FÜNF JAHRE AUTOLAT-EINSATZ AN DER UNIVERSITÄT LEIPZIG

Hans-Gert Gräbe
Universität Leipzig
graebe@informatik.uni-leipzig.de
Version vom 05.08.2015

Zusammenfassung

Das von Johannes Waldmann entwickelte A*utotool* ermöglicht es, spezifische Konzepte in der MINT-Ausbildung durch interaktiven Zugang zu einem E-Learning-Werkzeug zu unterstützen, welches die zu lehrenden Kalküle „beherrscht“. Damit ist ein Übungsbetrieb ganz neuer Qualität möglich, der nicht nur die Personalressourcen schont, sondern sich auch durch einen hohen Lerneffekt für die Studierenden auszeichnet, der sich aus der Möglichkeit ergibt, sofort weitere Lösungsversuche einzureichen, in denen die Hinweise des Werkzeugs zu früheren Lösungsversuchen berücksichtigt sind.

Eine Autotool-Anbindung ist im OpenOlat-Portal der Fakultät für Mathematik und Informatik seit 2010 im Einsatz. Über die Ergebnisse von fünf Jahren Einsatz dieser Autotool-Anbindung wird hier ebenso berichtet wie über das Potenzial, derartige Aufgaben als *Open Educational Resources* mit interessierten Lehrenden anderer Einrichtungen auszutauschen.

1 Einführung

In der Session 4 *Forschendes Lernen mit Beispielen aus der Lehrpraxis* des zweiten „Tags der Lehre“ an der Universität Leipzig am 12. November 2014 thematisierte Dr. Monika Rummler (TU Berlin) in ihrem Einführungsvortrag u.a. die Besonderheiten in den didaktischen Anforderungen und Zielstellungen des Übungsbetriebs in den MINT-Fächern. Diese ergeben sich zu einem großen Teil aus der technischen Tiefe der zu vermittelnden Inhalte, die bis hin zu algorithmischem Verständnis und Kompetenzentwicklung beim Einsatz von Kalkülen reichen. Dabei steht die Entwicklung von Fähigkeiten zum exakten Sprachgebrauch in einem umfassenden Verständnis, nicht nur der Sprache der Mathematik, im Vordergrund.

Das von Johannes Waldmann entwickelte A*utotool*[1] unterstützt diese spezifischen Anforderungen in der MINT-Ausbildung durch interaktiven Zugang zu einem E-Learning-Werkzeug, welches die zu lehrenden Kalküle „beherrscht“. Damit ist es insbesondere möglich, in Teilen Übungsblätter abzulösen, wo sich Fortschrittskontrollen nur durch personalintensive Korrekturen realisieren lassen, deren Ergebnisse den Studierenden in der Regel erst nach Wochen-

1http://www.imn.htwk-leipzig.de/~waldmann/autotool/

frist vorliegen und wofür auch zunehmend entsprechend qualifiziertes Lehrpersonal nicht mehr zur Verfügung steht. Studierende heben in ihren Beurteilungen des Autotool-Einsatzes vor allem den hohen Lerneffekt hervor, der sich aus der Möglichkeit ergibt, unmittelbar weitere Lösungsversuche einzureichen, in denen die Hinweise des Werkzeugs zu früheren Lösungsversuchen berücksichtigt sind.

In einem vom AK E-Learning geförderten Projekt[2] wurde 2009/10 eine Autotool-Einbindung für den Opal-Vorgänger OLAT 6 geschaffen, die seither an der Universität Leipzig an aktuelle OpenOlat-Versionen angepasst wurde und im OO-Portal[3] der Fakultät für Mathematik und Informatik regelmäßig lehrveranstaltungsbegleitend in der Ausbildung zur Theoretischen Informatik im Einsatz ist. Über die Ergebnisse von fünf Jahren Einsatz dieser Autotool-Anbindung im Betrieb des OO-Portals unserer Fakultät wird hier berichtet.

