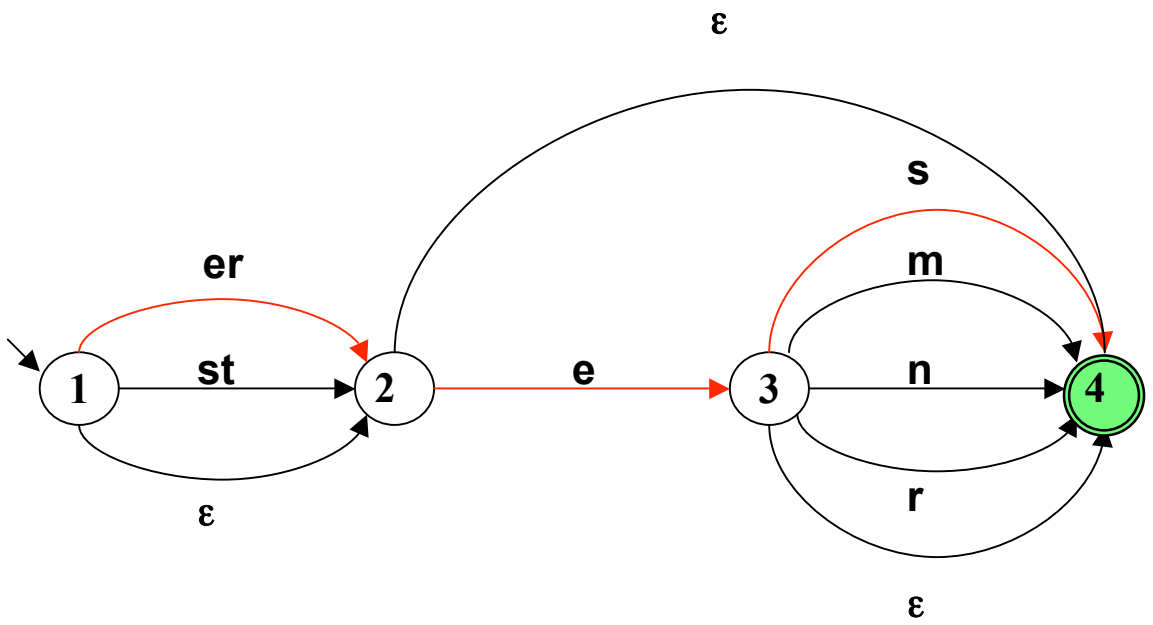
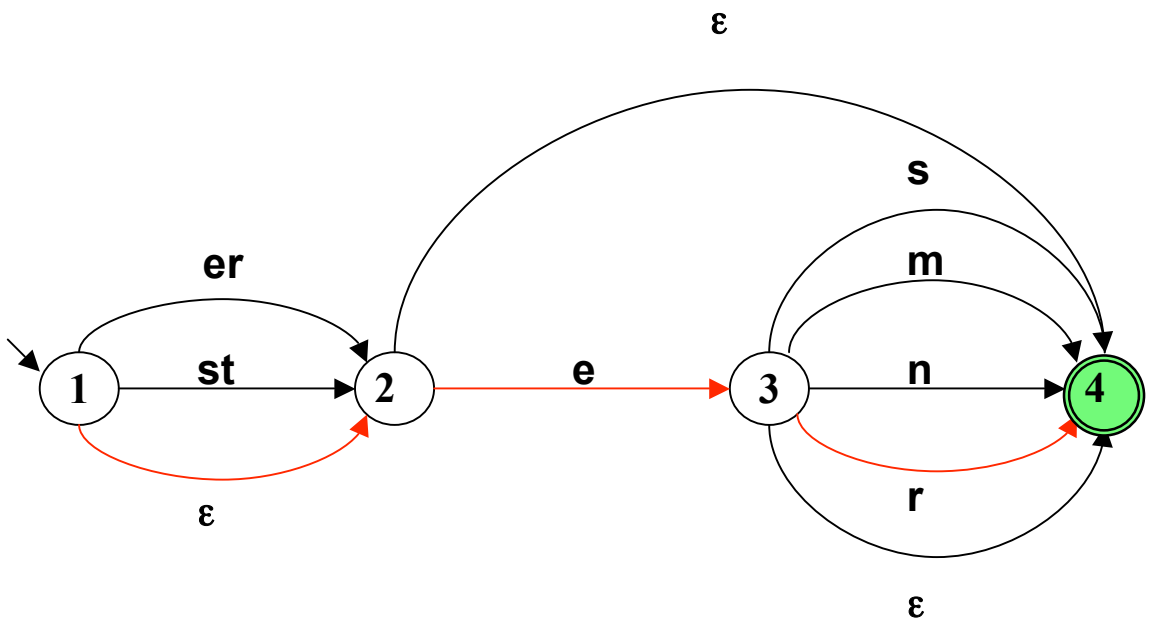
Ein Weg durchs Diagramm



klein eres

Ein alternativer Weg durchs Diagramm



klein er|es

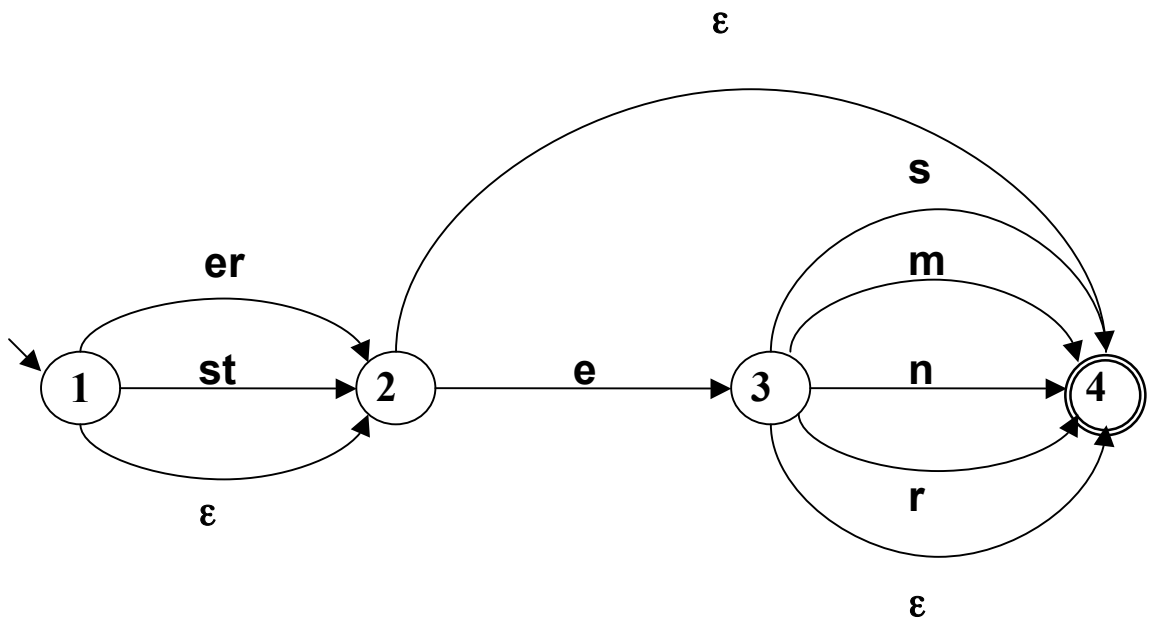
Das Problem

- 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 gleich vor, sondern legen die alternativen 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?
- Wenn wir die Agenda als Stapel („Stack“) betrachten, legen wir 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!).

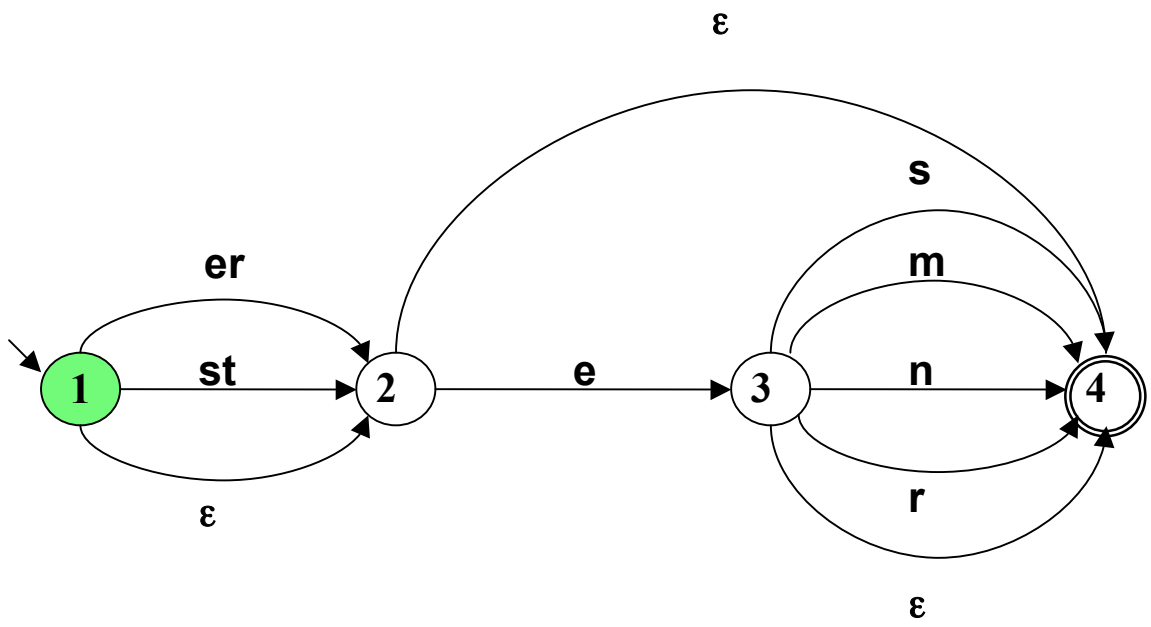
Pfadsuche: Startkonfiguration: Startzustand und Eingabewort auf der Agenda



Eingabewort:

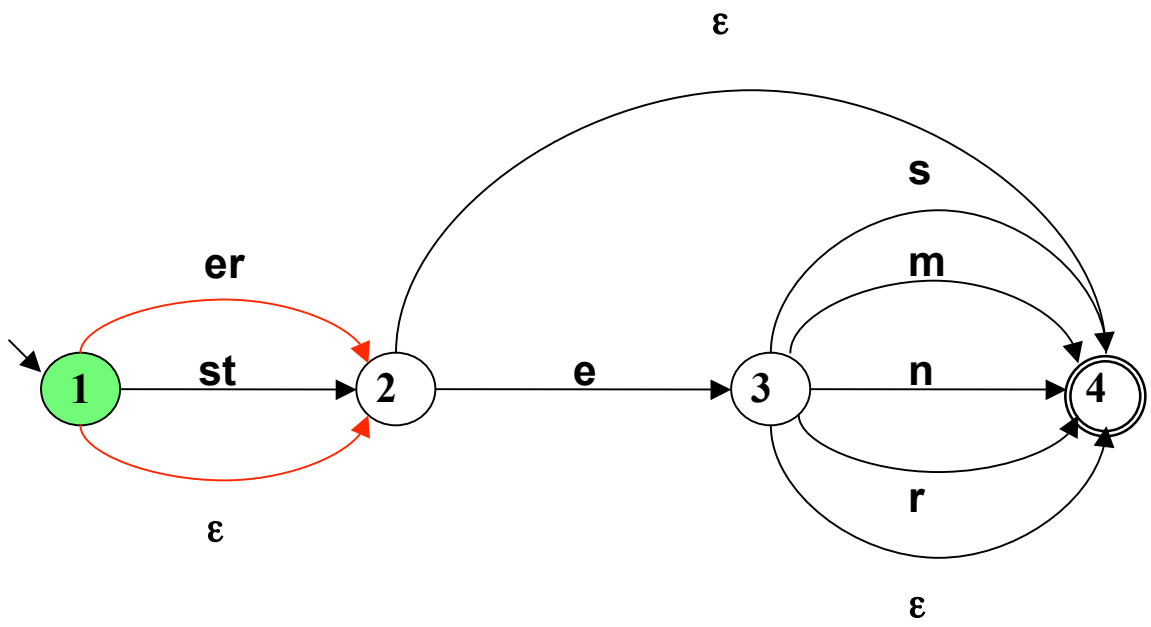
Agenda: 1 -- klein eres

Pfadsuche: Nimm Aufgabe von der Agenda



Eingabewort: klein eres Agenda: _____

Pfadsuche: Generiere neue Aufgaben

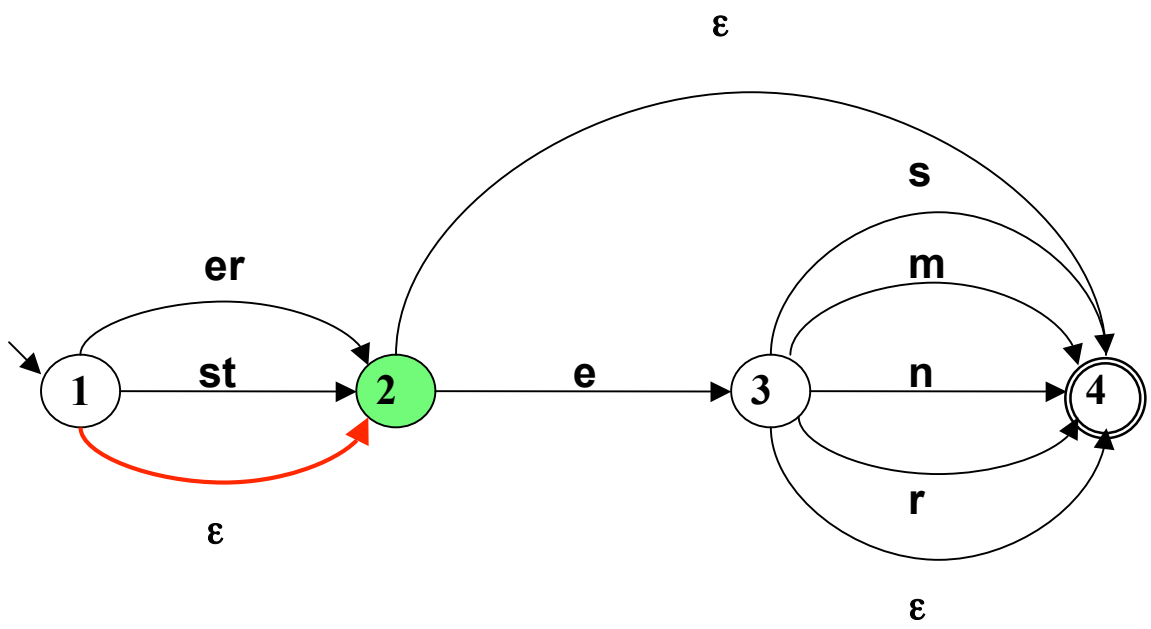


Eingabewort: klein eres

Agenda:

2 -- klein eres
2 -- klein eres

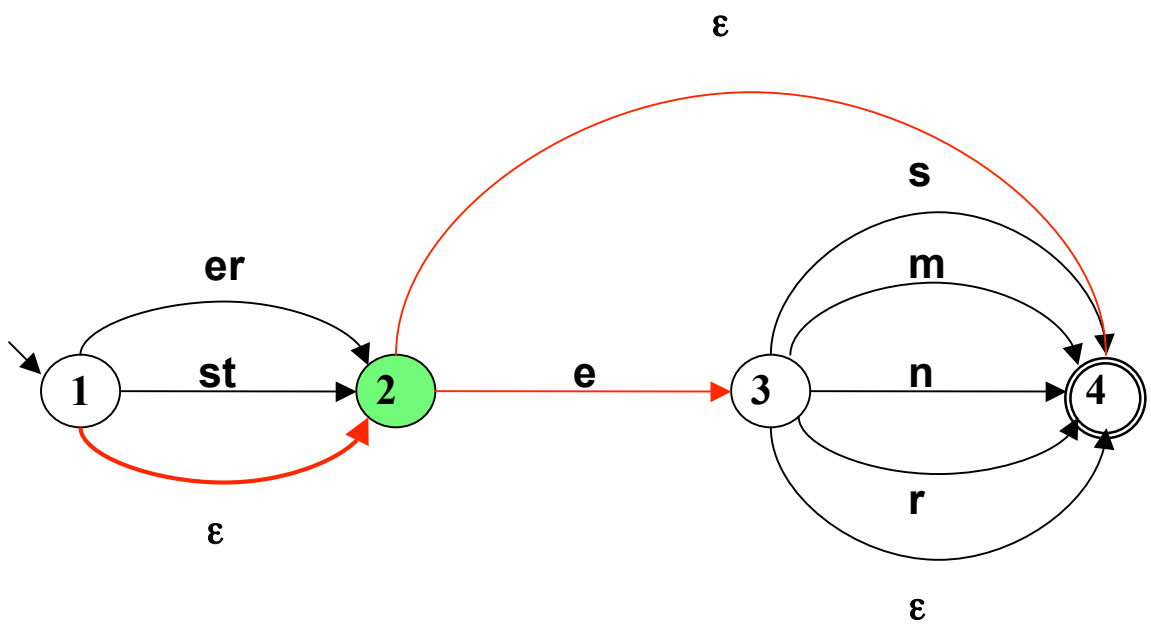
Pfadsuche: Nimm Aufgabe von der Agenda



Eingabewort: klein eres

Agenda: 2 -- klein eres

Pfadsuche: Generiere neue Aufgaben

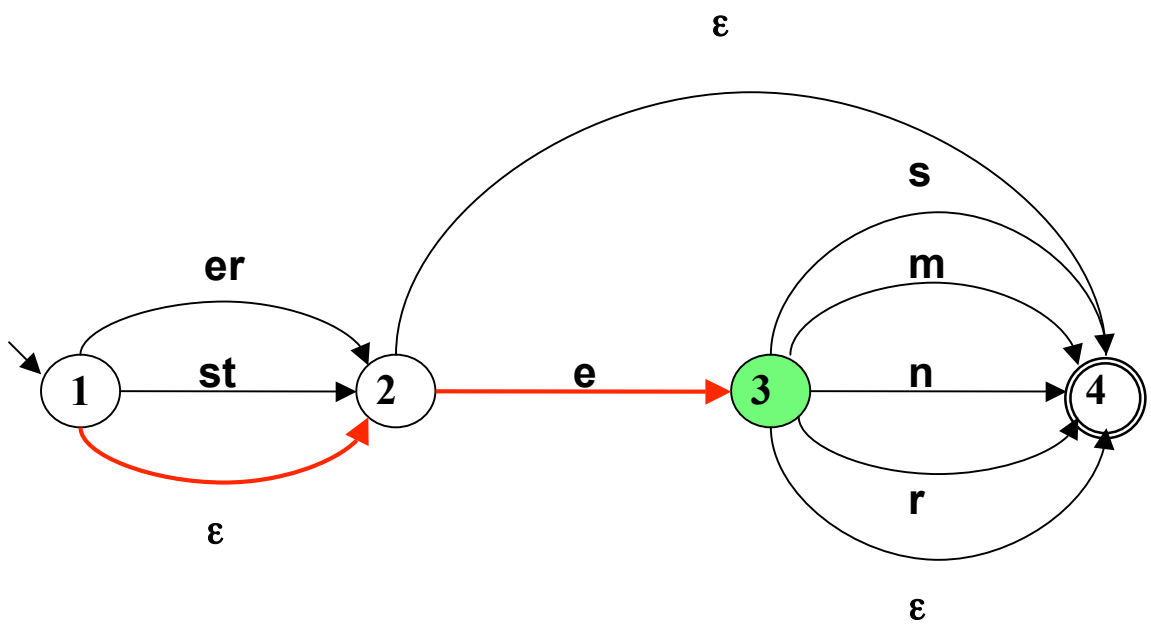


Eingabewort: klein eres

Agenda:

3 -- klein eres
4 -- klein eres
2 -- klein eres

Pfadsuche: Nimm Aufgabe von der Agenda

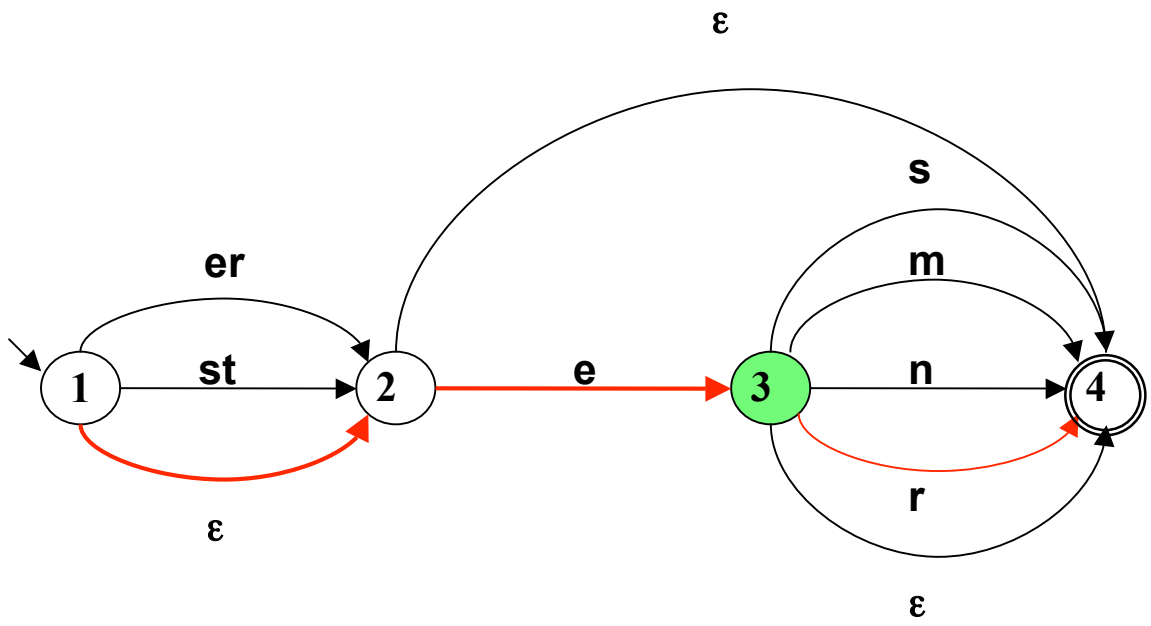


Eingabewort: klein eres

Agenda:

4 -- klein eres
2 -- klein eres

Pfadsuche: Generiere neue Aufgaben

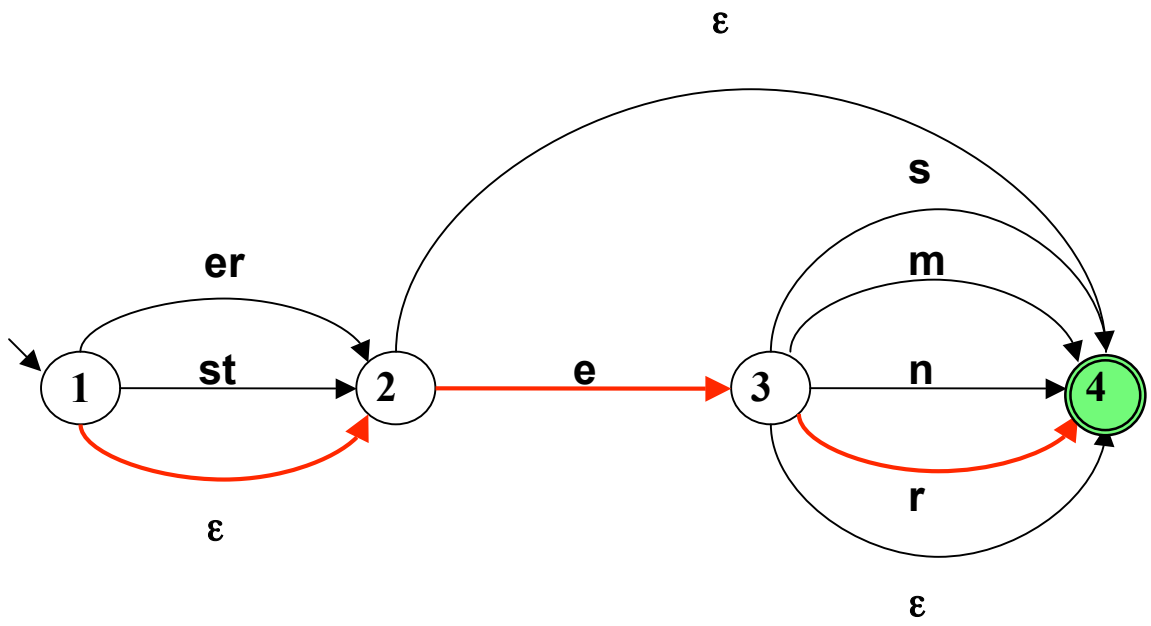


Eingabewort: klein eres

Agenda:

4 -- klein eres
4 -- klein eres
2 -- klein eres

Pfadsuche: Nimm Aufgabe von der Agenda
Keine neue Aufgabe!

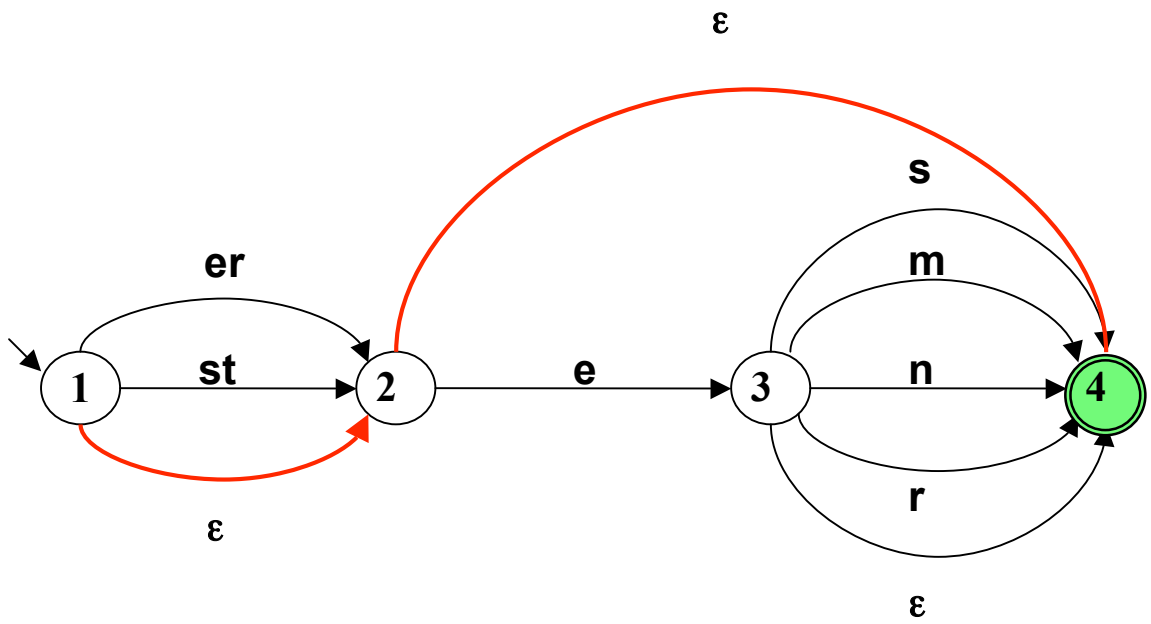


Eingabewort: klein eres

Agenda:

4 -- klein eres
2 -- klein eres

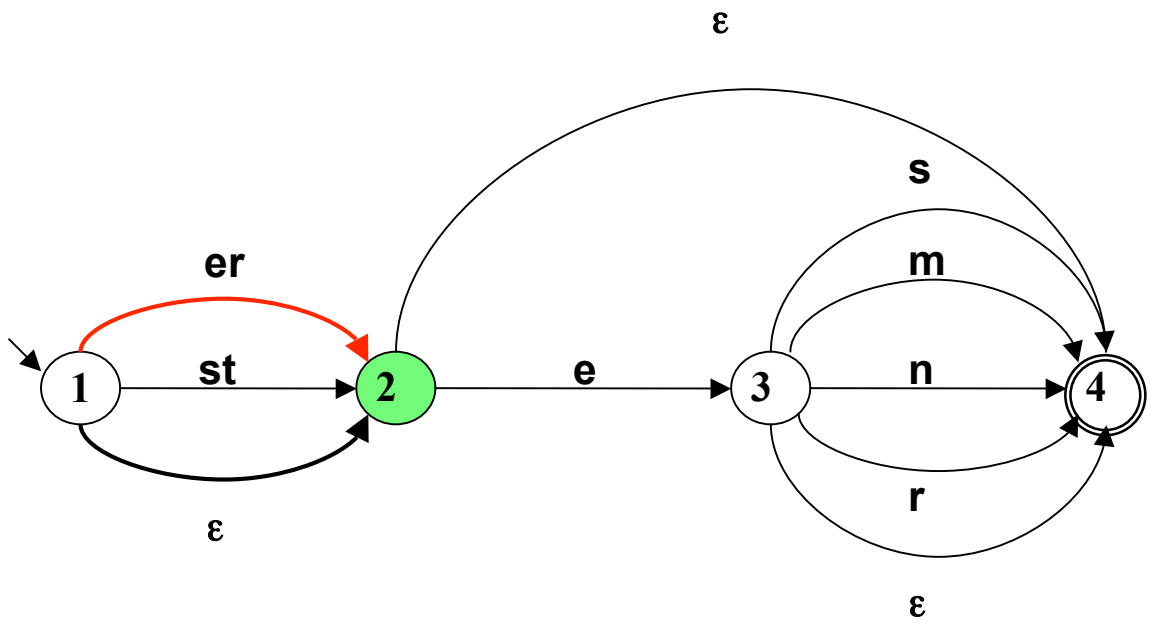
Pfadsuche: Nimm Aufgabe von der Agenda
Keine neue Aufgabe!



