

Vorlesung Compilerbau (SoSe2018)

Teil 3: Lexikalische Analyse




Compilerbau Lexikalische Analyse

Vorlesung des BA-Studiums
Prof. Johann Christoph Freytag, Ph.D.
Institut für Informatik, Humboldt-Universität zu Berlin
SoSe2018

Bitte Handys ausschalten! Danke.

© Prof. J.C. Freytag, Ph.D. 3.1



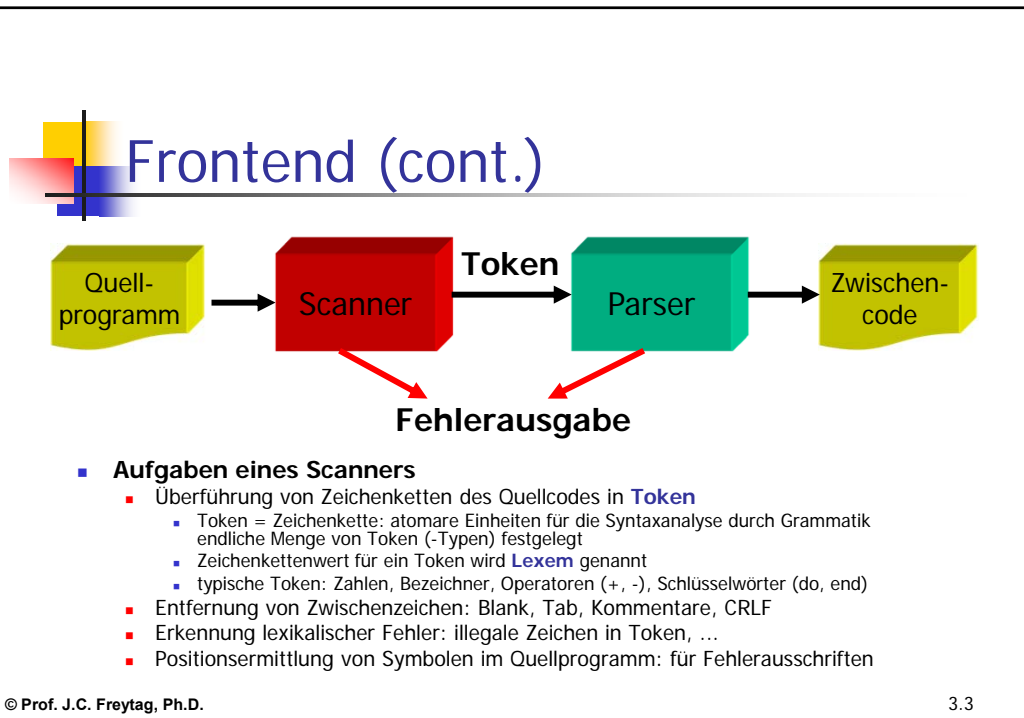
Überblick

- Reguläre Ausdrücke
- Endliche Automaten
 - Deterministisch, nicht-deterministisch
 - Überführung
- Erzeugen und Erkennen regulärer Ausdrücke
- Architektur eines Scanners
- Praktische Aspekte eines Scanners

© Prof. J.C. Freytag, Ph.D. 3.2

Vorlesung Compilerbau (SoSe2018)

Teil 3: Lexikalische Analyse



Muster und Sätze

Scanner erkennt (atomare) Einheiten der Syntax

- Beispiel:
 - "Max und Moritz"
 - wird überführt in
 - <Max> <und> <Moritz>
- Idee:
 - Zeichen in "sinvolle" Zeichenketten überführen, die wiederum Teile eines Satzes bilden

© Prof. J.C. Freytag, Ph.D. 3.4



Muster und Sätze

Scanner erkennt (atomare) Einheiten der Syntax

■ **Beispiel:**

```
float match0(char *s) /*find a zero*/
{if (Istrncmp (s, "0.0", 3))
    return 0.;
}
```

wird z.B. überführt in:

```
FLOAT ID(match0) LPAREN CHAR STAR ID(s) RPAREN
LBRACE IF LPAREN BANG ID(strncmp) LPAREN ID(s)
COMMA STRING(0.0) COMMA NUM(3) RPAREN RPAREN
RETURN REAL(0.0) SEMI RBRACE EOF
```



Muster und Sätze

Scanner erkennt (atomare) Einheiten der Syntax

■ **Beispiel:**

```
float match0(char *s) /*find a zero*/
{if (Istrncmp (s, "0.0", 3))
    return 0.;
}
```

wird z.B. überführt in:

```
FLOAT ID(match0) LPAREN CHAR STAR ID(s) RPAREN
LBRACE IF LPAREN BANG ID(strncmp) LPAREN ID(s)
COMMA STRING(0.0) COMMA NUM(3) RPAREN RPAREN
RETURN REAL(0.0) SEMI RBRACE EOF
```



Muster und Sätze

Scanner erkennt (atomare) Einheiten der Syntax

■ **Beispiel:**

```
float match0(char *s) /*find a zero*/  
{if (strcmp (s, "0.0", 3))  
    return 0.;  
}
```

wird z.B. überführt in:

```
FLOAT ID(match0) LPAREN CHAR STAR ID(s) RPAREN  
LBRACE IF LPAREN BANG ID(strcmp) LPAREN ID(s)  
COMMA STRING(0.0) COMMA NUM(3) RPAREN RPAREN  
RETURN REAL(0.0) SEMI RBRACE EOF
```



Muster und Sätze

Scanner erkennt (atomare) Einheiten der Syntax

■ **Beispiel:**

```
float match0(char *s) /*find a zero*/  
{if (strcmp (s, "0.0", 3))  
    return 0.;  
}
```

wird z.B. überführt in:

```
FLOAT ID(match0) LPAREN CHAR STAR ID(s) RPAREN  
LBRACE IF LPAREN BANG ID(strcmp) LPAREN ID(s)  
COMMA STRING(0.0) COMMA NUM(3) RPAREN RPAREN  
RETURN REAL(0.0) SEMI RBRACE EOF
```



Muster und Sätze

Scanner erkennt (atomare) Einheiten der Syntax

■ **Beispiel:**

```
float match0(char *s) /*find a zero*/  
{if (strcmp (s, "0.0", 3))  
    return 0.;  
}
```

wird z.B. überführt in:

```
FLOAT ID(match0) LPAREN CHAR STAR ID(s) RPAREN  
LBRACE IF LPAREN BANG ID(strcmp) LPAREN ID(s)  
COMMA STRING(0.0) COMMA NUM(3) RPAREN RPAREN  
RETURN REAL(0.0) SEMI RBRACE EOF
```



Muster und Sätze

Scanner erkennt (atomare) Einheiten der Syntax

■ **Beispiel:**

```
float match0(char *s) /*find a zero*/  
{if (strcmp (s, "0.0", 3))  
    return 0.;  
}
```

wird z.B. überführt in:

```
FLOAT ID(match0) LPAREN CHAR STAR ID(s) RPAREN  
LBRACE IF LPAREN BANG ID(strcmp) LPAREN ID(s)  
COMMA STRING(0.0) COMMA NUM(3) RPAREN RPAREN  
RETURN REAL(0.0) SEMI RBRACE EOF
```



Muster und Sätze

Scanner erkennt (atomare) Einheiten der Syntax

■ **Beispiel:**

```
float match0(char *s) /*find a zero*/  
{if (strcmp (s, "0.0", 3))  
    return 0.;  
}
```

wird z.B. überführt in:

```
FLOAT ID(match0) LPAREN CHAR STAR ID(s) RPAREN  
LBRACE IF LPAREN BANG ID(strcmp) LPAREN ID(s)  
COMMA STRING(0.0) COMMA NUM(3) RPAREN RPAREN  
RETURN REAL(0.0) SEMI RBRACE EOF
```



Muster und Sätze

Scanner erkennt (atomare) Einheiten der Syntax

■ **Beispiel:**

```
float match0(char *s) /*find a zero*/  
{if (strcmp (s, "0.0", 3))  
    return 0.;  
}
```

wird z.B. überführt in:

```
FLOAT ID(match0) LPAREN CHAR STAR ID(s) RPAREN  
LBRACE IF LPAREN BANG ID(strcmp) LPAREN ID(s)  
COMMA STRING(0.0) COMMA NUM(3) RPAREN RPAREN  
RETURN REAL(0.0) SEMI RBRACE EOF
```



Muster und Sätze (cont.)

- Einfach:
 - Zwischenzeichen (white spaces)
 - `<WS> ::= <WS> ' ' | <WS> '\t' | ' ' | '\t'`
 - Schlüsselwörter und Operatoren
 - Als Muster zu spezifizieren: 'do' , 'end'
 - Kommentare:
 - "öffnende und schließende Klammern": `'/*'` und `'*/'`



Muster und Sätze (cont.)

