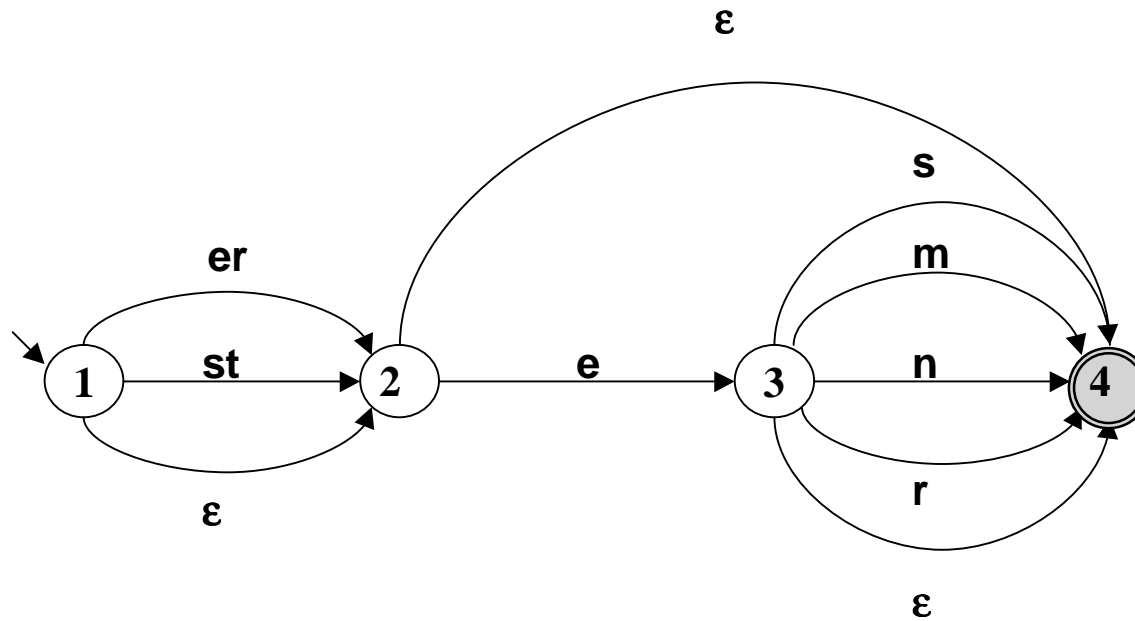
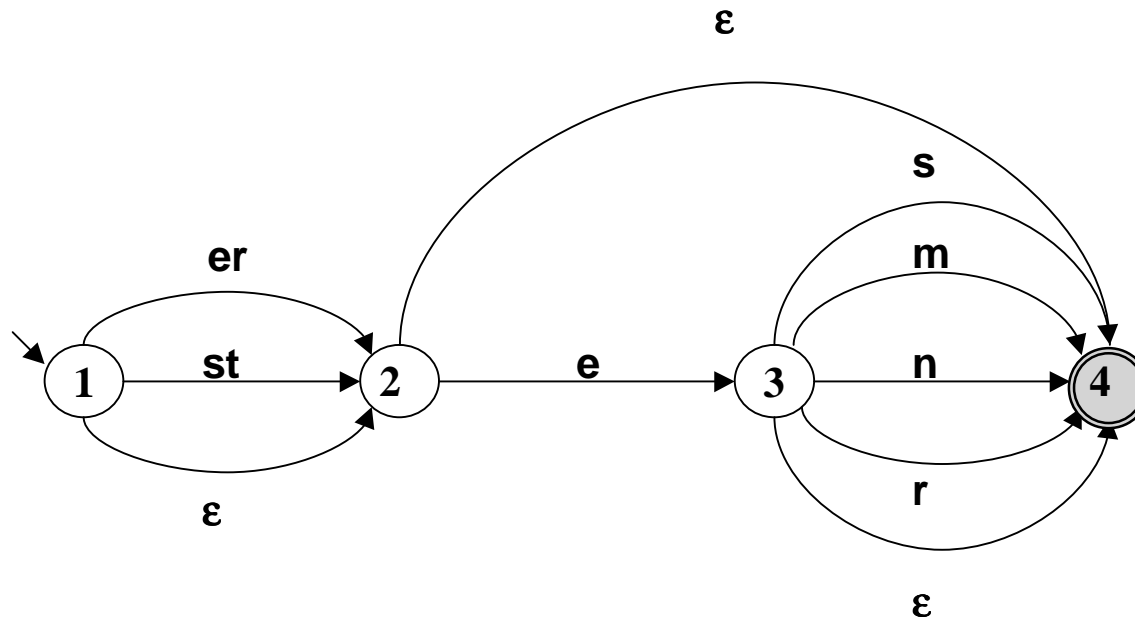


# Ein Weg durchs Diagramm



klein eres\_

## Ein alternativer Weg durchs Diagramm



klein eres

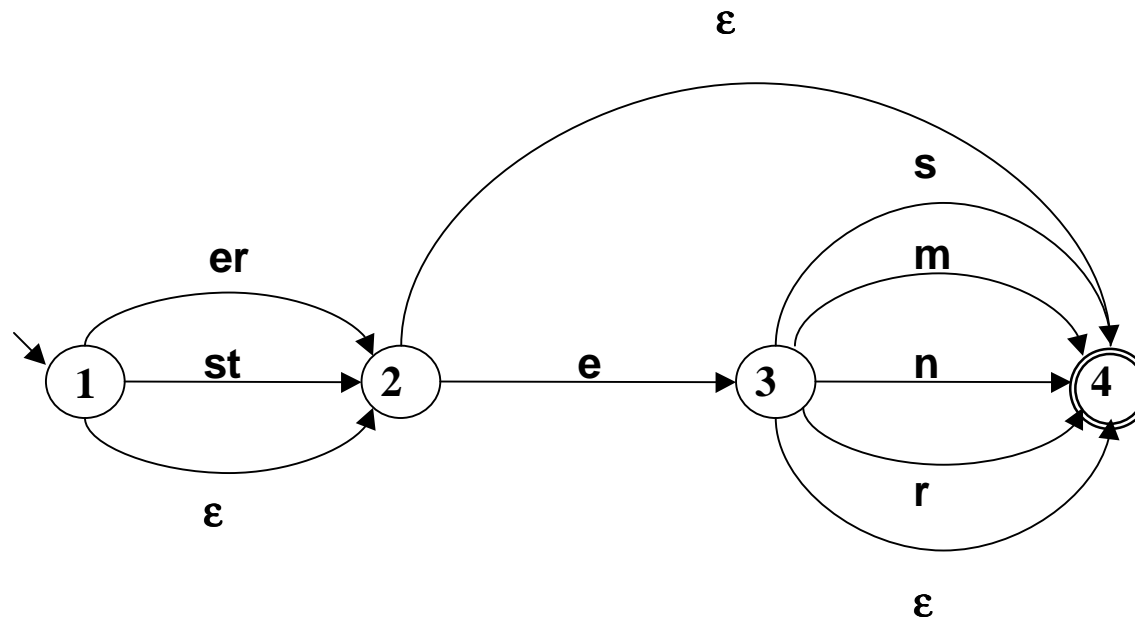
# Das Problem

- 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 aus der Agenda, den letzten oder obersten, wenn wir uns die Agenda als Stapel („Stack“) vorstellen, betrachten ihn als neue aktuelle Konfiguration, testen mögliche nächste Schritte, legen die Resultate auf die Agenda usw. – bis wir die Eingabe erfolgreich abgearbeitet haben (akzeptieren!) oder die Agenda leer ist (zurückweisen!).

