Einführung in die Computerlinguistik

Morphologie und Automaten II

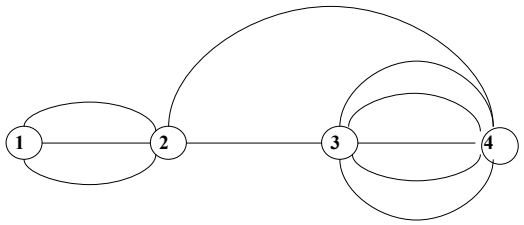
WS 2008/2009

Manfred Pinkal

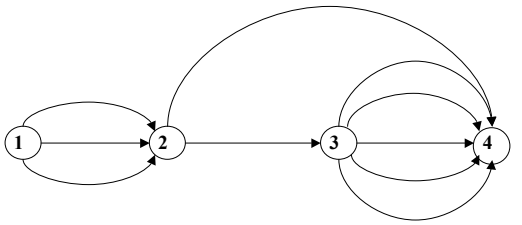
Definition: Zustandsdiagramm [1]

Ein Zustandsdiagramm ist ein gerichteter Graph mit Kanteninschriften.

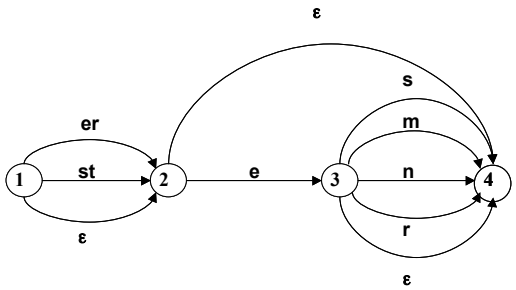
Ein Graph



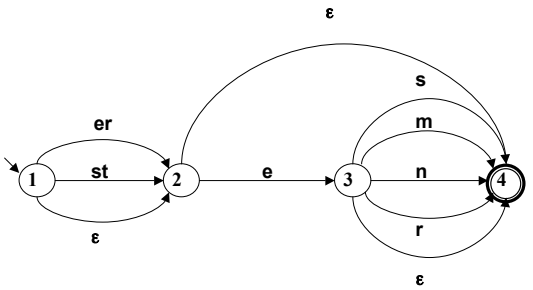
Ein gerichteter Graph



Ein gerichteter Graph mit Kanteninschriften



Zustandsdiagramm



Zustandsdiagramm

Ein Zustandsdiagramm besteht aus

- Knoten (Zuständen) (im Beispiel: 1,2,3,4)
- davon ein Startknoten (1)
- einem oder mehreren Zielknoten (4)
- Kanten zwischen den Knoten, die
  - gerichtet und
  - beschriftet sind
- Die Kanteninschriften bestehen aus Ketten von Symbolen über einem Alphabet (im Beispiel: e,r,m,n,s,t).
- Auch das leere Wort (**ε**) ist als Kantenbeschriftung zugelassen; **ε** ist kein Symbol des Alphabets, sondern bezeichnet den Grenzfall der „aus 0 Symbolen bestehenden" leeren Kette.

Definition: Zustandsdiagramm [2]

Formal wird ein Zustandsdiagramm definiert als ein Quintupel (Folge von 5 Elementen)

A = <K, Σ, Δ, s, F>, wobei

- K nicht-leere endliche Menge von Knoten (Zuständen)
- Σ nicht-leeres Alphabet
- $s \in K$ Startzustand
- $F \subseteq K$ Menge von Endzuständen
- $\Delta : K \times \Sigma^* \times K$ Menge von beschrifteten Kanten (Übergangsrelation)

Anmerkung: Das Zustandsdiagramm heißt auch „nicht-deterministischer endlicher Automat" (NEA), engl.: „non-deterministic finite-state automaton" (NFA) (Erklärung später)

Beispiel: Das Adjektivendungs-Diagramm

NEA A = <K, Σ, Δ, s, F> mit

- K= {1,2,3,4} (Zustände)
- Σ = {e, m, n, r, s, t} (Alphabet)
- s = 1 (Startzustand)
- F = {4} (einziger Endzustand)
- Δ = {<1,*er*,2>, <1,*st*,2>, <1,ε, 2>, <2,*e*,3>, <2,ε,4>, ...} (Übergangsrelation)

Die Interpretation des Zustandsdiagramms

- Das Zustandsdiagramm beschreibt alle Symbolkombinationen oder Worte, die sich dadurch ergeben, dass man das Diagramm vom Startknoten zu einem Zielknoten durchläuft und die Inschriften der Kanten, die man dabei beschreitet, aufliest und aneinanderhängt. Man nennt die Menge der Worte, die sich so erzeugen lassen, die vom Diagramm beschriebene Sprache.
- Üblicherweise werden Diagramme verwendet, um Eingabeketten zu testen. Man spricht von endlichen Automaten, und sagt, dass ein Automat ein Wort akzeptiert und eine Sprache akzeptiert oder definiert.

