Das System `autotool` zur automatischenKontrolle von Übungsaufgaben zur (Theoretischen) Informatik

Mirko Rahn, Uni Karlsruhe
Alf Richter, Uni Leipzig
Johannes Waldmann, HTWK Leipzig

Typische Anwendung

roblem (Bsp: 3SAT)

- Instanz: eine Formel in 3-KNF
- Lösung: eine erfüllende Belegung

blauf mit `autotool` :

- Tutor konfiguriert Generator
- `autotool` würfelt Instanz (für jeden Studenten eine andere)
- Student gibt (vermutete) Lösung ein
- `autotool` verifiziert Lösung, gibt ausführlichen Bericht (sofort)

Bestandteile des `autotool`

- Semantik-Module
  (Automaten, Grammatiken, Graphen, . . .)
- Generatoren für Aufgaben
- Korrektoren für Einsendungen
- Datenbank
  für Konfiguration der Aufgaben bzw. Generatoren,
  erreichte Punkte
- Web-Schnittstelle
  für Studenten (seit 2003), Tutoren (seit 2004)

Wertung von Aufgaben

ufgaben markieren als

- Demo (zu leicht, illustrieren Prinzip)
- Mandatory (Pflicht, Einsenden einer korrekten Lösung innerhalb der Deadline genügt)
- Optional (zu schwer, zum Basteln, Highscore)

udent kann auch außerhalb der Deadlines

- Aufgaben bearbeiten (wird korrigiert, nicht gespeichert)
- vorige Bewertung ansehen

tzlich z. B. zur Prüfungsvorbereitung

Aufgabenbereiche

- formale Sprachen:
  Grammatiken (alle Chomsky-Typen), reguläre Ausdrücke
- Automaten/Berechnungsmodelle:
  endlich (Wort, Baum), Keller, Turing, Registermaschine,
  (primitiv) rekursive Funktionen
- Graphen: Parameter, Färbungen, Wege, . . .
- diskrete Mathematik und Logik: Zahlensysteme, . . .
- Datenstrukturen: Suchbäume, . . .

`autotool` als Verifizierer

- Der Idealfall sind Probleme aus NP (z. B. 3SAT):
  - Student muß lange überlegen/suchen (N)
  - `autotool` kann schnell verifizieren (P)
- Manchmal sind die Lösungen länger (PCP)
- oder das Verifizieren dauert länger (Äquivalenz regulärer Ausdrücke)
- Manchmal kann man nicht verifizieren, sondern nur testen (Äquivalenz von CFG).

etet immer viele Möglichkeiten für Diskussionen mi
tudenten über Berechenbarkeit und Komplexität

Wichtige Datentypen

ür Automaten und Sprachen)

- endlicher Automat, Kellerautomat
- Grammatik
- regulärer Ausdruck

epräsentation möglichst nahe an der
athematischen Wahrheit,
att Tupel benutze Strukturen mit benannten
omponenten,
eschweifte Klammern sind dafür reserviert,
eswegen Mengen als `mkSet [ 1, 2, 3 ]`

Endliche Automaten

```
A { alphabet = mkSet "ab" , states = mkSet [ 1, 2, 3 ]
  , starts = mkSet [ 2 ] , finals = mkSet [ 2 ]
  , trans = collect
        [ ( 1, 'a', 2 ), ( 2, 'a', 1 )
        , ( 2, 'a', 3 ), ( 2, 'b', 3 ), ( 3, 'b', 2 )
        ]
  }
```

genschaften (für Test der Einsendungen und ggf.
onfiguration der Generatoren)

```
Min_Size Int | Max_Size Int
Alphabet ( Set c )
Deterministic | Minimal | Complete | Reduced
```

Reguläre Ausdrücke

`(ba)^* + (bb)^*)^* + (ab)^*`

efinition:

- atomar: Buchstabe oder Name (`Sigma, All`)
- zusammengesetzt mit Operationen:
  Verkettung (`.`), Vereinigung `+`, Durchschnitt `&`,
  Differenz `-`, symmetrische Differenz `<>`, Shuffle `$`,
  Quotienten `\,/`, Potenzen `^int, ^*, ^+`

genschaften:

```
 Min_Size Int | Max_Size Int | Alphabet ( Set
| Simple -- nur Plus, Mal, Stern
| Extended -- alles
```

Grammatiken

```
ammatik { terminale = mkSet "01"
        , variablen = mkSet "ST"
, start = 'S'
, regeln = mkSet [ ( "S", "" ), ( "S", "T0S" ), ( "T", "11" )
}
```

genschaften:

- monoton, kontextsensitiv, kontextfrei, linear, rechts/links-linear,
- für CFG: epsilon-frei, ketten-frei, Chomsky-normal, Greibach-normal
- eindeutig (natürlich nur Test)

Eine clevere Lösung

ufgabe war: finde (kleine) CFG für
$= \Sigma^* \setminus \{ww \mid w \in \Sigma^*\}$ für $\Sigma = \{0, 1\}$.
e wesentliche Idee der Lösung ist:

- $L \to AB, L \to BA$ (start)
- $A \to 0, A \to SAS$ (in der Mitte eine 0)
- $B \to 1, B \to SBS$ (in der Mitte eine 1)
- $S \to 0, S \to 1$ (ein beliebiges Zeichen)

tzt fehlt aber noch etwas triviales.
ie schafft man das mit nur zwei (nicht drei) weiteren
egeln?

Einsatz in Lehrveranstaltungen

- Automaten und Sprachen (Uni Leipzig, 2001–2004
- Berechenbarkeit und Komplexität (Uni L)
- Automaten und Sprachen im Compilerbau (HTWK Leipzig, 2003–)
- Automaten und Sprachen in Grundlagen der Informatik (Nebenfach) (HTWK L)
- Datenstrukturen, diskrete Mathematik in Grundlagen der Informatik (Nebenfach) (HTWK L)
- Datenstrukturen, disk. Math. in Grundl. Inf. (Nebenfach) (Uni Karlsruhe, 2005)

Erfahrungen

- `autotool` zur Unterstützung des Übungsbetriebes.
- wird von Studenten gut angenommen,
- sie sind erfreut über sofortige und ausführliche Antwort.
- Highscore-Wertung (mit Preisen) schafft zusätzlichen Anreiz
- etwa die Hälfte der Aufgaben mit `autotool` korrigieren (spart HiWis) (. . . die andere Hälfte sind Beweis-Aufgaben)

Installation, Nutzung

eparate Installation benötigt

- halbwegs schnellen Rechner mit GNU/Linux, Apache Webserver, GHC-Compiler, MySQL-Datenbankserver (. . . `theo1`)
- dafür Administrator, der System und Datenbank einrichtet und im Betrieb bei Bedarf Patches einspielt und kompiliert

nfacher ist wohl die Benutzung eines gemeinsamen `totool`-Servers durch mehrere Einrichtungen (Uni nd HTWK)

autotool intern

nplementiert in Haskell (purely functional, strictly
ped, polymorphic, lazy)

on Waldmann, Rahn, Richter seit ca. 2001

nige Teile waren Belegarbeiten für Vorlesung
unktionale Programmierung (Gerber)

mfang:

- Bibliothek (allg. Datenstrukturen und endliche Automaten): 300 Module, 15 kLOC;
- Tool: 600 Module, 45 kLOC

0 Zeilen pro Arbeitstag (Fred Brooks, 1972) $\longrightarrow$ . . .)

Weiterentwicklung

- derzeit Waldmann (Leipzig) und Rahn (Karlsruhe) nach „Eigenbedarf“
- Bibliothek nachgenutzt für Matchbox (180 Module, 15 kLOC) (findet Beweise für Termination von Wort- und Termersetzungssystemen)

r systematische Weiterentwicklung und
ufräum-Arbeiten (Oberfläche, Dokumentation):
enötigen HiWis oder Diplomanden mit Kenntnissen ir
unktionaler Programmierung
orlesung ab SS06 an HTWK)