- Schwerer zu erkennen sind Muster:
 - Identifikatoren
 - "Zeichen des Alphabets gefolgt von einem alphanumerischen Zeichen"
 - Zahlen:
 - Integer: Ziffern 0-9 beliebig häufig hintereinander
 - Decimals: Integer '.' Ziffern von 0-9
 - Real: (Integer oder Decimal) 'E' ('+' oder '-') gefolgt von beliebigen Ziffern
 - Complex: '(' Real ',' Real ')'
- Brauchen (mächtige?) Notation, um diese "Muster" zu spezifizieren



Operationen auf Sprachen

- Seien L und M beliebige Sprachen

Operation	Definition
Vereinigung von L und M: $L \cup M$	$L \cup M = \{s \mid s \in L \text{ oder } s \in M\}$
Konkatenation von L und M: LM	$LM = \{st \mid s \in L \text{ und } t \in M\}$
Kleene Stern von L: L^*	$L^* = \bigcup_{i=0}^{\infty} L^i$
Positive Hülle von L: L^+	$L^+ = \bigcup_{i=1}^{\infty} L^i$

© Prof. J.C. Freytag, Ph.D.

3.15



Reguläre Ausdrücke

Muster werden häufig als Ausdrücke einer **regulären Sprache** über einem Alphabet Σ definiert

- entweder durch **reguläre Grammatik** oder
- durch **reguläre Ausdrücke**

- Ziel:**
Charakterisierung einer (möglicherweise unendlichen) Sprache, die ein endliches Alphabet besitzt ($\Sigma = \text{ASCII}$)
 - per Klassifikation der gültigen Zeichenketten
 - ein regulärer Ausdruck repräsentiert eine Menge gültiger Zeichenketten

© Prof. J.C. Freytag, Ph.D.

3.16



Reguläre Ausdrücke (cont.)

Definition eines Regulären Ausdrucks (RA):

- **Symbol:** Ist $a \in \Sigma$, dann ist a ein RA der Sprache L , der die Menge $\{a\}$ spezifiziert ($L(a) = \{a\}$), d.h. " a " ist gültige Zeichenkette
- **Alternative:** Sind r und s RA, die $L(r)$ und $L(s)$ spezifizieren, dann ist $(r \mid s)$ ein RA, der die Sprache $L(r) \cup L(s)$ spezifiziert
- **Konkatenation:** Sind r und s RA, die $L(r)$ und $L(s)$ spezifizieren, dann ist $(r \cdot s)$ ein RA der die Sprache $L(r)L(s)$ spezifiziert

Beispiel: RA $((a \mid b) \cdot a)$ definiert " aa " und " ba " als gültige Zeichenketten



Reguläre Ausdrücke (cont.)

Definition eines Regulären Ausdrucks (RA) – cont.:

- **Epsilon:** ϵ (das leere Wort) ist ein RA der Sprache L , der die Menge $\{\epsilon\}$ spezifiziert
- **Beispiel:** RA $((a \mid b) \cdot \epsilon)$ definiert " a " und " b " als gültige Zeichenketten
- **Wiederholung:** Sei r RA, der $L(r)$ spezifiziert, dann ist die Kleene'sche Hülle $(r)^*$ ein RA, der die Sprache $L(r)^*$ spezifiziert.
 - ein String gehört zu $L(r)^*$, wenn er eine Verkettung von null oder mehr Strings darstellt, die gültige Strings vom RA r sind
- **Beispiel:** RA $((a \mid b) \cdot a)^*$ definiert die unendliche Menge von Strings $\{ "", "aa", "ba", "aaaa", "baaa", "aaba", "baba", "aaaaaa", \dots \}$



Reguläre Ausdrücke (cont.)

Definition eines Regulären Ausdrucks (RA) – cont.:

- **positive Wiederholung:** Sei r RA, der $L(r)$ spezifiziert, dann ist die positive Kleene'sche Hülle $(r)^+$ ein RA, der die Sprache $L(r)^+$ spezifiziert:
 - ein String gehört zu $L(r)^+$, wenn er eine Verkettung von mindestens einem oder mehr Strings darstellt, die gültige Strings vom RA r sind
- **Bemerkung:**
 - falls Präzedenzen für die Operatoren definiert werden, können die Klammern weggelassen werden
 - Kleene'scher Hüllenoperator hat Präzedenz über Vereinigung und Verkettung
 - Konkatenation $(.)$ hat Präzedenz über Alternative $(|)$



Beispiele für Reguläre Ausdrücke

- **Bezeichner**
 - $\text{letter} \rightarrow (a|b|c|\dots|z|A|B|C|\dots|Z)$
 - $\text{digit} \rightarrow (0|1|2|\dots|8|9)$
 - $\text{id} \rightarrow \text{letter} (\text{letter}|\text{digit})^*$
- **Zahl**
 - $\text{integer} \rightarrow (+|-|\epsilon) (0|(1|2|3|\dots|9) \text{digit}^*)$
 - $\text{decimal} \rightarrow \text{integer} . (\text{digit})^*$
 - $\text{real} \rightarrow (\text{integer} | \text{decimal}) E (+|-) \text{digit}^+$
 - $\text{complex} \rightarrow '(' \text{real} , \text{real} ')'$
- Zahlen können jedoch weit komplexer spezifiziert werden
- die meisten Token können mittels RA spezifiziert werden

Vorlesung Compilerbau (SoSe2018)

Teil 3: Lexikalische Analyse



Kurznotation Regulärer Ausdrücke

a	gewöhnliches Zeichen
ϵ	leere Zeichenkette
$s \mid r$	Alternative
$s \cdot r$	Verkettung
sr	andere Schreibweise für Verkettung
s^*	Wiederholung (null oder beliebig oft)
s^+	Wiederholung (einmal oder beliebig oft)
$s?$	optional Wiederholung (null oder einmal)
$[a-zA-Z]$	Zeichenalternativen
$.$	jedes beliebiges Zeichen (außer newline)
$"a. + *"$	String

© Prof. J.C. Freytag, Ph.D.

3.21



Algebraische Eigenschaften von RA

Axiome	Beschreibung
$r s = s r$	Kommutativität von Alternative
$r (s t) = (r s) t$	Assoziativität von Alternative
$(rs) t = r (st)$	Assoziativität von Verkettung
$r(s t) = rs rt$ $(s t)r = sr tr$	Distributivität von Verkettung und Alternative
$\epsilon r = r$ $r \epsilon = r$	Neutrales Element der Verkettung
$r^* = (r \epsilon)^*$	Beziehung zwischen $*$ und ϵ
$r^{**} = r^*$	$*$ ist idempotent

© Prof. J.C. Freytag, Ph.D.

3.22

Beispiele RA

- Sei $\Sigma = \{a, b\}$
 - $a|b$ beschreibt die Menge $\{a, b\}$
 - $(a|b)(a|b)$ beschreibt die Menge $\{aa, ab, ba, bb\}$, i.e. $(a|b)(a|b) = aa|ab|ba|bb$
 - a^* beschreibt die Menge $\{\epsilon, a, aa, aaa, \dots\}$
 - $(a|b)^*$ beschreibt die Menge aller Zeichenketten aus a's und b's (einschließlich ϵ); i.e. $(a|b)^* = (a^*b^*)^*$
 - $a|a^*b$ beschreibt die Menge $\{a, b, ab, aab, aaab, \dots\}$

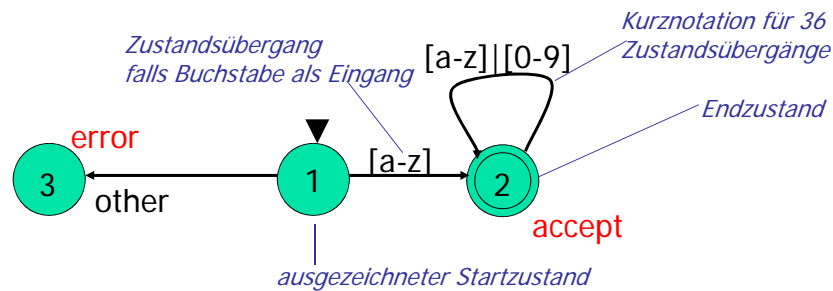
Frage...

- RA: beschreiben eine reguläre Sprache?
- Frage: Gibt es Formalismus, der den Prozess der Spracherkennung für RA erlaubt?
- Antwort: Es gibt diesen Formalismus
 - Endliche Automaten (engl. Finite Automata)
 - Deterministischer endlicher Automat (DFA)
 - Nicht-Deterministischer endlicher Automat (NFA)
 - Deterministisch:
 - Engl. Deterministic = bestimmt, begrenzt, festgelegt
 - Engl. To determine
 - Wurde entwickelt Michael O. Rabin (1959)



Deterministischer endlicher Aut.

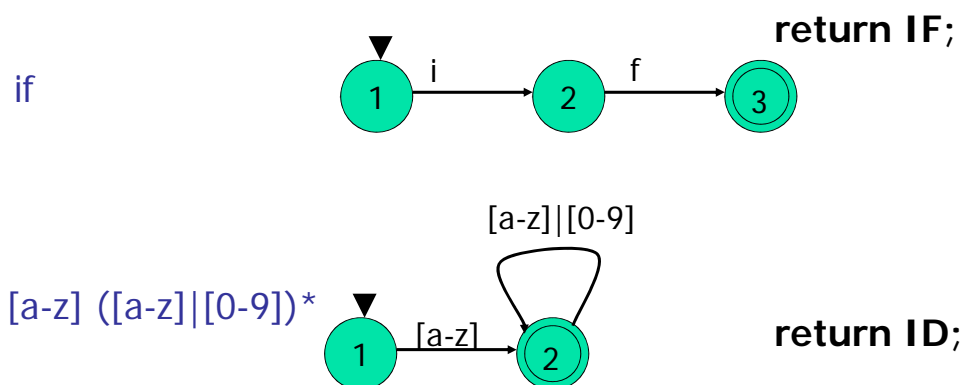
- Aus einem RA kann ein deterministischer endlicher Automat (engl. Deterministic Finite Automaton - DFA) konstruiert werden
- Beispiel: (Erkennungs-) Automat" für "identifizier":



© Prof. J.C. Freytag, Ph.D.

