. Dabei werden die Terminale der zugehörigen Grammatik (s. o.) in Tokens für den Scanner überführt. Die Produktionsregeln dieser Grammatik werden für den Parser verwendet. Die generierten Definitionen für Scanner und Parser zeigt Abb. 3.

Sowohl Automat, Grammatik als auch Parser- in Kombination mit der Scanner-Definition sind äquivalente Beschreibungsmittel für die in der Aufgabenstellung beschriebene Sprache.

Der Parser liefert einen Syntaxfehler, wenn das aktuelle Eingabewort eine Zahl repräsentiert, die nicht durch 4 teilbar ist. Der Parser realisiert damit das Verhalten eines Akzeptors. Je nach gewünschter Zielsprache ist es möglich, mit AtoCC einen vollwertigen Compiler zu generieren. Der Compiler liegt dann als ausführbares Programm vor[9].

Abbildung 3: VCC generierter Scanner und Parser

Nachfolgend wird ein Aufrufbeispiel in Scheme sowohl für den Scanner als auch für die Verkettung von Scanner und Parser angegeben. Der Scanner liefert dabei eine Liste von Token-Value-Paaren. Der Parser gibt entweder „wahr“ oder „falsch“ zurück und folgt damit dem oben angesprochenen Akzeptor-Verhalten:

```
> (scanner "1234")
((token_1 . "1") (token_2 . "2") (token_3 . "3") (token_4 . "4"))
> (parser (scanner "12348082345678876544564"))
#t
> (parser (scanner "1234808234567887654456423 21"))
Syntax Error
#f
```

Ein Aufruf der Form (compiler "Eingabedatei.txt" "Ausgabedatei.txt") wird verwendet, um die gesamte Übersetzung (mit Zielcodegenerierung) durchzuführen.

Jede einzelne Regel in VCC kann hierfür mit einem ausführbaren Code-Schnipsel verbunden werden (S-Attributgrammatik). Diese werden während des Parsens bei erfolgreicher Regelanwendung ausgeführt. Um dies zu demonstrieren wird in unserem Beispiel ein neues Spitzensymbol „Start“ angelegt. Die Regel Start→q0 wird mit der Ausgabe "Die Zahl " $1 " ist durch 4 teilbar." belegt, s. Abb. 4.

In die Datei „eingabe.txt“ wird die Zeichenkette "12344" geschrieben und im Compilerverzeichnis abgelegt. Durch den nachfolgenden Aufruf wird der Eingabetext in eine sehr einfache Zielsprache übersetzt:

```
> (compiler "eingabe.txt" "ausgabe.txt")
#t
```

[9] Wahlweise handelt es sich um ein Scheme-, Java oder C#-Programm. Detaillierte Kenntnisse dieser Sprachen sind nicht erforderlich. Lediglich die jeweilige Ausführungsumgebung muss (einmalig) bereitgestellt werden.

Abbildung 4: S-Attribute in VCC

In der Datei „ausgabe.txt“ steht nun der Inhalt: „Die Zahl 12344 ist durch 4 teilbar.“ Hierfür wurden Platzhalter wie $n für einzelne Regelelemente verwendet.

Für einen erstellten Compiler kann jederzeit per Mausklick ein zugehöriges T-Diagramm generiert werden (Abb. 5).

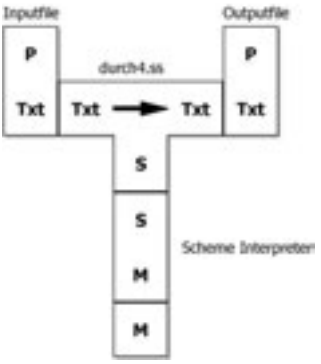

Abbildung 5: T-Diagramm für Beispiel-Compiler

Die (vom System generierte) Beschreibung des Übersetzungskonzepts durch ein T-Diagramm ist bei Wieder- bzw. Weiterverwendung „abgelegter“ Lösungen durchaus sinnvoll: Hier sollen sich Schülerinnen und Schüler am Modell orientieren können.

Der typische Arbeitsstil verläuft jedoch umgekehrt, nämlich als Modellierungsaufgabe der Architektur des zu entwickelnden Übersetzers, wie z. B. Bootstrapping, mehrstufige und Cross-Compiler. Hierfür bieten T-Diagramme eine hervorragende Entwurfsunterstützung und machen konkrete Lehrplanziele [LIH] erreichbar.

