Foundations of Language Science and Technology

Finite State Methods for Lexicon and Morphology

Bernd Kiefer

`Bernd.Kiefer@dfki.de`

Deutsches Forschungszentrum für künstliche Intelligenz

Morphological Parsing

- Break a surface form into morphemes:
  - foxes into fox (noun stem) and -e -s (plural suffix + e-insertion)
- Compute stem and features
  - goose → goose `+N +SG` or `+V`
  - geese → goose `+N +PL`
  - gooses → goose `+V +3SG`
- Needed for (among others)
  - spell-checking: is steadyly or steadily correct?
  - identify a word's part-of-speech
  - reduce a word to its stem

Morphological Knowledge

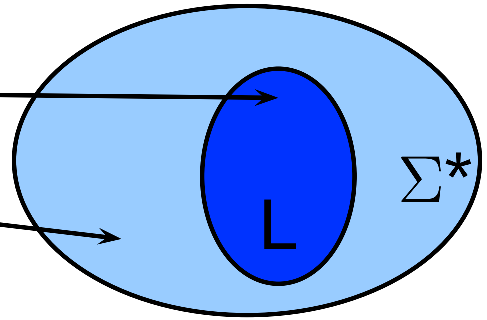
Components needed in a morphological parser:

1. **Lexicon:** list of stems and class information (base, inflectional class etc.)
2. **Morphotactics:** a model of morphological processes like English adjective inflection on the last slide
   - lexical and morphotactic knowlegde will be encoded using *finite-state automata*
3. **Orthography:** a model of how the spelling changes when morphemes combine, e.g.,
   - city+s $\rightarrow$ *cities*
   - in $\rightarrow$ il in context of l, like in- +legal
   - will be modeled using *finite-state transducers*

Detour: Describing Languages

- Language: a set of finite sequences of symbols
- Symbols can be anything like graphemes, phonemes, etc.
- Alphabet: the inventory of symbols
- We want formal devices to describe the strings in a language

- Alphabet Σ (Sigma): a nonempty finite set of symbols
- Strings of a language: arbitrary finite sequences of symbols in Σ
  - ➤ ϵ (epsilon) denotes the empty string
  - ➤ Σ^* is the set of all strings over Σ, including ϵ
- A language L is a subset of Σ^*, $L \subseteq \Sigma^*$
  - ➤ grammatical sentences $w \in L$
  - ➤ ungrammatical sentences $v \notin L$



Formal Grammars - Definitions

- Mathematical devices to describe languages
- Goal: separate the grammatical from the ungrammatical strings
- One of the devices: rule systems
  - Two alphabets: terminals Σ, nonterminals N
  - Rules rewrite strings in ($\Sigma \cup$ N)* into new strings in ($\Sigma \cup$ N)*
- Languages differ in complexity
- Complexity depends on the type of rule system / device needed

Chomsky Hierarchy

- Type 3: regular languages

  Rules of type A $\rightarrow \alpha$, A $\rightarrow \alpha$ B; A,B $\in$ N; $\alpha \in \Sigma$*

- Type 2: context free languages

  A $\rightarrow \psi$; $\psi \in (\Sigma \cup$ N)*

- Type 1: context sensitive languages

  α A $\beta \rightarrow \alpha\psi\beta$; α, $\beta \in \Sigma$*

- Type 0: unrestricted

  α A $\beta \rightarrow \psi$

- The following inclusions hold:

  Type 3 $\subset$ Type 2 $\subset$ Type 1 $\subset$ Type 0

- Simplest formal languages, rules A $\rightarrow$ x, A $\rightarrow$ x B
- Alternative characterization:
  use symbols from the alphabet and combine them using
  - concatenation $\bullet$
  - alternative |
  - Kleene star * (repeat zero or more times)
- Examples:

  {the}•{gifted}•{student}

  {the}•({very}|{extremely})•{gifted}•{student}

  ({0}|{1}|{2}|{3}|{4}|{5}|{6}|{7}|{8}|{9})*•({0}|{2}|{4}|{6}|{8})

Properties of Regular Languages

- Rule systems are *right linear*
- Nonterminal always at the right end of the rule's right hand side: A $\rightarrow$ x , A $\rightarrow$ x B
- A linear (in size of the string) number of steps is enough to answer: w $\in$ L ?

Properties of Regular Languages

- Rule systems are *right linear*
- Nonterminal always at the right end of the rule's right hand side: A $\rightarrow$ x , A $\rightarrow$ x B
- A linear (in size of the string) number of steps is enough to answer: w $\in$ L ?
- Can describe arbitrary long strings, e.g., sheep talk: *ba(a)*h*

Properties of Regular Languages

- Rule systems are *right linear*
- Nonterminal always at the right end of the rule's right hand side: A $\rightarrow$ x , A $\rightarrow$ x B
- A linear (in size of the string) number of steps is enough to answer: w $\in$ L ?
- Can describe arbitrary long strings, e.g., sheep talk: *ba(a)*h*
- Can describe infinite languages

Properties of Regular Languages

- Rule systems are *right linear*
- Nonterminal always at the right end of the rule's right hand side: A $\rightarrow$ x , A $\rightarrow$ x B
- A linear (in size of the string) number of steps is enough to answer: w $\in$ L ?
- Can describe arbitrary long strings, e.g., sheep talk: *ba(a)*h*
- Can describe infinite languages
- What is the simplest thing not possible (*Hotz's question*) $a^n b^n, n \in \mathbf{N}$ only finite counting!

Properties of Regular Languages

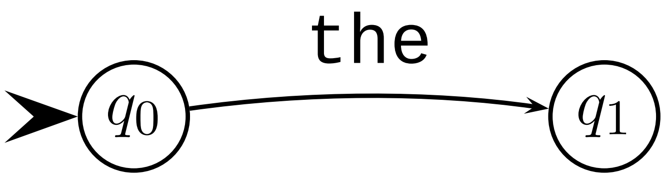
- Rule systems are *right linear*
- Nonterminal always at the right end of the rule's right hand side: A $\rightarrow$ x , A $\rightarrow$ x B
- A linear (in size of the string) number of steps is enough to answer: w $\in$ L ?
- Can describe arbitrary long strings, e.g., sheep talk: *ba(a)*h*
- Can describe infinite languages
- What is the simplest thing not possible (*Hotz's question*) $a^n b^n, n \in \mathbf{N}$ only finite counting!
- Equivalent to *finite automata*

- A finite set of states Q, containing a start state q_0 and a subset of final states F
- An input tape containing the input string and a pointer to mark the current input position
- A transition relation $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times Q$
- Possible moves depend on:
  - the current state
  - the current input symbol
- every move advances the input pointer
- graphical representation: directed graph, states are nodes, edges are state transitions

Nondeterministic Finite Automata

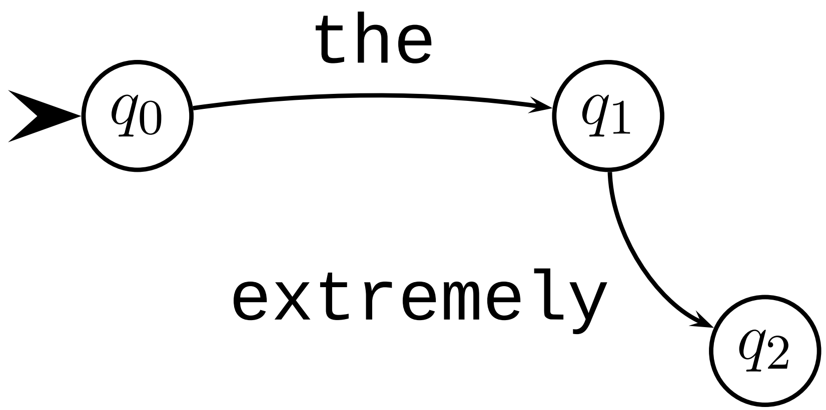
- Automata where δ is a relation and ϵ arcs are allowed are called *nondeterministic automata*
- The move may not be uniquely determined based on the next input symbol
- ex: the (extremely gifted|ϵ) gifted* student

- Automata where δ is a relation and ϵ arcs are allowed are called *nondeterministic automata*
- The move may not be uniquely determined based on the next input symbol
- ex: the (extremely gifted|ϵ) gifted* student



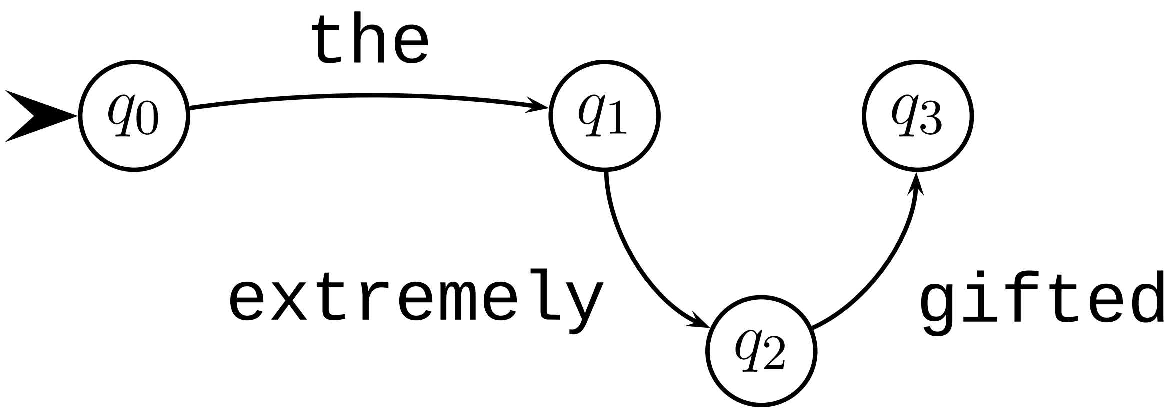
Nondeterministic Finite Automata

- Automata where δ is a relation and ϵ arcs are allowed are called *nondeterministic automata*
- The move may not be uniquely determined based on the next input symbol
- ex: the (extremely gifted|ϵ) gifted* student



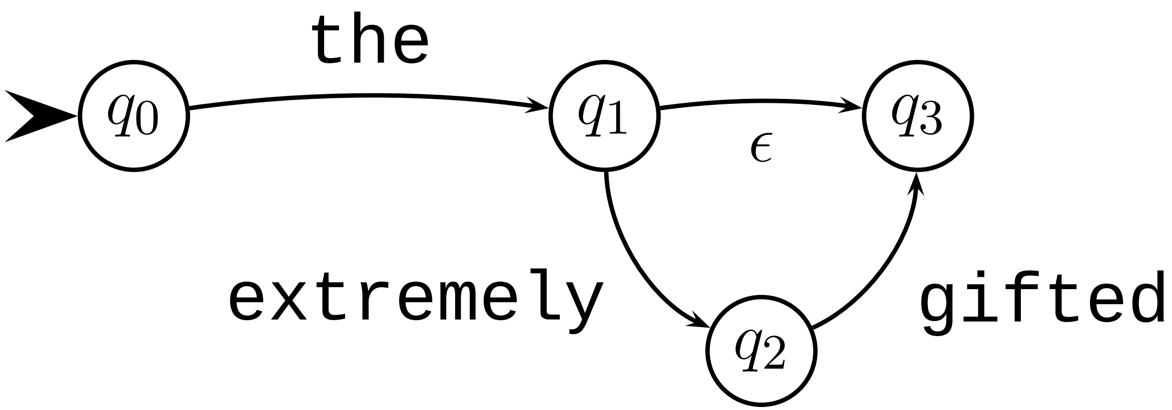
Nondeterministic Finite Automata

- Automata where δ is a relation and ϵ arcs are allowed are called *nondeterministic automata*
- The move may not be uniquely determined based on the next input symbol
- ex: the (extremely gifted|ϵ) gifted* student