3.25

Endliche Automat für lexikalische Token

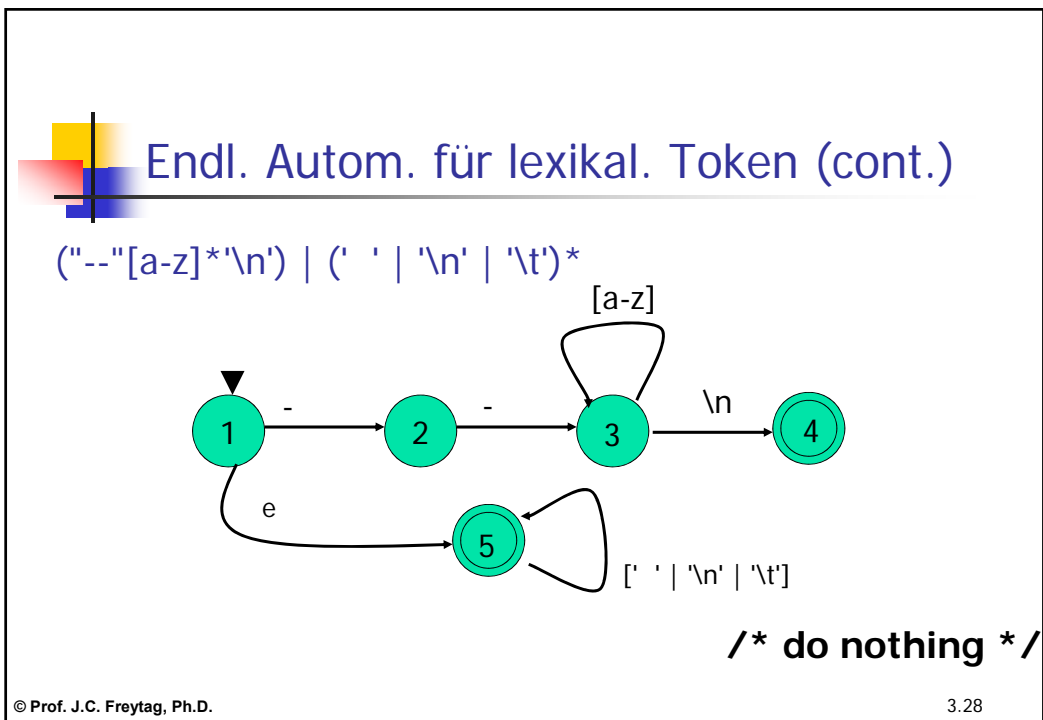
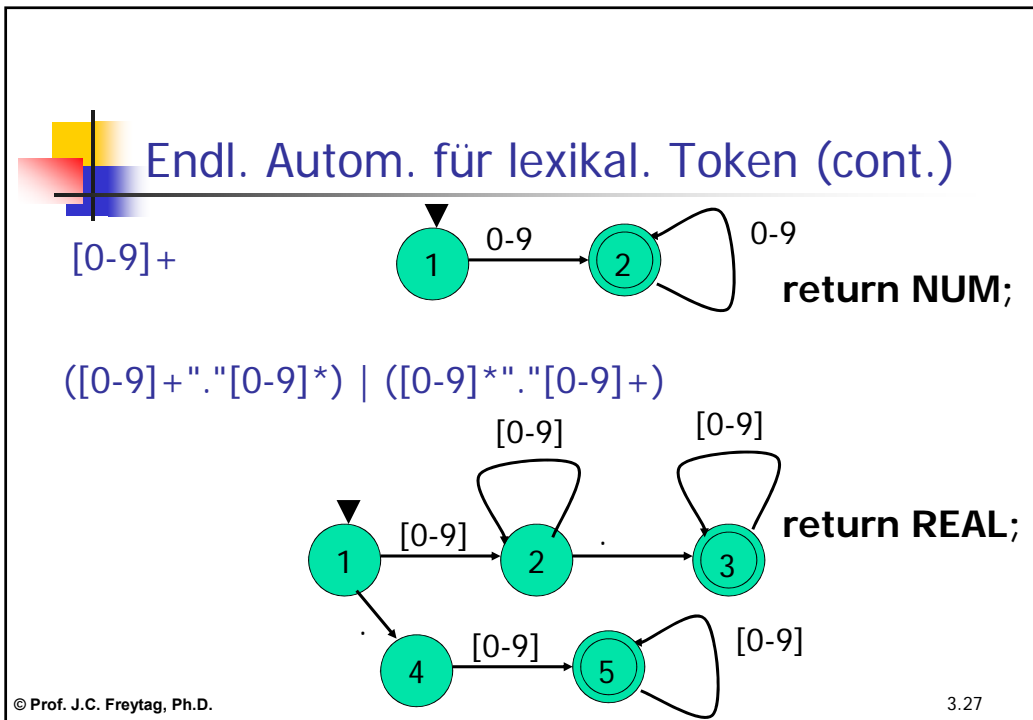


© Prof. J.C. Freytag, Ph.D.

3.26

Vorlesung Compilerbau (SoSe2018)

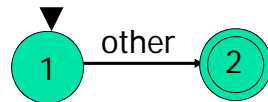
Teil 3: Lexikalische Analyse



Vorlesung Compilerbau (SoSe2018)

Teil 3: Lexikalische Analyse

Endl. Autom. für lexikal. Token (cont.)

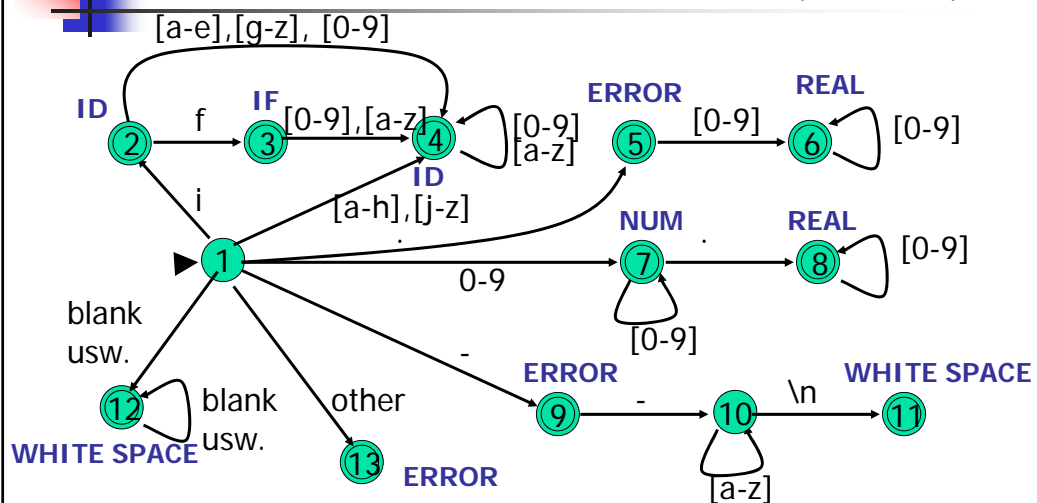


error();

© Prof. J.C. Freytag, Ph.D.

3.29

Kombin. von Einzelautomaten (ad-hoc)



© Prof. J.C. Freytag, Ph.D.

3.30



Arbeitsweise des Automaten

- Liegt in einem **beliebigen Zustand** (Startzustand, Zwischenzustand, Terminalzustand) ein **akzeptables Eingabezeichen** vor, wird
 - das Zeichen akzeptiert (d.h. konsumiert) und
 - der Folgezustand in Abhängigkeit des akzeptierten Zeichens angenommen.
- Liegt ein **nicht akzeptables Zeichen** vor, sind folgende alternative Verhaltensformen des Automaten (**im Unterschied zur theoretischen Definition**) möglich:
 - aktueller Zustand = **Startzustand** oder **beliebiger Zwischenzustand**
 - a) es gibt einen expliziten Übergang für "other" mit Fehlerbehandlungsprozedur
 - b) es gibt einen impliziten Übergang, der für den "Rest" in einen Fehlerzustand führt
 - aktueller Zustand = **ERROR-Endzustand**: unvollständige Token-Bildung, das Zeichen wird nicht konsumiert
 - explizite Fehlerbehandlungsprozedur
 - aktueller Zustand = **Endzustand (kein ERROR)**
Token- bzw. Eliminationstext-Bildung ist abgeschlossen, Ausgabe des Textes
 - das aktuelle Zeichen wird **nicht** konsumiert, der Automat stoppt.