In Abb. 6 ist ein T-Diagramm für einen mehrstufigen Compilerprozess dargestellt. Eine Sprache MyLang wird zunächst von einem zu entwickelten Compiler in die Sprache TASM (Turbo Assembler) übersetzt, anschließend von TASM kompiliert (Bildmitte) und von TLINK in ein lauffähiges Programm überführt. Die Programme TASM.exe und TLINK.exe werden zur Bearbeitung der Aufgabe zur Verfügung gestellt. Die eigentliche

Aufgabe besteht also im Entwurf von MyLang→TASM. Der Rest besteht aus technischem Beiwerk, was fast vollständig von AtoCC übernommen wird.

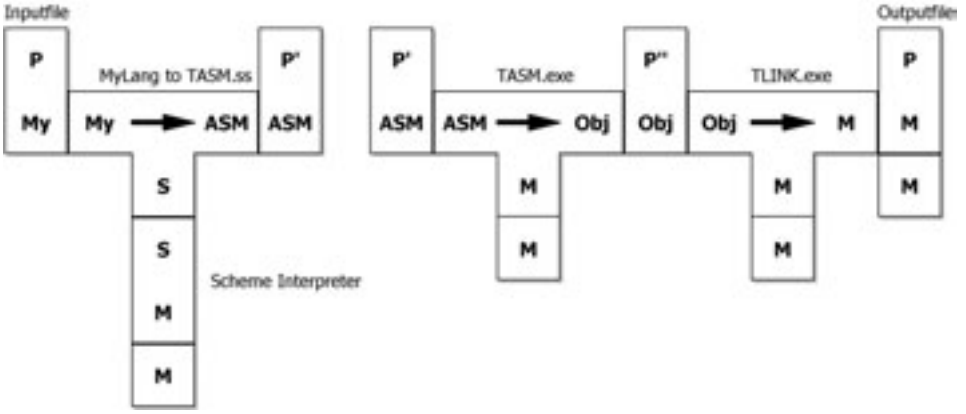

Abbildung 6: T-Diagramm für MyLang → TASM Compiler

Schließlich wird eine Stapelverarbeitungsdatei generiert, die die einzelnen Übersetzungsschritte in einem Aufruf zusammenfasst. TDiag liefert die Ausgabe wie in Abb. 7.

```
 1 ;Starting batch file that will execute the diagram.
 2 ;=================================================
 3 Starting: petite --script tasm.ss quelltext.txt out.asm
 4 ...
 5
 6 Done with: petite --script tasm.ss quelltext.txt out.asm
 7 ;=================================================
 8 Starting: tasm.exe out.asm out.obj
 9 ...
10 Turbo Assembler  Version 2.0  Copyright (c) 1988, 1990 Borland International
11
12 Assembling file:   out.asm
13 Error messages:    None
14 Warning messages:  None
15 Passes:            1
16 Remaining memory:  431k
17
18
19 Done with: tasm.exe out.asm out.obj
20 ;=================================================
21 Starting: tlink.exe out.obj
22 ...
23 Turbo Link  Version 3.01 Copyright (c) 1987, 1990 Borland International
24
25 Done with: tlink.exe out.obj
26 ;=================================================
27 Starting: out.exe
28 ...
29 **********************
30  Dahler
```

Abbildung 7: Ausgabe bei „Ausführung des T-Diagramms“

Die generierte Stapelverarbeitungsdatei kann unabhängig von AtoCC beliebig oft verwendet werden.

4 Erfahrungen im praktischen Einsatz von AtoCC

AtoCC hat die zweite Entwicklungsphase abgeschlossen. Während in der ersten Phase der System-Entwurf auf der Basis von Lehrerfahrungen im Hochschulbereich im Mittelpunkt stand, widmete sich die zweite Phase der Systemmodifikation in Reaktion auf entsprechende Rückkopplungen von Lehrenden an Hochschulen und Gymnasien sowie von Studierenden. Die hier berichteten Erfahrungen stammen aus dieser ca. zweijährigen Phase und haben in den meisten Fällen zu didaktisch motivierten Systemverbesserungen bzw. -erweiterungen beigetragen. Einige Beispiele:

- eine an typischen IDEs und den taktilen Gewohnheiten orientierte Maussteuerung kombiniert mit der Eingabemöglichkeit textueller Attribute beim Automaten- und T-Diagramm-Entwurf
- Ergänzung eines Notizblocks, um einem Automaten auch einfache Zusatzinformationen anheften zu können
- Modifikation der Simulationskomponente für nichtdeterministische Automaten
- Sequentielle Simulation mehrerer Zufallswörter wählbarer Länge → Testunterstützung und Vorgabemöglichkeit von Eingabewortlisten für Automaten
- Aufnahme von Automaten mit Ausgabe – Mealy und Moore (s. [LIH])
- Ausbau der Aufgabensammlung (Workbook-Komponente) mit erprobten Beiträgen von Lehrerinnen und Lehrern
- Konsolidierung der Lernendenführung bei der schrittweisen Entwicklung eines Automaten bzw. eines Übersetzers

In der nächsten Phase stehen die didaktisch-methodischen Aspekte im Vordergrund. Hierzu ist es notwendig, AtoCC weiter bekannt zu machen und an der Einführung mitwirkende Lehrende zu gewinnen.

