HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Eine Toolbox zur Visualisierung von Algorithmen der theoretischen Informatik

Isabel Wingen

Institut für Informatik
Heinrich-Heine-Universität Düsseldorf

18.05.2017

2017-05-17

Eine Toolbox zur Visualisierung von Algorithmen der theoretischen Informatik

Gliederung

Gliederung
1 Einleitung
2 Stand Vorher
3 Änderungen
Benutzerebene
Build-Prozess
Entwicklung
Bugs und Testen
4 Live-Demo
5 Fazit

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Einleitung

- Weiterführung der Bachelorarbeit von Fabian Ruhland

1. Weiterführung der Arbeit von Fabian Ruhland
2. in Java geschrieben

Einleitung
- Weiterführung der Bachelorarbeit von Fabian Ruhland

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Einleitung

- Weiterführung der Bachelorarbeit von Fabian Ruhland
- Toolbox zur Visualisierung von Automaten und Grammatiken

1. Weiterführung der Arbeit von Fabian Ruhland
2. in Java geschrieben
3. Toolbox zur Visualisierung von Automaten und Grammatiken

Einleitung
- Weiterführung der Bachelorarbeit von Fabian Ruhland
- Toolbox zur Visualisierung von Automaten und Grammatiken

Einleitung

- Weiterführung der Bachelorarbeit von Fabian Ruhland
- Toolbox zur Visualisierung von Automaten und Grammatiken
- Fokus auf Automaten und Compilerbau

Eine Toolbox zur Visualisierung von Algorithmen der theoretischen Informatik
└Einleitung
2017-05-17
└Einleitung

1. Weiterführung der Arbeit von Fabian Ruhland
2. in Java geschrieben
3. Toolbox zur Visualisierung von Automaten und Grammatiken
4. Fokus: Automaten und Compilerbau

Einleitung
- Weiterführung der Bachelorarbeit von Fabian Ruhland
- Toolbox zur Visualisierung von Automaten und Grammatiken
- Fokus auf Automaten und Compilerbau

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Einleitung

- Weiterführung der Bachelorarbeit von Fabian Ruhland
- Toolbox zur Visualisierung von Automaten und Grammatiken
- Fokus auf Automaten und Compilerbau
- Weiterentwicklung des Programmes

1. Weiterführung der Arbeit von Fabian Ruhland
2. in Java geschrieben
3. Toolbox zur Visualisierung von Automaten und Grammatiken
4. Fokus: Automaten und Compilerbau
5. Weiterentwicklung

Einleitung
- Weiterführung der Bachelorarbeit von Fabian Ruhland
- Toolbox zur Visualisierung von Automaten und Grammatiken
- Fokus auf Automaten und Compilerbau
- Weiterentwicklung des Programmes

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Einleitung

- Weiterführung der Bachelorarbeit von Fabian Ruhland
- Toolbox zur Visualisierung von Automaten und Grammatiken
- Fokus auf Automaten und Compilerbau
- Weiterentwicklung des Programmes
- sowohl GUI als auch Konsole

1. Weiterführung der Arbeit von Fabian Ruhland
2. in Java geschrieben
3. Toolbox zur Visualisierung von Automaten und Grammatiken
4. Fokus: Automaten und Compilerbau
5. Weiterentwicklung
6. GUI und Konsole
7. ENDE

Einleitung
- Weiterführung der Bachelorarbeit von Fabian Ruhland
- Toolbox zur Visualisierung von Automaten und Grammatiken
- Fokus auf Automaten und Compilerbau
- Weiterentwicklung des Programmes
- sowohl GUI als auch Konsole

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Motivation

Warum weiterentwicklen?

- Erweiterung des Funktionsumfangs

Motivation
Warum weiterentwicklen?
- Erweiterung des Funktionsumfangs

Warum weiterentwickeln?

1. Funktionsumfang erweiteren

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Motivation

Warum weiterentwicklen?

- Erweiterung des Funktionsumfangs
  - Algorithmen für Grammatiken

Motivation
Warum weiterentwicklen?
- Erweiterung des Funktionsumfangs
- Algorithmen für Grammatiken

Warum weiterentwicklen?

1. Funktionsumfang erweiteren
2. Mehr Algorithmen für Grammatiken
3. Chomsky-Normalform oder CYK

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Motivation

Warum weiterentwicklen?

- Erweiterung des Funktionsumfangs
  - Algorithmen für Grammatiken
  - Kommentierte Ausführung & LaTeX-Generierung

Warum weiterentwickeln?

1. Funktionsumfang erweiteren
2. Mehr Algorithmen für Grammatiken
3. Chomsky-Normalform oder CYK
4. Kommentierte Ausführung der Algorithmen
5. Latex

Motivation
Warum weiterentwicklen?
- Erweiterung des Funktionsumfangs
  - Algorithmen für Grammatiken
  - Kommentierte Ausführung & LaTeX-Generierung

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Motivation

Warum weiterentwicklen?

- Erweiterung des Funktionsumfangs
  - Algorithmen für Grammatiken
  - Kommentierte Ausführung & LaTeX-Generierung
- Verbesserung der Benutzerfreundlichkeit

Motivation
Warum weiterentwicklen?
- Erweiterung des Funktionsumfangs
  - Algorithmen für Grammatiken
  - Kommentierte Ausführung & LaTeX-Generierung
- Verbesserung der Benutzerfreundlichkeit

Warum weiterentwickeln?

1. Funktionsumfang erweiteren
2. Mehr Algorithmen für Grammatiken
3. Chomsky-Normalform oder CYK
4. Kommentierte Ausführung der Algorithmen
5. Latex
6. Verbesserung Benutzerfreundlichkeit

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Motivation

Warum weiterentwicklen?

- Erweiterung des Funktionsumfangs
  - Algorithmen für Grammatiken
  - Kommentierte Ausführung & LaTeX-Generierung
- Verbesserung der Benutzerfreundlichkeit
  - Änderung rückgängig machen

Motivation
Warum weiterentwicklen?
- Erweiterung des Funktionsumfangs
  - Algorithmen für Grammatiken
  - Kommentierte Ausführung & LaTeX-Generierung
- Verbesserung der Benutzerfreundlichkeit
  - Änderung rückgängig machen

Warum weiterentwickeln?

1. Funktionsumfang erweiteren
2. Mehr Algorithmen für Grammatiken
3. Chomsky-Normalform oder CYK
4. Kommentierte Ausführung der Algorithmen
5. Latex
6. Verbesserung Benutzerfreundlichkeit
7. Änderungen rückgängig

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Motivation

Warum weiterentwicklen?

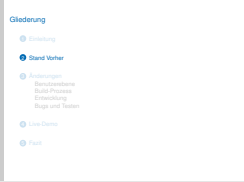
- Erweiterung des Funktionsumfangs
  - Algorithmen für Grammatiken
  - Kommentierte Ausführung & LaTeX-Generierung
- Verbesserung der Benutzerfreundlichkeit
  - Änderung rückgängig machen
  - komfortablere Benutzeroberfläche

Motivation
Warum weiterentwicklen?
- Erweiterung des Funktionsumfangs
  - Algorithmen für Grammatiken
  - Kommentierte Ausführung & LaTeX-Generierung
- Verbesserung der Benutzerfreundlichkeit
  - Änderung rückgängig machen
  - komfortablere Benutzeroberfläche

Warum weiterentwickeln?

1. Funktionsumfang erweiteren
2. Mehr Algorithmen für Grammatiken
3. Chomsky-Normalform oder CYK
4. Kommentierte Ausführung der Algorithmen
5. Latex
6. Verbesserung Benutzerfreundlichkeit
7. Änderungen rückgängig
8. GUI Benutzer komfortabler

Gliederung



HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Datentypen

- (kontextfreie) Grammatiken

Datentypen
- (kontextfreie) Grammatiken

Datentypen der Toolbox

1. kontextfreie Grammatiken
2. endliche Automaten, sowohl deterministisch als auch nicht-deterministisch
3. Aufzählung aller Symbole, dann Regeln, Grund für kontextfrei
4. SableCC ist ein Parser-Generator

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Datentypen

- (kontextfreie) Grammatiken
- (endliche) Automaten

Datentypen
- (kontextfreie) Grammatiken
- (endliche) Automaten

Datentypen der Toolbox

1. kontextfreie Grammatiken
2. endliche Automaten, sowohl deterministisch als auch nicht-deterministisch
3. Aufzählung aller Symbole, dann Regeln, Grund für kontextfrei
4. SableCC ist ein Parser-Generator

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Datentypen

- (kontextfreie) Grammatiken
- (endliche) Automaten
- Eigenes Speicherformat

Datentypen
- (kontextfreie) Grammatiken
- (endliche) Automaten
- Eigenes Speicherformat

Datentypen der Toolbox

1. kontextfreie Grammatiken
2. endliche Automaten, sowohl deterministisch als auch nicht-deterministisch
3. Aufzählung aller Symbole, dann Regeln, Grund für kontextfrei
4. SableCC ist ein Parser-Generator

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Datentypen

- (kontextfreie) Grammatiken
- (endliche) Automaten
- Eigenes Speicherformat
- Parser durch SableCC

Datentypen
- (kontextfreie) Grammatiken
- (endliche) Automaten
- Eigenes Speicherformat
- Parser durch SableCC

Datentypen der Toolbox

1. kontextfreie Grammatiken
2. endliche Automaten, sowohl deterministisch als auch nicht-deterministisch
3. Aufzählung aller Symbole, dann Regeln, Grund für kontextfrei
4. SableCC ist ein Parser-Generator

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Datentypen

- Grammatiken

1. Was konnte die Toolbox mit den Daten machen?
2. Grammatiken
3. anzeigen
4. und bearbeiten
5. LL-Parsing-Tabelle, First- und Follow-Set
6. Automaten
7. anzeigen
8. und bearbeiten
9. minimalisieren
10. epsilon-Übergänge entfernen
11. Vervollständigen

Datentypen
- Grammatiken

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Datentypen

- Grammatiken
  - Anzeigen

Datentypen
- Grammatiken
  - Anzeigen

1. Was konnte die Toolbox mit den Daten machen?
2. Grammatiken
3. anzeigen
4. und bearbeiten
5. LL-Parsing-Tabelle, First- und Follow-Set
6. Automaten
7. anzeigen
8. und bearbeiten
9. minimalisieren
10. epsilon-Übergänge entfernen
11. Vervollständigen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Datentypen

- Grammatiken
  - Anzeigen
  - Bearbeiten

Datentypen
- Grammatiken
  - Anzeigen
  - Bearbeiten

1. Was konnte die Toolbox mit den Daten machen?
2. Grammatiken
3. anzeigen
4. und bearbeiten
5. LL-Parsing-Tabelle, First- und Follow-Set
6. Automaten
7. anzeigen
8. und bearbeiten
9. minimalisieren
10. epsilon-Übergänge entfernen
11. Vervollständigen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Datentypen

- Grammatiken
  - Anzeigen
  - Bearbeiten
  - LL-Parsing und First- und Follow-Set

Datentypen
- Grammatiken
  - Anzeigen
  - Bearbeiten
  - LL-Parsing und First- und Follow-Set

1. Was konnte die Toolbox mit den Daten machen?
2. Grammatiken
3. anzeigen
4. und bearbeiten
5. LL-Parsing-Tabelle, First- und Follow-Set
6. Automaten
7. anzeigen
8. und bearbeiten
9. minimalisieren
10. epsilon-Übergänge entfernen
11. Vervollständigen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Datentypen

