Automatische Überprüfung und Hilfestellung zu Vorlesungs-begleitenden Übungen

Klaus von der Heide, Norman Hendrich

Universität Hamburg, Fachbereich Informatik

Vogt-Kölln-Str. 30, D 22527 Hamburg

hendrich@informatik.uni-hamburg.de

`tams-www.informatik.uni-hamburg.de`

Übersicht

Motivation

- Ausgangssituation für das Projekt
- Das interaktive Skript

Integrierte Übungsaufgaben

- Klassifikation der "typischen" T-Übungsaufgaben
- Ansätze zur automatischen Überprüfung
- Ansätze zur Hilfestellung für die Studenten

Implementierung

- Anforderungen und Plattform
- Beispiele

Diskussion

Ausgangssituation

- T-Lehrstoff gilt als schwer
- geringes Interesse vieler Studenten

"Augen zu und durch"-Ansatz:

- Vorlesung anhören, aber kaum nachbereiten
- sehr schlechte aktive Beteiligung an den Übungen (T1/T2)
- Praktikum als Nachhilfekurs nutzen
- Klausur/Prüfung versuchen (man darf ja mehrmals)
- Stoff möglichst schnell wieder vergessen (?)

=> Übungen direkt in die Vorlesung/Skript integrieren
=> und zwar mit interaktiven Hilfsmitteln ("Applets")
=> sofortige automatische Überprüfung der Lösungen
=> Hilfestellungen zu Lösungsansätzen, soweit möglich

Ziele

- Übungsaufgaben im Skript integriert:

Unterstützung der Studierenden:

- geringere Hemmschwelle zur Bearbeitung der Aufgaben
- automatische Überprüfung der Lösungen
- sofortiger Feedback (nicht erst eine Woche später)

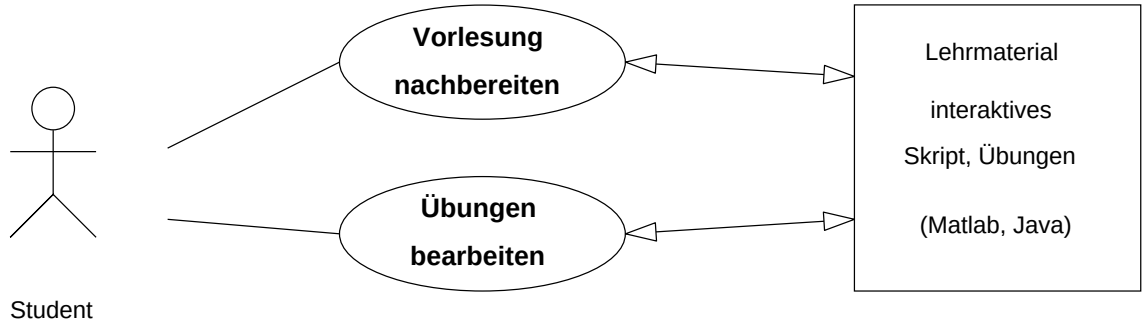
- kontextabhängige Hilfestellungen
- gezielte Gegenbeispiele helfen bei der Fehlersuche

Unterstützung der Übungsgruppenleiter

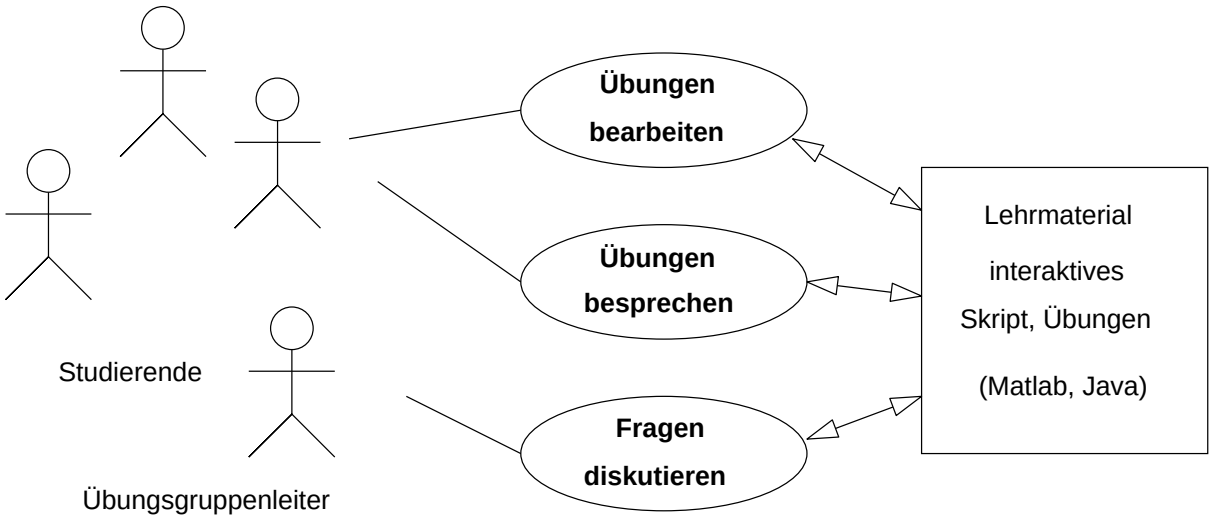
- erleichtert das "Ausprobieren" während der Übungsstunden
- automatische (Vor-) Korrektur vieler Aufgaben

Use-Cases

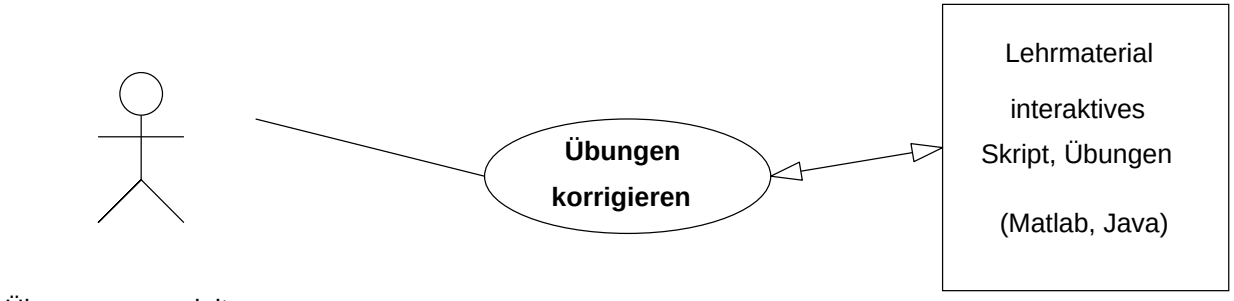
a)



b)



c)



Das interaktive Skript

bzw. "interaktives Lehrbuch"

- erläuternde Texte, eingebettete Formeln
- eingebettete Medien: Graphiken, Animationen, Audio, Video
- Hyperlinks, Verweise im Skript, auf Webseiten, auf externe Programme

- eingebettete aktive Applets (mit GUI)
- eingebettete aktive Skripte
- eingebettete Übungen (mit sofortiger) Überprüfung

- jederzeit erweiterbar (auch von den Studierenden)
- auch später im Berufsleben produktiv nutzbar

- einfache Content-Erstellung

Das interaktive Skript: Browser

t4_4

```
randb(Anzahl,[-1,+1])
```

Achten Sie darauf, dass alle Muster nur die Symbole {-1,+1} benutzen.
Setzen Sie folgende Parameter:

```
neutral = 1 ;   tol = 0.1;
```

```
schrittsync( 100, 102.3, randb(20,[-1 +1]), 0.4, 1, 0.1);
```

Versuch 4.3: Wiederholen Sie Versuch 4.2 unter Einsatz des Bitstuffing.
Verwenden Sie hierbei die Funktion bitstuffing oder/und geeignete Bitmuster wie z.B.

```
2*[1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1] - 1
```

Folgende Fragen sind zu beantworten:
c. Wird das Maximum des Spektrums an der Schrittfrequenz deutlicher?
d. Werden andere Maxima geschwächt oder gar kräftiger?

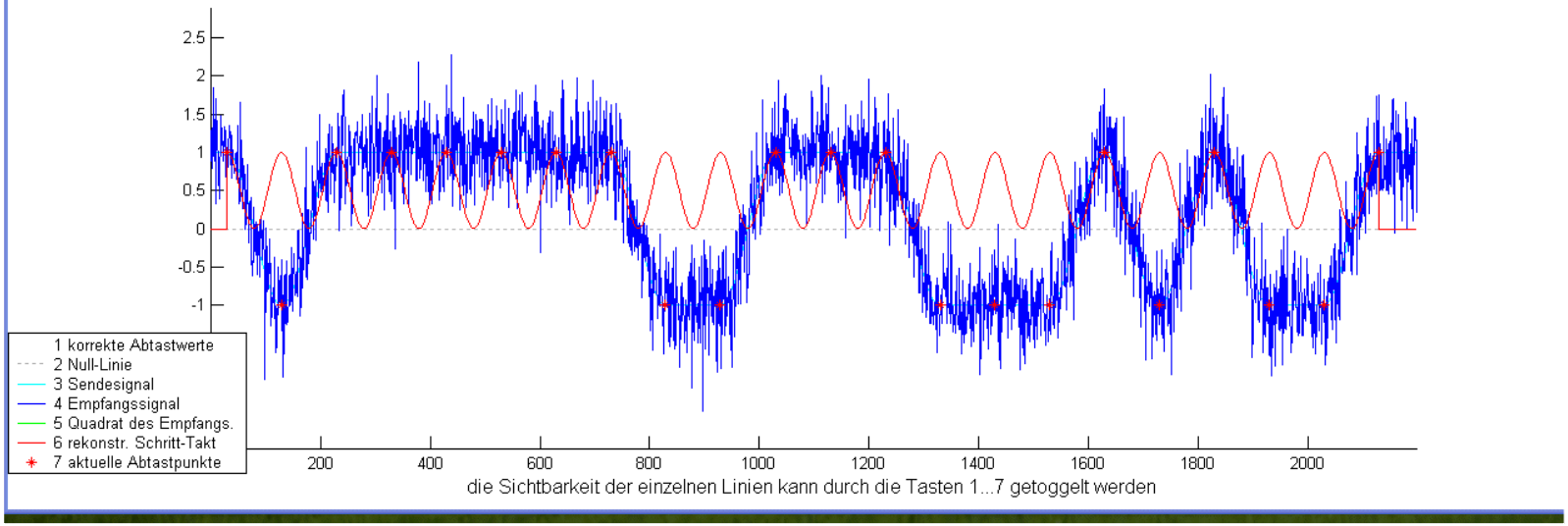
t4

T4-Praktikum

1. Zeitbereich und Frequenzbereich
2. Abtastung
3. Datenübertragung im Basisband
4. Schrittsynchronisation
5. Modulation

Schrittsync

File Edit Tools Window Help



Eingebettete Skripte:

- Beispiel-Code im Matlab-Browser: **t1_3_1.m**

```
% Die folgende Funktion bietet eine interaktive Demo für die
% drei Formate nach IEEE-754:
demoieee754
% Werden Operationen durchgeführt mit der Repräsenation für
% {\fontname{Courier}}NaN}, so ist das Resultat immer ...
% ...
inf/(-1+1)   % liefert das Resultat +inf
inf/-(1-1)   % liefert das Resultat -inf
```

- markierter Code kann in den Editor kopiert werden
- Experimente mit anderen Parametern
- Erweiterung der bestehenden Funktionen

- einfache Content-Erstellung durch "Kommentar-Trick"

Das interaktive Skript: "Applets"

These: Verständnis erfordert "Spielen" mit dem Stoff

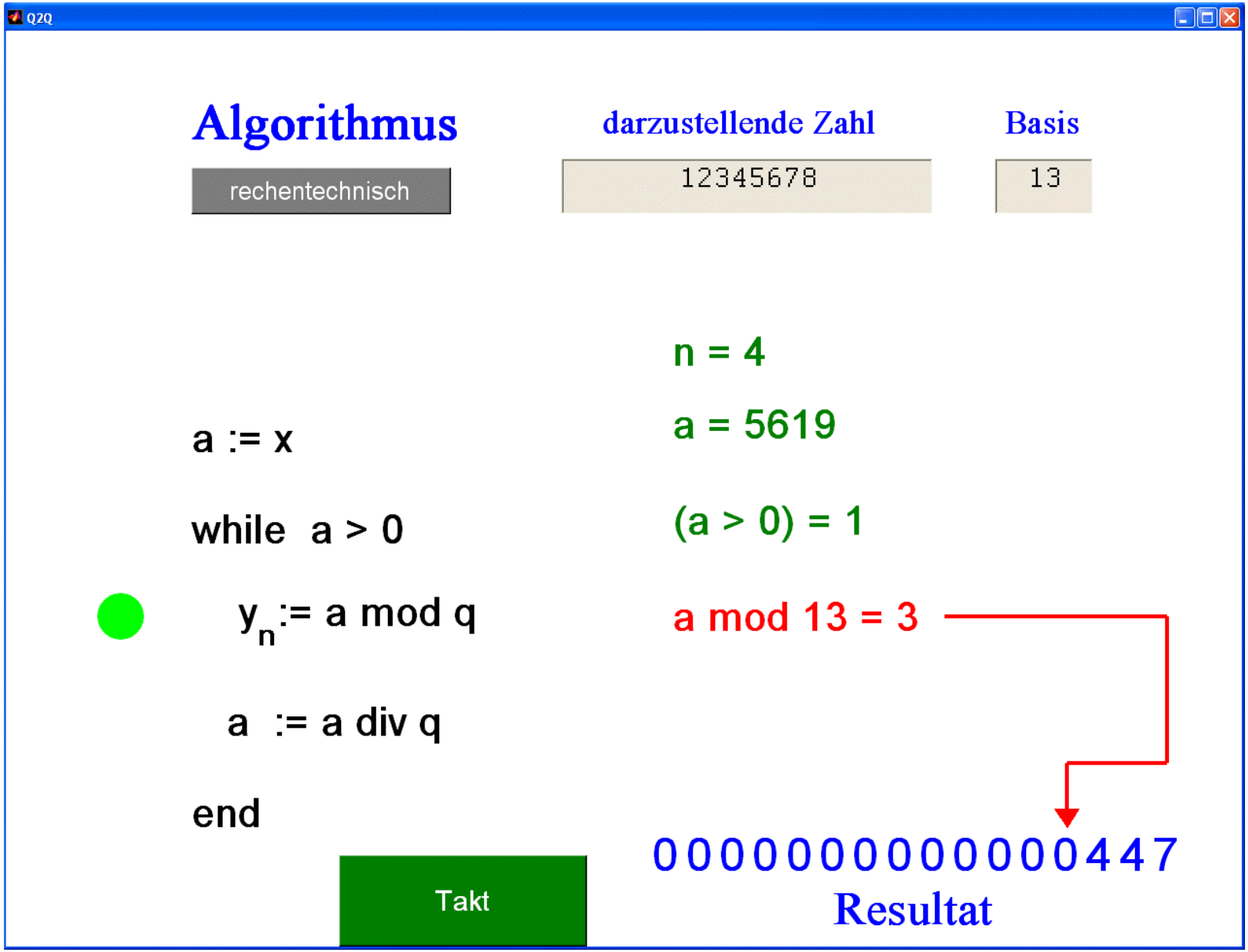
- Umgang mit Matlab kann nicht immer vorausgesetzt werden
- vereinfachter Zugang wünschenswert

=> zusätzliche interaktive "Applets"

- im Skript eingebettete kleine Applikationen
- eigenes, möglichst eingängiges User-Interface
- einfachere Bedienung als über die Kommandozeile
- auch standalone einsetzbar und erweiterbar
- Matlab: handle-graphics unterstützt alle üblichen GUI-Komponenten

- Demo (Zahldarstellung / IEEE-754 / t1_3_1.m)

Das interaktive Skript: Applets



Übersicht

Motivation

- Ausgangssituation für das Projekt
- Das interaktive Skript

Integrierte Übungsaufgaben

- Klassifikation der "typischen" T-Übungsaufgaben
- Ansätze zur automatischen Überprüfung
- Ansätze zur Hilfestellung für die Studenten

Implementierung

- Anforderungen und Plattform
- Beispiele

Diskussion

Lehrstoff im T-Zyklus:

T1

- Information, Repräsentation
- Zahlensystem, Arithmetik
- Schaltnetze, Schaltwerke
- Schaltungsentwurf
- von-Neumann Rechner

T2

- Lineare Netze
- Bauelemente, Logik-Glieder
- Halbleiterspeicher
- Programierbare Bausteiene
- Mikroelektronik, Prozesse

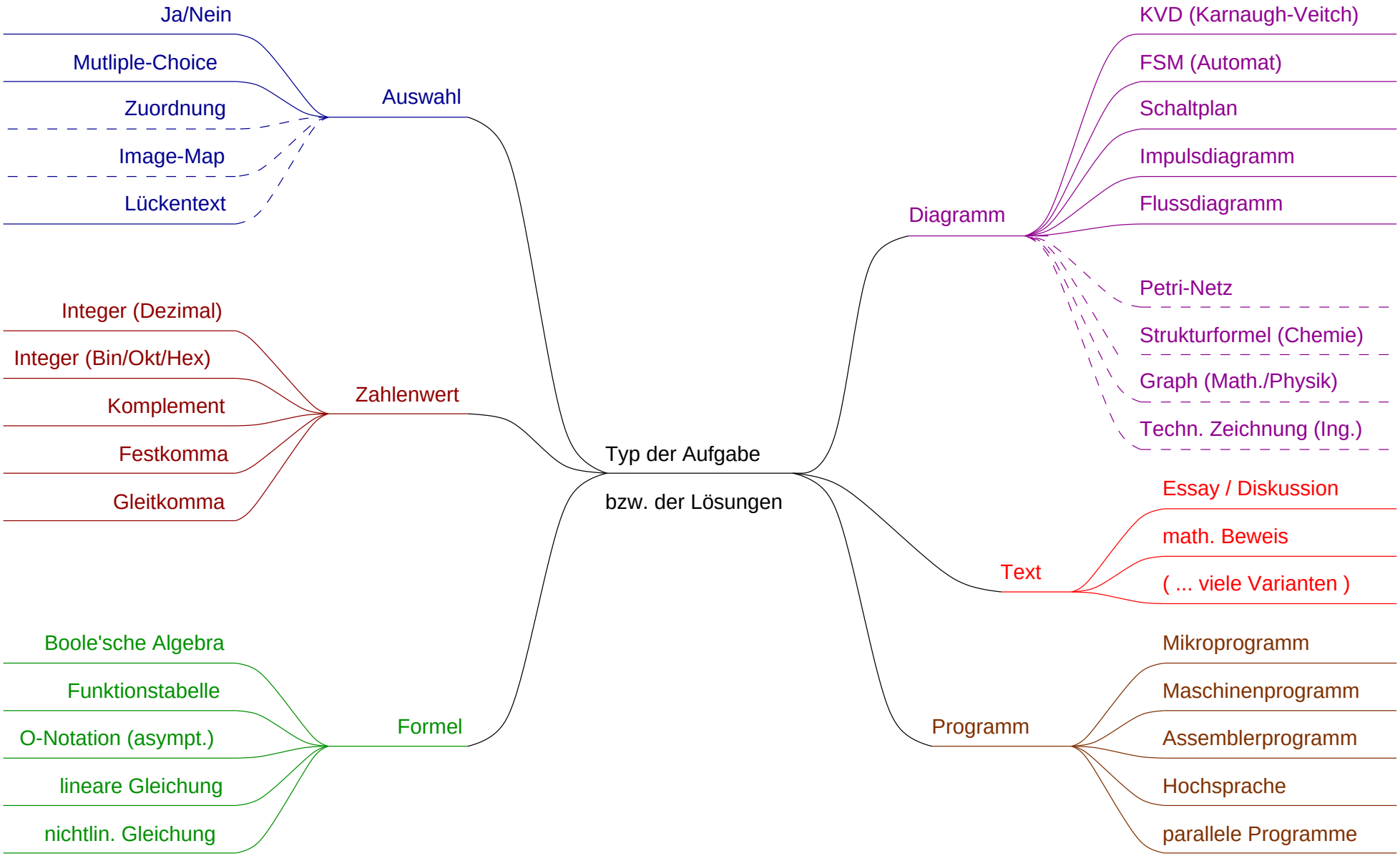
T3

- Rechnerarchitektur
- Pipelining, Caches
- Ein-/Ausgabe, Interrupts
- Betriebssysteme
- Scheduling
- Schutzmeachnismen