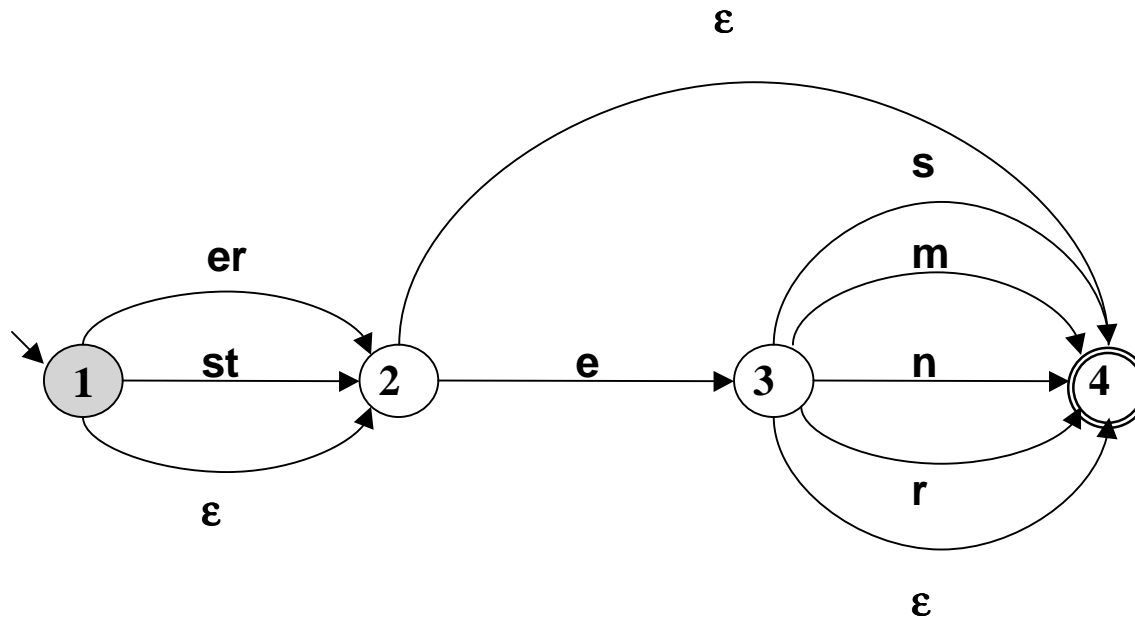
# Pfadsuche: Startkonfiguration: Startzustand und Eingabewort auf der Agenda



Eingabewort:

Agenda: 1 -- klein eres

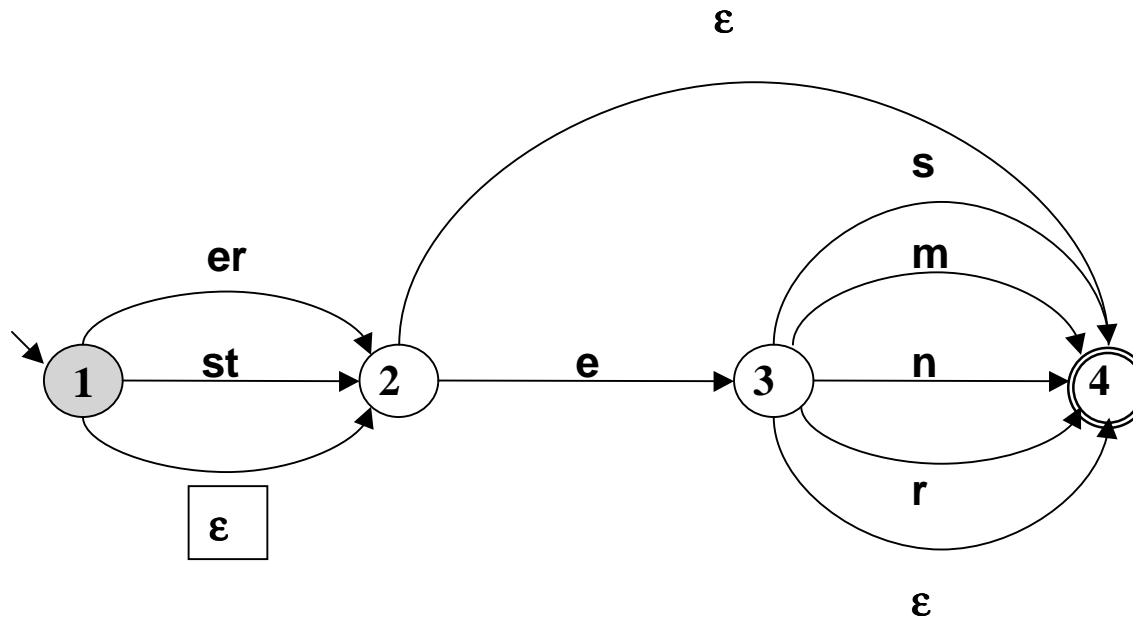
# Pfadsuche: Nimm Aufgabe von der Agenda



