Patrick Simianer
Visualisierung regulärer Ausdrücke

Patrick Simianer 2508483
2010-06-28

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

Gliederung

- Einleitung
  - Überlegungen
  - Protoypisches Vorgehen
  - Konkreter Aufbau
- Einfaches Parsing regulärer Ausdrücke
  - Recursive Descent-Methode
  - Thompson's Algorithmus
  - Beispiel
- Überführung NDEA zu einem DEA
  - ϵ-Abschluss
  - Beispiel
- Demo
- Weiterentwicklung

- Einleitung
  - Überlegungen
  - Protoypisches Vorgehen
  - Konkreter Aufbau
- Einfaches Parsing regulärer Ausdrücke
  - Recursive Descent-Methode
  - Thompson's Algorithmus
  - Beispiel
- Überführung NDEA zu einem DEA
  - ϵ-Abschluss
  - Beispiel
- Demo
- Weiterentwicklung

Visualisierung regulärer Ausdrücke

Wie soll die Visualisierung aussehen?

- Hervorheben von ***Matches*** oder **Gruppen** in einem String oder Text
- Darstellung und Simulation durch einen **Automaten**

- Es existieren bereits viele Implementierungen, basierend auf RE-Implementierung der jeweiligen Sprache
  $\longrightarrow$ keine "*step by step*"-Visualisierung möglich
- Grafische Umsetzung schwierig, eigene RE-Implementierung nötig
  $\longrightarrow$ jeder Schritt nachvollziehbar

Visualisierung regulärer Ausdrücke /2

- Wie können reguläre Ausdrücke möglichst einfach und effizient implementiert werden?
  - "Herkömmliche" **Backtracking**-Methode (*Perl*, *PCRE*)

  $\Rightarrow$ Direkte Konstruktion eines **endlichen Automaten**

- Soll der Automat dargestellt werden und wenn ja, wie?

  $\Rightarrow$ **Ja**, im besten Fall mit Animationen...

- In welcher Umgebung können alle Teile (1. Parser, 2. GUI, 3. Visualisierung) gut implementiert werden?

  $\Rightarrow$ **Browser**-basiert (1. *JavaScript*, 2. *HTML*, 3. *SVG*)

