informatica didactica 1 (2000) - http://www.informatica-didactica.de

Visualisierung und Animation der semantischen Analyse von Programmen

Andreas Kerren

Universität des Saarlandes, FR 6.2 Informatik
Postfach 15 11 50, D-66041 Saarbrücken
E-Mail: kerren@cs.uni-sb.de
WWW: http://www.cs.uni-sb.de/~kerren

Zusammenfassung Im Gebiet der Informatik sind lernunterstützende Methoden sehr sinnvoll, da häufig mit abstrakten Modellen umgegangen wird, die nur unbefriedigend konventionell zu vermitteln sind. Wir haben eine Lernsoftware unter Microsoft Windows entwickelt, die es den Lernenden ermöglicht, sich besser mit den Prinzipien des Übersetzerbaus, hier speziell der semantischen Analyse, auseinanderzusetzen. Dazu bietet die Software einerseits eine interaktive Einführung in die Problematik der semantischen Analyse, in der die wichtigsten Definitionen und Algorithmen in graphisch ansprechender Form präsentiert werden. Andererseits hat der Lernende die Möglichkeit, selbst Programmbeispiele und Spezifikationen einzugeben, an denen das vorher erlernte Wissen in Form von dynamisch erzeugten Animationen und Visualisierungen vertieft und das korrekte Verständnis gesichert werden kann. Unser exploratives Lernsystem kann sowohl zur Unterstützung des Lehrers im Unterricht, als auch vom Lernenden allein verwendet werden. In diesem Papier beschreiben wir unsere Zielsetzung, unsere Ansprüche an eine Lösung, das entwickelte System und geben Designprinzipien für Lernsoftware an, die wir bei der Entwicklung des Systems und dessen Präsentation gesammelt haben.

Abstract In computer science methods to aid learning are very important, because abstract models are used frequently. For this conventional teaching methods do not suffice. We have developed an educational software, that helps the learner to better understand principles of compiler construction, in particular the semantical analysis of the source program. The software offers on the one hand an interactive introduction to the problems of the subtasks of the semantical analysis, in which the most important definitions and algorithms are presented in graphically appealing form. On the other hand the learner has the possibility to enter examples and to test and reinforce his/her new knowledge graphically on dynamically generated animations and visualizations. Examples can be input programs, expressions or specifications. Our educational software can be used as learning help in teaching as well as for self-instruction. We discuss design principles used throughout the design of the software, give a brief description of the implementation, show some examples and discuss related work.

1 Einführung

Ein Informatikdozent/-lehrer sieht sich vor die Aufgabe gestellt, den Zuhörern abstraktes Wissen zu vermitteln und vor allem das richtige und nachhaltige Verständnis dieses Wissens zu fördern. Nehmen wir an, daß ein Dozent die Funktionsweise eines Kellerautomaten erläutern will. Bestenfalls steht ihm dazu eine große Tafel und genügend farbige Kreide zur Verfügung. Nun steht er vor der Herausforderung, den Ablauf des Automaten anhand einer kleinen Beispieleingabe, einer endlichen Zahl von Zuständen und eines Kellerbildes zu erklären. Spätestens nach drei oder vier Schritten beginnt er Zustände oben auf dem Kellerbild wegzuwischen, neue einzutragen usw. Der Zuhörer wird mehr Energie aufzuwenden haben, den komplizierten Ablauf des Wischens und Neuschreibens nachzuvollziehen und in dem Durcheinander einen Sinn zu entdecken, als die Funktionsweise eines Kellerautomaten verstehen zu lernen. Damit ist die Vorführung eines solchen Automaten auch nur sehr schwer reproduzierbar. Als eine mögliche Lösung für dieses Dilemma bietet sich die Visualisierung und graphische Aufarbeitung des Kellerautomaten mit Hilfe eines Computers an. Wegen der dynamischen Abarbeitung sind Animationen bei derartigen technischen Problemstellungen das

Mittel der Wahl. Mit Hilfe von Computeranimationen lassen sich technische Prozesse anschaulich und intuitiv erklären. Weiterhin gestatten sie es den Lernenden selbst aktiv zu werden, indem diese die Animationen steuern und u.a. in einer für sie angenehmen Geschwindigkeit ablaufen lassen können.

Es ist wichtig Informationen so aufzubereiten, daß die kognitive und die affektive Informationsverarbeitung des Menschen angesprochen werden. Erstere verarbeitet eins nach dem anderen, denkt logisch und entscheidet nach Regeln und Gesetzmäßigkeiten. Die zweitgenannte Art denkt in Bildern, nutzt Analogien, sprengt Regeln, reagiert spontan und kreativ. Zeigt man zu einem abstrakten Begriff ein geeignetes Bild, so nutzt man beide „Informationskanäle“ und ermöglicht die Verbindung des eigentlichen Begriffs mit einer bildhaften Vorstellung. Man spricht in diesem Fall von Integrationslernen (siehe dazu [Alt93] und [Wei97]). Wer es versteht Informationen gut zu visualisieren, kann mit diesem Vorgehen die Merkfähigkeit und das Verständnis bei den Lernenden steigern.

2 Übersetzerbau als Domäne

Wir wählten als Wissenskontext und Beispiel für ein technisches Lehrgebiet den Übersetzerbau, speziell die semantische Analyse. Eine erste Grobstrukturierung des (konzeptionellen) Übersetzungsprozesses von Programmen ist die Unterteilung in eine Analysephase und eine Synthesephase, siehe hierzu Abbildung 1. In der Analysephase werden die syntaktische Struktur und ein Teil der semantischen Eigenschaften berechnet. Die von einem Übersetzer berechenbaren semantischen Eigenschaften nennt man die *statische Semantik*. Sie umfaßt alle semantische Information, die man lediglich aufgrund des vorliegenden Programms, d.h. ohne die Ausführung mit Eingabedaten herausfinden kann. Ein Beispiel dafür ist der Typ eines Bezeichners in streng getypten Programmiersprachen, wie etwa *Java*. Die Analysephase hat als Resultat entweder Meldungen über im Eingabeprogramm vorhandene syntaktische bzw. semantische Fehler oder eine geeignete Darstellung der syntaktischen Struktur und der statischen Eigenschaften. Im Allgemeinen handelt es sich dabei um einen mit der semantischen Information dekorierten abstrakten Syntaxbaum. Im Idealfall ist diese Phase von den Eigenschaften der Zielsprache und der Zielmaschine unabhängig. Die Synthesephase eines Übersetzers nimmt nun diese Programmdarstellung und konvertiert sie in evtl. mehreren Schritten in ein äquivalentes Zielprogramm (siehe [WM95]).