Eingabewort: klein eres

Agenda: 2 -- klein eres

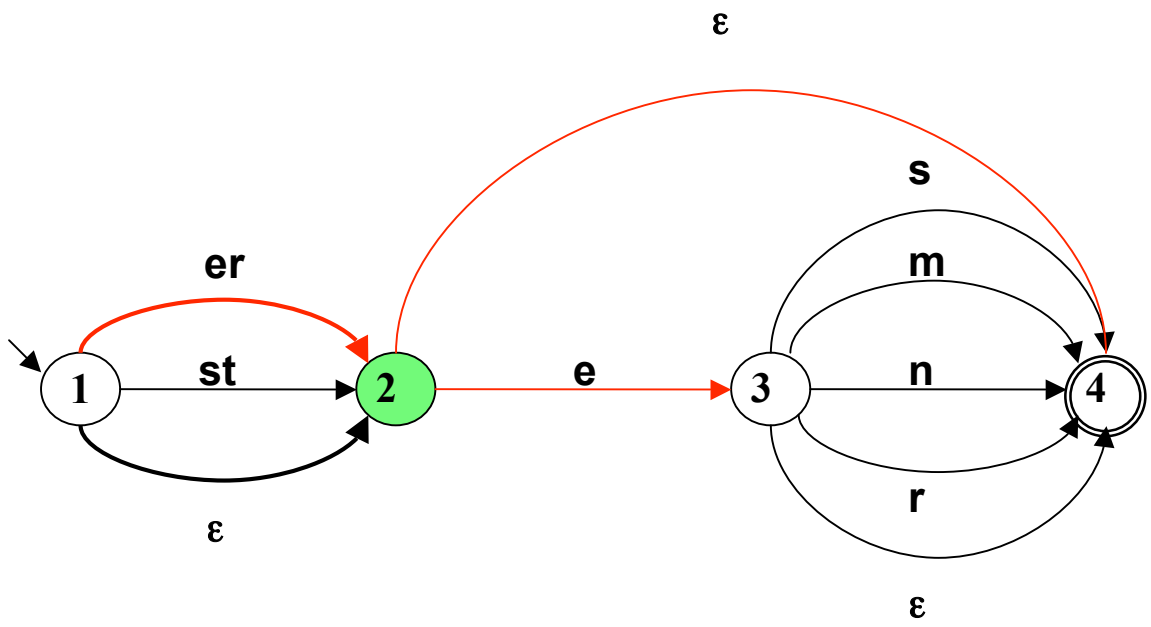
Pfadsuche: Nimm Aufgabe von der Agenda
Backtracking!



Eingabewort: klein eres

Agenda: _____

Pfadsuche: Generiere Aufgaben



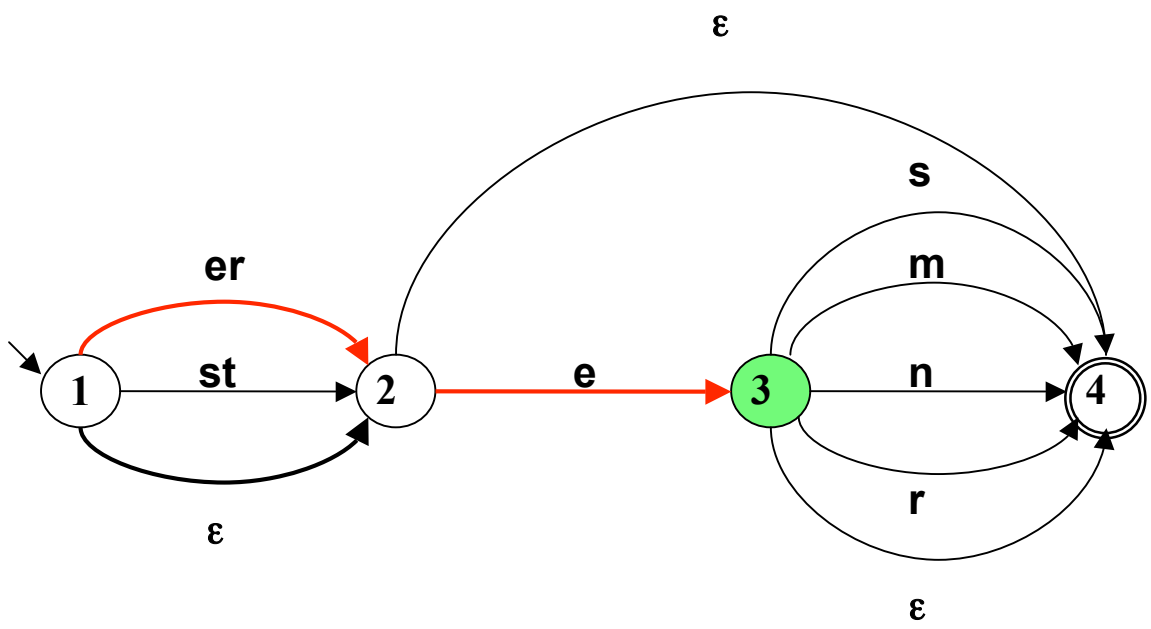
Eingabewort: klein er<u>e</u>s

Agenda:

3 -- klein ere<u>s</u>

4 -- klein er<u>e</u>s

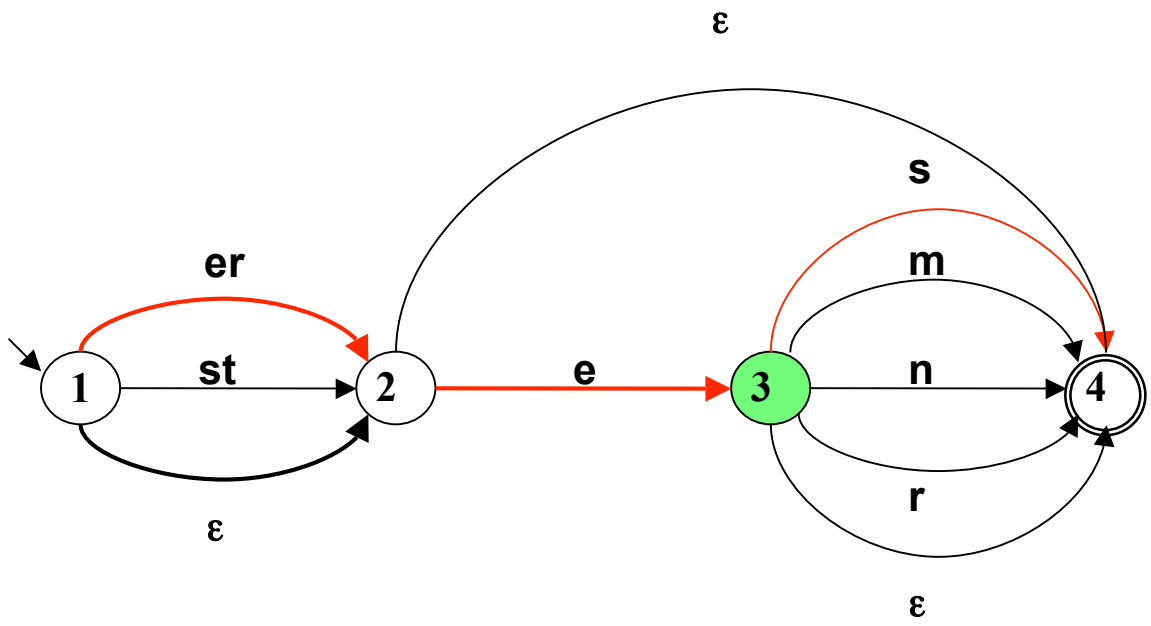
Pfadsuche: Nimm Aufgabe von der Agenda



Eingabewort: klein eres

Agenda: 4 -- klein eres

Pfadsuche: Generiere Aufgabe

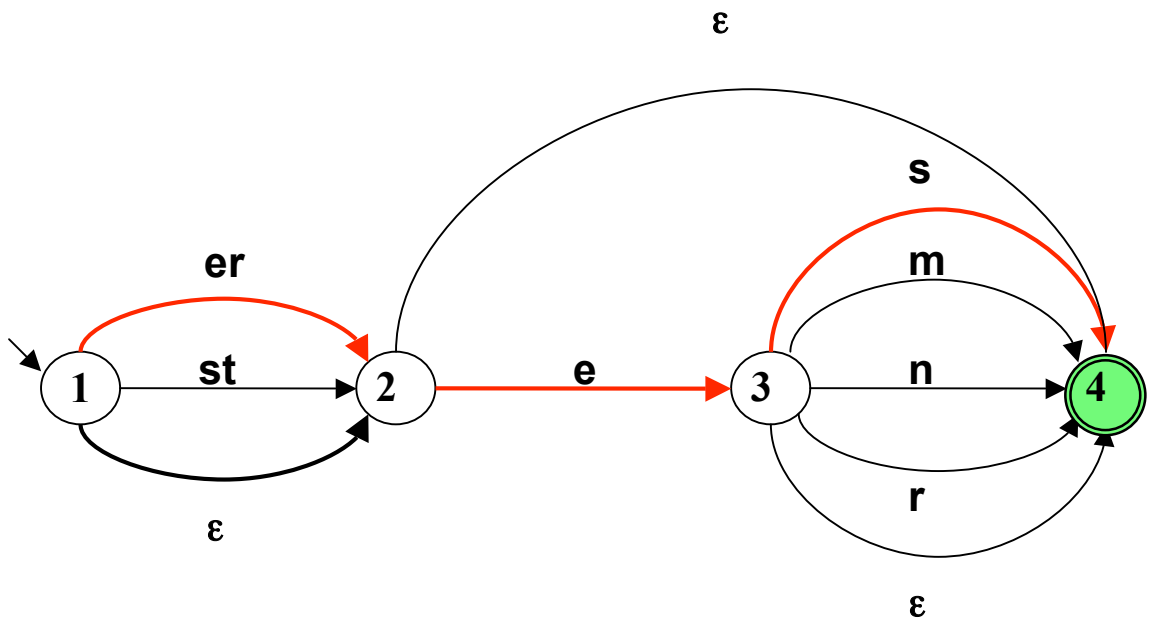


Eingabewort: klein eres

Agenda:

4 -- klein eres_
4 -- klein eres

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

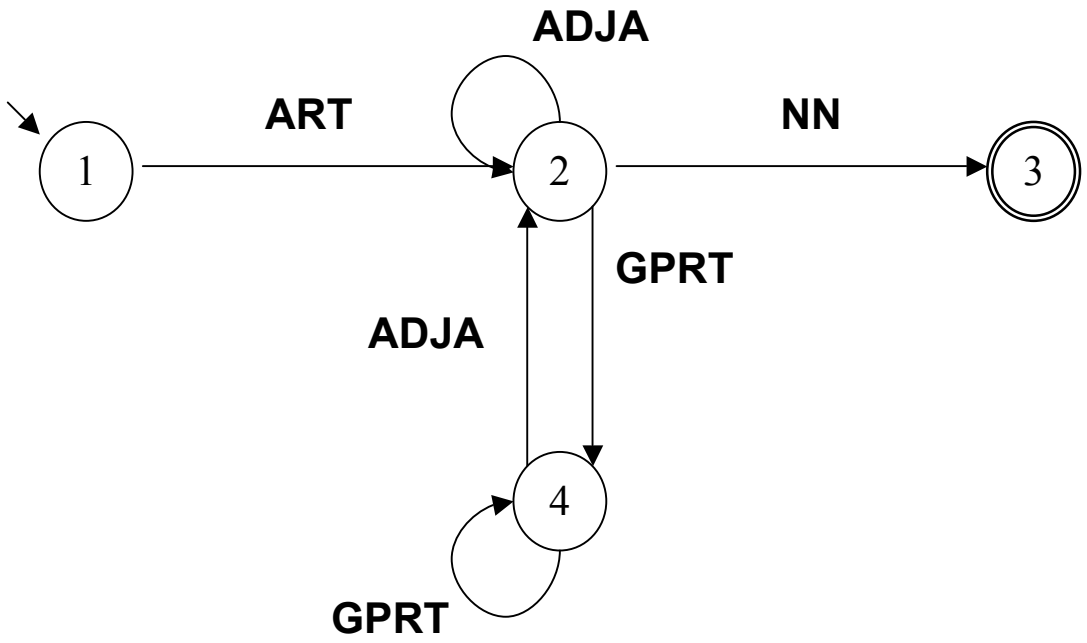


Eingabewort: klein eres_ Agenda: 4 -- klein eres

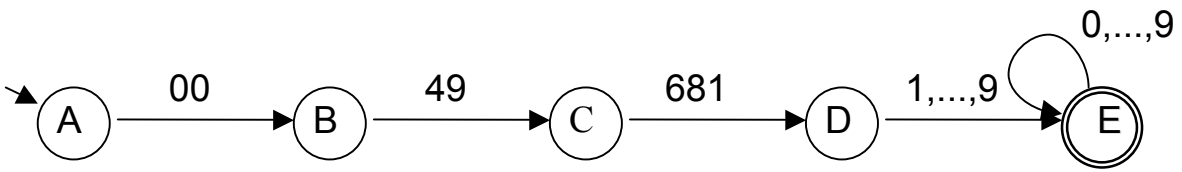
Anmerkung zum Pfadsuche-Algorithmus

- 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 Algirithmus 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: Wenn es für jede Konfiguration (Kombination aus Zustand und Eingabewort) durchschnittlich zwei erlaubte Kanten gibt, benötigt der Algorithmus zur vollständigen Abarbeitung eines Eingabewortes w im schlechtesten Fall etwa $2^{|w|}$ Schritte. Der Zeitbedarf wächst exponentiell mit der Wortlänge.
- Man kann die Situation verbessern (z.B. durch Testen, ob eine neu generierte Andem 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

