Automatische Überprüfung von Übungsaufgaben

Norman Hendrich, Klaus von der Heide

Fachbereich Informatik, AB Technische Aspekte Multimodaler Systeme
Universität Hamburg, Vogt-Kölln-Str. 30, D-22527 Hamburg
{hendrich,heide}@informatik.uni-hamburg.de

Abstract: Dieser Beitrag beschreibt ein Konzept zur automatischen Überprüfung von Übungsaufgaben. Gezielte Hilfestellungen unterstützen die Studierenden während der Bearbeitung der Aufgaben, so dass viele Fehler vermieden und die spätere Korrektur erleichtert wird. Ausgehend von einer Analyse und Klassifikation der typischen Aufgabentypen in der technischen Informatik zeigt sich, dass Überprüfung und Hilfestellung für viele Übungsaufgaben möglich ist. Die zugehörige Software ist mit Java-Applets realisiert und kann auch für andere Fachgebiete und Veranstaltungen genutzt sowie in die gängigen e-Learning Plattformen integriert werden.

1 Motivation

Übungsaufgaben bilden seit jeher eine zentrale Komponente der Ausbildung und sind wegen der intensiven Beschäftigung mit dem Lehrstoff besonders zur Vertiefung geeignet. Nicht zuletzt wegen der hohen Teilnehmerzahlen im Grundstudium ist der Übungsbetrieb aber oft ineffizient und sowohl für die Lernenden als auch die Lehrenden unbefriedigend. Einerseits fehlt in der entscheidenden Phase, nämlich beim Lösen der Aufgaben, eine Hilfestellung für die Studierenden, und andererseits kann später bei der Besprechung der Lösungen wegen der großen Gruppen kaum noch auf individuelle Fragen eingegangen werden. Auch das Korrigieren der Übungsaufgaben ist nicht nur eine aufwendige sondern sogar bedrückende Aufgabe, denn häufig wird klar, dass eine viel früher angesetzte Korrektur das Lernen gefördert und typische Fehler vermieden hätte.

Durch den Einsatz von e-Learning-Software zur Überprüfung der Übungsaufgaben können wir unseren Studierenden bereits unmittelbar während der Bearbeitung der Übungsaufgaben einen Feedback über den Lernfortschritt geben und die Fehlersuche durch gezielte Hilfestellungen erleichtern. Damit werden nicht nur Flüchtigkeitsfehler vermieden, sondern die Lernenden werden rechtzeitig auf falsche Lösungswege aufmerksam und können ihre verfügbare Zeit wesentlich effizienter nutzen [Sch97].

Die im folgenden vorgestellten Ansätze und Algorithmen wurden als Ergänzung der Übungen zu unserer Grundstudiumsvorlesung über Technische Informatik entwickelt. Sie decken jedoch eine Vielzahl von Aufgabentypen ab und sollten sich daher auch für andere Fachgebiete und Veranstaltungen einsetzen lassen. Abschnitt 2 präsentiert zunächst eine

Analyse und Klassifikation der in der technischen Informatik vorkommenden Aufgabentypen. Daraus werden in Abschnitt 3 diverse Ansätze zur Überprüfung und Hilfestellung für die verschiedenen Aufgaben abgeleitet und zusammengefasst. Die von uns gewählte Softwarearchitektur basiert auf Java-Applets und wird in Abschnitt 4 beschrieben.

1.1 Use-Cases

Für die automatische Überprüfung von Übungsaufgaben sind zunächst drei Anwendungsszenarien vorgesehen:

- Die *Bearbeitung der Übungsaufgaben* durch die Studierenden, entweder einzeln oder in kleinen Gruppen gemeinsam vor einem Rechner. Die Software ermöglicht die sofortige Überprüfung der Lösungen und liefert Hilfestellungen mit Tips zur Fehlersuche. Sowohl der online-Zugriff als auch die offline-Nutzung ohne Internetverbindung sind vorgesehen.
- Die *Korrektur* der von den Studierenden eingesandten Lösungen durch die Übungsgruppenleiter. Gerade bei komplexen Aufgabenstellungen ist durch die automatische Überprüfung eine beträchtliche Zeitersparnis möglich; als Beispiel mag Abbildung 4 mit einem Boole'schen Ausdruck von sechs Variablen dienen.
- *Vorführung und Diskussion* der Übungsaufgaben während der Präsenzübungen. Alternative Lösungsansätze und Vorschläge können sofort interaktiv erprobt werden.