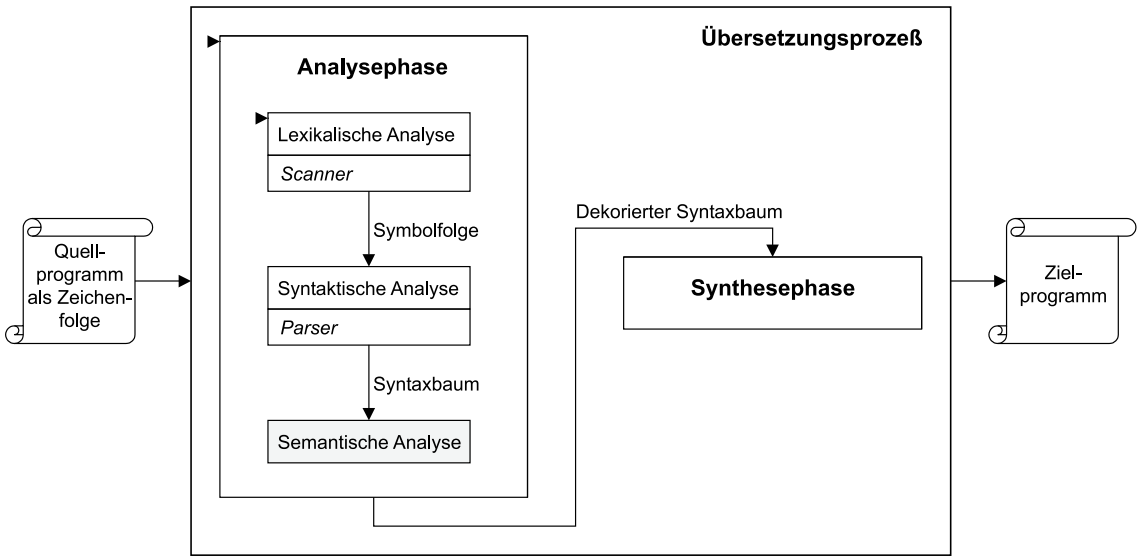

Abbildung1. Konzeptionelle Übersetzungsstruktur mit Angabe der Programmzwischendarstellungen.

Speziell zu den Aufgaben der semantischen Analyse gehören neben den Gültigkeits- und Sichtbarkeitsregeln, die Überprüfung der Kontextbedingungen (Deklarations-Analysator und Typkonsistenz-Analysator), die Überladung von Bezeichnern und schließlich die Berechnung eines allgemeinen Typs von Ausdrücken einer fest vorgegebenen polymorph getypten Programmiersprache. Dabei heißt eine Programmiersprache parametrisch polymorph, wenn es möglich ist, eine Definition einer Funktion anzugeben, die für eine Menge von Kombinationen von Operanden- und Ergebnistypen im wesentlichen dasselbe tut. Ein sogenannter Typinferenzalgorithmus berechnet hierbei für jede polymorph getypte Funktion ihren allgemeinsten Typ. Solch ein allgemeinster Typ besteht i.a. aus Typvariablen und Operatoren. Wir verwenden hier als Beispielsprache die funktionale Programmiersprache LaMa, die in [WM95] definiert wurde. Um den Typ zu berechnen, benutzt der Typinferenzalgorithmus für die komplexeren zusammengesetzten LaMa-Ausdrücke (z.B. if_then_else, ...) bestimmte Typkombinationsregeln.

In der Lehre bieten die o.g. Konzepte eine Vielzahl von Problemfeldern, die sich mit Hilfe von Visualisierungen und Animationen gut lösen lassen:

- Schwierige Notationen, die häufig zurück ins Gedächtnis gerufen werden müssen.
- Die semantische Analyse operiert auf abstrakten Syntaxbäumen, also baumartigen Strukturen, die inhärent schon eine graphische Intuition besitzen.
- Komplizierte Algorithmen, deren Pseudocode als reine Textform nur schwer nachzuvollziehen ist.
- Diverse Datenstrukturen, wie verkettete Listen, Symboltabellen, ...

Abschnitt 6 enthält Beispiele, wie die verschiedenen Aufgaben der semantischen Analyse mit Hilfe unseres Lernsystems erläutert und animiert werden.

3 Systemdesign

Unser System präsentiert und erläutert zunächst schrittweise die Definitionen der semantischen Analyse. Diese werden anschließend anhand von animierten Beispielen verdeutlicht. Beides geschieht vollständig interaktiv, d.h. die Benutzer können sich per Mausklick durch eine graphische Umgebung bewegen, Themen, die sie interessieren auswählen, vertiefen und Beispielanimationen steuern. Schließlich besteht die Möglichkeit, Beispiele selbst anzugeben und die vorgestellten Algorithmen dynamisch auf den abstrakten Syntaxbäumen der eingegebenen Beispiele animiert ablaufen zu lassen (vgl. Abs. 6). Beispiele können Eingabeprogramme, Ausdrücke oder Spezifikationen sein. Sämtliche vom Benutzer einzugebenden Probleme sind in unserem Fall entscheidbar und vom System korrekt lösbar.

Das System wurde mit dem Autorensystem Multimedia ToolBook™ 3.0 („MTB“) der Firma Asymetrix™ unter Windows 3.1 (16 Bit) entwickelt. Der dynamische Teil des Systems ist jedoch aufgrund der besseren Performanz und einigen Beschränkungen des ToolBook-Systems in C unter Verwendung der Windows-API implementiert. Eine solche Beschränkung ist z.B. die nicht akzeptable, limitierte Seitengröße für größere Bäume. Die Lernsoftware ist unter Windows 3.1/95/98/NT4 lauffähig und bedarf einer Runtime-Umgebung, die allerdings frei verfügbar ist.

Die Einordnung dieser Arbeit liegt weder in der reinen Visualisierung von komplexen Datenstrukturen noch in der reinen Animation von Algorithmen. Unsere Intention lag in der Schaffung eines multimedialen Lernsystems für den Übersetzerbau, mit dessen Hilfe sich der Anwender selbständig oder im Rahmen des Unterrichts in die Thematik einarbeiten und sein erlerntes Wissen durch die Eingabe eigener Beispielprogramme bzw. -spezifikationen (Eingabedaten für die hier betrachteten Algorithmen) festigen kann. Es werden nicht nur Algorithmen visualisiert, was auch der Anspruch der Ausgaben der gängigen Algorithmenanimationssysteme (siehe [BS84], [Bro87], [BN96], [Sta96], ...) ist. Das System abstrahiert graphisch auch die Definitionen in der Theorie, Beispiele, der Sinn von Sätzen, etc. Selbst Notationen werden sofort durch Anklicken anhand von animierten oder statischen Beispielen verdeutlicht, ohne in einem Buch hin- und herblättern zu müssen. Hat man zum Beispiel während des Ablaufs einer Algorithmenanimation einen wichtigen Aspekt des Algorithmus oder eine wichtige Definition vergessen, z.B. was ein definierendes Vorkommen eines Bezeichners ist, dann kann man sich sofort die entsprechende Definition bzw.

den Algorithmus zunächst als Textanimation anzeigen lassen oder zu der entsprechenden Lektion springen und danach dort wieder einsteigen, wo man vorher aufgehört hat.

Durch die bereits oben erwähnte Eingabe eigener Beispiele läßt sich unser System nicht nur zur reinen Informationspräsentation nutzen, sondern im Hinblick auf die Systematik der intelligenten Lehr- und Lernsysteme unter Systeme zum entdeckenden (explorativen) Lernen eingruppieren. Übungsaufgaben mit adaptiver Hilfestellung und korrekter Lösung bietet unsere Lernsoftware nicht an.

