

Patrick Simianer  
Visualisierung regulärer Ausdrücke

Patrick Simianer 2508483

2010-06-28

Endliche Automaten HS bei Dr. Karin Haenelt  
Universität Heidelberg im Sommersemester 2010

# Gliederung

- 1 Einleitung
  - Überlegungen
  - Protoypisches Vorgehen
  - Konkreter Aufbau
- 2 Einfaches Parsing regulärer Ausdrücke
  - Recursive Descent-Methode
  - Konstruktion nach Thompson
  - Beispiel
- 3 Überführung NDEA zu einem DEA
  - $\epsilon$ -Abschluss
  - Beispiel
- 4 Demo
- 5 Weiterentwicklung

## 1 Einleitung

Überlegungen  
Protoypisches Vorgehen  
Konkreter Aufbau

## 2 Einfaches Parsing regulärer Ausdrücke

Recursive Descent-Methode  
Konstruktion nach Thompson  
Beispiel

## 3 Überführung NDEA zu einem DEA

$\epsilon$ -Abschluss  
Beispiel

## 4 Demo

## 5 Weiterentwicklung

## Visualisierung regulärer Ausdrücke

Wie soll die Visualisierung aussehen?

- 1 Hervorheben von **Matches** oder **Gruppen** in einem String oder Text
  - 2 Darstellung und Simulation anhand eines **Automaten**
- 
- 1 Es existieren bereits viele Implementierungen, basierend auf RE-Implementierung der jeweiligen Sprache.  
→ Keine „*step by step*“-Visualisierung möglich
  - 2 Grafische Umsetzung schwierig, eigene RE-Implementierung nötig  
→ Jeder Schritt der Erkennung nachvollziehbar

## Visualisierung regulärer Ausdrücke /2

- 1 Wie können reguläre Ausdrücke möglichst einfach und effizient implementiert werden?
  - „Herkömmliche“ *backtracking*-Methode (*Perl*, *PCRE*)
  - ⇒ Direkte Konstruktion eines **endlichen Automaten**
- 2 Soll der Automat dargestellt werden – und wenn ja, wie?
  - ⇒ **Ja**, im besten Fall mit Animationen...
- 3 In welcher Umgebung können alle Teile (1. Parser, 2. GUI, 3. Visualisierung) gut implementiert werden?
  - ⇒ Im **Browser** (1. *JavaScript*, 2. *HTML*, 3. *SVG*)

## Protoypisches Vorgehen

- 1 **Parsen** des Ausdrucks
- 2 Umsetzung in einen **nichtdeterministischen endlichen Automaten**
- 3 Übersetzung des NDEA in einen **deterministischen** endlichen Automaten
- 4 Grafische **Darstellung** des Automaten und dessen **Simulation** durch Animation

