

Programmierkurs Python II

Vorlesung 5: Endliche Automaten II

Michaela Regneri & Stefan Thater
FR 4.7 Allgemeine Linguistik (Computerlinguistik)
Universität des Saarlandes

Sommersemester 2011



Übersicht

- Kurze Wiederholung
- Effiziente(re) Suche
- Weitere Abschlusseigenschaften
 - Komplement
 - Schnitt
- Determinisierung (NEA \Rightarrow DEA)

2

Nicht-deterministische Automaten (NFA)

- **Nicht-deterministischer** Automat: $\langle Q, \Sigma, \Delta, q_0, F \rangle$
 - Q ist eine endliche, nicht-leere Menge von Zuständen
 - Σ ist ein endliches Alphabet
 - $Q \cap \Sigma = \emptyset$
 - Δ ist eine Überführungsrelation: $Q \times \Sigma^* \times Q$
 - q_0 ist ein Startzustand
 - F ist eine Menge von Endzuständen

3

Nicht-deterministische Automaten (NFA)

- **Konfigurationen:** $Q \times \Sigma^*$
 - aktueller Zustand + noch zu lesende Eingabe
- **Transitionen:** $(q, w) \vdash (q', w')$
 - gdw. $w = uw'$ ($u \in \Sigma^*$) und $(q, u, q') \in \Delta$
- **Reflexiv-transitive Hülle:** \vdash^*
- Ein deterministischer Automat $M = (Q, \Sigma, \delta, q_0, F)$
akzeptiert eine Eingabekette $w = a_1, \dots, a_n$
 - gdw. $(q_0, w) \vdash^* (q_f, \epsilon)$, $q_f \in F$
- **Akzeptierte Sprache** = Menge der akzeptierten Ketten.

4

Das Wortproblem

- Als **Wortproblem** einer formalen Sprache bezeichnet man das Entscheidungsproblem, zu einem gegebenen Wort festzustellen, ob dieses zur Sprache gehört oder nicht.
 - Wird ein Wort von einem Automaten akzeptiert?
 - Kann als Suchproblem aufgefasst werden.
- **Deterministische Automaten:** Die Suche wird durch Eingabe eindeutig gesteuert $\Rightarrow O(n)$
- **Nichtdeterministische Automaten:** Suche muss ggf. an einem Knoten verzweigen
 - Naive Implementierung nicht effizient
 - Determinisierung nicht immer möglich (Speicherplatz)

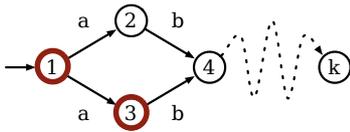
5

Suche (NFA, dritte Version)

```
class State(dict):
    ...
    def rcgnz(self, strng, final):
        if strng == '':
            return any(s in final for s in self.closure())
        for epsi in self.closure():
            for state in epsi.get(strng[0], []):
                if state.rcgnz(strng[1:], final):
                    return True
        return False
    ...
```

6

Suche (NFA, dritte Version)



```
def rcgnz(self, strng, final):
    if strng == '':
        return any(s in final for s in self.closure())
    for epsi in self.closure():
        for state in epsi.get(strng[0], []):
            if state.rcgnz(strng[1:], final):
                return True
    return False
```

```
S1.rcgnz("abcde", ...)
S2.rcgnz("bcde", ...)
S4.rcgnz("cde", ...)
... ⇒ False
S3.rcgnz("bcde", ...)
S4.rcgnz("cde", ...)
... ⇒ False
```

7

Suche (NFA, dritte Version)

- Für ein Eingabewort w und einen NFA mit k Zuständen gibt es im ungünstigsten Fall ...
 - exponentiell viele Pfade durch den Automaten,
 - aber nur $k|w|$ verschiedene Konfigurationen
- ⇒ Die gleiche Konfiguration wird mehrfach besucht.
- ⇒ Effizientere Suche durch Speichern von bereits besuchten Konfigurationen.

8

Suche (Variante, vorläufig)

```
def rcgnz(self, strng, final):
    stack = [(self, strng)]
    while stack:
        (state, strng) = stack.pop()
        if strng == '':
            if any(s in final for s in state.closure()):
                return True
        else:
            for epsi in state.closure():
                for other in epsi.get(strng[0], []):
                    stack.append((other, strng[1:]))
    return False
```

rcgnz als Tiefensuche
(nicht-rekursiv)

9

Suche (Variante)

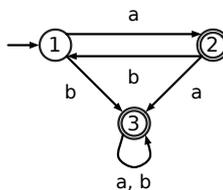
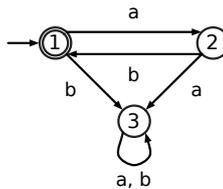
```
def rcgnz(self, strng, final):
    stack = [(self, strng)]
    visited = set()
    while stack:
        (state, strng) = stack.pop()
        if (state, strng) in visited:
            continue
        visited.add((state, strng))
        if strng == '':
            ...
        else:
            ...
    return False
```

Wenn eine Konfiguration ein zweites Mal besucht wird, können wir abbrechen (warum?)

