Programmierkurs Python II

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

Sommersemester 2010

Übersicht

- Effiziente(re) Suche
- Weitere Abschlusseigenschaften
  - Komplement
  - Schnitt
- Determinisierung (NEA ⇒ DEA)

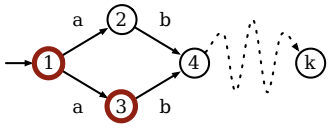
Das Wortproblem

- 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 ⇒ O(n)
- Nichtdeterministische Automaten: Suche muss ggf. an einem Knoten verzweigen
  - Naive Implementierung nicht effizient
  - Determinisierung nicht immer möglich (Speicherplatz)

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
   ...
```


Suche (NFA, dritte Version)




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

```
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
```


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.

Suche (Variante, vorläufig)

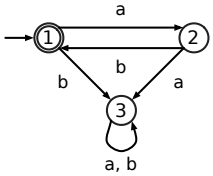
```
def rcgnz(self, strng, final):
   stack = [(self, strng)]
   while stack:
      (state, strng) = stack.pop()
      if strng == '':
          if any(s in final for s in state.closure()):
             return True
      else:
         for epsi in state.closure():
            for other in epsi.get(strng[0],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

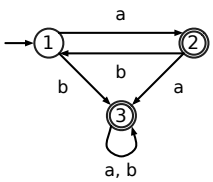

Suche (Variante)

```
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
```


Komplementautomat

- Gegeben: ein DFA M = ⟨Q, Σ, δ, q_0, F⟩
- Gesucht ist ein Automat, der das Komplement der Sprache L(M) beschreibt.
- Lösung: Vertausche die Endzustände
  - M' = ⟨Q, Σ, δ, q_0, Q\F⟩
- Voraussetzung: der Automat muss vollständig sein
  - δ(q,a) ist für alle a ∈ Σ, q ∈ Q definiert





Schnitt

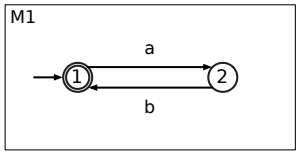
- Gegeben sind zwei Automaten M_1 und M_2
- Gesucht wird ein Automat M_3, der $L(M_1) \cap L(M_2)$ akzeptiert.
- Idee: die Automaten werden parallel laufengelassen
  - wenn beide Automaten nach dem Lesen der Eingabe in einem Endzustand stehen ⇒ das Wort liegt im Schnitt von L_1 und L_2.

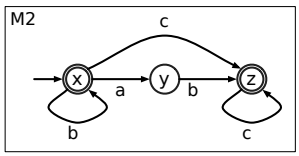
Schnitt (Produktautomat)

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

Schnitt (Produktautomat)

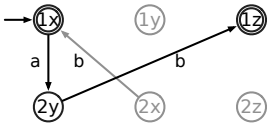
- Alle Zustände paaren
- Endzustände: alle Paare aus zwei Endzuständen
- Startzustand: das Paar aus den zwei Startzuständen

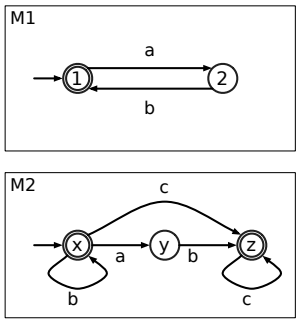




Schnitt (Produktautomat)

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





Schnitt über Abschlusseigenschaften

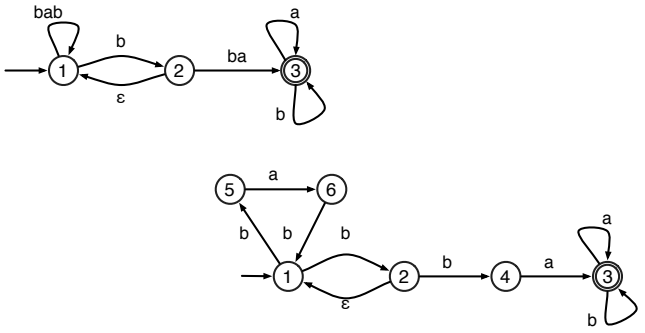
- $L(M_1) \cap L(M_2) = \Sigma^* - ((\Sigma^* - L(M_1)) \cup (\Sigma^* - L(M_2)))$
- Vereinige die Komplementautomaten und bilde davon den Komplementautomaten
- Notwendiger Zwischenschritt: Determinisierung