Protoypisches Vorgehen

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

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

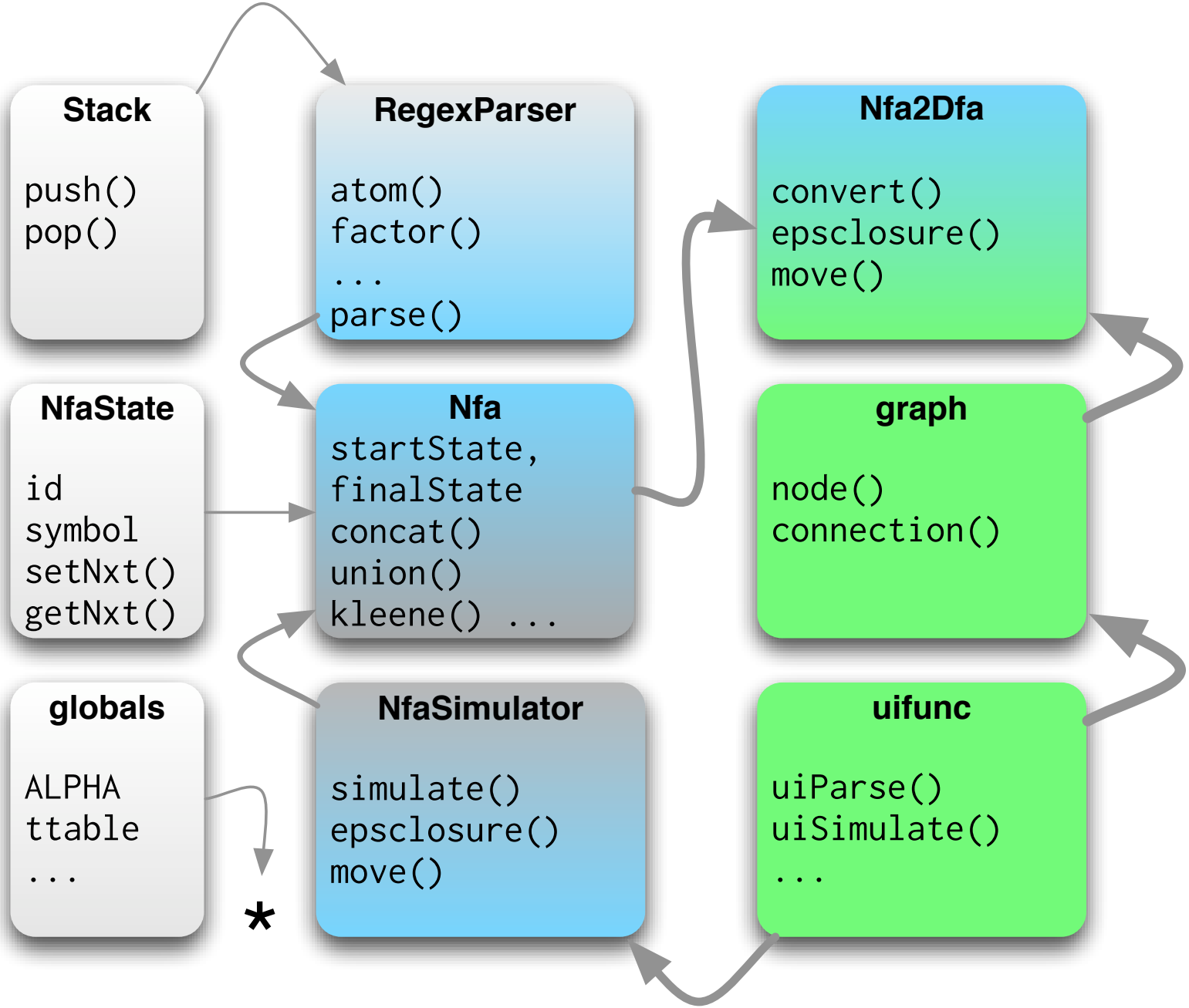


Abbildung: Konkreter System-Aufbau

- Einleitung
  - Überlegungen
  - Protoypisches Vorgehen
  - Konkreter Aufbau
- Einfaches Parsing regulärer Ausdrücke
  - Recursive Descent-Methode
  - Thompson's Algorithmus
  - Beispiel
- Überführung NDEA zu einem DEA
  - ϵ-Abschluss
  - Beispiel
- Demo
- Weiterentwicklung

Recursive Descent-Methode

Grammatik:

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

Code:

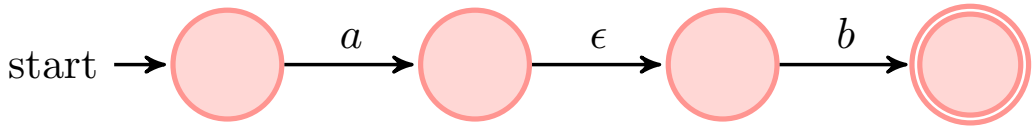
```
RegexParser.prototype.expr = function() {
    var nfa = this.term();
    if (this.trymatch('|')) {
        return nfa.union(this.expr());
    };
    return nfa;
};
```

- Nahezu direktes Übersetzen einer Grammatik[1] in den Quelltext des Parsers (LL(1))
- $\forall$ Nichtterminale $\exists$ Funktion, welche die rechte Seite der jeweiligen Regel behandelt
- Direkte Erzeugung des NDEA, mittels Konstruktion nach Thompson
- Max. $2m$ Zustände, $4m$ Transitionen (m Länge des Alphabets)

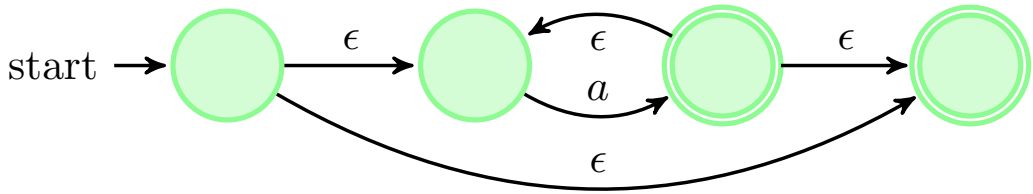
[1] keine Links-Rekursionen, sonst: Endlosschleife