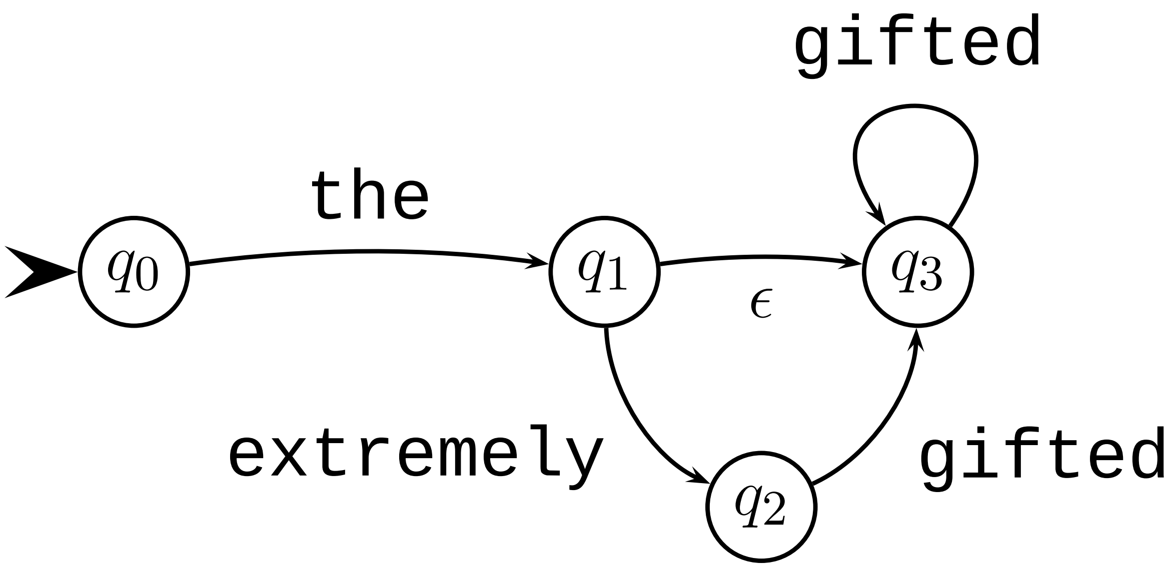
Nondeterministic Finite Automata

- Automata where δ is a relation and ϵ arcs are allowed are called *nondeterministic automata*
- The move may not be uniquely determined based on the next input symbol
- ex: the (extremely gifted|ϵ) gifted* student



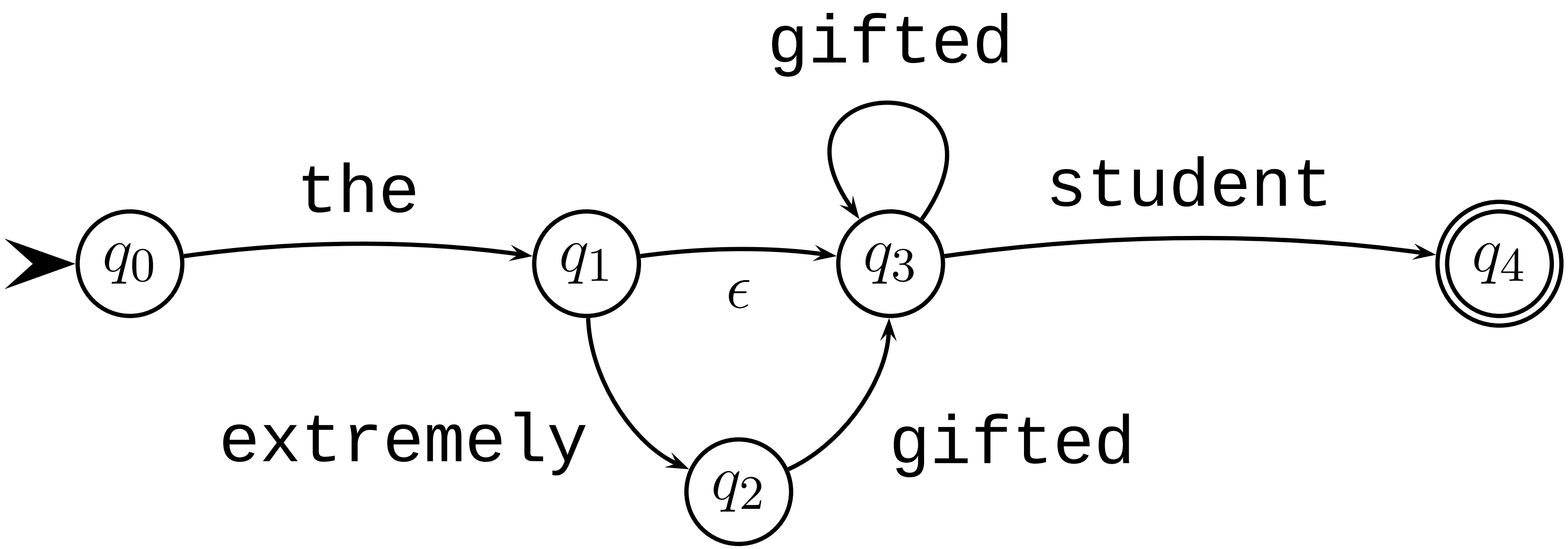
Nondeterministic Finite Automata

- Automata where δ is a relation and ϵ arcs are allowed are called *nondeterministic automata*
- The move may not be uniquely determined based on the next input symbol
- ex: the (extremely gifted|ϵ) gifted* student



Nondeterministic Finite Automata

- Automata where δ is a relation and ϵ arcs are allowed are called *nondeterministic automata*
- The move may not be uniquely determined based on the next input symbol
- ex: the (extremely gifted|ϵ) gifted* student

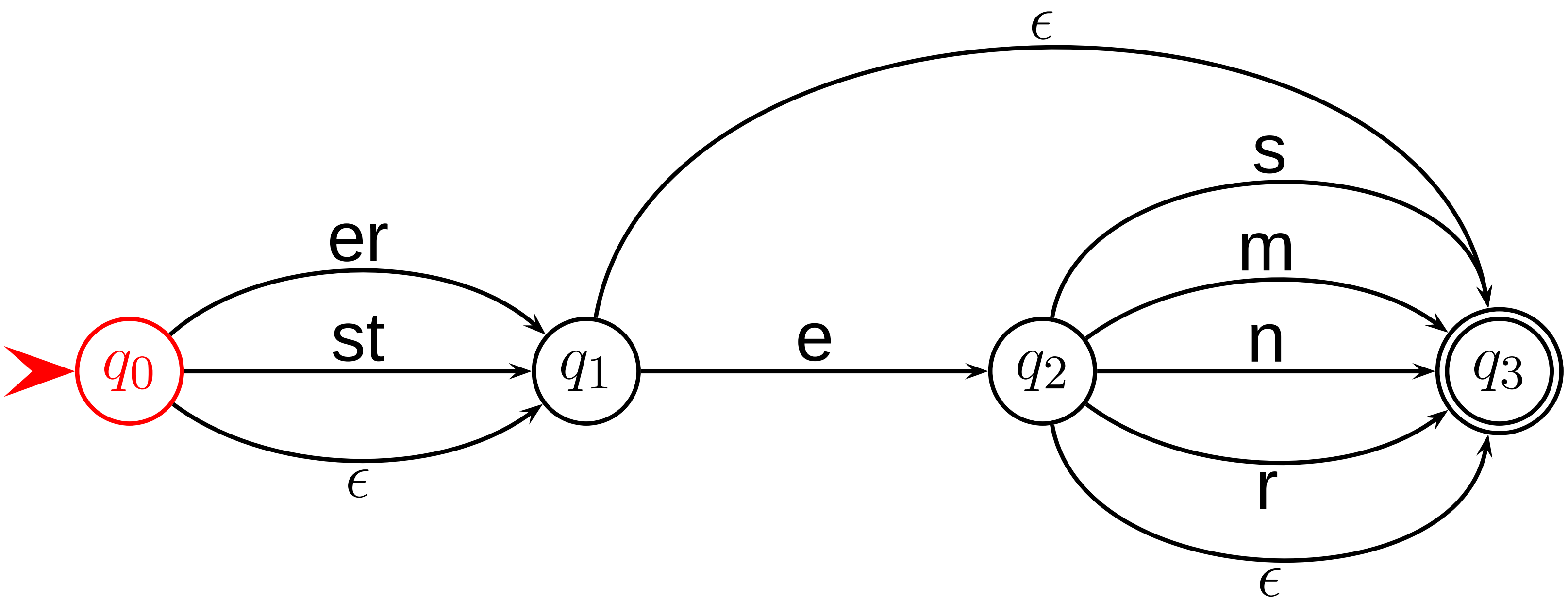


Closure Properties

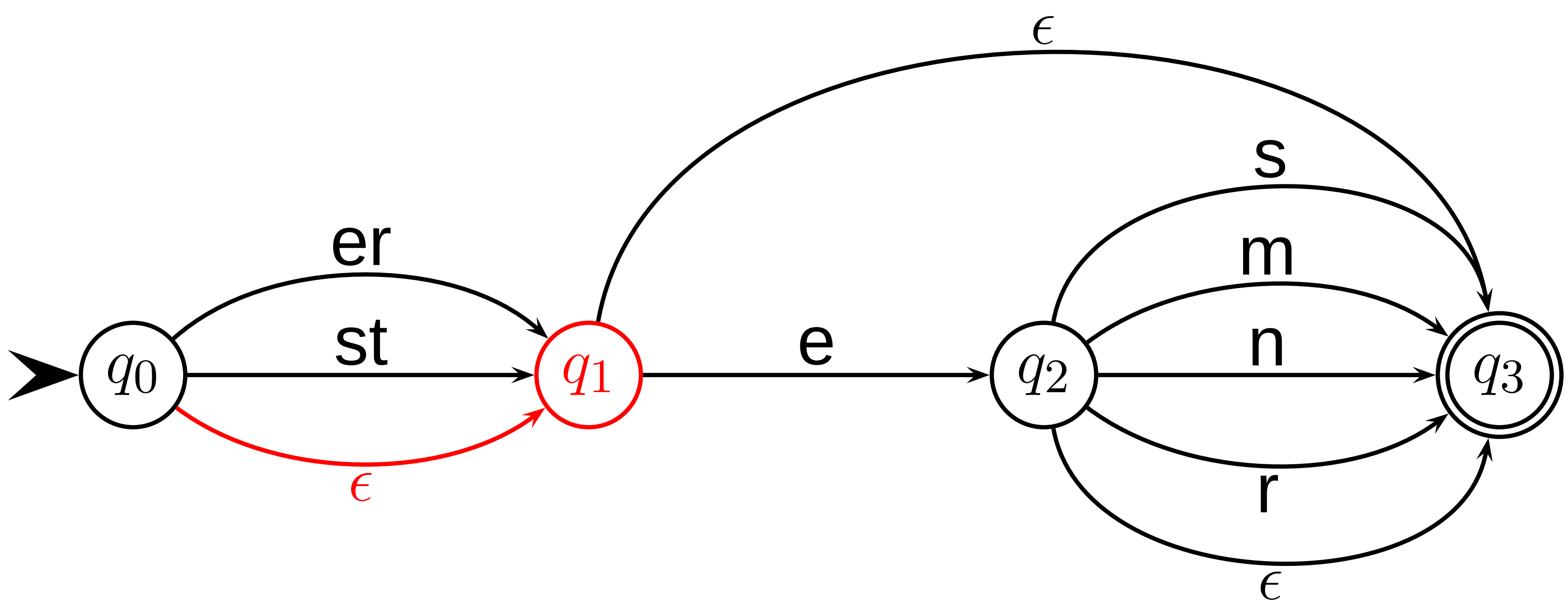
- Language type *A* is closed unter operation *x* means: applying *x* to members of *A* results in element of the same type
- Regular languages are closed under
  - ➢ Concatenation, Union (trivial)
  - ➢ Complementation: Exchange final and nonfinal states of an automaton
  - ➢ Intersection: $\mathsf{L}_1 \cap \mathsf{L}_2 = \neg(\neg\mathsf{L}_1 \cup \neg\mathsf{L}_2)$
- Applicability of these operations facilitates modularization
- E.g., concatenate automaton for base word forms with one for inflectional suffixes