Analyseprobleme

- Fragen
 - ist "if8" ein Bezeichner oder zwei Token: <if> <8>?
 - beginnt "if 89" mit einem Schlüsselwort oder einem Bezeichner?
- wichtige Regeln beim Scanner-Bau
 1. nächstes Token wird stets als **maximale mögliche** gültige Zeichenkette bestimmt
 2. Die Bestimmung der Token-Klasse erfolgt entsprechend einer **priorisierten Liste** (d.h. Reihenfolge der RA ist signifikant)
- Antworten
 - "if8" wird als Bezeichner erkannt: nach der 1.Regel
 - "if" in "if 89" wird als Schlüsselwort erkannt: nach der 2.Regel

Generierung des Scanner-Codes

- Ziel: **Nutzung** des DFA um ein DFA-Programm **abzuleiten**, das der Erkennungsfunktion des DFA entspricht
- Herausforderung: Kann Code für einen Scanner erzeugt werden, der unabhängig von der zu erkennenden Sprache ist?
 - Antwort: fast...
 - zwei Tabellen werden gebraucht:
 - **char_class**: Zeichen → Zeichenklasse
 - **next_state**: (Zeichenklasse, Zustand) → Zustand

© Prof. J.C. Freytag, Ph.D.

3.33

Tabellen für einen Scanner

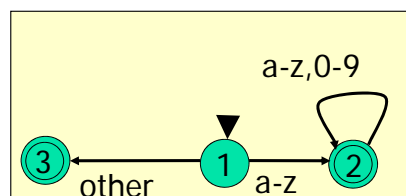
- zwei Tabellen steuern den Scanner
- Zeichenklasse (char_class):

	a-z	0-9	other
value	letter	digit	other

- Zustandsübergangstabelle (next_state):

	1	2	3
letter	2	2	-
digit	3	2	-
other	3	-	-

Pseudoklasse
(kontextabhängig)



- neue Sprache bedeutet neue Tabellen

© Prof. J.C. Freytag, Ph.D.

3.34

Pseudocode für einen Scanner

```
char ← next_char();
state ← 1; /* code for state 1 */
done ← false;
token_value ← " " /* empty string */
while (not done) {
  class ← char_class[char] ;
  state ← next_state[class, state] ;
  switch (state) {
    case 2: /* building an id */
      token_value ← concatenate(token_value, char);
      char ← next_char();
      break;
    case -: /* accept state */
      token_type ← identifier;
      done ← true;
      break;
    case 3: /* error */
      token_type ← error;
      done ← true;
      break;
  }
}
```

© Prof. J.C. Freytag, Ph.D.

3.35

Automatische Konstruktion

- "Scanner Generator" konstruiert automatisch Code aus einem RA
- Schritte:
 1. Konstruiere deterministischen endlichen Automaten (DFA)
 2. Wende Techniken zur Zustandsminimierung an
 3. Erzeuge Code automatisch für den Scanner (entweder tabellengetrieben oder Code direkt)
- **Wichtig:** Erzeugung einer "Standardschnittstelle" zum Parser
- **(F)Lex** ist ein "Scanner Generator" unter UNIX
 - Generiert Code für einen Scanner
 - Erzeugt Macro-Definitionen für jedes Token (die im Parser genutzt werden können)

© Prof. J.C. Freytag, Ph.D.

3.36



Grammatiken für reguläre Sprachen

- Frage: Gibt es Einschränkungen auf die Form der Grammatik, so dass diese ausschließlich reguläre Sprachen beschreiben?
- Beweisbar:
Für jeden regulären Ausdruck r gibt es eine Grammatik G mit $L(r) = L(G)$
- Grammatiken, die Mengen regulärer Ausdrücke erzeugen, heißen reguläre Grammatiken:
- Die Produktionen (Regeln) haben die folgende Form:
 1. $A \rightarrow aA$
 2. $A \rightarrow a$ A ist ein Nicht-Terminalsymbol, a ein Terminalsymbol
- Reguläre Grammatiken heißen auch "Typ-3" Grammatiken

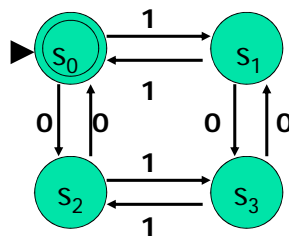
© Prof. J.C. Freytag, Ph.D.

3.37



Reguläre Sprachen - FAs

- Beispiel:
 - Menge aller Zeichenketten, die aus einer geraden Anzahl an Nullen und Einsen bestehen



- Der reguläre Ausdruck ist:
 $(00|11)^*((01|10)(00|11)^*(01|10)(00|11)^*)^*$

© Prof. J.C. Freytag, Ph.D.

3.38

Kürzerer Ausdruck.....

- Original

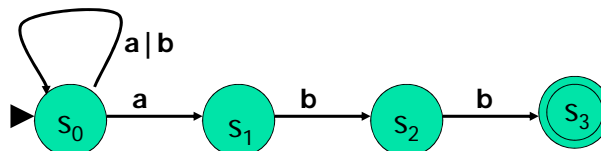
- $(00|11)^*((01|10)(00|11)^*(01|10)(00|11)^*)^*$

- Umgeschrieben

- $(00|11|(01|10)(00|11)^*(01|10))^*$

Übergänge in einem NFA

- Welches ist der FA der den RA $(a|b)^*abb$ erkennt??



- Zustand S_0 hat mehrere Übergänge für a
⇒ nicht-deterministischer endlicher Automat (NFA)

$0 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{b} 3 \dots$
 $0 \xrightarrow{a} 0 \xrightarrow{a} 0 \xrightarrow{b} 0 \xrightarrow{b} 0$
 viele Übergänge möglich, aber nur einer führt zur Akzeptanz

- Zustand s_0 hat mehrere Übergänge auf a
⇒ nicht-deterministischer endlicher Automat (NFA)



Endliche Automaten (Finite Aut.)

- Def. nicht-deterministischer endlicher Automat (NFA):
 - Eine Menge von Zuständen $\mathbf{S} = \{s_0, \dots, s_n\}$
 - Eine Menge von Eingabesymbolen (Alphabet) Σ
 - Eine (Transitions-) Übergangsfunktion \mathbf{T} , die die Zustands-Symbol-Paare in Zustandsmengen überführt
$$\mathbf{T}: \mathbf{S} \times \Sigma \rightarrow 2^{\mathbf{S}}$$
 - Ein ausgezeichnete Startzustand s_0
 - Eine Menge von akzeptierenden (End-) Zuständen \mathbf{F} $\Rightarrow \text{NFA} = (\mathbf{S}, \Sigma, \mathbf{T}, s_0, \mathbf{F})$
- Ein NFA akzeptiert die Zeichenkette x iff
 - ein Pfad durch den Transitionsgraphen existiert, der in s_0 beginnt und in einem Endzustand so endet, dass die Sequenz der durchlaufenen Kanten genau x ergeben

© Prof. J.C. Freytag, Ph.D.

3.41



Endliche Autom. (Finite Aut.) (2)

- Ein deterministischer endlicher Automat (DFA) ist ein Spezialfall:
 - Es gibt keine ε -Transitionen (Übergang ohne Zeichen)
 - Für jeden Zustand s und jedes Eingabesymbol a existiert maximal eine Kante a , die s verlässt, i.e. für jedes Zustands-Symbol-Paar gibt es maximal einen neuen Zustand
- Ein DFA akzeptiert die Zeichenkette x iff
 - ein Pfad durch den Transitionsgraph existiert, der in s_0 beginnt und in einem Endzustand so endet, dass die Sequenz der durchlaufenen Kanten genau x ergeben

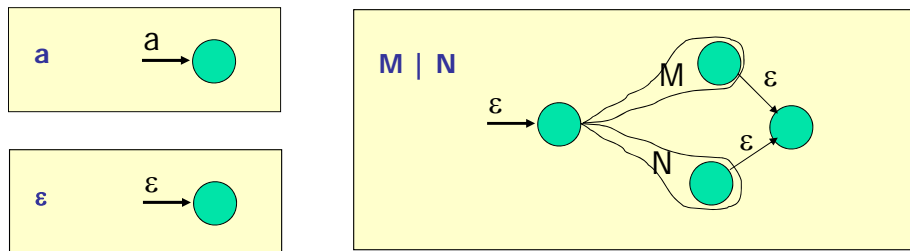
© Prof. J.C. Freytag, Ph.D.

3.42

Abbildung von RAs zu NFAs

Wie konstruiert man einen NFA?

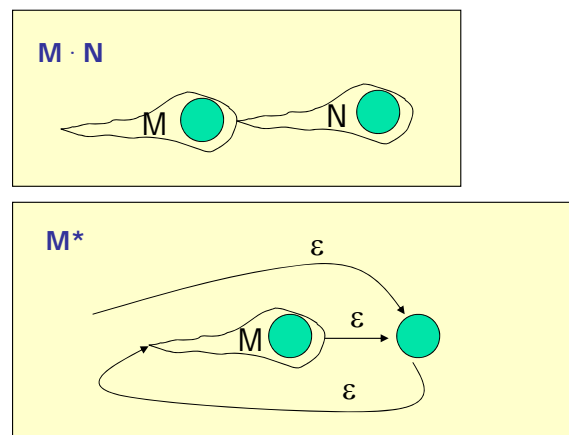
- man baut Einzelautomaten und
- fügt diese nach Vorschrift zusammen (beim DFA: nur ad-hoc)



© Prof. J.C. Freytag, Ph.D.