- Grammatiken
  - Anzeigen
  - Bearbeiten
  - LL-Parsing und First- und Follow-Set
- Automaten

Datentypen
- Grammatiken
  - Anzeigen
  - Bearbeiten
  - LL-Parsing und First- und Follow-Set
- Automaten

1. Was konnte die Toolbox mit den Daten machen?
2. Grammatiken
3. anzeigen
4. und bearbeiten
5. LL-Parsing-Tabelle, First- und Follow-Set
6. Automaten
7. anzeigen
8. und bearbeiten
9. minimalisieren
10. epsilon-Übergänge entfernen
11. Vervollständigen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Datentypen

- Grammatiken
  - Anzeigen
  - Bearbeiten
  - LL-Parsing und First- und Follow-Set
- Automaten
  - Anzeigen

Datentypen
- Grammatiken
  - Anzeigen
  - Bearbeiten
  - LL-Parsing und First- und Follow-Set
- Automaten
  - Anzeigen

1. Was konnte die Toolbox mit den Daten machen?
2. Grammatiken
3. anzeigen
4. und bearbeiten
5. LL-Parsing-Tabelle, First- und Follow-Set
6. Automaten
7. anzeigen
8. und bearbeiten
9. minimalisieren
10. epsilon-Übergänge entfernen
11. Vervollständigen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Datentypen

- Grammatiken
  - Anzeigen
  - Bearbeiten
  - LL-Parsing und First- und Follow-Set
- Automaten
  - Anzeigen
  - Bearbeiten

Datentypen
- Grammatiken
  - Anzeigen
  - Bearbeiten
  - LL-Parsing und First- und Follow-Set
- Automaten
  - Anzeigen
  - Bearbeiten

1. Was konnte die Toolbox mit den Daten machen?
2. Grammatiken
3. anzeigen
4. und bearbeiten
5. LL-Parsing-Tabelle, First- und Follow-Set
6. Automaten
7. anzeigen
8. und bearbeiten
9. minimalisieren
10. epsilon-Übergänge entfernen
11. Vervollständigen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Datentypen

- Grammatiken
  - Anzeigen
  - Bearbeiten
  - LL-Parsing und First- und Follow-Set
- Automaten
  - Anzeigen
  - Bearbeiten
  - Minimalisieren

Datentypen
- Grammatiken
  - Anzeigen
  - Bearbeiten
  - LL-Parsing und First- und Follow-Set
- Automaten
  - Anzeigen
  - Bearbeiten
  - Minimalisieren

1. Was konnte die Toolbox mit den Daten machen?
2. Grammatiken
3. anzeigen
4. und bearbeiten
5. LL-Parsing-Tabelle, First- und Follow-Set
6. Automaten
7. anzeigen
8. und bearbeiten
9. minimalisieren
10. epsilon-Übergänge entfernen
11. Vervollständigen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Datentypen

- Grammatiken
  - Anzeigen
  - Bearbeiten
  - LL-Parsing und First- und Follow-Set
- Automaten
  - Anzeigen
  - Bearbeiten
  - Minimalisieren
  - ϵ-Übergänge entfernen

Datentypen
- Grammatiken
  - Anzeigen
  - Bearbeiten
  - LL-Parsing und First- und Follow-Set
- Automaten
  - Anzeigen
  - Bearbeiten
  - Minimalisieren
  - ε-Übergänge entfernen

1. Was konnte die Toolbox mit den Daten machen?
2. Grammatiken
3. anzeigen
4. und bearbeiten
5. LL-Parsing-Tabelle, First- und Follow-Set
6. Automaten
7. anzeigen
8. und bearbeiten
9. minimalisieren
10. epsilon-Übergänge entfernen
11. Vervollständigen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Datentypen

- Grammatiken
  - Anzeigen
  - Bearbeiten
  - LL-Parsing und First- und Follow-Set
- Automaten
  - Anzeigen
  - Bearbeiten
  - Minimalisieren
  - ϵ-Übergänge entfernen
  - Vervollständigen

Datentypen
- Grammatiken
  - Anzeigen
  - Bearbeiten
  - LL-Parsing und First- und Follow-Set
- Automaten
  - Anzeigen
  - Bearbeiten
  - Minimalisieren
  - ε-Übergänge entfernen
  - Vervollständigen

1. Was konnte die Toolbox mit den Daten machen?
2. Grammatiken
3. anzeigen
4. und bearbeiten
5. LL-Parsing-Tabelle, First- und Follow-Set
6. Automaten
7. anzeigen
8. und bearbeiten
9. minimalisieren
10. epsilon-Übergänge entfernen
11. Vervollständigen

GUI

GUI Grammar Modus, Menü wechseln

GUI

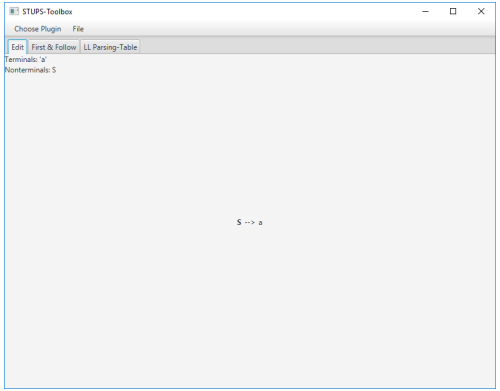


Figure: Die GUI hatte zwei Modi, Grammar und Automaton. Über die Bedienleiste lässt sich zwischen den Modi wechseln.

GUI
Figure: Die GUI hatte zwei Modi, Grammar und Automaton. Über die Bedienleiste lässt sich zwischen den Modi wechseln.

GUI Grammar Modus, Menü wechseln

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Plugin-System

Modularer Aufbau

- Funktionen sind über sog. Plugins eingebunden

1. Funktionen = Plugins

Plugin-System
Modularer Aufbau
• Funktionen sind über sog. Plugins eingebunden

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Plugin-System

Modularer Aufbau

- Funktionen sind über sog. Plugins eingebunden
- ein Plugin ist eine Klasse, die bestimmtes Interface implementiert

1. Funktionen = Plugins
2. Implementiert bestimmtes Interface
3. automatisch einbinden

Plugin-System
Modularer Aufbau
- Funktionen sind über sog. Plugins eingebunden
- ein Plugin ist eine Klasse, die bestimmtes Interface implementiert

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Plugin-System

Modularer Aufbau

- Funktionen sind über sog. Plugins eingebunden
- ein Plugin ist eine Klasse, die bestimmtes Interface implementiert
- in dem Plugin ist ein Input- und ein Output-Typ definiert, auf dem das Plugin arbeitet

Plugin-System
Modularer Aufbau
- Funktionen sind über sog. Plugins eingebunden
- ein Plugin ist eine Klasse, die bestimmtes Interface implementiert
- in dem Plugin ist ein Input- und ein Output-Typ definiert, auf dem das Plugin arbeitet

1. Funktionen = Plugins
2. Implementiert bestimmtes Interface
3. automatisch einbinden
4. hat Input-Typ und Output-Typ, bekommt Instanz Input, Instanz Output zurück

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Plugin-System

Modularer Aufbau

- Funktionen sind über sog. Plugins eingebunden
- ein Plugin ist eine Klasse, die bestimmtes Interface implementiert
- in dem Plugin ist ein Input- und ein Output-Typ definiert, auf dem das Plugin arbeitet
- die Plugins verändern ggf. das Objekt, auf dem sie arbeiten

1. Funktionen = Plugins
2. Implementiert bestimmtes Interface
3. automatisch einbinden
4. hat Input-Typ und Output-Typ, bekommt Instanz Input, Instanz Output zurück
5. Instanz verändern

Plugin-System
Modularer Aufbau
- Funktionen sind über sog. Plugins eingebunden
- ein Plugin ist eine Klasse, die bestimmtes Interface implementiert
- in dem Plugin ist ein Input- und ein Output-Typ definiert, auf dem das Plugin arbeitet
- die Plugins verändern ggf. das Objekt, auf dem sie arbeiten

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Plugin-System

Modularer Aufbau

- Funktionen sind über sog. Plugins eingebunden
- ein Plugin ist eine Klasse, die bestimmtes Interface implementiert
- in dem Plugin ist ein Input- und ein Output-Typ definiert, auf dem das Plugin arbeitet
- die Plugins verändern ggf. das Objekt, auf dem sie arbeiten
- je Typ nur eine Instanz

1. Funktionen = Plugins
2. Implementiert bestimmtes Interface
3. automatisch einbinden
4. hat Input-Typ und Output-Typ, bekommt Instanz Input, Instanz Output zurück
5. Instanz verändern
6. je typ nur eine Instanz, neue Grammatik: alte Speichern, neue Laden

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Plugin-System

Modularer Aufbau

- Funktionen sind über sog. Plugins eingebunden
- ein Plugin ist eine Klasse, die bestimmtes Interface implementiert
- in dem Plugin ist ein Input- und ein Output-Typ definiert, auf dem das Plugin arbeitet
- die Plugins verändern ggf. das Objekt, auf dem sie arbeiten
- je Typ nur eine Instanz
- leichte Weiterentwicklung ohne Einarbeitung in den Legacy-Code möglich

Plugin-System
Modularer Aufbau
- Funktionen sind über sog. Plugins eingebunden
- ein Plugin ist eine Klasse, die bestimmtes Interface implementiert
- in dem Plugin ist ein Input- und ein Output-Typ definiert, auf dem das Plugin arbeitet
- die Plugins verändern ggf. das Objekt, auf dem sie arbeiten
- je Typ nur eine Instanz
- leichte Weiterentwicklung ohne Einarbeitung in den Legacy-Code möglich

1. Funktionen = Plugins
2. Implementiert bestimmtes Interface
3. automatisch einbinden
4. hat Input-Typ und Output-Typ, bekommt Instanz Input, Instanz Output zurück
5. Instanz verändern
6. je typ nur eine Instanz, neue Grammatik: alte Speichern, neue Laden
7. Weiterentwicklung

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Probleme

Aber: Für die Weiterentwicklung ergaben sich einige Probleme.

Probleme Weiterentwicklung
Kurz erläutern, dann weiter

Probleme
Aber: Für die Weiterentwicklung ergaben sich einige Probleme.

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Probleme

Aber: Für die Weiterentwicklung ergaben sich einige Probleme.

- Modifizierbarkeit der Objekte

Probleme Weiterentwicklung
Kurz erläutern, dann weiter

1. Objekte waren modifizierbar

Probleme
Aber: Für die Weiterentwicklung ergaben sich einige Probleme.
- Modifizierbarkeit der Objekte

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Probleme

Aber: Für die Weiterentwicklung ergaben sich einige Probleme.

- Modifizierbarkeit der Objekte
  - Plugins veränderten das Objekt. Keine einfache Speicherung von Vorgängerversionen möglich.

Probleme Weiterentwicklung
Kurz erläutern, dann weiter

1. Objekte waren modifizierbar
2. keine einfache Speicherung, Deep-Copy

Probleme

Aber: Für die Weiterentwicklung ergaben sich einige Probleme.
- Modifizierbarkeit der Objekte
  - Plugins veränderten das Objekt. Keine einfache Speicherung von Vorgängerversionen möglich.

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Probleme

Aber: Für die Weiterentwicklung ergaben sich einige Probleme.

