Natural Language Processing

Timo Baumann

OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

This week

- Your questions after the first week
- recap: Chomsky hierarchy of formal languages
- our first real topic:
  finite state methods applied to morphology
- exercise: work with finite state transducers (Friday)
  - by hand and using pre-existing software

Your questions

Formal Languages and Automata

- Automata theory and theory of formal languages are a part of theoretical computer science
- Their concepts originate in theoretical linguistics: Noam Chomsky is the originator of the Chomsky hierarchy of formal languages

Why talk about it?

- complexity of sub-systems of natural language informs complexity of automatic processing machinery
- Fundamental results from theoretical computer science have direct implications on implementations for language technology applications

Definitions

- a **LANGUAGE** is a collection of sentences of finite length all constructed from a finite alphabet of symbols
- a **GRAMMAR** can be regarded as a device that enumerates the sentences of a language
- a grammar of language L can be regarded as a function whose range is exactly L

Formal Grammar

A **formal grammar** is a quad-tuple *G = (Φ,Σ,R,S)* where

- Φ is a finite set of non-terminals
- Σ a finite set of terminals, disjoint from Φ
- R a finite set of production rules of the form
  $\alpha \in (\Phi \cup \Sigma)^* \rightarrow \beta \in (\Phi \cup \Sigma)^*$ with $\alpha \neq \varepsilon$ and $\alpha \notin \Sigma^*$
- S, Element of Φ : start symbol

Derivation, Formal Language, Automaton

Let $G = (\Phi,\Sigma,R,S)$ be a formal grammar and let $u,v \in (\Phi \cup \Sigma)^*$.

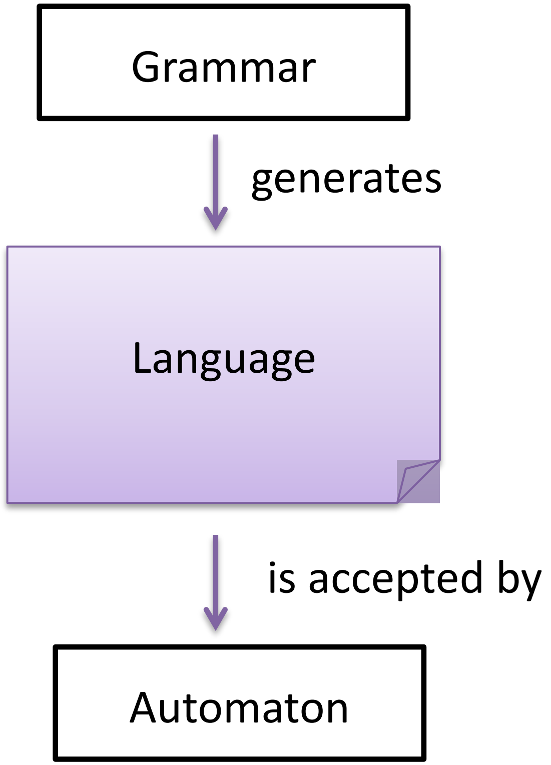
1. v is **directly derivable** from u, noted $u \Rightarrow v$, if
   $u = awb$, $v=azb$ and $w \rightarrow z$ is a production rule in R.

2. v is **derivable** from u, noted $u \overset{*}{\Rightarrow} v$, if there are words $w_0..w_k$, such that $u \Rightarrow w_0$, $w_{n-1} \Rightarrow w_n$ for all $0<n\leq k$ and $w_n \Rightarrow v$.

Let $G = (\Phi,\Sigma,R,S)$ be a formal grammar. Then, $L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$ is the **formal language** generated by G.

An **automaton** is a device that decides, whether a given sentence belongs to a formal language

Generation and Acceptance



The complexity of the generating grammar influences the complexity of the accepting automaton

Type-0 Grammar: unrestricted

The mechanism of unrestricted grammars allows the definition of very complex languages that in turn need very complex automata. Restrictions on the form of production rules lead to different types of grammars.

An **unrestricted** formal grammar is called **Type-0 grammar** and can be accepted by a **Turing Machine**. It produces **recursively enumerable** languages.

Type-1 Grammar: context-sensitive

A grammar *$G = (\Phi,\Sigma,R,S)$* is **context-sensitive**, iff all production rules in R obey the form

- either $\alpha A\gamma \rightarrow \alpha\beta\gamma$ with $\alpha,\beta,\gamma \in (\Phi\cup\Sigma)^*$, $A\in\Phi$, $\beta\neq\varepsilon$
- or $S\rightarrow\varepsilon$.

If $S\rightarrow\varepsilon$, then S cannot appear in the righthand side of rules in R.

The language of a type-1 grammar is accepted by a **linear bounded automaton** (a nondeterministic Turing machine whose tape is bounded by a constant times the length of the input).

Type-2 Grammar: context-free

A grammar $G = (\Phi,\Sigma,R,S)$ is **context-free**, iff all production rules in R obey the form $A \rightarrow \alpha$ with $A \in \Phi$, $\alpha \in (\Phi \cup \Sigma)^*$.

Context-free languages are closed under the following operations:

- Union: if F and G are context-free, so is $F \cup G$
- Concatenation: if F and G are context-free, so is $F \bullet G$
- Kleene Star: if G is context-free, so is G^*

Context-free languages are not closed under the following operations:

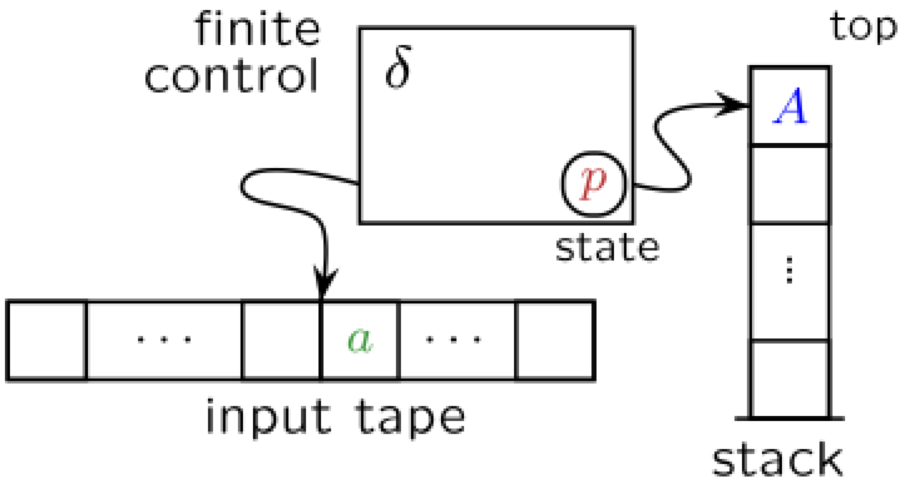
- Intersection: from F and G are context-free does not follow that $F \cap G$ *is.*
- Complement: from G context-free does not follow that $\neg G$ is.
- Difference: from F and G are context-free does not follow that $F \setminus G$ *is.*

Pushdown Automaton (=Kellerautomat)

Context-free grammars are accepted by **non-deterministic pushdown automata**.