Eingabewort: klein eres

Agenda: \_\_\_\_\_

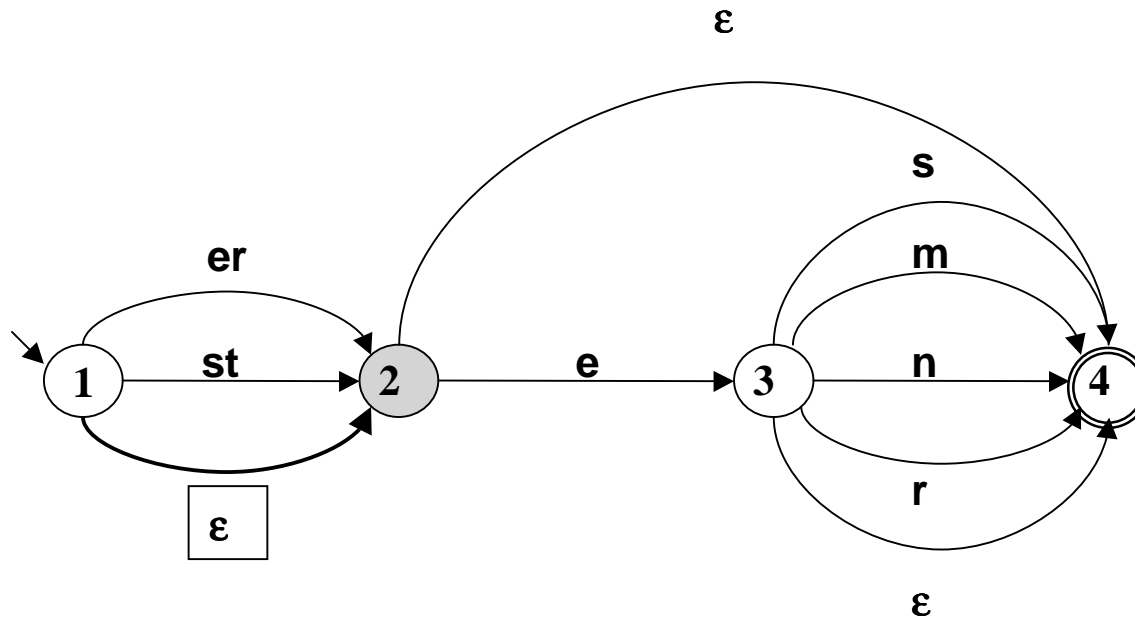
# Pfadsuche: Generiere neue Aufgaben



Eingabewort: klein eres

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

# Pfadsuche: Nimm Aufgabe von der Agenda

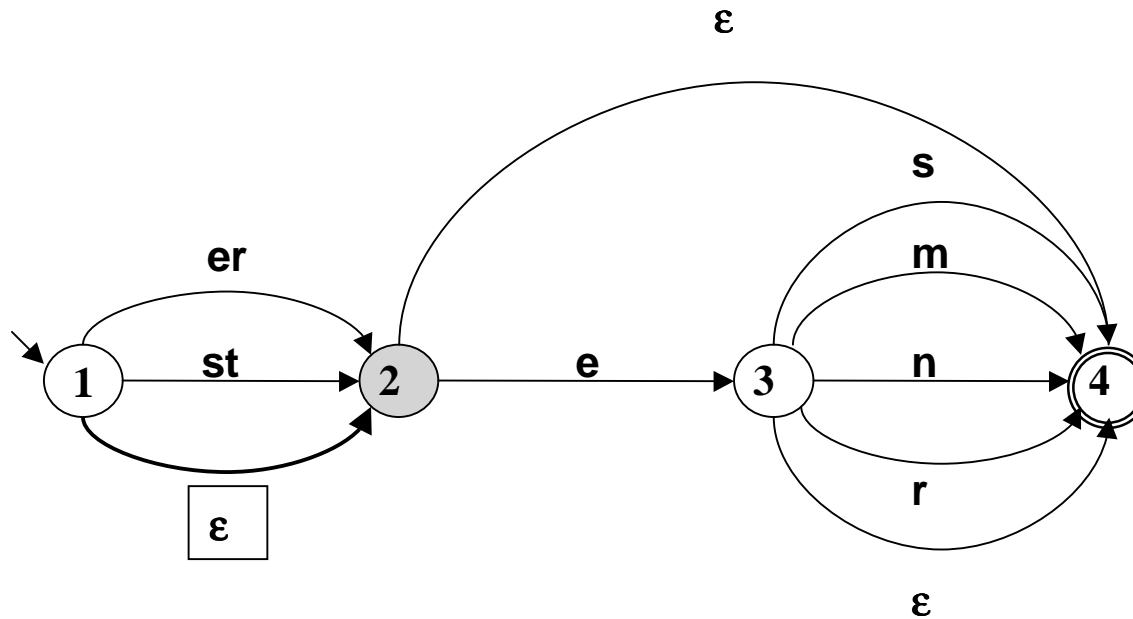


Eingabewort: klein eres

Agenda: 2 -- klein eres



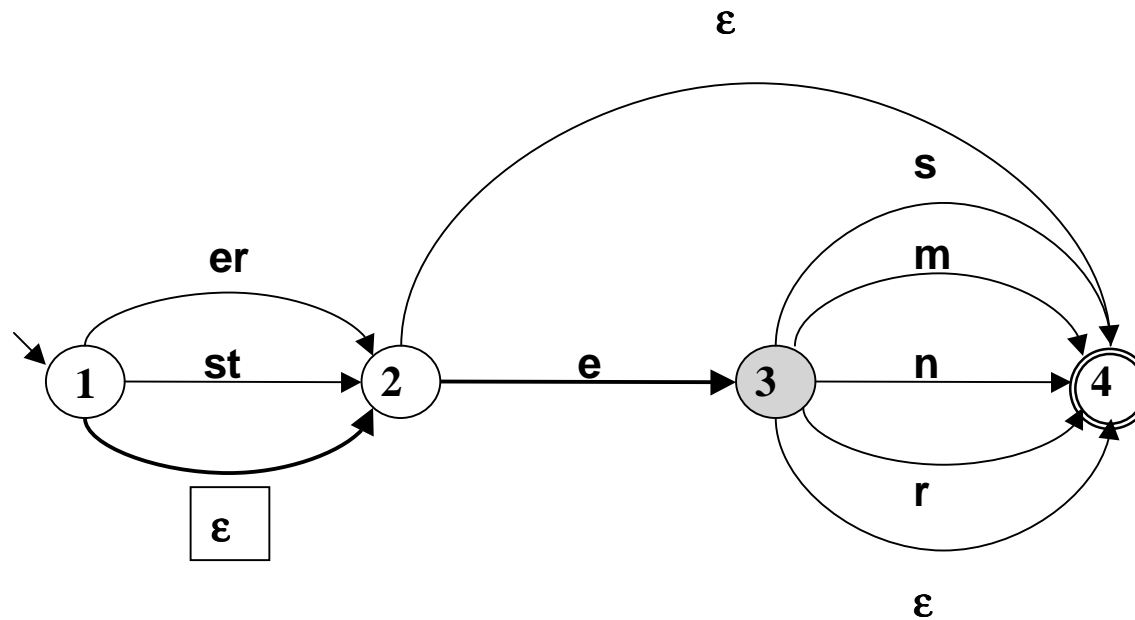
# Pfadsuche: Generiere neue Aufgaben



Eingabewort: klein eres

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

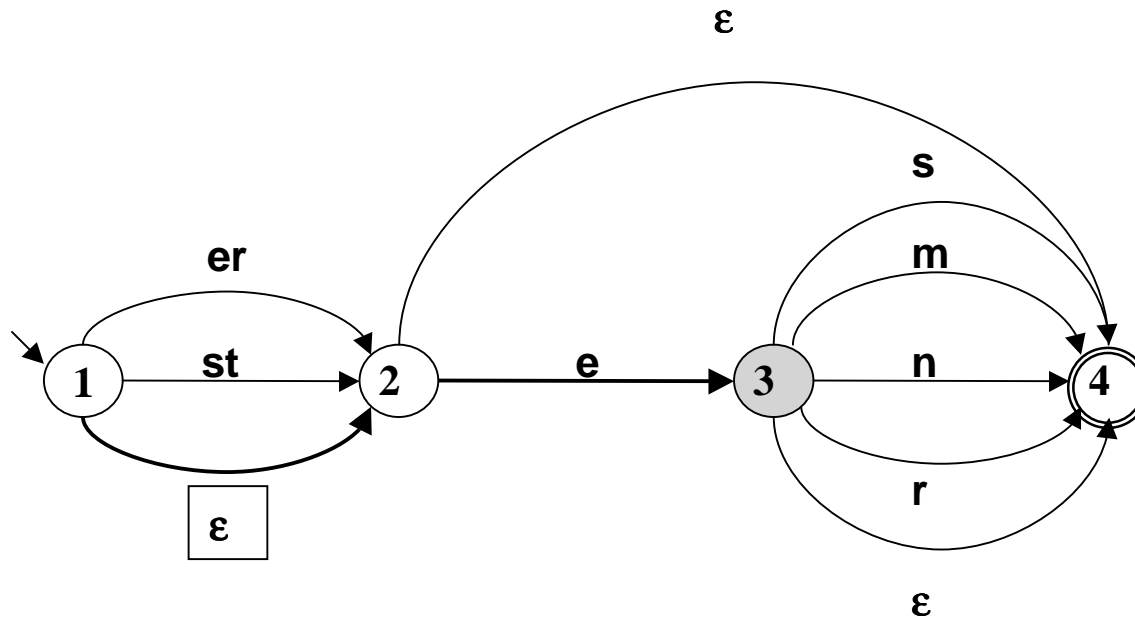
# Pfadsuche: Nimm Aufgabe von der Agenda