Wir schon weiter oben festgestellt, fordert der hessische Lehrplan [LIH] in der 13. Jahrgangsstufe eine sehr breite inhaltliche Palette dieser Themen. Aufgrund der dort geforderten ‚Simulation realer Automaten, wie etwas Getränkeautomaten', müssen neben den bekannten Akzeptoren sogar Automaten mit Ausgabe (Mealy, Moore) thematisiert werden. Auch dies ist inzwischen mit AtoCC umsetzbar.

Zur Sicherung einer Lehrplan-konformen Wissensvermittlung und Kompetenzentwicklung kann AtoCC eine enorme Hilfe sein. Bisher empfohlenen Systemen ist AtoCC im Allg. überlegen, siehe z. B. die „Simulationsumgebung" in [LIH]. Dies untermauern zahlreiche positive Rückmeldungen, wie: „Von dem Programmpaket AtoCC kenne ich bisher nur die AutoEdit Komponente. Ich habe damit im vergangenen Schuljahr das erste Mal gearbeitet. Das Programm hat mir sehr gut gefallen und ich möchte weiter damit arbeiten.

Das Programm ist sehr bedienerfreundlich und veranschaulicht die Inhalte der in Frage stehenden Themen hervorragend. Durch die geringen Systemanforderungen und automa-

tisierte Installation der Software (von nur wenigen ca. 1.5 MB) war es auch Schülerinnen und Schülern mit älteren Systemen (wie Win98, WinME) möglich, AtoCC für Hausaufgaben zu verwenden.“ Nach Aussagen befragter Schüler/innen ist die Bedienung von AtoCC intuitiv. Sowohl Schüler/innen als auch Lehrende überzeugte dabei besonders die visuell und konzeptionell einheitliche Gestaltung der einzelnen Module, die nahtlos „zugeschaltet“ werden können. Die Modularisierung von AtoCC erlaubt eine gezielte Auswahl einzelner Werkzeuge zur Behandlung unabhängiger Schwerpunkte.

Andererseits wurde auch deutlich, dass LehrerInnen aus verschiedenen Gründen nicht in der Lage sind, die didaktischen Möglichkeiten des Systems im Selbststudium vollständig zu erschließen. Zur Unterstützung gibt es eine Plattform, die Tutorials sowie Aufgaben mit Musterlösungen bereithält. Darüber hinaus wird die Durchführung von Workshops im Rahmen geeigneter Fortbildungsmaßnahmen angestrebt. Hat man erst einmal die richtige Erwartungshaltung aufgebaut, ist die Einarbeitungszeit sehr kurz.

Die Betrachtung von Übersetzungsprozessen ist für das Verständnis der Verarbeitung höherer Programmiersprachen unentbehrlich. Dies gilt nicht nur für das Informatik-Studium, sondern ebenso für den Pflichtbereich des Informatik-Unterrichts in der gymnasialen Oberstufe natürlich mit angemessener Betrachtungstiefe. In mehreren Bundesländern findet sich dieser Themenschwerpunkt im Wahlpflichtbereich, z. B. NRW (Siegen). Sind Lehrplanziele aus dem Übersetzerbau ohne AtoCC wirklich erfüllbar? Der Einsatz von VCC/TDiag würde auch dem hessischen Lehrplan eine echte Realisierungschance einräumen: Dort werden Delphi bzw. Java als Basissprachen für die Grundlagen der Programmierung empfohlen, Prolog für den Übersetzerbau. Da VCC optional Java-Code generiert, kann ein Paradigmenbruch (objekt-orientiert vs. logik-basiert) vermieden werden.

5 Ausblick

Die Systementwicklung ist grundsätzlich abgeschlossen, so dass die nun verstärkt einsetzende didaktische Erprobungsphase nicht durch immer neue Versionen beeinträchtigt wird. Dennoch arbeiten wir an internen Verbesserungen, wie beispielsweise an der Erzeugung minimaler regulärer Ausdrücke als Ergebnis der Transformation aus endlichen Automaten. Die theoretischen Grundlagen für eine geeignete Approximation in vertretbarer Zeit liegen inzwischen vor [GS05]. Unter Verwendung dieser minimierten regulären Ausdrücke kann auch die Transformation deterministischer in nichtdeterministische endliche Automaten deutlich verbessert werden.

