Programmierkurs Python II

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

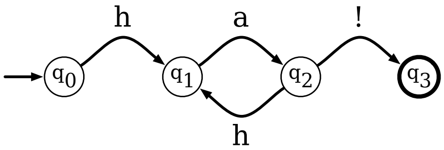
Sommersemester 2010

Endliche Automaten

- Endliche Automaten sind einfache Modelle zur Beschreibung regulärer Sprachen.
  - ⇒ für jede reguläre Sprache L gibt es einen entsprechenden Automaten M, der L erkennt (akzeptiert).
- Endliche Automaten sind äquivalent zu regulären Ausdrücken:
  - ⇒ für jeden regulären Ausdruck gibt es einen äquivalenten endlichen Automaten, und umgelehrt.

Zustandsdiagramme

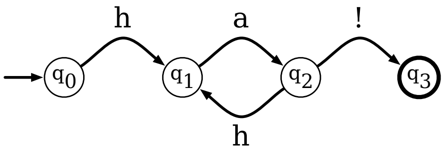
- Endliche Automaten können informell durch Zustandsdiagramme beschrieben werden.
  - akzeptierte Wörter ~ Pfaden zu Endzuständen



ha!
haha!
hahaha!
hahahaha!
...

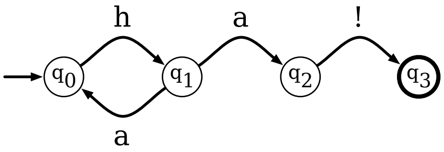
Endliche Automaten

- Deterministische endliche Automaten:
  - für jeden Zustand gibt es für jedes Symbol (Zeichen) genau einen Nachfolgezustand.
  - fehlende Kanten in Zustandsdiagrammen ~ Kanten in einen impliziten Nicht-Endzustand.



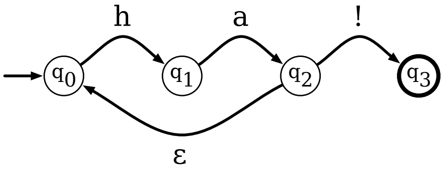
Endliche Automaten

- Nichtdeterministische Automaten können Zustände haben, die mehrere ausgehenden Kanten für einen Buchstaben haben.



Endliche Automaten

- In nichtdeterministischen Automaten sind die Kanten mit Wörtern (statt einzelner Buchstaben) etikettiert.
- Insbesondere sind ε-Übergänge erlaubt, die keine Eingabe konsumieren (ε = leeres Wort).



Alphabet & Wort

- Ein Alphabet Σ ist eine endliche, nicht-leere Menge von Symbolen.
- Ein Wort $w \in \Sigma^*$ über dem Alphabet Σ ist eine endliche, möglicherweise leere Kette von Symbolen aus Σ.
- Die Länge $|w|$ eines Wortes w ist die Anzahl der verketteten Symbole von w.
- Das leere Wort ε ist das Wort mit Wortlänge 0 ($|\varepsilon|=0$).

Sprachen

- Ein Sprache über einem Alphabet Σ ist eine Menge von Worten über Σ.
  - typischerweise sind Sprachen unendlich
- Einige besondere Sprachen:
  - Die leere Wortmenge $\emptyset$ heißt die „leere Sprache“
  - Die Menge Σ^* umfasst alle Worte über Σ (incl. ε)
  - Die Menge Σ^+ umfasst alle Worte über Σ ohne ε

Deterministische Automaten

- Deterministischer endlicher 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 Überführungsfunktion: $Q \times \Sigma \rightarrow Q$
  - q_0 ist ein Startzustand
  - F ist eine Menge von Endzuständen

Deterministische Automaten

- Konfigurationen: Q x Σ^*
  - aktueller Zustand + noch zu lesende Eingabe
- Transitionen: $\langle q, w\rangle \vdash \langle q', w'\rangle$
  - gdw. w = aw' und q' = $\delta(q, a)$
- Reflexiv-transitive Hülle: $\vdash^*$
- Ein deterministischer Automat M = $\langle Q, \Sigma, \delta, q_0, F\rangle$ akzeptiert eine Eingabekette w = $a_1, \ldots, a_n$
  - gwd. $\langle q_0, w\rangle \vdash^* \langle q_f, \varepsilon\rangle$, $q_f \in F$
- Akzeptierte Sprache = Menge der akzeptierten Ketten.