- Modifizierbarkeit der Objekte
  - Plugins veränderten das Objekt. Keine einfache Speicherung von Vorgängerversionen möglich.
- keine Vergleichbarkeit

Probleme Weiterentwicklung
Kurz erläutern, dann weiter

1. Objekte waren modifizierbar
2. keine einfache Speicherung, Deep-Copy
3. keine Vergleichbarkeit

Probleme

Aber: Für die Weiterentwicklung ergaben sich einige Probleme.
- Modifizierbarkeit der Objekte
  - Plugins veränderten das Objekt. Keine einfache Speicherung von Vorgängerversionen möglich.
- keine Vergleichbarkeit

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Probleme

Aber: Für die Weiterentwicklung ergaben sich einige Probleme.

- Modifizierbarkeit der Objekte
  - Plugins veränderten das Objekt. Keine einfache Speicherung von Vorgängerversionen möglich.
- keine Vergleichbarkeit
  - Zwei Nichtterminale mit gleichem Namen waren nicht gleich

Probleme Weiterentwicklung
Kurz erläutern, dann weiter

1. Objekte waren modifizierbar
2. keine einfache Speicherung, Deep-Copy
3. keine Vergleichbarkeit
4. genau diesselbe Referenz

Probleme

Aber: Für die Weiterentwicklung ergaben sich einige Probleme.
- Modifizierbarkeit der Objekte
  - Plugins veränderten das Objekt. Keine einfache Speicherung von Vorgängerversionen möglich.
- keine Vergleichbarkeit
  - Zwei Nichtterminale mit gleichem Namen waren nicht gleich

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Probleme

Aber: Für die Weiterentwicklung ergaben sich einige Probleme.

- Modifizierbarkeit der Objekte
  - Plugins veränderten das Objekt. Keine einfache Speicherung von Vorgängerversionen möglich.
- keine Vergleichbarkeit
  - Zwei Nichtterminale mit gleichem Namen waren nicht gleich
  - bei Algorithmen mussten immer **genau dieselben** Referenzen verwendet werden

Probleme Weiterentwicklung
Kurz erläutern, dann weiter

1. Objekte waren modifizierbar
2. keine einfache Speicherung, Deep-Copy
3. keine Vergleichbarkeit
4. gleicher Name, nicht gleich
5. genau diesselbe Referenz

Probleme

Aber: Für die Weiterentwicklung ergaben sich einige Probleme.
- Modifizierbarkeit der Objekte
  - Plugins veränderten das Objekt. Keine einfache Speicherung von Vorgängerversionen möglich.
- keine Vergleichbarkeit
  - Zwei Nichtterminale mit gleichem Namen waren nicht gleich
  - bei Algorithmen mussten immer **genau dieselben** Referenzen verwendet werden

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

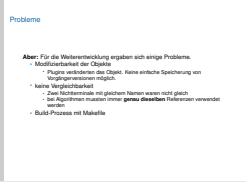
Probleme

Aber: Für die Weiterentwicklung ergaben sich einige Probleme.

- Modifizierbarkeit der Objekte
  - Plugins veränderten das Objekt. Keine einfache Speicherung von Vorgängerversionen möglich.
- keine Vergleichbarkeit
  - Zwei Nichtterminale mit gleichem Namen waren nicht gleich
  - bei Algorithmen mussten immer **genau dieselben** Referenzen verwendet werden
- Build-Prozess mit Makefile

Probleme Weiterentwicklung
Kurz erläutern, dann weiter

1. Objekte waren modifizierbar
2. keine einfache Speicherung, Deep-Copy
3. keine Vergleichbarkeit
4. gleicher Name, nicht gleich
5. genau diesselbe Referenz
6. MakeFile, Linux-Befehlen



HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Probleme

Aber: Für die Weiterentwicklung ergaben sich einige Probleme.

- Modifizierbarkeit der Objekte
  - Plugins veränderten das Objekt. Keine einfache Speicherung von Vorgängerversionen möglich.
- keine Vergleichbarkeit
  - Zwei Nichtterminale mit gleichem Namen waren nicht gleich
  - bei Algorithmen mussten immer **genau dieselben** Referenzen verwendet werden
- Build-Prozess mit Makefile
  - kein plattformunabhängiger Build-Prozess möglich

Probleme

Aber: Für die Weiterentwicklung ergaben sich einige Probleme.
- Modifizierbarkeit der Objekte
  - Plugins veränderten das Objekt. Keine einfache Speicherung von Vorgängerversionen möglich.
- keine Vergleichbarkeit
  - Zwei Nichtterminale mit gleichem Namen waren nicht gleich
  - bei Algorithmen mussten immer **genau dieselben** Referenzen verwendet werden
- Build-Prozess mit Makefile
  - kein plattformunabhängiger Build-Prozess möglich

Probleme Weiterentwicklung
Kurz erläutern, dann weiter

1. Objekte waren modifizierbar
2. keine einfache Speicherung, Deep-Copy
3. keine Vergleichbarkeit
4. gleicher Name, nicht gleich
5. genau diesselbe Referenz
6. MakeFile, Linux-Befehlen
7. nicht plattformunabhängig, blöd

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Probleme

Aber: Für die Weiterentwicklung ergaben sich einige Probleme.

- Modifizierbarkeit der Objekte
  - Plugins veränderten das Objekt. Keine einfache Speicherung von Vorgängerversionen möglich.
- keine Vergleichbarkeit
  - Zwei Nichtterminale mit gleichem Namen waren nicht gleich
  - bei Algorithmen mussten immer **genau dieselben** Referenzen verwendet werden
- Build-Prozess mit Makefile
  - kein plattformunabhängiger Build-Prozess möglich
  - Erstellung einer JAR nicht möglich

Probleme

Aber: Für die Weiterentwicklung ergaben sich einige Probleme.
- Modifizierbarkeit der Objekte
  - Plugins veränderten das Objekt. Keine einfache Speicherung von Vorgängerversionen möglich.
- keine Vergleichbarkeit
  - Zwei Nichtterminale mit gleichem Namen waren nicht gleich
  - bei Algorithmen mussten immer **genau dieselben** Referenzen verwendet werden
- Build-Prozess mit Makefile
  - kein plattformunabhängiger Build-Prozess möglich
  - Erstellung einer JAR nicht möglich

Probleme Weiterentwicklung
Kurz erläutern, dann weiter

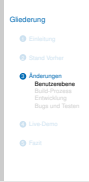
1. Objekte waren modifizierbar
2. keine einfache Speicherung, Deep-Copy
3. keine Vergleichbarkeit
4. gleicher Name, nicht gleich
5. genau diesselbe Referenz
6. MakeFile, Linux-Befehlen
7. nicht plattformunabhängig, blöd
8. keine Jar, externe Bibliotheken

Gliederung

Gliederung
1 Einleitung
2 Stand Vorher
3 Änderungen
Benutzerebene
Build-Prozess
Entwicklung
Bugs und Testen
4 Live-Demo
5 Fazit

Änderungen

Gliederung



HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Benutzerebene

GUI

- Gestaltung durch CSS-Stylesheet

Benutzerebene
GUI
- Gestaltung durch CSS-Stylesheet

zunächst zur GUI

1. css-Stylesheet

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Benutzerebene

GUI

- Gestaltung durch CSS-Stylesheet
- Design an bekannten IDEs orientiert.

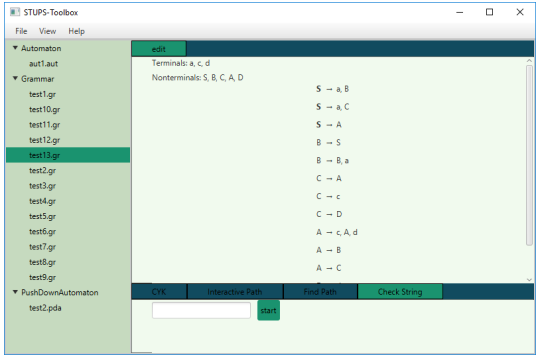


Figure: die GUI in der Toolbox 2.0

2017-05-17

Eine Toolbox zur Visualisierung von Algorithmen der theoretischen Informatik
└Änderungen
└Benutzerebene
└Benutzerebene

zunächst zur GUI

1. css-Stylesheet
2. GUI m Standard-Farbschema

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Benutzerebene

GUI

- Gestaltung durch CSS-Stylesheet
- Design an bekannten IDEs orientiert.

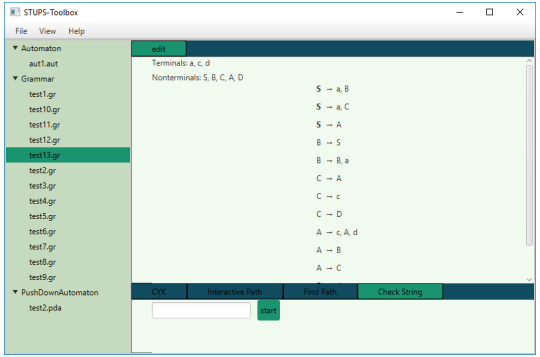


Figure: die GUI in der Toolbox 2.0

2017-05-17

Eine Toolbox zur Visualisierung von Algorithmen der theoretischen Informatik
└Änderungen
└Benutzerebene
└Benutzerebene

zunächst zur GUI

1. css-Stylesheet
2. GUI m Standard-Farbschema
3. IDES. Ei-Di-Is, Links Baum mit Objekten, rechts Bearbeiten, unten Funktionalität

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Benutzerebene

GUI

- Gestaltung durch CSS-Stylesheet
- Design an bekannten IDEs orientiert.
- Unicode-Zeichen Darstellung

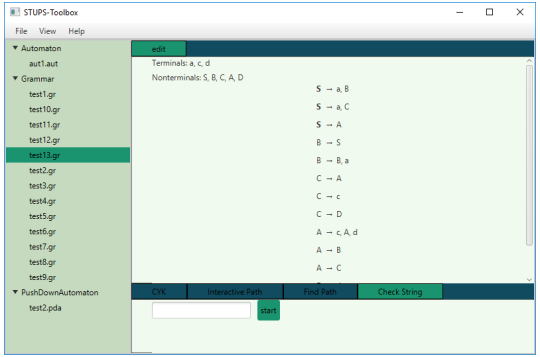


Figure: die GUI in der Toolbox 2.0

zunächst zur GUI

1. css-Stylesheet
2. GUI m Standard-Farbschema
3. IDES. Ei-Di-Is, Links Baum mit Objekten, rechts Bearbeiten, unten Funktionalität
4. Sonderzeichen Unicode

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Benutzerebene

Neu hinzugefügte Features

- dokumentierte Durchführung der Algorithmen → auch in LaTeX

neu hingefügte Features

1. dokumentierte Durchführung der Algorithmen Latex, Lerneffekt, Skripte

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Benutzerebene

Neu hinzugefügte Features

- dokumentierte Durchführung der Algorithmen → auch in LaTeX
- paralleles Arbeiten auf mehreren Objekten

neu hingefügte Features

1. dokumentierte Durchführung der Algorithmen Latex, Lerneffekt, Skripte
2. parallel, mehrere Objekte. Vorher: eins

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Benutzerebene

Neu hinzugefügte Features

- dokumentierte Durchführung der Algorithmen → auch in LaTeX
- paralleles Arbeiten auf mehreren Objekten
- Rückgängig machen

Benutzerebene
Neu hinzugefügte Features
- dokumentierte Durchführung der Algorithmen → auch in LaTeX
- paralleles Arbeiten auf mehreren Objekten
- Rückgängig machen

neu hingefügte Features

1. dokumentierte Durchführung der Algorithmen Latex, Lerneffekt, Skripte
2. parallel, mehrere Objekte. Vorher: eins
3. Änderungen rückgängig