A non-deterministic push-down automaton PDA=($\Phi,\Sigma,\Delta,\square,\delta,S,F$) consists of

- Alphabet Φ of states
- Alphabet Σ of input symbols, disjunct with Φ
- Alphabet Δ of stack symbols, disjunct with Φ
- initial stack symbol □
- transition relaton δ: $\Phi\times\Sigma\times(\Delta\cup\square)\rightarrow\rho(\Phi\times\Delta^*)$
- start state $S\in\Phi$
- set of final states $F\subset\Phi$



Type-3 Grammar: regular, left/right linear

A grammar $G = (\Phi,\Sigma,R,S)$ is **right linear (left linear)**, iff all the production rules in *R* obey the forms

- $A \rightarrow w$
- $A \rightarrow wB$ (left linear: $A \rightarrow Bw$)

with $A,B \in \Phi$ and $w \in \Sigma^*$.

Left (right) linear grammars generate **regular** languages:

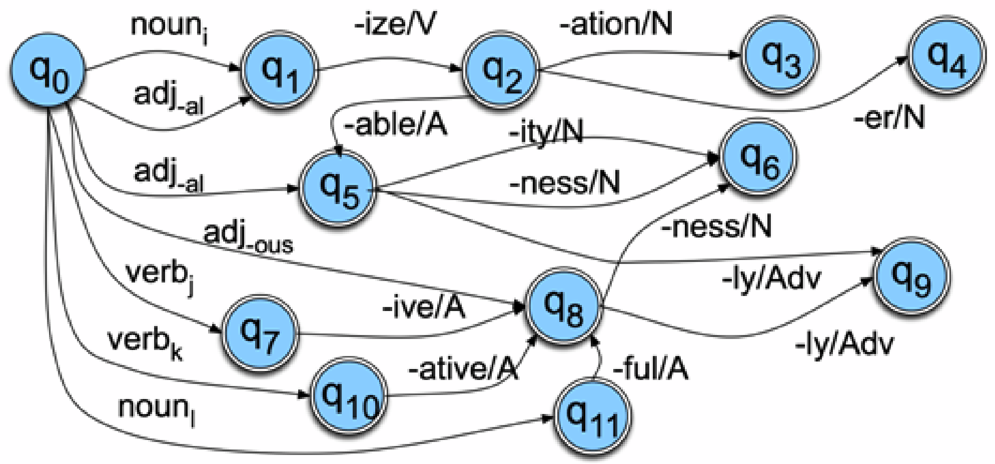
- $\varnothing$ is regular
- $\{a_i\}$ is regular for $a_i \in$ alphabet Σ
- if the sets L_1 and L_2 are regular, then $(L_1 \cap L_2)$ is regular
- if the sets L_1 and L_2 are regular, then $(L_1 \bullet L_2)$ is regular
- if set L is regular, then also L^*.

These languages can be described through **regular expressions**

... and are accepted by **finite state automata**

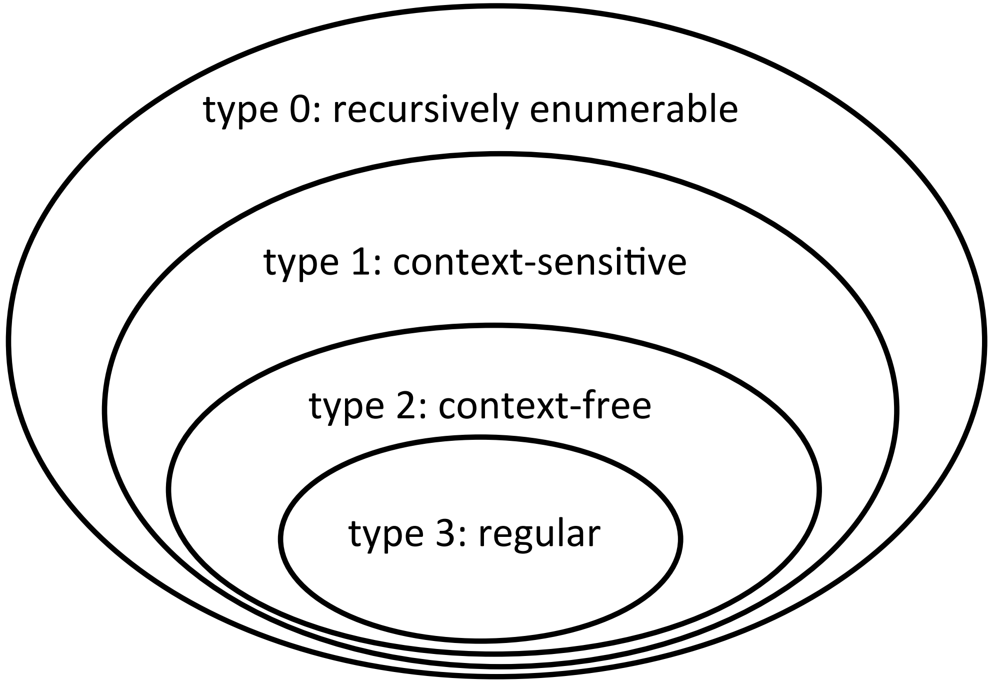
Finite State Automaton

- an FSA is a 5-tupel FSA=(Φ,Σ,δ,S,F) consisting of:
  - a set of states Φ
  - input alphabet Σ, Φ∩Σ=∅
  - one start state S⊂Φ
  - set of final states F⊂Φ
  - transition function δ⊆Φ×(Σ∪{ϵ})→Φ
    - with ϵ being the "spontaneous transition" that doesn't consume a symbol



OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

The Chomsky hierarchy of formal languages



Turing Machine (TM)

linearly bounded TM

Pushdown Automaton (PDA)

Finite State Automaton (FSA)

- The different classes are proper subsets of each other: the expressivity of type-(n) grammars is truly smaller than type-(n-1) grammars.
- Several other classes are known, e.g. corresponding to deterministic context-free grammars, tree adjoining grammars …

Principle of Compositionality

- Speech sounds (or letters) form words
- Words follow each other, forming sequences
- sequences of words have an inner structure (syntax)
- meaning (semantics) can be derived from the structure