Das Zustandsdiagramm für Wortartmuster: Ein deterministisches Diagramm



Internationale Vorwahl für Saarbrücken



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 „determinstc 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. ε) 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

$A = \langle K, \Sigma, \delta, s, F \rangle$, wobei

- K nicht-leere endliche Menge von Knoten (Zuständen)
- Σ nicht-leeres Alphabet
- $s \in K$ Startzustand
- $F \subseteq K$ Menge von Endzuständen
- $\delta : K \times \Sigma \rightarrow K$ Übergangsfunktion

Beispiel: Das Zustandsdiagramm für Wortartmuster [1]

DEA A = <K, Σ, δ, s, F> mit

- K= {1,2,3,4}
- Σ = {ART, ADJA, NN, GPRT}
- s = 1
- F = {3}
- δ definiert durch: δ(1,ART) = 2
  δ(2, ADJA) = 2
  δ(2,NN) = 3
  δ (2,GPRT) = 4
  ... ...

Beispiel [2]: Übergangstabelle für δ

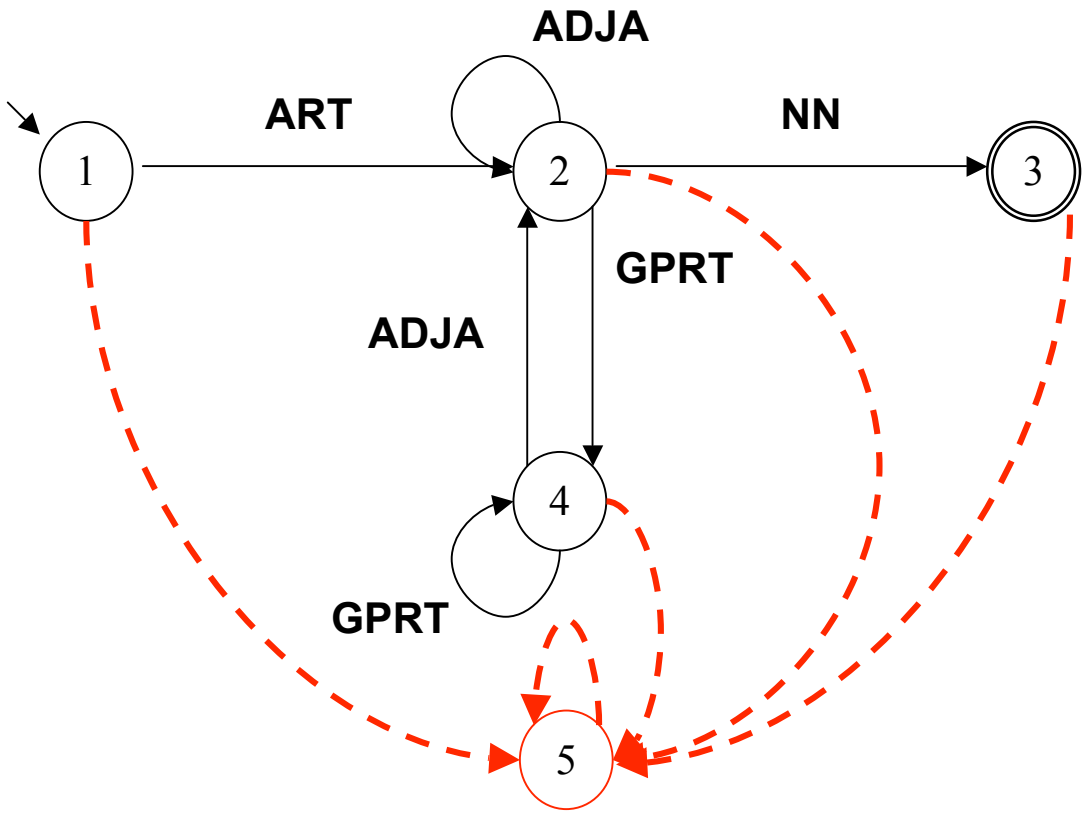
δ:	ART	ADJA	NN	GPRT
1	2			
2		2	3	4
3				
4		2		4

Beispiel [3]: Übergangstabelle für δ, komplettiert

δ:	ART	ADJA	NN	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: Nein.

Deterministische und nicht-deterministische Automaten [2]

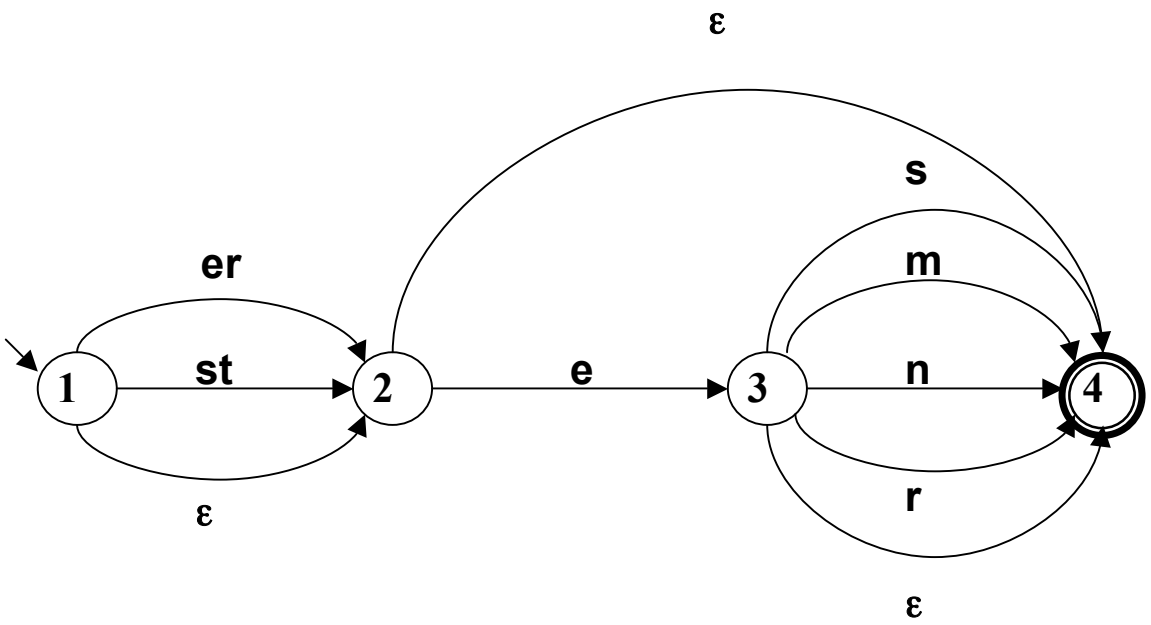
- Jede Sprache, die von einem NEA akzeptiert wird, kann auch durch einen DEA beschrieben werden (und, trivialerweise, auch umgekehrt: ein DEA ist ein spezieller NEA).
- Es gibt ein Konstruktionsverfahren, d.h., ein Verfahren, das es erlaubt, zu jedem NEA A einen DEA A‘ zu konstruieren, so dass L(A‘) = 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 $A = \langle K, \Sigma, \Delta, s, F \rangle$.

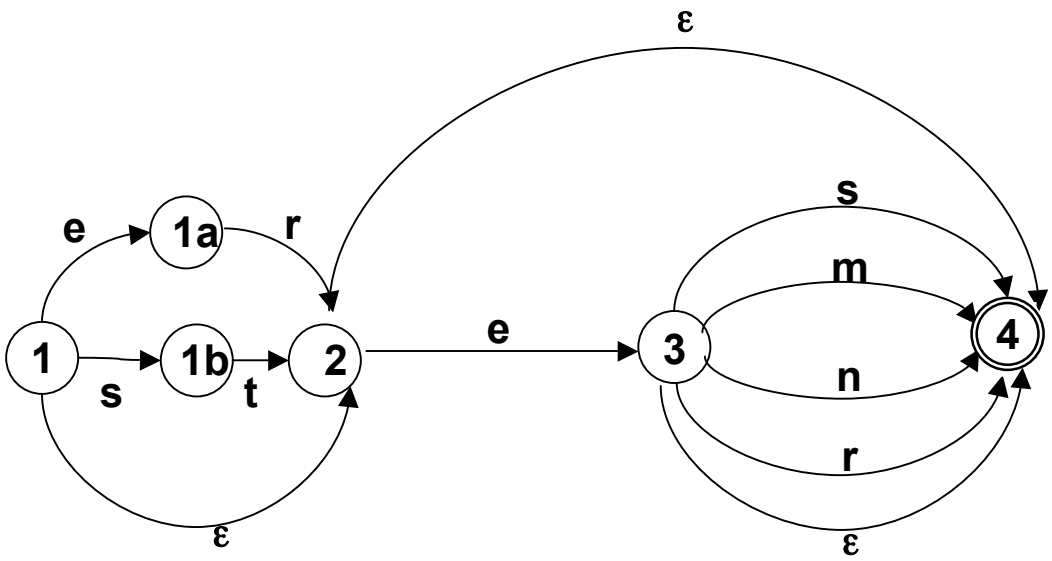
- Für alle Kanten $\langle q,w,q' \rangle$ mit $w = a_1 \ldots a_n$, $n > 1$:

  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:



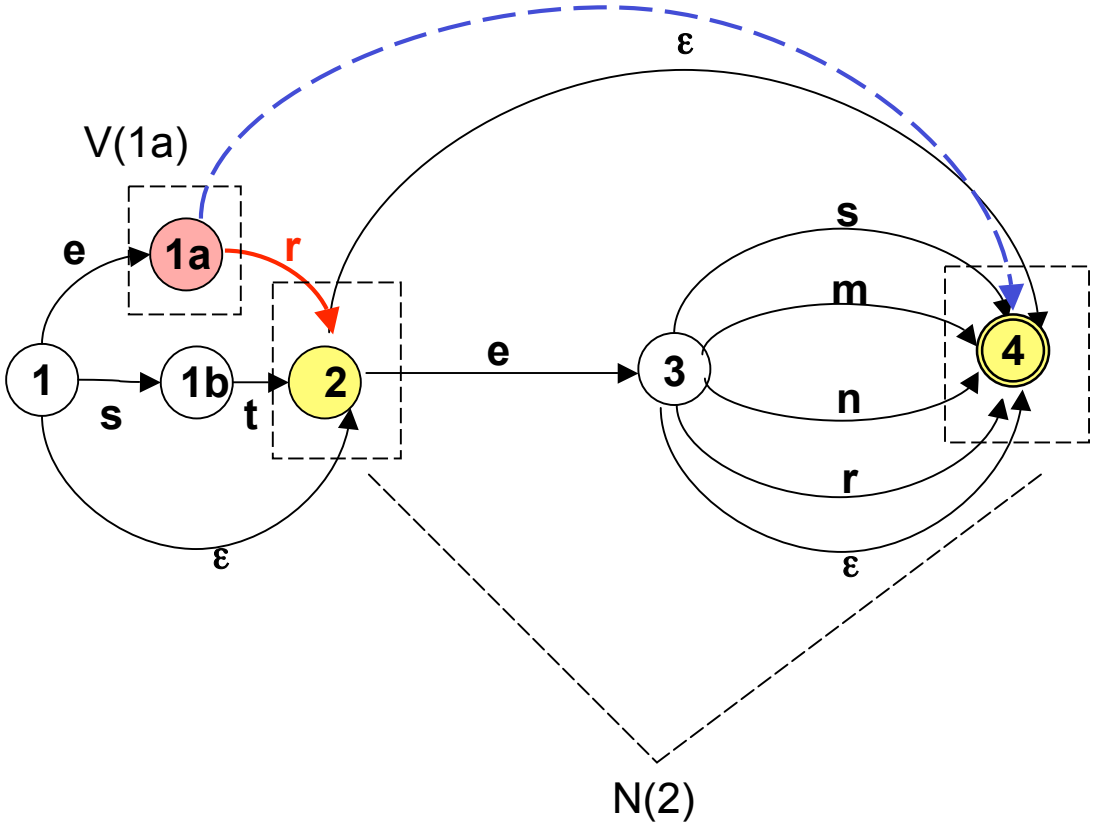
Schritt 2: Beseitigung von ε-kanten

- 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$\}$

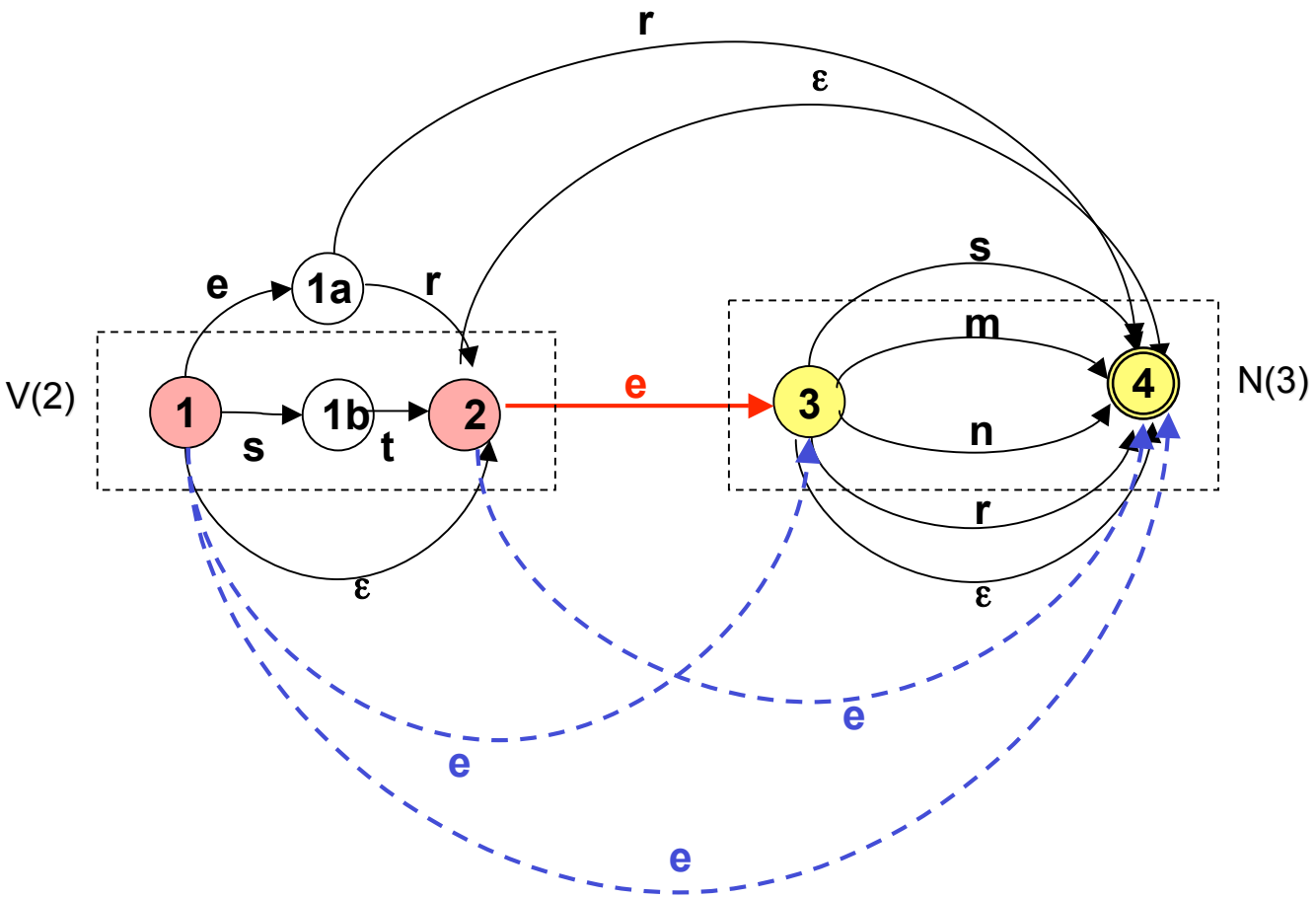
  Anmerkung: $V_\varepsilon(p)$ und $N_\varepsilon(p)$ enthalten insbesondere p selbst.

- Für jede nicht-leere Kante $<p, a, q> \in \Delta$: 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.