Im Rahmen dieser Entwicklungen wurde auch ein XML-basiertes Austauschformat für Autotool-Aufgaben entwickelt. Auf dieser Basis werden Aufgaben im Kontext der Autotool-Anwender interessierten Lehrenden als *Open Educational Resources* (OER) über ein git-Repo zur Verfügung gestellt. Erste Arbeiten an einem entsprechenden OER-Katalog auf RDF-Basis sind begonnen worden.

2 Das OO-Portal der Fakultät für Mathematik und Informatik der Universität Leipzig

Im Softwaretechnik-Praktikum (SWP)[4], das seit 2003 fester Bestandteil im vierten Semester der Ausbildung im Studiengang Bachelor Informatik an der Universität Leipzig ist, sind in Teams von 6 bis 8 Studierenden komplexe IT-Projekte im Umfang von etwa 800 Personenstunden von der Anforderungsanalyse über Planung und Entwurf bis hin zur Implementierung umzusetzen. Die Themen ergeben sich zum großen Teil aus Forschungsthemen und Drittmittelprojekten unserer Abteilung, womit wir den Ansatz des *Forschenden Lehrens* bereits in der Bachelorausbildung curricular fest verankert haben.

Nach dem Prinzip „eat your own dog food“[5] wurden im SWP auch immer wieder Aufgaben gestellt, deren Thematik sich an Aspekten der unmittelbaren studentischen Arbeitsbedingungen orientierte. Nach einer Eigenentwicklung „UebManager“ in den Jahren 2003–2006, einer E-Learning-Lösung zur Begleitung des Übungsbetriebs, entschieden wir uns 2006 mit Blick auf entsprechende Weichenstellungen auf Landesebene, diesen Themenstarng auf begleitende Entwicklungen für das LMS OLAT zu fokussieren.

2http://bis.informatik.uni-leipzig.de/OLAT/autolat
3https://olat.informatik.uni-leipzig.de
4http://bis.informatik.uni-leipzig.de/de/Lehre/Graebe/SWTPraktikum
5https://en.wikipedia.org/wiki/Eating_your_own_dog_food

Seit dem Herbst 2007 betreibt die Fakultät dazu eine OLAT-Produktivinstanz als eigene E-Learning-Plattform, deren Wartung und Weiterentwicklung eng in die studentische Ausbildung eingebunden ist. So wurde im Herbst 2009 nach entsprechenden Vorarbeiten die studentische Authentifizierung auf Studserv-Zugangsdaten umgestellt und zum Sommersemester 2012 eine neue Version des Portals auf der Basis von OpenOlat 8 in Betrieb genommen. Seit dem Herbst 2012 steht auch die Autotool-Einbindung für OpenOlat im OO-Portal (wieder) zur Verfügung. Aktuell sind am Portal über 1100 Nutzer registriert.

3. Autotool – Konzeptionelle Grundlagen

Viele Theoriebereiche von Naturwissenschaft und Technik zeichnen sich dadurch aus, dass diese nicht nur zu qualitativen Voraussagen über realweltliche Prozesse in der Lage sind, sondern oft umfangreiche Kalküle entwickelt haben, um genauere quantitative Beschreibungen dieser Prozesse zu ermöglichen. Die studentische Unterweisung in diesen Kalkülen sowohl in deren konzeptioneller als auch kalkulatorischer Dimension ist ein wesentlicher und betreuungsintensiver Bestandteil der meisten MINT-Curricula. Mathematik als die *lingua franca* der Wissenschaft spielt dabei eine besondere und übergreifende Rolle – einmal in der Frage der Beherrschung elementarer mathematischer Konzepte und Kalküle selbst und zum anderen in den notationalen Gewohnheiten und der exakten deduktiven Methode, die in den Technik- und Naturwissenschaften in verschiedenem Umfang Grundlage eigener Kalküle sind.

Im Computerzeitalter haben sich hierbei in der studentischen Ausbildung einige Gewichte verschoben – die eigenen kalkulatorisch-algorithmischen Fertigkeiten rücken zugunsten genauerer konzeptioneller Durchdringung und exakterer Ausdrucksfähigkeit etwas in den Hintergrund, was vor allem den Übungsbetrieb zur Festigung entsprechender Fertigkeiten in der Analyse und Aufbereitung von Problemen als *Methodenkompetenz* und die Bewertung und Umsetzung von Ergebnissen als *Interpretationskompetenz* prägt.