Deterministische Automaten

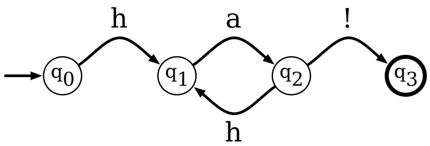
```
class DFA:
   def __init__(self, initial, transitions, final):
      self.initial = initial
      self.final = set(final)
      self.trns = dict()
      for (src, char, tgt) in transitions:
         try:
            self.trns[src][char] = tgt
         except KeyError:
            self.trns[src] = {char:tgt}
   def rcgnz(self, strng):
      ...
```


Deterministische Automaten

```
class DFA:
   ...
   def rcgnz(self, string):
      state = self.initial
      try:
         for char in string:
            state = self.trns[state][char]
      except KeyError: # impliziter Nicht-Endzustand
         return False
      return state in self.final
```

Lachmaschine

```
m = DFA(0, [(0,'h',1), (1,'a',2), (2,'h',1), (2,'!',3)], [3])
m.rcgnz('haha!')
⇒ True
m.rcgnz('hah!')
⇒ False
```



(Lewis & Papadimitriou, 1981)

Nicht-deterministische Automaten

- Nicht-deterministischer Automat: ⟨Q, Σ, Δ, q_0, F⟩
  - Q ist eine endliche, nicht-leere Menge von Zuständen
  - Σ ist ein endliches Alphabet
    - Q ∩ Σ = ∅
  - Δ ist eine Überführungsrelation: Q x Σ* x Q
  - q_0 ist ein Startzustand
  - F ist eine Menge von Endzuständen
- Transitionen: ⟨q, w⟩ ⊢ ⟨q’, w’⟩
  - gdw. w = uw’ (u ∈ Σ*) und ⟨q, u, q’⟩ ∈ Δ

Nichtdeterministische Automaten (erste Version)

```
class NFA:
   def __init__(self, initial, transitions, final):
      self.initial = initial
      self.final = set(final)
      self.trns = dict()
      for (src, strng, tgt) in transitions:
         try:
            self.trns[src].add((strng, tgt))
         except KeyError:
            self.trns[src] = set([(strng, tgt)])
   ...
```

Nichtdeterministische Automaten (erste Version)

```
class NFA:
   ...
   def rcgnz(self, strng):
      return self._rcgnz(self.initial, strng)
   def _rcgnz(self, state, strng):
      if strng == '' and state in self.final:
         return True
      for (prefix, _state) in self.trns[state]:
         prfxlen = len(prefix)
         if prefix == strng[:prfxlen]:
            if self._rcgnz(_state, strng[prfxlen:]):
               return True
      return False
```


Probleme & sinnvolle Einschränkungen

- Das Programm terminiert nicht notwendigerweise.
  - ε-Übergänge
- Ineffiziente Suche nach „passenden" Folgezuständen
  - Iteration über alle Folgezustände
- Sinnvolle Einschränkung:
  - Nur Übergänge (q, w, q') ∈ Δ mit |w| = 1 zulassen
  - ⇒ ε-Abschluss („ε-closure")

Nichtdeterministische Automaten (zweite Version)

- Repräsentation ähnlich wie bei deterministischen Automaten:
  - `dict` (state) → `dict` (char) → `set` (state)
- ε-closure: alle über ε-Übergänge erreichbaren Zustände

Nichtdeterministische Automaten (zweite Version)

```
class NFA:
   def __init__(self, initial, trns, final):
      self.initial = initial
      self.final = set(final)
      self.trns = dict()
      for (src, char, tgt) in transitions:
         if src in self.trns:
            if char in self.trns[src]:
               self.trns[src][char].add(tgt)
            else:
               self.trns[src][char] = set([tgt])
         else:
            self.trns[src] = {char:set([tgt])}
```


Nichtdeterministische Automaten (zweite Version)

```
class NFA:
   ...
   def closure(self, state):
      def add(state, states):
         if state in states:
            return states
         states.add(state)
         for other in self.trns.get(state, {}).get('', []):
            add(other, states)
         return states
      return add(state, set())
   ...
```


Nichtdeterministische Automaten (zweite Version)

```
class NFA:
   ...
   def rcgnz(self, strng):
      return self._rcgnz(self.initial, strng)
   def _rcgnz(self, state, strng):
      if strng == '':
         return self.closure(state) & self.final
      for e in self.closure(state):
         for s in self.trns.get(e, {}).get(strng[0], []):
            if self._rcgnz(s, strng[1:]):
               return True
      return False
```

Abschlusseigenschaften

- Die Klasse der von endlichen Automaten akzeptierten Sprachen ist abgeschlossen unter:
  - Vereinigung
  - Konkatenation
  - Kleene Stern
  - Komplement
  - Schnitt