T4

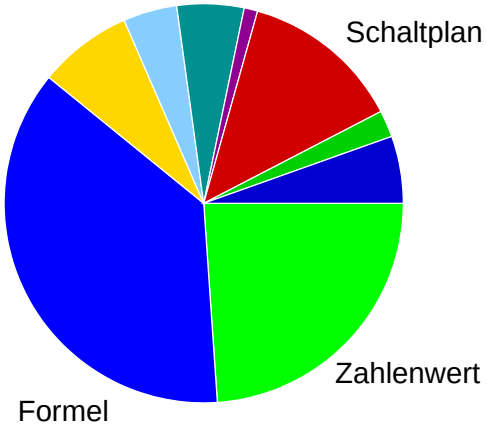
- Informationstheorie
- Parallelverarbeitung
- Datenübertragung
- Vermittlungsnetze
- Lokale Rechnernetze
- Modellierung, Analyse, Entwurf

T-Übungsaufgaben: Klassifikation

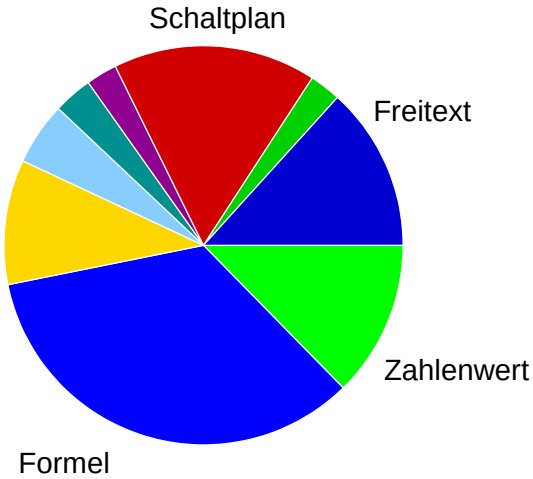


Häufigkeiten

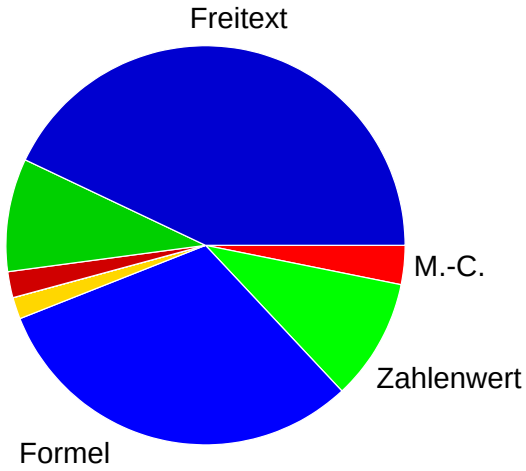
von der Heide (T1/T2)

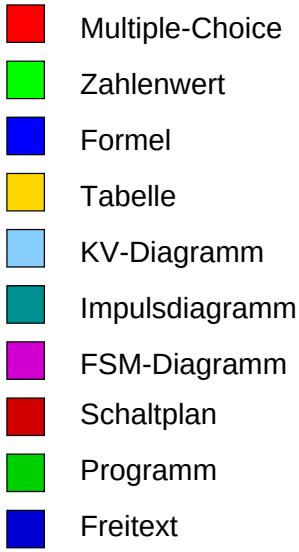


Schiffmann/Schmitz (T1/T2)



Tanenbaum (T1/T3)





- (fast) keine Auswahl-Aufgaben
- diverse Repräsentationen für Zahlenwerte
- logische Ausdrücke unter Formeln eingeordnet
- Funktionstabellen, Schaltpläne, FSMs

- T4 nicht berücksichtigt (eigenes Projekt)

"State of the Art" bei Frameworks

- Übungsaufgaben im Skript integriert:

Unterstützung durch E-Learning Frameworks? Beispiel CLIX:

- Multiple-Choice
- Zuordnungs-Fragen, Reihenfolgen
- Image-Maps
- Numerische Zahlenwerte
- Lückentexte

- Fließtext (aber manuelle Auswertung)

=> deckt weniger als 3% aller untersuchten Aufgaben ab

=> für technische Informatik völlig unzureichend

E-Learning Software: Klassen

- "am besten lernt man im Streit"
- "Instruktion" vs. "Lernen"

Interaktion ist zentrales Konzept; z.B. sechs Klassen:

0	statische Inhalte, keine Interaktion	(HTML Webseiten)
1	Repräsentation/Darstellung ändern	(CMOS demo)
2	Inhalte ändern	(T1 IEEE-Demo)
3	1) und 2) kombiniert	(T3-Praktikum: Speicherinhalte)
4	Inhalte konstruieren	(T1 FSM-Editor, Hades, Cinderella)
5	4) kombiniert mit Feedback	(dieses Projekt, T3-Tutor, ...)

- unterschiedliche Akzeptanz (z.B. Ingenieure vs. Geisteswissenschaftler)
- Feedback schnell (< 24h), sonst Frustration des Lernenden

(Schulmeister 2003)

Überprüfung und Hilfestellung

- Vergleich mit Musterlösung
- Normalformen (Sortierung, Vereinfachung, Substitution, usw.)

- Überprüfung von Bedingungen ("constraints")
- z.B. Existenz / Typrestriktion / Anordnung / Referenz

- Simulation, Testläufe
- erfordert lauffähige Spezifikation (z.B. keine Syntaxfehler)
- bei Programmen: Vorkorrektur von "Tippfehlern"

- Kombination der Verfahren
- abgestuftes Feedback: falsch / was ist falsch / ... / Korrekturvorschlag
- stimulierendes Feedback (sukzessive Verbesserung der Lösungen)
- aber: Struktur der "richtigen" Lösung muss bekannt sein

- "Verraten der Lösung" ?

Ansätze zur Überprüfung

	Art der Aufgabe:	Ansatz zur Überprüfung
■	Auswahl	Vergleich mit Musterlösung
■	Zahlenwert	Werte- und Einheitenvergleich
■	Formel	symbolische und numerische Auswertung, Plausibilität
■	Tabellen	wie Zahlenwerte und Formeln
■	Diagramm	anwendungsspezifisch
■	KV-Diagramm	vollständige Auswertung, evtl. Normalform
■	FSM	Simulation und Test der Struktur
■	Schaltplan	Simulation (vollständig, pseudozufällig)
■	Programm	Ausführung, Vergleich der Ausgaben
■	Text	nicht unterstützt

Ansätze zur Hilfestellung

	Art der Aufgabe:	Ansatz zur Überprüfung
■	Auswahl	nicht möglich / anwendungsspezifisch
■	Zahlenwert	Plausibilität, Wertebereich, Dreher, Einheiten
■	Formel	Gegenbeispiele aus vollständiger Auswertung Plausibilität
■	Tabellen	wie Zahlenwerte und Formeln
■	Diagramm	anwendungsspezifisch
■	KV-Diagramm	Gegenbeispiele, Hinweise auf Probleme
■	FSM	Test der Struktur, Gegenbeispiele
■	Schaltplan	Test der Struktur, Gegenbeispiele
■	Programm	anwendungsspezifisch
■	Text	nicht unterstützt

"Design-for-Überprüfbarkeit" ?!

Auswahl oder Formulierung der Übungsaufgaben
in Hinblick auf einfache automatische Überprüfbarkeit:

- machbar?
- didaktisch vertretbar?
- lohnt das überhaupt?

- Beispiel: multiple-choice Klausuraufgaben
  - sehr einfach zu überprüfen und zu bewerten
  - aber unrealistische Aufgabenstellung

- bleibt ein akzeptables Aufgabenspektrum erhalten?

Beispiel: einschrittiger Code (1)

Aufgabe T1.2.4:

Begründen Sie, warum es keinen zyklisch-einschrittigen Code mit ungerader Anzahl von Codewörtern geben kann.