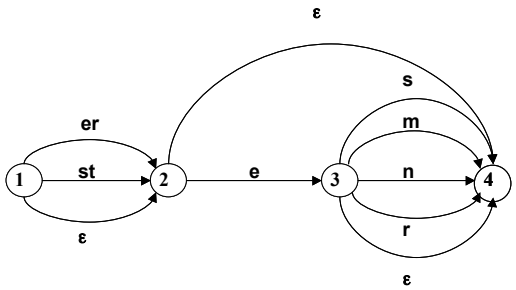
Durch NEA akzeptiertes Wort/definierte Sprache

- Ein Wort $w \in \Sigma^*$ wird durch den NEA
  A =<K,Σ,Δ,s,F> akzeptiert
  gdw. es eine Folge von Kanten (einenPfad durch den NEA)
  $<s,u_1, k_1>, <k_1, u_2, k_2>, \ldots, <k_{n-1}, u_n, k_n>$ gibt, sodass $k_n \in F$ und $u_1 u_2 \ldots u_n = w$ (die Konkatenation, das Aneinanderhängen der Inschriften der durchlaufenen Kanten ergibt das Wort w).
- Die vom NEA A =<K,Σ,Δ,s,F> definierte (akzeptierte) Sprache L(A) ist die Menge der von A akzeptierten Worte.

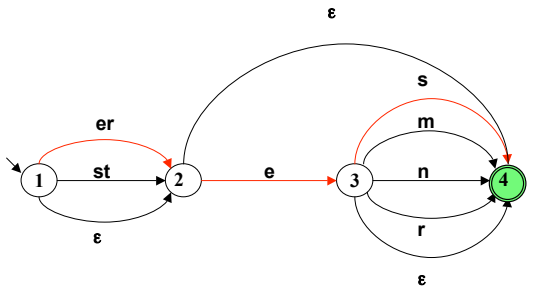
Eine methodische Bemerkung

- Die Definition des Zustandsdiagramms/NEA spezifiziert eine formale Notation, die für sich genommen keine Bedeutung hat.
- Durch die Definitionen der letzten Folie (akzeptiertes Wort/definierte Sprache) wird diese Datenstruktur interpretiert: Wir verwenden Zustandsdiagramme, um (die Zugehörigkeit von Symbolketten zu) Sprachen zu spezifizieren.
- Hinzu kommen muss ein handhabbares Verfahren, ein Algorithmus, um den Zugehörigkeitstest tatsächlich durchzuführen.

Ein Suchproblem

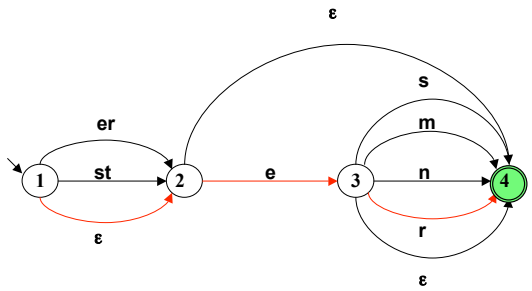


Ein Weg durchs Diagramm



klein eres

Ein alternativer Weg durchs Diagramm



klein er|es

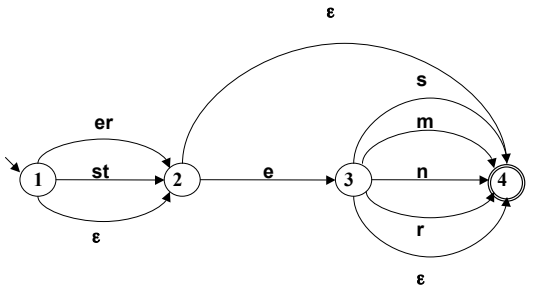
Ein Suchproblem

- Das Diagramm erlaubt typischerweise an einem Knoten/ einem Zustand mehrere Übergänge bei derselben Eingabe (deshalb „nicht-deterministisch").
- Die zufällige Wahl einer Kante kann sich erst viel später als falsch herausstellen. Sie gibt keine Gewähr, dass tatsächlich alle möglichen Wege durch das Diagramm getestet wurden.
- Wir benötigen ein Verfahren, das uns die vollständige Suche garantiert.