4 Verwandte Arbeiten

Im Kontext des Übersetzerbaus wurden an der Universität des Saarlandes auch andere Visualisierungen entwickelt, darunter etwa Visualisierungen abstrakter Maschinen bestimmter imperativer, logischer und funktionaler Programmiersprachen ([Koh95], [Wir95] und [Ste92]). Diese Visualisierungen wurden unter X-Windows (UNIX) implementiert. Sie zeigen die Effekte der Ausführung von Maschineninstruktionen auf Laufzeitkeller und Heap, wobei sie jedoch kaum Animationen enthalten. Weiterhin wurde ein Tool für die Visualisierung von Graphen aus dem Übersetzerbau, genannt VCG („Visualization of Compiler Graphs“), entwickelt [San95]. Es existieren VCG-Versionen für mehrere Computersysteme, auch für das MS Windows System.

Ein anderes an der Universität des Saarlandes entwickeltes Lernsystem ist die „Animation der lexikalischen Analyse“ [BDKW99]. Dieses Lernprogramm bietet einerseits eine interaktive Einführung in die Problematik der lexikalischen Analyse, in der die wichtigsten Definitionen und Algorithmen präsentiert werden. Andererseits zeigen Animationen, wie endliche Automaten aus regulären Ausdrücken generiert werden und wie diese Automaten arbeiten. Die Eingabe eigener Beispiele ist mit diesem System allerdings nicht möglich.

Weitergehend existiert gegenwärtig eine große Anzahl von Algorithmenanimationen, aber es gibt nur wenige fundamentale Arbeiten auf diesem Gebiet. Marc H. Brown entwickelte mehrere Algorithmenanimationssysteme, wie z.B. BALSA, ZEUS, CAT, etc. Diese Systeme sind Frameworks, in denen Algorithmen durch spezielle Annotationen („interesting events“) und durch die Definition graphischer Sichten („views“) animiert werden können ([BN96], [Bro87], [Bro92], [Bro93] und [BS84]). John T. Stasko entwickelte das Pfadtransitionsparadigma und implementierte es in den Systemen TANGO, XTANGO, SAMBA, etc., siehe [Sta90a], [Sta90b] und [Sta96]. Diese Systeme verwenden ebenfalls das Konzept der „interesting events“. Alle neueren Versionen der oben genannten Systeme sind komplette Umgebungen, die Editoren für die Kreierung der Views anbieten, in denen die Algorithmen animiert werden. Das WEB-basierte Animationssystem CAT (bzw. das neuere JCAT, das in der Programmiersprache *Java* geschrieben wurde) stellt eine vollständige Entwicklungsumgebung für die Schaffung von Algorithmenanimationen im WWW dar. (J)CAT bietet weitaus mehr Möglichkeiten, als ad-hoc programmierte Java-Applets. Hiermit ist es möglich, Algorithmenanimationen zu kreieren, die ein Lehrer seinen Schülern online demonstrieren kann. Die Interaktion der Schüler kann durch den Lehrer limitiert oder erweitert werden. Sie können die Animationen steuern und andere Sichten auf den Algorithmus auswählen, aber z.B. zunächst nicht die Eingabe ändern. Daher stellt das System einen Schritt in Richtung des sogenannten „Elektronischen Klassenraums“ dar. Abschließend gibt das Papier [HD96] eine gute Übersicht über die hier vorgestellten Systeme.

5 Designprinzipien

Eine notwendige Voraussetzung für die Entwicklung einer guten Lernsoftware ist die Anwendung von guten Designrichtlinien. Derartige Richtlinien wurden vor der Implementierungsphase des hier vorgestellten Systems entwickelt und während der Implementierung permanent überarbeitet [Ker97]. Einige dieser Richtlinien resultieren aus der Forschung über die sogenannte Mensch-Maschine-Interaktion (HCI), etwa das wohlbekannte Konsistenzprinzip, das in [Shn97] folgendermaßen beschrieben wird: „This principle is the most frequently violated one, and yet is the easiest one to repair and avoid. Consistent sequences of actions should be required in similiar situations;

identical terminology should be used in prompts, menus, and help screens; and consistent commands should be employed throughout. Exceptions, such as no echoing of passwords or confirmation of the DELETE command, should be comprehensible and limited in number."

Animationen spielen sich auf der Benutzeroberfläche eines Programms ab, daher gelten hier auch grundsätzlich die Prinzipien zur Erstellung von graphischen Benutzerschnittstellen („GUIs"), z.B. das o.g. Konsistenzprinzip. Zusätzlich geben wir für Animationen in folgender Übersicht eigene Richtlinien an, welche die Dynamik von Animationen berücksichtigen:

- *Flexible Steuerung:* Animationen sollten in der Geschwindigkeit regelbar, schrittweise durchführbar, zu jedem Zeitpunkt zu stoppen und rückzusetzen sein. Ein Rückwärtslaufenlassen von Animationen ist nur bedingt sinnvoll und häufig auch technisch schwierig umzusetzen. Als Alternative ist eine Undo-Operation jedoch angebracht, die eine begrenzte Zahl von Rückwärtsschritten zu festgelegten Backtrackpunkten ermöglicht.
- *Klar definierte Objektbewegungen:* Bewegungen von Objekten sollten möglichst direkt zu ihrem Ziel, aber nicht über zu viele andere Objekte hinweg erfolgen. Es sollten nicht mehrere, logisch unzusammenhängende Objekte gleichzeitig bewegt werden und die Bewegung selbst nicht zu komplex und nicht ruckartig sein.
- *Feingranuläre Steuerbarkeit:* Wird eine Animation angehalten, so sollte dieser Vorgang sofort erfolgen und die Animation nicht erst noch eine Weile weiterlaufen.
- *Minimale Merkanforderungen an die Anwender:* Animationen und entsprechende Erklärungen sollten innerhalb eines räumlichen und logischen Rahmens ablaufen. Bei dynamischen, zur Laufzeit automatisch erzeugten Animationen kollidiert dieses Prinzip häufig mit zu hohem räumlichen Platzanspruch.
- *Highlights:* Ab und zu können gewisse Teile von Animationen etwas spektakulärer ausfallen, als es der reine Informationsgehalt der Animation erfordert. Dies dient der Steigerung der Aufmerksamkeit, Motivation und Neugier auf spätere Animationen.

Der ein oder andere Punkt kann je nach Anwendungsgebiet eine stärkere bzw. schwächere Gewichtung erhalten. Kommerzielle Präsentationen auf Messen enthalten oft viele spektakuläre Animationen, die nur die wichtigsten Informationen übermitteln sollen, wogegen Lernsoftware mehr Wert auf didaktisch sinnvolle Animationen legt.