Thompson's Algorithmus

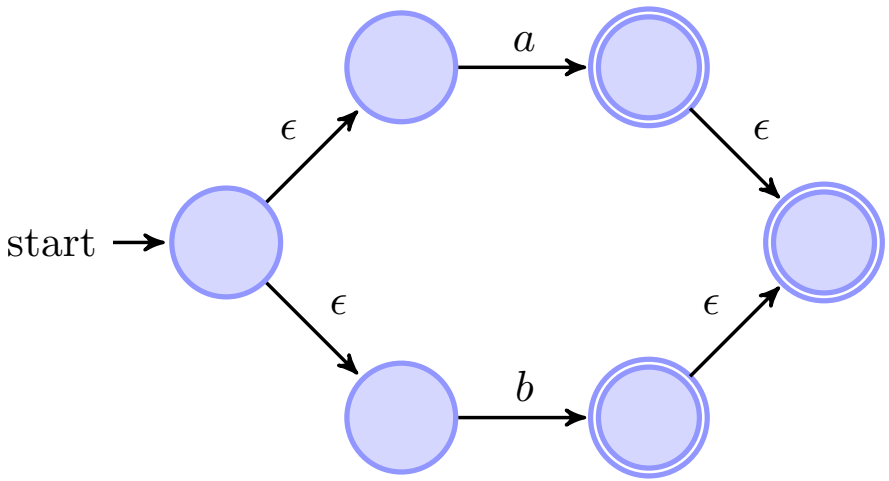
Konkatenation: `ab`



Hülle: `a*`

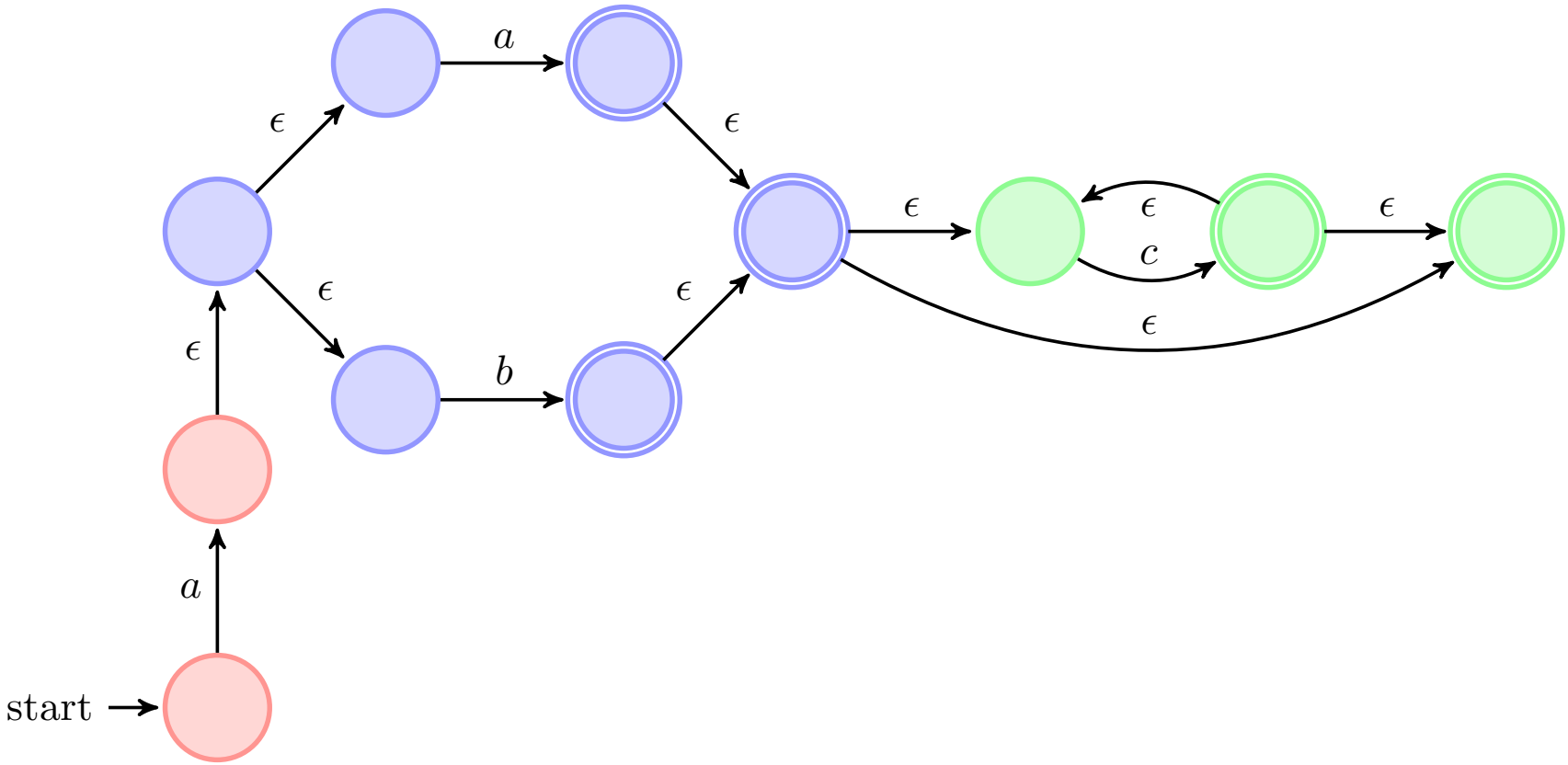


Vereinigung: `(a|b)`



Thompson's Algorithmus: Beispiel

Regulärer Ausdruck: `a(a|b)c*`



- Einleitung
  - Überlegungen
  - Protoypisches Vorgehen
  - Konkreter Aufbau
- Einfaches Parsing regulärer Ausdrücke
  - Recursive Descent-Methode
  - Thompson's Algorithmus
  - Beispiel
- Überführung NDEA zu einem DEA
  - ϵ-Abschluss
  - Beispiel
- Demo
- Weiterentwicklung

Einleitung

Warum den erzeugten NDEA in einen DEA überführen?

- *trade-off*:

Platzbedarf	−NDEA	+**DEA**
Erstellungszeit	+**NDEA**	−DEA
Ausführungszeit	−NDEA	+**DEA**

- NDEAs[2] umfassen für gewöhnlich sehr viele Zustände, die Darstellung eines DEA ist praktikabler

[2] insbesondere die hier erzeugten

ϵ-Abschluss

Pseudo-Code

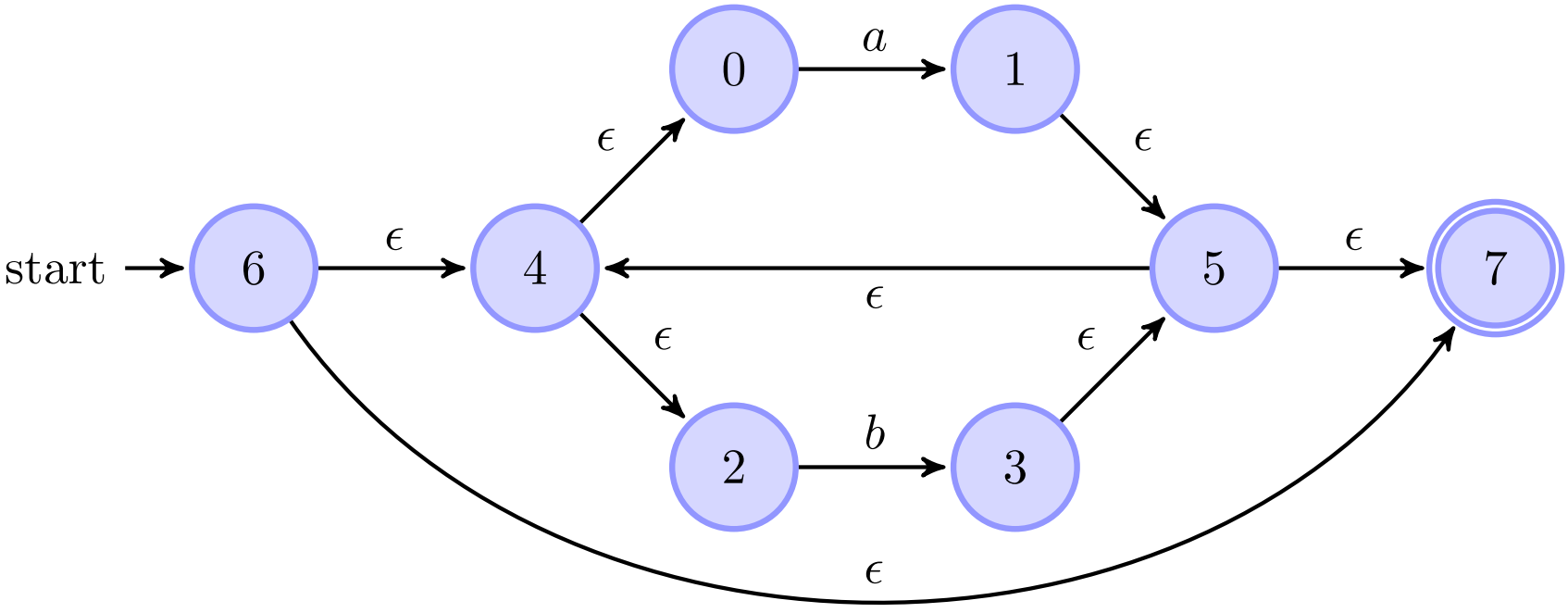
```
epsclosure(dState): stack s
foreach nState in dState {
 s.push(nState)
}
while s not empty {
 nState1 = s.pop()
 foreach nState1 e> nState2 {
  if nState2 not in dState {
   dState.add(nState2)
   s.push(nState2)
  }
 }
}
return dState
```

```
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
```