Means of Compositionality

- differencces across linguistic layers
  - morphology is (mostly) regular
  - syntax uses (mainly context free) grammars
  - semantics can be expressed through first order logics or relation graphs

Principle of Compositionality

- Speech sounds (or letters) form words
  - there are language-specific rules for how speech sounds can be combined (=phonotactics)
  - German allows consonant clusters like in “Strumpf” /ʃ t r ʊ m p f/, where most languages are less liberal.
  - Then again, German sometimes requires “ch” to be /ç/ as in “Nächte” or /x/ as in “Nacht”. Try saying this the opposite way, it will feel weird.
  - can you deduce the rule for when “ch” is /ç/ vs. /x/?

finite state-based techniques in Natural Language Processing:

1. Morphological analysis
2. Grapheme-phoneme conversion

Morphology

- words have internal structure
  - base form

    + morphological properties
  - “largest” → large+Adj+Sup
  - “better” → good+Adj+Comp (exceptions from rules)
  - “runs” → run+Verb+Present+3sg
    OR run+Noun+Pl (ambiguity)

Morphology

- rules are (by and large) regular, thus, a finite state automaton (FSA) is appropriate

  (wanna crack a nut? → use only a small sledgehammer)

- task for morphologic models
  - given a full word, decompose it into morphemes
  - given base form+morphemic information, generate the resulting “surface structure”

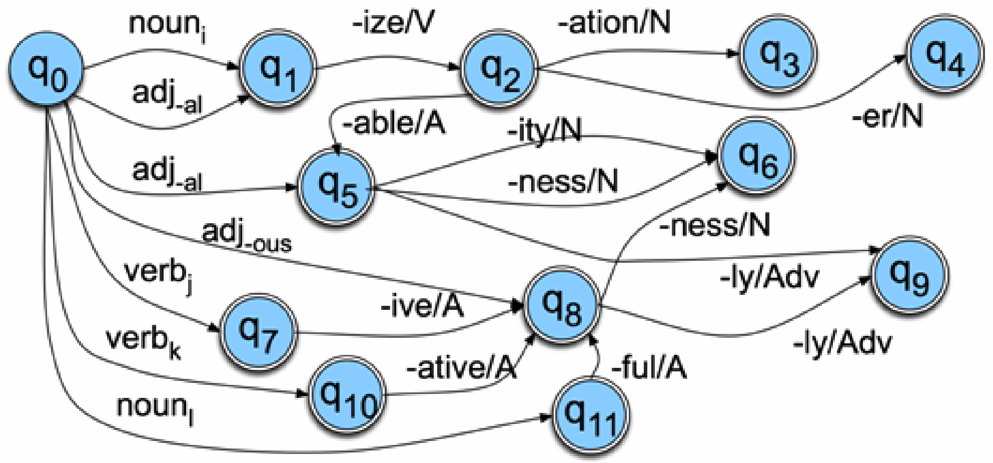
Why build a model at all?

We could simply store all word forms in a table.

What can be advantages of a morphological model?

Finite State Transducer

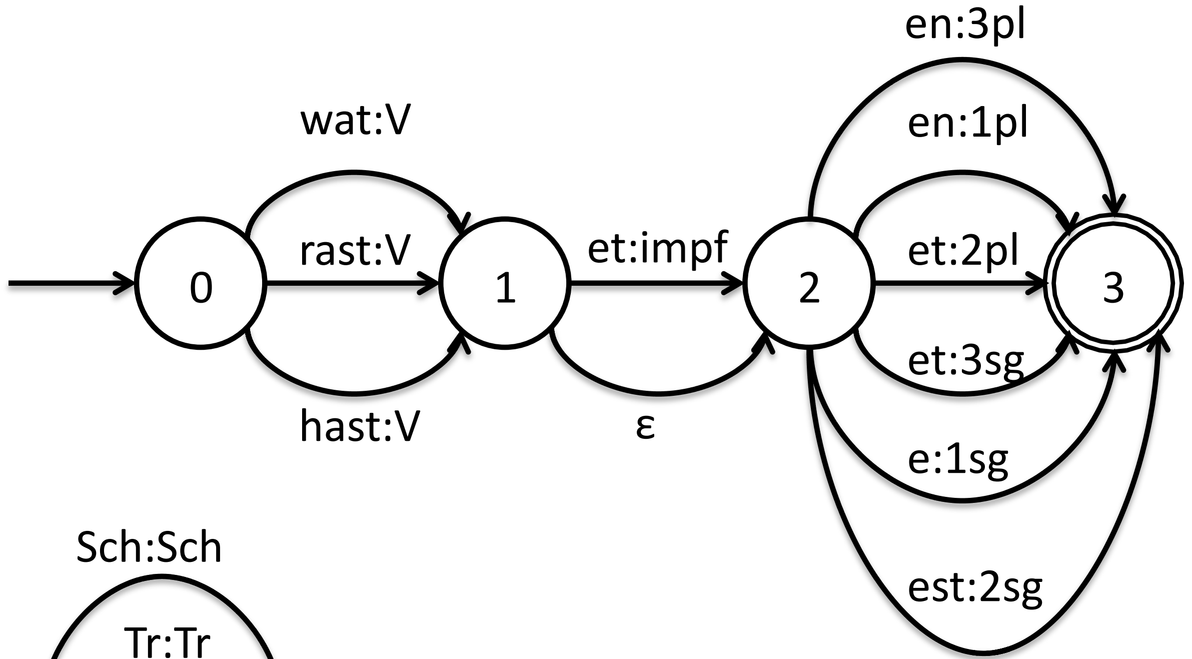
- a finite state **transducer** is a 6-tupel FST=(Φ,Σ,Γ,δ,S,F) consisting of:
  - einer Menge Zustände Φ
  - Eingabealphabet Σ
  - Ausgabealphabet Γ
  - Startzuständen S⊂Φ
  - Terminalzuständen F⊂Φ
  - Übergangsfunktion $\delta \subseteq \Phi \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \times \Phi$
    - with ϵ being the "spontaneous transition"