6 Visualisierungs- und Lernbeispiele

Im folgenden Abschnitt zeigen und erläutern wir einige Auszüge des Lernprogramms. Die hier abgedruckten Bilder geben von den Visualisierungen und Animationen naturgemäß nur eine sehr unvollkommene Vorstellung. Ihre Graurasterung wurde verändert, um eine bessere Druckqualität zu erreichen. Sie spiegeln z.T. auch lediglich die Situation wider, in denen der Benutzer eine Animation einfrieren kann. Um sich eine bessere Vorstellung von der Funktionsweise und dem Animationslayout zu machen, kann der interessierte Leser die Software kostenlos aus dem Netz herunterladen [Ker98].

6.1 Statische Animationen

Zunächst stellen wir den Teil des Lernsystems vor, in dem die Animationen und Visualisierungen auf fest vorgegebenen Beispielen beruhen, die vom Lernenden nicht modifiziert werden können. Dabei stellt das System die Aufgaben der semantischen Analyse vor, erläutert die Problematik an vielen kleinen Animationen und bietet Problemlösungen anhand von Algorithmen an. Diese werden jeweils durch Algorithmenanimationen an einem festen, statischen Beispiel erklärt.

Terminologie Auf der linken Hälfte der Seite in Abb. 2 sind die in der semantischen Analyse verwendeten Begriffe kurz definiert. Klickt man auf einen blau gefärbten Textteil, so erscheint auf der rechten Hälfte eine detailliertere Erklärung des gewählten Begriffs. Innerhalb dieses Bereichs lassen sich auch ein oder mehrere Beispiele zu dem ausgewählten Begriff anzeigen.

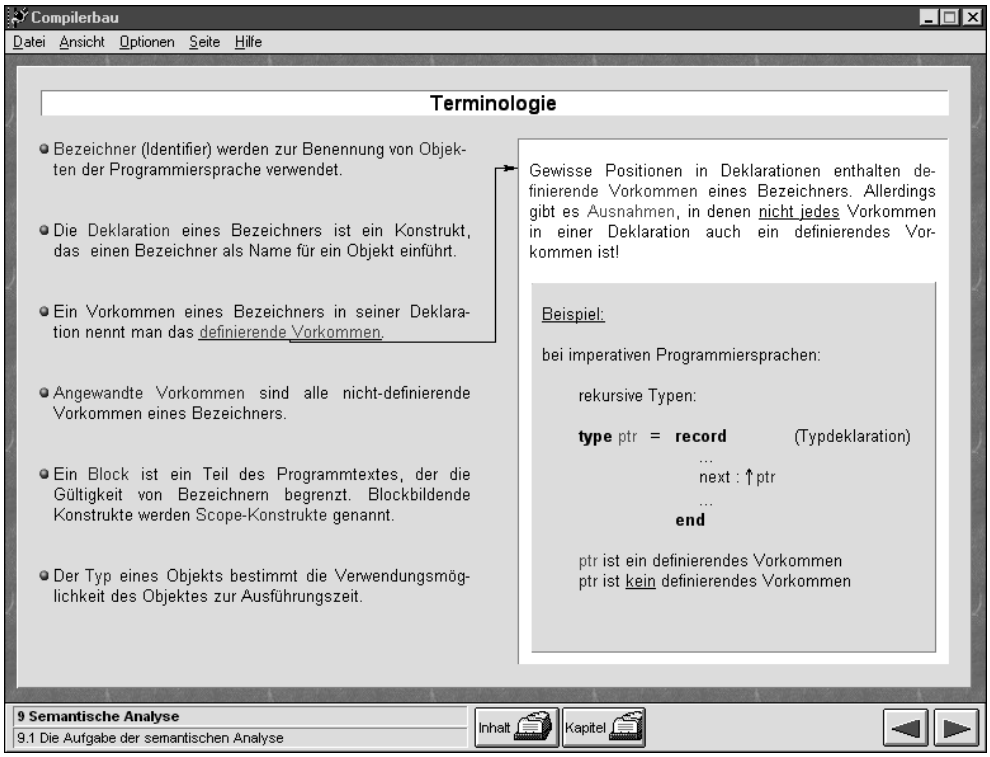

Abbildung2. Terminologie zur semantischen Analyse

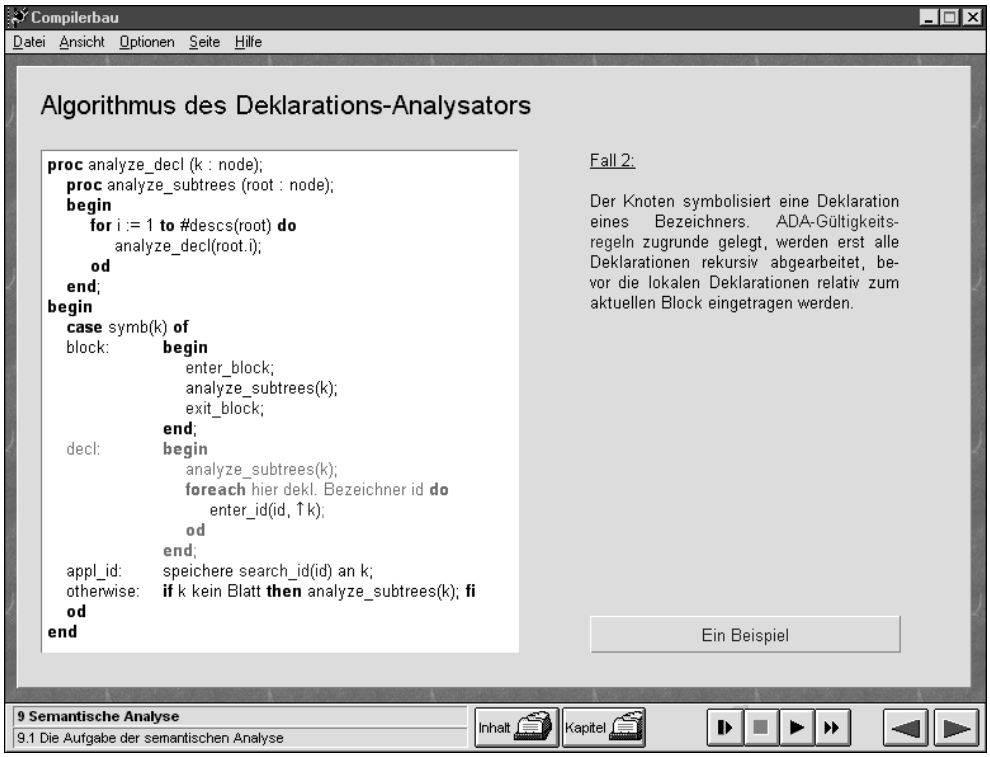

Abbildung3. Statische Algorithmenanimation eines Deklarationsanalysators

Überprüfung der Kontextbedingungen In unserem zweiten Beispiel (Abb. 3) soll die Funktionsweise eines Deklarationsanalysators zunächst an einer gedachten Übersetzungssituation erläutert werden. Der rot gefärbte Quelltext des Algorithmus kennzeichnet die aktuelle Stelle, die auf der rechten Hälfte der Seite erklärt wird. Mit den Steuerungsknöpfen unten in der Kontrolleiste kann der Algorithmus komplett durchlaufen werden (Textanimation). Zu allen wichtigen Bestandteilen des Algorithmus lassen sich Beispielanimationen aufrufen, die ihrerseits wieder aus mehreren Seiten bestehen können.