2 Eine Klassifikation von Übungsaufgaben

Ausgangspunkt für die Softwareentwicklung war eine umfangreiche Analyse der tatsächlich in Übungsaufgaben (zur technischen Informatik) vorkommenden Aufgabentypen. Dabei wurden diverse Lehrbücher und Vorlesungen berücksichtigt, für die sowohl Übungsaufgaben als auch die zugehörigen Musterlösungen verfügbar waren. Insgesamt lassen sich zwanglos sechs Kategorien von Aufgaben unterscheiden, die in Abbildung 1 als Mindmap dargestellt sind:

- Die Klasse der *Auswahlaufgaben* umfasst Fragestellungen, bei denen die Lösung durch Auswahl aus vorgegebenen Alternativen erfolgen kann, darunter die beliebten Ja/Nein- und Multiple-Choice- Aufgaben, aber auch Zuordnungsaufgaben oder Image-Maps. In allen Fällen ist die automatische Überprüfung durch trivialen Vergleich mit einer Musterlösung möglich; aus diesem Grund wurden auch die Lückentexte mit in diese Kategorie aufgenommen.
- Die nächste Klasse enthält alle Aufgaben, deren Antwort aus *Zahlenwerten* besteht. Neben den üblichen Integer- und Gleitkommazahlen sind im Rahmen der technischen Informatik auch diverse Varianten zu berücksichtigen, insbesondere die Binär- oder Hexadezimalformate und Komplementdarstellungen. Die Überprüfung erfolgt durch Wertevergleich unter Berücksichtigung der jeweiligen Nebenbedingungen.

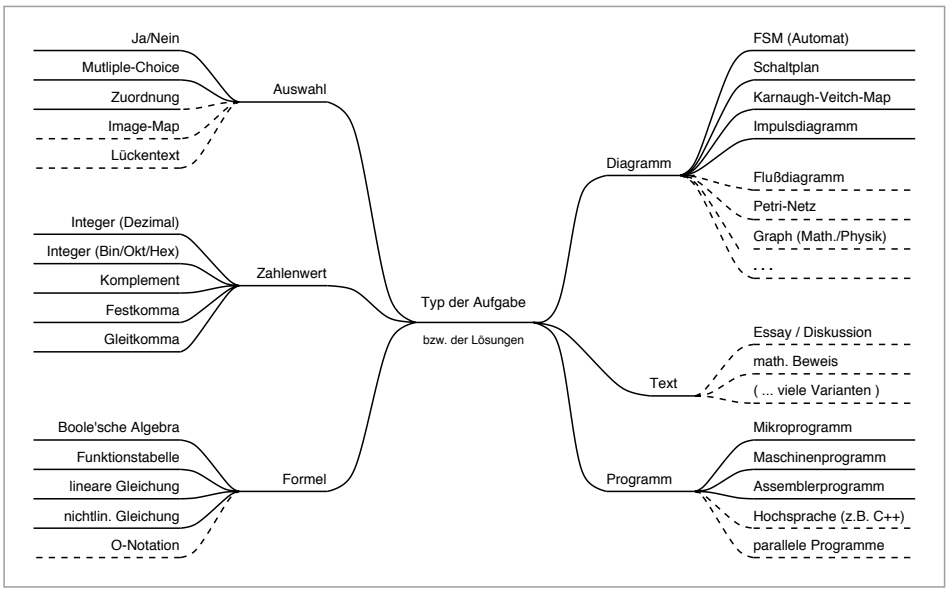

Abbildung 1: Klassifikation der Übungsaufgaben zur Technischen Informatik als Mindmap. Alle mit durchgezogenen Linien dargestellten Aufgabentypen werden von unserer Software unterstützt.

- Die Kategorie der *Formeln* umfasst lineare und nichtlineare Gleichungen, Gleichungssyteme, logische Ausdrücke und die Boole'sche Algebra. Wegen der ähnlichen Strategie zur Auswertung fallen auch Funktions- und Wertetabellen in diese Klasse.
- Die Klasse der *Diagramme* umfasst eine Vielfalt von verschiedenen Untergruppen. Im Rahmen der technischen Informatik sind besonders die Schaltpläne, aber auch Zustandsdiagramme von Automaten und Flussdiagramme von Algorithmen wichtig. Für andere Anwendungsgebiete wären hier weitere Diagrammtypen zu ergänzen, beispielsweise Strukturformeln in der Chemie.
- Die Klasse der *Text*-Aufgaben bereitet für die automatische Überprüfung die größten Schwierigkeiten. Trotz beträchtlicher Fortschritte durch Einsatz von Methoden der Sprachverarbeitung und statistischen Verfahren [SB03] sowie ersten kommerziellen Systemen zur automatischen Auswertung von Essays [ETS04], dürfte in den meisten Fällen eine manuelle Korrektur durch die Übungsgruppenleiter erforderlich sein. Textaufgaben werden von unserer Software bisher nicht unterstützt.
- In die letzte Klasse der *Programme* fallen alle Aufgaben, bei denen die Softwareentwicklung im Mittelpunkt steht. Im Kontext der technischen Informatik interessieren hier zunächst die unteren Abstraktionsebenen bis zur Assemblerprogrammierung. In vielen Fällen kann die automatische Überprüfung durch Übersetzen und Ausführen der Programme erfolgen. Für höhere Abstraktionsebenen sind wesentlich leistungsfähigere Ansätze und Analysen möglich [BKW03].