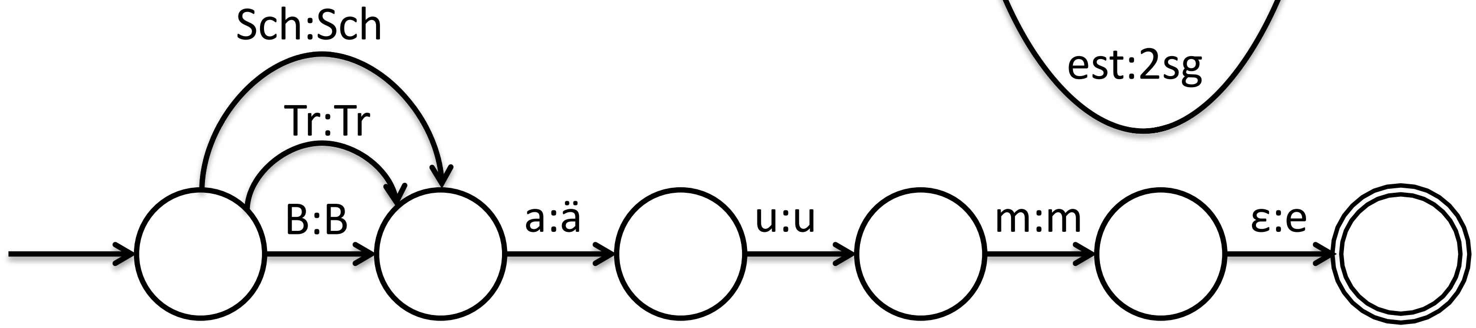


FSTs

- FSTs correspond to a finite state automaton but have two alphabets (Σ and Γ), which are used on two different “tapes” (for input and output)
- an FST transduces an input string x to an output string y if there is a sequence of transitions that starts in the start state, ends in a final state and has x as its input and y as its output string
- FSTs can be composed from simpler FSTs just like FSAs (cmp. regular expressions)

examples of FSTs (for morphology)

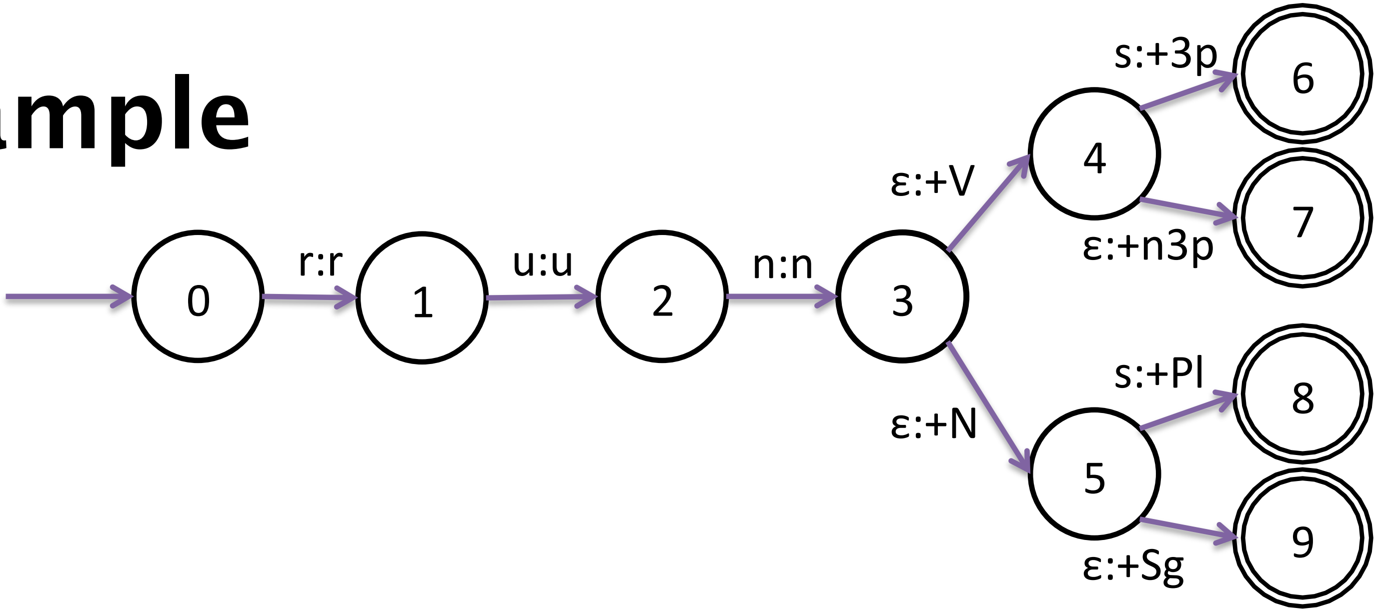




Dealing with Ambiguity through Nondeterminism

- language is ambiguous (on most levels); we embrace nondeterminism as a mechanism to reflect this
- as long as we can't resolve ambiguity, we carry along multiple hypotheses in parallel
  - for FSA: we don't know what path we've taken
  - for FST: different paths produce different output strings
- we keep a **set** of possible current states
- acceptance of FST if there's at least one path to a final state