10

Komplementautomat

- **Gegeben:** ein DFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$
- **Gesucht** ist ein Automat, der das Komplement der Sprache $L(M)$ beschreibt.
- **Lösung:** Vertausche die Endzustände
 - $M' = \langle Q, \Sigma, \delta, q_0, Q \setminus F \rangle$
- **Voraussetzung:** der Automat muss vollständig sein
 - $\delta(q, a)$ ist für alle $a \in \Sigma, q \in Q$ definiert



11

Schnitt

- Gegeben sind zwei Automaten M_1 und M_2
- Gesucht wird ein Automat M_3 , der $L(M_1) \cap L(M_2)$ akzeptiert.
- Idee: die Automaten werden parallel laufengelassen
 - wenn beide Automaten nach dem Lesen der Eingabe in einem Endzustand stehen \Rightarrow das Wort liegt im Schnitt von L_1 und L_2 .

12

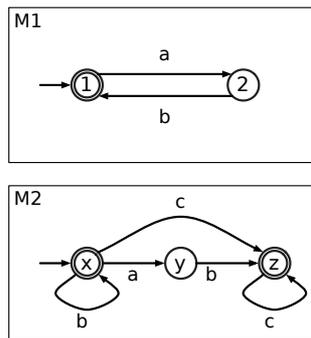
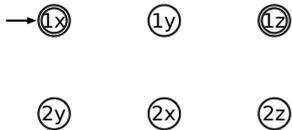
Schnitt (Produktautomat)

- Gegeben zwei Sprachen L_1 und L_2 , die jeweils von den Automaten M_1 und M_2 akzeptiert werden.
 - $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$
 - $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$
- Produktautomat $M = (Q, \Sigma, \delta, s, F)$
 - $Q = Q_1 \times Q_2$
 - $F = F_1 \times F_2$
 - $s = (s_1, s_2)$
 - $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$

13

Schnitt (Produktautomat)

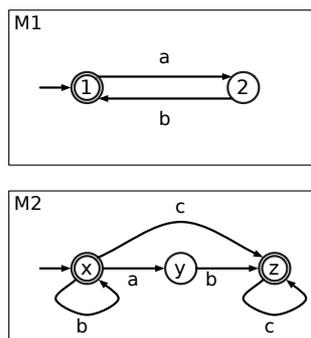
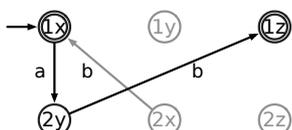
- Alle Zustände paaren
- Endzustände: alle Paare aus zwei Endzuständen
- Startzustand: das Paar aus den zwei Startzuständen



14

Schnitt (Produktautomat)

- Übergänge:
 - $\delta((p_1, q_1), a) = (p_2, q_2)$ gdw.
 - $\delta_1(p_1, a) = p_2$ und
 - $\delta_2(q_1, a) = q_2$
- Nicht erreichbare Zustände können gelöscht werden.



15

Schnitt über Abschlusseigenschaften

- $L(M_1) \cap L(M_2) = \Sigma^* - ((\Sigma^* - L(M_1)) \cup (\Sigma^* - L(M_2)))$
- Vereinige die Komplementautomaten und bilde davon den Komplementautomaten
- Notwendiger Zwischenschritt: Determinisierung

16

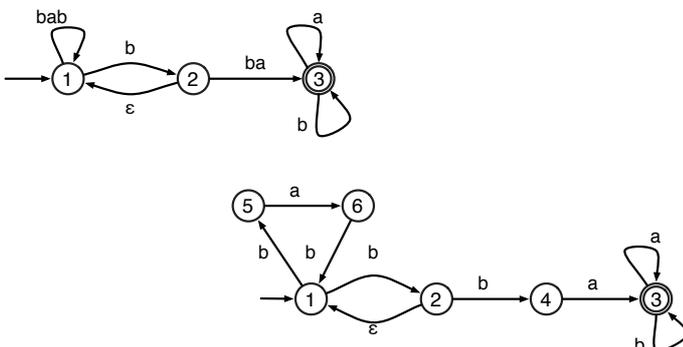
Äquivalenz von DEA und NEA

- Für jeden nichtdeterministischen Automaten M gibt es einen deterministischen Automaten M' mit $L(M) = L(M')$
- Der Beweis ist gleichzeitig der Korrektheitsbeweis für den Konstruktionsalgorithmus.
- Der resultierende Automat
 - ist deterministisch
 - und beschreibt die gleiche Sprache wie der NEA
- Wir beschränken uns hier auf die Beschreibung des Algorithmus