Ein solcher von „trial and error“ geprägter Lernprozess kann durch einen erfahrenen Tutor sehr beschleunigt werden, wenn dieser die entsprechenden Lösungsversuche beobachtet und bei Fehlern und Irrwegen steuernd eingreift. Das *Autotool* setzt genau an dieser Stelle an, denn das Finden von Lösungswegen und das Aufdecken von Fehlern sind zwei auch komplexitätstheoretisch sehr unterschiedliche Fragestellungen. Während ersteres oft Heuristiken und ein genaues konzeptionelles Verständnis der Kalküle erfordert, kann zweiteres in vielen Fällen auf eine einfache Probe reduziert werden. Ist ein Lösungsweg oder ein Teil davon erst einmal gefunden und in einer exakten, dem Computer verständlichen Notation fixiert, so ist es oft einfach, die Lösung auf ihre Richtigkeit oder Plausibilität zu prüfen.

Das *Autotool* greift hierzu auf ein Compute-Backend zu, in dem die zur Lösungsverifikation erforderlichen Kalküle implementiert sind, und übergibt diesem die studentische Lösung zur Prüfung. Im Zuge dieser Überprüfung kann – je nach Implementierung im Backend – das *Autotool* sehr genau, umfangreich und geduldig auf die Lösung eingehen und von syntaktisch-notationalen Prüfungen über die Prüfung von Teil- und Zwischenergebnisse bis zur Prüfung des Endergebnisses umfangreiches Feedback generieren. Ein wesentlicher Vorteil für Studierende gegenüber der Abgabe von Übungsserien ergibt sich daraus, dass dieses Feedback nach jedem Lösungsversuch unmittelbar zur Verfügung steht und korrigierte Lösungen eingereicht werden können, wenn dies das entsprechende Lernerszenario vorsieht.

Das Konzept soll am Beispiel einer Aufgabe auf dem Kurs „Automaten und Sprachen“ genauer erläutert werden: Mit der Aufgabe soll geübt werden, einen endlichen determinstischen Automaten zu entwerfen, der eine vorgegebene Sprache akzeptiert. Die vom Übungsleiter eingestellte Aufgabe lautet dazu

```
Gesucht ist ein endlicher Automat, der die Sprache
  { ve | v in {e,f}* }
akzeptiert und folgende Eigenschaften hat:
  [ Alphabet (mkSet "ef"), Deterministic , Sane , Complete]
```

Bereits hier wird deutlich, dass Verstehen und Anschreiben der Sprache des Kalküls ein wesentliches Lernziel ist. Die Lösung ist in einer ähnlich formalisierten Weise in ein Textfeld einzugeben, wobei den Studierenden ein syntaktisch korrektes Muster vorgegeben wird, in diesem Fall ein endlicher Automat (NFA) über dem Alphabet {e,f} (mkset “ef“), Mengen von Zuständen, Startzuständen und Endzuständen sowie eine Übergangsfunktion, angegeben als Funktionstabelle

```
NFA
{ alphabet = mkSet "ef" , states = mkSet [ 1 , 2 , 3 ],
  starts = mkSet [ 2 ] , finals = mkSet [ 2 ],
  trans = collect [ ( 1 , 'e' , 2 ) , ( 2 , 'e' , 1 ),
    ( 2 , 'e' , 3 ) , ( 2 , 'f' , 3 ) , ( 3 , 'f' , 2 ) ]
}
```

Schickt man diese Eingabe unmittelbar als Lösung ans Backend, so erhält man prompt eine umfangreiche Analyse der Einsendung zurück:

1. Die Lösung wird zur Kontrolle zurückgeschickt.
2. Zur Veranschaulichung wird die Übergangsfunktion als Transitionsgraph gerendert.
3. Danach wird geprüft, ob die Eingabe syntaktisch und semantisch korrekt ist:

```
Ist die Menge
    Startzustände = mkSet [ 2 ]
Teilmenge der Menge
    Zustandsmenge = mkSet [ 1 , 2 , 3 ]?
Ja.
Ist die Menge
    akzeptierende Zustände = mkSet [ 2 ]
Teilmenge der Menge
    Zustandsmenge = mkSet [ 1 , 2 , 3 ]?
Ja.
Ist die Menge
    Alphabet des Automaten = mkSet "ef"
Teilmenge der Menge
    mkSet "ef" = mkSet "ef"?
Ja.
Der Automat soll deterministisch sein.
Diese Regeln sind nicht deterministisch:
    [ ( ( 2 , 'e' ) , mkSet [ 1 , 3 ] ) ]
```

In der Tat ist der vorgeschlagene Automat nicht deterministisch, denn im Zustand 2 gibt es zwei Produktionsregeln für die Eingabe 'e'. Diese Analyseergebnisse fallen beim Parsen der Eingabe ab und müssen nur an den entsprechenden Stellen im Compute-Prozess des Backends abgegriffen werden.

Bisher haben wir uns noch nicht mit dem Inhalt der Aufgabe befasst, was wir nun nachholen wollen. Der gesuchte Automat soll genau die Worte über dem Alphabet {e,f} akzeptieren, die auf 'e' enden. Dazu sollten zwei Zustände ausreichen, ein nicht finaler, in den wir schalten, wenn ein 'f' gelesen wurde, und ein finaler, wenn ein 'e' gelesen wurde. Versuchen wir es also mit der folgenden Modifikation des obigen Lösungsvorschlags:

```
NFA
{ alphabet = mkSet "ef" , states = mkSet [ 1 , 2  ],
  starts = mkSet [ 2 ] , finals = mkSet [ 2 ],
  trans = collect [ ( 1 , 'e' , 2 ) , ( 2 , 'e' , 2 ),
    ( 1 , 'f' , 1 ) , ( 2 , 'f' , 1 ) ] }
```

Die umfangreiche Prüfung der Eingabe auf syntaktische und semantische Korrektheit verläuft nun erfolgreich und das *Autotool* geht zur inhaltlichen Prüfung über:

```
Ist Sprache der Aufgabenstellung gleich Sprache Ihres Auto-
maten ?
    Ist Sprache der Aufgabenstellung Teilmenge der
    Sprache Ihres Automaten ?
Ja.
    Ist Sprache Ihres Automaten Teilmenge der
    Sprache der Aufgabenstellung ?
Nein. Wenigstens diese Wörter sind in Sprache Ihres Automa-
ten, aber nicht in Sprache der Aufgabenstellung:
    [ "" ]
```

```
Bewertung der Einsendung: No
```

In der Tat war unser Ergebnis fast richtig, denn auf allen Eingaben außer dem leeren Wort arbeitet unser Automat korrekt. Deshalb noch ein dritter Versuch:

```
NFA
{ alphabet = mkSet "ef" , states = mkSet [ 1 , 2  ],
  starts = mkSet [ 1 ] , finals = mkSet [ 2 ],
  trans = collect [ ( 1 , 'e' , 2 ) , ( 2 , 'e' , 2 ),
    ( 1 , 'f' , 1 ) , ( 2 , 'f' , 1 ) ] }
```

Nun verläuft auch die letzte Prüfung positiv und die Erfolgsmeldung wird an die Autotool-Einbettung weitergeleitet

```
    Ist Sprache Ihres Automaten Teilmenge der
    Sprache der Aufgabenstellung ?
Ja.
Bewertung der Einsendung: Okay
                              { punkte = 1 , size_ = 2 }
```

An dieser Stelle soll auf ein weiteres Konzept des Autotools hingewiesen werden, das von den Studierenden gern angenommen wird. Oft gibt es zu einer Aufgabe mehrere Lösungen, die sich in ihrer Effizienz und Eleganz unterscheiden. Dies kann über einen Parameter 'size' ausgewertet (in unserem Beispiel misst dieser Parameter, mit wie vielen Zuständen der Automat auskommt) und in einer „Highscore Liste“ zusammengetragen werden. Auf diese Weise lassen sich Studierende motivieren, später noch vertieft über Aufgaben nachzudenken und ihre eigenen Ergebnisse zu verbessern, auch wenn sich dies nicht mehr in der Punktbewertung der Übungsserie niederschlägt.