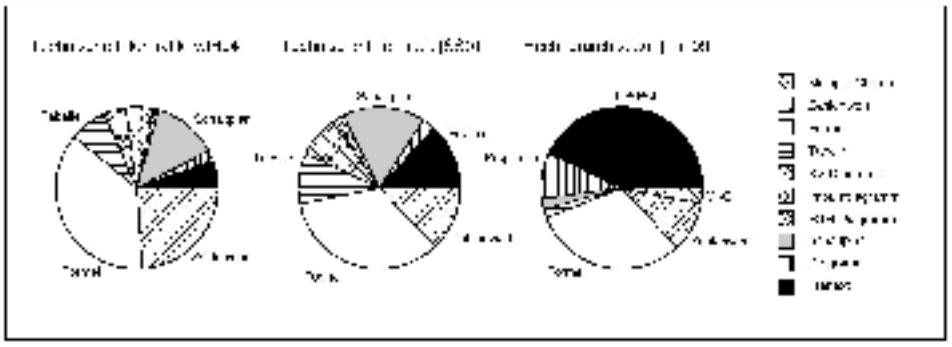

Abbildung 2: Drei typische Beispiele für die Häufigkeiten der verschiedenen Aufgabentypen aus [vdH04, SS01, Tan99]. Berücksichtigt wurde die Anzahl der Aufgaben ohne Gewichtung der Komplexität.

Die relative Häufigkeit der verschiedenen Aufgabentypen hängt stark von der jeweiligen Quelle und den dort gesetzten Schwerpunkten ab. Die drei typischen Beispiele in Abbildung 2 entstammen zwei Einführungen in die technische Informatik [vdH04, SS01] und einer Einführung in die Rechnerarchitektur [Tan99]. Eingetragen ist jeweils die Anzahl der Aufgaben bzw. Teilaufgaben, jedoch ohne Gewichtung nach Komplexität der Aufgabenstellung.

Trotz beträchtlicher Unterschiede im Detail lassen sich mehrere Gemeinsamkeiten erkennen, insbesondere ein sehr hoher Anteil der Zahlenwert- und Formelaufgaben. Die sonst so beliebten Multiple-Choice und Zuordnungs-Aufgaben fehlen dagegen fast völlig. Die Gewichtung der Aufgaben zum Logik- und Schaltungsentwurf bzw. zur Programmierung ist natürlich durch die unterschiedliche Thematik bedingt. Auffällig ist der sehr hohe Anteil der Freitext-Aufgaben („Begründen Sie…“) in [Tan99], der einen sehr hohen Aufwand bei der Korrektur erwarten lässt.

2.1 Geeignete Auswahl der Übungsaufgaben

Natürlich kann die automatische Überprüfung durch geeignete Auswahl der Aufgaben erheblich erleichtert oder überhaupt erst möglich werden. Der Schwerpunkt unserer bestehenden Aufgaben liegt im Training der Fertigkeiten zur Auswahl und dem Einsatz von *Methoden* der Problemlösung, während das Abfragen von Faktenwissen nur eine geringe Rolle spielt. Der übliche Ausweg durch verstärkten Einsatz von Multiple-Choice-Aufgaben kommt daher kaum in Betracht.

Ein Beispiel für diesen Aspekt bieten zwei auf den ersten Blick sehr ähnliche Aufgaben und direkt aufeinanderfolgende Übungen aus [vdH04]:

> **Aufgabe T1-2.10**: Finden Sie einen zyklisch-einschrittigen Code mit 12 Codewörtern. Ein solcher Code könnte z. B. für eine Winkelcodierung in 30^o-Schritten benutzt werden.