Eingabewort: klein eres

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

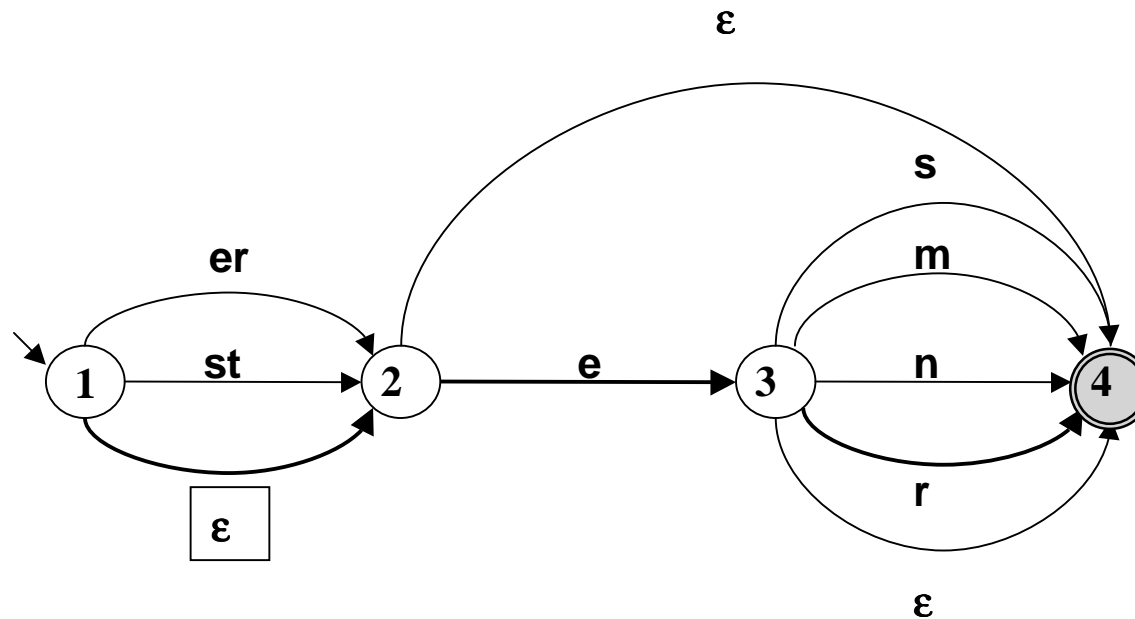
# Pfadsuche: Generiere neue Aufgaben



Eingabewort: klein eres

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

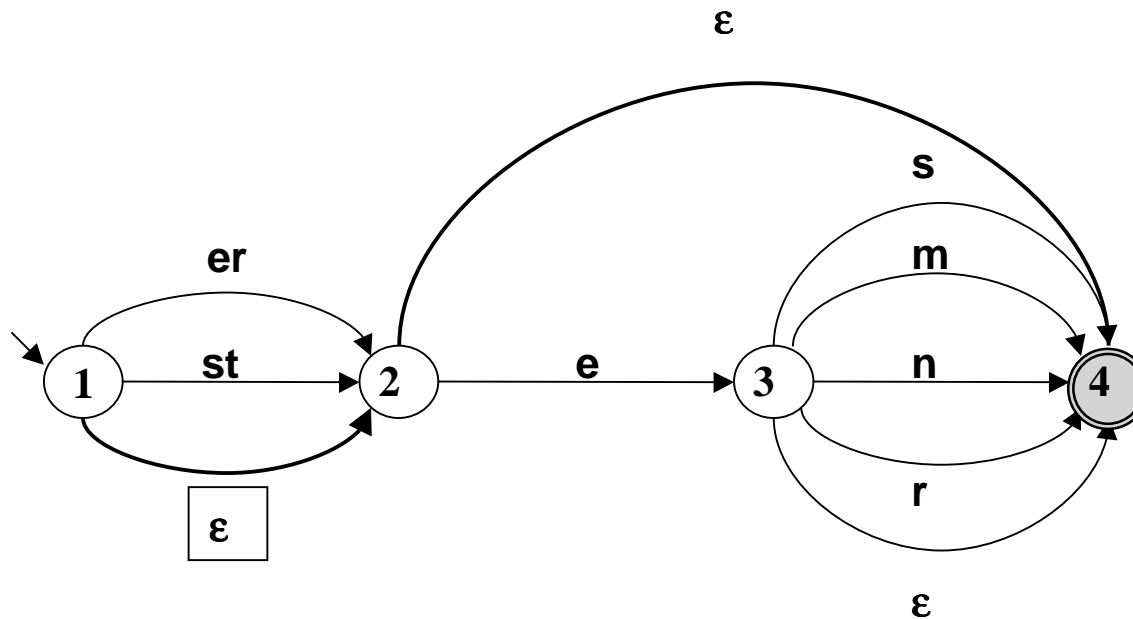
Pfadsuche: Nimm Aufgabe von der Agenda  
Keine neue Aufgabe!



Eingabewort: klein eres

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

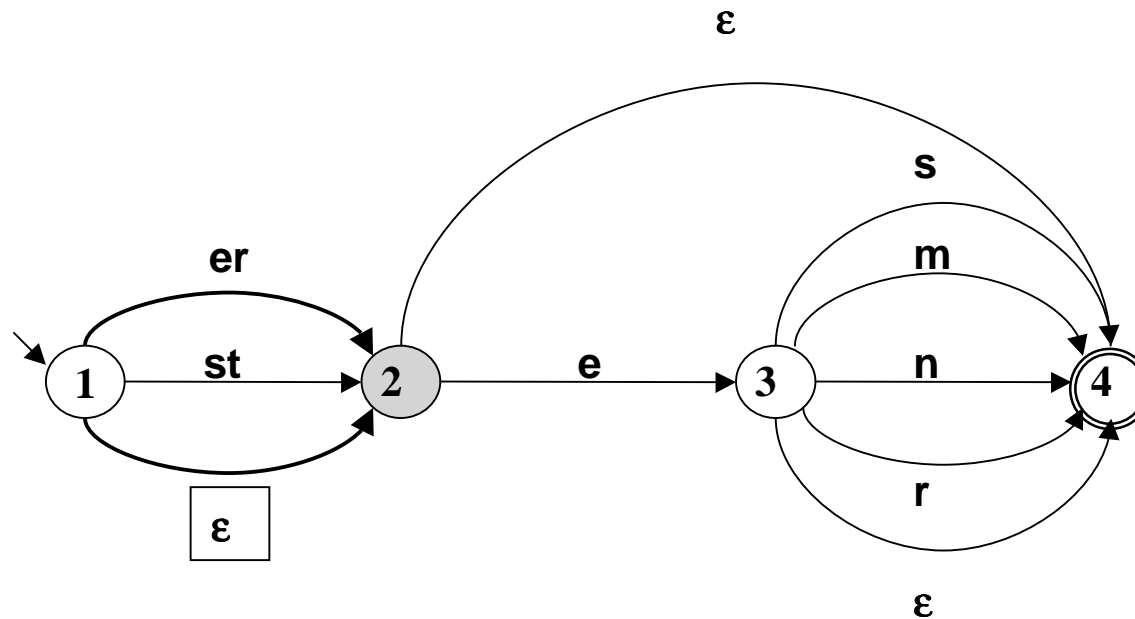
Pfadsuche: Nimm Aufgabe von der Agenda  
Keine neue Aufgabe!



Eingabewort: klein eres

Agenda: 2 -- klein eres

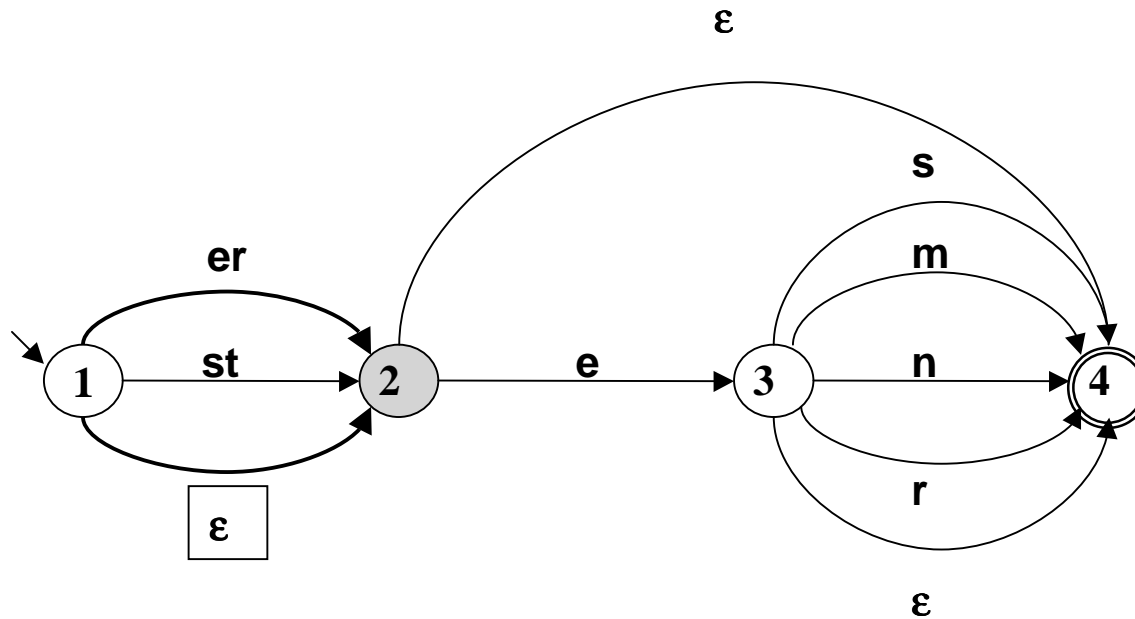
# Pfadsuche: Nimm Aufgabe von der Agenda Backtracking!