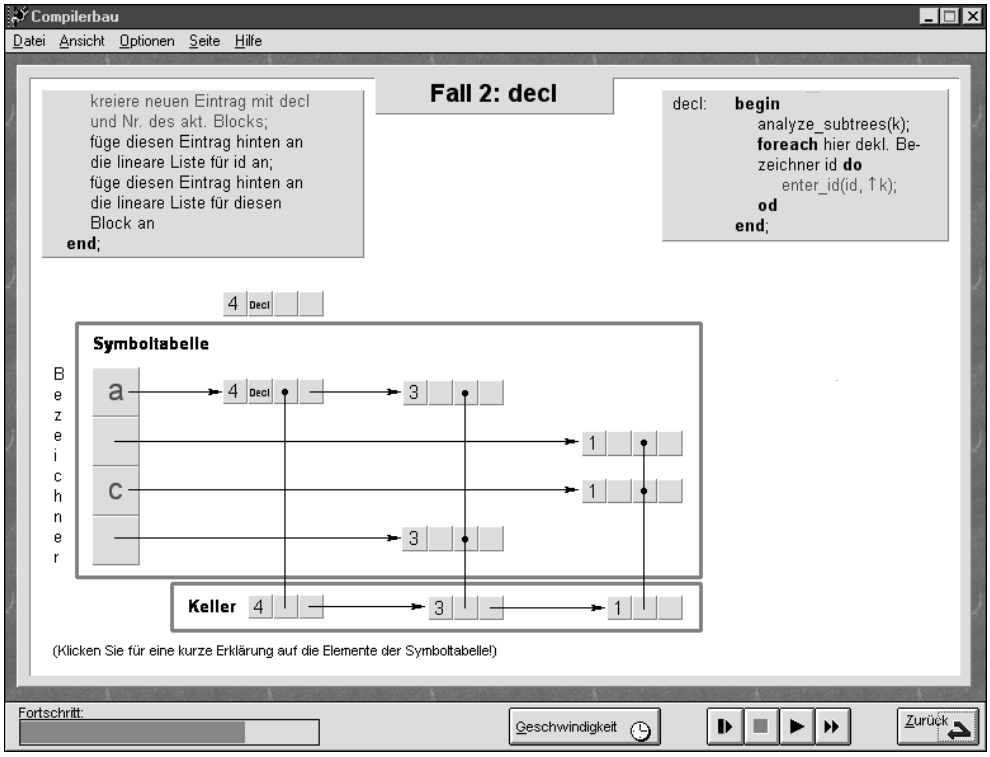

Abbildung4. Animation zu einer Symboltabellenoperation

Die nächste Abbildung 4 zeigt einen Schritt in einer Animationsabfolge, die durch Anklikken des Knopfes *Ein Beispiel* aus Abb. 3 aufgerufen wird. Rechts oben befindet sich die aktuelle Stelle im Quelltext zum Algorithmus für den Deklarations-Analysator. Man kann die Elemente der Symboltabelle anklicken, um eine kurze Erklärung zu dem ausgewählten Element zu erhalten. Hier hat der Benutzer auf die erste Zelle eines Eintrags geklickt, um sich deren Bedeutung anzeigen zu lassen. Die Symboltabelle ist als verkettete Liste dargestellt, deren Einträge aus Strukturen von je vier Elementen bestehen. Das erste Element ist die Blocknummer, das zweite Element die Adresse der entsprechenden Deklaration im Syntaxbaum, etc. Die Animation zeigt, daß bereits ein Bezeichner a im aktuellen vierten Block neu deklariert wurde. Der Bezeichner c, der schon im ersten Block ein deklarierendes Vorkommen hat, wird nun im aktuellen vierten Block erneut deklariert. Dazu wird in der Symboltabelle ein neuer Eintrag generiert und in die entsprechende Stelle der Symboltabelle eingefügt. Genau dieser Prozess ist in der oben gezeigten Animation zu sehen. In der Quelltextzeile rechts oben ist ein Funktionsaufruf $enter\_id(id, \uparrow k)$ enthalten. Diese Funktion ist nun im Feld links oben zu sehen. Der Anwender kann den Quelltext für diese Funktion zeilenweise abarbeiten lassen. Die Animationssteuerung gestattet es, die Animation mit unterschiedlichen, frei einstellbaren Geschwindigkeitsniveaus ablaufen zu lassen, anzuhalten und einzelne Schritte auszuführen. Dabei wird in einer Fortschrittsleiste (links unten) der aktuelle Stand innerhalb der Animation angezeigt.

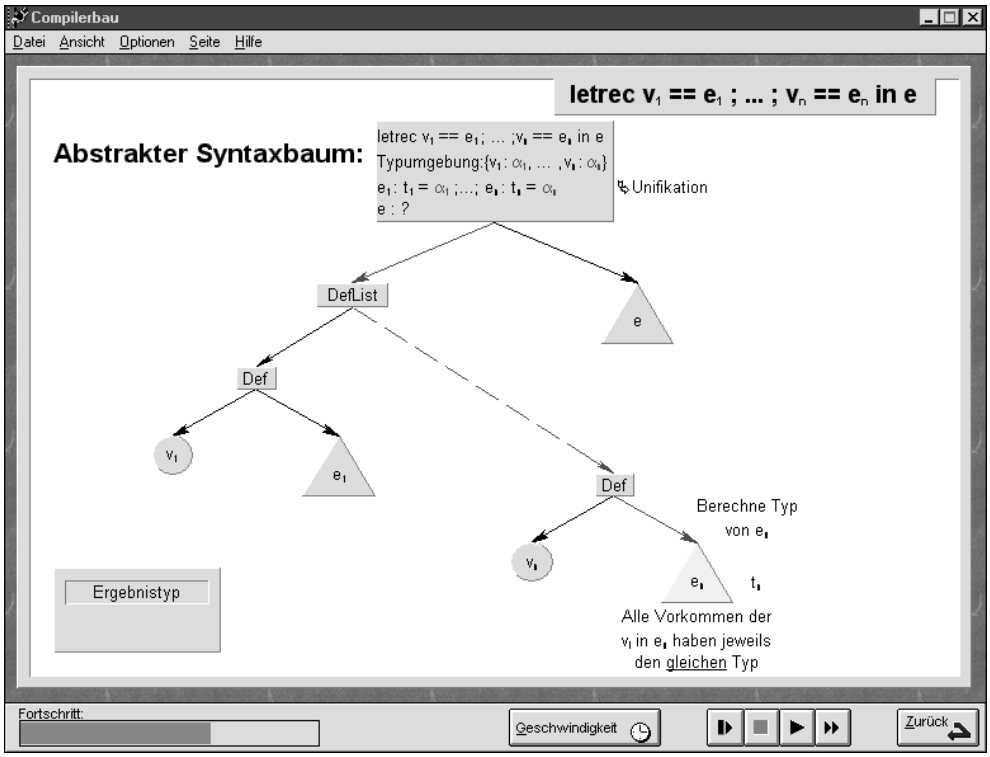

Abbildung5. Animation zur Typkombinationsregel eines *LETREC*-Ausdrucks

Typinferenz Zu dem zusammengesetzten LaMa-Ausdruck *LETREC* wird eine Animation der Typkombinationsregel an einem verallgemeinerten Beispiel angezeigt (siehe Abbildung 5). Man sieht einen der letzten Schritte der Animation. Der Typkombinationsregel entsprechend werden die einzelnen polymorphen Typen der Unterbaumwurzeln berechnet, eine Typumgebung erzeugt, neue Typvariablen eingeführt, etc. Die Animation zeigt einen Schritt in dieser Berechnung. Man sieht den Teilbaum des *LETREC*-Konstrukts, in dessen Wurzelknoten die neue Typumgebung und neue Typvariablen eingetragen werden. Dieser expandierte Wurzelknoten entspricht der intuitiven Speicherung der Typinformationen in den Knoten des realen Syntaxbaumes. Am Ende der Animation wird der neue Ergebnistyp in das Feld links unten eingetragen.