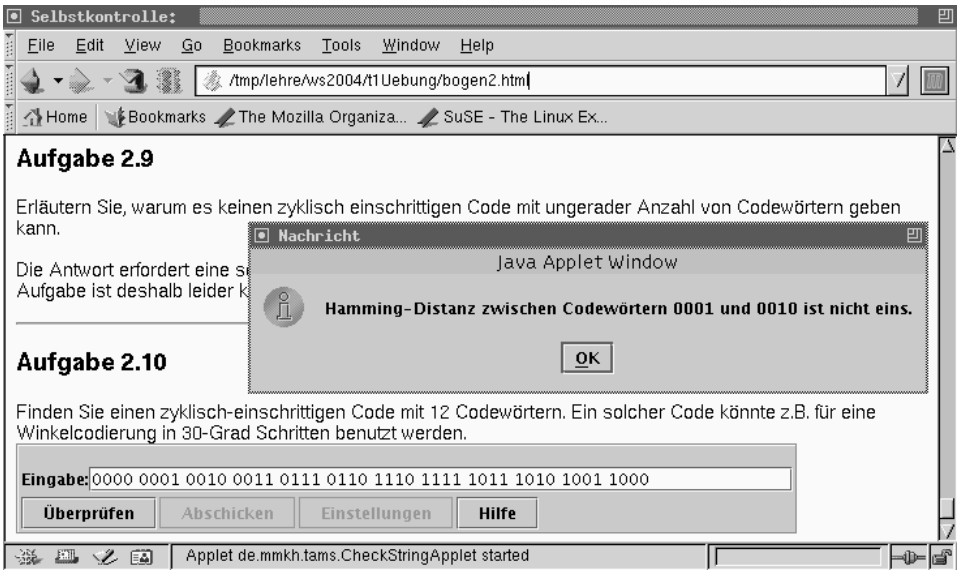

Abbildung 3: Überprüfung einschrittiger Codes mit einem Beispiel für die Hilfestellung. Das Applet überprüft die Eigenschaften des eingegebenen Codes und akzeptiert alle gültigen Lösungen (bei einstellbarer Anzahl der Codewörter). Wegen der Vielzahl der möglichen und gleichwertigen Lösungen ist ein Vergleich mit einer Musterlösung nicht praktikabel.

Eine korrekte Lösung zu dieser Aufgabe besteht einfach aus einer Liste mit den gesuchten Codewörtern. Allerdings existiert eine Vielzahl von gleichwertigen Lösungen, so dass ein Vergleich der eingegebenen Codewörter mit einer Musterlösung kaum in Frage kommt. Trotzdem ist die automatische Überprüfung für die Aufgabe problemlos realisierbar, indem zunächst die Anzahl und Länge der Codewörter überprüft wird und anschließend die Hamming-Distanz der benachbarten Codewörter berechnet wird. Aus verletzten Bedingungen ergeben sich sofort geeignete Hilfestellungen, siehe Abbildung 3.

Dagegen erscheint eine automatische Überprüfung für die direkt vorangestellte Aufgabe praktisch unmöglich — trotz der identischen Thematik und ähnlichen Aufgabenstellung:

> **Aufgabe T1-2.9**: Erläutern Sie, warum es keinen zyklisch-einschrittigen Code mit ungerader Zahl von Codewörtern geben kann.

In diesem Fall besteht die korrekte Antwort in einem Hinweis darauf, dass sich die Parität aufeinanderfolgender Codewörter unterscheiden muss, dies aber bei ungerader Anzahl von Codewörtern zu einem Widerspruch führt. Angesichts der Vielzahl der möglichen Formulierungen dürfte eine automatische Auswertung des Lösungstextes selbst die derzeit fortschrittlichsten Werkzeuge der Textanalyse überfordern.

3 Überprüfung und Hilfestellungen

Angesichts des großen Stoffumfangs und der verschiedenen Aufgabentypen kommt eine vollständige und aufwendige Repräsentation des Lehrstoffs, etwa in der Art eines regelbasierten KI-Systems, kaum in Frage. Statt dessen setzen wir eine Reihe von Standardtechniken ein. Neben dem Vergleich mit einer Musterlösung, eventuell nach vorheriger Vereinfachung und Umwandlung in eine Normalform, sind dies die Überprüfung von Randbedingungen und die Simulation bzw. Testläufe:

- Für *Auswahlaufgaben* (Multiple-Choice usw.) erfolgt die Überprüfung durch Vergleich mit der vorgegebenen Musterlösung. Eine Hilfestellung ist kaum möglich, da angesichts der ebenfalls vorgegebenen Lösungsvarianten kaum auf die eigentliche Fehlerursache zurückgeschlossen werden kann.
- *Zahlenwerte* werden durch Wertevergleich mit der Musterlösung überprüft. Sofern Nebenbedingungen wie Zahlenformat oder Stellenanzahl gefordert sind, können diese für sinnvolle Hilfestellungen genutzt werden („Bitte eine Oktalzahl mit genau vier Ziffern eingeben“). Optional ist auch ein Einheitenvergleich vorgesehen.
- Der erste Schritt zur Überprüfung von *Formeln* besteht in der Syntaxprüfung. Für Boole'sche Ausdrücke setzen wir derzeit auf einen Vergleich der Funktionstabelle mit der Musterlösung; dabei werden Don't-Care Terme berücksichtigt. Bei Fehlern werden Gegenbeispiele erzeugt und ausgegeben, siehe Abbildung 4.
- Für *arithmetische Ausdrücke* ist sowohl die symbolische als auch die numerische Auswertung möglich; bei Abweichungen außerhalb der geforderten Toleranz werden Gegenbeispiele generiert.
- Bei *Tabellen* erfolgt die Überprüfung entsprechend der Tabellenelemente für die einzelnen Zahlenwerte oder Formeln.
- Für *Diagramme* sind jeweils anwendungsspezifische Ansätze notwendig. Derzeit sind die folgenden Varianten implementiert:
- Die *Logikminimierung mit KV-Diagrammen* wird durch einen eigenen Editor mit einfacher Benutzeroberfläche unterstützt. Die Überprüfung erfolgt wiederum durch Vergleich der Funktionstabelle und der Realisierungskosten (Gatteranzahl) mit einer Musterlösung.
- Für die Überprüfung von *digitalen Schaltungen* auf Gatterebene setzen wir auf die Simulation mit pseudozufälligen Eingabedaten und Signaturanalyse der Ausgangswerte. Nur eine funktionsfähige Schaltung liefert die korrekten Signaturen der Musterlösung [Hen02]. Hilfestellungen ergeben sich aus statischen Plausibilitätstests (z.B. offene Gattereingänge) und zusätzlichen Vergleichen während der Simulation (z.B. undefinierte Werte, Hazards). Diese Algorithmen wurden in ein vorhandenes Java-basiertes Simulations-Framework integeriert.

- Die Überprüfung von *Programmen* auf Maschinen und (eingeschränkt) Assemblerebene erfolgt ebenfalls durch Testläufe und Vergleich der Programmausgaben mit der geforderten Musterlösung, ergänzt um zusätzliche Plausibilitätstests und einer statischen Analyse der Programmtexte. Dieses Vorgehen ist natürlich nur sinnvoll, weil die entsprechenden Programme sehr kurz und einfach sind.
- Für einige Aufgaben kommen zusätzliche aufgabenspezifische Techniken zum Einsatz (Abbildung 3).

Bei einigen Aufgabentypen sind zusätzliche Funktionen integriert, die die bisherigen Lösungsversuche berücksichtigen und die Hilfestellungen nach mehreren fehlerhaften Versuchen sukzessive ausführlicher gestalten oder Teile der Lösung verraten.

4 Implementierung

Um einen plattformübergreifenden Einsatz der Software zu ermöglichen und Probleme mit komplizierter Installation zu vermeiden, wurden die meisten der oben genannten Verfahren als Kollektion von aufeinander aufbauenden Java-Applets implementiert. Die Applets sind in die Webseite der Vorlesung bzw. Übungen eingebettet und können von dort ohne weitere Installationshürden gestartet werden. Alternativ ist aber auch der Download dieser Webseiten für die Offline-Nutzung vorgesehen.

Für die Überprüfung von arithmetischen und Boole'schen Ausdrücken kommen eigene Parser zum Einsatz, die mit Hilfe des JavaCC Compiler-Generators entwickelt wurden [JCC97]. Dies erlaubt die interaktive Eingabe und Auswertung von Ausdrücken auch in der compilierten Java-Umgebung; der direkte Zugriff auf die Parse-Bäume erleichtert darüberhinaus zusätzliche Analyseschritte, die zur Überprüfung der verschiedenen Nebenbedingungen erforderlich sind.

Alle Applets zeichnen sich durch eine einheitliche und bewusst einfach gehaltene Benutzeroberfläche aus; in vielen Fällen wird die Lösung einfach als Text in das zentrale Textfeld eingegeben und dann der *Überprüfen*-Knopf gedrückt. Dies ist notwendig, um den Aufwand zur Einarbeitung möglichst gering zu halten und die Akzeptanz der Überprüfung zu erhöhen. Die für alle Applets vorgesehene Option, die Lösungen nach der Überprüfung direkt per E-Mail an die Übungsgruppenleiter zu senden, wurde aus organisatorischen Gründen bisher noch nicht genutzt.