3.43

Abbildung von RAs zu NFAs (2)

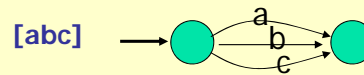


© Prof. J.C. Freytag, Ph.D.

3.44

Abbildung von RAs zu NFAs (3)

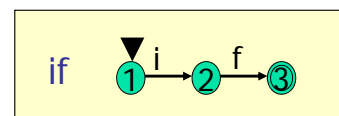
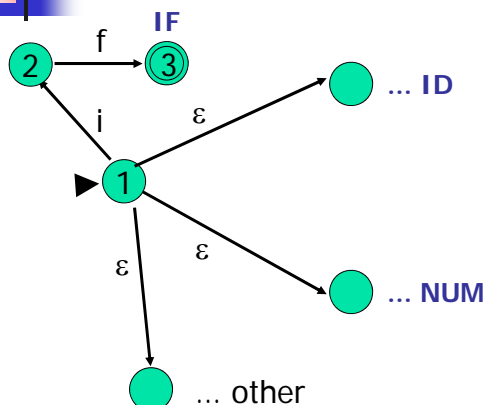
M^+ konstruiert als $M \cdot M^*$



$M?$ konstruiert als $M \mid \varepsilon$

$'abc'$ konstruiert als $a \cdot b \cdot c$

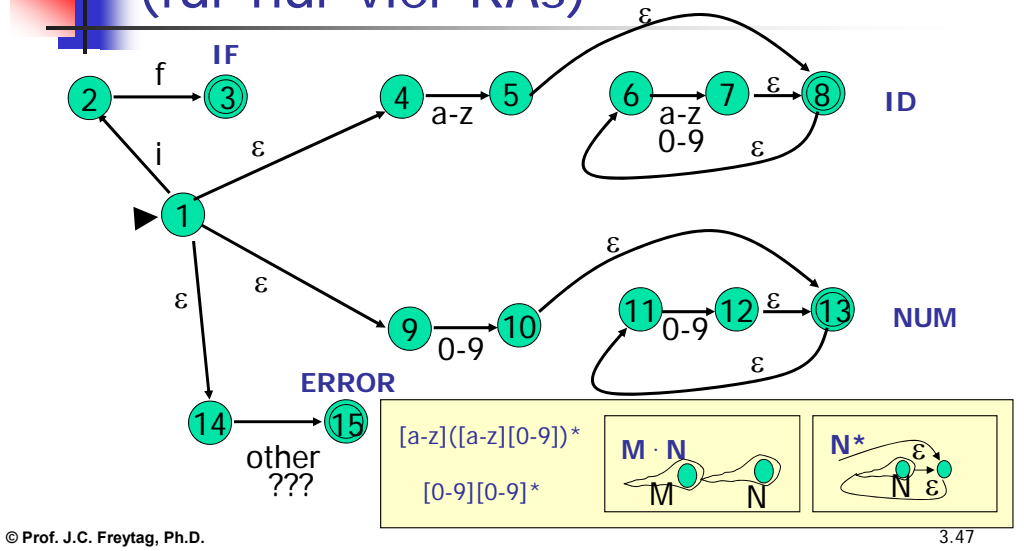
Konstruktion eines NFAs (für nur vier RAs)



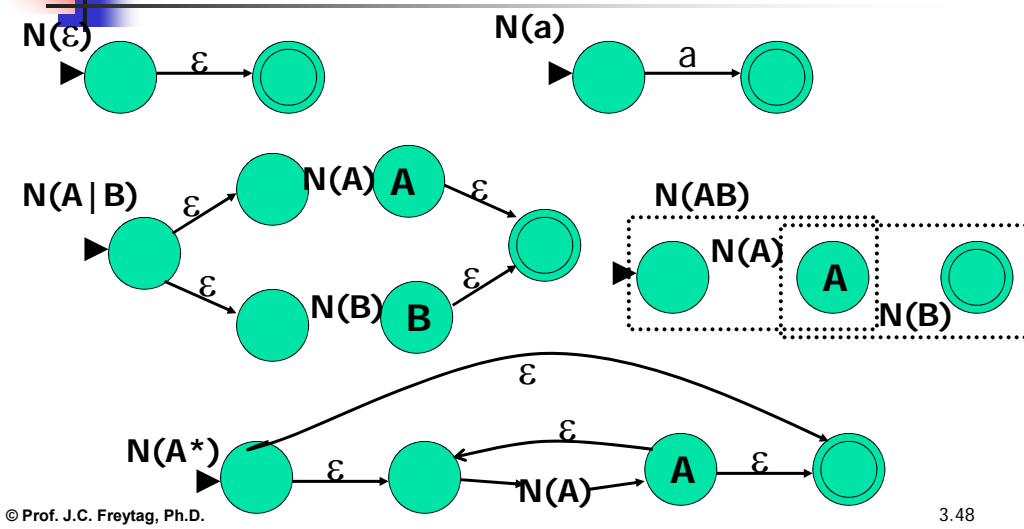
Vorlesung Compilerbau (SoSe2018)

Teil 3: Lexikalische Analyse

Konstruktion eines NFAs (für nur vier RAs)

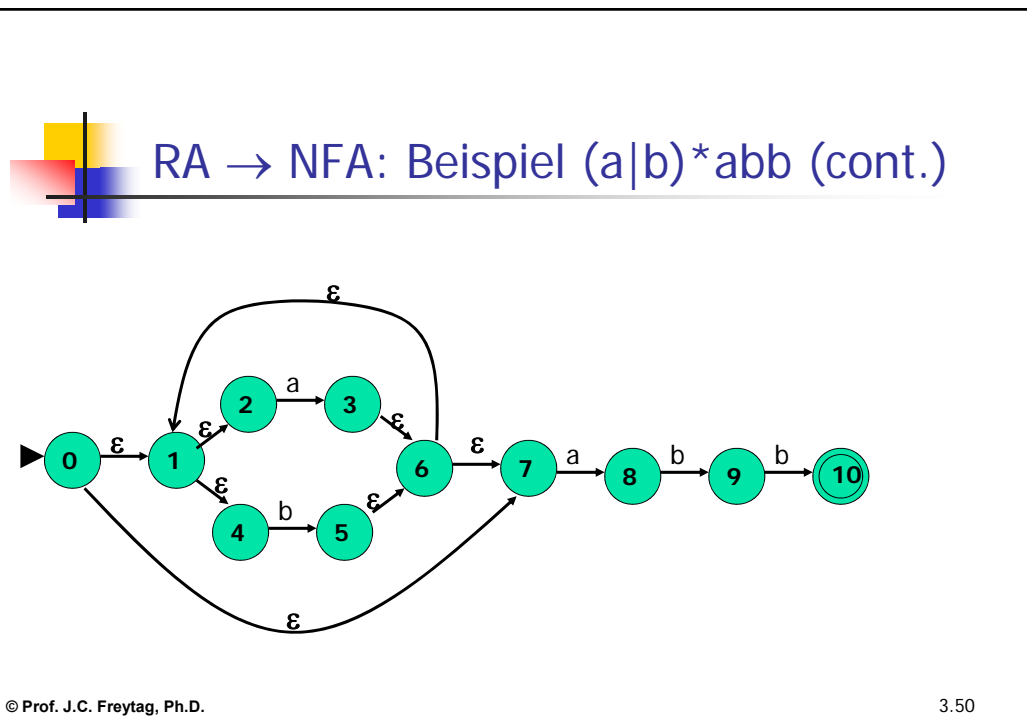
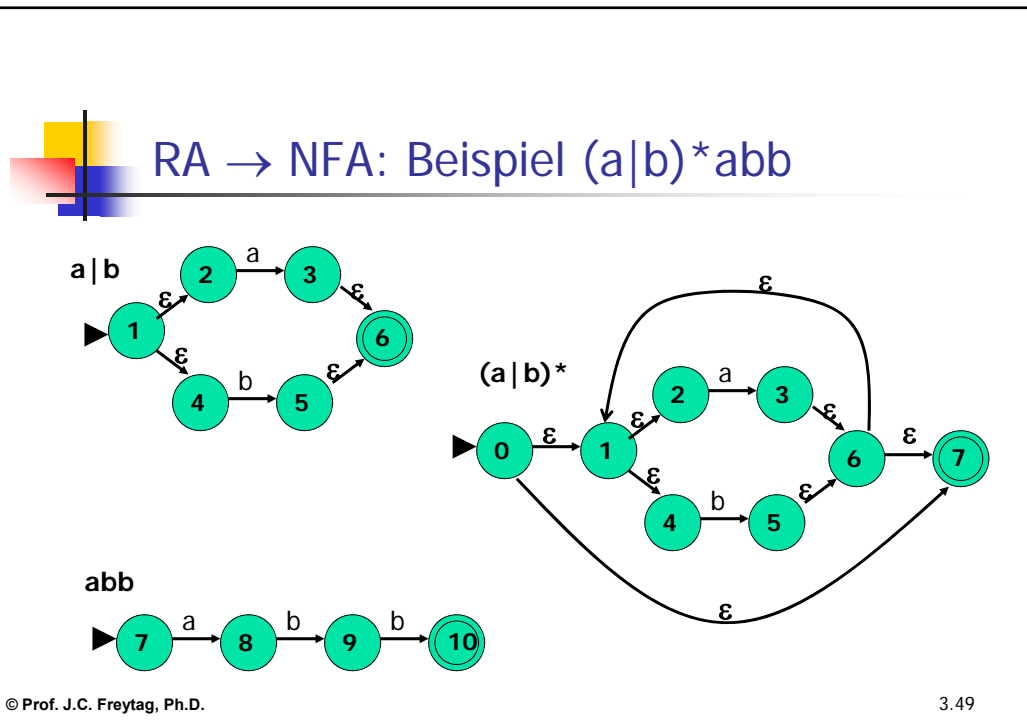


RA \rightarrow NFA: Konstruktion (mal anders...)