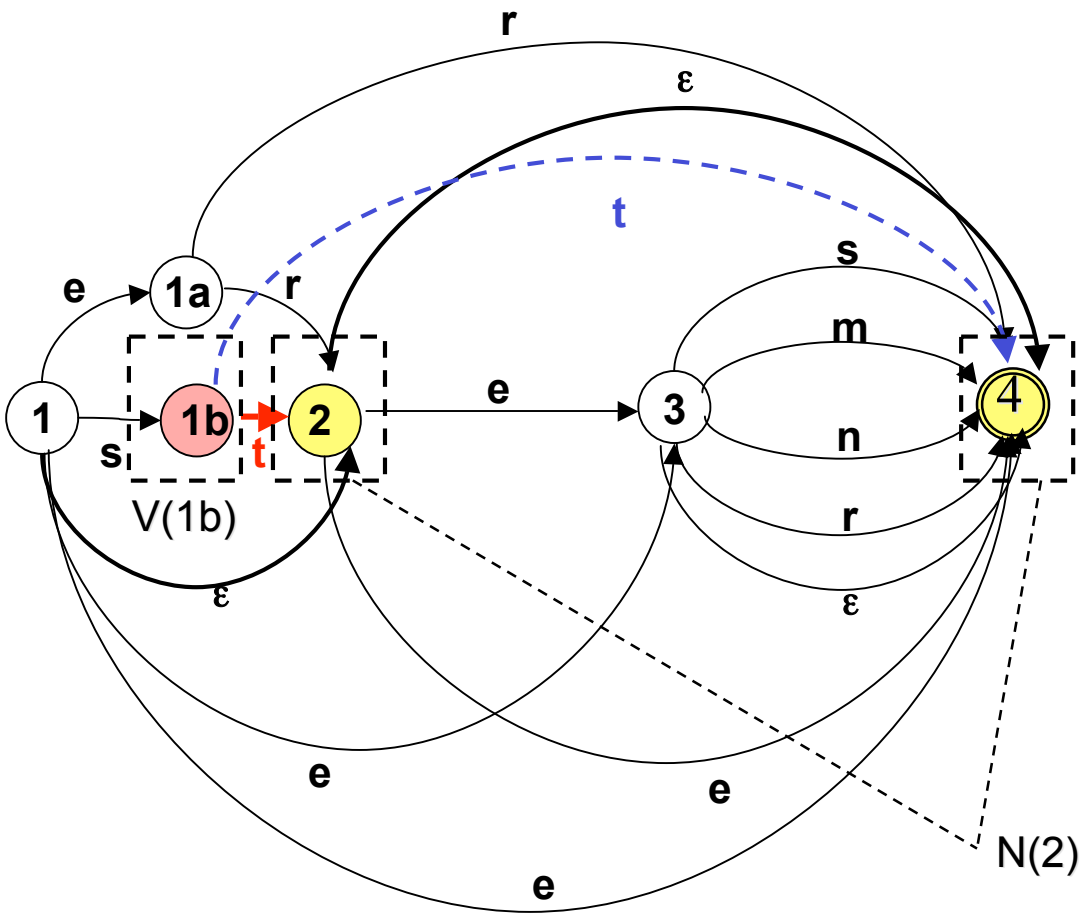
Schritt 2: Beseitigung von ε-Kanten



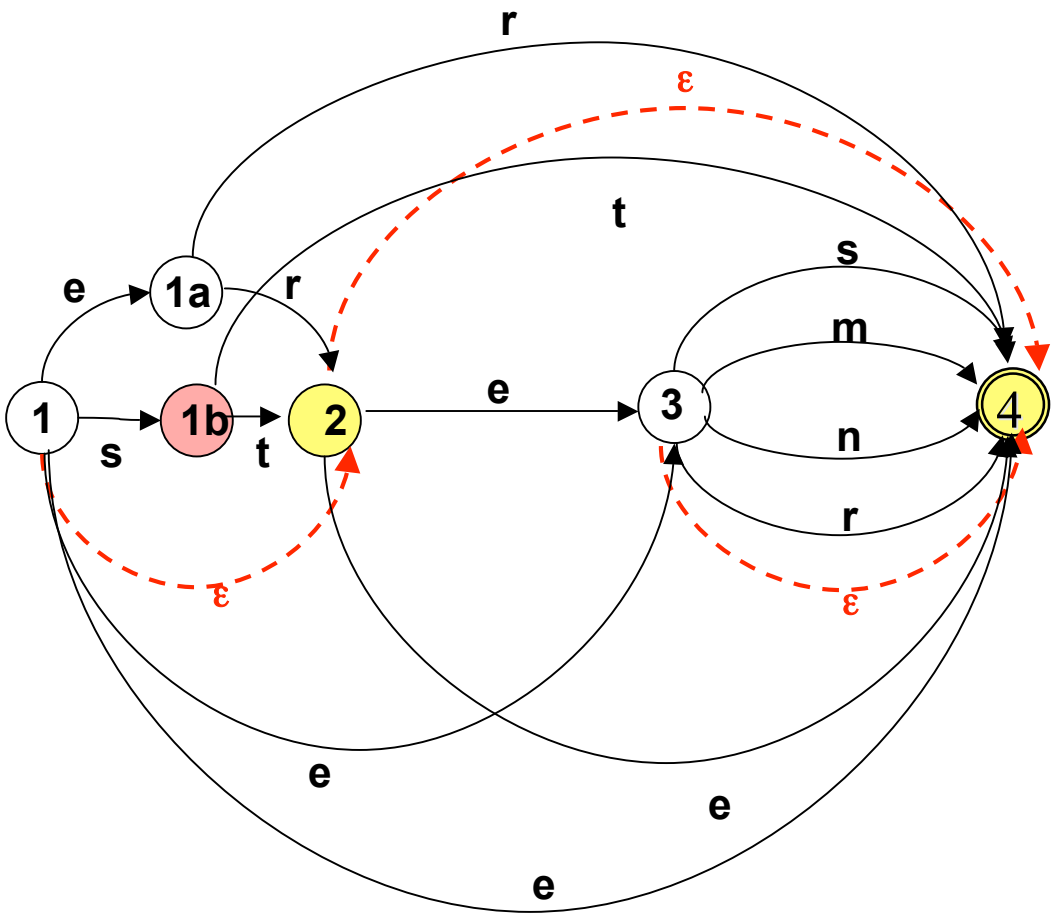
Schritt 2: Beseitigung von ε-kanten



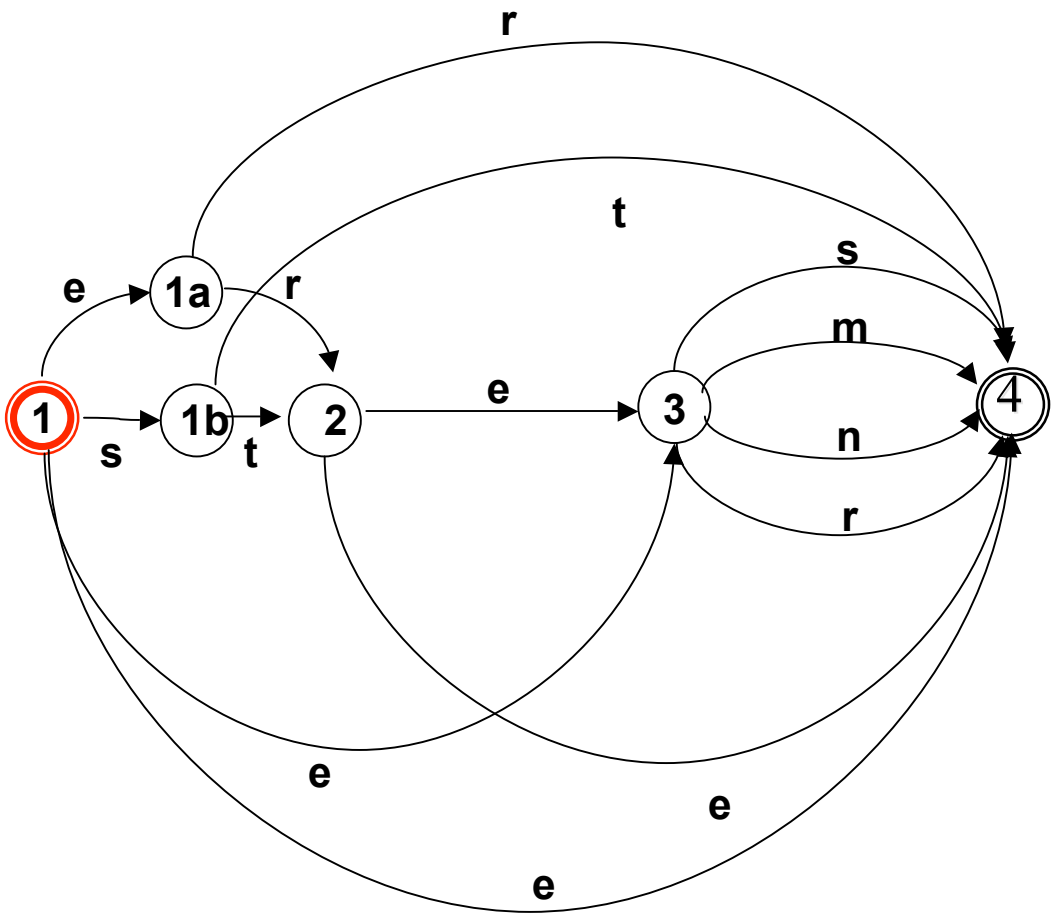
Schritt 2: Beseitigung von ε-kanten



Schritt 2: Beseitigung von ε-kanten



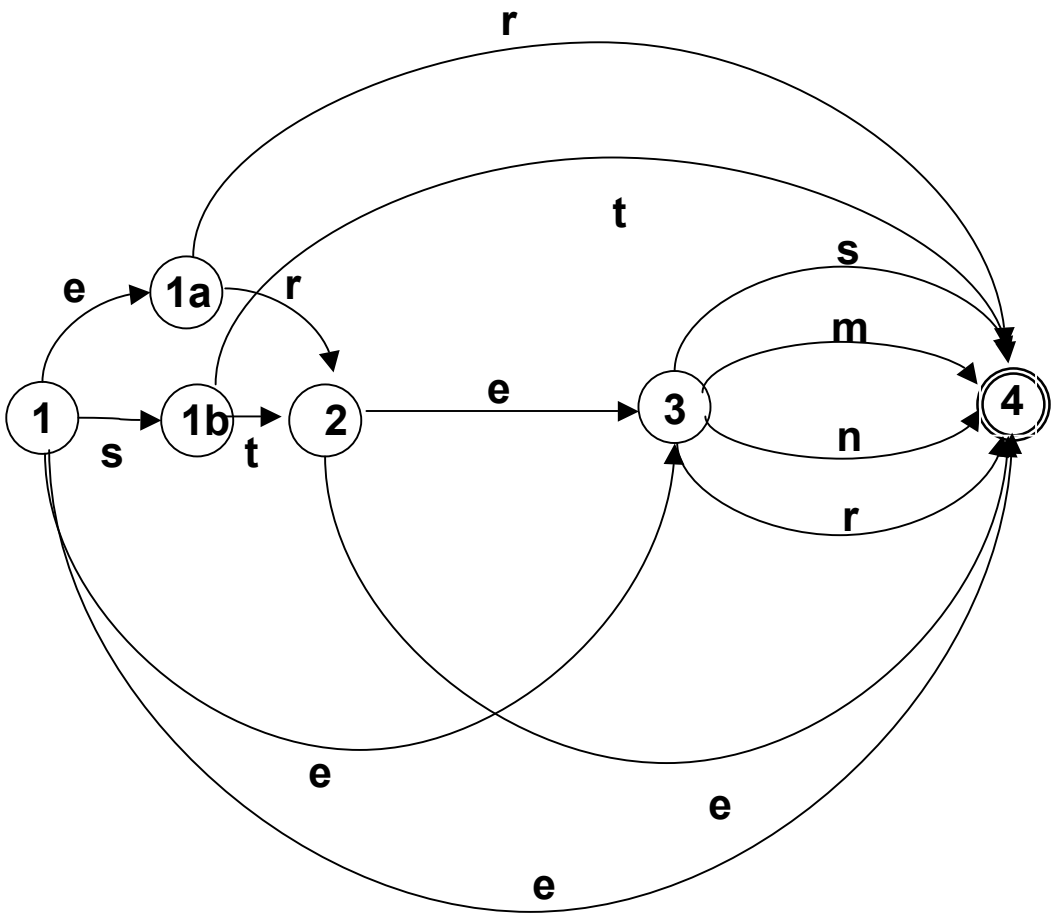
Schritt 2: Beseitigung von ε-kanten: Resultat ist „buchstabierender Automat“



Schritt 3: Potenzautomaten-Konstruktion, Vorüberlegung

- Wir haben einen Algorithmus zur Pfadsuche am Beispiel des unbearbeiteten Adjektivendungs-Diagramms kennengelernt: „Tiefensuche mit Backtracking“. Durch die Organisation der Agenda als Stapel/Stack („last in – first out“) wird eine Alternative so weit wie möglich verfolgt; bei endgültigem Scheitern wird das System zurückgesetzt.
- Durch die Organisation der Agenda als Warteschlange (queue), bei der die Aufgaben in der Reihenfolge ihrer Generierung abgearbeitet werden („first in – first out“), erhalten wir Breitensuche. Die alternativen Pfade werden (quasi) parallel verfolgt.

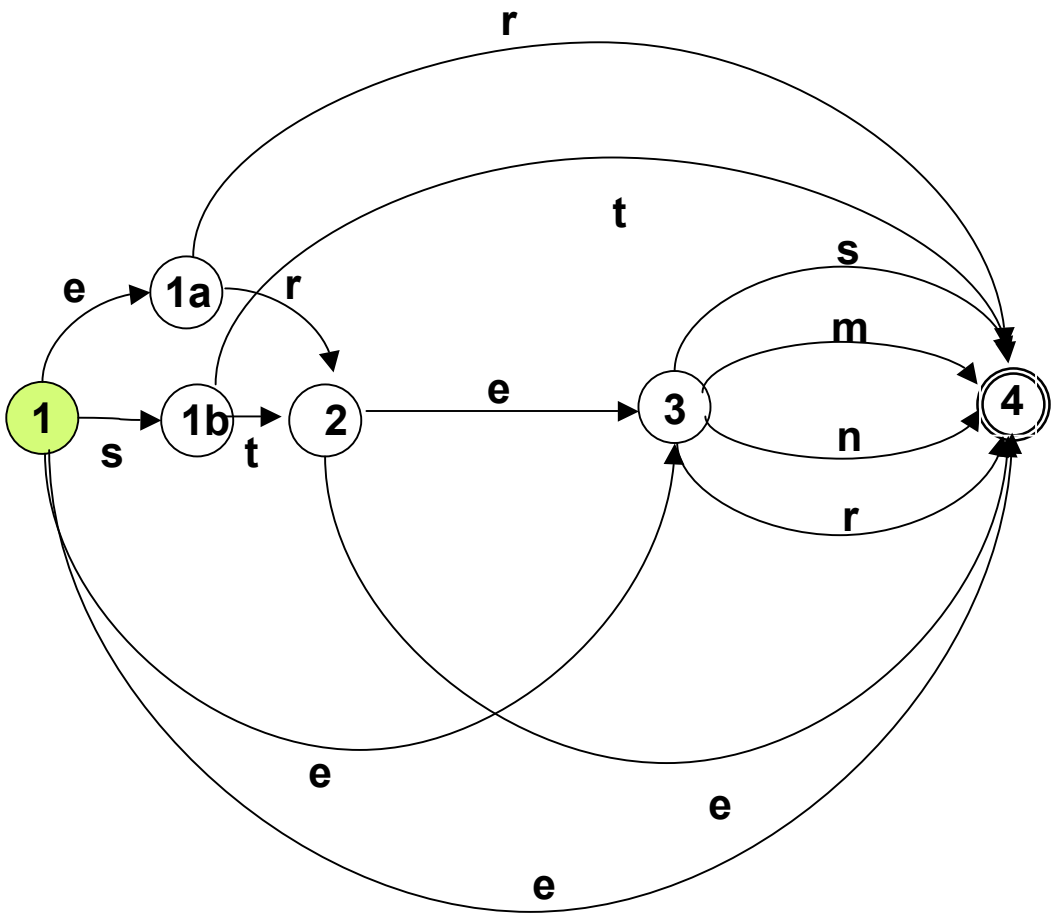
Pfadsuche als Breitensuche



Eingabewort:

Agenda: 1 -- klein eres

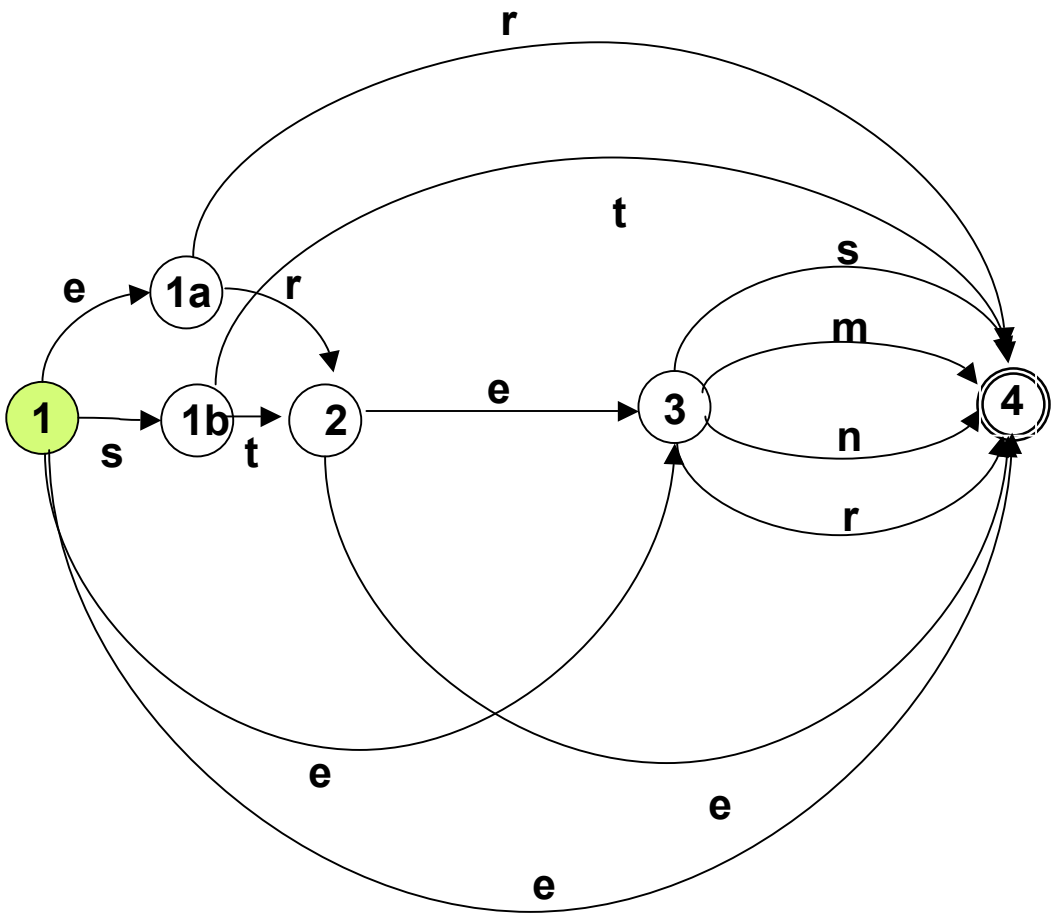
Pfadsuche als Breitensuche



Eingabewort: klein eres

Agenda: _____

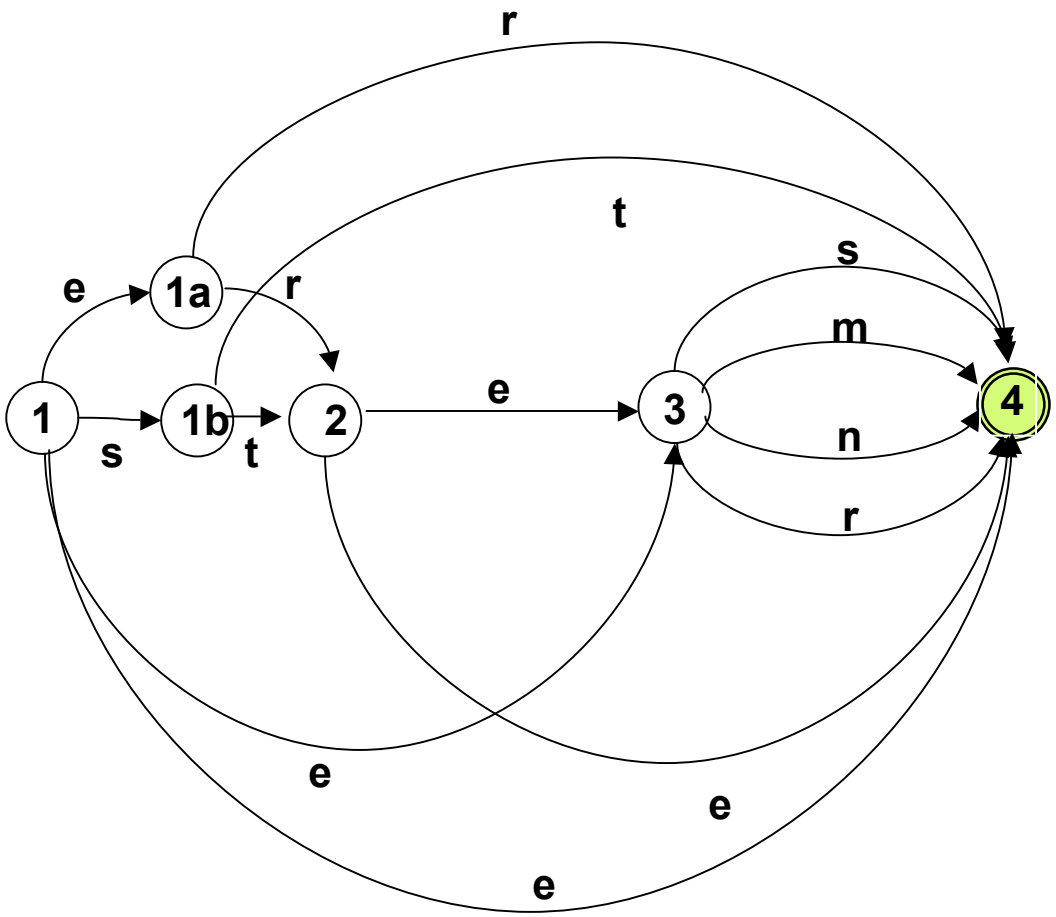
Pfadsuche als Breitensuche



Eingabewort: klein eres

Agenda:
1a -- klein eres
3 -- klein eres
4 -- klein eres

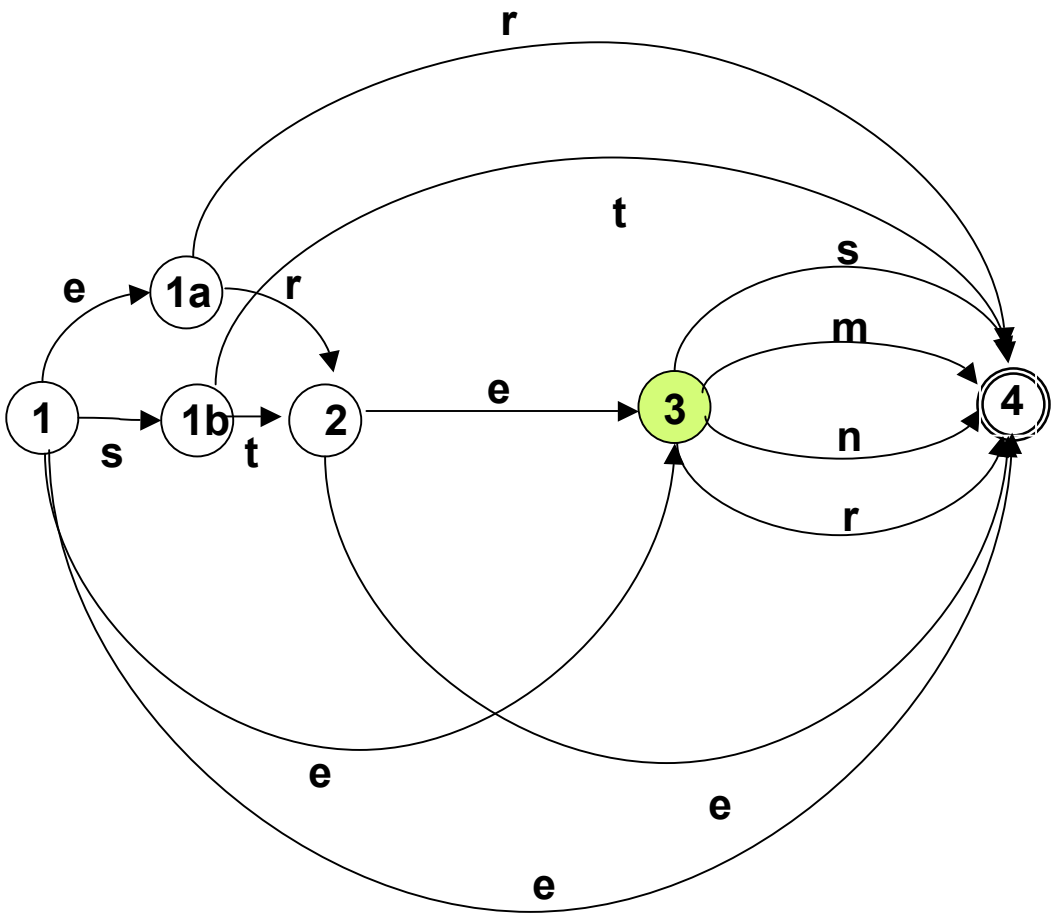
Pfadsuche als Breitensuche



Eingabewort: klein e<u>r</u>es

Agenda: 1a -- klein e<u>r</u>es
3 -- klein e<u>r</u>es

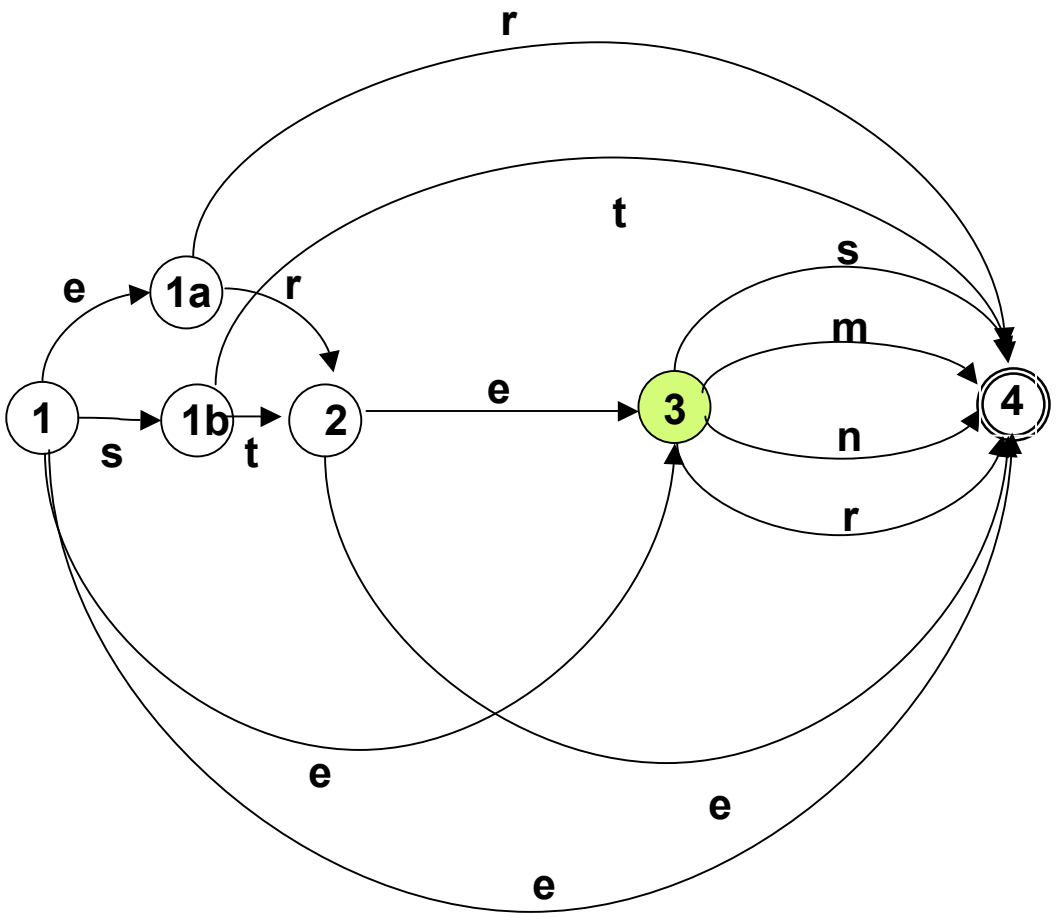
Pfadsuche als Breitensuche



Eingabewort: klein eres

Agenda: 1a -- klein eres

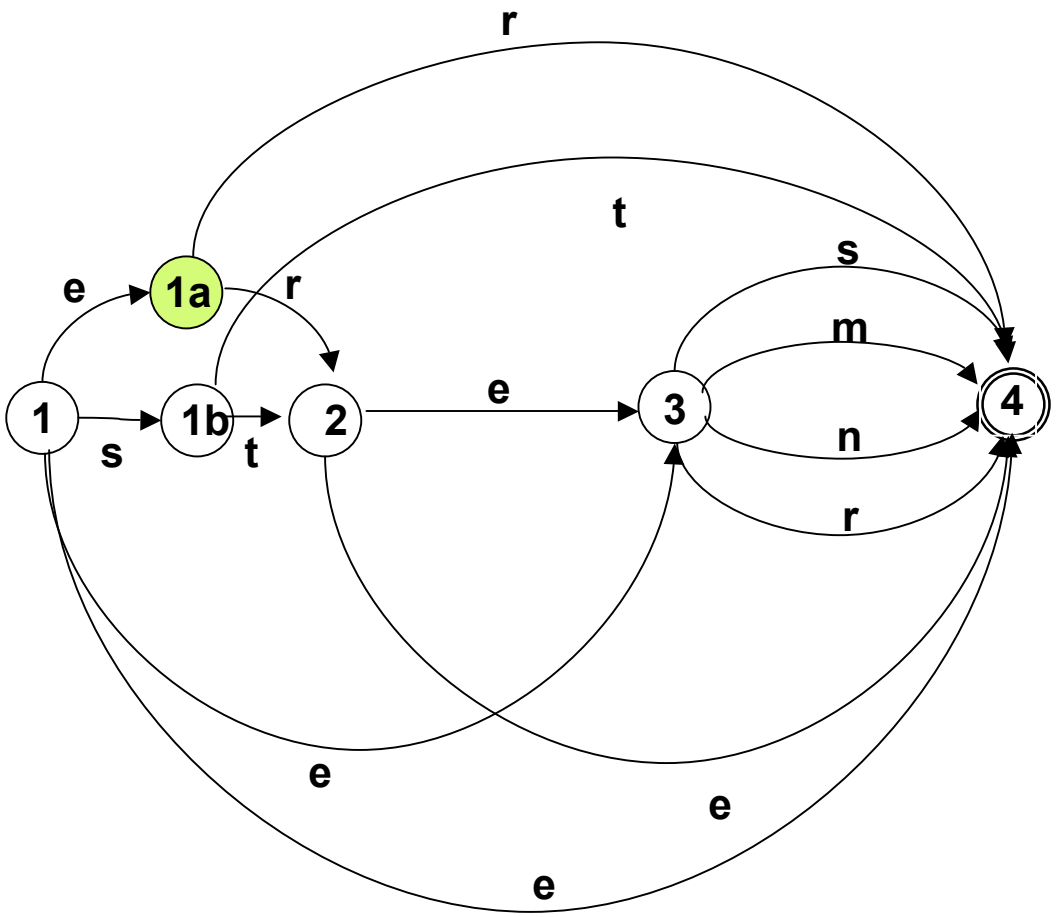
Pfadsuche als Breitensuche



Eingabewort: klein eres

Agenda: 4 -- klein eres
1a -- klein eres

Pfadsuche als Breitensuche



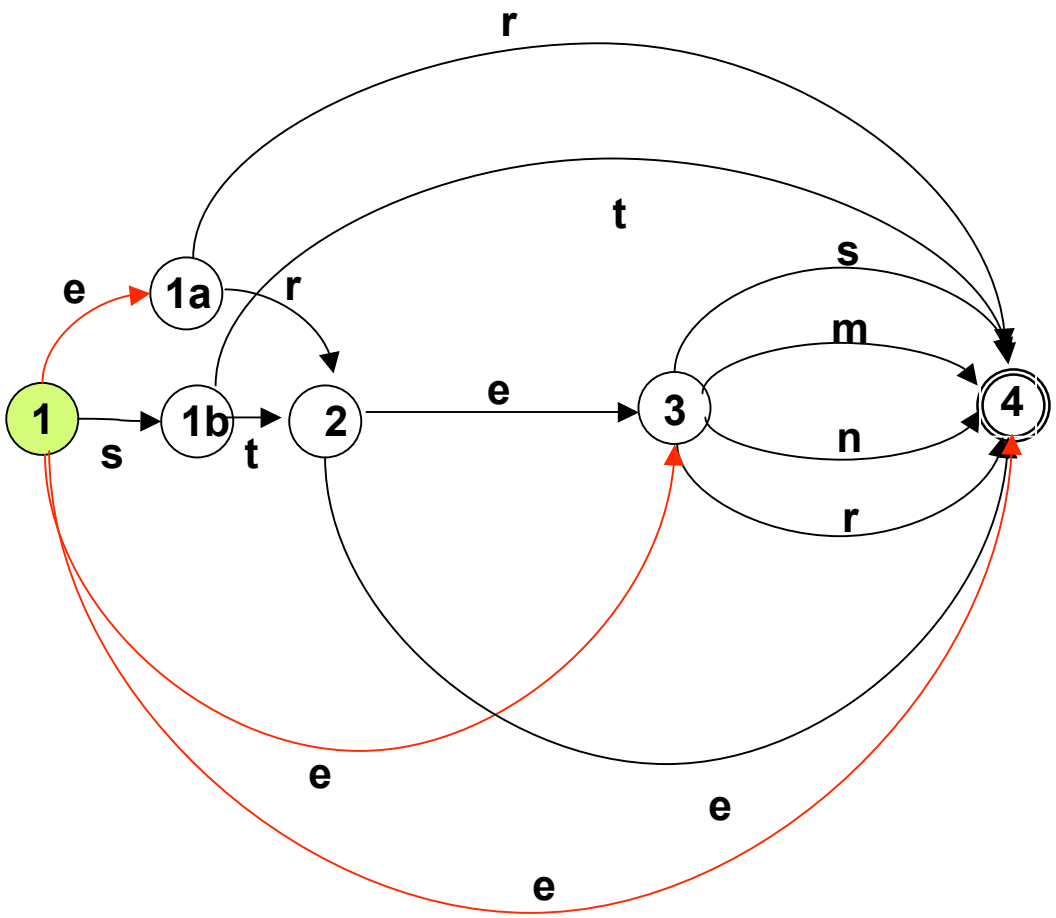
Eingabewort: klein e<u>r</u>es

Agenda:
2 -- klein er<u>e</u>s
4 -- klein er<u>e</u>s
4 -- klein er<u>e</u>s

Schritt 3: Potenzautomaten-Konstruktion, Vorüberlegung [2]

- Wir können „getaktete“ Breitensuche in einem buchstabierenden NEA so beschreiben:
  - Wir ermitteln alle Zustände, die durch die Abarbeitung des ersten Eingabesymbols vom Startzustand aus erreicht werden können.
  - Wir ermitteln alle Zustände, die durch die Abarbeitung des zweiten Eingabesymbols von einem Zustand dieser Zustandsmenge erreicht werden können, usf.
  - Wenn die Zustandsmenge, die wir auf diese Weise nach Abarbeiten des kompletten Wortes w enthalten, einen Endzustand des NEA enthält, wird w akzeptiert.