Umsetzung im Browser: *JavaScript* (*Raphaël* für *SVG*, *jQuery*), *HTML+CSS*

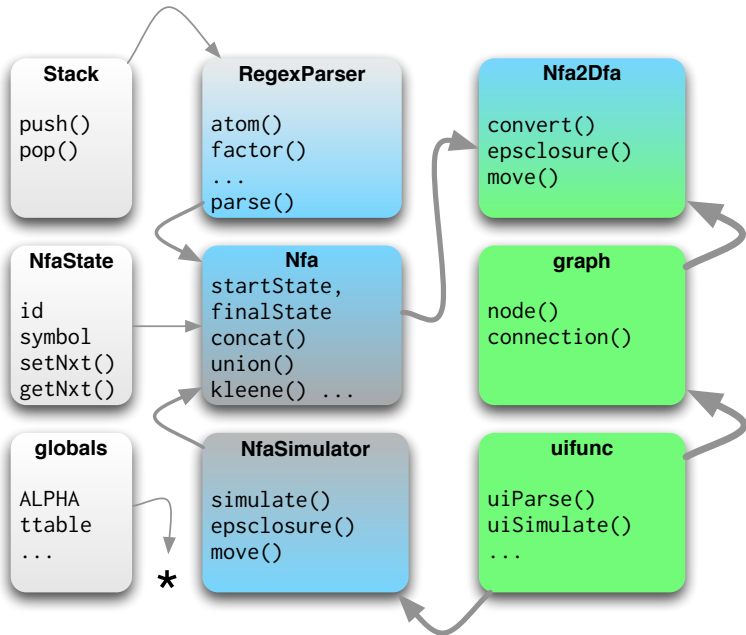


Abbildung: Konkreter System-Aufbau

- 1 Einleitung
  - Überlegungen
  - Protoypisches Vorgehen
  - Konkreter Aufbau
- 2 Einfaches Parsing regulärer Ausdrücke
  - Recursive Descent-Methode
  - Konstruktion nach Thompson
  - Beispiel
- 3 Überführung NDEA zu einem DEA
  - $\epsilon$ -Abschluss
  - Beispiel
- 4 Demo
- 5 Weiterentwicklung



## Recursive Descent-Methode

### Grammatik:

**expr** → term | term | expr  
**term** → factor | term  
**factor** → atom kleene  
**atom** → literal | ( expr )  
**kleene** → \* kleene | ε  
**literal** → a | b | c | %

### Code:

```

RegexParser.prototype.expr = function() {
  var nfa = this.term();
  if (this.trymatch('|')) {
    return nfa.union(this.expr());
  }
  return nfa;
};

```

- Nahezu direktes Übersetzen einer Grammatik<sup>1</sup> in den Quelltext des Parsers, *top-down* (LL(1))
- $\forall$  Nichtterminale  $\exists$  Funktion, welche die rechte Seite der jeweiligen Regel behandelt
- Direkte Erzeugung des NDEA, mittels Konstruktion nach Thompson in  $O(|r|)$  ( $|r|$  Länge des regulären Ausdrucks)
- Ergebnis: Automat mit max.  $2|r|$  Zuständen,  $4|r|$  Transitionen

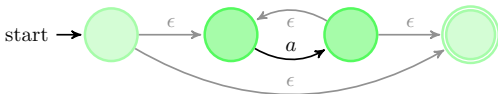
<sup>1</sup>keine Links-Rekursionen, sonst: Endlosschleife