Eingabewort: klein eres

Agenda: \_\_\_\_\_

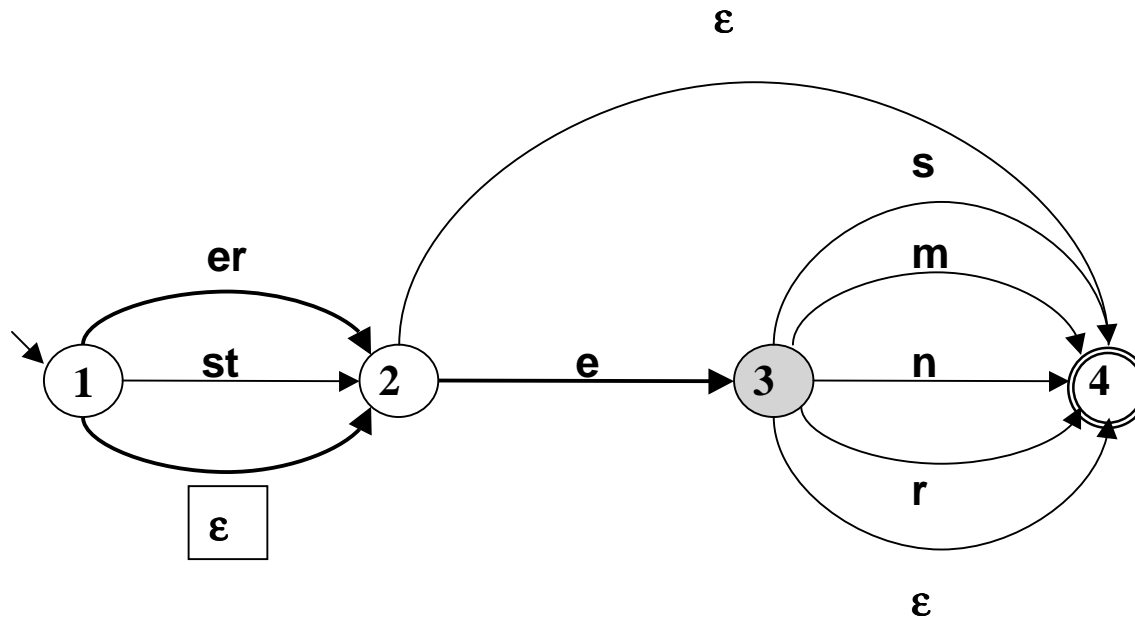
# Pfadsuche: Generiere Aufgaben



Eingabewort: klein eres

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

# Pfadsuche: Nimm Aufgabe von der Agenda

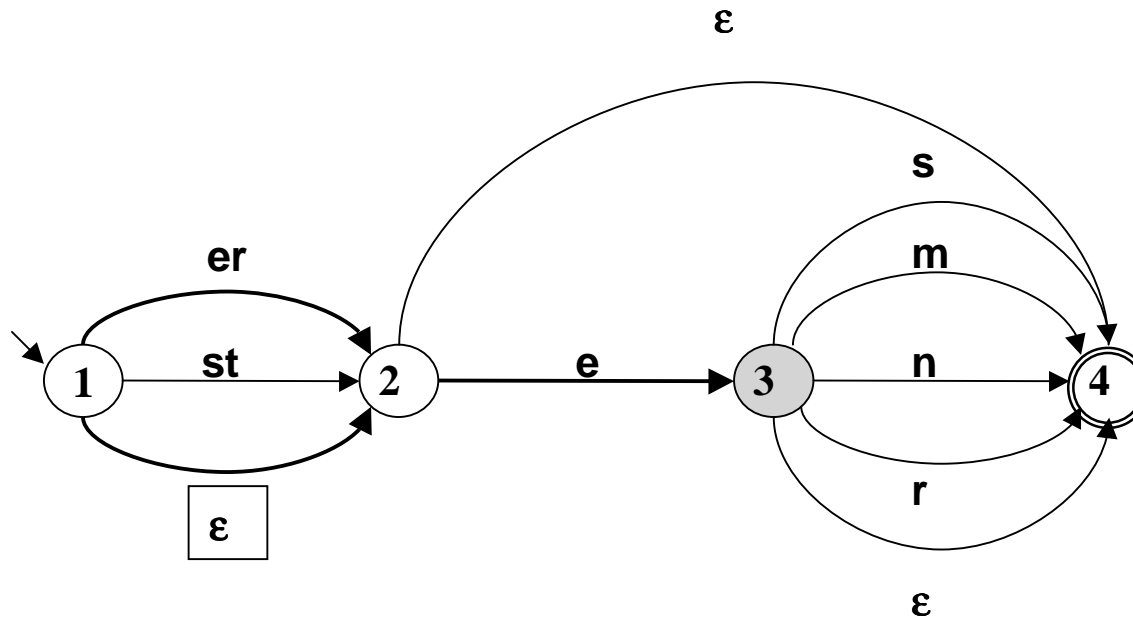


Eingabewort: klein eresu

Agenda: 4 -- klein eresu



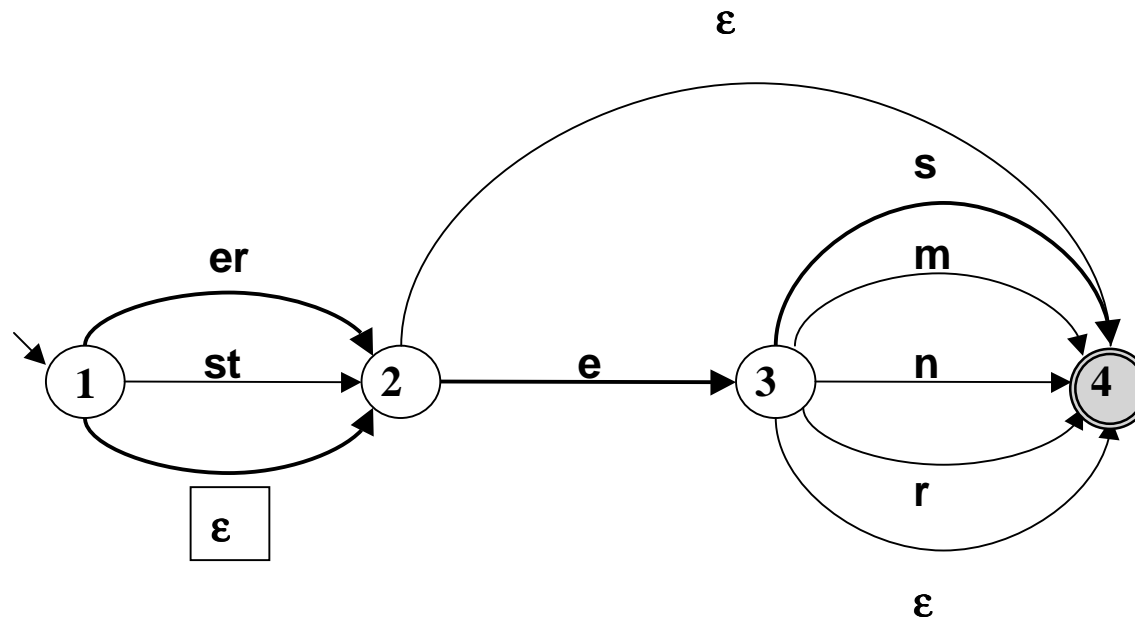
# Pfadsuche: Generiere Aufgabe



Eingabewort: klein eresε

Agenda: 4 -- klein eresε

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



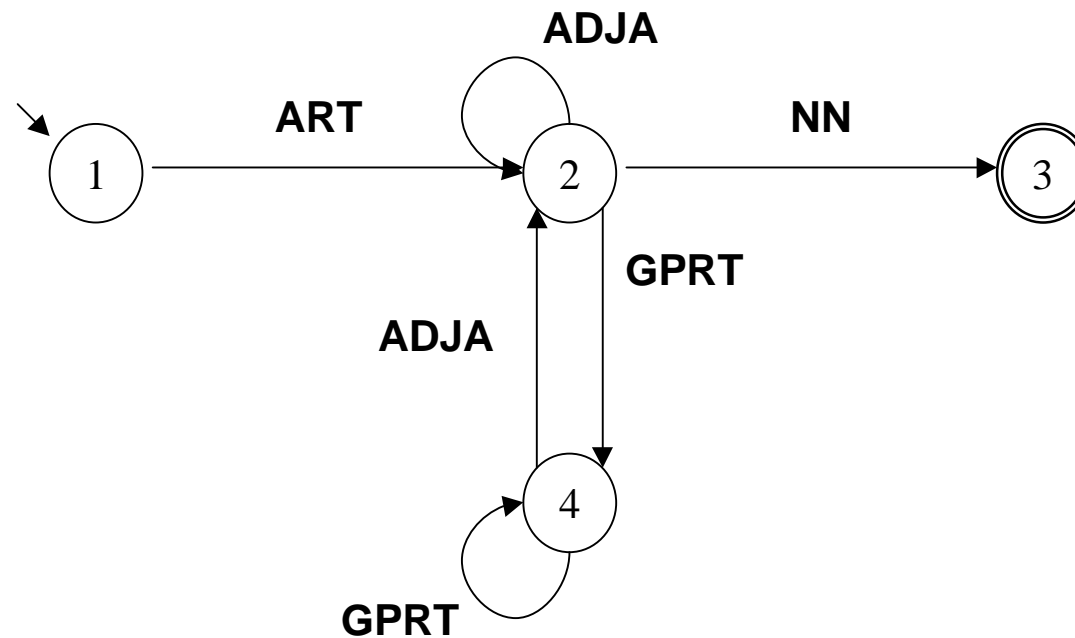
Eingabewort: klein eres\_

Agenda: 4 -- klein eres

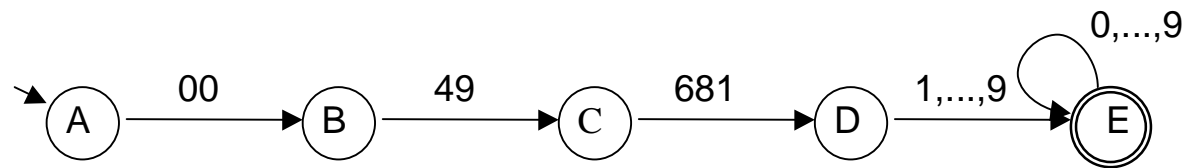
## Anmerkung zum Pfadsuche-Algorithmus

- Der vorgestellte Pfadsuche-Algorithmus („Tiefensuche mit Backtracking“) ist vollständig – wenn das Diagramm/der Automat keine Leerwort-Zyklen enthält (Schleifen, bei deren Durchlaufen das leere Wort abgearbeitet wird).
- Der Pfadsuche-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, indem man andere Algorithmen verwendet, den Algorithmus optimiert, oder das Diagramm geschickt schreibt („teure“ Strukturen vermeiden). Wir werden eine grundsätzlich andere Möglichkeit betrachten.