Die Konfiguration der Applets erfolgt durch Applet-Parameter direkt aus den HTML-Seiten. Als Beispiel zeigt Abbildung 5 den Quelltext für das darüber in Abbildung 4 dargestellte Applet. Über die Applet-Parameter ist auch eine Internationalisierung der Benutzeroberfläche (Button-Labels, Hilfetexte, etc.) möglich. Die von den Applets erzeugten Ausgaben und Hilfestellungen liegen dagegen derzeit nur auf Deutsch vor; eine Anpassung ist über den Java *ResourceBundle*-Mechanismus möglich, wegen des vergleichsweise hohen Aufwands aber bisher nur teilweise implementiert.

Um trotz der Einbettung in HTML alle möglichen (ASCII-) Zeichen in den Musterlösungen nutzen zu können, verwenden wir bei Bedarf eine modifizierte Base64-

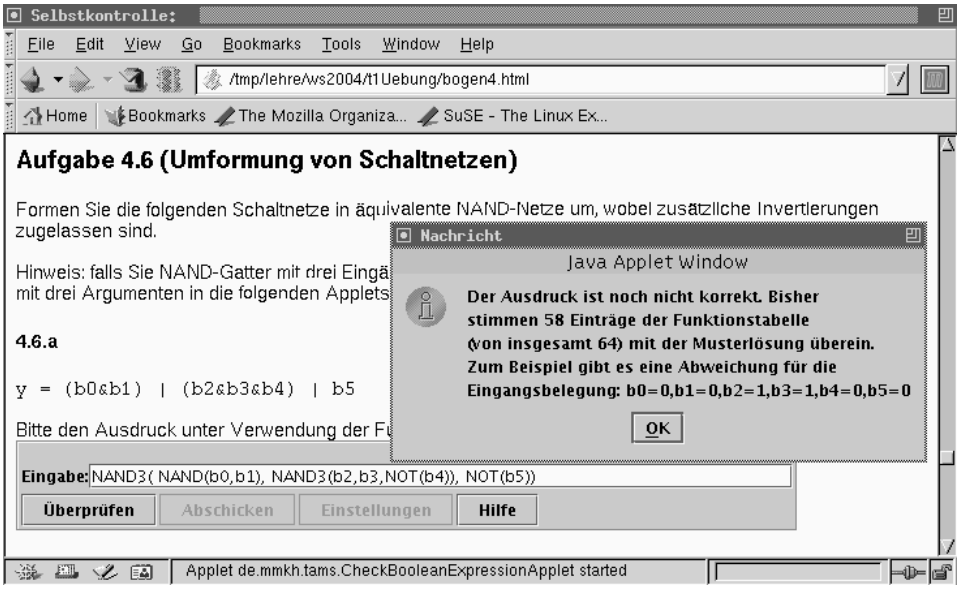

Abbildung 4: Applet zur Überprüfung von Boole'schen Ausdrücken. Neben der Infix-Darstellung wird auch die funktionale Schreibweise unterstützt. Die Hilfestellung enthält eine Bewertung der aktuellen Lösung und liefert Hinweise zur Suche nach den jeweiligen Fehlern.

```
...
<h4>4.6.a</h4>
<code>y = (b0&b1) | (b2&b3&b4) | b5</code>
<p>
Bitte den Ausdruck unter Verwendung der Funktionen NAND3(),
NAND() und NOT() eingeben:
<applet
 code=de.mmkh.tams.CheckBooleanExpressionApplet.class
 codebase="." archive="checkapplets.jar"
 width=600 height=90
>
<param name="default"  value="NOT(NAND3(b0,b1,b2))"> </param>
<param name="table"    value="0001000100010001000..."> </param>
<param name="varnames" value="b0,b1,b2,b3,b4,b5"> </param>
<param name="form"     value="NANDNOT"> </param>
</applet>
...
```

Abbildung 5: Das Codebeispiel zeigt den HTML-Quellcode für das oben dargestellte Applet. Die gesuchte Funktion wird hier via *table* als Funktionstabelle übergeben, während *varnames* die Variablennamen definiert. Eine Verschlüsselung oder Verschleierung ist nicht notwendig, da über den *form*-Parameter die gesuchte funktionale Schreibweise mit den Funktionen *NAND()* und *NOT* angefordert wird.

Kodierung für diese Parameter. Dies erschwert gleichzeitig das ansonsten für einige Studierende verlockende Ablesen der Lösungen aus den HTML-Quelltexten.