Ab sofort soll AtoCC vor allem in Workshops, die sich an den entsprechenden Lehrplaninhalten orientieren, stärker bekannt gemacht werden. Dies gilt sowohl national als auch international, was durch die Herstellung einer englischen und einer polnischen Version vorbereitet wurde. Begleitet wird dies durch den Ausbau des Workbooks und Verbreiterung der AtoCC-Kommunikationsplattform, die bereits jetzt Tutorials und Handreichungen bereithält.

Literaturverzeichnis

[Wa05] Wagenknecht, Chr.: Mediendidaktische Begleitung im Informatikunterricht mit SchRepo, SchemeNet und SchemeGrader. In (Ed.: H. Rohland): Unterrichtskonzepte für informatische Bildung -Praxisband-. INFOS'05 28.-30.09.05 Dresden , Seite 21-24.

[Wa06] Wagenknecht, Chr.: Theoretische Informatik.- In (Engelmann, L. Hrsg.): Informatik: Lehrbuch S II - ISBN 3-89818-622-9, Berlin: DUDEN PAETEC, 2006.

[GS05] Gramlich, G.; Schnitger, G.: Minimizing NFA's and Regular Expressions. In (Diekert, Volker Hrsg.): STACS 2005, 22nd Annual Symposium on Theoretical Aspects of Computer Science Stuttgart, Germany 24.-26. February 2005). Bd. 3404. Heidelberg: Springer-Verlag, 2005, S. 399-411.

[HW06] Hielscher, M.; Wagenknecht, Chr.: AtoCC - Learning Environment for Teaching Theory of Automata and Formal Languages. ITiCSE'06, June 26-28, 2006, Bologna, Italy. ACM 1-59593-055-8/06/0006

[HW05] Hielscher, M.; Wagenknecht, Chr.: AutoEdit - ein Werkzeug zum Editieren, Simulieren, Transformieren und Publizieren abstrakter Automaten. In (Ed.: H. Rohland): Unterrichtskonzepte für informatische Bildung -Praxisband-. INFOS'05 28.-30.09.05 Dresden, Seite 25-27.

[LIH] Informatik-Lehrplan Hessen: http://lernarchiv.bildung.hessen.de/reposit2/12226/LPGymInformatik.pdf?user_id=Anonymous+User&is_viewable=1&digest=6fa80232cfce34daa0499dbfdf175e11

[LINR] Informatik-Lehrplan NRW: http://www.learn-line.nrw.de/angebote/abitur-gost-07/download/inf-vorgaben-2007.pdf

[LISH] Informatik-Lehrplan SH: http://lehrplan.lernnetz.de/intranet1/links/materials/1107165545.pdf

[LISL] Informatik-Lehrplan Saarland: http://www.saarland.de/dokumente/thema_bildung/INFeb2006.pdf

[RNH04] Reichert, R.; Nievergelt J.; Hartmann W.: Programmieren mit Kara. Ein spielerischer Zugang zur Informatik. 2. erweiterte Auflage, ISBN: 3540238190, Springer, Berlin Dezember 2004.

[RF06] Rodger, S.; Finley, T.: JFLAP - An Interactive Formal Languages and Automata Package, ISBN 0763738344, Jones and Bartlett, 2006

[Rö05] Rößling, G.: Visualisierung von dynamischen Systemen. thema Forschung. E-Learning 1/2005. pp. 78-82, TU Darmstadt, 2005. ISBN 1434-7768 (ISSN).

[RAK06] Rößling, G.; Ackermann, T.; Kulessa, S.: Visualisierung von Algorithmen und Datenstrukturen. – In (Mühlhäuser, Rößling, Steinmetz Hrsg.): DeLFI 2006 4. e-Learning Fachtagung Informatik 11. - 14. September 2006 in Darmstadt, Seite 231-242.

[Sc92] Schöning, Uwe: Theoretische Informatik kurz gefasst.- Mannheim u.a.: BI Wissenschaftsverlag, 1992.

[TLN02] Tscherter, V.; Lamprecht, R.; Nievergelt, J.: Exorciser: Automatic Generation and Interactive Grading of Exercises in the Theory of Computation. Proceedings of ICNEE 2002, Lugano, Switzerland, May 2002.

[Wi06] Wikipedia; http://de.wikipedia.org/wiki/AtoCC; 21.01.2007