6.2 Dynamische Animationen

Nun kommen wir zu den Visualisierungen, in denen der Anwender ein beliebiges Beispielprogramm oder -spezifikation eingeben kann. Das Layout des resultierenden Syntaxbaumes wird automatisch berechnet und im Anwendungsbereich angezeigt. Der Benutzer hat Einfluß auf das Baumlayout, er kann die Abstände von Geschwisterknoten, benachbarten Knoten und Eltern/Kind-Knoten verändern. Weiterhin besteht die Möglichkeit den Baum in seiner Größe zu verändern und in vier Richtungen zu orientieren. Diese Einflußmöglichkeiten sind notwendig, um den Baum im Fenster optimal zu plazieren. Es ist z.B. möglich, das Baumlayout so zu verändern, daß der Baum komplett in das Fenster paßt. Dies erhöht die Übersichtlichkeit bei der Animation. Alle anderen zur Visualisierung gehörenden Graphikelemente, wie etwa kleine Informationsfenster an den einzelnen Knoten, zusätzliche Kanten etc., werden unmittelbar an das neue Layout angepaßt. Der Baumlayoutalgorithmus entspricht bis auf die Möglichkeit der Größenveränderung einer Arbeit von J. Walker [Wal90]. Grenzen in der Baumgröße sind nur durch den verfügbaren Speicher und ein über 16 Bit Variablen adressierbares logisches Koordinatensystem gegeben.

Überprüfung der Kontextbedingungen Der in Abb. 6 angegebene Bildschirmausschnitt zeigt eine Visualisierung der Überprüfung der Kontextbedingungen eines durch den Anwender eingege-