Ein Verfahren für die Pfadsuche: Die Idee

- Angenommen, wir befinden uns an einem Knoten k des Diagramms (dem „aktuellen Zustand“) und an einer Position im Eingabewort w (der „aktuellen Position“) – beides zusammen nennen wir die aktuelle Konfiguration.
- Wir testen, welche Kanten wir beschreiten können.
- Wir rücken nicht direkt im Automaten vor, sondern legen die alternativ möglichen Resultatkonfigurationen – Zustand und Position im Wort – in einer Liste noch zu erledigender Teilaufgaben, der Agenda, ab.
- Dann nehmen wir einen Eintrag von der Agenda. – Welchen? Es gibt unterschiedliche Möglichkeiten, die unterschiedlichen Suchverfahren entsprechen.
- Wir betrachten die Agenda zunächst als Stapel („Stack“), legen neue Aufgaben oben auf die Agenda und nehmen einzelne Aufgaben ebenfalls von oben von der Agenda ("Last In – First Out").
- Wir führen die Aufgabe aus (d.h., rücken im Eingabewort auf die bezeichnete Stelle vor und gehen in den bezeichneten Zustand), testen mögliche nächste Schritte, legen die Resultate auf die Agenda usw. – bis wir die Eingabe erfolgreich abgearbeitet haben (akzeptieren!) oder die Agenda keine Aufgaben mehr enthält ist (zurückweisen!).

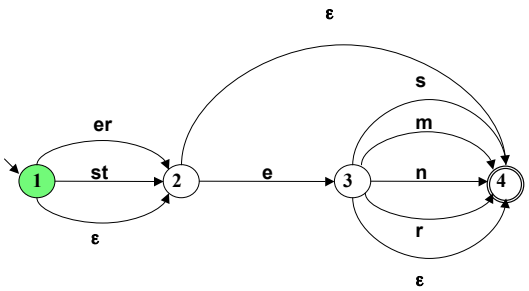
Pfadsuche: Startkonfiguration: Startzustand und Eingabewort auf der Agenda



Eingabewort:

Agenda: 1 -- klein eres

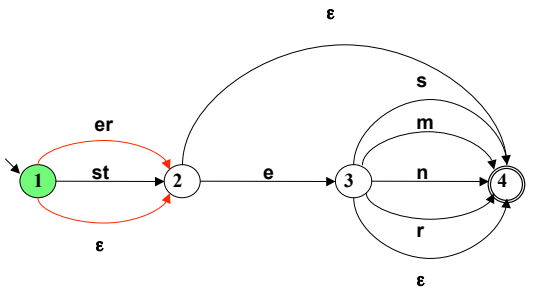
Pfadsuche: Nimm Aufgabe von der Agenda



Eingabewort: klein eres

Agenda:

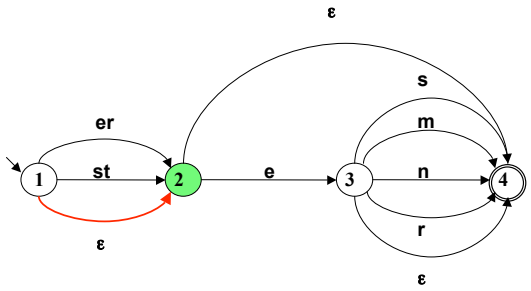
Pfadsuche: Generiere neue Aufgaben



Eingabewort: klein eres

Agenda: 2 -- klein eres
2 -- klein ere<u>s</u>

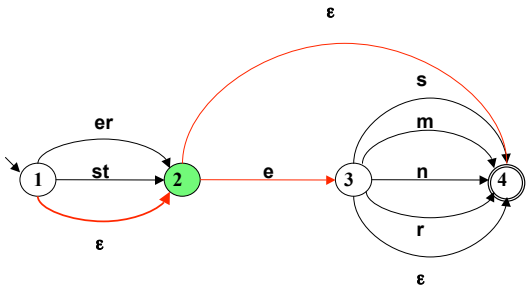
Pfadsuche: Nimm Aufgabe von der Agenda



Eingabewort: klein eres

Agenda: 2 -- klein eres

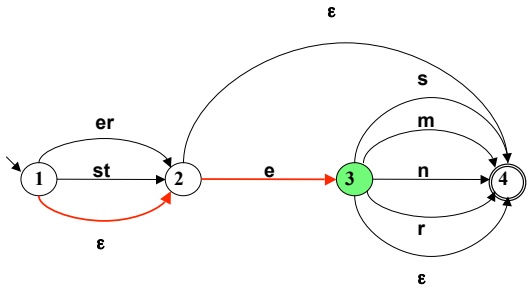
Pfadsuche: Generiere neue Aufgaben



Eingabewort: klein eres

Agenda: 3 -- klein eres
4 -- klein eres
2 -- klein eres

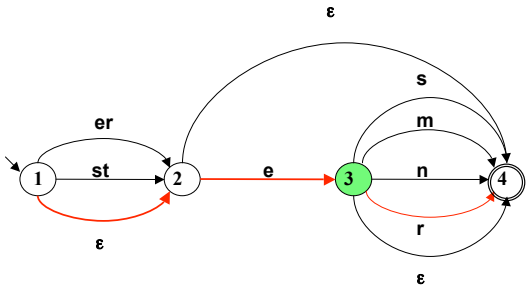
Pfadsuche: Nimm Aufgabe von der Agenda



Eingabewort: klein eres

Agenda: 4 -- klein eres
2 -- klein eres

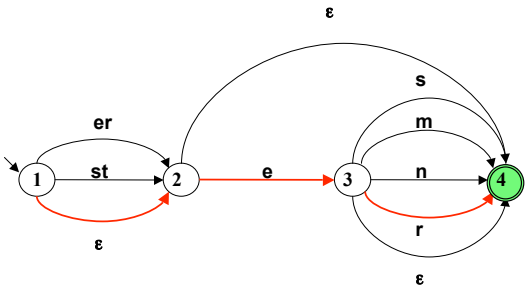
Pfadsuche: Generiere neue Aufgaben



Eingabewort: klein eres

Agenda: 4 -- klein eres
4 -- klein eres
2 -- klein eres

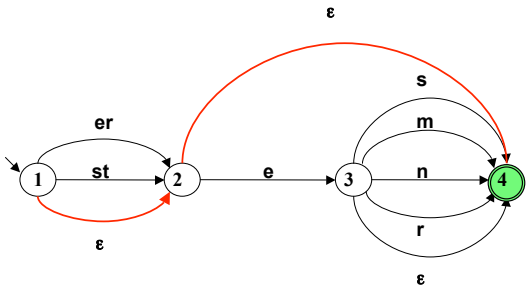
Pfadsuche: Nimm Aufgabe von der Agenda
Keine neue Aufgabe!



Eingabewort: klein eres

Agenda: 4 -- klein eres
2 -- klein eres

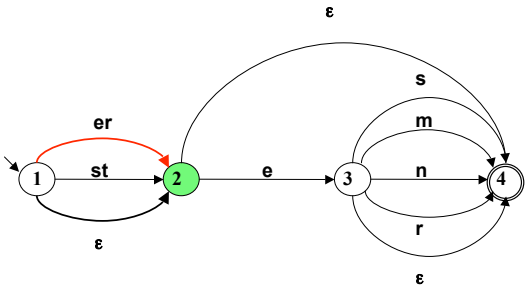
Pfadsuche: Nimm Aufgabe von der Agenda
Keine neue Aufgabe!



Eingabewort: klein eres

Agenda: 2 -- klein eres

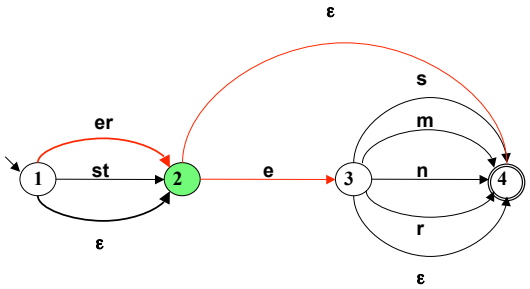
Pfadsuche: Nimm Aufgabe von der Agenda
Backtracking!