4.1 Klassenhierarchie

Derzeit sind die folgenden Java-Klassen vorhanden:

- *CheckStringApplet* dient zur Überprüfung von ein- oder mehrzeiligen Texten gegen eine vorgegebene Musterlösung. Es erfolgt keine Textanalyse mit Methoden der Sprachverarbeitung, aber verschiedene Optionen wie das Ignorieren von Leerzeichen oder Klein- und Großschreibung ermöglichen einen flexiblen Einsatz. Als Hilfestellung wird die Übereinstimmung zwischen Lösung und Musterlösung berechnet und ausgegeben.
- *CheckNumberApplet* ermöglicht die Überprüfung von Zahlenwerten in den gängigen Formaten (Integer, Festkomma, Gleitkomma) und mit diversen Optionen (Anzahl der Vor- und Nachkommastellen, Zahlenbasis, Komplementdarstellung, Vorgabe der Genauigkeit). Die Musterlösung und Optionen werden über Parameter eingestellt. Das Applet deckt praktisch alle in der technischen Informatik relevanten Aufgabentypen mit Zahlenwerten ab.
- *CheckMultipleChoiceApplet* dient als Demonstrator für die gängigen Multiple-Choice Aufgaben; die jeweiligen Einträge und zugehörige Musterlösung werden über Applet-Parameter übergeben. Dieses Applet wird in unseren Aufgaben derzeit nicht verwendet.
- *CheckBooleanExpressionApplet* erwartet die Eingabe eines Boole'schen Ausdrucks in Infix- oder funktionaler Schreibweise. Anzahl und Namen der Variablen sowie die Musterlösung und diverse Optionen werden über Applet-Parameter eingestellt. Unter anderem kann eine zweistufige Realisierung oder die Realisierung als disjunktive, konjunktive oder Reed-Muller-Normalform gefordert werden.
- *CheckFormulaApplet* überprüft arithmetische Ausdrücke der vorgegebenen Variablen durch numerische Auswertung an vorgegebenen Stützstellen bei einstellbarer Genauigkeit.

Diese grundlegenden und universell einsetzbaren Klassen werden durch zusätzliche Applets ergänzt, die jeweils einen spezialisierten Anwendungsbereich aus der technischen Informatik abdecken oder teilweise sogar speziell für einzelne Übungsaufgaben entwickelt wurden:

- *CheckGrayCodeApplet* und *CheckFanoCodeApplet* überprüfen, ob die eingegebenen Werte einen einschrittigen Code bzw. einen Fano-Code mit der geforderten Häufigkeitsverteilung der Codewörter bilden. Beide Applets erkennen und akzeptieren alle der vielen gleichwertigen Lösungen, siehe Abbildung 3.

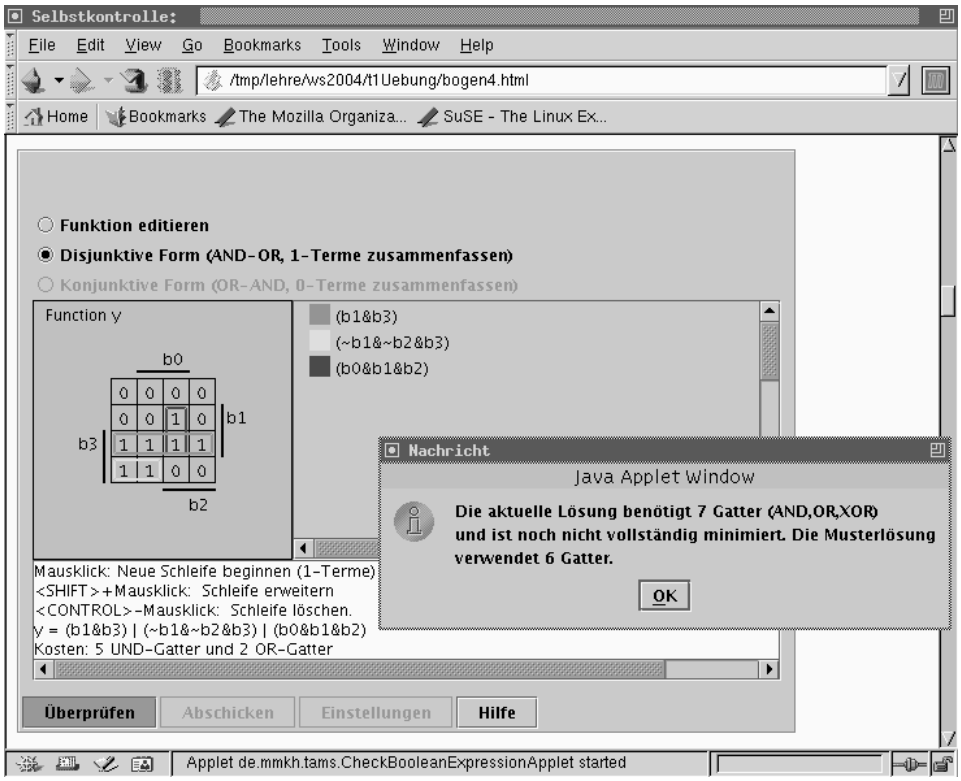

Abbildung 6: Applets zur interaktiven Logikminimierung von Funktionen mit Karnaugh-Veitch Diagrammen. Die Benutzeroberfläche der Applets ist bewusst einfach und einheitlich gehalten, um keine Hemmschwelle zur Benutzung der Überprüfung aufzubauen.