Abschlusseigenschaften: Konkatenation

- Seien L_1, L_2 zwei reguläre Sprachen
  - $M_1 = \langle Q_1, \Sigma, \Delta_1, s_1, F_1\rangle$, $L(M_1) = L_1$
  - $M_2 = \langle Q_2, \Sigma, \Delta_2, s_2, F_2\rangle$, $L(M_2) = L_2$
- Dann akzeptiert $M = \langle Q, \Sigma, \Delta, s_1, F_2\rangle$ die Sprache $L_1 \circ L_2$:
  - $Q = Q_1 \cup Q_2$
  - $\Delta = \Delta_1 \cup \Delta_2 \cup (F_1 \times \{\varepsilon\} \times \{s_2\})$

Abschlusseigenschaften: Vereinigung

- Seien L_1, L_2 zwei reguläre Sprachen
  - $M_1 = \langle Q_1, \Sigma, \Delta_1, s_1, F_1\rangle$, $L(M_1) = L_1$
  - $M_2 = \langle Q_2, \Sigma, \Delta_2, s_2, F_2\rangle$, $L(M_2) = L_1$
- Dann akzeptiert $M = (Q, \Sigma, \Delta, s, F)$ die Sprache $L_1 \cup L_2$:
  - $Q = Q_1 \cup Q_2 \cup \{s\}$
    - $s \notin Q_1 \cup Q_2$
  - $\Delta = \Delta_1 \cup \Delta_2 \cup \{\langle s, \varepsilon, s_1\rangle, \langle s, \varepsilon, s_2\rangle\}$
  - $F = F_1 \cup F_2$

Abschlusseigenschaften: Kleene Stern

- Seien L_1 eine reguläre Sprachen
  - $M_1 = \langle Q_1, \Sigma, \Delta_1, s_1, F_1\rangle$, $L(M_1) = L_1$
- Dann akzeptiert $M = \langle Q, \Sigma, \Delta, s, F\rangle$ die Sprache L_1^*:
  - $Q = Q_1 \cup \{s\}$
    - $s \notin Q_1$
  - $F = F_1 \cup \{s\}$
  - $\Delta = \Delta_1 \cup (F_1 \times \{\varepsilon\} \times \{s_1\})$

Nichtdeterministische Automaten (dritte Version)

```
class State(dict):
   def __hash__(self):
      return id(self)
   def add(self, char, state):
      try:
         self[char].add(state)
      except KeyError:
         self[char] = set([state])
   ...
```


Nichtdeterministische Automaten (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
   def closure(self):
      return <per ε-Transition erreichbare Zustände>
```

Nichtdeterministische Automaten (dritte Version)

```
class NFA:
   def __init__(self, initial, final):
      self.initial = initial
      self.final = final
   def rcgnz(self, string):
      return self.initial.rcgnz(string, self.final)
   @staticmethod
   def singleton(char):
      final = set([State()])
      return NFA(State([(char, final)]), final)
   def concat(self, other):
      for final in self.final:
         final.add('', other.initial)
      return NFA(self.initial, other.final)
```


Nichtdeterministische Automaten (dritte Version)

```
class NFA:
   ...
   def union(self, other):
      initial = State()
      initial.add('', self.initial)
      initial.add('', other.initial)
      return NFA(initial, self.final | other.final)
   def star(self):
      initial = State()
      initial.add('', self.initial)
      for final in self.final:
         final.add('', initial)
      return NFA(initial, set([initial]))
```


Lachmaschine

```
m = NFA.singleton('h').\
        concat(NFA.singleton('a')).\
        star().\
        concat(NFA.singleton('!'))
m.rcgnz('haha!')
⇒ True
m.rcgnz('hah!')
⇒ False
```

Reguläre Ausdrücke über Σ

- Jedes Zeichen a ∈ Σ ∪ {∅} ist ein regulärer Ausdruck
- Wenn a, b reguläre Ausdücke sind, dann auch
  - (ab) (Konkatenation)
  - (a ∪ b) (Disjunktion)
  - a^* (Kleene-Stern)

Reguläre Ausdrücke über Σ

- Jeder reguläre Ausdruck denotiert eine reguläre Sprache:
  - $L(\emptyset) = \emptyset$
  - $L(a) = \{a\}$, für jedes $a \in \Sigma$
  - $L(\alpha\beta) = L(\alpha) \circ L(\beta)$
  - $L(\alpha \cup \beta) = L(\alpha) \cup L(\beta)$
  - $L(\alpha^*) = L(\alpha)^*$

Reguläre Ausdrücke über Σ

- Jeder reguläre Ausdruck kann in einen endlichen Automaten übersetzt werden:
  - a: $(\{q_1, q_2\}, \Sigma, \{(q_1, a, q_2)\}, q_1, \{q_2\})$
  - ∅: $(\{q\}, \Sigma, \emptyset, q, \emptyset)$
  - Konkatenation, Kleene Stern, Vereinigung: siehe Abschlusseigenschaften