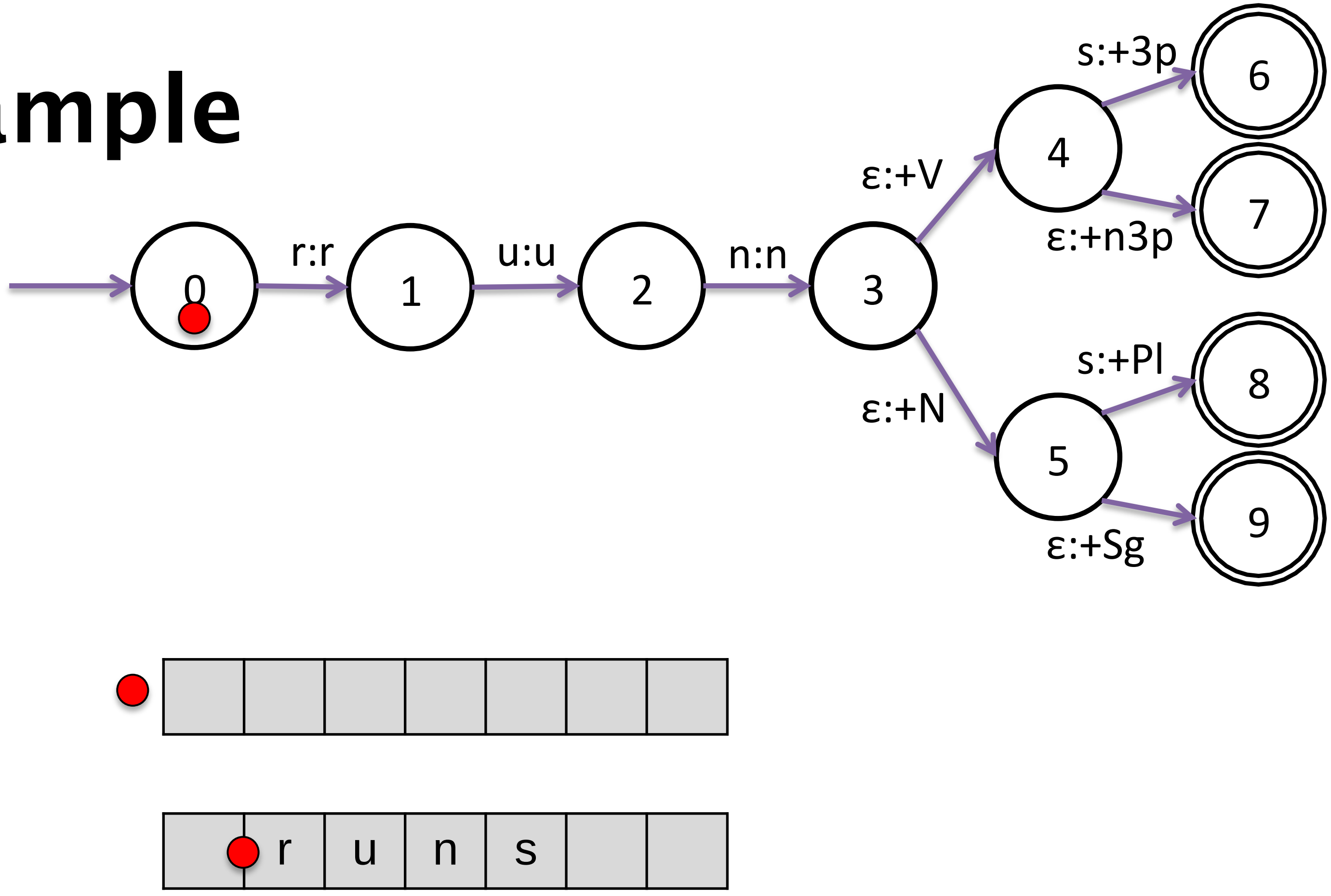
Example



input string

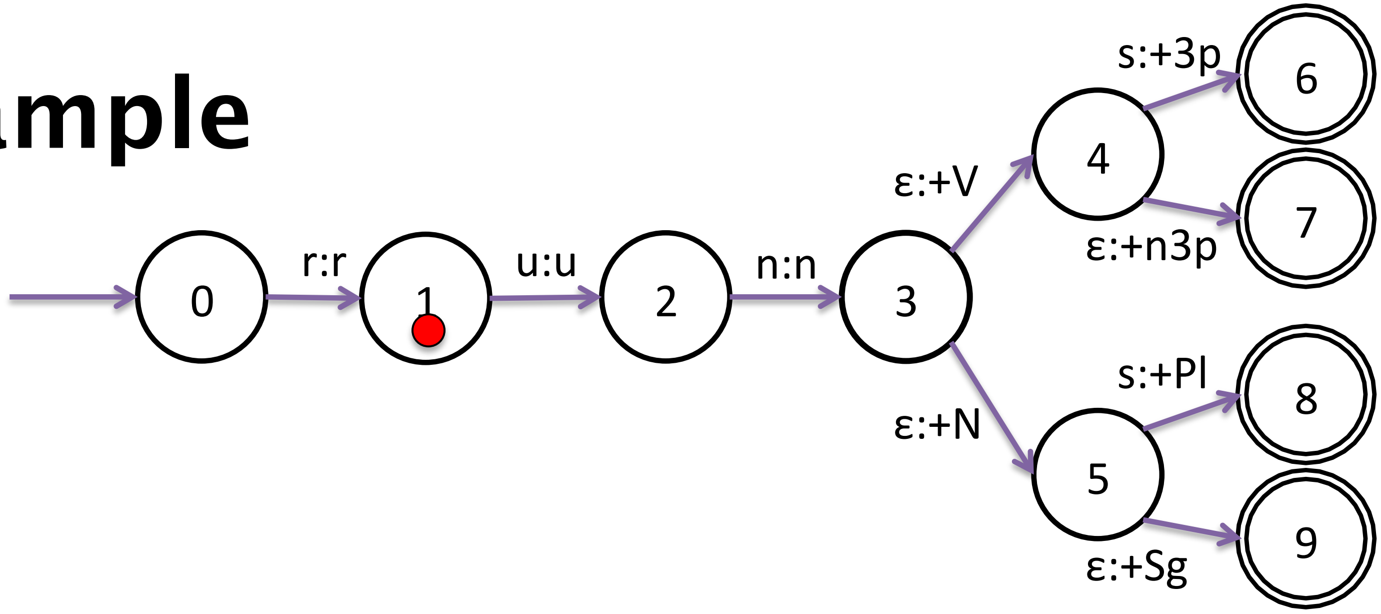
	r	u	n	s		

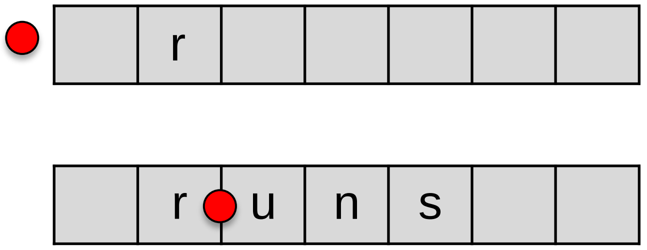
Example



Dots: Keep track of current state and output generated so far.