Studierende heben in ihren Beurteilungen des *Autotools* vor allem den hohen Lerneffekt hervor, der sich aus der Möglichkeit ergibt, unmittelbar weitere Lösungsversuche einzureichen, in denen diese Hinweise des Werkzeugs berücksichtigt werden können.

4. Autolat – Die OpenOlat-Einbindung des Autotool

Wie bereits in der Einleitung erwähnt, wurde 2009/10 in einem vom AK E-Learning geförderten gemeinschaftlichen Projekt[6] der Uni Leipzig und der HTWK Leipzig eine Autotool-Einbindung für den Opal-Vorgänger OLAT 6 geschaffen, die für eine gewisse Zeit auch im BPS-Opal verfügbar war. Parallel wurde die Lösung in unser lokales OO-Portal integriert und ist seit dem Sommersemester 2010 im produktiven Einsatz.

6http://bis.informatik.uni-leipzig.de/OLAT/autolat

Eine solche Autotool-Einbindung hat den Vorteil, die E-Learning-Strukturen eines ausgebauten Portals wie Authentifizierung, Nutzerverwaltung, Kurse, Übungsgruppen, Lerner-Szenarien usw. nutzen zu können. Auch für das *Autotool* lassen sich in der Einbindung verschiedene Lerner-Szenarien einstellen, etwa kann die Zahl der Versuche an einem Beispiel begrenzt werden oder es werden aus einer Aufgabenklasse studierendenspezifische Aufgaben generiert. Auf diese Weise lassen sich die geringen Ressourcen, die für die Weiterentwicklung des *Autotool* zur Verfügung stehen, auf dessen Kernkompetenzen konzentrieren.

Seither ist viel Wasser die Pleiße hinuntergeflossen. Die OLAT-Entwicklergemeinde hat sich in einem Fork Ende 2011 in drei Teile geteilt, die seither getrennte Wege gehen. Die Universität Leipzig setzte auf eine eigene universitätsweite E-Learning-Lösung auf Moodle-Basis und war aus dem BPS-Finanzierungsverbund ausgeschieden, womit auch unsere Anträge zur Weiterentwicklung des Autotools keine Berücksichtigung mehr fanden. Mit Blick auf die ausbildungsrelevanten Ziele, die an unserer Fakultät mit dem OLAT-Einsatz verfolgt werden, und auf die Verfügbarkeit von Quellcode und Expertise war für uns die Entscheidung naheliegend, sich dem OpenOlat-Zweig anzuschließen und das fakultätseigene Portal auf diese neue Basis zu portieren.

Im Zuge dieser Anpassungen wurde auch die Autolat-Einbindung neu gefasst und enger mit verschiedenen OpenOlat-Features wie etwa dem Kursexport oder dem Bewertungswerkzeug verknüpft. Für ersteres wurde insbesondere noch einmal die datenmäßige Einbindung der Autotool-Knoten innerhalb von OpenOlat geändert und das XML-Exportformat für Autotool-Aufgaben überarbeitet, so dass sich nun im Rahmen des allgemeinen Kursexports auch die Autolat-Bestandteile mit exportieren und ebenfalls wieder importieren lassen, was das Nachnutzbarkeitspotenzial unserer Kurse in der OpenOlat-Welt noch einmal verbessert.

5. Autolat-Einsatz an der Universität Leipzig

Es folgt eine Übersicht der Kurse, in denen Autotool-Aufgaben im Rahmen des Übungsbetriebs über die Autolat-Einbindung im OO-Portal der Fakultät angeboten und eingesetzt wurden. In Klammern sind die jeweiligen Übungsleiter genannt, ohne deren Expertise und Engagement der Autotool-Einsatz nicht in dieser Intensität möglich gewesen wäre.