Vorlesung Compilerbau (SoSe2018)

Teil 3: Lexikalische Analyse





Implementierung eines NFAs

- es ist einfacher, aus einem RA einen NFA als einen DFA zu konstruieren
- aber: Implementation eines NFA ist im Allgemeinen nicht effizient möglich
 - z.B. Einsatz von Backtracking-Verfahren
 - **problematisch** sind alternative Übergänge aus einem Zustand mit spontanen (ϵ -Transitionen) und Eingabe-getriebenen Übergängen
- **besser**: Umwandlung eines konstruierten NFA in einen DFA

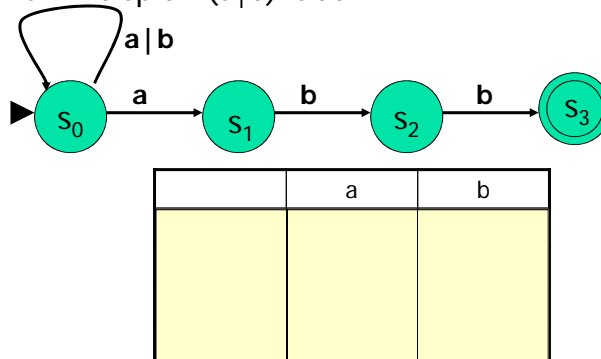


Eigenschaften von FAs

- **DFAs und NFAs sind äquivalent**
 - D.h. für jeden NFA gibt es einen DFA der die gleiche Sprache erkennt
- Warum?
 - DFAs sind eine Untermenge von NFAs
 - **Jeder NFA kann in einen DFA konvertiert werden**, indem die gleichzeitig erreichten Zustände des NFA im DFA "simuliert" werden:
 - Jeder Zustand im DFA korrespondiert mit einer Menge an Zuständen im NFA
 - Möglicherweise exponentiell viele Zustände im Vergleich zum NFA (Warum??)

Überführung: NFA \Rightarrow DFA

- Verfahren der Teilmengenkonstruktion
am Beispiel: $(a|b)^*abb$

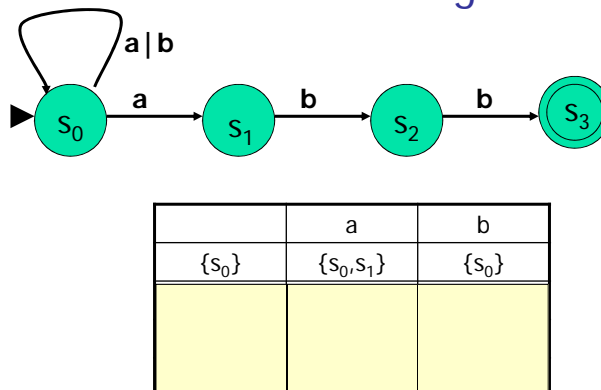


© Prof. J.C. Freytag, Ph.D.

3.53

Überführung NFA \Rightarrow DFA

- Verfahren der Teilmengenkonstruktion



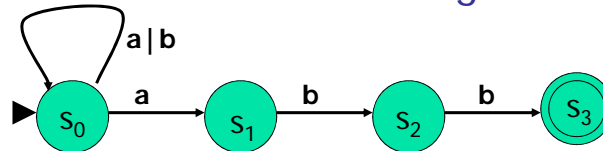
© Prof. J.C. Freytag, Ph.D.

3.54



Überführung NFA \Rightarrow DFA

■ Verfahren der Teilmengenkonstruktion



	a	b
$\{s_0\}$	$\{s_0, s_1\}$	$\{s_0\}$
$\{s_0, s_1\}$	$\{s_0, s_1\}$	$\{s_0, s_2\}$

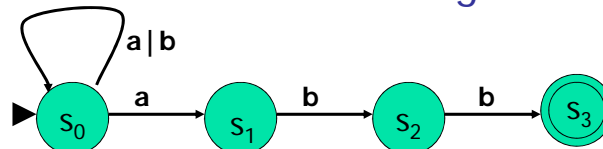
© Prof. J.C. Freytag, Ph.D.

3.55



Überführung NFA \Rightarrow DFA

■ Verfahren der Teilmengenkonstruktion



	a	b
$\{s_0\}$	$\{s_0, s_1\}$	$\{s_0\}$
$\{s_0, s_1\}$	$\{s_0, s_1\}$	$\{s_0, s_2\}$
$\{s_0, s_2\}$	$\{s_0, s_1\}$	$\{s_0, s_3\}$

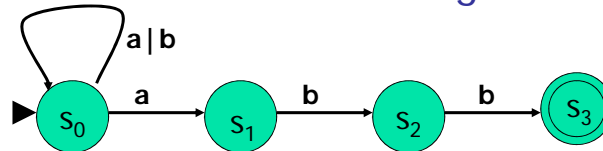
© Prof. J.C. Freytag, Ph.D.

3.56



Überführung NFA \Rightarrow DFA

- Verfahren der Teilmengenkonstruktion



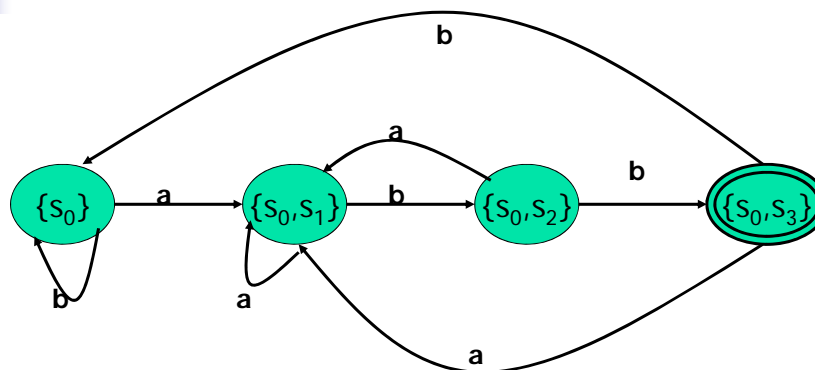
	a	b
$\{s_0\}$	$\{s_0, s_1\}$	$\{s_0\}$
$\{s_0, s_1\}$	$\{s_0, s_1\}$	$\{s_0, s_2\}$
$\{s_0, s_2\}$	$\{s_0, s_1\}$	$\{s_0, s_3\}$
$\{s_0, s_3\}$	$\{s_0, s_1\}$	$\{s_0\}$

© Prof. J.C. Freytag, Ph.D.

3.57



Überführung NFA \Rightarrow DFA (2)

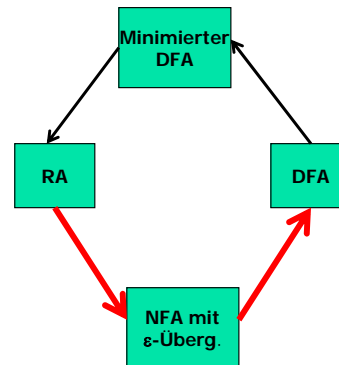


© Prof. J.C. Freytag, Ph.D.

3.58

Konstruktion eines DFA aus RA

- RA → NFA mit ε -Übergängen
 - Erzeuge NFA für jeden Teilausdruck
 - Verbinde mit ε -Übergängen
- NFA mit ε -Übergängen → DFA
 - Durch "Teilmengen"-Konstruktion
- DFA → **minimierter** DFA
 - Führe "kompatible" Zustände zusammen
- DFA → RA
 - Konstruiere durch "Lösen von Gleichungen"



© Prof. J.C. Freytag, Ph.D.

3.59

NFA → DFA: Teilmengenkonstruktion

Eingabe: NFA N

Ausgabe: Ein DFA D mit Zuständen D_Z und Transitionen D_T mit $L(D) = L(N)$

Methode: Sei s ein Zustand in N und T eine Menge von Zuständen in N , dann benutze die folgenden Operationen:

Operation	Definition
ε -Hülle(s)	Menge aller NFA-Zustände, die vom NFA-Zustand s mit ε -Übergängen allein erreicht werden können
ε -Hülle(T)	Menge aller NFA-Zustände, die von einem NFA-Zustand s aus T allein mit ε -Übergängen erreicht werden können
$\text{move}(T, a)$	Menge aller NFA-Zustände, zu denen ein Übergang für das Eingabesymbol a von einem NFA-Zustand s aus T führt

© Prof. J.C. Freytag, Ph.D.