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Benutzerebene

Neu hinzugefügte Features

- dokumentierte Durchführung der Algorithmen → auch in LaTeX
- paralleles Arbeiten auf mehreren Objekten
- Rückgängig machen
- Automatisches Speichern der Änderung beim Beenden

Benutzerebene
Neu hinzugefügte Features
- dokumentierte Durchführung der Algorithmen → auch in LaTeX
- paralleles Arbeiten auf mehreren Objekten
- Rückgängig machen
- Automatisches Speichern der Änderung beim Beenden

neu hingefügte Features

1. dokumentierte Durchführung der Algorithmen Latex, Lerneffekt, Skripte
2. parallel, mehrere Objekte. Vorher: eins
3. Änderungen rückgängig
4. automatisch Speichern Beenden. vorher: manuell

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Benutzerebene

Neu hinzugefügte Features

- dokumentierte Durchführung der Algorithmen → auch in LaTeX
- paralleles Arbeiten auf mehreren Objekten
- Rückgängig machen
- Automatisches Speichern der Änderung beim Beenden
- Kellerautomaten als neuer Datentyp

Benutzerebene
Neu hinzugefügte Features
- dokumentierte Durchführung der Algorithmen → auch in LaTeX
- paralleles Arbeiten auf mehreren Objekten
- Rückgängig machen
- Automatisches Speichern der Änderung beim Beenden
- Kellerautomaten als neuer Datentyp

neu hingefügte Features

1. dokumentierte Durchführung der Algorithmen Latex, Lerneffekt, Skripte
2. parallel, mehrere Objekte. Vorher: eins
3. Änderungen rückgängig
4. automatisch Speichern Beenden. vorher: manuell
5. Kelleratuomaten neu

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Benutzerebene

Algorithmen

- Kellerautomaten

welche Algorithmen wurden implementiert?

1. Zu Kellerautomaten:

Benutzerebene
Algorithmen
- Kellerautomaten

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Benutzerebene

Algorithmen

- Kellerautomaten
  - Umwandeln zu Grammatik

Benutzerebene
Algorithmen
- Kellerautomaten
  - Umwandeln zu Grammatik

welche Algorithmen wurden implementiert?

1. Zu Kellerautomaten:
2. Kellerautomat zu Grammatik

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Benutzerebene

Algorithmen

- Kellerautomaten
  - Umwandeln zu Grammatik
  - Schritt-für-Schritt-Durchlauf

Benutzerebene
Algorithmen
- Kellerautomaten
  - Umwandeln zu Grammatik
  - Schritt-für-Schritt-Durchlauf

welche Algorithmen wurden implementiert?

1. Zu Kellerautomaten:
2. Kellerautomat zu Grammatik
3. Schritt-für-Schritt Durchlauf

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Benutzerebene

Algorithmen

- Kellerautomaten
  - Umwandeln zu Grammatik
  - Schritt-für-Schritt-Durchlauf
- Grammatik

Benutzerebene
Algorithmen
- Kellerautomaten
  - Umwandeln zu Grammatik
  - Schritt-für-Schritt-Durchlauf
- Grammatik

welche Algorithmen wurden implementiert?

1. Zu Kellerautomaten:
2. Kellerautomat zu Grammatik
3. Schritt-für-Schritt Durchlauf
4. Zusätzlich Grammatik

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Benutzerebene

Algorithmen

- Kellerautomaten
  - Umwandeln zu Grammatik
  - Schritt-für-Schritt-Durchlauf
- Grammatik
  - CYK

Benutzerebene
Algorithmen
- Kellerautomaten
  - Umwandeln zu Grammatik
  - Schritt-für-Schritt-Durchlauf
- Grammatik
  - CYK

welche Algorithmen wurden implementiert?

1. Zu Kellerautomaten:
2. Kellerautomat zu Grammatik
3. Schritt-für-Schritt Durchlauf
4. Zusätzlich Grammatik
5. CYK. Wortproblem, kubisch

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Benutzerebene

Algorithmen

- Kellerautomaten
  - Umwandeln zu Grammatik
  - Schritt-für-Schritt-Durchlauf
- Grammatik
  - CYK
  - Umwandeln in Chomsky-Normal-Form

Benutzerebene
Algorithmen
- Kellerautomaten
  - Umwandeln zu Grammatik
  - Schritt-für-Schritt-Durchlauf
- Grammatik
  - CYK
  - Umwandeln in Chomsky-Normal-Form

welche Algorithmen wurden implementiert?

1. Zu Kellerautomaten:
2. Kellerautomat zu Grammatik
3. Schritt-für-Schritt Durchlauf
4. Zusätzlich Grammatik
5. CYK. Wortproblem, kubisch
6. Chomsky-Normal-Form

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Benutzerebene

Algorithmen

- Kellerautomaten
  - Umwandeln zu Grammatik
  - Schritt-für-Schritt-Durchlauf
- Grammatik
  - CYK
  - Umwandeln in Chomsky-Normal-Form
  - Entfernen von λ-Regeln

Benutzerebene
Algorithmen
- Kellerautomaten
  - Umwandeln zu Grammatik
  - Schritt-für-Schritt-Durchlauf
- Grammatik
  - CYK
  - Umwandeln in Chomsky-Normal-Form
  - Entfernen von λ-Regeln

welche Algorithmen wurden implementiert?

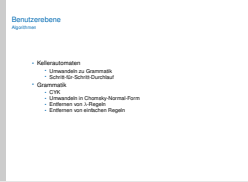
1. Zu Kellerautomaten:
2. Kellerautomat zu Grammatik
3. Schritt-für-Schritt Durchlauf
4. Zusätzlich Grammatik
5. CYK. Wortproblem, kubisch
6. Chomsky-Normal-Form
7. Entfernen von lambda-Regeln

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Benutzerebene

Algorithmen

- Kellerautomaten
  - Umwandeln zu Grammatik
  - Schritt-für-Schritt-Durchlauf
- Grammatik
  - CYK
  - Umwandeln in Chomsky-Normal-Form
  - Entfernen von λ-Regeln
  - Entfernen von einfachen Regeln



welche Algorithmen wurden implementiert?

1. Zu Kellerautomaten:
2. Kellerautomat zu Grammatik
3. Schritt-für-Schritt Durchlauf
4. Zusätzlich Grammatik
5. CYK. Wortproblem, kubisch
6. Chomsky-Normal-Form
7. Entfernen von lambda-Regeln
8. Entfernen von einfachen Regeln

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Benutzerebene

Algorithmen

- Kellerautomaten
  - Umwandeln zu Grammatik
  - Schritt-für-Schritt-Durchlauf
- Grammatik
  - CYK
  - Umwandeln in Chomsky-Normal-Form
  - Entfernen von λ-Regeln
  - Entfernen von einfachen Regeln
  - Umwandeln in Kellerautomat

Benutzerebene
Algorithmen
- Kellerautomaten
  - Umwandeln zu Grammatik
  - Schritt-für-Schritt-Durchlauf
- Grammatik
  - CYK
  - Umwandeln in Chomsky-Normal-Form
  - Entfernen von λ-Regeln
  - Entfernen von einfachen Regeln
  - Umwandeln in Kellerautomat

welche Algorithmen wurden implementiert?

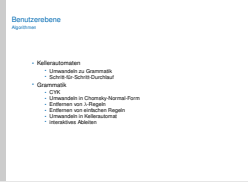
1. Zu Kellerautomaten:
2. Kellerautomat zu Grammatik
3. Schritt-für-Schritt Durchlauf
4. Zusätzlich Grammatik
5. CYK. Wortproblem, kubisch
6. Chomsky-Normal-Form
7. Entfernen von lambda-Regeln
8. Entfernen von einfachen Regeln
9. Umwandeln zu Kellerautomat

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Benutzerebene

Algorithmen

- Kellerautomaten
  - Umwandeln zu Grammatik
  - Schritt-für-Schritt-Durchlauf
- Grammatik
  - CYK
  - Umwandeln in Chomsky-Normal-Form
  - Entfernen von λ-Regeln
  - Entfernen von einfachen Regeln
  - Umwandeln in Kellerautomat
  - interaktives Ableiten



welche Algorithmen wurden implementiert?

1. Zu Kellerautomaten:
2. Kellerautomat zu Grammatik
3. Schritt-für-Schritt Durchlauf
4. Zusätzlich Grammatik
5. CYK. Wortproblem, kubisch
6. Chomsky-Normal-Form
7. Entfernen von lambda-Regeln
8. Entfernen von einfachen Regeln
9. Umwandeln zu Kellerautomat
10. interaktives Ableiten

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Benutzerebene

Algorithmen

- Kellerautomaten
  - Umwandeln zu Grammatik
  - Schritt-für-Schritt-Durchlauf
- Grammatik
  - CYK
  - Umwandeln in Chomsky-Normal-Form
  - Entfernen von λ-Regeln
  - Entfernen von einfachen Regeln
  - Umwandeln in Kellerautomat
  - interaktives Ableiten
  - automatisches Ableiten

Benutzerebene
Algorithmen
- Kellerautomaten
  - Umwandeln zu Grammatik
  - Schritt-für-Schritt-Durchlauf
- Grammatik
  - CYK
  - Umwandeln in Chomsky-Normal-Form
  - Entfernen von λ-Regeln
  - Entfernen von einfachen Regeln
  - Umwandeln in Kellerautomat
  - interaktives Ableiten
  - automatisches Ableiten

welche Algorithmen wurden implementiert?

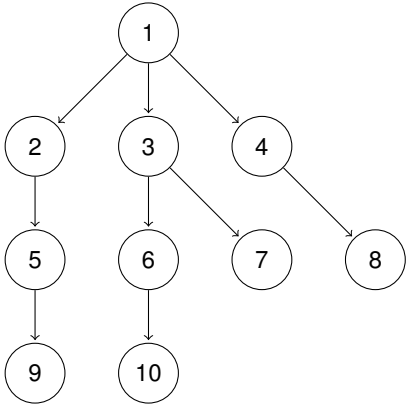
1. Zu Kellerautomaten:
2. Kellerautomat zu Grammatik
3. Schritt-für-Schritt Durchlauf
4. Zusätzlich Grammatik
5. CYK. Wortproblem, kubisch
6. Chomsky-Normal-Form
7. Entfernen von lambda-Regeln
8. Entfernen von einfachen Regeln
9. Umwandeln zu Kellerautomat
10. interaktives Ableiten
11. automatisches Suchen

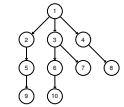
HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Breitensuche

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Breitensuche





HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Umsetzung der Algorithmen

- Umsetzung oberflächlich genau wie im Info IV-Skript

1. oberflächlich Info 4 Skript

Umsetzung der Algorithmen
- Umsetzung oberflächlich genau wie im Info IV-Skript

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Umsetzung der Algorithmen

- Umsetzung oberflächlich genau wie im Info IV-Skript
- aber: Anpassung für Computer notwendig

Umsetzung der Algorithmen
- Umsetzung oberflächlich genau wie im Info IV-Skript
- aber: Anpassung für Computer notwendig

1. oberflächlich Info 4 Skript
2. Anpassung Computer

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Umsetzung der Algorithmen

- Umsetzung oberflächlich genau wie im Info IV-Skript
- aber: Anpassung für Computer notwendig
- Beispiel: Finde alle Zykel $B_1 \to B_2, \ldots, B_n \to B_1$