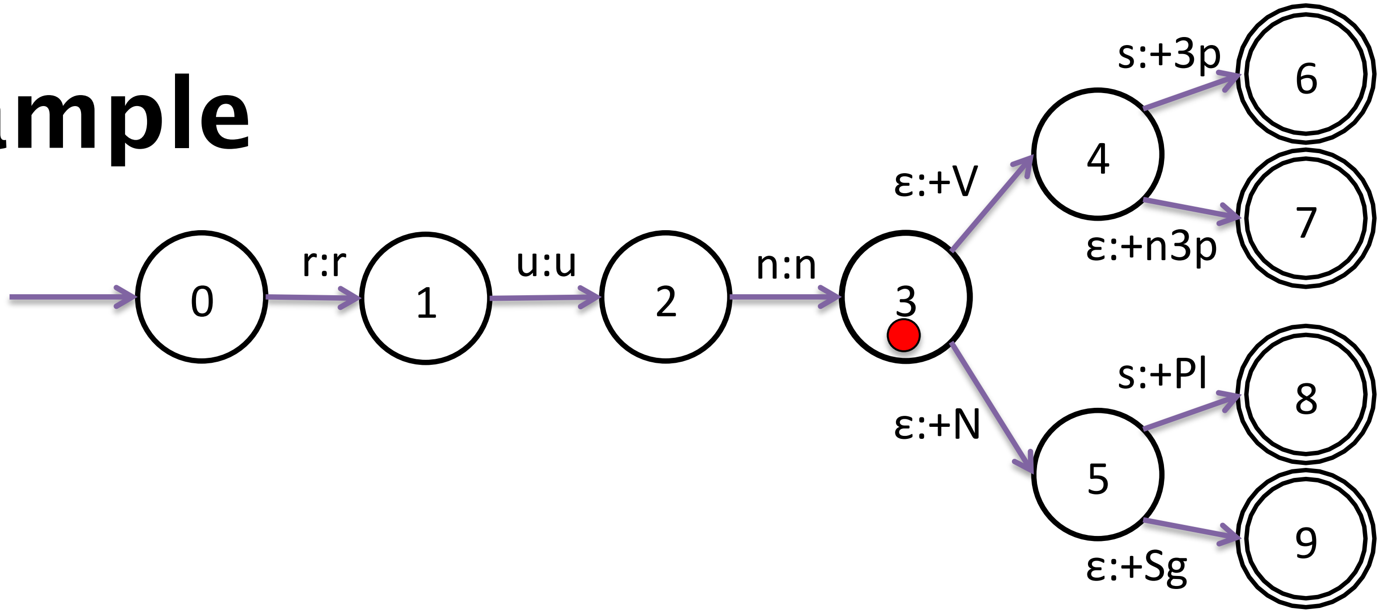
Example

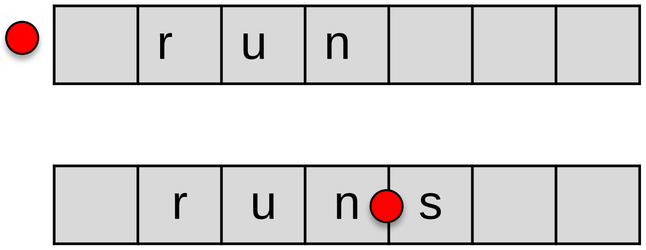




Transition: dot moves on input tape and to next state, generating output.

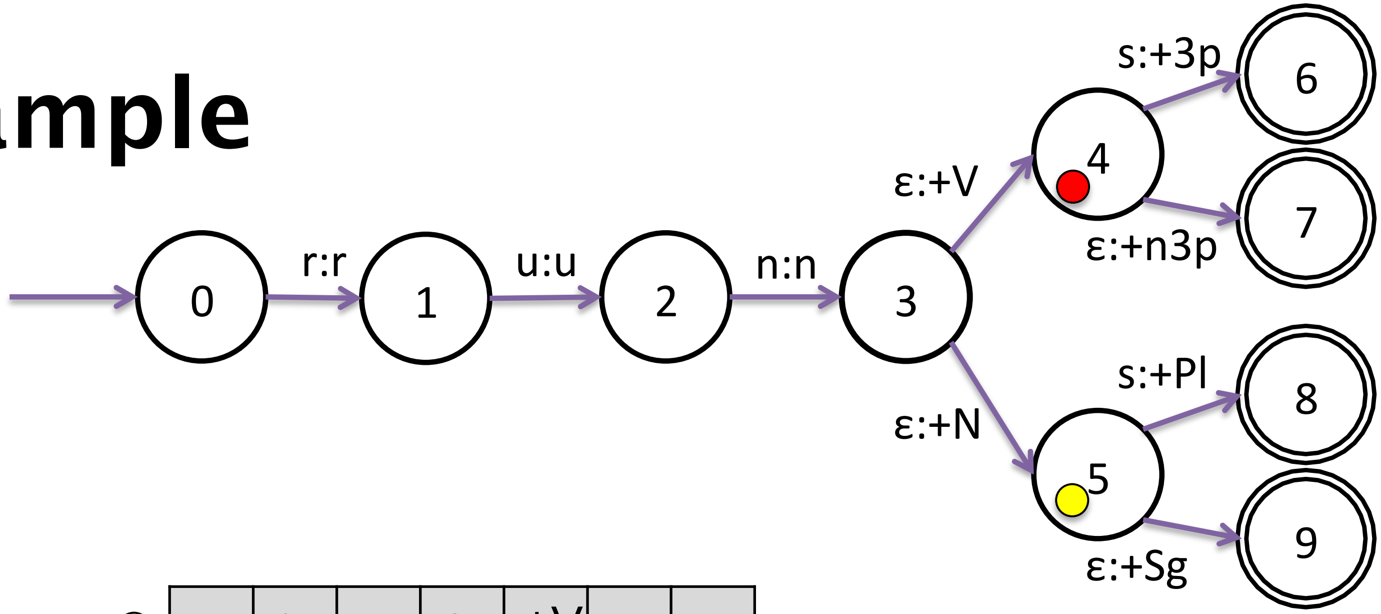
Example

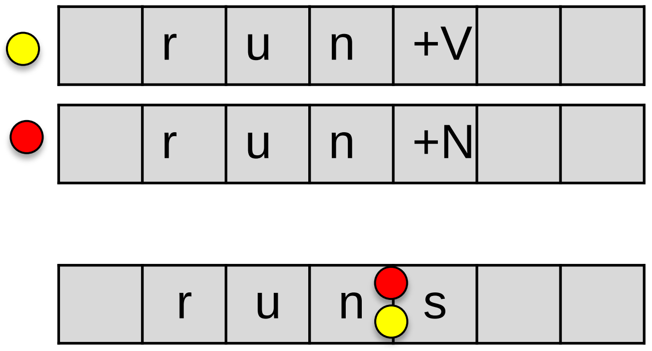




Transition: dot moves on input tape and to next state, generating output.