# 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 beschriftet 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.  $\epsilon$ ) 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)
- $\Sigma$  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  $A = \langle K, \Sigma, \delta, s, F \rangle$  mit

- $K = \{1, 2, 3, 4\}$
- $\Sigma = \{\text{ART}, \text{ADJA}, \text{NN}, \text{GPRT}\}$
- $s = 1$
- $F = \{3\}$
- $\delta$  definiert durch:
  - $\delta(1, \text{ART}) = 2$
  - $\delta(2, \text{ADJA}) = 2$
  - $\delta(2, \text{NN}) = 3$
  - $\delta(2, \text{GPRT}) = 4$
  - ...      ...

## Beispiel [2]: Übergangstabelle für $\delta$

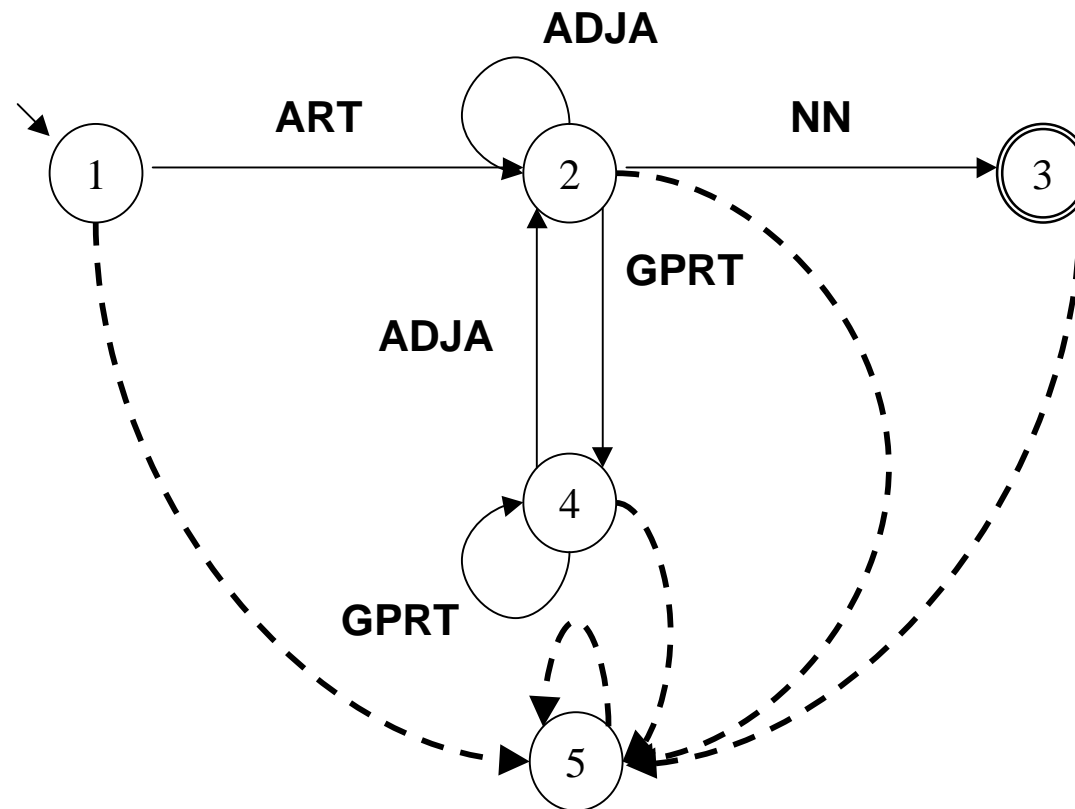
$\delta$ :	ART	ADJA	NN	GPRT
1	2			
2		2	3	4
3				
4		2		4

## Beispiel [3]: Übergangstabelle für $\delta$ , komplettiert

$\delta$ :	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 $\delta$ 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]

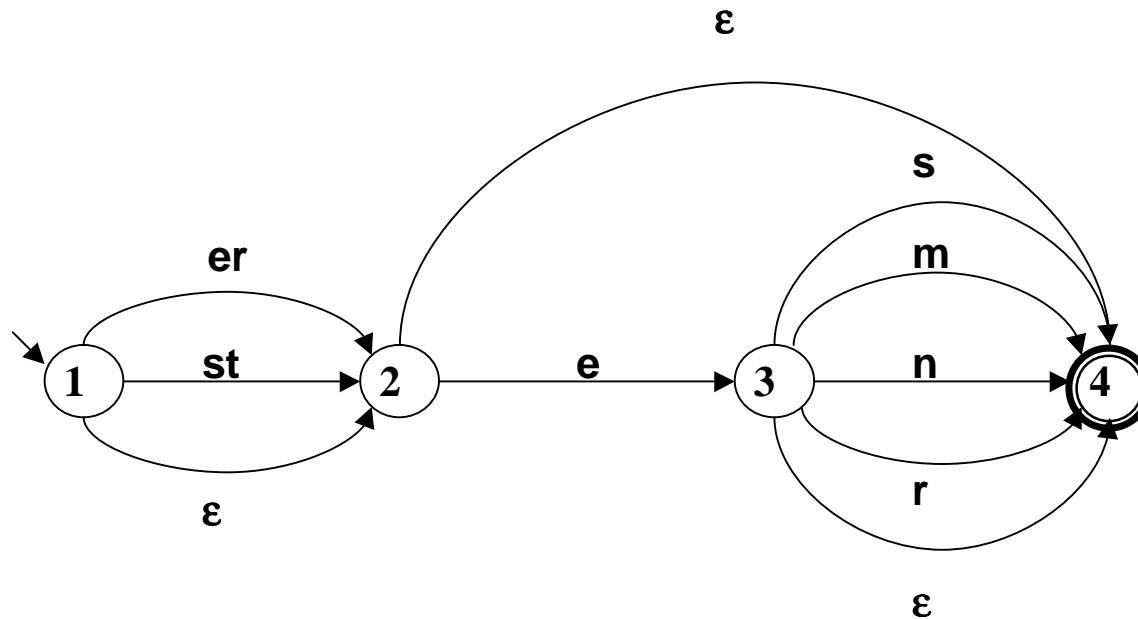
- Jede Sprache, die von einem NEA akzeptiert wird, kann auch durch einen DEA beschrieben werden (und, trivialerweise, auch umgekehrt: ein DEA ist ein spezieller NEA).
- Es gibt einen konstruktiven Beweis, d.h., ein Verfahren, das es erlaubt, zu jedem NEA  $A$  einen DEA  $A'$  zu konstruieren, so dass  $L(A') = L(A)$ .