Finite Automata: Search

- German adjective ending
- Input: **klein** + er + es



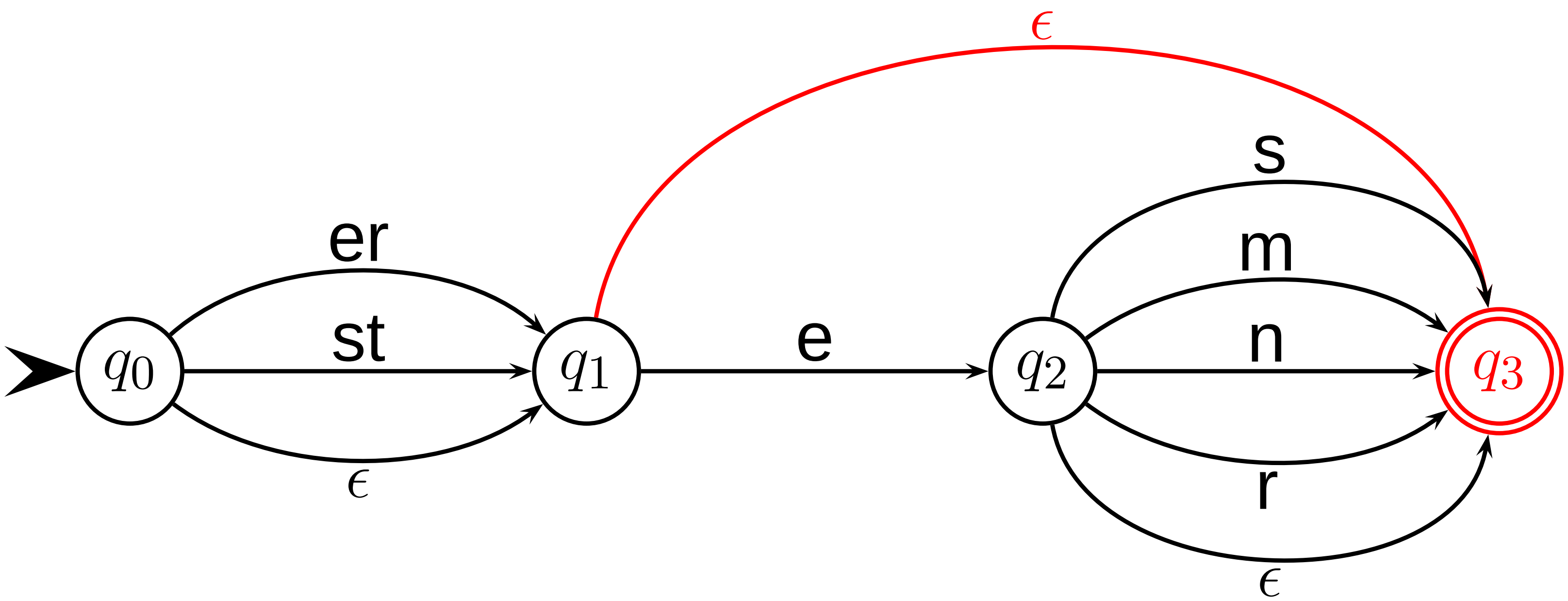
- German adjective ending
- Input: **klein** + er + es



Finite Automata: Search

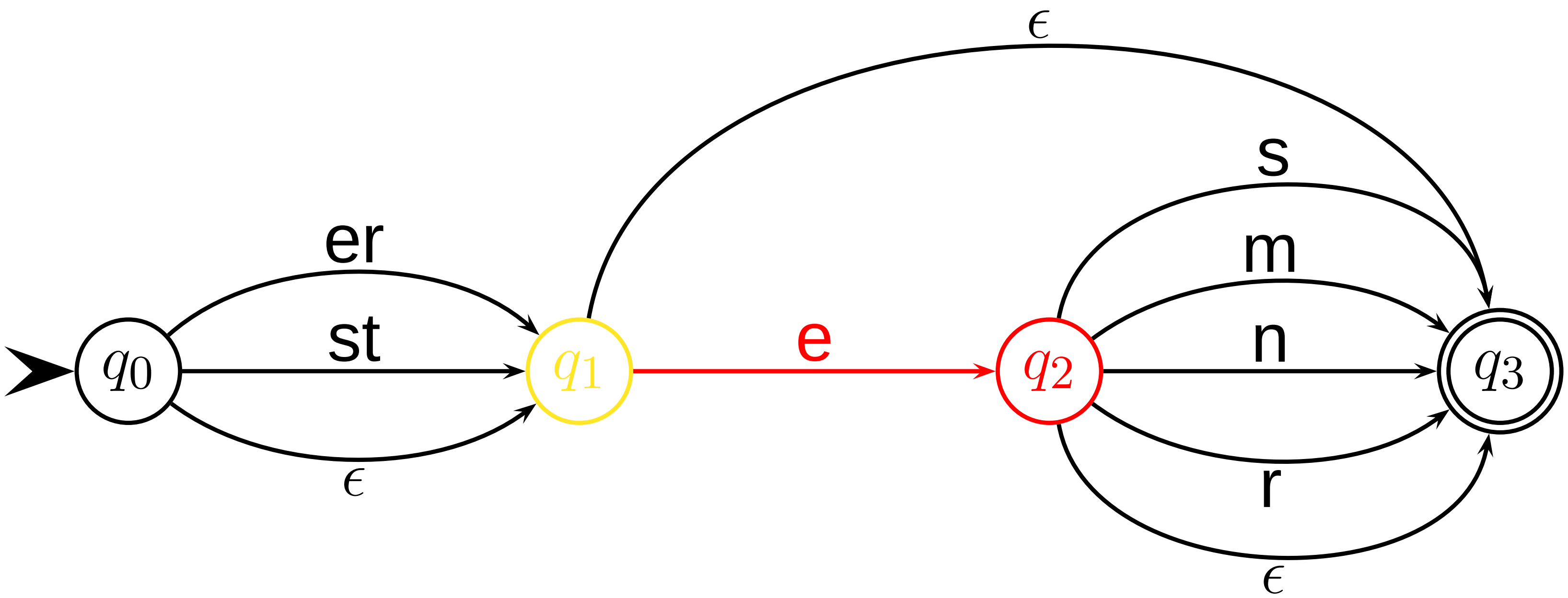
- German adjective ending
- Input: **klein** + er + es

Failure!



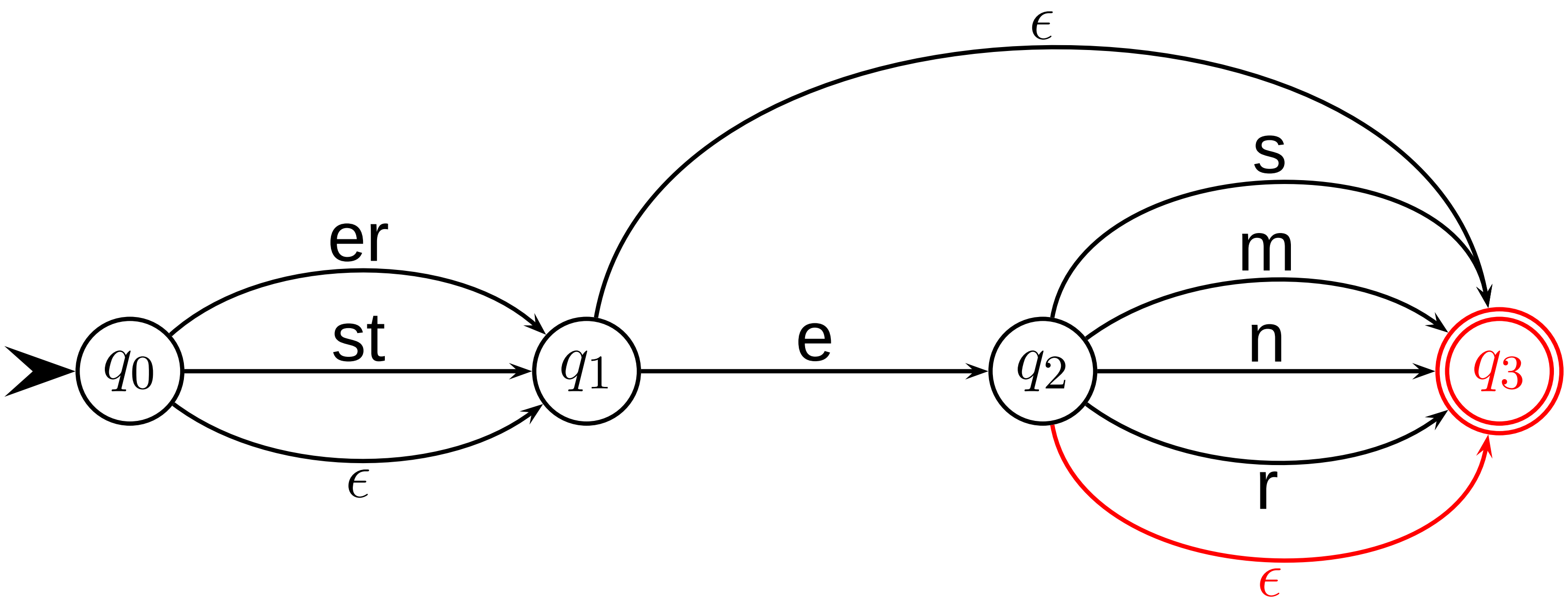
- German adjective ending
- Input: **klein** + er + es