## Konstruktion nach Thompson

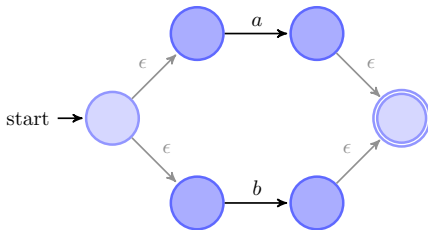
Konkatenation:  $ab$



Hülle:  $a^*$

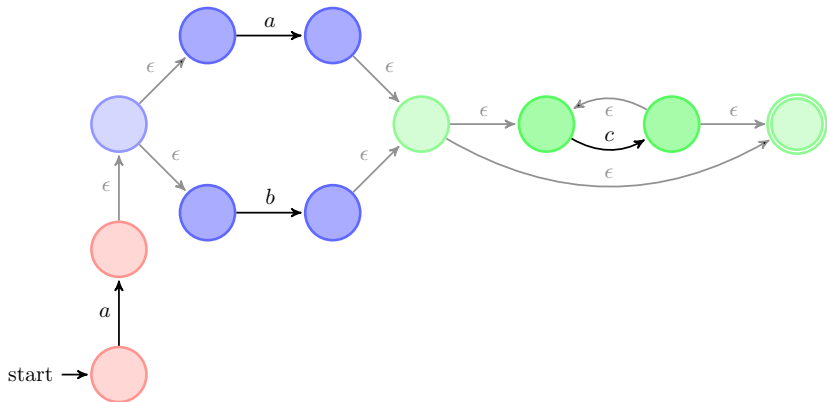


Vereinigung:  $(a|b)$



## Thompsons Algorithmus: Beispiel

Regulärer Ausdruck:  $a(a|b)c^*$



- 1 Einleitung
  - Überlegungen
  - Protoypisches Vorgehen
  - Konkreter Aufbau
- 2 Einfaches Parsing regulärer Ausdrücke
  - Recursive Descent-Methode
  - Konstruktion nach Thompson
  - Beispiel
- 3 Überführung NDEA zu einem DEA
  - $\epsilon$ -Abschluss
  - Beispiel
- 4 Demo
- 5 Weiterentwicklung

## Einleitung

Warum den erzeugten NDEA in einen DEA überführen?

*trade-off:*

	NDEA	DEA
Platzbedarf	$2 r  = m, 4 r  = n$	$2^m$
Erstellungszeit	$O( r )$	$O( r ^2 2^{ r }), O( r ^3)$
Simulation	$O( x (m+n))$ $\approx O( x  *  r )$	$O( x )$

$|x|$  Länge des Eingabe-Strings; bei Erstellungszeit DEA: links *worst-case*, rechts der Durchschnittsfall

- Die hier erzeugten NDEA umfassen für gewöhnlich sehr viele Zustände, die Darstellung eines DEA ist praktikabler

## ε-Abschluss

### Pseudo-Code

```

epsclosure(dState): stack s
foreach nState in dState {
  s.push(nState)
}
while s not empty {
  nState1 = s.pop()
  foreach nState1 e> nState2 {
    if nState2 not in dState {
      dState.add(nState2)
      s.push(nState2)
    }
  }
}
return dState

```

```

nfa2dfa(NFA): stack s, DFA d
s.push(epsclosure(nfa.start))
d.add(epsclosure(nfa.start))
while s not empty {
  dState1 = s.pop()
  foreach ch in ALPHA {
    dState2 = move(dState1, ch)
    next = epsclosure(dState2)
    if next not in DFA {
      d.add(dState ch> next)
    }
  }
}
return d

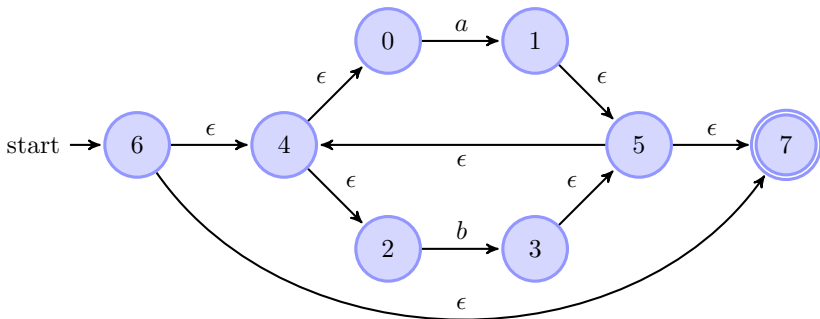
```

a e> b: ε-Übergang von Z. a nach b; a ch> b: Übergang mit Zeichen ch

- **Formal:**  $\forall p \in Q : E(\{p\}) = \{q \in Q : p \rightarrow_{\epsilon} q\}$  (ε-Abschl. v. a. p aus Q)
- **Simulation** des NDEA, oder vollständige Erzeugung eines **DEA**