- $\forall p \in Q : E(\{p\}) = \{q \in Q : p \rightarrow_\epsilon q\}$
- **Laufzeit:** $O(nm^2)$ (bei Vorberechung aller ϵ-Abschlüsse: $O(m)$)

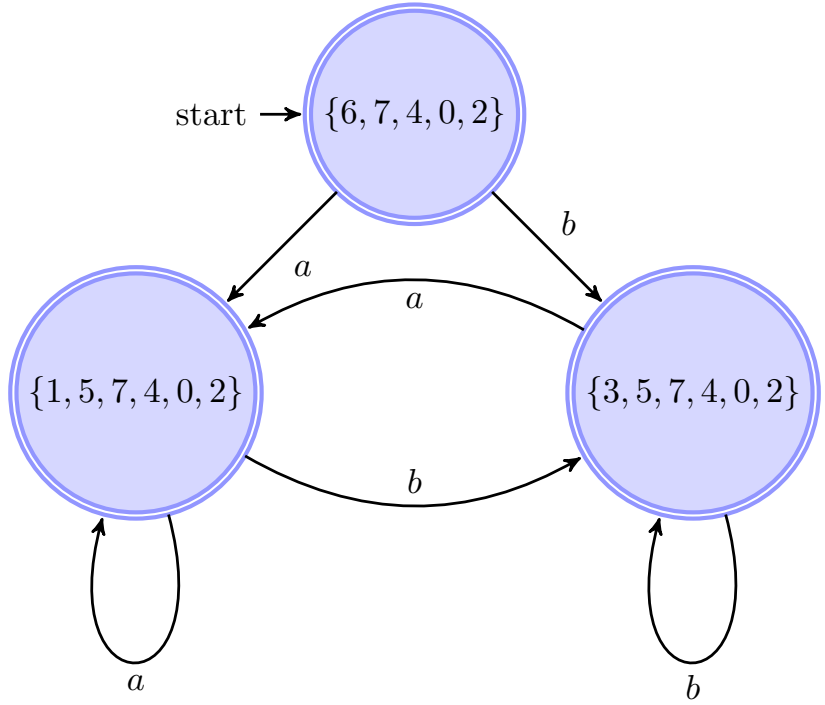
NDEA $\longrightarrow$ DEA Beispiel

Regulärer Ausdruck: `(a|b)*`



NDEA $\longrightarrow$ DEA Beispiel /2

Dfa ID	Symbol	$\longrightarrow$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}



Literatur I

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`)

- Einleitung
  - Überlegungen
  - Protoypisches Vorgehen
  - Konkreter Aufbau
- Einfaches Parsing regulärer Ausdrücke
  - Recursive Descent-Methode
  - Thompson's Algorithmus
  - Beispiel
- Überführung NDEA zu einem DEA
  - ϵ-Abschluss
  - Beispiel
- Demo
- Weiterentwicklung

Demo

- Einleitung
  - Überlegungen
  - Protoypisches Vorgehen
  - Konkreter Aufbau
- Einfaches Parsing regulärer Ausdrücke
  - Recursive Descent-Methode
  - Thompson's Algorithmus
  - Beispiel
- Überführung NDEA zu einem DEA
  - ϵ-Abschluss
  - Beispiel
- Demo
- Weiterentwicklung

Weiterentwicklung

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