3.60

Vorlesung Compilerbau (SoSe2018)

Teil 3: Lexikalische Analyse



NFA → DFA: Teilmengenkonstruktion (2)

Idee des Verfahrens:

- jeder Zustand des DFA entspricht einer Menge von Zuständen des NFA
- DFA merkt sich in seinen Zuständen alle möglichen Zustände, in denen sich der NFA nach Lesen von Eingabezeichen befinden kann, d.h., der DFA befindet sich nach Eingabe von $a_1 a_2 \dots a_n$ in einem Zustand, der die Teilmenge T aller Zustände vom NFA bestimmt, die vom Startzustand des NFA entlang eines mit $a_1 a_2 \dots a_n$ markierten Pfades erreicht werden kann

Bemerkung

- Die Anzahl von DFA-Zuständen kann exponentiell größer sein, als die des NFA (Dieser Fall tritt aber in der Praxis kaum auf)



NFA → DFA: Teilmengenkonstr. (2)

- **Algorithmus:**

Anfangsschritt: ϵ -Hülle(s_0) bildet den Startzustand von D

Füge Zustände T aus ϵ -Hülle(s_0) als unmarkierte Zustände zu D_Z hinzu

while es existieren noch unmarkierte Zustände T in D_Z **do**

markiere T ;

for jedes Eingabesymbol a **do**

$U = \epsilon$ -Hülle(move(T, a));

if U nicht in D_Z **then** füge U als unmarkiert D_Z hinzu **endif**

$D_T[T, a] = U$

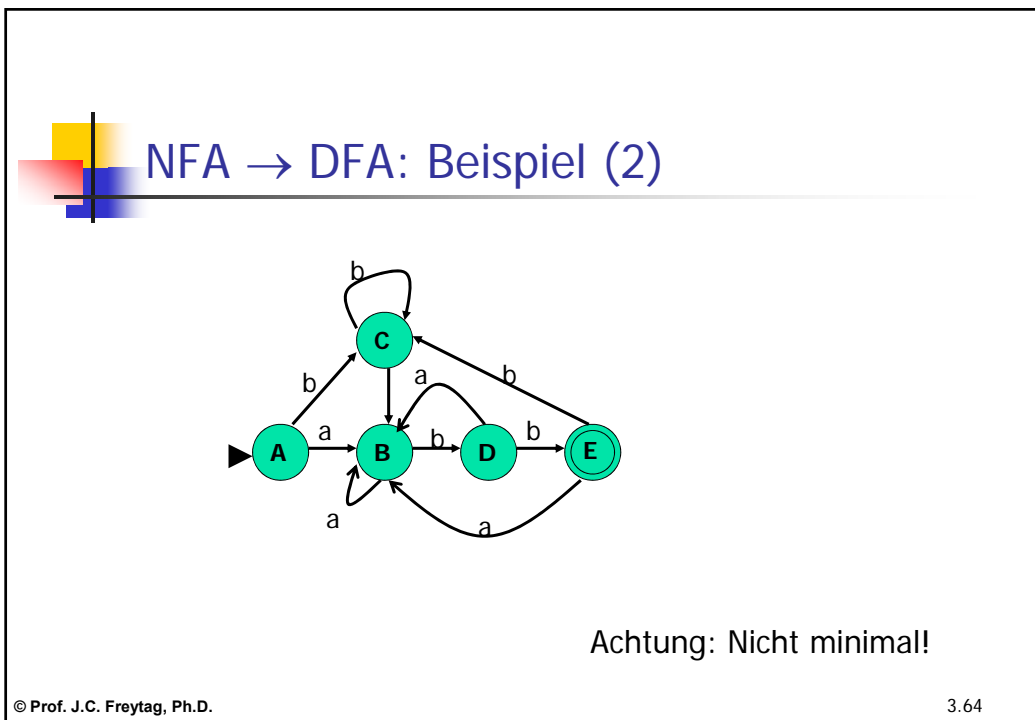
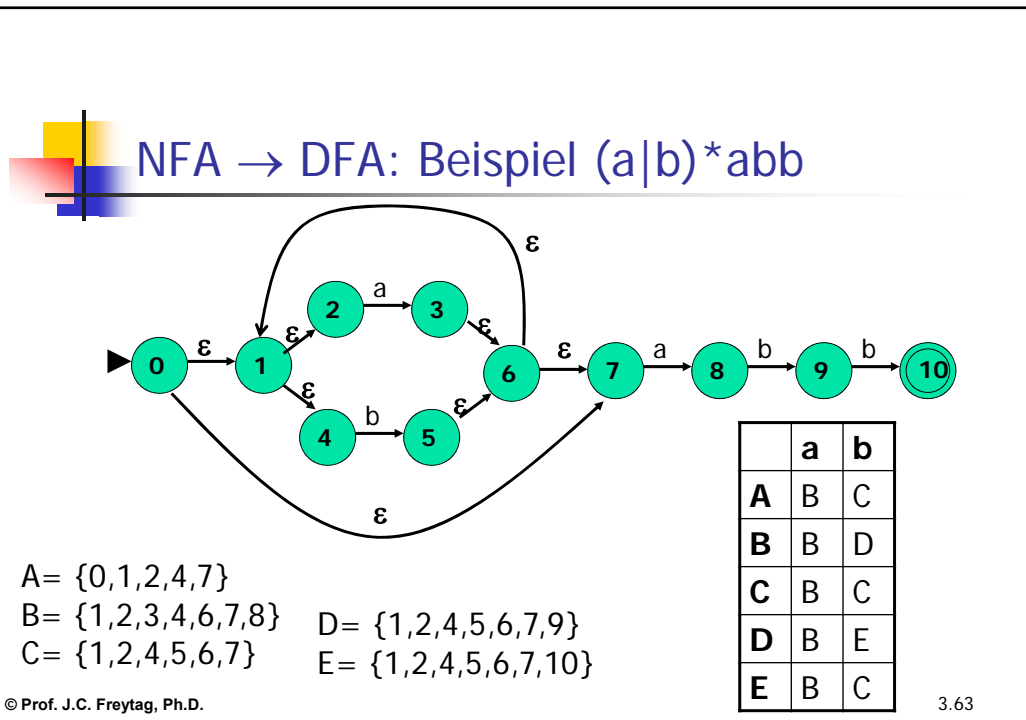
endfor

endwhile

- Ein Zustand von D ist ein Endzustand, falls er mindestens einen Endzustand aus N enthält

Vorlesung Compilerbau (SoSe2018)

Teil 3: Lexikalische Analyse





Vom Automaten zu RA



Ableitung von Gleichungen aus RAs

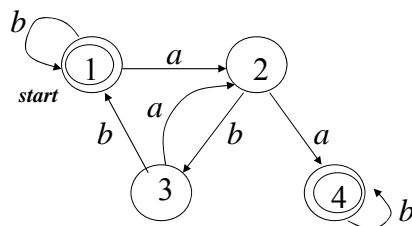
- **Ableiten von Gleichungen** aus den Transitionen eines DFA (oder auch DNA)
 - Jede Gleichung repräsentiert einen "Ein-Schritt"-Übergang, die **zu** einem bestimmten Zustand (linke Seite) führt
 - Form der Gleichung : $s = s':a ? \dots ? s'':z$
 - Mit Zuständen $s, s', \dots s''$ und "Ereignissen": a, \dots, z
 - Zustand s als Zieltransition in $(s', a, s) \dots (s'', z, s)$
 - Falls der Startzustand auch Endzustand ist, füge ϵ auf der rechten Seite der Gleichung hinzu
- **Lösen der Gleichungen**
 - Das Ergebnis ist eine Menge von RA's, die die akzeptierten Zeichensequenzen, ausgehend von jedem Zustand, beschreiben

Konstruktion der "1-Schritt" Gleichungen

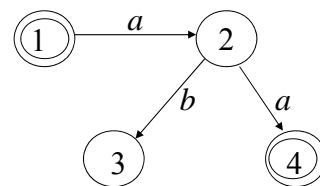
1. Konstruieren einen "Spanning Tree", dessen Wurzel der Startzustand ist

Beispiel:

Automat:



"Spanning Tree" :



Knotenreihenfolge in Präorder des "spanning trees": [1,2,3,4]

© Prof. J.C. Freytag, Ph.D.

3.67

Konstruktion der "1-Schritt" Gleichungen (2)

2. Für **jeden Endzustand** erzeuge eine Gleichung die sich aus der Summe der 1-Schritt-Transitionen ergibt, die **in** diesen Zustand hineinlaufen ("summieren")
 - $1 = 1:b ? 3:b ? \epsilon$
 - $4 = 2:a ? 4:b$
3. Für jeden Zustand, der auf der rechten Seite (RHS) in einer "1-Schritt"-Transition erscheint, summiere die "1-Schritt"-Transitionen, die in diesen Zustand hineinlaufen
 - $2 = 1:a ? 3:a$
 - $3 = 2:b$
4. Iteriere Schritt (3), bis für jeden Endzustand eine Gleichung existiert, wie dieser (über einen regulären Ausdruck) erreicht werden kann

Füge ϵ hinzu, da Zustand 1 Start- & Endzustand ist

© Prof. J.C. Freytag, Ph.D.