Eingabewort: klein eres

Agenda:

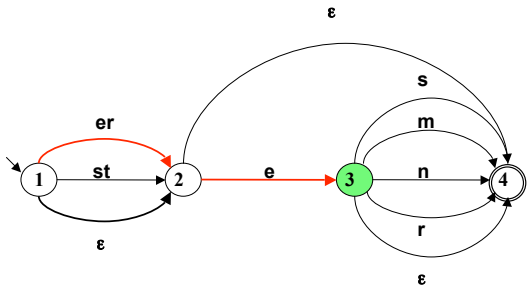
Pfadsuche: Generiere Aufgaben



Eingabewort: klein eres

Agenda: 3 -- klein eres
4 -- klein eres

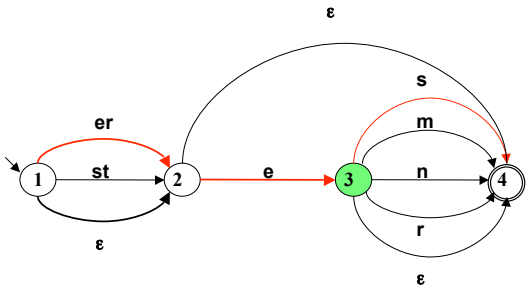
Pfadsuche: Nimm Aufgabe von der Agenda



Eingabewort: klein eres

Agenda: 4 -- klein eres

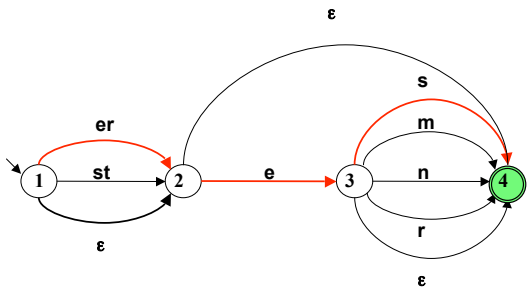
Pfadsuche: Generiere Aufgabe



Eingabewort: klein eres

Agenda: 4 -- klein eres_
4 -- klein eres

Pfadsuche: Nimm Aufgabe von der Agenda: Eingabe abgearbeitet, Zielzustand: Akzeptiere!



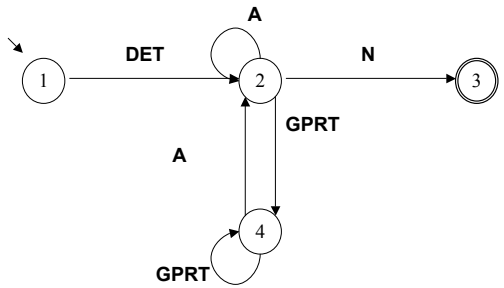
Eingabewort: klein eres_

Agenda: 4 -- klein eres

Anmerkung zum Pfadsuche-Algorithmus

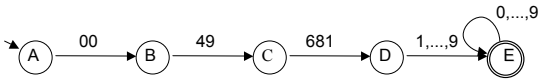
- Der vorgestellte Pfadsuche-Algorithmus ist ein Beispiel für „Tiefensuche mit Backtracking": Die Last In – First Out-Regel führt dazu, dass ein einmal gewählter Pfad so weit wie möglich weiter verfolgt wird. Wenn die Suche in eine Sackgasse gerät, werden systematisch die in der Agenda gespeicherten, noch nicht begangenen Verzweigungen ausprobiert.
- Der Algorithmus ist vollständig nur dann, wenn das Diagramm/der Automat keine Leerwort-Zyklen enthält (Schleifen, bei deren Durchlaufen das leere Wort abgearbeitet wird).
- Der Algorithmus ist ineffizient: Bei einem maximalen Verzweigungsfaktor n benötigt der Algorithmus zur vollständigen Abarbeitung eines Eingabewortes w im schlechtesten Fall $n^{|w|}$ Schritte. Der Zeitbedarf wächst exponentiell mit der Wortlänge.
- Man kann die Situation verbessern, indem man
  - die Information im Diagramm geschickt anordnet
  - den Suchalgorithmus optimiert (z.B. durch Testen, ob eine neu generierte Konfiguration schon einmal vorgekommen ist)
  - eine alternative Repräsentation wählt, die effizientere Verarbeitung erlaubt

Ein deterministisches Diagramm



Beobachtung: Bestimmte Diagramme erfordern keine Suche, weil Übergänge bei gegebenem Zustandund Eingabesymbol eindeutig festgelegt sind.