1. oberflächlich Info 4 Skript
2. Anpassung Computer
3. Beispiel: Finden von Zykeln

Umsetzung der Algorithmen
- Umsetzung oberflächlich genau wie im Info IV-Skript
- aber: Anpassung für Computer notwendig
- Beispiel: Finde alle Zykel $B_1 \to B_2, \ldots, B_n \to B_1$

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Umsetzung der Algorithmen

- Umsetzung oberflächlich genau wie im Info IV-Skript
- aber: Anpassung für Computer notwendig
- Beispiel: Finde alle Zykel $B_1 \to B_2, \ldots, B_n \to B_1$
- für Menschen auf einen Blick lösbar

1. oberflächlich Info 4 Skript
2. Anpassung Computer
3. Beispiel: Finden von Zykeln
4. Für Menschen klar

Umsetzung der Algorithmen
- Umsetzung oberflächlich genau wie im Info IV-Skript
- aber: Anpassung für Computer notwendig
- Beispiel: Finde alle Zykel $B_1 \to B_2, \ldots, B_n \to B_1$
- für Menschen auf einen Blick lösbar

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Umsetzung der Algorithmen

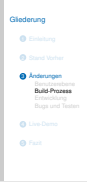
- Umsetzung oberflächlich genau wie im Info IV-Skript
- aber: Anpassung für Computer notwendig
- Beispiel: Finde alle Zykel $B_1 \to B_2, \ldots, B_n \to B_1$
- für Menschen auf einen Blick lösbar
- Computer muss Tiefensuche laufen lassen und Rückwärtskanten bestimmen

1. oberflächlich Info 4 Skript
2. Anpassung Computer
3. Beispiel: Finden von Zykeln
4. Für Menschen klar
5. Computer Tiefensuche

Umsetzung der Algorithmen

- Umsetzung oberflächlich genau wie im Info IV-Skript
- aber: Anpassung für Computer notwendig
- Beispiel: Finde alle Zykel $B_1 \to B_2, \ldots, B_n \to B_1$
- für Menschen auf einen Blick lösbar
- Computer muss Tiefensuche laufen lassen und Rückwärtskanten bestimmen

Gliederung



HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Build-Prozess

- Umstieg von Makefile auf Gradle

1. BuildProzess
2. makeFile Gradle

Build-Prozess
- Umstieg von Makefile auf Gradle

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Build-Prozess

- Umstieg von Makefile auf Gradle
- kleines Workaround wegen SableCC nötig

Build-Prozess
- Umstieg von Makefile auf Gradle
- kleines Workaround wegen SableCC nötig

1. BuildProzess
2. makeFile Gradle
3. Workaround Sablecc

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Build-Prozess

- Umstieg von Makefile auf Gradle
- kleines Workaround wegen SableCC nötig
  - SableCC mischt Source und Resource

1. BuildProzess
2. makeFile Gradle
3. Workaround Sablecc
4. SableCC Source Resource, selber Ordner

Build-Prozess
- Umstieg von Makefile auf Gradle
- kleines Workaround wegen SableCC nötig
  - SableCC mischt Source und Resource

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Build-Prozess

- Umstieg von Makefile auf Gradle
- kleines Workaround wegen SableCC nötig
  - SableCC mischt Source und Resource
  - Probleme mit Gradle und *getResource()*

1. BuildProzess
2. makeFile Gradle
3. Workaround Sablecc
4. SableCC Source Resource, selber Ordner
5. Probleme getResource()

Build-Prozess
- Umstieg von Makefile auf Gradle
- kleines Workaround wegen SableCC nötig
  - SableCC mischt Source und Resource
  - Probleme mit Gradle und getResource()

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Build-Prozess

- Umstieg von Makefile auf Gradle
- kleines Workaround wegen SableCC nötig
  - SableCC mischt Source und Resource
  - Probleme mit Gradle und *getResource()*
  - Lösung: Gradle-Skript, welches Dateien verschiebt und einige Zeilen im Code anpasst

1. BuildProzess
2. makeFile Gradle
3. Workaround Sablecc
4. SableCC Source Resource, selber Ordner
5. Probleme getResource()
6. Gradle-Skript, passt an

Build-Prozess
- Umstieg von Makefile auf Gradle
- kleines Workaround wegen SableCC nötig
  - SableCC mischt Source und Resource
  - Probleme mit Gradle und getResource()
  - Lösung: Gradle-Skript, welches Dateien verschiebt und einige Zeilen im Code anpasst

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Build-Prozess

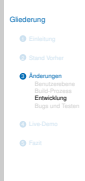
- Umstieg von Makefile auf Gradle
- kleines Workaround wegen SableCC nötig
  - SableCC mischt Source und Resource
  - Probleme mit Gradle und *getResource()*
  - Lösung: Gradle-Skript, welches Dateien verschiebt und einige Zeilen im Code anpasst
- ausführbare Jar kann nun per Plugin erstellt werden

Build-Prozess

- Umstieg von Makefile auf Gradle
- kleines Workaround wegen SableCC nötig
  - SableCC mischt Source und Resource
  - Probleme mit Gradle und getResource()
  - Lösung: Gradle-Skript, welches Dateien verschiebt und einige Zeilen im Code anpasst
- ausführbare Jar kann nun per Plugin erstellt werden

1. BuildProzess
2. makeFile Gradle
3. Workaround Sablecc
4. SableCC Source Resource, selber Ordner
5. Probleme getResource()
6. Gradle-Skript, passt an
7. JAR per Plugin, mit Dependencies

Gliederung



HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung

Unmodifizierbarkeit I

- Wunsch, vollführte Änderungen rückgängig zu machen

1. Als nächstes: Änderungen auf Entwickler-Ebene
2. Änderungen rückängig

Entwicklung
Unmodifizierbarkeit I
- Wunsch, vollführte Änderungen rückgängig zu machen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung

Unmodifizierbarkeit I

- Wunsch, vollführte Änderungen rückgängig zu machen
- hierfür Referenz auf Vorgängerversion

1. Als nächstes: Änderungen auf Entwickler-Ebene
2. Änderungen rückgängig
3. Referenz Vorgänger

Entwicklung
Unmodifizierbarkeit I
- Wunsch, vollführte Änderungen rückgängig zu machen
- hierfür Referenz auf Vorgängerversion

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung

Unmodifizierbarkeit I

- Wunsch, vollführte Änderungen rückgängig zu machen
- hierfür Referenz auf Vorgängerversion
- Problem: Plugins verändern das Objekt

1. Als nächstes: Änderungen auf Entwickler-Ebene
2. Änderungen rückgängig
3. Referenz Vorgänger
4. Plugins verändern Objekt

Entwicklung
Unmodifizierbarkeit I
- Wunsch, vollführte Änderungen rückgängig zu machen
- hierfür Referenz auf Vorgängerversion
- Problem: Plugins verändern das Objekt

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung

Unmodifizierbarkeit I

- Wunsch, vollführte Änderungen rückgängig zu machen
- hierfür Referenz auf Vorgängerversion
- Problem: Plugins verändern das Objekt
- Lösung: Unmodifizierbarkeit

1. Als nächstes: Änderungen auf Entwickler-Ebene
2. Änderungen rückängig
3. Referenz Vorgänger
4. Plugins verändern Objekt
5. Lösung: unmodifizierbarkeit

Entwicklung
Unmodifizierbarkeit I
- Wunsch, vollführte Änderungen rückgängig zu machen
- hierfür Referenz auf Vorgängerversion
- Problem: Plugins verändern das Objekt
- Lösung: Unmodifizierbarkeit

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung

Unmodifizierbarkeit I

- Wunsch, vollführte Änderungen rückgängig zu machen
- hierfür Referenz auf Vorgängerversion
- Problem: Plugins verändern das Objekt
- Lösung: Unmodifizierbarkeit
  - Objekte nach Erstellung nicht mehr veränderbar

Entwicklung
Unmodifizierbarkeit I
- Wunsch, vollführte Änderungen rückgängig zu machen
- hierfür Referenz auf Vorgängerversion
- Problem: Plugins verändern das Objekt
- Lösung: Unmodifizierbarkeit
  - Objekte nach Erstellung nicht mehr veränderbar

1. Als nächstes: Änderungen auf Entwickler-Ebene
2. Änderungen rückgängig
3. Referenz Vorgänger
4. Plugins verändern Objekt
5. Lösung: unmodifizierbarkeit
6. nicht veränderbar

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung

Unmodifizierbarkeit I

- Wunsch, vollführte Änderungen rückgängig zu machen
- hierfür Referenz auf Vorgängerversion
- Problem: Plugins verändern das Objekt
- Lösung: Unmodifizierbarkeit
  - Objekte nach Erstellung nicht mehr veränderbar
  - Zwingt dazu, neue Objekte anzulegen

Entwicklung
Unmodifizierbarkeit I
- Wunsch, vollführte Änderungen rückgängig zu machen
- hierfür Referenz auf Vorgängerversion
- Problem: Plugins verändern das Objekt
- Lösung: Unmodifizierbarkeit
  - Objekte nach Erstellung nicht mehr veränderbar
  - Zwingt dazu, neue Objekte anzulegen

1. Als nächstes: Änderungen auf Entwickler-Ebene
2. Änderungen rückgängig
3. Referenz Vorgänger
4. Plugins verändern Objekt
5. Lösung: unmodifizierbarkeit
6. nicht veränderbar
7. für Änderung neues Objekt

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung

Unmodifizierbarkeit II

- Vorteile

1. Umständlich? Viele Vorteile

Entwicklung
Unmodifizierbarkeit II
- Vorteile

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung

Unmodifizierbarkeit II

- Vorteile
  - Objektreferenz auf Vorgänger direkt im Konstruktor

1. Umständlich? Viele Vorteile
2. Objektref Vorgänger Konstruktor

Entwicklung
Unmodifizierbarkeit II
- Vorteile
  - Objektreferenz auf Vorgänger direkt im Konstruktor

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung

Unmodifizierbarkeit II

- Vorteile
  - Objektreferenz auf Vorgänger direkt im Konstruktor
  - keine Konflikte mehr mit Hashcode-Berechnung

1. Umständlich? Viele Vorteile
2. Objektref Vorgänger Konstruktor
3. HashSet Konflikt

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung

Unmodifizierbarkeit II

- Vorteile
  - Objektreferenz auf Vorgänger direkt im Konstruktor
  - keine Konflikte mehr mit Hashcode-Berechnung
- Anpassung kostete viel Zeit

1. Umständlich? Viele Vorteile
2. Objektref Vorgänger Konstruktor
3. HashSet Konflikt
4. Anpassung kostete Zeit

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung

Unmodifizierbarkeit II

- Vorteile
  - Objektreferenz auf Vorgänger direkt im Konstruktor
  - keine Konflikte mehr mit Hashcode-Berechnung
- Anpassung kostete viel Zeit
  - Entfernung von *set()*-Methoden

1. Umständlich? Viele Vorteile
2. Objektref Vorgänger Konstruktor
3. HashSet Konflikt
4. Anpassung kostete Zeit
5. Setter entfernen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung

Unmodifizierbarkeit II

- Vorteile
  - Objektreferenz auf Vorgänger direkt im Konstruktor
  - keine Konflikte mehr mit Hashcode-Berechnung
- Anpassung kostete viel Zeit
  - Entfernung von *set()*-Methoden
  - Anpassung der bestehenden Algorithmen + Testen