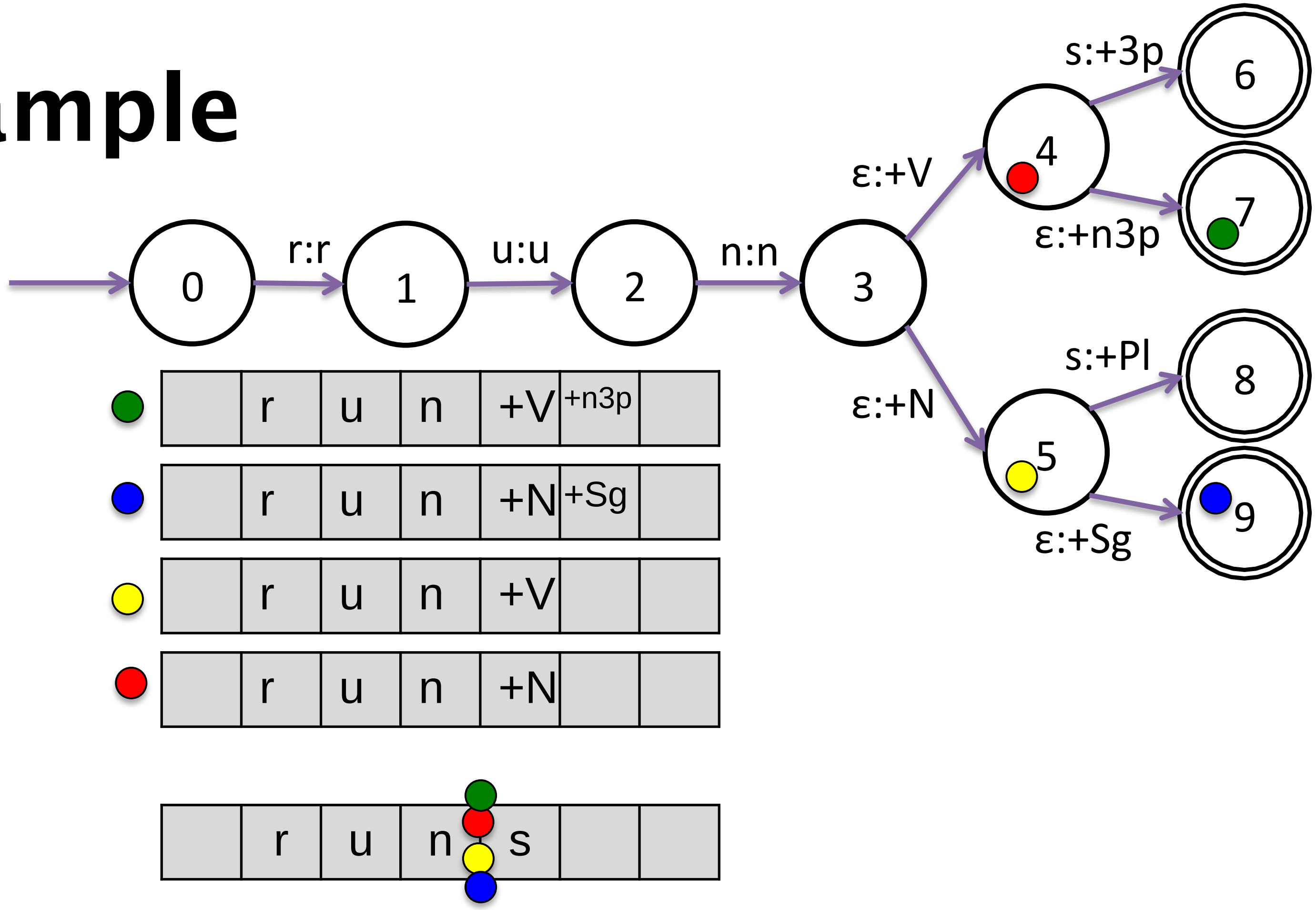
Example





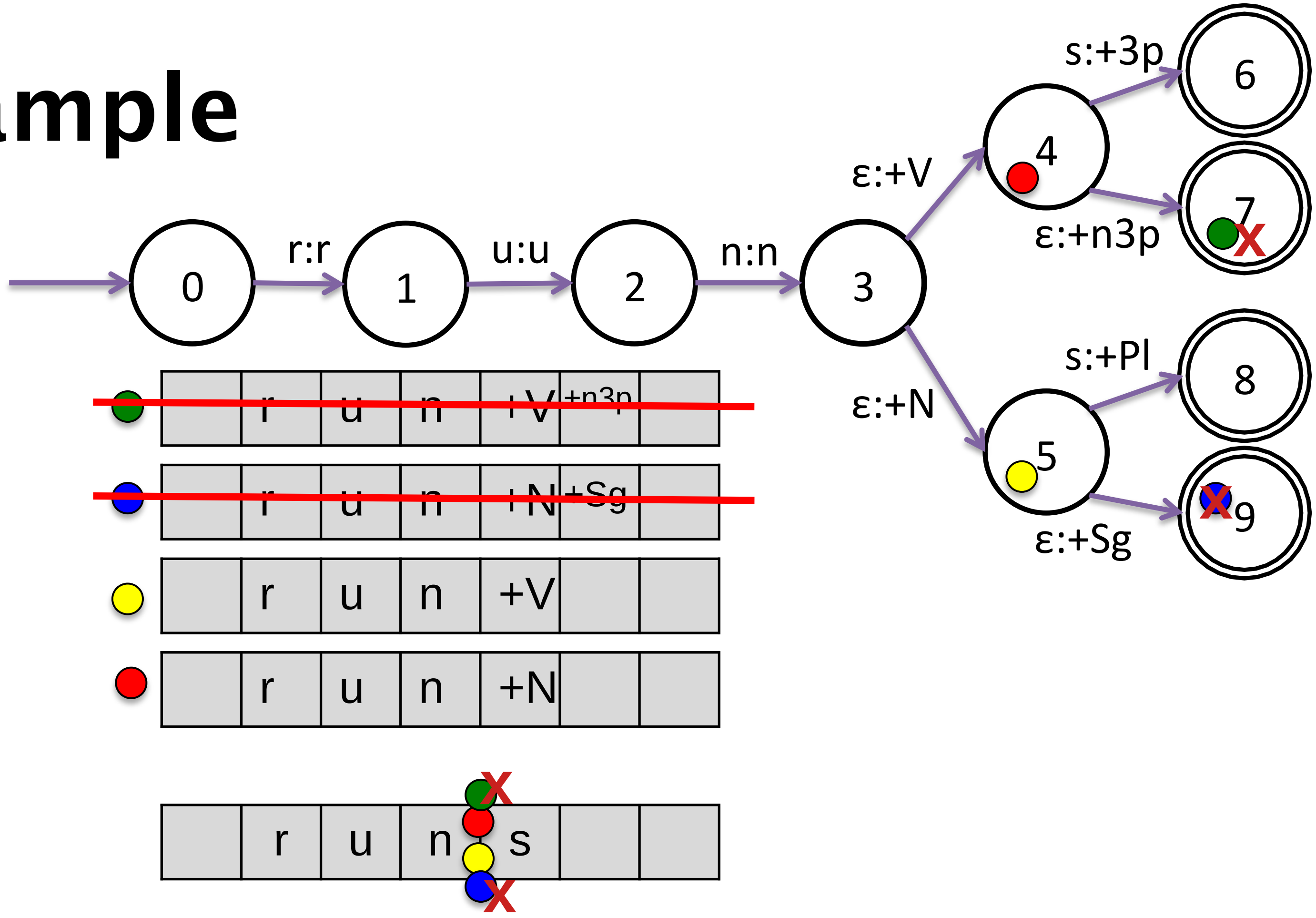
Non-determinism: Dot splits. Two possibilities for output tape.