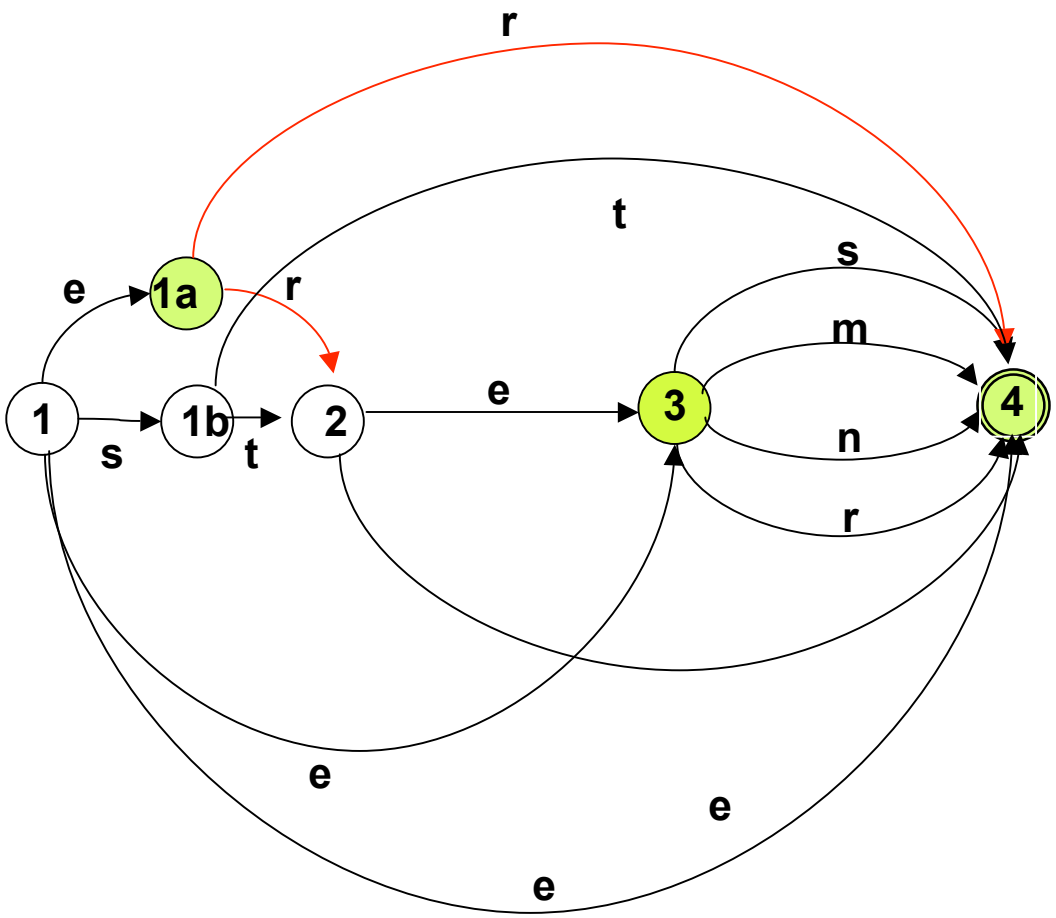
Pfadsuche als Breitensuche



Eingabewort: klein eres

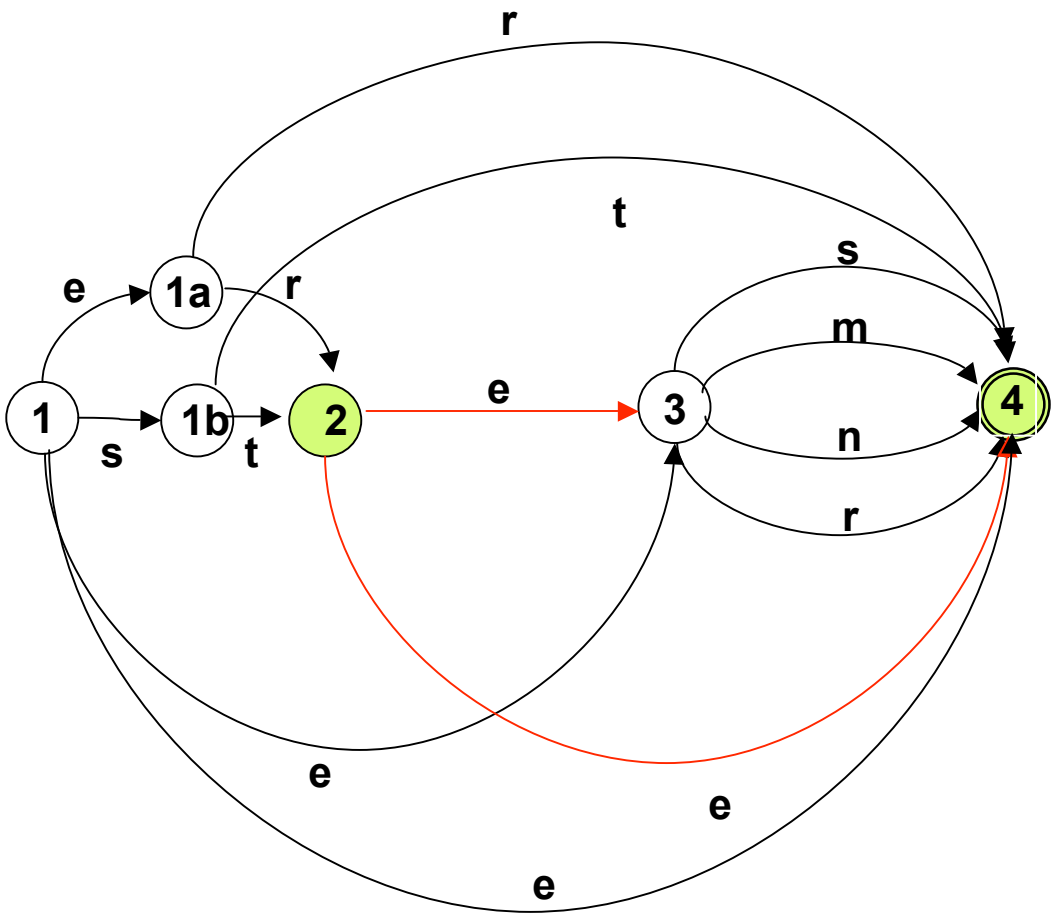
Agenda:
1a -- klein eres
3 -- klein eres
4 -- klein eres

Pfadsuche als Breitensuche



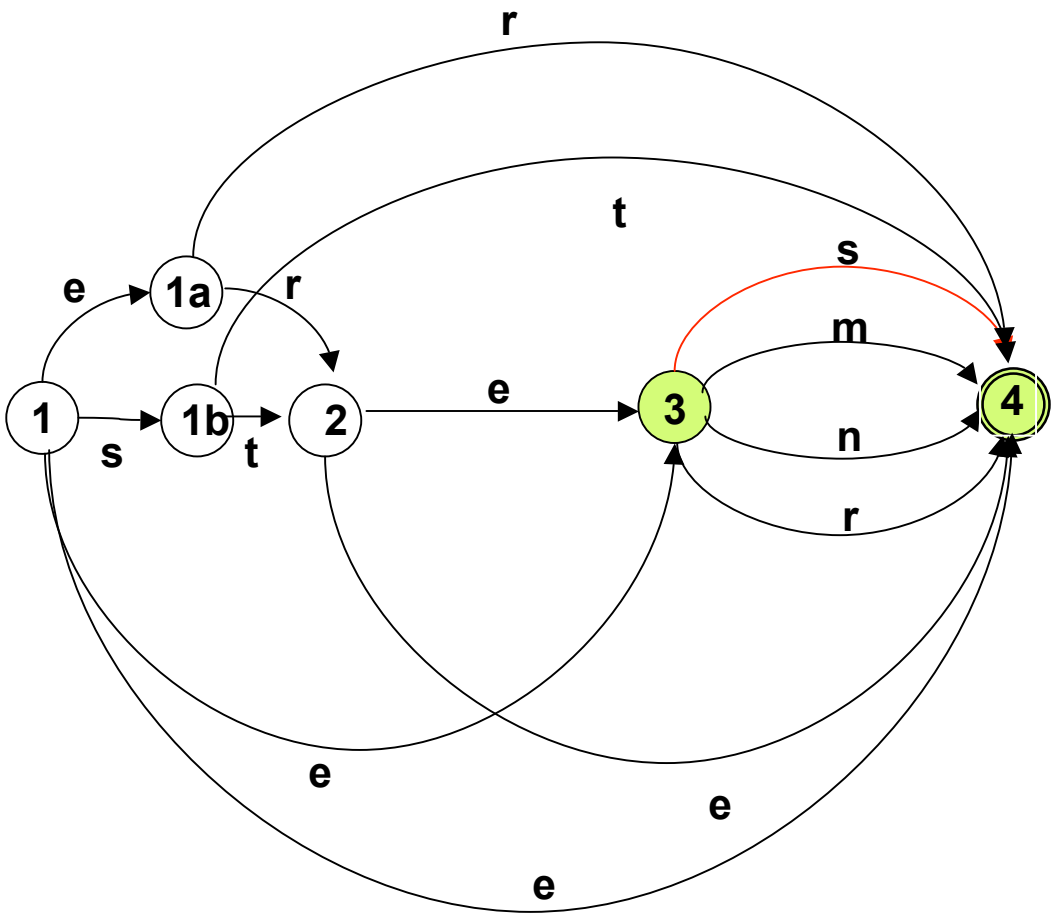
Eingabewort: klein e<u>r</u>es

Pfadsuche als Breitensuche



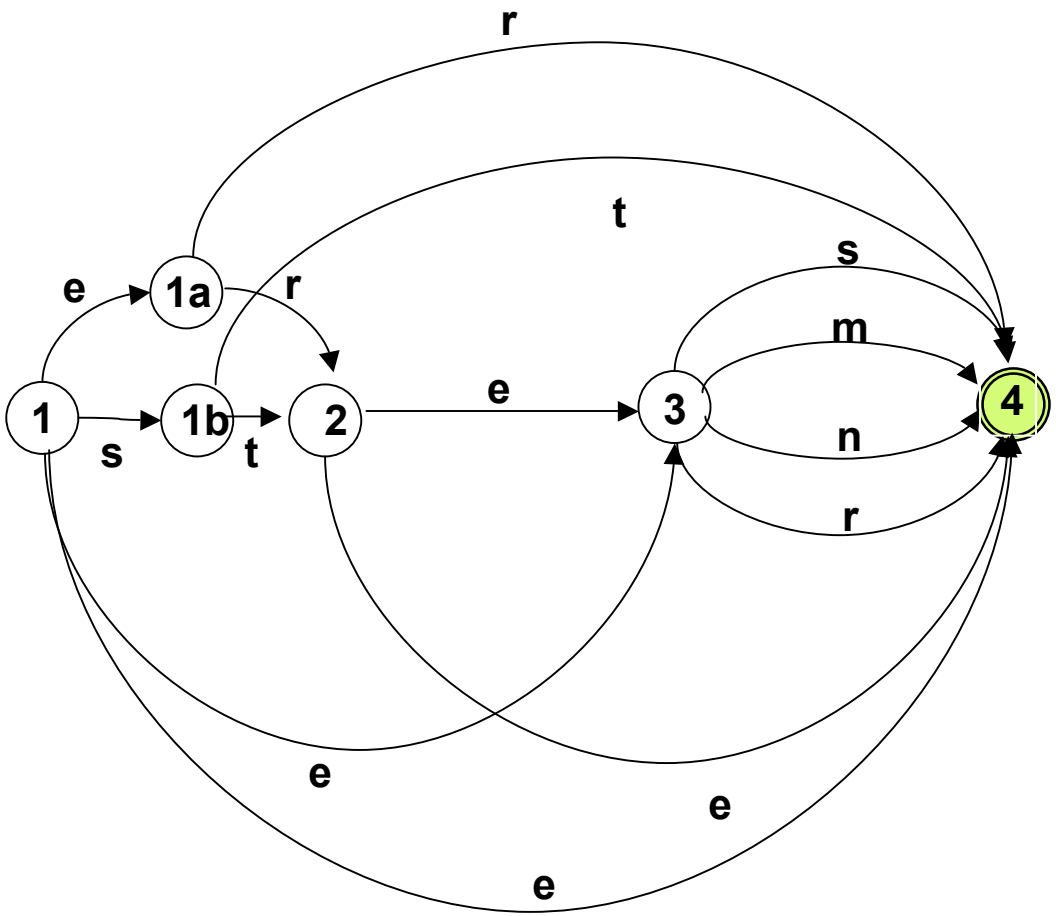
Eingabewort: klein eres

Pfadsuche als Breitensuche



Eingabewort: klein eres

Pfadsuche als Breitensuche



Eingabewort: klein eres_

Schritt 3: Potenzautomaten-Konstruktion, Vorüberlegung [3]

- Wir können diese "getaktete Suche" selbst mit einem endlichen Automaten beschreiben:
  - Zustände des neuen Automaten lassen sich als Mengen von Zuständen des NEA beschreiben. Am Beispiel: Nach Abarbeiten des ersten Symbols „e“ befindet er sich in dem Zustand, dass es die Zustandsmenge des NEA {1a, 2, 4} als mögliche aktuelle Zustände erkannt hat.
  - Wenn die Eingabekette abgearbeitet ist, und der Automat sich in einem Zustand befindet, der einen Endzustand des NEA enthält, ist die Eingabe akzeptiert.
  - Die „möglichen Zustände“ des NEA, die sich durch ein bestimmtes Eingabe-Symbol erreichen lassen, sind eindeutig definiert. Der neue Automat ist also ein DEA.

Schritt 3: Potenzautomaten-Konstruktion: Die Definition

Der Potenzautomat zum buchstabierenden NEA

A = <K, Σ, Δ, s, F> ist der DEA A' = <K', Σ, δ, s', F'>

mit:

- K' = $\wp$ (K) (die Potenzmenge der Zustandsmenge des NEA)
- s' = {s}
- δ(p', a) = {q| es gibt p$\in$p' und <p,a,q>$\in$D} für jedes p'$\subseteq$K, a$\in \Sigma$
- q' $\in$ F' gdw. q' $\cap$ F $\neq \varnothing$

Praktisches Vorgehen

Der Potenzautomat A' zu A = <K, Σ, Δ, s, F> hat $2^{|\Delta|}$ Zustände. In der Regel sind viele dieser Zustände unerreichbar (vom Startzustand {s} aus) und deshalb funktionslos.

Praktisches Konstruktionsverfahren:

Beginne mit {s}, berechne die Übergangsfunktion für {s}, für alle direkt von s erreichbaren Zustände usw., bis keine neuen erreichbaren Zustände hinzukommen.

Beispiel: DEA für Adjektiv-Endungen

- Grundlage: der buchstabierende Automat

  $A = \langle \{1, 1a, 1b, 2, 3, 4\}, \{e, m, n, r, s, t\}, \Delta, 1, \{1, 4\} \rangle$,

  Δ wie im Diagramm Folie 42

- Potenzautomat ist $A' = \langle K', \Sigma, \delta, s', F' \rangle$

  mit $K' = \wp(K)$

  $s' = \{s\}$

  $F' = \{q' \in K' \mid 1 \in q' \text{ oder } 4 \in q'\}$

  δ s. Übergangstabelle nächste Folie

DEA für Adjektiv-Endungen, Übergangstabelle

δ:	e	m	n	r	s	t
{1}	{1a,3,4}	∅	∅	∅	{1b}	∅
{1a,3,4}	∅	{4}	{4}	{2,4}	{4}	∅
{1b}	∅	∅	∅	∅	∅	{2,4}
{2,4}	{3,4}	∅	∅	∅	∅	∅
{3,4}	∅	{4}	{4}	{4}	{4}	∅
{4}	∅	∅	∅	∅	∅	∅
∅	∅	∅	∅	∅	∅	∅

Das Diagramm