1. Umständlich? Viele Vorteile
2. Objektref Vorgänger Konstruktor
3. HashSet Konflikt
4. Anpassung kostete Zeit
5. Setter entfernen
6. Algorithmen anpassen + testen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung

Unmodifizierbarkeit II

- Vorteile
  - Objektreferenz auf Vorgänger direkt im Konstruktor
  - keine Konflikte mehr mit Hashcode-Berechnung
- Anpassung kostete viel Zeit
  - Entfernung von *set()*-Methoden
  - Anpassung der bestehenden Algorithmen + Testen
- daher Automaton weiterhin modifizierbar

1. Umständlich? Viele Vorteile
2. Objektref Vorgänger Konstruktor
3. HashSet Konflikt
4. Anpassung kostete Zeit
5. Setter entfernen
6. Algorithmen anpassen + testen
7. Automaten modifizierbar

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung
Vergleichbarkeit

- Überschreiben der *equals()*- und *toHash()*-Methoden

1. Objekte nicht vergleichbar
2. Für Vergleichbarkeit: Überschreiben equals(), to Hash()

Entwicklung
Vergleichbarkeit
- Überschreiben der *equals()*- und *toHash()*-Methoden

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung
Vergleichbarkeit

- Überschreiben der *equals()*- und *toHash()*-Methoden
  - ermöglicht Vergleichbarkeit von Komponenten

1. Objekte nicht vergleichbar
2. Für Vergleichbarkeit: Überschreiben equals(), to Hash()
3. dadurch Vergleichbarkeit

Entwicklung
Vergleichbarkeit
- Überschreiben der *equals()*- und *toHash()*-Methoden
  - ermöglicht Vergleichbarkeit von Komponenten

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung
Vergleichbarkeit

- Überschreiben der *equals()*- und *toHash()*-Methoden
  - ermöglicht Vergleichbarkeit von Komponenten
  - vereinfacht Algorithmen, da es nicht mehr nötig ist, genau die richtige Referenz zu haben

1. Objekte nicht vergleichbar
2. Für Vergleichbarkeit: Überschreiben equals(), to Hash()
3. dadurch Vergleichbarkeit
4. Vereinfacht Code, nicht mehr genau Referenz

Entwicklung
Vergleichbarkeit
- Überschreiben der *equals()*- und *toHash()*-Methoden
  - ermöglicht Vergleichbarkeit von Komponenten
  - vereinfacht Algorithmen, da es nicht mehr nötig ist, genau die richtige Referenz zu haben

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung
Vergleichbarkeit

- Überschreiben der *equals()*- und *toHash()*-Methoden
  - ermöglicht Vergleichbarkeit von Komponenten
  - vereinfacht Algorithmen, da es nicht mehr nötig ist, genau die richtige Referenz zu haben
  - Konsistenz in *HashSets*

1. Objekte nicht vergleichbar
2. Für Vergleichbarkeit: Überschreiben equals(), to Hash()
3. dadurch Vergleichbarkeit
4. Vereinfacht Code, nicht mehr genau Referenz
5. gleiche Objekte, gleicher Hash

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung
Vergleichbarkeit

- Überschreiben der *equals()*- und *toHash()*-Methoden
  - ermöglicht Vergleichbarkeit von Komponenten
  - vereinfacht Algorithmen, da es nicht mehr nötig ist, genau die richtige Referenz zu haben
  - Konsistenz in *HashSets*
- Dafür mussten die Models umgeschrieben werden

1. Objekte nicht vergleichbar
2. Für Vergleichbarkeit: Überschreiben equals(), to Hash()
3. dadurch Vergleichbarkeit
4. Vereinfacht Code, nicht mehr genau Referenz
5. gleiche Objekte, gleicher Hash
6. Models ändern

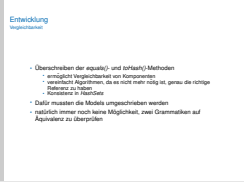
Entwicklung
Vergleichbarkeit
- Überschreiben der equals()- und toHash()-Methoden
  - ermöglicht Vergleichbarkeit von Komponenten
  - vereinfacht Algorithmen, da es nicht mehr nötig ist, genau die richtige Referenz zu haben
  - Konsistenz in HashSets
- Dafür mussten die Models umgeschrieben werden

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung
Vergleichbarkeit

- Überschreiben der *equals()*- und *toHash()*-Methoden
  - ermöglicht Vergleichbarkeit von Komponenten
  - vereinfacht Algorithmen, da es nicht mehr nötig ist, genau die richtige Referenz zu haben
  - Konsistenz in *HashSets*
- Dafür mussten die Models umgeschrieben werden
- natürlich immer noch keine Möglichkeit, zwei Grammatiken auf Äquivalenz zu überprüfen

1. Objekte nicht vergleichbar
2. Für Vergleichbarkeit: Überschreiben equals(), to Hash()
3. dadurch Vergleichbarkeit
4. Vereinfacht Code, nicht mehr genau Referenz
5. gleiche Objekte, gleicher Hash
6. Models ändern

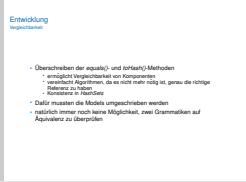


HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung
Vergleichbarkeit

- Überschreiben der *equals()*- und *toHash()*-Methoden
  - ermöglicht Vergleichbarkeit von Komponenten
  - vereinfacht Algorithmen, da es nicht mehr nötig ist, genau die richtige Referenz zu haben
  - Konsistenz in *HashSets*
- Dafür mussten die Models umgeschrieben werden
- natürlich immer noch keine Möglichkeit, zwei Grammatiken auf Äquivalenz zu überprüfen

1. Objekte nicht vergleichbar
2. Für Vergleichbarkeit: Überschreiben equals(), to Hash()
3. dadurch Vergleichbarkeit
4. Vereinfacht Code, nicht mehr genau Referenz
5. gleiche Objekte, gleicher Hash
6. Models ändern
7. Grammatiken äquivalent?



HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung
Persistierung

- Interface *Storable*

1. Interface Storable
2. Vorteil: Super-Typ
3. Workspace, Objekte im Speicher, Baum, Pendant Festplatte
4. Beenden Festplatte
5. Verzeichnis jar
6. Config-Datei, Benutzer Änderungen

Entwicklung
Persistierung
- Interface *Storable*

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung
Persistierung

- Interface *Storable*
- Vorteil: gemeinsamer Super-Typ

Entwicklung
Persistierung
- Interface *Storable*
- Vorteil: gemeinsamer Super-Typ

1. Interface Storable
2. Vorteil: Super-Typ
3. Workspace, Objekte im Speicher, Baum, Pendant Festplatte
4. Beenden Festplatte
5. Verzeichnis jar
6. Config-Datei, Benutzer Änderungen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung
Persistierung

- Interface *Storable*
- Vorteil: gemeinsamer Super-Typ
- Workspace

Entwicklung
Persistierung
- Interface *Storable*
- Vorteil: gemeinsamer Super-Typ
- Workspace

1. Interface Storable
2. Vorteil: Super-Typ
3. Workspace, Objekte im Speicher, Baum, Pendant Festplatte
4. Beenden Festplatte
5. Verzeichnis jar
6. Config-Datei, Benutzer Änderungen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung
Persistierung

- Interface *Storable*
- Vorteil: gemeinsamer Super-Typ
- Workspace
- wird beim Beenden auf der Festplatte gespeichert

Entwicklung
Persistierung
- Interface *Storable*
- Vorteil: gemeinsamer Super-Typ
- Workspace
- wird beim Beenden auf der Festplatte gespeichert

1. Interface Storable
2. Vorteil: Super-Typ
3. Workspace, Objekte im Speicher, Baum, Pendant Festplatte
4. Beenden Festplatte
5. Verzeichnis jar
6. Config-Datei, Benutzer Änderungen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung

Persistierung

- Interface *Storable*
- Vorteil: gemeinsamer Super-Typ
- Workspace
- wird beim Beenden auf der Festplatte gespeichert
- selbes Verzeichnis wie jar

1. Interface Storable
2. Vorteil: Super-Typ
3. Workspace, Objekte im Speicher, Baum, Pendant Festplatte
4. Beenden Festplatte
5. Verzeichnis jar
6. Config-Datei, Benutzer Änderungen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung
Persistierung

- Interface *Storable*
- Vorteil: gemeinsamer Super-Typ
- Workspace
- wird beim Beenden auf der Festplatte gespeichert
- selbes Verzeichnis wie jar
- config Datei

Entwicklung
Persistierung
- Interface *Storable*
- Vorteil: gemeinsamer Super-Typ
- Workspace
- wird beim Beenden auf der Festplatte gespeichert
- selbes Verzeichnis wie jar
- config Datei

1. Interface Storable
2. Vorteil: Super-Typ
3. Workspace, Objekte im Speicher, Baum, Pendant Festplatte
4. Beenden Festplatte
5. Verzeichnis jar
6. Config-Datei, Benutzer Änderungen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung
LaTeX-Generierung

- im *LaTeX-Modus* Schreiben von kompilierbaren LaTeX-Dateien

Latex, Erklärung wie

1. Ziel: Erstellung Latex File

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung
LaTeX-Generierung

- im *LaTeX-Modus* Schreiben von kompilierbaren LaTeX-Dateien
- dokumentierte Ausführung von Algorithmen oder Drucken von Objekten

Entwicklung
LaTeX-Generierung
- im *LaTeX-Modus* Schreiben von kompilierbaren LaTeX-Dateien
- dokumentierte Ausführung von Algorithmen oder Drucken von Objekten

Latex, Erklärung wie

1. Ziel: Erstellung Latex File
2. alles, was gemacht wird
3. Lerneffekt, Skript

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung

LaTeX-Generierung

- im *LaTeX-Modus* Schreiben von kompilierbaren LaTeX-Dateien
- dokumentierte Ausführung von Algorithmen oder Drucken von Objekten
- manuelles Starten und Beenden

Entwicklung
LaTeX-Generierung
- im *LaTeX-Modus* Schreiben von kompilierbaren LaTeX-Dateien
- dokumentierte Ausführung von Algorithmen oder Drucken von Objekten
- manuelles Starten und Beenden

Latex, Erklärung wie

1. Ziel: Erstellung Latex File
2. alles, was gemacht wird
3. Lerneffekt, Skript
4. manuell starten + beenden

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung
LaTeX-Generierung

- im *LaTeX-Modus* Schreiben von kompilierbaren LaTeX-Dateien
- dokumentierte Ausführung von Algorithmen oder Drucken von Objekten
- manuelles Starten und Beenden
- Herausforderungen

Entwicklung
LaTeX-Generierung
- im *LaTeX-Modus* Schreiben von kompilierbaren LaTeX-Dateien
- dokumentierte Ausführung von Algorithmen oder Drucken von Objekten
- manuelles Starten und Beenden
- Herausforderungen

Latex, Erklärung wie

1. Ziel: Erstellung Latex File
2. alles, was gemacht wird
3. Lerneffekt, Skript
4. manuell starten + beenden
5. Welche Herausforderungen?