- *CheckKMapApplet* erlaubt die interaktive Eingabe und Minimierung von Boole'schen Ausdrücken mit KV-Diagrammen (disjunktive und konjunktive Form, auch mit Don't-Cares), siehe Abbildung 6. Die zugrundeliegende Überprüfung der Ausdrücke basiert auf CheckBooleanExpressionApplet.
- *CheckTwolevelApplet* dient zur interaktiven Eingabe von zweistufigen Logikschaltungen durch Anklicken der Verbindungen in einer UND-ODER (bzw. ODER-UND) Matrix. Das User-Interface verwendet die für programmierbare PAL- bzw. GAL-Logikbausteine übliche Darstellung.
- *CheckWaveformApplet* erlaubt die interaktive Eingabe und Überprüfung von Impulsdiagramme für die Zeitabläufe in Logikschaltungen. Das wiederum bewusst einfach gehaltene User-Interface basiert auf einem festen Zeitraster und dem 0/1/X/Z-Logikmodell; durch Anklicken eines Feldes wird der jeweilige Wert geändert.
- Funktionen zur Überprüfung von Automaten und digitalen Schaltungen wurden separat in bereits vorhandene Applets integriert.

Insgesamt können mit den verschiedenen Applets bereits etwa 85% der Übungsaufgaben zu unserer Grundlagenvorlesung zur technischen Informatik abgedeckt werden.

Durch die objektorientierte Architektur ist die Erweiterung um zusätzliche Applets jederzeit problemlos möglich; im einfachsten Fall muss lediglich die zugehörige *check()*-Methode neu implementiert werden. Der eigentliche Hauptaufwand besteht natürlich in der Analyse der Aufgabenstellung und dem Generieren nützlicher Hilfestellungen für die typischen Fehler der Studierenden.

5 Zusammenfassung und Ausblick

Die Applets wurden im WS'2004 erstmals in den Übungen zur Grundvorlesung Technische Informatik eingesetzt. Die Evaluierung ist noch nicht abgeschlossen, da die Applets auch im laufenden Semester eingesetzt werden, aber die bisherigen Resultate sind ermutigend. Sowohl das Bedienkonzept als auch die Hilfestellungen wurden überwiegend positiv bewertet. Fast alle Studierenden gaben an, durch die Überprüfung Fehler in ihren Lösungen gefunden (und vermieden) zu haben.

Die nächsten Schritte bestehen in der Erweiterung um zusätzliche Applets für weitere Aufgabentypen und der Integration verbesserter Hilfestellungen bei der Überprüfung von Automaten und digitalen Schaltungen.

Literatur

[Sch97] R. Schulmeister, *Grundlagen hypermedialer Lernsysteme: Theorie – Didaktik – Design*, Oldenbourg, 1997, ISBN 3-486-24419-1

[vdH04] K. v. d. Heide, *Vorlesung Technische Informatik 1*, Universität Hamburg, FB Informatik, WS'2004, http://tams-www.informatik.uni-hamburg.de/lehre/ws2004/vorlesungen/t1/

[SS01] W. Schiffmann, R. Schmitz, *Übungsbuch zur Technischen Informatik 1 und 2* (2. Auflage), Springer Verlag, 2001, ISBN 3-540-42171-8

[Tan99] A. S. Tanenbaum, *Structured Computer Organization, 4th. Edition*, Prentice Hall, 1999 ISBN 0-13-020435-8

[SB03] M. D. Shermis and J. C. Burstein (Eds.), *Automated Essay Scoring*, Lawrence Erlbaum Associates, 2003 ISBN 0-8058-3973-9

[ETS04] Educational Testing Service (ETS), Princeton, New Jersey *e-rater*, www.ets.org/erater/

[BKW03] C. Beierle, M. Kulas, M. Widera, *Automatic Analysis of Programming Assignments*, Proc. DeLFI-2003, 144–153

[Hen02] N. Hendrich, *Automatic checking of students' designs using built-in selftest methods*, Proc. 3rd European Workshop on Microelectronics Education, EWME-2002, Baiona, 2002

[JCC97] *Java compiler compiler - the Java parser generator*, https://javacc.dev.java.net/