3.68

Konstruktion der "1-Schritt" Gleichungen (3)

- Beispiel der "1-Schritt"-Transitionen:
 - $1 = 1:b ? 3:b ? \epsilon$
 - $2 = 1:a ? 3:a$
 - $3 = 2:b$
 - $4 = 2:a ? 4:b$

Lösen der Gleichungen

4. Schreibe die Transitionsgleichungen so um, dass diese die Transitionen vom Startzustand zum Endzustand ausdrücken
- A. Ersetze Zwischenzustände, die nur von Zuständen erreicht werden können, die auf der RHS einer Gleichung in einer *früheren Gleichung (in Präorder)* genannt werden
- Zustand 3 kann nur von Zustand 2 erreicht werden. Ersetze 3 durch seine RHS, 2:b, in den Gleichungen 2 und 1
 - $1 = 1:b ? 2:bb ? \epsilon$
 - $2 = 1:a ? 2:ba$
 - $4 = 2:a ? 4:b$
 - Zustand 3 taucht nicht länger auf der rechten Seite (RHS) einer Gleichung auf
- 1.) Achtung: Alternative im "Ablauf" (?) wird in Konkatination in einen (Teil-) RA überführt (siehe B)**
- B. Falls ein Zustand nur noch durch Transitionen ausgedrückt wird, die entweder auf sich selbst verweisen oder auf Zustände, die in der Präorder **vor** dem jetzigen Zustand liegen, dann schreibe diese Gleichung, falls vorhanden, mit dem RA früherer Zustände um, gefolgt von der Eigentransition versehen mit dem Kleene-Stern (*).
- $2 = 1:a(ba)^*$
 - $4 = 2:a b^*$ Ersetzung durch RA für 2: $4 = 1:a(ba)^* a b^*$
- Wiederhole die Schritte 4A and 4B, ...

Lösen der Gleichungen (2)

4. Schreibe die Transitionsleichungen so um, dass diese die Transitionen vom Startzustand zum Endzustand ausdrücken

A. Ersetze Zwischenzustände, die nur von Zuständen erreicht werden können, die auf der RHS einer Gleichung in einer früheren Gleichung (in Präorder) genannt wird

Ersetze 2: durch $1:a(ba)^*$ in der Gleichung für Zustand 1

$$1 = 1:b ? 1:a(ba)^*bb ? \varepsilon$$

~~$$2 = 1:a(ba)^*$$~~

$$4 = 1: a(ba)^*ab^*$$

Achtung – Fehler bisher. Jetzt richtig

Füge die Terme (mit Alternative) im zu bilden zusammen, die dem gleichen Zustand "entspringen" (i.e. $1 = 1:b ? 1:a(ba)^*bb ? \varepsilon$)

B. Falls ein Zustand nur noch durch Transitionen ausgedrückt wird, die entweder auf sich selbst verweisen oder auf Zustände die in der Präorder vor dem jetzigen Zustand liegen, dann schreibe diese Gleichung, falls vorhanden mit dem RA früherer Zustände um, gefolgt von der Eigentransition versehen mit dem Kleene-Stern (*).

$$1 = \varepsilon \mid (b \mid a(ba)^*bb)^* \quad (\text{RA definiert für Endzustand 1})$$

$$4 = (b \mid a(ba)^*bb)^*a(ba)^*ab^* \quad (\text{RA definiert für Endzustand 4})$$

Erzeugen des RAs

5. Konstruiere den RA aus den Gleichungen für die Endzustände durch Alternative:

$$\blacksquare (b \mid a(ba)^*bb)^* \mid (b \mid a(ba)^*bb)^*a(ba)^*ab^*$$

■ Das Ergebnis kann mit Hilfe der algebraischen Regeln für RA vereinfacht werden

$$(b \mid a(ba)^*bb)^* \mid (b \mid a(ba)^*bb)^*a(ba)^*ab^*$$

$$= (b \mid a(ba)^*bb)^*(\varepsilon \mid a(ba)^*ab^*)$$



Grenzen regulärer Ausdrücke (RA)

- Nicht alle Sprachen sind regulär
- Es existiert kein DFA der folgende Sprachen erkennt:
 - $L = \{a^n b^n\}$
 - $L = \{wcw^k \mid w \in \Sigma^*\}$
- Folgerung (intuitiv...):
 - es gibt auch keinen regulären Ausdruck für diese Sprachen (DFA kann nicht "zählen"!)
- Aber, für folgende Sprachen existiert ein DFA:
 - Alternierende Nullen (0) und Einsen (1): $(\epsilon|1)(01)^*(\epsilon|0)$
 - Menge von Paaren aus Nullen und Einsen: $(01|10)^+$

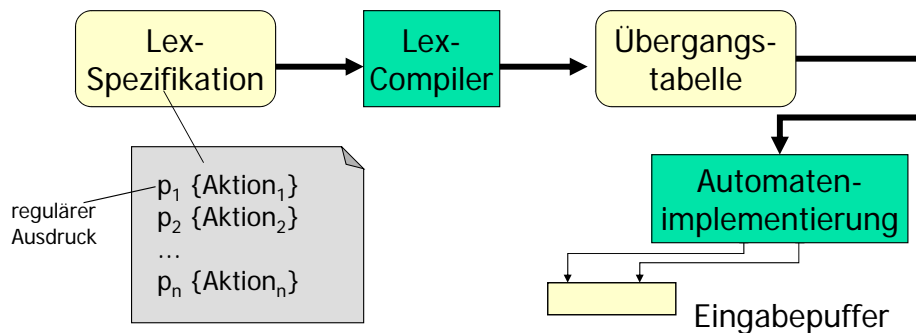


Herausforderungen?

- Spracheigenschaften können Probleme machen:
 - Reservierte Wörter: PL/I hat keine reservierten Wörter:
 - if then then then=else; else else = then;
 - Signifikante Leerzeichen
 - Fortran und Algol ignorieren Leerzeichen
 - String Konstanten
 - Sonderzeichen in Strings: newline, tab, quote, Kommentarbegrenzungszeichen
 - Namensgebung
 - Einige Sprachen begrenzen Namenslänge
 - Fortran 66: max. 6 Zeichen
 - Dieses Problem wird meist in Sprachentwürfen nicht berücksichtigt!

Entwurf eines Scanner-Generators ((F)Lex)

- Konstruktion eines Automaten ist eine automatisierbare Aufgabe

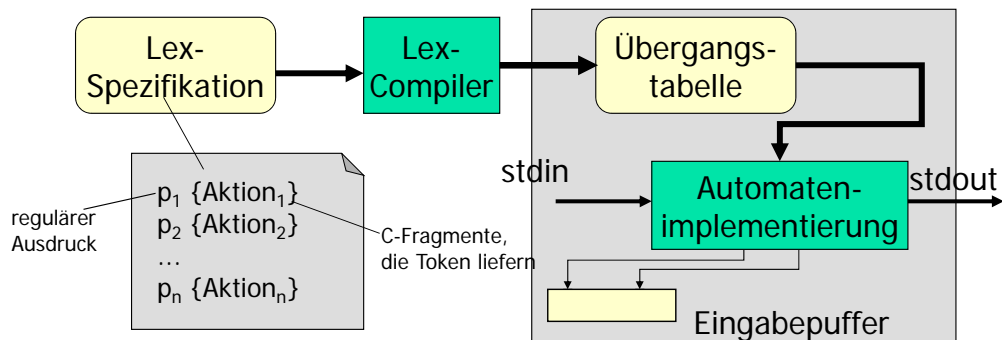


© Prof. J.C. Freytag, Ph.D.

3.75

Entwurf eines Scanner-Generators (Lex)

- Konstruktion eines Automaten ist eine automatisierbare Aufgabe



© Prof. J.C. Freytag, Ph.D.

3.76



Zusammenfassung

- Funktionsanforderungen
- NFA/DFA und RA
- Überführung ineinander
- Herausforderungen bei der Umsetzung



Fragen??

```
main(t,_,a)
char *a;
{return!0<t?t<3?main(-79,-13,a+main(-87,1-_,
main(-86,0,a+1)+a)):1,t<_?main(t+1,_,a):3,main(-94,-27+t,a
)&&t==2?_<13?main(2,_,+1,"%s %d %d\n"):9:16:t<0?t<-72?main(
t,"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l+,/n{n+\\
,/+##n+,/#;#q#n+,/+k#;*,/'r:'d*3,}{w+K w'K:'+}e#';dq#'l q#'+d'K#!\\
+k#;q#r'eKK#}w'r'eKK{nl]'/#;#q#n')}{#}w')}{nl]'/+##n';d'rw' i;# }{n\\
l]!n{n#'; r{#w'r nc{nl]'/#{l,+K {rw' iK{;[{nl]'/w#q#\\
n'wk nw' iwk{KK{nl]!/w{%l##w#' i; :{nl]'/*{q#ld;r'}{nlwb!/*de}'c \\
;:{nl'-(}{rw]'/+,,)##*')#nc,',#nw]'/+kd'+e}+;\\
#rdq#w! nr/' ) }+}{rl#'{n' ')# }'+}##(!!/"
:t<-50?_==*a?putchar(a[31]):main(-65,_,a+1):main((*a=='/')+t,_,a\\
+1):0<t?main(2,2,"%s"):a=='/'||main(0,main(-61,*a,"!ek;dc \\
i@bK'(q)-[w]*%n+r3#l,{;:\\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);}
```