Backtracking



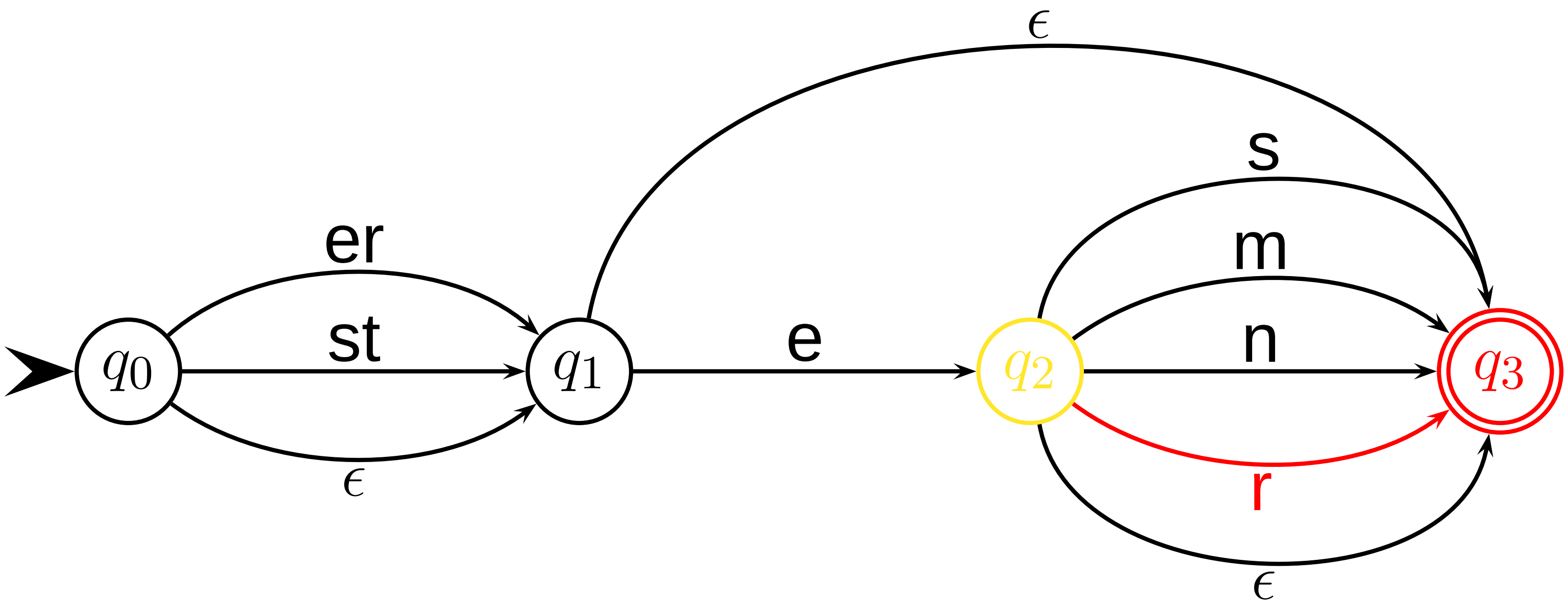
- German adjective ending
- Input: **klein** + er + es

Failure!



- German adjective ending
- Input: **klein** + er + es

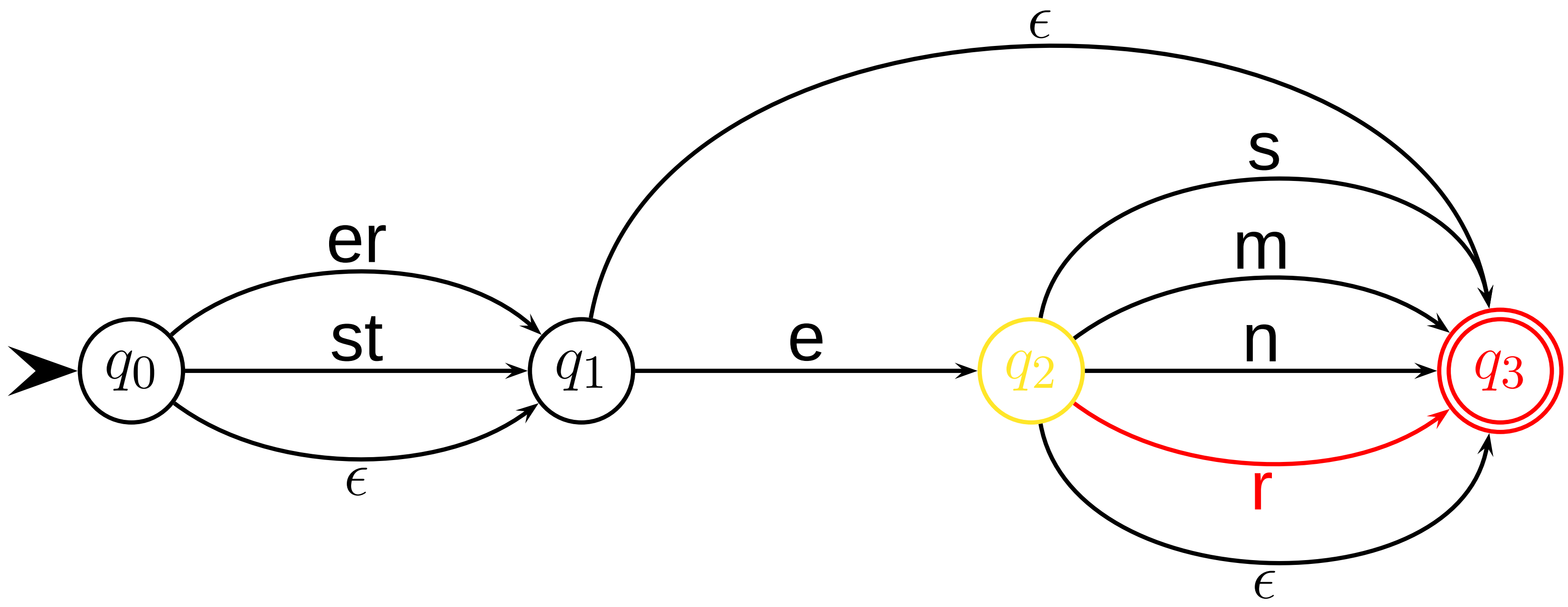
Backtracking



- German adjective ending
- Input: **klein** + er + es

Backtracking

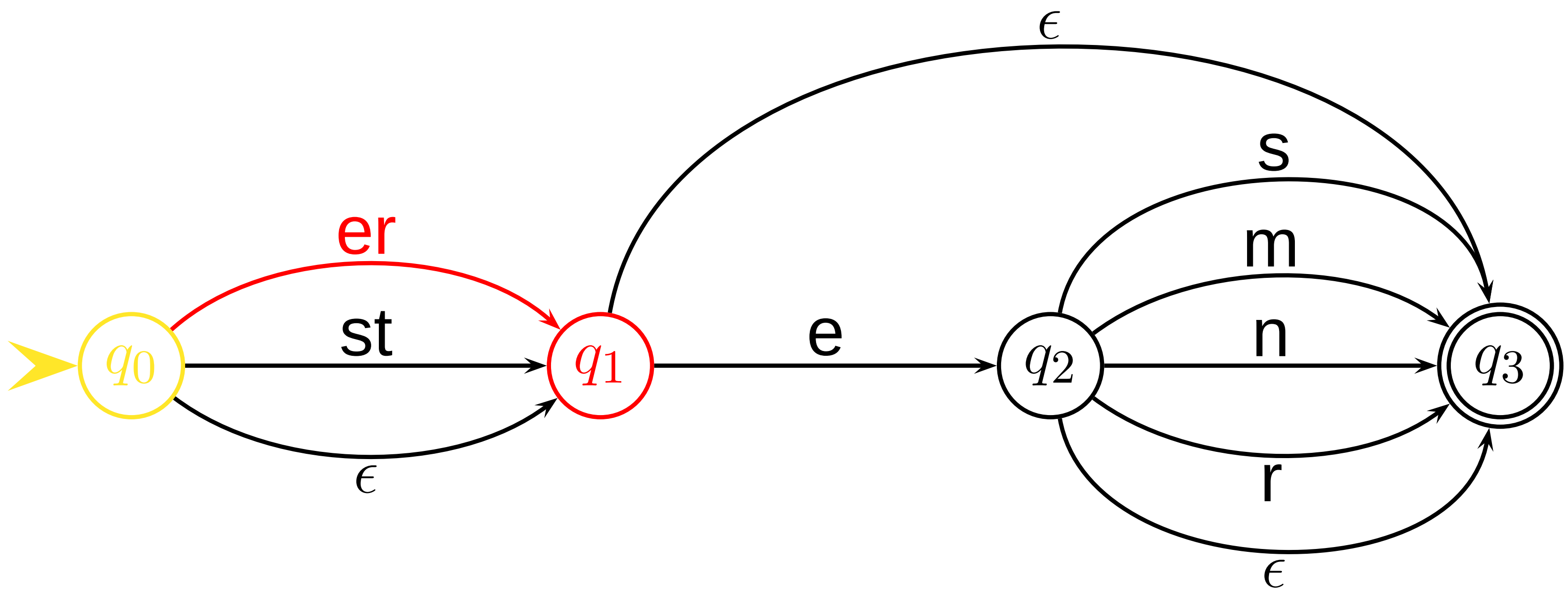
Failure!



Finite Automata: Search

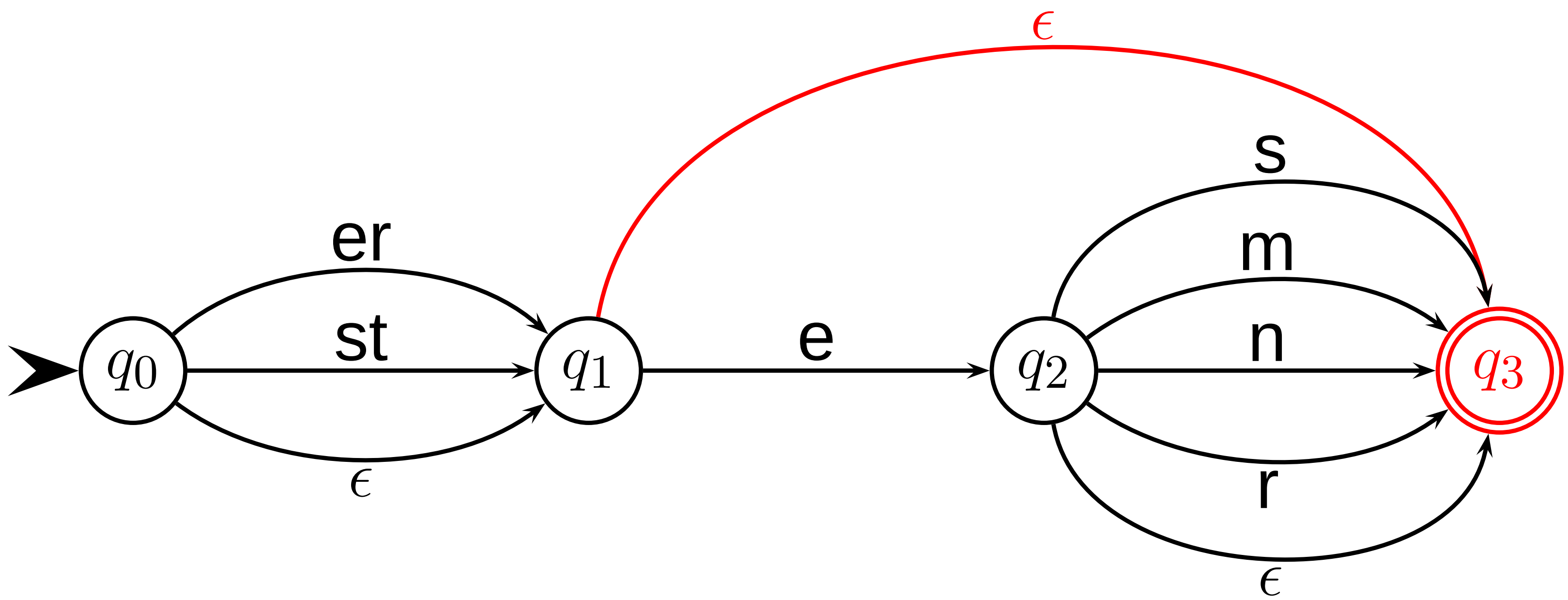
- German adjective ending
- Input: **klein** + er + es