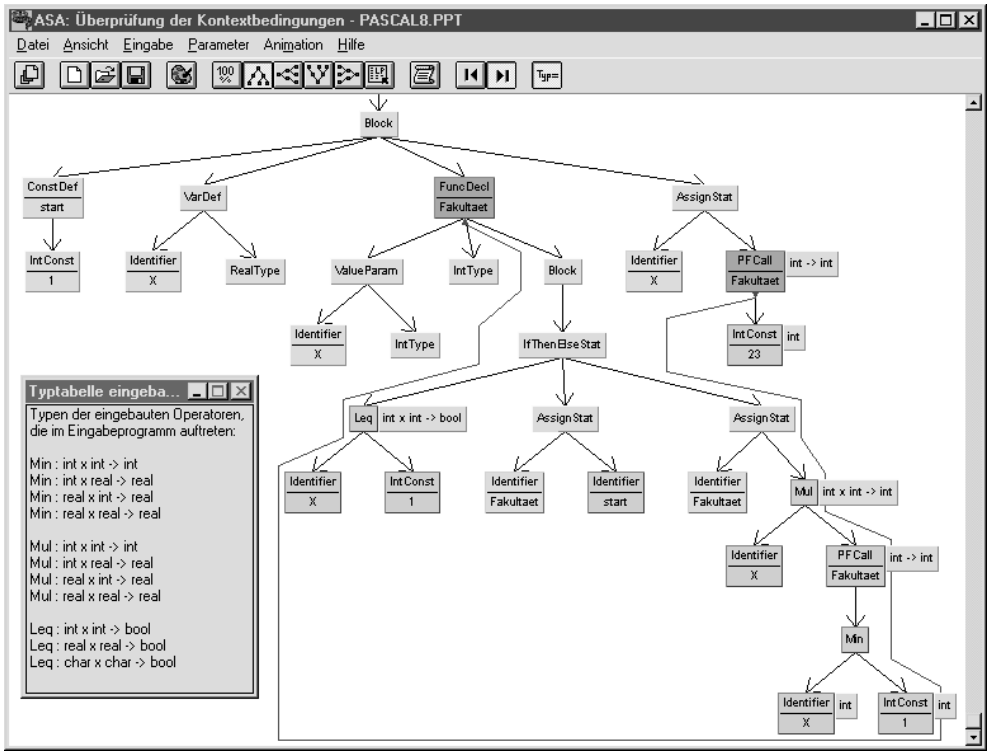

Abbildung6. Visualisierung der Überprüfung der Kontextbedingungen

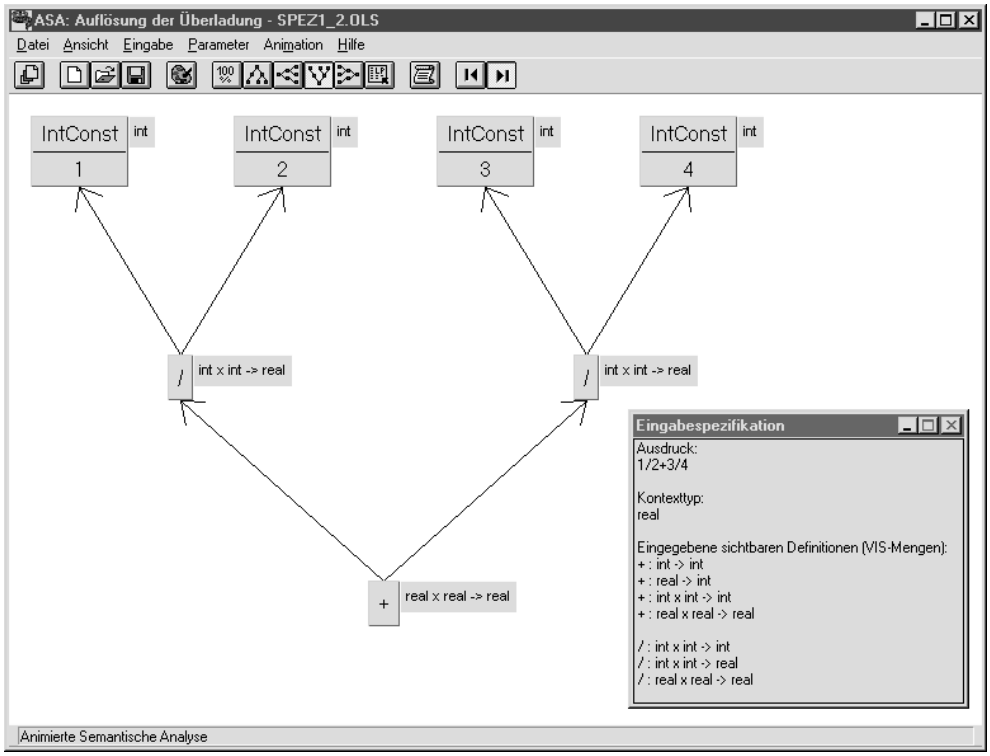

Abbildung7. Visualisierung der Auflösung der Überladung

benen Beispielprogramms (s.u.). Eine komfortable Eingabemöglichkeit dieser Programme ist der eingebaute Editor, der auch eine Funktion zum Syntaxtesten bereithält. Ist das Eingabeprogramm syntaktisch fehlerhaft, so markiert der Editor das Fehlersymptom. Syntaktische Korrektheit ist eine Voraussetzung zur semantischen Analyse und wird daher vom System getestet. Der zugehörige abstrakte Syntaxbaum ist in der Abbildung fast vollständig dargestellt. Zu einigen Syntaxbaumknoten sind die Typattribute zu sehen. Grundlage für deren Berechnung sind die in einem Hilfsfenster (links unten) angegebenen Typen für die im Beispielprogramm verwendeten und eingebauten Operatoren. Für ein angewandtes Vorkommen des Bezeichners *Fakultaet* wurde das nach den Gültigkeits- und Sichtbarkeitsregeln errechnete definierende Vorkommen nach einem Mausklick auf das angewandte Vorkommen rot markiert und der entsprechende Link mit einer roten Kante symbolisiert. Wird kein definierendes Vorkommen gefunden, dann öffnet sich ein Dialogfenster mit einer entsprechenden Fehlermeldung. Der Lernende hat nun die Möglichkeit, das Beispielprogramm abzuändern und die Analyse neu ablaufen zu lassen. Der Pascal-Code des Beispielprogramms ist in Programm 1 angegeben. Die entsprechende Pascal-Grammatik wurde in der Online-Hilfe definiert.

Programm 1 Pascal-Code des Eingabebeispiels

```
program pascal8;
  const start = 1;
  var X : real;

  function Fakultaet(X : int) : int;
    begin
    if X <= 1 then
      Fakultaet := start
    else
      Fakultaet := X * Fakultaet(X-1)
    end;

  begin
    X := Fakultaet(23)
  end.
```

Überladung von Operatoren Unser letztes Beispiel (Abb. 7) demonstriert die Auflösung der Überladung von Operatoren. Rechts unten ist ein Hilfsfenster mit der aktuellen Eingabespezifikation eingeblendet, die zuvor mit Hilfe einer Eingabemaske erstellt und syntaktisch überprüft worden war. Der „+“-Operator ist vierfach und der „/“-Operator dreifach überladen. Im Animationsbereich ist der auf 140% vergrößerte Ausdrucksbaum des Eingabeausdrucks (hier „$1/2 + 3/4$“) mit den zu jedem Operatorknoten assoziierten Mengen von Definitionsalternativen zu sehen. Alle Mengen sind nach der Berechnung einelementig, wobei die Integerkonstanten nullstellige Operatoren vom Typ **int** sind. Die Überladung der Operatoren wurde also erfolgreich aufgelöst. Im Falle eines Fehlers würde mindestens eine Menge mehrelementig und entsprechend markiert sein.

7 Evaluation

Unsere Zielgruppen sind Schüler, die in ihren Schulen einen Informatikkurs belegt haben, sowie Studenten der Informatik. Für eine Evaluation der hier vorgestellten Lernsoftware kooperieren wir mit Mitarbeitern des Lehrstuhls für kognitive Modellierung und Methodologie des Fachbereichs Psychologie an der Universität des Saarlandes und bereiteten gemeinsam ein Experiment vor, um statistisch korrekt auswertbare Daten unserer Software zu erhalten. Leider mußten wir feststellen, daß trotz der Bereitschaft der saarländischen Informatiklehrer nicht genug Schüler für das Experiment gewonnen werden konnten. Daher planten wir das Experiment mit Informatikstudenten im

zweiten Semester, um eine hinsichtlich der Statistik ausreichende Teilnehmeranzahl zu erreichen. Um dem Studienplan besser zu entsprechen, führten wir das Experiment zunächst erst mit der Lernsoftware „Animation der lexikalischen Analyse“ [BDKW99] durch. Da dieses Lernsystem zu dem hier vorgestellten System bis auf die freie Eingabe eigener Beispiele identisch gestaltet wurde, sollten die Ergebnisse zur Softwarebewertung bis zu einem gewissen Grad auch auf die „Animation der semantischen Analyse“ zutreffen. Trotz unserer Bemühungen konnten wir die Anzahl der Probanden nicht wesentlich steigern, da lediglich 15 Informatikstudenten an unserem Experiment teilnahmen. Wir benötigen mindestens 30 Studenten, um statistisch signifikante Daten zu erhalten. Daher sind alle hier vorgestellten Resultate nur vorläufig und bis zu einer Wiederholung des Experiments dementsprechend nicht endgültig.

Die Evaluation basiert auf einem Leistungstest zwischen zwei Teilnehmergruppen. Die erste Gruppe lernte ein Teilgebiet der lexikalischen Analyse mit Hilfe der Software, die zweite Gruppe das gleiche Teilgebiet mit Hilfe eines äquivalenten Lehrtextes. Danach erhielten beide Gruppen einen identischen Bogen mit 19 inhaltlichen Testfragen. Die Software-Gruppe bekam zusätzlich einen speziellen Fragebogen zur Anwendbarkeit und Bewertung der Software, etwa zum Seitenlayout oder zur Animationssteuerung. Der Test bestand aus 9 Wissensfragen, deren Beantwortung primär die Erinnerung und Reproduktion explizit genannter Sachverhalte forderte, sowie aus 10 Transferfragen, deren Beantwortung vom Probanden ein tieferes Verständnis des Lerninhaltes abverlangten.

Der Leistungsmittelwert der Computergruppe war sowohl für die Wissens- als auch für die Transferfragen höher als der Leistungsmittelwert der Textgruppe. Ein um die Standardabweichung korrigierte Größe ist die Effektgröße d. Die berechnete Effektgröße ist für die Wissensfragen $d \approx 0.74$ und $d \approx 0.70$ für die Transferfragen. Nach gängiger Konvention in den Sozialwissenschaften wird ein Effekt *mittel* genannt, wenn $0.5 \leq d < 0.8$ und *stark*, wenn $d \geq 0.8$. Gemessen an diesen Richtwerten ist die festgestellte Leistungssteigerung durch die Lernsoftware durchaus beachtlich. Da aber nur 15 Probanden an der Evaluation teilgenommen haben, läßt sich dieses Ergebnis noch nicht statistisch absichern.