Beweisidee:

- benachbarte Codewörter unterscheiden sich in ihrer Parität
- gilt bei zyklischem Code auch für erstes und letztes Codewort
- nur bei gerader Anzahl von Wörtern erfüllbar

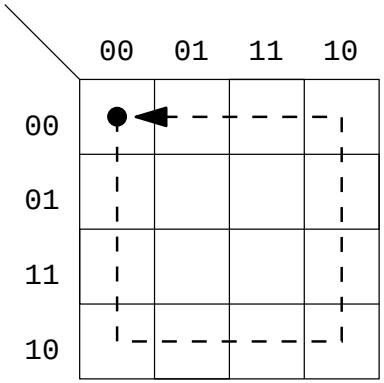
automatische Überprüfung unmöglich (sehr aufwendig):

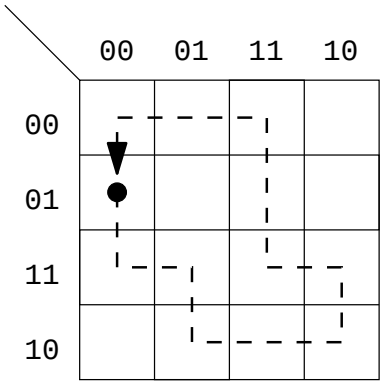
- Textanalyse mit Textverständnis
- einfache Verfahren (Schlüsselwortsuche etc.) reichen nicht aus

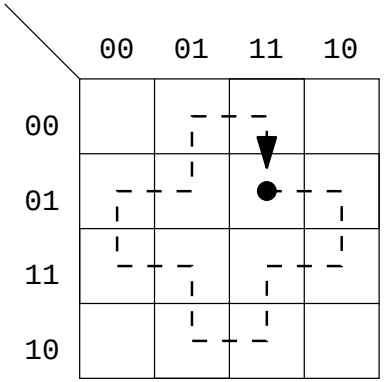
Beispiel: einschrittiger Code (2)

Aufgabe T1.2.5:

Finden Sie einen zyklisch-einschrittigen Code mit 12 Codewörtern. Ein solcher Code könnte z.B. für eine Winkelcodierung in 30 Grad Schritten benutzt werden.







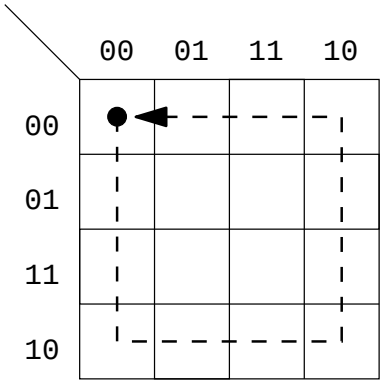
- Eingabe ist Liste mit 12 Codewörtern
- Lösung ist nicht eindeutig, viele Möglichkeiten

automatische Überprüfung ist einfach:

- Überprüfung von Anzahl und Länge der Wörter
- Überprüfung der Einschrittigkeit
- Beispielcode

Beispiel: einschrittiger Code (3)

Finden Sie einen zyklisch-einschrittigen Code mit 12 Codewörtern. Ein solcher Code ...




```
def check_T1_2_5( stringWithValues ):
    values = convertToValues( stringWithValues )
    if (len(values) != 12): error( 'falsche Anzahl der Codewörter' )
    for i in range(0, 12):
      if( nbits(values[i]) < 4): error( 'Codewort zu kurz:', values[i] )
    for i in range(0, 12):
      for j in range(0, 12):
        if (i == j) continue
        if (values[i] == values[j]): error( 'Codewort doppelt:', values[i] )
    for i in range(0, 12):
      j = (i+1) mod 12
      if( hamming(values[i],values[j])!=1): error( 'nicht einschrittig' )
    return 'Lösung ist korrekt'
```

Übersicht

Motivation

- Ausgangssituation für das Projekt
- Das interaktive Skript

Integrierte Übungsaufgaben

- Klassifikation der "typischen" T-Übungsaufgaben
- Ansätze zur automatischen Überprüfung
- Ansätze zur Hilfestellung für die Studenten

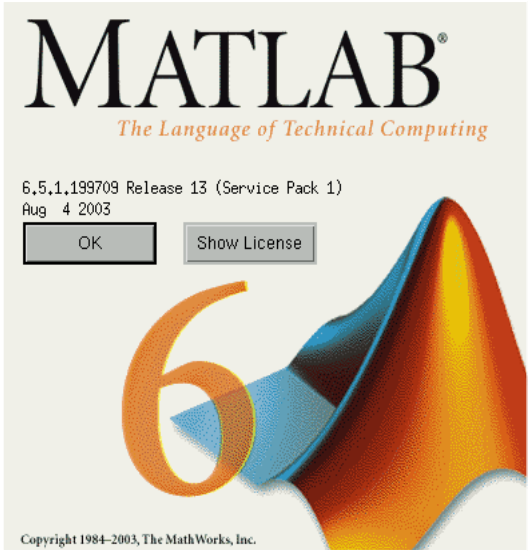
Implementierung

- Anforderungen und Plattform
- Beispiele

Diskussion

Matlab:

"Matrix Laboratory"



- System für rechnergestützte Mathematik
- entwickelt seit ~1984, www.mathworks.de
- Schwerpunkt auf technischen Anwendungen

- einfache Syntax für Vektor-/Matrixoperationen
- numerische und symbolische Algorithmen

• "workspace"	IDE mit Editor und Debugger
• "handle-graphics"	objektorientierte Graphik/Plots
• "toolboxes"	anwendungsspezifische Erweiterungen
• "simulink"	graphische Modellierung
• "system-level design"	Codeerzeugung für DSPs / FPGAs

Matlab: Codebeispiele (1)

```
x = 0 : pi/100 : 2*pi;    % Vektor mit 200 Elementen
y = sin( x );             % ditto
plot( x, y );             % 2D-Funktionsplot, autoscale
xlabel( '0 : 2\pi' );     % x-Label, TeX-Annotation
...

% Samplerate, Tonfrequenz, Bits pro Minute
fs = 4000; frequenz = 800; bpm = 160; string = 'ELCHTEST';
morse = morsecode( string, round(fs/10*bpm/60)*[1 3 1 3 4]));
envel = filter( fir1( 60, bpm/f2 ), 1, morse );
tone  = envel .* sin(2*pi*frequenz*(1:length(envel))/fs);
sound( tone, fs );
```

- Demo (morsesound aus t1_4.m)

Matlab: Codebeispiele (2)

```
A = [1,1,1; 1,2,3; 1,3,6]; % Pascal(3) Matrix
b = [3; 1; 4];             % Spaltenvektor
x = A \ b;                 % Lösung des Gleichungssystems Ax=b
...

s = '((x-2).^2 - 5)';        % String
f = inline( s );             % Inline-Funktion
v = feval( f, 0.3 );         % Auswertung f(0.3)
z = fzero( f, 0.1 );         % sucht Nullstelle nahe x=0.1
m = fminbnd( f, 0, 4 );      % Minimum in [0..4]
fplot( f, [0, 6] );          % Funktionsplot
...
```

- alle gängigen Matrixoperationen verfügbar

Matlab: Java-Interface

seit Matlab 5.3 auch Zugriff auf Java-Objekte:

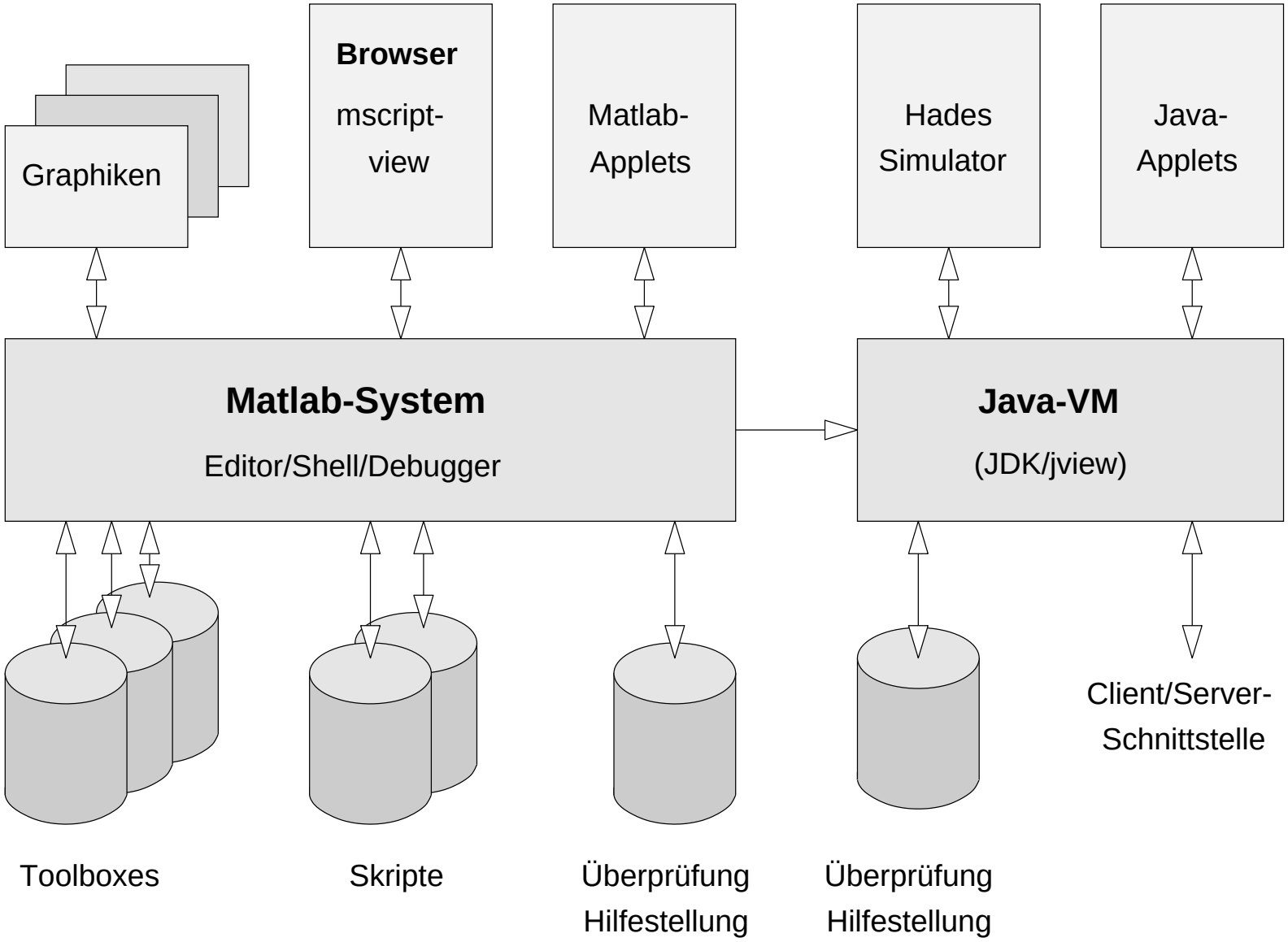
```
java on;                          % für Matlab 5.3

generator = java.lang.Random % Konstruktor
x = generator.nextDouble     % Methodenaufruf

editor = hades.gui.Editor    % Hades-Editor
editor.doOpenDesign( 'test.hds', 0 );
editor.getSimulator.runFor( 3.0 ); % simuliert bis t=3.0 sec.
...
```