# Die NEA-DEA-Überführung

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

1. Beseitigung von Mehrsymbol-Kanten
2. Beseitigung von  $\varepsilon$ -Kanten
3. Die „Potenz-Automaten“-Konstruktion

# Adjektivendungen: Zustandsdiagramm



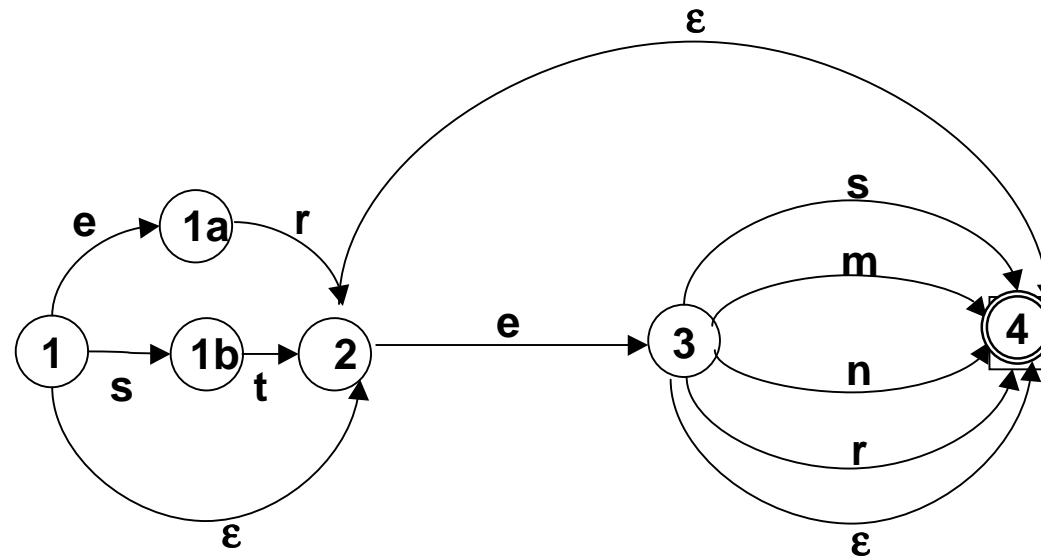


## Schritt 1: Beseitigung von Mehrsymbolkanten

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

- Für alle Kanten  $\langle q, w, q' \rangle$  mit  $w = a_1 \dots a_n$ ,  $n > 1$ :  
Entferne  $\langle q, w, q' \rangle$  aus  $\Delta$ .
- Erweitere  $K$  um neue Zustände  $q_1, \dots, q_{n-1}$ .
- Erweitere  $\Delta$  um neue Kanten  
 $\langle q, a_1, q_1 \rangle, \langle q_1, a_2, q_2 \rangle, \dots, \langle q_{n-1}, a_n, q' \rangle$

## Beispiel-Automat nach Schritt 1:



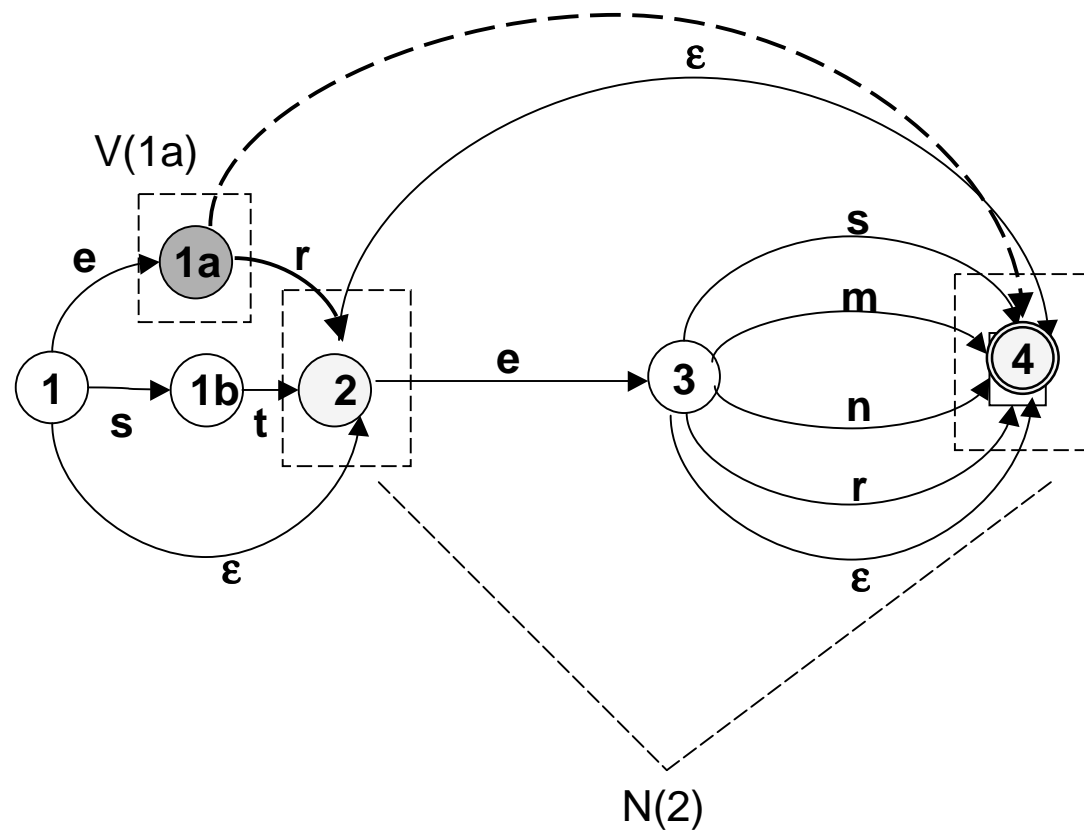
## Schritt 2: Beseitigung von $\varepsilon$ -kanten

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

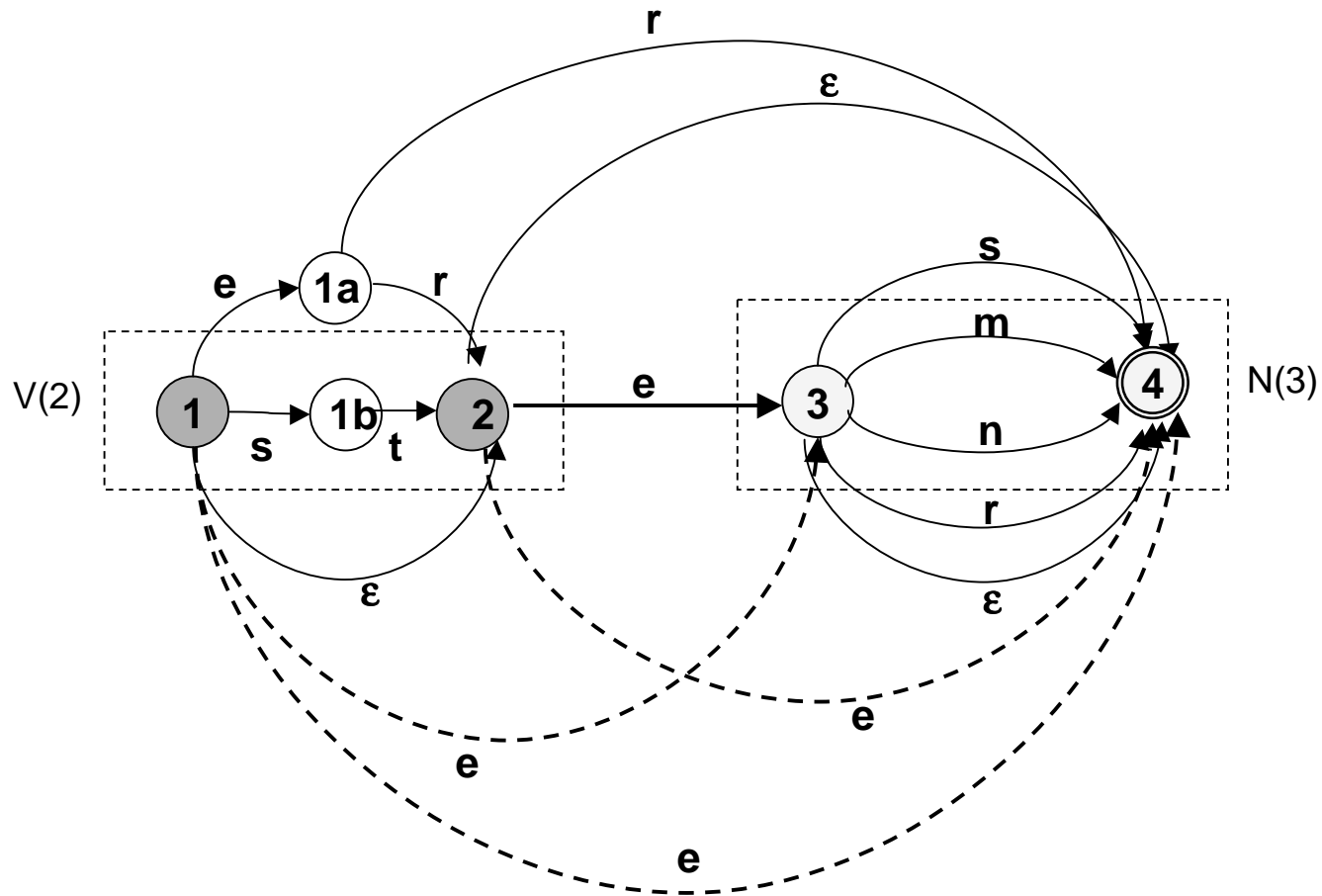
Anmerkung:  $V_\varepsilon(p)$  und  $N_\varepsilon(p)$  enthalten insbesondere  $p$  selbst.