Äquivalenz von DEA und NEA

- Für jeden nichtdeterministischen Automanten 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

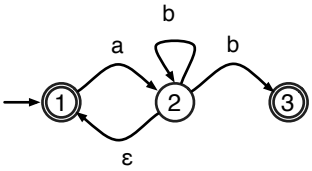
Determinisierung

- Vorverarbeitungsschritt: Mehrsymbolkanten entfernen



Determinisierung

- ε-Abschluss:
  - E(s) = { s' | ⟨s, ε⟩ ⊢* ⟨s', ε⟩ }
- Beispiel:
  - E(q_1) = {q_1}
  - E(q_2) = {q_1, q_2}
  - E(q_3) = {q_3}



Determinisierung

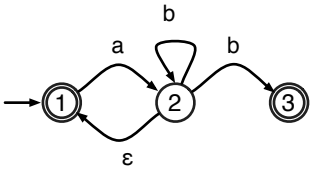
- Die Zustände des determinierten Automaten sind Mengen von Zuständen des ursprünglichen Automaten
- Idee: Die Zustandsmengen geben an, in welchem Zustand sich der ursprüngliche Automat nach Lesen der Eingabe befinden kann.

Determinisierung

- NEA M = ⟨Q, Σ, Δ, q0, F⟩
- DEA M' = ⟨Q', Σ, δ', q0', F'⟩ mit
  - Q' = 2^Q
  - $q_0' = E(q_0)$
  - F' = { K ⊆ Q' | K ∩ F ≠ ∅ }
  - Für alle K ⊆ Q' und a ∈ Σ:
    - δ'(K, a) = ⋃{ E(p) | p ∈ Q und für ein q ∈ K: Δ(q, a, p)}

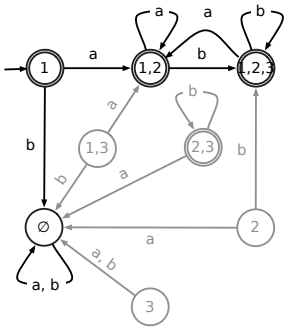
Determinisierung

	1	2	3	1,2	1,3	2,3	1,2,3	∅
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	
∅								a,b



Determinisierung

	1	2	3	1,2	1,3	2,3	1,2,3	∅
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	
∅								a,b

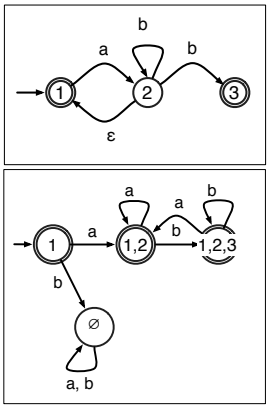


Determinisierung (Variante)

- 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

Determinisierung (Variante)

- Startzustand: {1}
- δ’({1}, a) = {1,2} (= Endz.)
- δ’({1,2}, a) = {1,2}
  δ’({1,2}, b) = {1,2,3} (= Endz.)
- δ’({1,2,3}, a) = {1,2}
  δ’({1,2,3}, b) = {1,2,3}
- δ’({2,3}, b) = {1,2,3}
- ggf. Senke hinzufügen



Determinisierung (Variante)

```
determinize(Q,Σ,δ,q₀,F):
   q₀’ ← {q₀}
   Q’ ← q₀’
   enqueue(Queue, q₀’)
   while Queue ≠ ∅:
      S ← dequeue(Queue)
      for a in Σ:
         δ’(S,a) = ⋃r∈S δ(r,a)
         if δ’(S,a) ∉ Q’:
            Q’ ← Q’ ∪ δ’(S,a)
            enqueue(Queue, δ’(S,a))
            if δ’(S,a) ∩ F ≠ ∅:
               F’ ← F’ ∪ {δ’(S,a)}
   return (Q’,Σ,δ’,q₀’,F’)
```

Annahme:
keine ε-Übergänge