Backtracking



- German adjective ending
- Input: **klein** + er + es

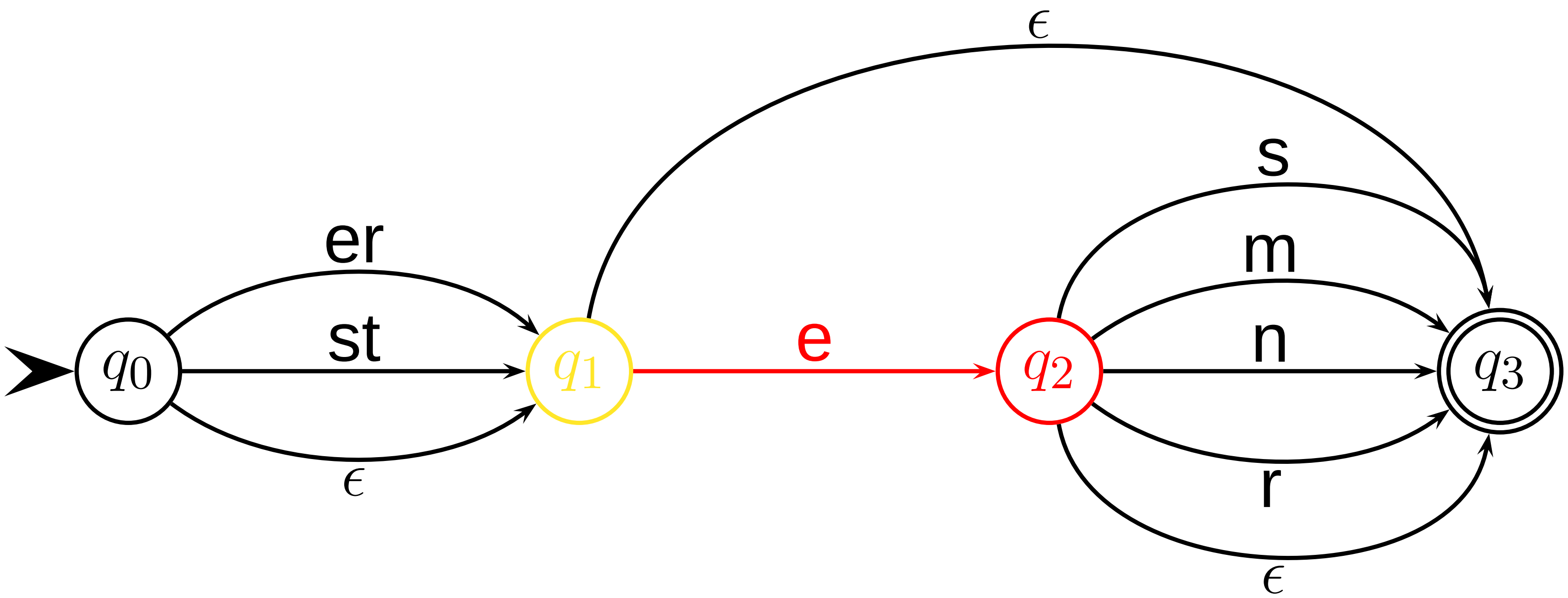
Failure!



Finite Automata: Search

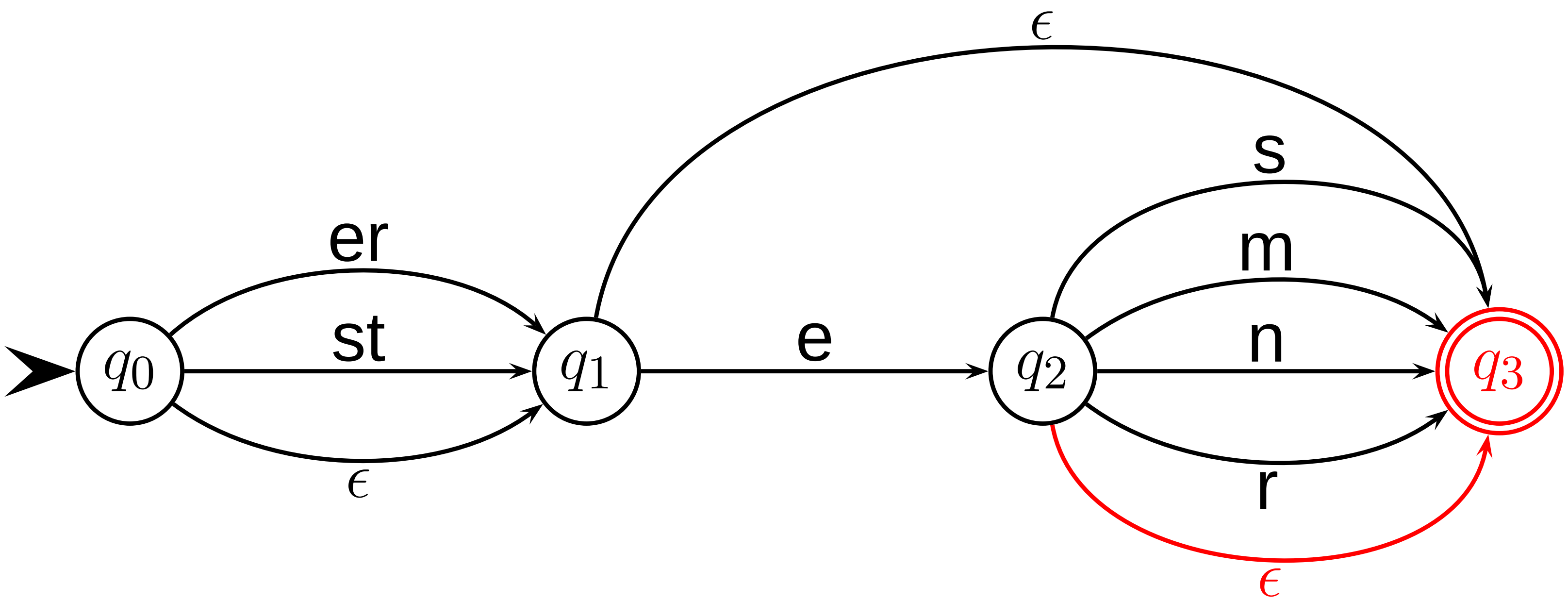
- German adjective ending
- Input: **klein** + er + es

Backtracking



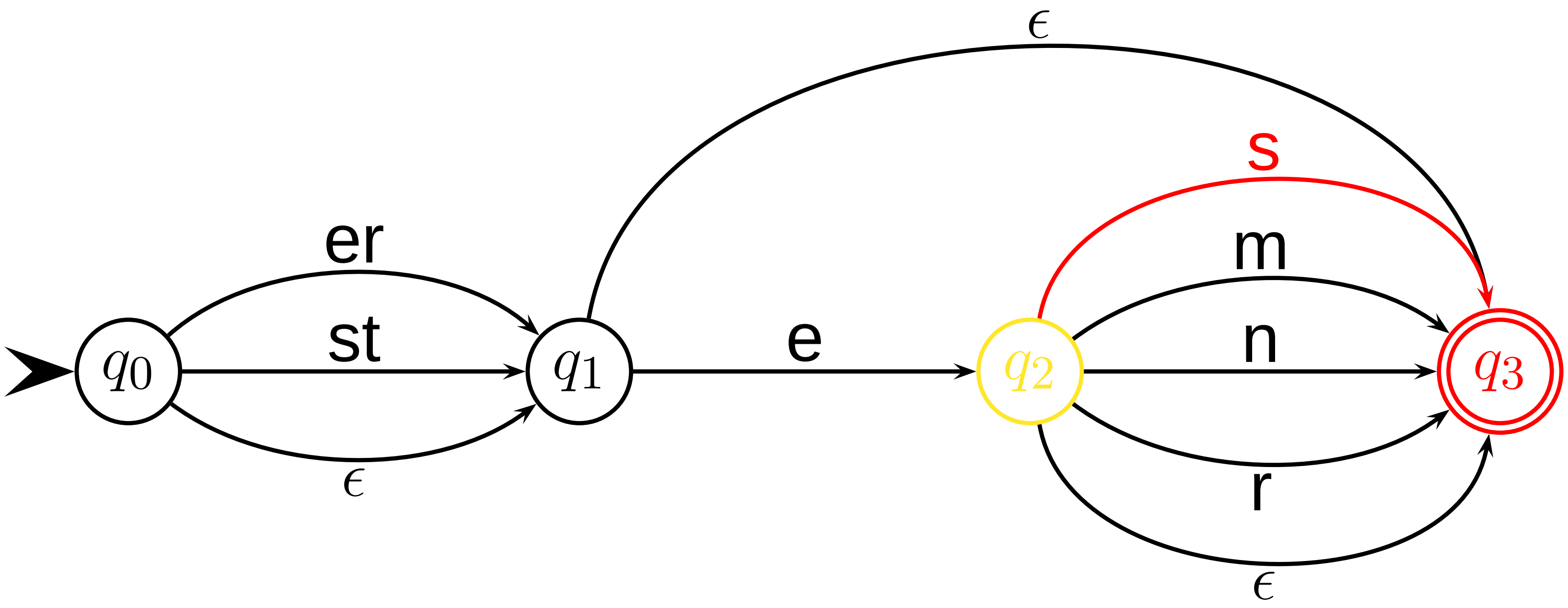
- German adjective ending
- Input: **klein** + er + es

Failure!