- Kurs "Berechenbarkeit", Prof. Brewka, Sommersemester 2010 (Frank Löbe)
- Kurs "Berechenbarkeit", Prof. Droste, Sommersemester 2011 (Doreen Heusel, Thomas Weidner)

- Kurs "Diskrete Strukturen", Prof. Brewka, Wintersemester 2011/12 (Frank Löbe)
- Kurs "Automaten und Sprachen", Prof. Lohrey, Wintersemester 2011/12 (Doreen Heusel)
- Kurs "Automaten und Sprachen", Prof. Brewka, Wintersemester 2012/13 (Frank Löbe)
- Kurs "Berechenbarkeit", Prof. Droste, Sommersemester 2013 (Frank Löbe, Ringo Baumann)
- Kurs "Automaten und Sprachen", Prof. Brewka, Wintersemester 2013/14 (Frank Löbe, Ringo Baumann)
- Kurs "Constraint Programming", Prof. Waldmann, Sommersemester 2015

Die Kurse "Berechenbarkeit", "Diskrete Strukturen" sowie “Automaten und Sprachen" sind Teil des Pflichtprogramms im Bachelorstudiengang Informatik mit üblicherweise 80 bis 100 Belegfällen.

Exemplarisch sei der Einsatz des Autotools im Kurs "Automaten und Sprachen" im Wintersemester 2013/14 beschrieben. Die Übungen fanden im 14-täglichen Rhythmus in der A- und B-Woche statt. Zu jeder der sechs Übungen gab es in klassischer Weise ein *Seminarblatt* zur Vorbereitung sowie ein *Hausaufgabenblatt*, dessen Lösungen abzugeben waren und als Prüfungsvorleistung gewertet wurden. Daneben gab es eine Seminarserie mit drei Autotoolaufgaben sowie sechs Hausaufgabenserien mit ebenfalls jeweils drei Autotoolaufgaben zu Themen wie Deterministische endliche Automaten verschiedener Komplexität, Reguläre Ausdrücke, Kellerautomaten und Beziehungen zwischen diesen. Details dazu sind in den Metainformationen des weiter unten beschriebenen OER-Portals zu finden.

6. Autotool-Aufgaben als OER

Das Konzept der OER – Open Educational Resources[7] – gewinnt zunehmend an Aufmerksamkeit und Bedeutung. Der Begriff wurde erstmals vom *UNESCO 2002 Forum on the Impact of Open Courseware for Higher Education in Developing Countries* verwendet. „Open“ bezeichnet dabei in verschiedenen Zusammensetzungen wie Open Source, Open Design, Open Hardware, Open Innovation, Open Knowledge usw. einen neuen kulturellen Modus des Umgangs mit Wissensschätzen – *Open Culture*. Freie Bildungsmaterialien

7https://de.wikipedia.org/wiki/Open_Educational_Resources

(OER) sind nach der Definition[8] der im „Bündnis Freie Bildung“[9] zusammengeschlossenen bundesweiten Akteure in diesen kulturellen Kontext einzuordnen.

Mit der Definition eines einheitlichen XML-Exportformats für Autotool-Aufgaben im Kontext der Arbeiten an der Autolat-Einbindung steht auch dem Austausch von Aufgabenmaterial zum Autotool-Einsatz und damit einer offenen Zugänglichkeit der verwendeten Materialien nichts mehr im Wege. Entsprechendes Übungsmaterial aus den oben aufgelisteten Kursen wird bereits seit längerer Zeit in einem git-Repo als Austauschplattform zusammengetragen. Zur Orientierung in diesem Aufgabenpool wurden aus den Export-Dateien Metainformationen im RDF Format, einem Semantic Web Standard, extrahiert und mit weiteren Informationen über die Kurse und Einsatzbedingungen angereichert. Diese Informationen können in unserem experimentell eingerichteten RDF Store unter

http://pcai003.informatik.uni-leipzig.de/kosemnet

durchsucht werden.

8Der Weg zur Stärkung freier Bildungsmaterialien. Positionspapier des „Bündnis Freie Bildung“, Februar 2015, http://buendnis-freie-bildung.de/positionspapier-oer/
9http://buendnis-freie-bildung.de/