- Für jede nicht-leere Kante  $\langle p, a, q \rangle \in \Delta$ : Erweitere  $\Delta$  um alle  $\langle p', a, q' \rangle$  mit  $p' \in V_\varepsilon(p)$ ,  $q' \in N_\varepsilon(q)$ .
- Entferne alle leeren Kanten aus  $\Delta$ .
- Wenn sich ein Endzustand im  $\varepsilon$ -Nachbereich des Startzustandes  $s$  befindet, füge  $s$  zu den Endzuständen hinzu.

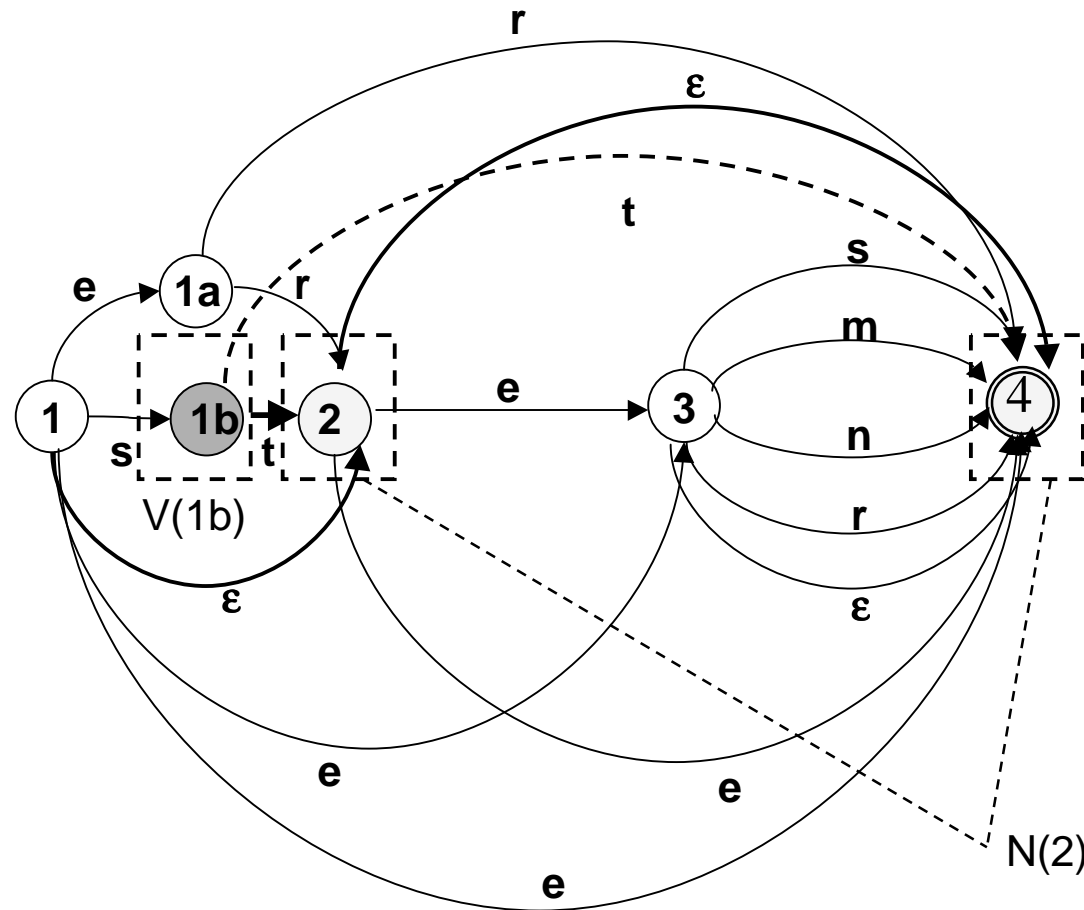
## Schritt 2: Beseitigung von $\varepsilon$ -Kanten



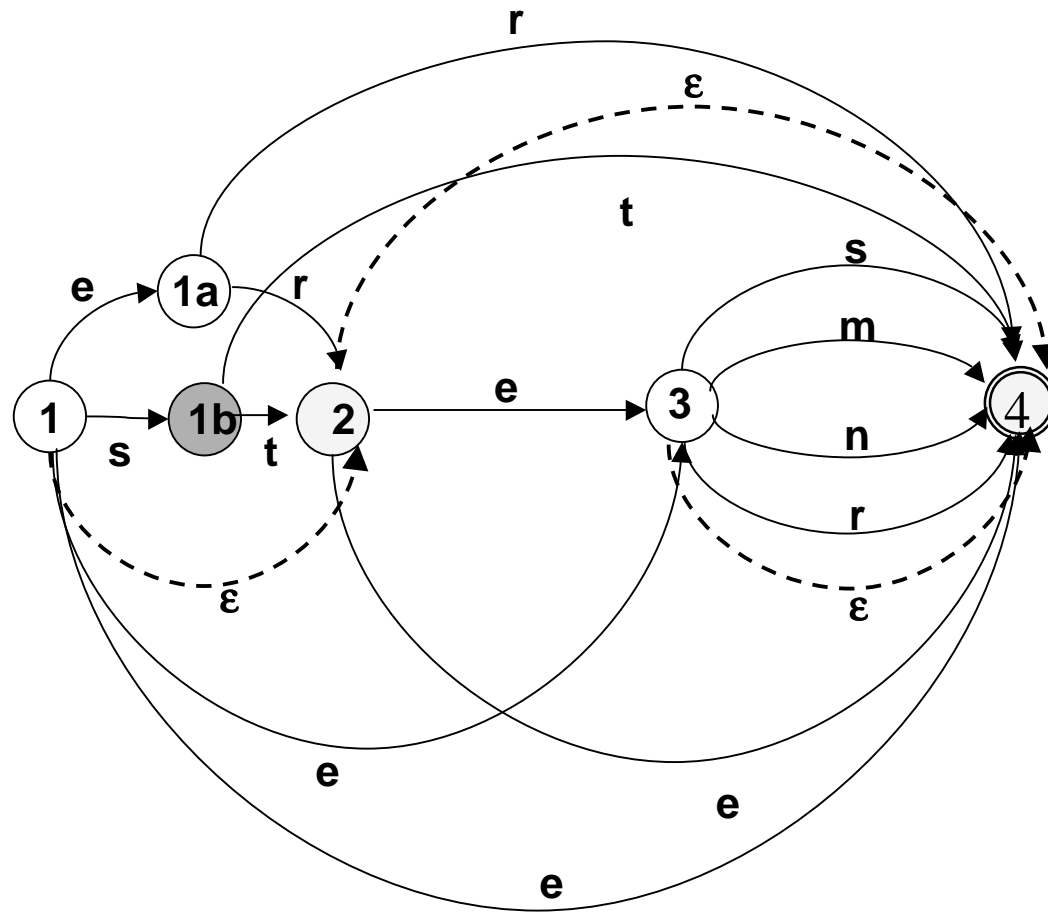
## Schritt 2: Beseitigung von $\varepsilon$ -kanten



## Schritt 2: Beseitigung von $\varepsilon$ -kanten



## Schritt 2: Beseitigung von $\epsilon$ -kanten



## Schritt 2: Beseitigung von $\epsilon$ -kanten: Resultat ist „buchstabierender Automat“