Example



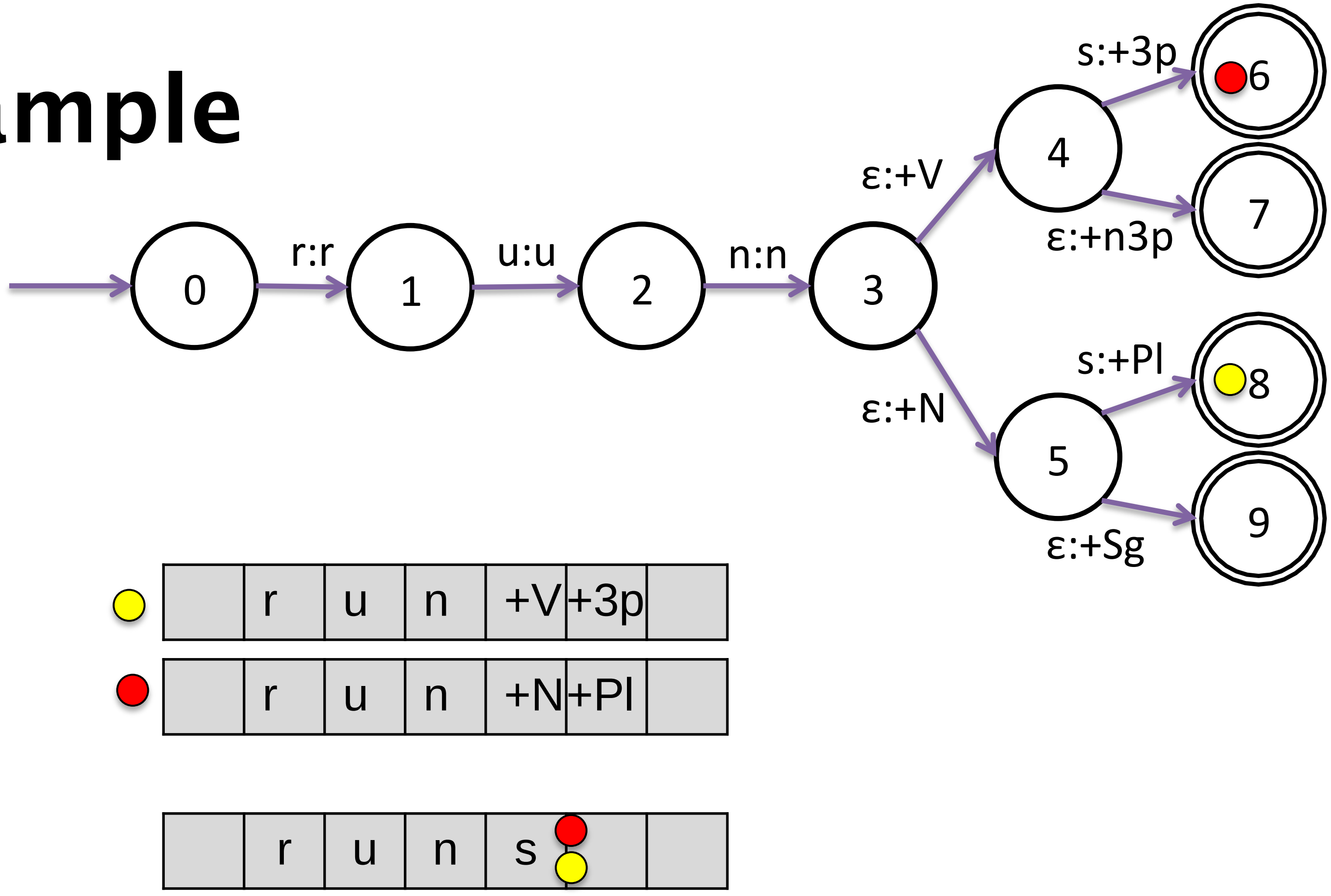
ϵ–transitions also introduce nondeterminism.

Example



Dots that do not have a follow-up state are removed as they form no valid hypothesis.

Example



End of input is reached. All dots at final states have successfully transduced.

Summary: FST

- given: some input sequence
- produces: some output sequence
  that is in a regular relation to input sequence
- non-deterministic algorithm to check input/output correspondence
  - create all possible outputs given the input
  - typically: weighted FST with beam search (→AGKI)
- FSTs can be composed & stacked together

Applications of FSAs/FSTs in NLP

- data structure for lexica (→ spell checking)
- morphological analysis & synthesis
- Segmentation
- Tokenisation
- Sentence boundary detection
  (not every full stop is a sentence boundary)
- Chunk parsing (through cascaded FSTs)
- decoding in speech recognition
- pronunciation modelling (exercise on Friday)

FSTs for spelling

- input alphabet: standard alphabet
- output alphabet: International Phonetic Alphabet, SAMPA, or some other representation for speech sounds
  1. start with some simple rules:
     - “s” → /s/
     - “sch” → /ʃ/
     - “st” → /ʃ t/
     - ...
  2. merge these together to form a complex FST
  - alternative: train from data

FSTs for spelling

$$\begin{tikzcd}
 & & \text{s\_} & & \\
\text{start} \arrow[r, "\varepsilon:/"] & \text{\_s} \arrow[ur, "s:s/"] \arrow[r, "s:\varepsilon"] \arrow[dr, "s:\varepsilon"'] & \text{sc} \arrow[r, "c:\varepsilon"] & \text{ch} \arrow[r, "h:\int/"] & \text{h\_} \\
 & & \text{st} \arrow[r, "t:\int t/"] & \text{t\_} &
\end{tikzcd}$$

Thank you, your questions?

`timo.baumann@oth-regensburg.de`

further reading

- Jurafsky and Martin (2009): chapter 3.2
- Carstensen et al. (2004): pp. 198–205

Lehrziele

- die Studierenden kennen automatenbasierte Verfahren in der Sprachverarbeitung
- die Studierenden verstehen den Vorteil von Modellen die sowohl zur Analyse als auch zur Generierung angewendet werden können
- die Studierenden kennen einen Algorithmus zur Dekodierung eines Automaten und können diesen anwenden
- die Studierenden kennen Anwendungen automatenbasierter Verfahren, wie z.B. morphologische Analyse und Aussprachemodellierung