- läuft auch mit 5.3 Student Edition (einige Einschränkungen)
- Konfiguration über CLASSPATH

Architektur



Datenformate

- Handschrifterkennung bisher nicht robust genug
- komplexe Dokumentenformate (Word .doc usw.)?
  - keine geeigneten, robusten Parser
  - keine eindeutige Repräsentation von Formeln / Diagrammen
  - Versionsvielfalt und -probleme

=> Beschränkung auf einfache Datenformate:

• ASCII-Textdateien	Zahlenwerte, Tabellen, Formeln
• Matlab-Quelltexte	Formeln, Funktionen, Graphiken
• Programm-Quelltexte	z.B. C / Java / Assembler
• Hades-Designdateien	Schaltnetze, Schaltwerke
• HTML-Formulare	direkte Einbettung in Webseiten
• XML	Metadaten / Verwaltung

Datenformate: Beispiel

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE uebungsaufgabe SYSTEM "aufgabe.dtd">
<student-name="Marianne Muster" />
<matrikelnummer="9870815" />

<aufgabe nummer="1.2">
  <zwischenrechnung>
      1998 : 2 = 999 Rest           0
       999 : 2 = 499 Rest          1
       499 : 2 = 249 Rest         1
       249 : 2 = 124 Rest        1
       124 : 2 =  62 Rest       0
        62 : 2 =  31 Rest      0
        31 : 2 =  15 Rest     1
        15 : 2 =   7 Rest    1
         7 : 2 =   3 Rest   1
         3 : 2 =   1 Rest  1
         1 : 2 =   0 Rest 1
      (1998)_10 =        (11111001110)_2
  </zwischenrechnung>

  <ergebnis>
      11111001110_2
  </ergebnis>
</aufgabe>
```

- unterstützt Integration in Frameworks
- XML-Parser trennt Meta-/Nutzdaten
- übrige Parser bekommen ASCII-Nutzdaten

Software:

- Matlab-Java-Interface:
  - GUI-Komponenten, JDialogFactory
  - Hades MatlabAdapter, BIST-Komponenten
  - ImageViewer
- Parser
  - NumberParser, DoubleParser, ArrayParser, ...
  - FormulaParser, BooleanExpressionParser
- Skripte
  - check_t1_1_1.m ... check_t3_14_x.m
- Jython-Console und -Tools

NumberParser

- Wertevergleich, Hilfestellungen optional (zu klein/zu groß/...)
- überprüft Zahlenbasis, Stellenzahl
- get/set-Methoden zum Einstellen der erwarteten Werte
- abgeleitete Klassen für Spezialformate (etwa Binärbrüche)
- Mitzählen der Anfragen, Zeitbudget (?)

```
np = de.mmkh.tams.NumberParser;
np.setExpectedBase( 16 );           // hexadezimal
np.setExpectedNumberOfDigits( 4 ); // vier Stellen
np.setExpectedValue( 42 );          // Klartext

status = np.parse( "0042_16" );     // ok.
msg    = np.getMessage              // "richtig!"
```

Mogeln ?!

- Übungsaufgaben und Überprüfung direkt im Skript
- übliche Täuschungsversuche (Abschreiben&Co) wie sonst auch...

aber zwei zusätzliche Probleme:

- Programme liegen im Quelltext vor
- oder als perfekt decompilierbare Java-Klassen

=> Lösung einfach im Skript ansehen

```
edit check_xxx.m      ...; setExpectedValue(42); ...
```

- interaktive Umgebung erlaubt Skripting
- und bietet Hilfestellungen an...

=> Lösungen per Skript durchprobieren

```
for i=1:10000000
  if (check_xxx(i) == 'richtig') result=i; break; end
end
```

NumberScrambler:

- Studenten haben den Quellcode der Skripte
- alle Parameter sind im Klartext zugänglich
- Java-Bytecode lässt sich disassemblieren

=> ganz so einfach sollte man nicht an die Lösung gelangen

- Erschwerung via "obfuscated" Methoden
- einfache XOR- und Shift-Operationen
- Verfahren ist natürlich nicht kryptographisch sicher

```
scrambler = de.mmkh.tams.NumberScrambler
mask      = scrambler.getRandomMask
hidden    = scrambler.scramble( mask, value )