- German adjective ending
- Input: **klein** + er + es

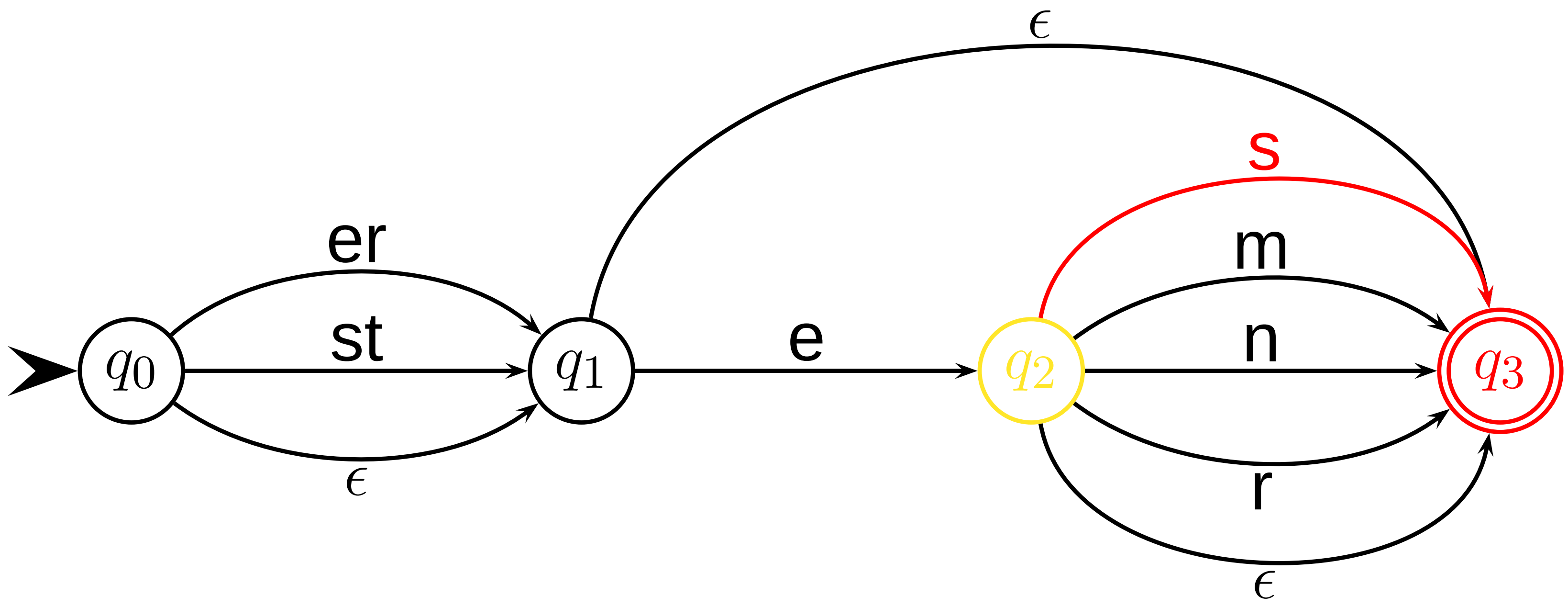
Backtracking



- German adjective ending
- Input: **klein** + er + es

Backtracking

Success!



Nondeterministic vs. Deterministic

- Search becomes a problem in big automata
- Solution: *determinisation*
  - the transition relation has to be a *total function* $Q \times \Sigma \rightarrow Q$: exactly one choice
  - for every nondeterministic automaton, a deterministic automaton can be constructed that accepts the same language
  - recognition linear in size of the string
  - but: the size of the automaton can be exponential in size of original automaton

Advantages of Finite Automata

- efficiency
  - ➢ very fast if deterministic or low-degree non-determinism
  - ➢ space: compressed representations of data
- system development and maintenance
  - ➢ modular design and automatic compilation of system components
  - ➢ high level specifications
- language modelling
  - ➢ uniform framework for modelling dictionaries and rules

FSA for Morphology

- Let's first have a look at concatenative morphology
  - cats : **cat** + s
  - unbelieveable: un + **believe** + able
- Use different automata for
  - prefixes
  - base form ⇒ lexicon (we'll do this first)
  - suffixes

  and combine them with concatenation
- recognition is not enough: analysis should return information, e.g., inflectional class
- idea: associate final states with information

Why not simply list all words?

Why not simply list all words?

stiff	*pos*
stiffer	*comp*
stiffest	*sup*
stiffly	*adv*
still	*pos & adv*
stiller	*comp*
stillest	*adv*
stout	*pos & adv*
stouter	*comp*
stoutest	*sup*
stony	*pos*
stonier	*com*
⋮	

- large, wasteful, incomplete

Lexicon representation

Why not simply list all words?

stiff	*pos*
stiffer	*comp*
stiffest	*sup*
stiffly	*adv*
still	*pos & adv*
stiller	*comp*
stillest	*adv*
stout	*pos & adv*
stouter	*comp*
stoutest	*sup*
stony	*pos*
stonier	*com*
⋮	

- large, wasteful, incomplete
- no (morphological) handling of new words

Lexicon representation

Why not simply list all words?

stiff	*pos*
stiffer	*comp*
stiffest	*sup*
stiffly	*adv*
still	*pos & adv*
stiller	*comp*
stillest	*adv*
stout	*pos & adv*
stouter	*comp*
stoutest	*sup*
stony	*pos*
stonier	*com*
⋮	

- large, wasteful, incomplete
- no (morphological) handling of new words
- what about languages with a more productive morphology, e.g., Finnish or Turkish?

Lexicon representation

Why not simply list all words?

stiff	*pos*
stiffer	*comp*
stiffest	*sup*
stiffly	*adv*
still	*pos & adv*
stiller	*comp*
stillest	*adv*
stout	*pos & adv*
stouter	*comp*
stoutest	*sup*
stony	*pos*
stonier	*com*
⋮	

- large, wasteful, incomplete
- no (morphological) handling of new words
- what about languages with a more productive morphology, e.g., Finnish or Turkish?

➤ Encode each phenomenon / process in one automaton

➤ Combine them and get an efficient machine

Lexicon representation

stiff	*pos*
stiffer	*comp*
stiffest	*sup*
stiffly	*adv*
still	*pos & adv*
stiller	*comp*
stillest	*adv*
stout	*pos & adv*
stouter	*comp*
stoutest	*sup*
stony	*pos*
stonier	*com*
⋮	

Separate base form and modifications e.g., (inflectional) affixes:

stiff
still
stout
stony
stolen
straight
⋮

+ ϵ	*pos*	
+ er	*comp*	
+ est	*sup*	
+ ly	*adv*	*really?*

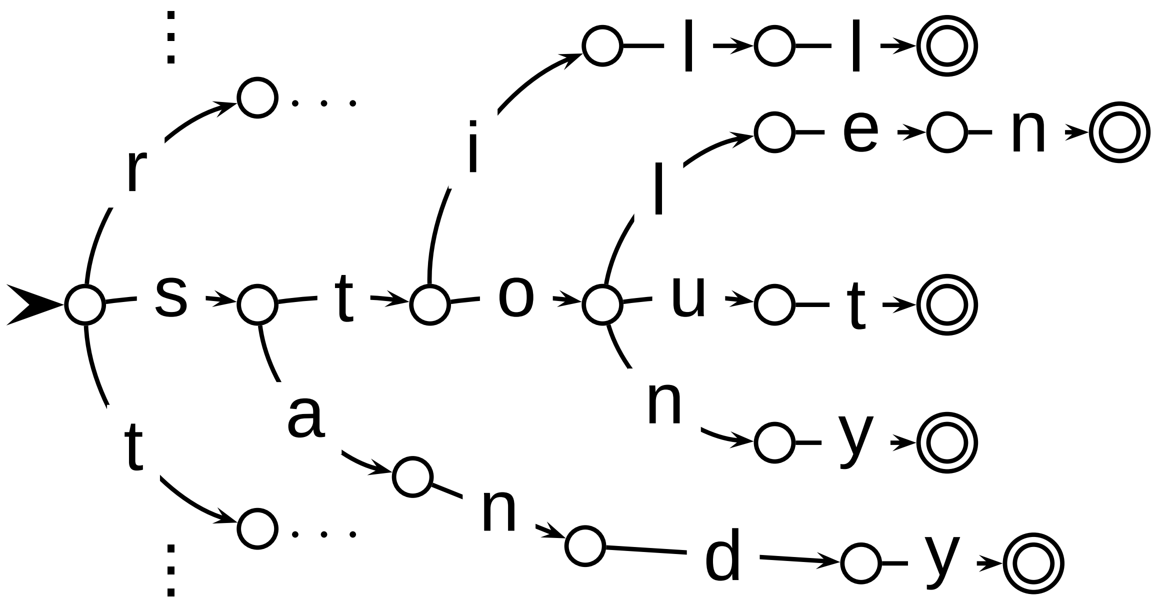
Other morphological processes like *un-*negation:

un + **happy**

un + **clear** + ly

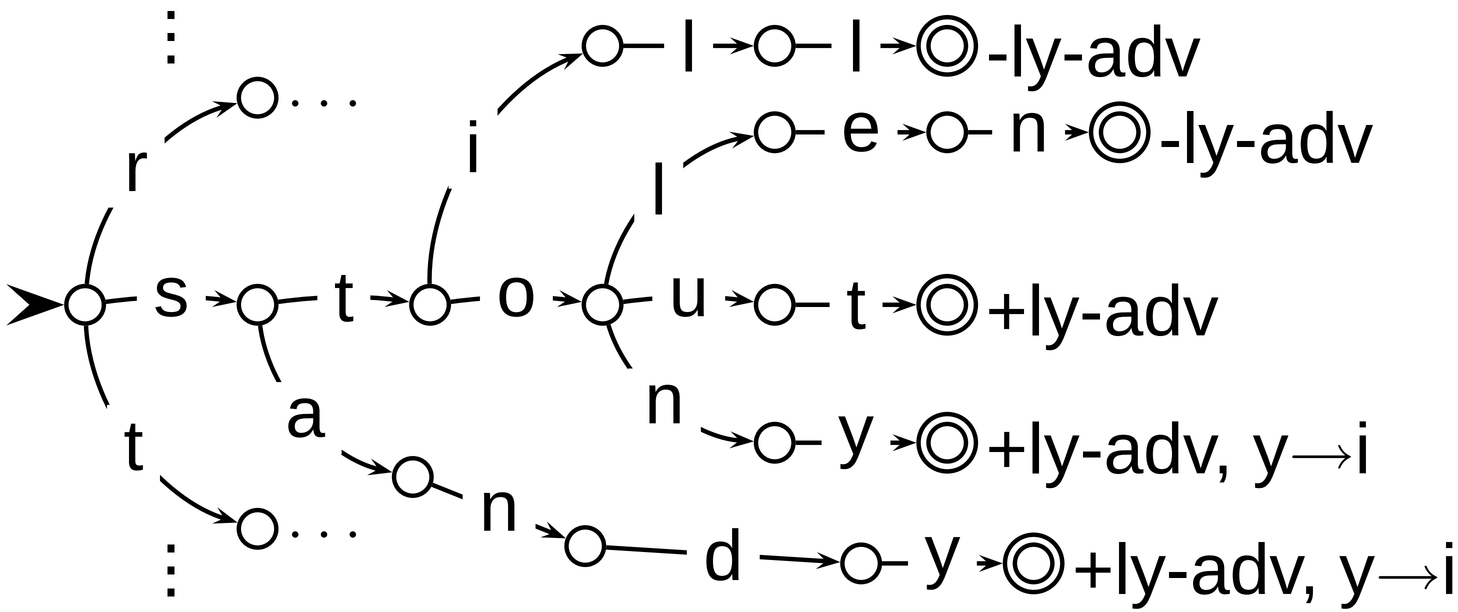
. . ., sandy, still, stolen, stony, stout, . . .

1. construct a letter tree (or *trie*); leaves ≡ final nodes



. . ., sandy, still, stolen, stony, stout, . . .