17

Determinisierung

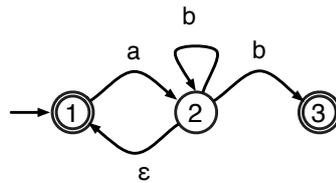
- **Vorverarbeitungsschritt:** Mehrsymbolkanten entfernen



18

Determinisierung

- ϵ -Abschluss:
 - $E(s) = \{ s' \mid (s, \epsilon) \vdash^* (s', \epsilon) \}$
- Beispiel:
 - $E(q_1) = \{q_1\}$
 - $E(q_2) = \{q_1, q_2\}$
 - $E(q_3) = \{q_3\}$



19

Determinisierung

- Die Zustände des determinierten Automaten sind Mengen von Zuständen des ursprünglichen Automaten
- **Idee:** Die Zustandsmengen geben an, in welchem Zustand sich der ursprüngliche Automat nach Lesen der Eingabe befinden kann.

20

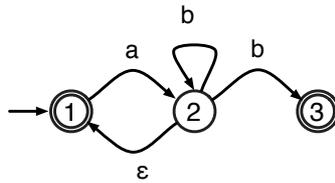
Determinisierung

- NEA $M = \langle Q, \Sigma, \Delta, q_0, F \rangle$
- DEA $M' = \langle Q', \Sigma, \delta', q_0', F' \rangle$ mit
 - $Q' = 2^Q$
 - $q_0' = E(q_0)$
 - $F' = \{ K \subseteq Q' \mid K \cap F \neq \emptyset \}$
 - Für alle $K \subseteq Q'$ und $a \in \Sigma$:
 - $\delta'(K, a) = \bigcup \{ E(p) \mid p \in Q \text{ und für ein } q \in K: \Delta(q, a, p) \}$

21

Determinisierung

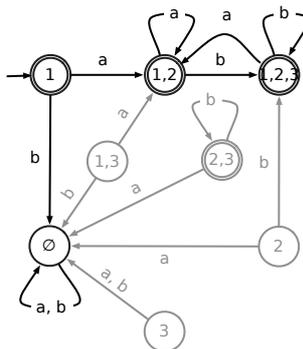
	1	2	3	1,2	1,3	2,3	1,2,3	\emptyset
1				a				b
2							b	a
3								a,b
1,2				a			b	
1,3				a				b
2,3						b		a
1,2,3				a			b	
\emptyset								a,b



22

Determinisierung

	1	2	3	1,2	1,3	2,3	1,2,3	\emptyset
1				a				b
2							b	a
3								a,b
1,2				a			b	
1,3				a				b
2,3						b		a
1,2,3				a			b	
\emptyset								a,b



23

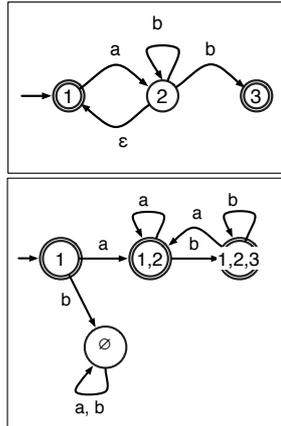
Determinisierung (Variante)

- Algorithmus zur Teilmengen-Konstruktion (*Subset construction*)
- Generiere nur die Zustände, die überhaupt besucht werden können
- Prinzip:
 - Ersetze Zustände durch ihren ϵ -Abschluss
 - Beginne beim Startzustand
 - Generiere „on the fly“ die Zielzustände
 - Verfahre mit den generierten Zielzuständen entsprechend

24

Determinisierung (Variante)

- Startzustand: $\{1\}$
- $\delta'(\{1\}, a) = \{1,2\}$ (= Endz.)
- $\delta'(\{1,2\}, a) = \{1,2\}$
 $\delta'(\{1,2\}, b) = \{1,2,3\}$ (= Endz.)
- $\delta'(\{1,2,3\}, a) = \{1,2\}$
 $\delta'(\{1,2,3\}, b) = \{1,2,3\}$
- $\delta'(\{2,3\}, b) = \{1,2,3\}$
- ggf. Senke hinzufügen



25

Determinisierung (Pseudocode)

determinize($Q, \Sigma, \delta, q_0, F$):

$q_{0'} \leftarrow \{q_0\}$

$Q' \leftarrow q_{0'}$

enqueue(Queue, $q_{0'}$)

while Queue $\neq \emptyset$:

$S \leftarrow$ dequeue(Queue)

for a **in** Σ :

$\delta'(S, a) = \bigcup_{r \in S} \delta(r, a)$

if $\delta'(S, a) \notin Q'$:

$Q' \leftarrow Q' \cup \delta'(S, a)$

 enqueue(Queue, $\delta'(S, a)$)

if $\delta'(S, a) \cap F \neq \emptyset$:

$F' \leftarrow F' \cup \{\delta'(S, a)\}$

return ($Q', \Sigma, \delta', q_{0'}, F'$)

Annahme:

keine ε -Übergänge

26