// np.setExpectedValue( 42 );
np.setExpectedValue( mask, hidden );
```

FormulaParser

- Umsetzung von Text in Matlab ("inline") Formeln
- get/set-Methoden zum Einstellen der erwarteten Werte
- symbolische oder numerische Auswertung
- eigene Unterklassen für logische Ausdrücke
- Funktionstabellen

- viele Funktionen bereits vorhanden
- (Variablensubstitution, Polynomdivision, etc.)

- Implementierung ab Februar ...
- Demo: nächstes Semester

Hades+BIST

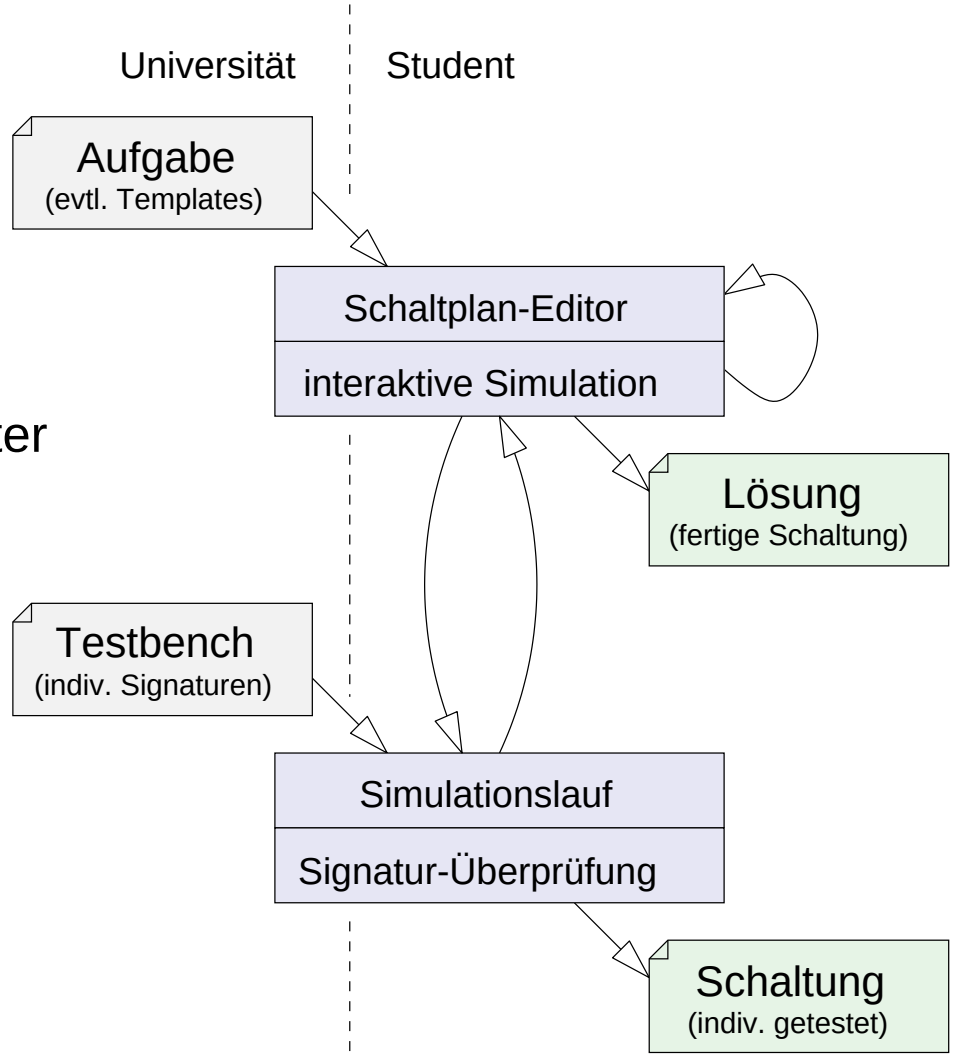
Überprüfung digitaler Schaltungen?

- formale Techniken
- built-in-selftest Simulation

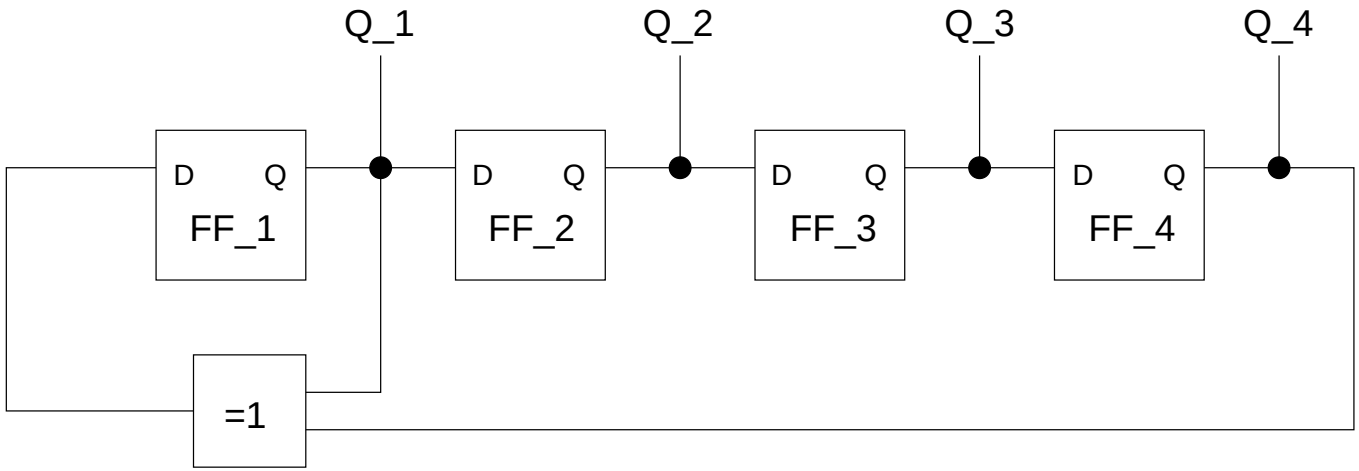
- vollständiger Test, alle Eingabemuster
- aufgaben-spezifische Muster
- pseudozufällige Muster

- individuelle Signaturen möglich
- auch Test von Teilschaltungen

- zusätzliche Plausibilitäts-Checks
- u.a. für die Hilfestellung



Linear-Feedback Shift-Register



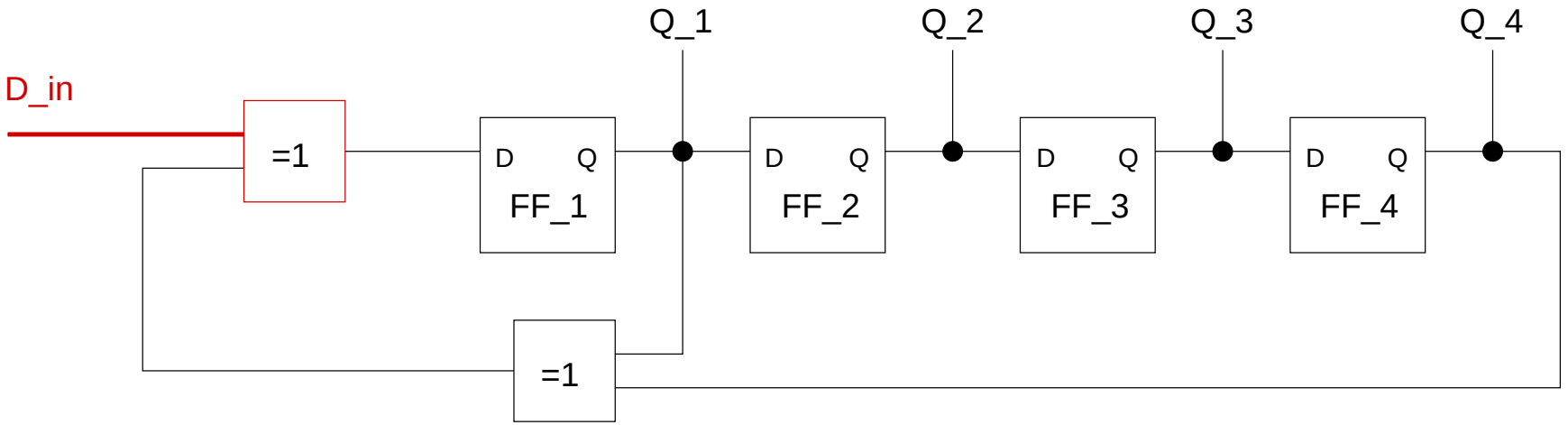
- n-bit Schieberegister, Rückkopplung über XOR-Gatter

falls die Auswahl der Gatter einem primitiven Polynom entspricht:

- Register durchläuft Sequenz aller Werte außer (0...0)
- Nutzung als n-bit (Pseudo-) Zufallsgenerator

- hades.models.utils.LFSR32 nutzt das (31 3 0)-Polynom

Signatur-Register



- n-bit Schieberegister, Rückkopplung über XOR-Gatter
- ein (oder mehrere) zusätzliche Einspeisungen über XOR-Gatter

- Summation der LFSR-Sequenz mit den Eingabedaten
- m Eingangsbits werden auf n Bits abgebildet

- Vergleich des Registerinhalts mit der bekannten korrekten Signatur

LFSR-Theorie

m Eingangsbits: mögliche Eingabefolgen 2^m

n Flipflops: mögliche Zustände 2^n

XOR- Verknüfpung ist linear (Summation im GF_2)

=> alle Einzelfehler werden erkannt

=> Anzahl der nicht erkannten Mehrbitfehler: $2^{m-n}-1$

=> Fehlererkennungsrate (bei Gleichverteilung)

$$F = 1 - (2^{m-n}-1) \;/\; (2^m - 1)$$
$$\approx 1 - 2^{-n}$$

$n = 16$: $F \approx 99.9985\%$

$n = 24$: $F \approx 99.999994\%$

$n = 32$: $F \approx 99.99999998\%$

BIST: Zusammenfassung

- Vorgabe der Testbench und evtl. von Teilschaltungen
- Simulation der Testbench mit Signaturanalyse
- pseudozufällige oder maßgescheiderte Eingabedaten

- korrekte Lösung liefert korrekte Signatur
- aber kein Rückschluß von falscher Signatur auf Fehlerursache
- getrennte Analyse für jeden Ausgang der Schaltung möglich
- geringer / akzeptabler Zeitaufwand
- zusätzliche Plausibilitäts-Checks und Tests für die Hilfestellung

- individuelle Startwerte für individuelle Signaturen
- (erschwert Abschreiben / Kopieren der Schaltung trotzdem möglich)
- Test des Zeitverhaltens möglich (z.B. ripple- vs. carry-lookahead adder)
- evtl. Probleme mit selten erreichten Zuständen / Ausgangswerten

Übersicht

Motivation

- Ausgangssituation für das Projekt
- Das interaktive Skript

Integrierte Übungsaufgaben

- Klassifikation der "typischen" T-Übungsaufgaben
- Ansätze zur automatischen Überprüfung
- Ansätze zur Hilfestellung für die Studenten

Implementierung

- Anforderungen und Plattform
- Beispiele

Diskussion

Alternative Plattformen?

diverse Anforderungen:

- Text, Formeln, Abbildungen, Hyperlinks, Drucken
- skriptfähige Umgebung, Ausführen von Text als Code
- kostenlos (aber nicht unbedingt GPL)
- portabel, mindestens Windows/Linux/Unix, ideal auch PDA
- einfache Installation und Konfiguration
- langfristig verfügbar und unterstützt
- intuitive Bedienung der Tools
- GUI-Bibliotheken, Applets, Audio/Video
- numerische und symbolische Mathematik

- Client- oder Serverbasierte Lösungen?
- mögliche Integration in E-Learning Frameworks?
- Eignung für Grund- und Hauptstudium?

Alternative Plattformen?

- skriptfähig, portabel, günstig, GUI, Numerik, symb. Mathematik, ...

- HTML und Applets (JavaScript, Java, Flash, ...)

- MS-Word und Matlab/Mathematica/Maple
- PDF und externe Programme

- Tcl/Tk für C/C++
- Python und C++
- Jython und Java

- jeweils individuelle Vor- und Nachteile
- keine optimale, eindeutige Lösung
- aber: P-Zyklus setzt derzeit auf Java
- weitere Kriterien?

Warum nicht einfach HTML+Applets?

- HTML Texte mit interaktiven Elementen
- einfachste Bedienung, keine Installation notwendig

aber:

- Probleme bei lokaler Installation (z.B. http:// vs. file://)
- sehr restriktive Sicherheitsmechanismen, z.B.
  - keine globalen (persistenten) Daten möglich
  - eingeschränkte Client-Server Kommunikation
  - eingeschränkter Zugriff auf Funktionsbibliotheken
- Dokumentenformat liegt fest (=HTML)
- Demo unter: http://tams-www/applets/jython/

=> lokale Installation als Applikation

=> oder eigener erweiterbarer (XHTML-) Browser

Jython

Python / Jython:

- objekt-orientierte Skriptsprache
- Guido van Rossum, CWI Amsterdam, 1991
- derzeit eine der "modernsten" Skriptsprachen (?!)

- Objekte, Module, Klassen, Mehrfachvererbung
- dynamische Typisierung, keine primitiven Datentypen
- einfache Syntax, Blockbildung über Einrückung
- umfangreiche Klassenbibliotheken

- Jython: Re-Implementierung in Java, B. Hugunin, 1999
- volle Integration in Java-API (Jython)
- praktisch gleiche Performance wie C-Python

Jython: Beispiel