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung
LaTeX-Generierung

- im *LaTeX-Modus* Schreiben von kompilierbaren LaTeX-Dateien
- dokumentierte Ausführung von Algorithmen oder Drucken von Objekten
- manuelles Starten und Beenden
- Herausforderungen
  - Sonderzeichen

Entwicklung
LaTeX-Generierung
- im *LaTeX-Modus* Schreiben von kompilierbaren LaTeX-Dateien
- dokumentierte Ausführung von Algorithmen oder Drucken von Objekten
- manuelles Starten und Beenden
- Herausforderungen
  - Sonderzeichen

Latex, Erklärung wie

1. Ziel: Erstellung Latex File
2. alles, was gemacht wird
3. Lerneffekt, Skript
4. manuell starten + beenden
5. Welche Herausforderungen?
6. Sonderzeichen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung
LaTeX-Generierung

- im *LaTeX-Modus* Schreiben von kompilierbaren LaTeX-Dateien
- dokumentierte Ausführung von Algorithmen oder Drucken von Objekten
- manuelles Starten und Beenden
- Herausforderungen
  - Sonderzeichen
  - Automaten

Entwicklung
LaTeX-Generierung
- im *LaTeX-Modus* Schreiben von kompilierbaren LaTeX-Dateien
- dokumentierte Ausführung von Algorithmen oder Drucken von Objekten
- manuelles Starten und Beenden
- Herausforderungen
  - Sonderzeichen
  - Automaten

Latex, Erklärung wie

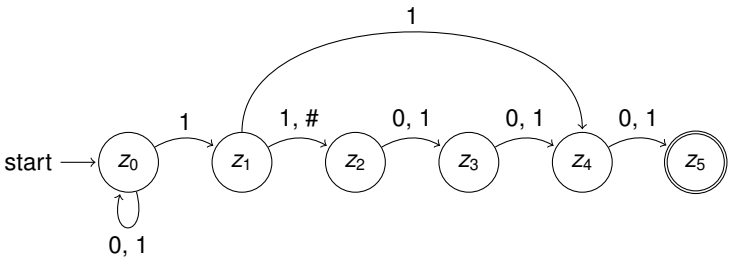
1. Ziel: Erstellung Latex File
2. alles, was gemacht wird
3. Lerneffekt, Skript
4. manuell starten + beenden
5. Welche Herausforderungen?
6. Sonderzeichen
7. generisch Automaten

ein hübscher Automat

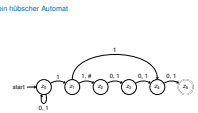
waagerecht, alphanumerisch, Winkel

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

ein hübscher Automat



ein hübscher Automat



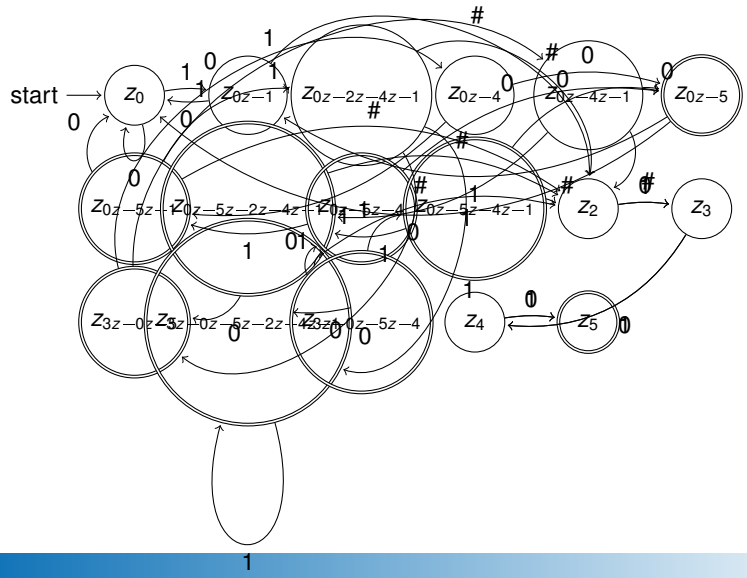
waagerecht, alphanumerisch, Winkel

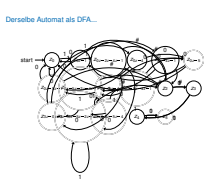
Derselbe Automat als DFA...

dieser Automato DFA

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

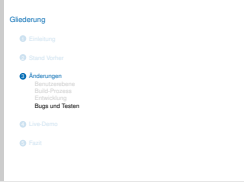
Derselbe Automat als DFA...





dieser Automato DFA

Gliederung



einige Herausforderungen, Bugs + Testen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

erwähnenswerte Bugs

- Workspace ausgewählt, der nicht valide ist? →

1. Workspace nicht valide?

erwähnenswerte Bugs
- Workspace ausgewählt, der nicht valide ist? →

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

erwähnenswerte Bugs

- Workspace ausgewählt, der nicht valide ist? →
- der ganze Ordner wird gelöscht!

1. Workspace nicht valide?
2. Ordner gelöscht

erwähnenswerte Bugs
- Workspace ausgewählt, der nicht valide ist? →
- der ganze Ordner wird gelöscht!

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

erwähnenswerte Bugs

- Workspace ausgewählt, der nicht valide ist? →
- der ganze Ordner wird gelöscht!
- ganz großen Automaten anzeigen lassen? →

1. Workspace nicht valide?
2. Ordner gelöscht
3. große Automaten

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

erwähnenswerte Bugs

- Workspace ausgewählt, der nicht valide ist? →
- der ganze Ordner wird gelöscht!
- ganz großen Automaten anzeigen lassen? →
- Exception!

1. Workspace nicht valide?
2. Ordner gelöscht
3. große Automaten
4. ArrayOutOfBoundException!

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

erwähnenswerte Bugs

- Workspace ausgewählt, der nicht valide ist? →
- der ganze Ordner wird gelöscht!
- ganz großen Automaten anzeigen lassen? →
- Exception!
- Grammatik eingegeben, die Schwachsinn ist? →

erwähnenswerte Bugs
- Workspace ausgewählt, der nicht valide ist? →
- der ganze Ordner wird gelöscht!
- ganz großen Automaten anzeigen lassen? →
- Exception!
- Grammatik eingegeben, die Schwachsinn ist? →

1. Workspace nicht valide?
2. Ordner gelöscht
3. große Automaten
4. ArrayOutOfBoundException!
5. ungültige Grammatik

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

erwähnenswerte Bugs

- Workspace ausgewählt, der nicht valide ist? →
- der ganze Ordner wird gelöscht!
- ganz großen Automaten anzeigen lassen? →
- Exception!
- Grammatik eingegeben, die Schwachsinn ist? →
- Grammatik weg!

1. Workspace nicht valide?
2. Ordner gelöscht
3. große Automaten
4. ArrayOutOfBoundException!
5. ungültige Grammatik
6. CNF Algo löscht alles

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

erwähnenswerte Bugs

- Workspace ausgewählt, der nicht valide ist? →
- der ganze Ordner wird gelöscht!
- ganz großen Automaten anzeigen lassen? →
- Exception!
- Grammatik eingegeben, die Schwachsinn ist? →
- Grammatik weg!

1. Workspace nicht valide?
2. Ordner gelöscht
3. große Automaten
4. ArrayOutOfBoundException!
5. ungültige Grammatik
6. CNF Algo löscht alles

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

erwähnenswerte Bugs

- Workspace ausgewählt, der nicht valide ist? →
- der ganze Ordner wird gelöscht!
- ganz großen Automaten anzeigen lassen? →
- Exception!
- Grammatik eingegeben, die Schwachsinn ist? →
- Grammatik weg!

erwähnenswerte Bugs
- Workspace ausgewählt, der nicht valide ist? →
- der ganze Ordner wird gelöscht!
- ganz großen Automaten anzeigen lassen? →
- Exception!
- Grammatik eingegeben, die Schwachsinn ist? →
- Grammatik weg!

1. Workspace nicht valide?
2. Ordner gelöscht
3. große Automaten
4. ArrayOutOfBoundException!
5. ungültige Grammatik
6. CNF Algo löscht alles

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

erwähnenswerte Bugs

- Workspace ausgewählt, der nicht valide ist? →
- der ganze Ordner wird gelöscht!
- ganz großen Automaten anzeigen lassen? →
- Exception!
- Grammatik eingegeben, die Schwachsinn ist? →
- Grammatik weg!

erwähnenswerte Bugs
- Workspace ausgewählt, der nicht valide ist? →
- der ganze Ordner wird gelöscht!
- ganz großen Automaten anzeigen lassen? →
- Exception!
- Grammatik eingegeben, die Schwachsinn ist? →
- Grammatik weg!

1. Workspace nicht valide?
2. Ordner gelöscht
3. große Automaten
4. ArrayOutOfBoundException!
5. ungültige Grammatik
6. CNF Algo löscht alles

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Testen

- Legacy-Code

1. Testen
2. Legacy-Code

Testen
- Legacy-Code

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Testen

- Legacy-Code
  - gut kommentiert, aber

1. Testen
2. Legacy-Code
3. gut Kommentare, keine Testabdeckung

Testen
- Legacy-Code
  - gut kommentiert, aber

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Testen

- Legacy-Code
  - gut kommentiert, aber
  - keine Testabdeckung vorhanden

1. Testen
2. Legacy-Code
3. gut Kommentare, keine Testabdeckung
4. Bugs Automaten, nicht behoben

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Testen

- Legacy-Code
  - gut kommentiert, aber
  - keine Testabdeckung vorhanden
  - es gibt einige Bugs, die nicht behoben werden konnten

Testen
- Legacy-Code
  - gut kommentiert, aber
  - keine Testabdeckung vorhanden
  - es gibt einige Bugs, die nicht behoben werden konnten

1. Testen
2. Legacy-Code
3. gut Kommentare, keine Testabdeckung
4. Bugs Automaten, nicht behoben
5. Algorithmen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Testen

- Legacy-Code
  - gut kommentiert, aber
  - keine Testabdeckung vorhanden
  - es gibt einige Bugs, die nicht behoben werden konnten
- Algorithmen

1. Testen
2. Legacy-Code
3. gut Kommentare, keine Testabdeckung
4. Bugs Automaten, nicht behoben
5. Algorithmen
6. manche Algorithmen: ja, Bspl LambdaFree

Testen
- Legacy-Code
  - gut kommentiert, aber
  - keine Testabdeckung vorhanden
  - es gibt einige Bugs, die nicht behoben werden konnten
- Algorithmen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Testen

- Legacy-Code
  - gut kommentiert, aber
  - keine Testabdeckung vorhanden
  - es gibt einige Bugs, die nicht behoben werden konnten
- Algorithmen
  - automatisches Überprüfen von Algorithmen: wird das richtige Ergebnis geliefert?