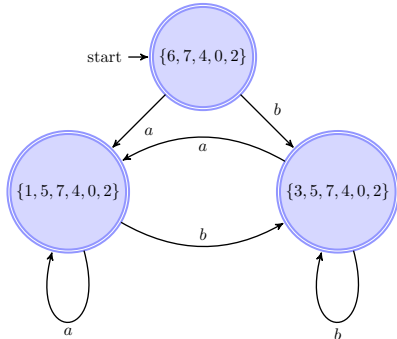
## NDEA $\rightarrow$ DEA Beispiel

Regulärer Ausdruck:  $(a|b)^*$



## NDEA $\rightarrow$ DEA Beispiel /2

Dfa ID	Symbol	$\rightarrow$ Dfa ID
$\{6, 7, 4, 0, 2\}$	$a$	$\{1, 5, 7, 4, 0, 2\}$
	$b$	$\{3, 5, 7, 4, 0, 2\}$
$\{1, 5, 7, 4, 0, 2\}$	$a$	$\{1, 5, 7, 4, 0, 2\}$
	$b$	$\{3, 5, 7, 4, 0, 2\}$
$\{3, 5, 7, 4, 0, 2\}$	$a$	$\{1, 5, 7, 4, 0, 2\}$
	$b$	$\{3, 5, 7, 4, 0, 2\}$





Fragen?

## Literatur I



Alfred V. Aho, Ravi Sethi, and Jeffrey David Ullman.

*Compilers. Principles, Techniques & Tools.*

Addison-Wesley, Reading, Mass. [u.a.], repr. with corr. edition, 1986.

Literaturverz. S. 752 - 779.



Russ Cox.

Regular Expression Matching Can Be Simple And Fast.

2007.

[Online; abgerufen 2010-06-06].



Hans Werner Lang.

*Algorithmen in Java.*

Oldenbourg, Wiesbaden, 2006.

## Literatur II



Gonzalo Navarro and Mathieu Raffinot.

Compact dfa representation for fast regular expression search, 2001.



K. Thompson.

Regular expression search algorithm.

*Comm. Assoc. Comp. Mach.*, 11(6):419–422, 1968.



Gertjan van Noord.

The treatment of epsilon moves in subset construction.

In *IN FINITE-STATE METHODS IN NATURAL LANGUAGE PROCESSING, ANKARA. CMP-LG/9804003*, pages 61–76, 1998.

## Ressourcen

- *Raphaël* JavaScript SVG Library (<http://raphaeljs.com/>)
- *jQuery* JavaScript Library (<http://jquery.com/>)
- Scalable Vector Graphics (SVG) 1.1 Specification (<http://www.w3.org/TR/SVG/>)
- Writing your own regular expression parser (<http://www.codeproject.com/KB/recipes/OwnRegExpressionsParser.aspx>)

- 1 Einleitung
  - Überlegungen
  - Protoypisches Vorgehen
  - Konkreter Aufbau
- 2 Einfaches Parsing regulärer Ausdrücke
  - Recursive Descent-Methode
  - Konstruktion nach Thompson
  - Beispiel
- 3 Überführung NDEA zu einem DEA
  - $\epsilon$ -Abschluss
  - Beispiel
- 4 Demo
- 5 Weiterentwicklung

Demo

- 1 Einleitung
  - Überlegungen
  - Protoypisches Vorgehen
  - Konkreter Aufbau
- 2 Einfaches Parsing regulärer Ausdrücke
  - Recursive Descent-Methode
  - Konstruktion nach Thompson
  - Beispiel
- 3 Überführung NDEA zu einem DEA
  - $\epsilon$ -Abschluss
  - Beispiel
- 4 Demo
- 5 Weiterentwicklung

## Weiterentwicklung

- Vorhanden:  $*$ ,  $|$ ,  $()$
- Zeichenklassen:  $.$ ,  $\backslash w$ ,  $\backslash d$ ,  $[ ]$ ,  $\dots$   $\rightarrow$  Einfach implementierbar, Vorverarbeitung der Eingabe
- Operatoren:  $+$ ,  $?$ ,  $\{m, n\}$ ,  $\dots$   $\rightarrow$  Ebenfalls durch Vorverarbeitung lösbar, beziehungsweise durch Anpassung des Automaten
- *Lookahead* oder *lookbehind* sind leider nicht ohne Weiteres mit endlichen Automaten zu implementieren, da die zugrunde liegenden Grammatiken nicht mehr regulär wären.