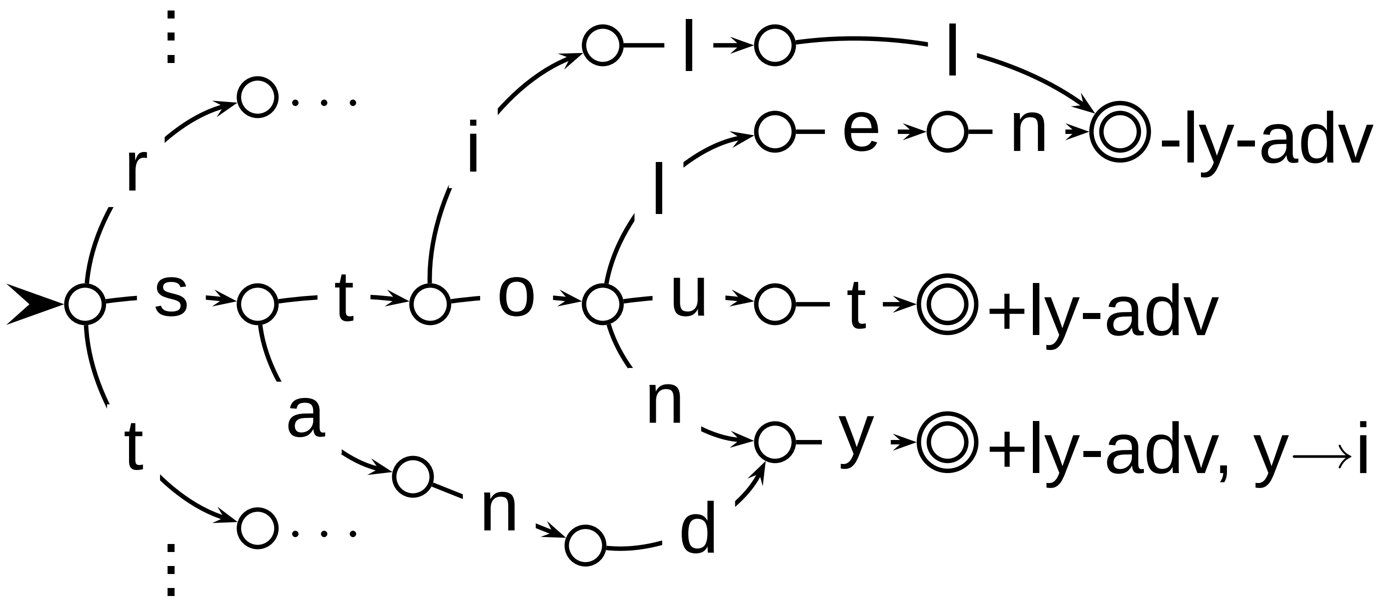
1. construct a letter tree (or *trie*); leaves ≡ final nodes
2. associate the leaves with lexical information



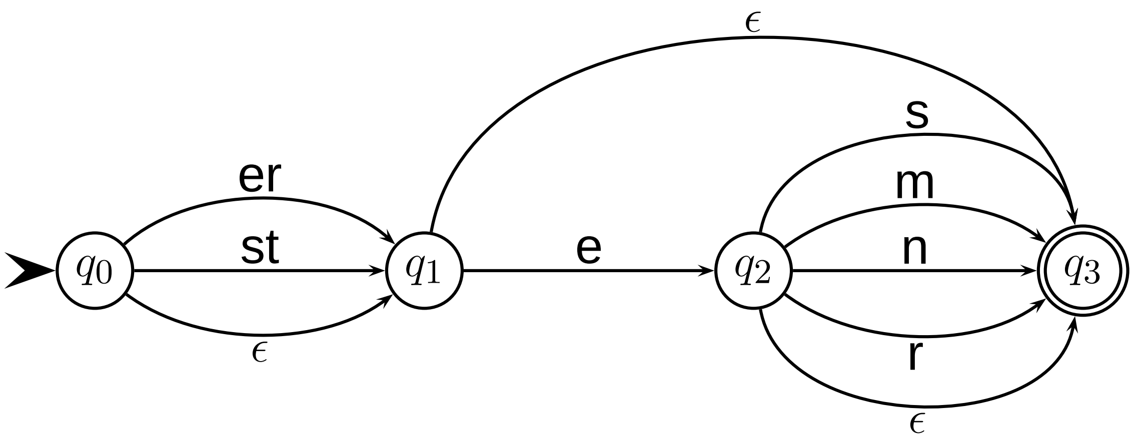
Lexicon Automaton

..., sandy, still, stolen, stony, stout, ...

1. construct a letter tree (or *trie*); leaves ≡ final nodes
2. associate the leaves with lexical information
3. merge the nodes with identical information
   - *minimize the automaton*

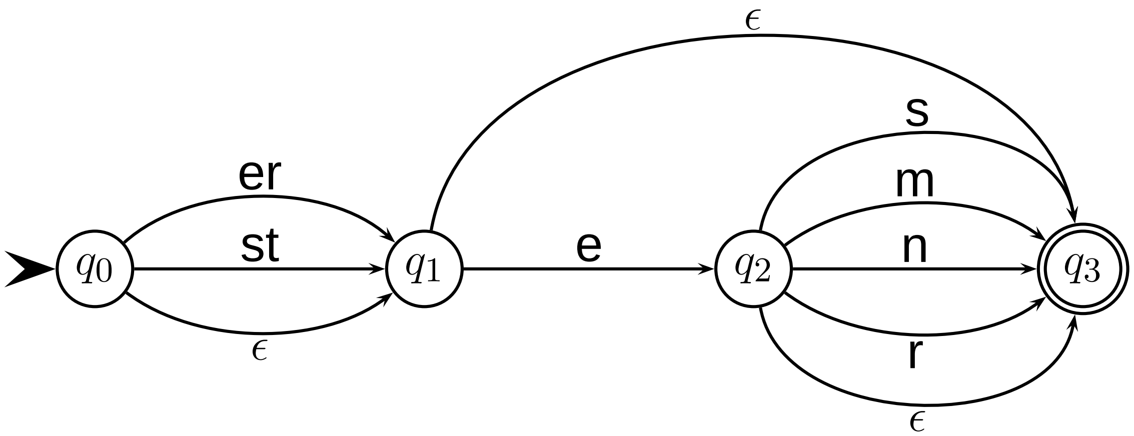


Suffixes: German Adjectives

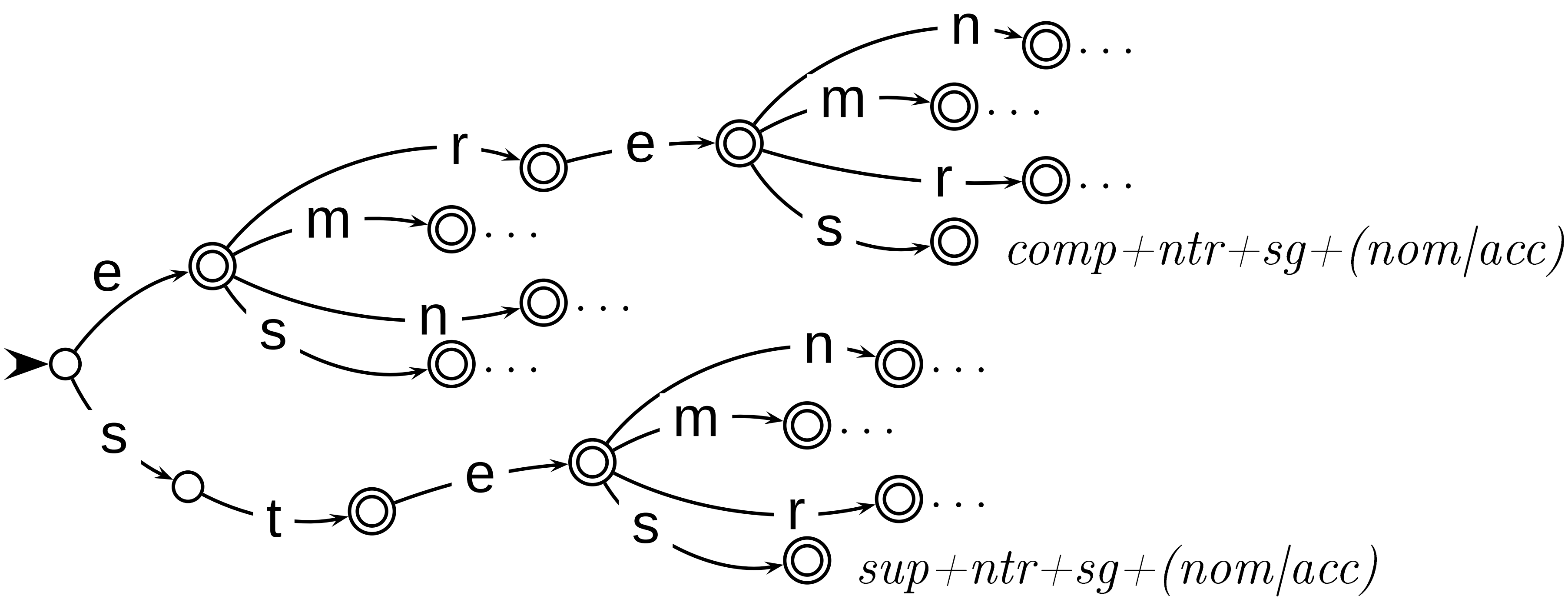


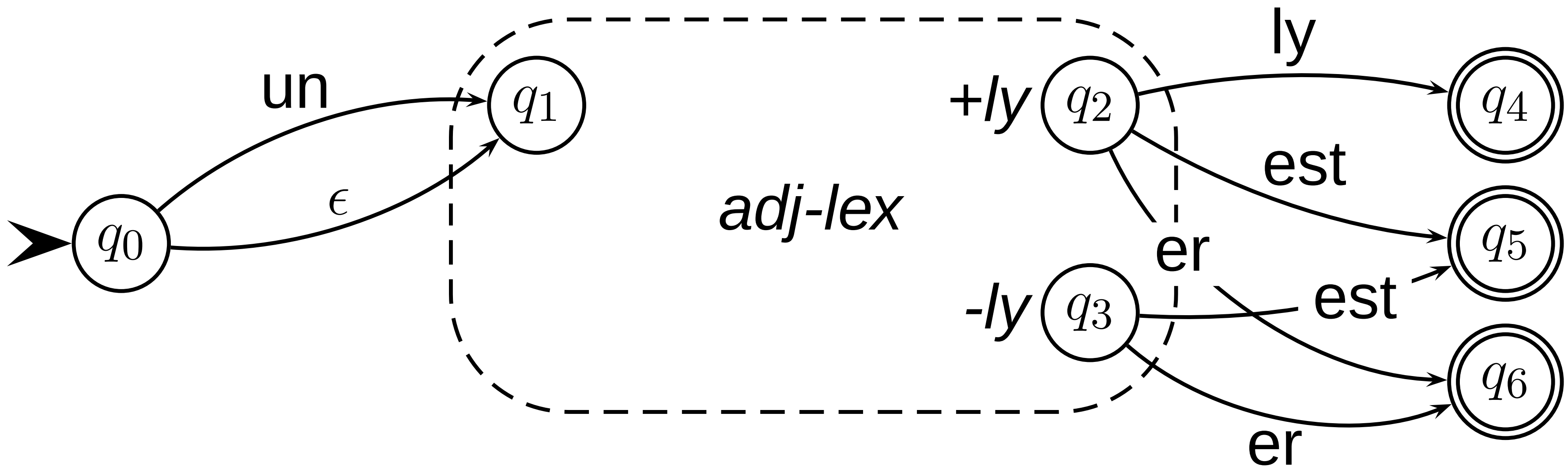
Only one final state: How to get the different values?