1. Testen
2. Legacy-Code
3. gut Kommentare, keine Testabdeckung
4. Bugs Automaten, nicht behoben
5. Algorithmen
6. manche Algorithmen: ja, Bspl LambdaFree
7. kein Überprüfen Äquivalenz

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Testen

- Legacy-Code
  - gut kommentiert, aber
  - keine Testabdeckung vorhanden
  - es gibt einige Bugs, die nicht behoben werden konnten
- Algorithmen
  - automatisches Überprüfen von Algorithmen: wird das richtige Ergebnis geliefert?
  - nicht möglich: Vergleich von Grammatiken/Automaten

1. Testen
2. Legacy-Code
3. gut Kommentare, keine Testabdeckung
4. Bugs Automaten, nicht behoben
5. Algorithmen
6. manche Algorithmen: ja, Bspl LambdaFree
7. kein Überprüfen Äquivalenz
8. manuell, kostete Zeit

Testen
- Legacy-Code
  - gut kommentiert, aber
  - keine Testabdeckung vorhanden
  - es gibt einige Bugs, die nicht behoben werden konnten
- Algorithmen
  - automatisches Überprüfen von Algorithmen: wird das richtige Ergebnis geliefert?
  - nicht möglich: Vergleich von Grammatiken/Automaten

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Testen

- Legacy-Code
  - gut kommentiert, aber
  - keine Testabdeckung vorhanden
  - es gibt einige Bugs, die nicht behoben werden konnten
- Algorithmen
  - automatisches Überprüfen von Algorithmen: wird das richtige Ergebnis geliefert?
  - nicht möglich: Vergleich von Grammatiken/Automaten
  - manuelles Überprüfen kostete viel Zeit

1. Testen
2. Legacy-Code
3. gut Kommentare, keine Testabdeckung
4. Bugs Automaten, nicht behoben
5. Algorithmen
6. manche Algorithmen: ja, Bspl LambdaFree
7. kein Überprüfen Äquivalenz
8. manuell, kostete Zeit
9. Latex

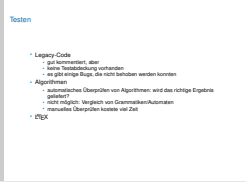
Testen
- Legacy-Code
  - gut kommentiert, aber
  - keine Testabdeckung vorhanden
  - es gibt einige Bugs, die nicht behoben werden konnten
- Algorithmen
  - automatisches Überprüfen von Algorithmen: wird das richtige Ergebnis geliefert?
  - nicht möglich: Vergleich von Grammatiken/Automaten
  - manuelles Überprüfen kostete viel Zeit

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Testen

- Legacy-Code
  - gut kommentiert, aber
  - keine Testabdeckung vorhanden
  - es gibt einige Bugs, die nicht behoben werden konnten
- Algorithmen
  - automatisches Überprüfen von Algorithmen: wird das richtige Ergebnis geliefert?
  - nicht möglich: Vergleich von Grammatiken/Automaten
  - manuelles Überprüfen kostete viel Zeit
- LaTeX

1. Testen
2. Legacy-Code
3. gut Kommentare, keine Testabdeckung
4. Bugs Automaten, nicht behoben
5. Algorithmen
6. manche Algorithmen: ja, Bspl LambdaFree
7. kein Überprüfen Äquivalenz
8. manuell, kostete Zeit
9. Latex
10. manuell Kompilierbarkeit



HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Testen

- Legacy-Code
  - gut kommentiert, aber
  - keine Testabdeckung vorhanden
  - es gibt einige Bugs, die nicht behoben werden konnten
- Algorithmen
  - automatisches Überprüfen von Algorithmen: wird das richtige Ergebnis geliefert?
  - nicht möglich: Vergleich von Grammatiken/Automaten
  - manuelles Überprüfen kostete viel Zeit
- LaTeX
  - manuelles Überprüfen auf Kompilierbarkeit

1. Testen
2. Legacy-Code
3. gut Kommentare, keine Testabdeckung
4. Bugs Automaten, nicht behoben
5. Algorithmen
6. manche Algorithmen: ja, Bspl LambdaFree
7. kein Überprüfen Äquivalenz
8. manuell, kostete Zeit
9. Latex
10. manuell Kompilierbarkeit
11. Sonderfälle, lambda

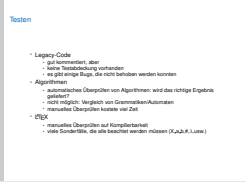
Testen
- Legacy-Code
  - gut kommentiert, aber
  - keine Testabdeckung vorhanden
  - es gibt einige Bugs, die nicht behoben werden konnten
- Algorithmen
  - automatisches Überprüfen von Algorithmen: wird das richtige Ergebnis geliefert?
  - nicht möglich: Vergleich von Grammatiken/Automaten
  - manuelles Überprüfen kostete viel Zeit
- LaTeX
  - manuelles Überprüfen auf Kompilierbarkeit

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Testen

- Legacy-Code
  - gut kommentiert, aber
  - keine Testabdeckung vorhanden
  - es gibt einige Bugs, die nicht behoben werden konnten
- Algorithmen
  - automatisches Überprüfen von Algorithmen: wird das richtige Ergebnis geliefert?
  - nicht möglich: Vergleich von Grammatiken/Automaten
  - manuelles Überprüfen kostete viel Zeit
- LaTeX
  - manuelles Überprüfen auf Kompilierbarkeit
  - viele Sonderfälle, die alle beachtet werden müssen (X_a_b,#,λ,usw.)

1. Testen
2. Legacy-Code
3. gut Kommentare, keine Testabdeckung
4. Bugs Automaten, nicht behoben
5. Algorithmen
6. manche Algorithmen: ja, Bspl LambdaFree
7. kein Überprüfen Äquivalenz
8. manuell, kostete Zeit
9. Latex
10. manuell Kompilierbarkeit
11. Sonderfälle, lambda



Gliederung

Gliederung
1 Einleitung
2 Stand Vorher
3 Änderungen
Benutzerebene
Build-Prozess
Entwicklung
Bugs und Testen
4 Live-Demo
5 Fazit

Live-Demo

Gliederung

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Gliederung
1 Einleitung
2 Stand Vorher
3 Änderungen
Benutzerebene
Build-Prozess
Entwicklung
Bugs und Testen
4 Live-Demo
5 Fazit

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Herausforderungen

- Unmodifizierbarkeit

Herausforderungen
- Unmodifizierbarkeit

Herausforderungen

1. Unmodifizierbarkeit
2. nicht so schwer, aber zeitaufwendig. fehlende Testabdeckung Problem

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Herausforderungen

- Unmodifizierbarkeit
- Algorithmen für Computer umsetzen

Herausforderungen
- Unmodifizierbarkeit
- Algorithmen für Computer umsetzen

Herausforderungen

1. Unmodifizierbarkeit
2. nicht so schwer, aber zeitaufwendig. fehlende Testabdeckung Problem
3. Algorithmen Computer
4. Skript Info 4 entnomment, für Menschen, bedurften Anpassung

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Herausforderungen

- Unmodifizierbarkeit
- Algorithmen für Computer umsetzen
- Weiterentwicklung eines bestehenden Programmes

Herausforderungen
- Unmodifizierbarkeit
- Algorithmen für Computer umsetzen
- Weiterentwicklung eines bestehenden Programmes

Herausforderungen

1. Unmodifizierbarkeit
2. nicht so schwer, aber zeitaufwendig. fehlende Testabdeckung Problem
3. Algorithmen Computer
4. Skript Info 4 entnomment, für Menschen, bedurften Anpassung
5. Weiterentwicklung
6. doch nötig Code anzugucken
7. nicht einfacher oder schwerer, komplett anders

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Herausforderungen

- Unmodifizierbarkeit
- Algorithmen für Computer umsetzen
- Weiterentwicklung eines bestehenden Programmes
- Latex

Herausforderungen
- Unmodifizierbarkeit
- Algorithmen für Computer umsetzen
- Weiterentwicklung eines bestehenden Programmes
- Latex

Herausforderungen

1. Unmodifizierbarkeit
2. nicht so schwer, aber zeitaufwendig. fehlende Testabdeckung Problem
3. Algorithmen Computer
4. Skript Info 4 entnomment, für Menschen, bedurften Anpassung
5. Weiterentwicklung
6. doch nötig Code anzugucken
7. nicht einfacher oder schwerer, komplett anders
8. Latex
9. viele Probleme, viele Sonderfälle
10. bemüht, alle abzudecken.

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Herausforderungen

- Unmodifizierbarkeit
- Algorithmen für Computer umsetzen
- Weiterentwicklung eines bestehenden Programmes
- Latex
- Testen

Herausforderungen
- Unmodifizierbarkeit
- Algorithmen für Computer umsetzen
- Weiterentwicklung eines bestehenden Programmes
- Latex
- Testen

Herausforderungen

1. Unmodifizierbarkeit
2. nicht so schwer, aber zeitaufwendig. fehlende Testabdeckung Problem
3. Algorithmen Computer
4. Skript Info 4 entnomment, für Menschen, bedurften Anpassung
5. Weiterentwicklung
6. doch nötig Code anzugucken
7. nicht einfacher oder schwerer, komplett anders
8. Latex
9. viele Probleme, viele Sonderfälle
10. bemüht, alle abzudecken.
11. Testen
12. keine bestehende Testabdeckung, JUnit
13. schwierig, wegen Äquivalenz, hauptsächlich manuell

Fazit

- Codequalität verbessert

1. Fazit
2. Codequalität: Aufgeräumt, MVC, Testabdeckung

Fazit
- Codequalität verbessert

Fazit

- Codequalität verbessert
- alle gewünschten Features konnten umgesetzt werden

1. Fazit
2. Codequalität: Aufgeräumt, MVC, Testabdeckung
3. gewünschte Features umgesetzt

Fazit
- Codequalität verbessert
- alle gewünschten Features konnten umgesetzt werden

Fazit

- Codequalität verbessert
- alle gewünschten Features konnten umgesetzt werden
- GUI übersichtlich und gut zu bedienen

1. Fazit
2. Codequalität: Aufgeräumt, MVC, Testabdeckung
3. gewünschte Features umgesetzt
4. GUI übersichtlich, gut zu bedienen

Fazit
- Codequalität verbessert
- alle gewünschten Features konnten umgesetzt werden
- GUI übersichtlich und gut zu bedienen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Fazit

- Codequalität verbessert
- alle gewünschten Features konnten umgesetzt werden
- GUI übersichtlich und gut zu bedienen
- trotz kleiner Bugs ein gut gelungenes Programm

1. Fazit
2. Codequalität: Aufgeräumt, MVC, Testabdeckung
3. gewünschte Features umgesetzt
4. GUI übersichtlich, gut zu bedienen
5. trotz kleiner Bugs, gut gelungen

Fazit
- Codequalität verbessert
- alle gewünschten Features konnten umgesetzt werden
- GUI übersichtlich und gut zu bedienen
- trotz kleiner Bugs ein gut gelungenes Programm

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Ausblick

- durchaus noch mehr Potential

1. mehr Potential
2. Turing-Maschinen
3. Programmaufruf mit Parametern
4. Kontextsensitive Grammatiken

Ausblick
- durchaus noch mehr Potential

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Ausblick

- durchaus noch mehr Potential
- Turing-Maschinen

1. mehr Potential
2. Turing-Maschinen
3. Programmaufruf mit Parametern
4. Kontextsensitive Grammatiken

Ausblick
- durchaus noch mehr Potential
- Turing-Maschinen

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Ausblick

- durchaus noch mehr Potential
- Turing-Maschinen
- Programmaufruf mit Parametern

1. mehr Potential
2. Turing-Maschinen
3. Programmaufruf mit Parametern
4. Kontextsensitive Grammatiken

Ausblick
- durchaus noch mehr Potential
- Turing-Maschinen
- Programmaufruf mit Parametern

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Ausblick

- durchaus noch mehr Potential
- Turing-Maschinen
- Programmaufruf mit Parametern
- kontextsensitive Grammatiken

1. mehr Potential
2. Turing-Maschinen
3. Programmaufruf mit Parametern
4. Kontextsensitive Grammatiken

Ausblick
- durchaus noch mehr Potential
- Turing-Maschinen
- Programmaufruf mit Parametern
- kontextsensitive Grammatiken

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Vielen Dank für die Aufmerksamkeit!

Vielen Dank für die Aufmerksamkeit!