Saarbrücker Telefonnummern (international)



Deterministische endliche Automaten

- Die beiden Diagramme unterscheiden sich von dem Adjektiv-Diagramm in einem wesentlichen Punkt: Für jeden Zustand/Knoten und jede Eingabe gibt es höchstens eine Kante, die beschritten werden kann. Sie sind deterministisch.
- Die Definition des „deterministischen endlichen Automaten" (DEA oder DFA, für „deterministic finite-state automaton") führt einige weitere, weniger wesentliche, aber nützliche Beschränkungen gegenüber dem NEA ein.

Deterministische und nicht-deterministische Automaten

- NEA erlaubt beliebige Worte (incl. e) als Kanteninschrift
- NEA erlaubt für einen Ausgangszustand und eine Eingabe mehrere oder gar keinen Zielzustand
- D.h.: NEA hat eine Übergangsrelation.

- DEA hat nur Einzelsymbole als Kanten-Inschriften, insbesondere sind Leerwort-Kanten nicht zulässig.
- DEA hat zu jedem Zustand und zu jedem Symbol genau eine wegführende Kante
- D.h.: DEA hat eine Übergangsfunktion.

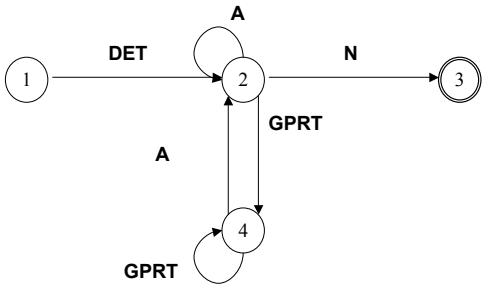
Definition: Deterministischer endlicher Automat

Ein deterministischer endlicher Automat ist ein Quintupel

A = <K, Σ, δ, s, F>, wobei

- K nicht-leere endliche Menge von Knoten (Zuständen)
- Σ nicht-leeres Alphabet
- s ∈ K Startzustand
- F ⊆ K Menge von Endzuständen
- δ : K × Σ → K Übergangsfunktion

Ein deterministisches Diagramm



Beispiel: Der DEA für Wortartmuster [1]

DEA A = <K, Σ, δ, s, F> mit

- K= {1,2,3,4}
- Σ = {DET, A, N, GPRT}
- s = 1
- F = {3}
- δ definiert durch: δ(1,DET) = 2
  δ(2, A) = 2
  δ(2,N) = 3
  δ (2,GPRT) = 4
  ... ...

Beispiel [2]: Übergangstabelle für δ

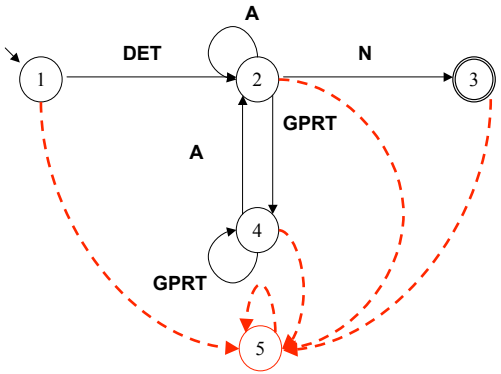
δ:	DET	A	N	GPRT
1	2			
2		2	3	4
3				
4		2		4

Beispiel [3]: Übergangstabelle für δ, komplettiert

δ:	DET	A	N	GPRT
1	2	5	5	5
2	5	2	3	4
3	5	5	5	5
4	5	2	5	4
5	5	5	5	5

- Der Zustand eines DEA, aus dem es keine Möglichkeit gibt, in einen Endzustand zu gelangen, heißt „Senke“ oder engl. „trap state“: Falle.

Das Zustandsdiagramm für Wortartmuster: Übergangsfunktion δ komplettiert



Deterministische und nicht-deterministische Automaten [1]

- DEAs erlauben den Test von Eingabeketten in linearer Zeit: Jedes Wort der Länge n wird in genau n Schritten abgearbeitet.
- DEAs haben allerdings ein eingeschränkteres Beschreibungs-Inventar als NEAs.
- Frage: Ist deshalb die Ausdrucksstärke des DEA-Formalismus eingeschränkter als die von NEAs? Das heißt, gibt es Sprachen, die durch einen NEA, aber nicht durch einen DEA beschrieben werden?
- Die Antwort lautet: Nein!