```
import java

def findObject( editor, comment ):
  iterator = editor.getObjects()
  while iterator.hasMoreElements():
    obj = iterator.nextElement()
    tmp = obj.getComment()
    if (tmp == comment):
      return obj
  return None

def transistorDemo( editor ):
  tran = findObject( editor, "T1-channel" )
  if (tran == None):
    print "Object 'T1-channel' not found, sorry!"
    return

  for i in range(0,10):
    tran.setLineColor( java.awt.Color.blue )
    editor.doRedraw()
    java.lang.Thread.currentThread().sleep( 500 )

    tran.setLineColor( java.awt.Color.red )
    editor.doRedraw()
    java.lang.Thread.currentThread().sleep( 500 )
  return

if __name__ == "__main__":
  import jfig
  editor = jfig.gui.JModularEditor()
  editor.doParseFile( "transistor.fig", 0 ) # don't merge
  java.lang.Thread.currentThread().sleep( 2000 )
  transistorDemo( editor )
```

Python: Syntax :-)

Jython: Status

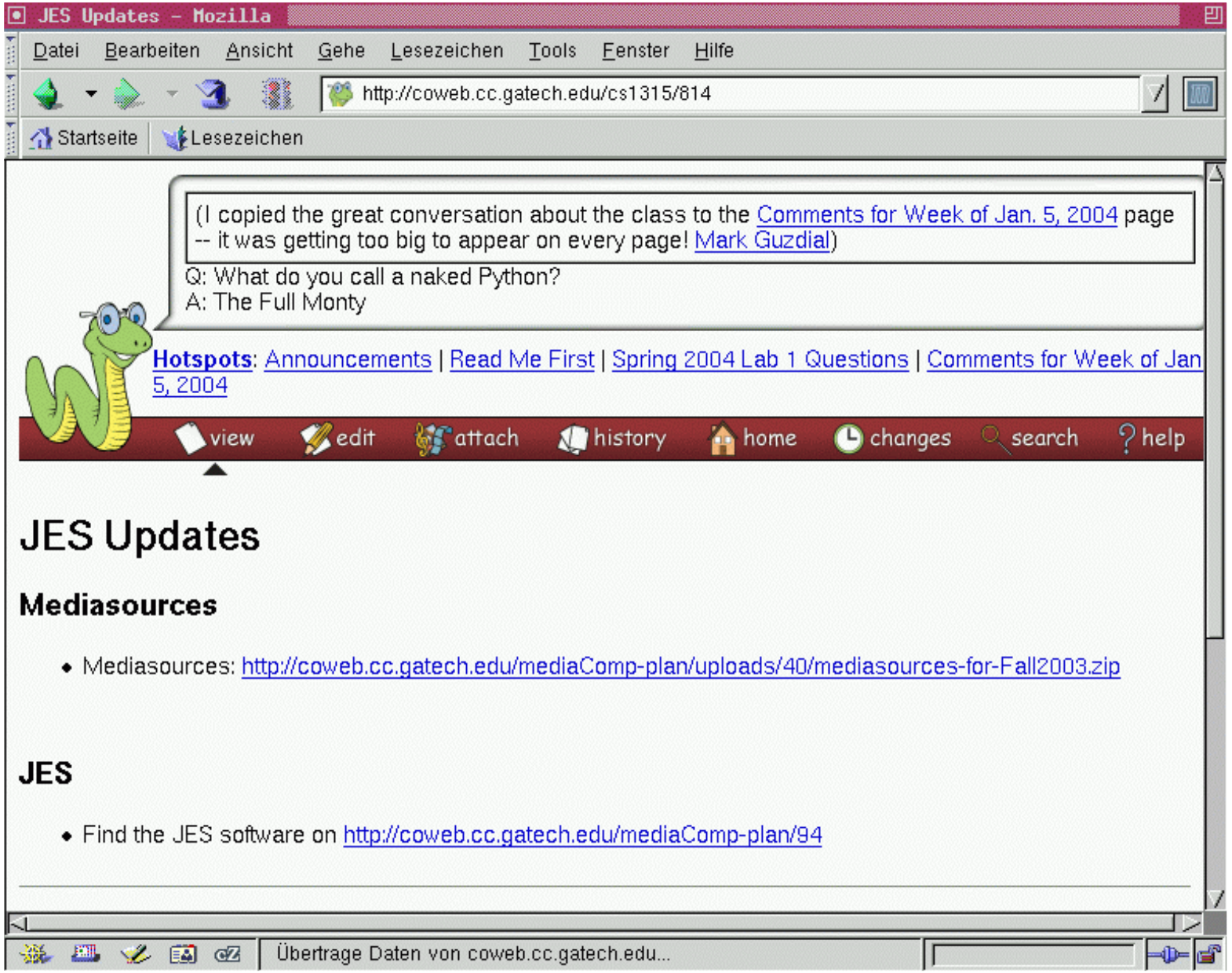
- gute Skriptsprache für Java-Applikationen
- Zugriff auf Java-Klassenbibliotheken und viele Python-Module
- akzeptable bis gute Performance

- rudimentäre Unterstützung komplexer Zahlen
- aber: Python Numerics-Modul funktioniert nicht
- keine brauchbaren Bibliotheken für numerische Mathematik

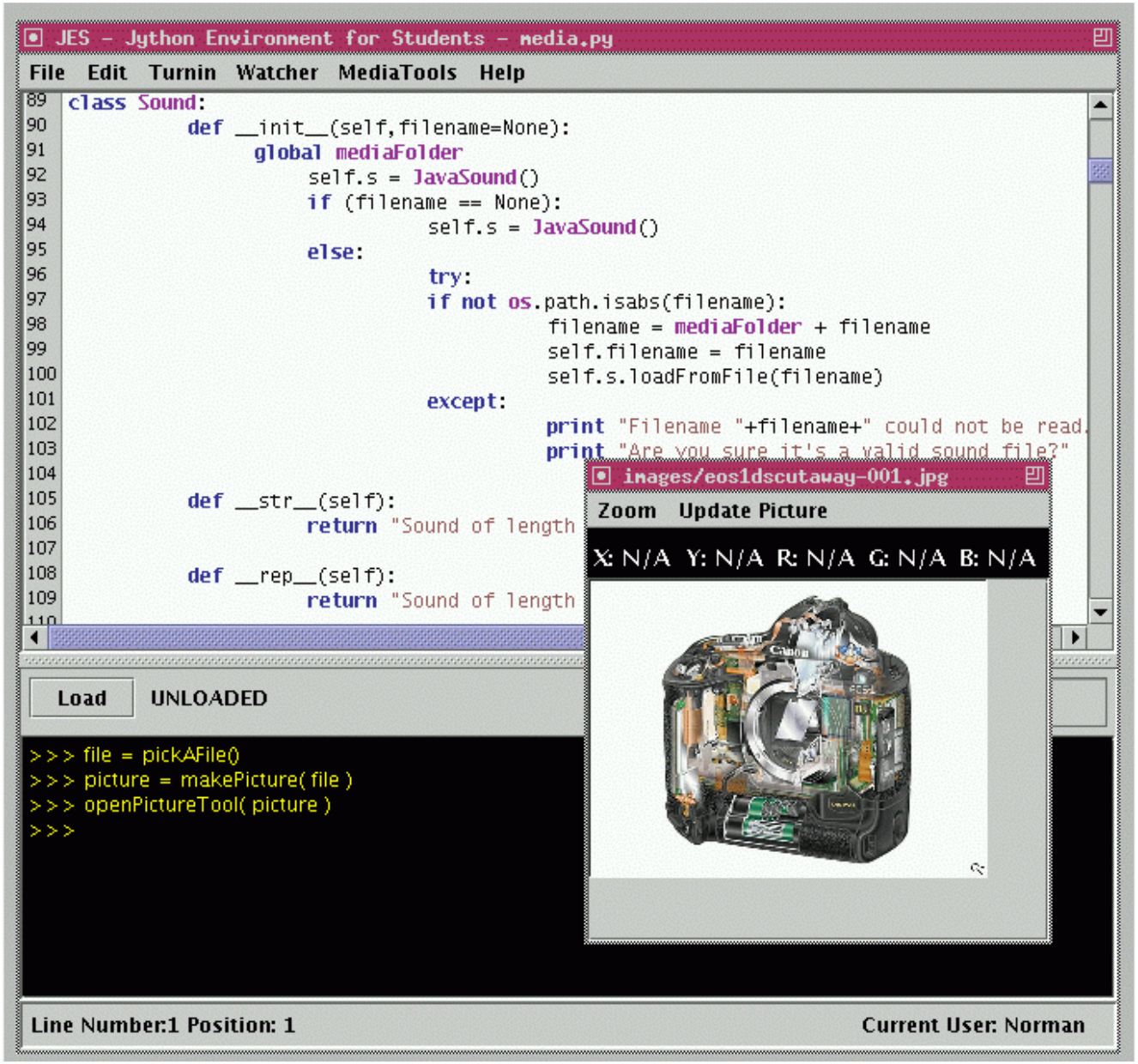
- diverse "Plot"-Bibliotheken (VISAD, SGT, jchart, ...)
- aber entweder geringer Funktionsumfang oder sehr komplex

- vgl: JES (Jython Environment for Students, GeorgiaTech, 2002)

JES: Jython Environment for Students



JES

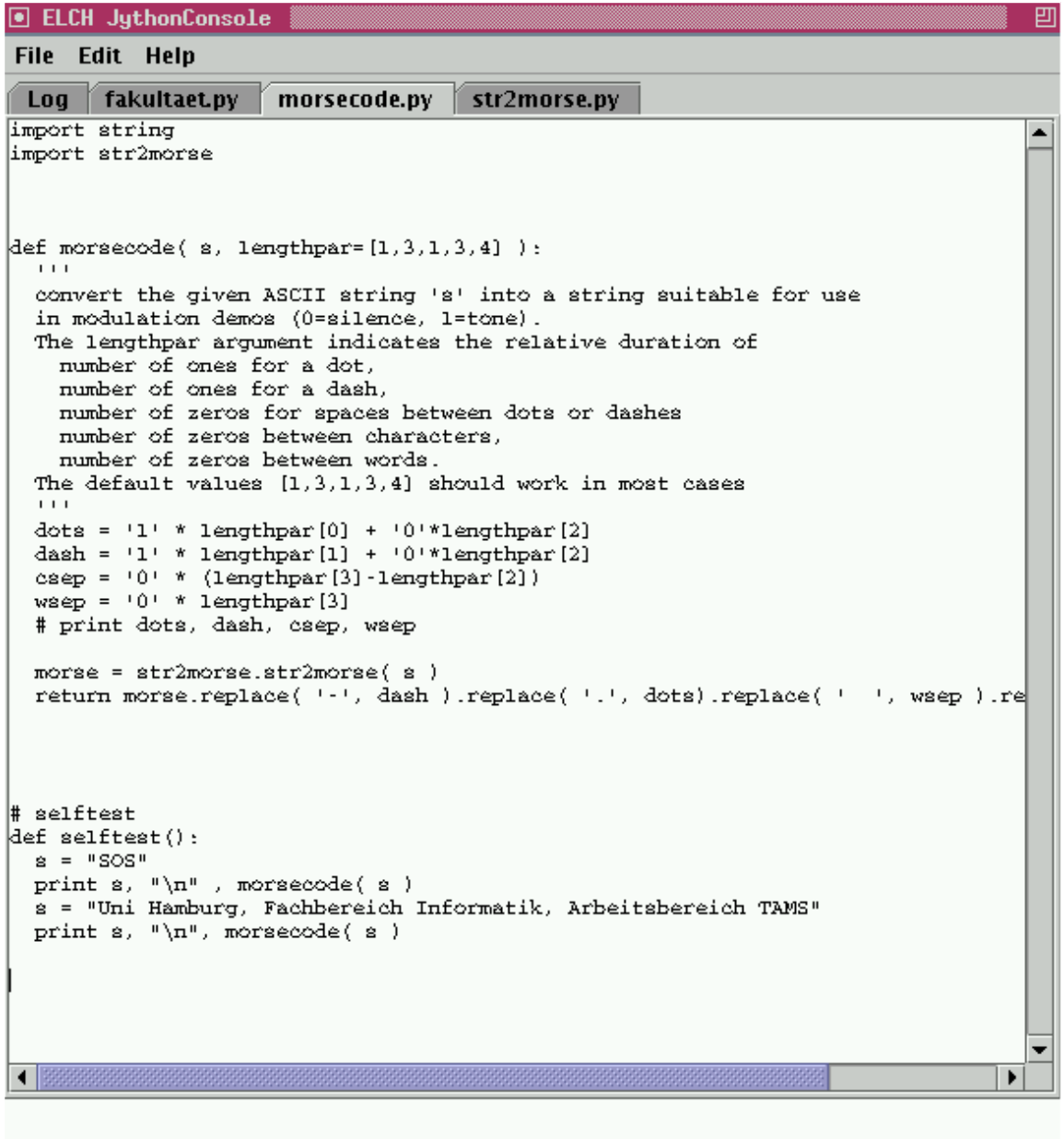


- Editor
- Interpreter
- Debugger

- Sound (wav)
- Pictures,
- Videos (Quicktime)

- Demo

JythonConsole



JythonConsole:

- Jython-Interpreter
- Swing-GUI

- Editor-Panels
- Log-Fenster
- Eingabezeile

- Ausführen des jeweils markierten Textes

- Demo

XHTML-Browser

Darstellung der Jython-Skripte?!

- JythonConsole: bisher nur reiner ASCII-Text
- HTML wäre nett, aber normale Browser sind nicht erweiterbar

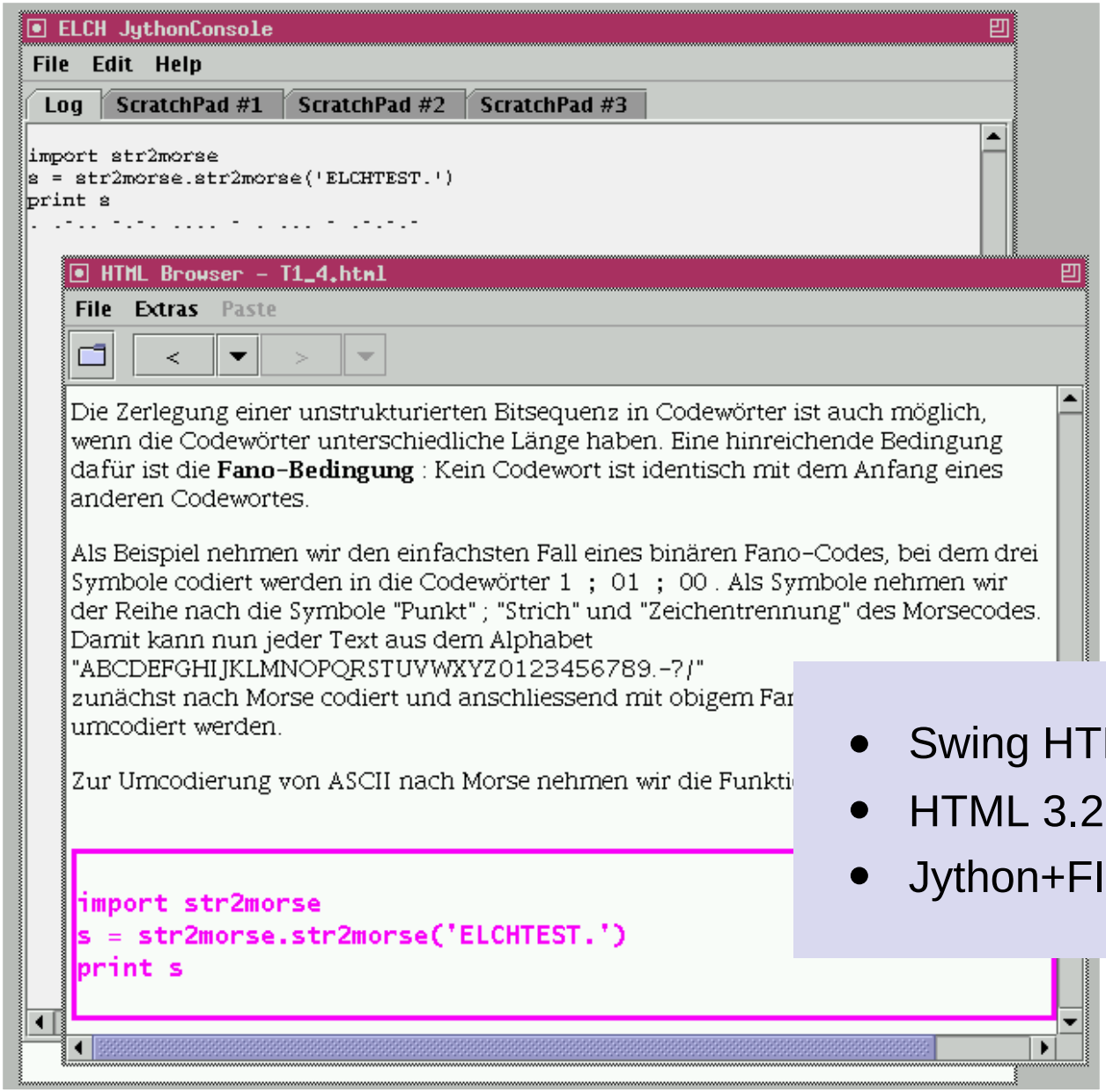
=> eigener, XML/XHTML-basierter Browser

- auf Grundlage von javax.swing.text.html
- erstaunlich brauchbarer HTML 3.2 Browser
- Erweiterungen über neue HTML-Tags integrierbar:

```
<jython>for i in range(1,10): print i</jython>
<tex>$e^{i\pi} + 1 = 0$</tex>
```

- Diplomarbeit Andreas Ruge
- Demo

XHTML-Browser



- Swing HTMLEditorKit
- HTML 3.2
- Jython+FIG+TeX

Status

- Analyse und Klassifikation der "T-Aufgaben"
- Konzepte und Algorithmen für die wichtigsten Aufgabentypen
- Dokumentenformate: ASCII, Matlab, XML-Metadaten
- bisher nur partiell implementiert / noch kein "Feldtest"

- Evaluation geeigneter Plattformen: Matlab, Java/Jython
- leider keine geeigneten Java-Bibliotheken für Numerik/Plots

- Java-Matlab Schnittstelle für jfig / Hades / ImageViewer
- JythonConsole, JythonApplet, Utilities, einzelne Skripte
- diverse Demos (Yield, SRAM, Hades-Applet Webseite)
- Java-basierter Browser: XHTML + Jython + Erweiterungen

TODO:

- weitere Parser/Algorithmen implementieren
- Skripte für die Übungsaufgaben erstellen
- (Matlab / Java / Jython / HTML)

- Einsatz in T1/T2 ab WS 2004/2005
- Übungsaufgaben geeignet formulieren
- erfordert robuste (XML-) Infrastruktur
- Feedback durch Studenten und Übungsgruppenleiter

- eigenes Projekt für "intelligenten Tutor" im T3-Praktikum
- vermutlich ab Februar 2004

- Ideen und Hinweise bitte an mich!

Diskussion

- Wünsche, Hinweise, Anregungen ?
- Welche Komponenten fehlen ?

- Interesse an interaktiven Skripten ?
- Datenformate / Repräsentation von Übungen ?
- andere interaktive, freie Frameworks ?
- Nachhaltigkeit ?

Backup-Folien

diverses...

(Nachhaltigkeit, Matlab, Jython, Octave)