Suffixes: German Adjectives



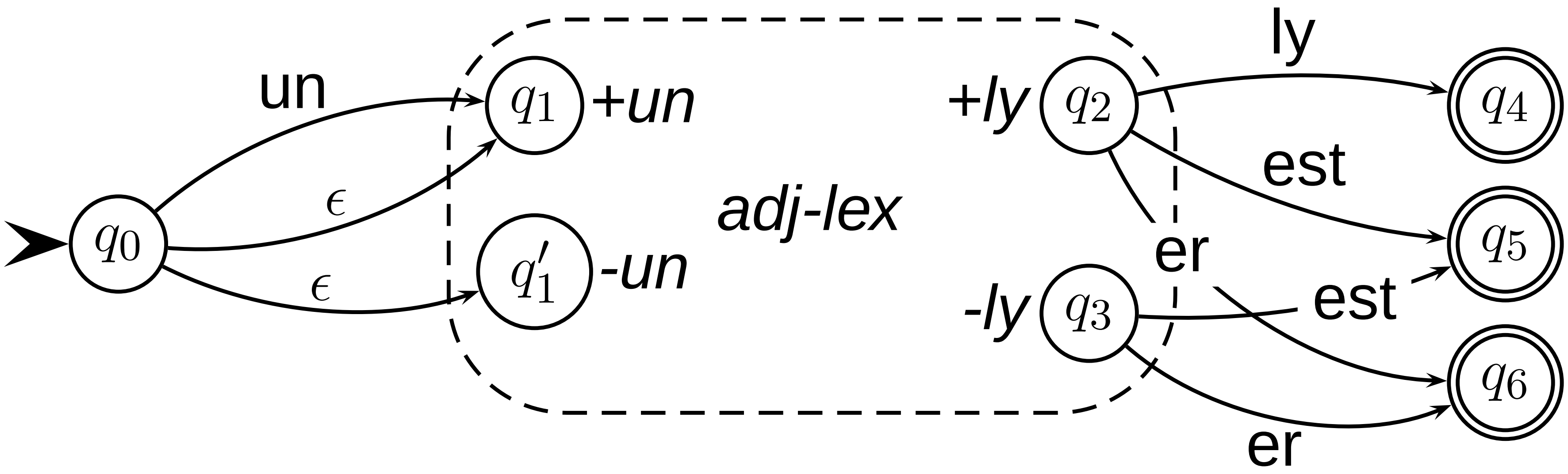
final states with different information can not be combined: *expand automaton*





- What about: un… with big; …ly with still?

Combining the Levels



- What about: un… with big; …ly with still?
- Split startnodes in *adj-lex*, like the final nodes
- But: splits the lexicon, less compact
- Alternative: special flags that are handled by the machinery

Two-Level Morphology

- Represents a word as correspondence between two levels
  - Lexical level: abstract morphemes and features
  - Surface level: the actual spelling of the word
- Can be implemented using *finite state transducers*
- A finite state transducer rewrites the input onto a second, additional tape

Lexical			**c**	**a**	**t**	**+N**	**+PL**			
Surface			**c**	**a**	**t**	**s**				

Automaton vs. Transducer

- Finite-state Automaton
  - ➤ Arcs are labeled with symbols like a and b
  - ➤ Accepts strings like aaab
  - ➤ Defines a regular language: { a, ab, aab, aaab, ... }
- Finite-state Transducer
  - ➤ Arcs are labeled with symbol pairs like a:b and b:b, but also b:ϵ and ϵ:a (and b as shorthand for b:b)
  - ➤ Accepts a *pair* of strings like aaab:aabb
  - ➤ Defines a regular relation: { a:b, aa:bb, aaa:bbb, ... }
- We will use it to accept string pairs like cat+N+PL:cats and fox+N+PL:foxes

Four Views on Transducers

Lexical			**c**	**a**	**t**	**+N**	**+PL**			
Surface			**c**	**a**	**t**	**s**				

1. **Recognizer:** machine that accepts or rejects pairs of strings
2. **Generator:** machine that outputs pairs of strings
3. **Translator:** machine that reads one string and outputs another string (in both directions)
4. **Set Relator:** machine that computes relations between sets

Cascaded Transducers

- To accomodate for all spelling / pronounciation changes, one transducer alone is not powerful enough
- Use *intermediate* tapes that contain the output of one transducer and serves as input to another transducer
- To handle irregular spelling changes, we can add intermediate tapes with intermediate symbols:
  ˆ for morpheme boundary, # for word boundary

Lexical			**f**	**o**	**x**	**+N**	**+PL**			
Surface			**f**	**o**	**x**	**ˆ**	**s**	**#**		

Some English Orthograpic Rules

- English orthographic rules that apply at particular morpheme boundaries

Name	Description of rule	Example
consonant doubling	consonant doubled before *-ing/-ed*	beg / begging
e-deletion	silent *e* dropped before *-ing/-ed*	make / making
e-insertion	e added between *-s, -z, -x, -ch, -sh* and *-s*	watch / watches
y-replacement	*-y* changes to *-ie* before *-s*, to *-i* before *-ed*	try / tries
k-insertion	verbs ending with vowel + *-c* add *-k*	panic / panicked

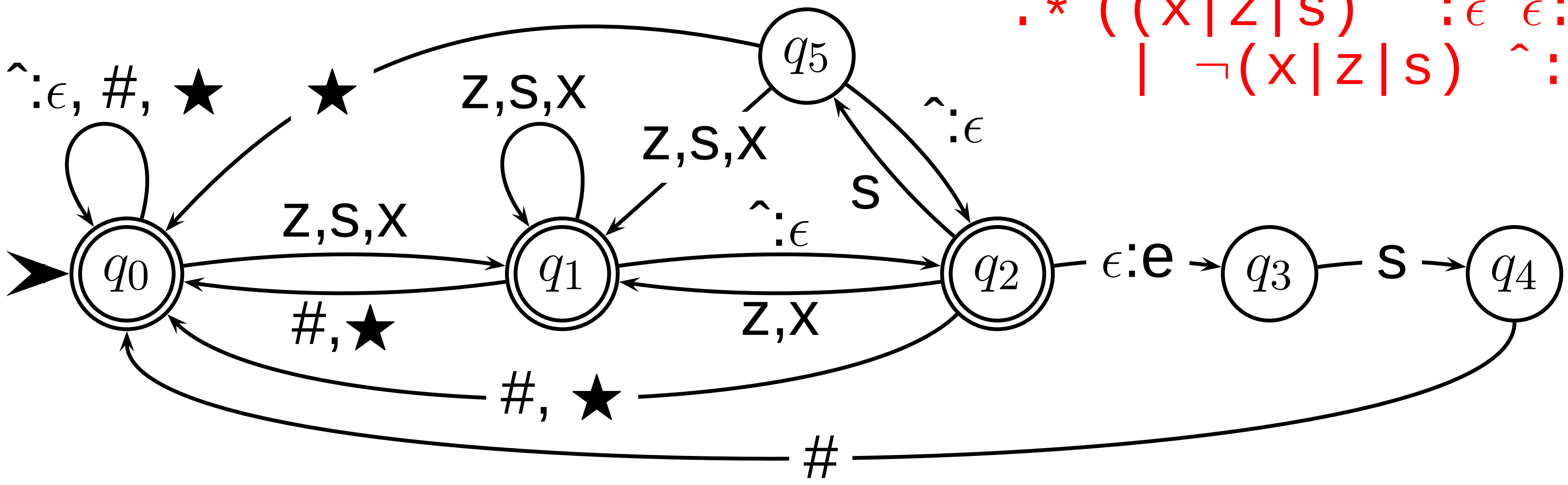
Orthograpic Rules II

- Spelling rules take the concatenation of morphemes – the *intermediate* tape – as input and produce the surface form
- Example: e-insertion rule is applied to the intermediate form foxˆs#

Lexical			**f**	**o**	**x**	**+N**	**+PL**			
Intermediate			**f**	**o**	**x**	**ˆ**	**s**	**#**		
Surface			**f**	**o**	**x**	**e**	**s**			

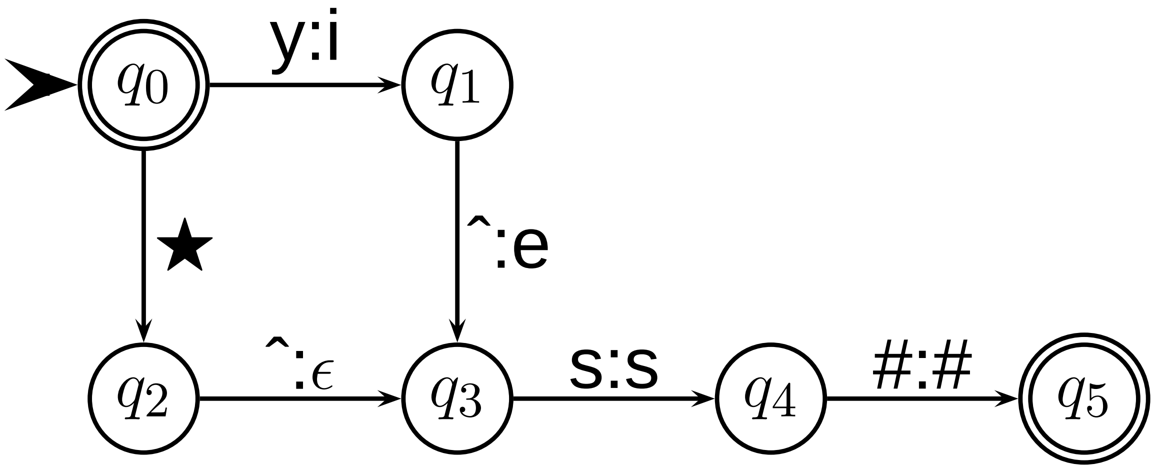
e-Insertion

.* ((x|z|s) ˆ:ϵ ϵ:e
| ¬(x|z|s) ˆ:ϵ) s#



- rule: ((z|s|x) ˆ:ϵ ϵ:e | ¬(z|s|x) ˆ:ϵ) s #
- ★: all pairs not in this transducer, remember y is y:y
- States q_0 and q_1 accept default pairs like `catˆs#:cats#`
- State q_5 rejects incorrect pairs like `foxˆs#:foxs#`

y-Replacement



- Ex.: spy+s $\rightarrow$ spies
- rule: .* ((y:i ^:e)|($\neg$ y ^:ϵ)) #
- All these machines do not change input to which they do not apply
- Nevertheless, the rule writer must take care of all interactions

Summary

- The task of morphological analysis/generation
- (Very short) introduction to formal languages
- Basics of regular languages
- Nondeterministic and deterministic finite automata
- Applying finite state techniques to morphological knowledge
  - ➤ Lexicon: compacted tries
  - ➤ Concatenative phenomena: finite automata
  - ➤ Associating information with final states
  - ➤ Derivational phenomena: finite state transducers

References

Beesley, Kenneth R. and Lauri Karttunen (2003). *Finite-State Morphology.* CSLI Publications. `www.fsmbook.com`

Jurafsky, Daniel and James H. Martin (2000). *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition.* New Jersey: Prentice Hall.

Koskenniemi, Kimmo (1983). *Two-level morphology: a general computational model for word-form recognition and production.* Publication No:11, University of Helsinki, Department of General Linguistics, 1983.

Mohri, Mehryar (1996). *On some Applications of finite-state automata theory to natural language processing.* In: Journal of Natural Language Egineering, 2, pp 1-20.

Xerox Finite State Compiler (Web Demo):
`http://www.xrce.xerox.com/competencies/content-analysis/fsCompiler/fsinput.html`