Deterministische und nicht-deterministische Automaten [2]

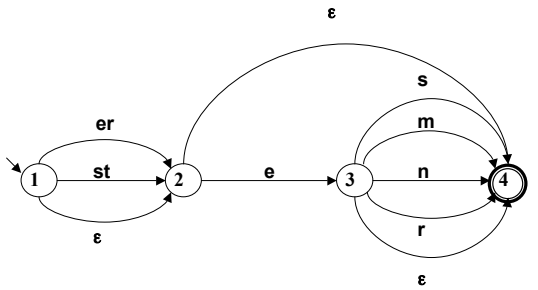
- Jede Sprache, die von einem NEA akzeptiert wird, kann auch durch einen DEA beschrieben werden (und, trivialerweise, auch umgekehrt: ein DEA ist ein spezieller NEA). NEAs und DEAs besitzen die gleiche Ausdrucksstärke, die Formalismen sind beschreibungsäquivalent.
- Das ist beweisbar. Noch wichtiger: Der Beweis ist konstruktiv, d.h.:
- Es gibt ein Konstruktionsverfahren, das es erlaubt, zu jedem NEA A einen DEA A' zu konstruieren, so dass L(A') = L(A).

Die NEA-DEA-Überführung

Der Algorithmus zur NEA-DEA-Überführung besteht aus drei Schritten:

1. Beseitigung von Mehrsymbol-Kanten
2. Beseitigung von ε-Kanten
3. Die „Potenz-Automaten“-Konstruktion

Adjektivendungen: Zustandsdiagramm



Schritt 1: Beseitigung von Mehrsymbolkanten

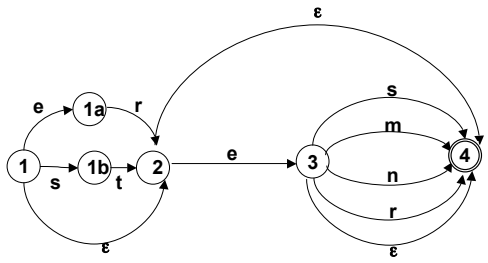
Gegeben sei der NEA $A = \langle K, \Sigma, \Delta, s, F \rangle$.

- Für alle Kanten $\langle q,w,q' \rangle$ mit $w = a_1 \ldots a_n$, $n > 1$:

  Entferne $\langle q,w,q' \rangle$ aus Δ.
- Erweitere K um neue Zustände $q_1, \ldots, q_{n-1}$.
- Erweitere Δ um neue Kanten

  $\langle q, a_1, q_1 \rangle, \langle q_1, a_2, q_2 \rangle, \ldots, \langle q_{n-1}, a_n, q' \rangle$

Beispiel-Automat nach Schritt 1:

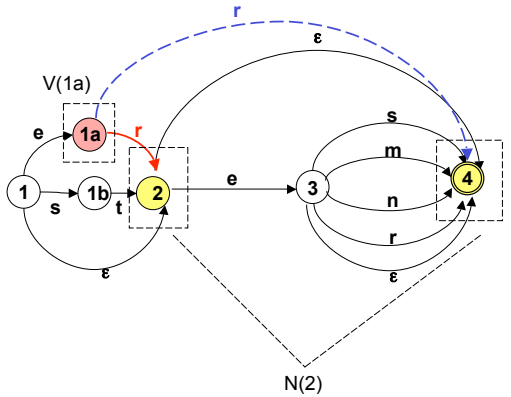


Schritt 2: Beseitigung von ε-kanten

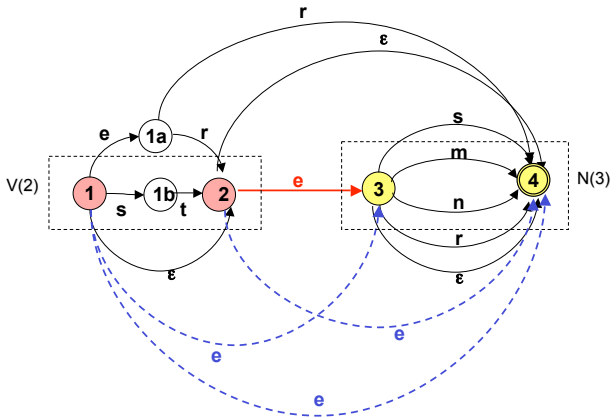
- Wir definieren zunächst als Hilfsbegriffe den „ε-Vorbereich" $V_\varepsilon(p)$ und den „ε-Nachbereich" $N_\varepsilon(p)$ von Zuständen:
  - $V_\varepsilon(p)$ = {q| p ist von q aus ohne Abarbeiten eines Symbols erreichbar}
  - $N_\varepsilon(p)$ = {q| q ist von p aus ohne Abarbeiten eines Symbols erreichbar}