8 Abschlußbemerkungen

Wir haben die Lernsoftware „Animation der semantischen Analyse“ vorgestellt, die eine umfassende Lernumgebung für die Konzepte einer Übersetzungsphase (semantische Analyse) bereitstellt. Sie bietet durch die Angabe von benutzerdefinierten Beispielprogrammen und Spezifikationen eine dynamisch erzeugte Visualisierung, mit deren Hilfe der Lerner sein erlerntes Wissen explorativ bestätigen und überprüfen kann. Wir stellten einige Prinzipien für die Erstellung von Lernsoftware und Animationen vor, die wir während der Entwicklungsphase des Systems entdeckten.

Mehrere Vorführungen vor Schülern, Lehrern und Studenten im Rahmen von Schülertagen und Präsentationsdemos haben großes Interesse und Neugier an diesem Projekt geweckt. In nachfolgenden Diskussionen wurde uns auch der Vorteil von Animationen in diesem Bereich bestätigt. Aus technischer Sicht läßt sich die Verwendung des Autorensystems MTB 3.0 kritisieren. Es unterliegt zu großen Einschränkungen und das Laufzeitsystem nimmt viel Speicherplatz in Anspruch. Aus diesen Gründen müssen bei Verwendung von Autorensystemen meist wichtige Teile der Software in einer anderen Sprache implementiert werden. Der Vorteil des Systems liegt in der Einfachheit der Bedienung.

Der generative Ansatz wird im laufenden Projekt GANIMAL[1] weiterverfolgt. Ziel des Projektes ist die Erstellung einer explorativen Lernsoftware für den Übersetzerbau, in der für jede Übersetzungsphase die Implementierung **und** die entsprechende Visualierung bzw. Animation aus Spezifikationen automatisch generiert werden. Die hier vorgestellte Lernsoftware wird dabei besonders in bezug auf den explorativen Aspekt als Erfahrungsgrundlage für das GANIMAL-Projekt dienen (siehe auch [DK00,DKP97]). Nicht nur auf dem technischen Gebiet der Informatik sind die gesammelten Erfahrungen einzusetzen. Sie lassen sich auch auf andere Themengebiete übertragen, in denen Prozesse visualisiert werden sollen, etwa der Medizin, Elektrotechnik, usw. Weitere Infor-

[1] Das Projekt wird von der Deutschen Forschungsgemeinschaft DFG gefördert.

mationen über den aktuellen Entwicklungsstand, sowie die neuesten Versionen der Software inkl. Runtime-Modul findet der Leser im WWW [Ker98].

Danksagung Dieser Artikel ist größtenteils eine Zusammenfassung meiner Diplomarbeit „Animation der semantischen Analyse“ [Ker97], die ich an der Universität des Saarlandes angefertigt habe. Mein Dank gilt daher Prof. Dr. Reinhard Wilhelm für die Vergabe dieser Arbeit, sowie seiner Unterstützung. Bedanken möchte ich mich auch bei Beatrix Braune und Dr. Stephan Diehl für die vielen und umfangreichen Diskussionen, die mir bei der Konzeptions- und Implementierungsphase des Systems neue Ideen und Sichtweisen eröffneten.

Literatur

[Alt93] A. Alteneder. *Visualize with the Help of Computers: Computergraphics and Computer Animations*. Siemens, VCH (in German), 1993.

[BDKW99] B. Braune, S. Diehl, A. Kerren, R. Wilhelm. *Animation of the Generation and Computation of Finite Automata for Learning Software*. To appear in Proceedings of Workshop of Implementing Automata WIA'99, Potsdam, Germany, July, 1999.

[BN96] M. H. Brown, M. A. Najork. *Collaborative Active Textbooks: A Web-based Algorithm Animation System for an Electronic Classroom*. SRC Research Report 142, DEC, 1996.

[Bro87] M. H. Brown. *Algorithm Animation*. MIT Press, 1987.

[Bro92] M. H. Brown. *Zeus: A System for Algorithm Animation and Multi-View Editing*. SRC Research Report 75, DEC, 1992.

[Bro93] M. H. Brown. *The 1992 SRC Algorithm Animation Festival*. In IEEE Symp. on Visual Languages, pp. 116-123, 1993.

[BS84] M. H. Brown, R. Sedgewick. *A System for Algorithm Animation*. In SIGGRAPH 84, Computer Graphics 18(3), pp. 177-186, 1984.

[DK00] S. Diehl, A. Kerren. *Increasing Explorativity by Generation*. To appear in Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications, EDMEDIA-2000, AACE, Montreal, Canada, 2000.

[DKP97] S. Diehl, T. Kunze, A. Placzek. *GANIMAM: Generierung interaktiver Animationen von abstrakten Maschinen*. In Proceedings of „Smalltalk und Java in Industrie und Ausbildung STJA'97“, pp. 185-190, Erfurt (Germany), 1997.

[HD96] A. Hausner, D. P. Dobkin. *Making Geometry Visible: an Introduction to the Animation of Geometric Algorithms*. Computer Science Department, Princeton University, 1996.

[Ker97] A. Kerren. *Animation of the Semantical Analysis*. Master's Thesis (in German), Universität des Saarlandes, 1997.

[Ker98] „http://www.cs.uni-sb.de/RW/anim/animcomp.html“ und „http://www.cs.uni-sb.de/∼kerren“

[Ker99] A. Kerren. *Animation of the Semantical Analysis*. In Proceedings of „8. GI-Fachtagung Informatik und Schule INFOS99“ (in German), pp. 108-120, Informatik aktuell, Springer, 1999.

[Koh95] G. Kohlmann. *Visualization of the abstract P-Machine*. Master's Thesis (in German), Universität des Saarlandes, 1995.

[San95] G. Sander. *Visualization Techniques for Compiler Construction*. Dissertation (in German), Universität des Saarlandes, 1995.

[Shn97] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 3nd Edition, Addison-Wesley, 1997.

[Sta90a] J. T. Stasko. *A Framework and System for Algorithm Animation*. Computer, 18(2), pp. 258-264, 1990.

[Sta90b] J. T. Stasko. *The Path-Transition Paradigm: A Practical Methodology for Adding Animation to Program Interfaces*. Journal of Visual Languages and Computing (1), pp. 213-236, 1990.

[Sta96] J. T. Stasko. *Using Student-Built Algorithm Animations as Learning Aids*. Technical Report GIT-GVU-96-19, Georgia Institute of Technology, Atlanta, 1996.

[Ste92] B. Steiner. *Visualization of the abstract Machine MaMa*. Master's Thesis (in German), Universität des Saarlandes, 1992.

[Wal90] J. Walker. *A node-Positioning Algorithm for General Trees*. In Software-Practice and Experience 20(7), pp. 685-705, 1990.

[Wei97] B. Weidenmann. *Multicodierung und Multimodalität im Lernprozeß*. In Issing und Klimsa (Hrsg.), Information und Lernen mit Multimedia, 2. Auflage, pp. 65-84, Beltz, 1997.

[Wir95] S. Wirtz. *Visualization of the abstract Machine WiM*. Master's Thesis (in German), Universität des Saarlandes, 1995.

[WM95] R. Wilhelm, D. Maurer. *Compiler Design: Theory, Construction, Generation*. Addison-Wesley, 1995.