  Anmerkung: $V_\varepsilon(p)$ und $N_\varepsilon(p)$ enthalten insbesondere p selbst.
- Für jede nicht-leere Kante <p, a, q> ∈ Δ: Erweitere Δ um alle <p', a, q'> mit p' ∈ $V_\varepsilon(p)$, q'∈$N_\varepsilon(q)$.
- Entferne alle leeren Kanten aus Δ.

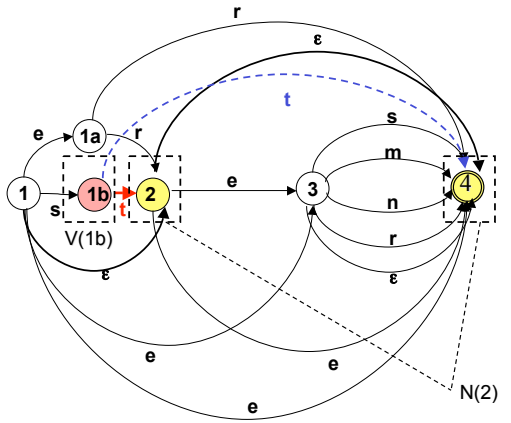
Schritt 2: Beseitigung von ε-Kanten



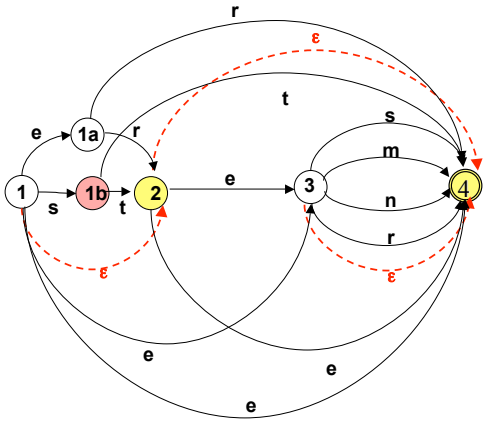
Schritt 2: Beseitigung von ε-kanten



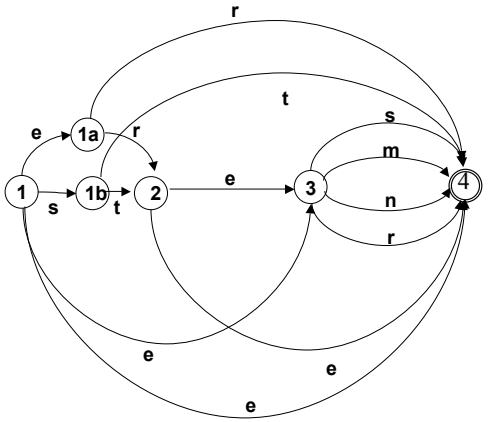
Schritt 2: Beseitigung von ε-kanten



Schritt 2: Beseitigung von ε-kanten



Schritt 2: Beseitigung von ε-kanten: Resultat ist „buchstabierender Automat"



Schritt 2: Beseitigung von ε-kanten

- Wir definieren zunächst als Hilfsbegriffe den „ε-Vorbereich" $V_\varepsilon(p)$ und den „ε-Nachbereich" $N_\varepsilon(p)$ von Zuständen:
  - $V_\varepsilon(p)$ = {q| p ist von q aus ohne Abarbeiten eines Symbols erreichbar}
  - $N_\varepsilon(p)$ = {q| q ist von p aus ohne Abarbeiten eines Symbols erreichbar}

  Anmerkung: $V_\varepsilon(p)$ und $N_\varepsilon(p)$ enthalten insbesondere p selbst.
- Für jede nicht-leere Kante <p, a, q> ∈ Δ: Erweitere Δ um alle <p', a, q'> mit $p' \in V_\varepsilon(p)$, $q' \in N_\varepsilon(q)$.
- Entferne alle leeren Kanten aus Δ.
- Wenn sich ein Endzustand im ε-Nachbereich des Startzustandes s befindet, füge s zu den Endzuständen hinzu.

Schritt 2: Beseitigung von ε-kanten: Resultat ist „buchstabierender Automat"