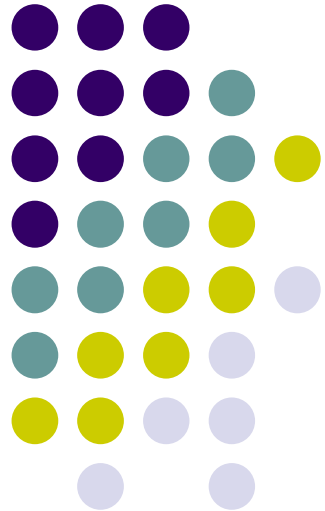
Finite-State Automata & Recursive Transition Network

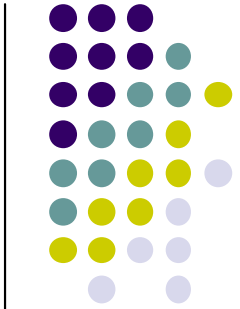
J. Savoy
Université de Neuchâtel



R. Mitkov (Ed): *The Oxford Handbook of Computational Linguistics*. Oxford University Press, Oxford, 2005.

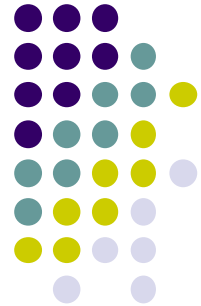
G. Gaznar & C. Mellish: *Natural Language Processing in PROLOG: An Intorduction to Computational Linguistics*. Addison-Wesley, 1989.

Purpose



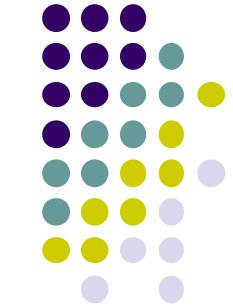
- Finite-State Automata (FSA)
- Very simple automaton
- Efficient / effective
- Use to recognize or to generate (an answer)
- Applications in various sublanguages
- A first step (and a first model)

Basic Notation



- Language: set of *strings* of any kind
- String: concatenation of zero (null string) or more *symbols*
- Symbols: single char (e.g., *a*) or multi-char (e.g., +*noun*)
  empty string (ε or *0*) → atomic entity
- Relation: a set of *ordered* string pairs e.g., {(a,bb), (cd,ε)}.
  The first member is the upper string (domain)
  The second member is the lower string (range)
- Identity relation: {(a,a), (c,c), …}.

Basic Notation



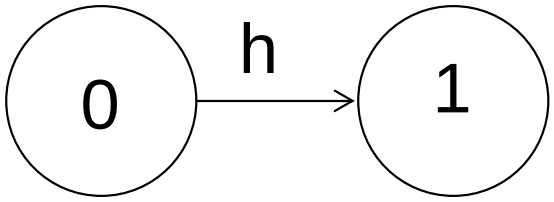
- Network: a finite-state automata or *network* *directed* graphs composed of states and arcs
- A single *initial state* (start state)
- Any number of *final states*
- Arc: may be *labeled* either by a single symbol (a) or a symbol pair (a:b) (or e.g., (car:voiture))
- Path: a *sequence* of arcs from the start state to a final state

Example

A finite-state automata to recognize the laughing language (ha!, haha!, hahaha!, etc.)

Where are the states?

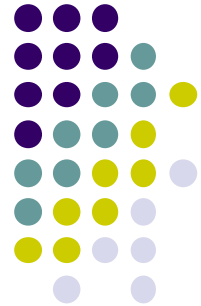
Where are the arcs? And their labels?



The initial state will be placed on the left.

When we're processing a string, we remove the front symbol corresponding to the label on the arc.

FSA Processing



If starting in the initial state, we reach a final state and the input string is empty at this point, we have found a *path* and the entry string is valid.

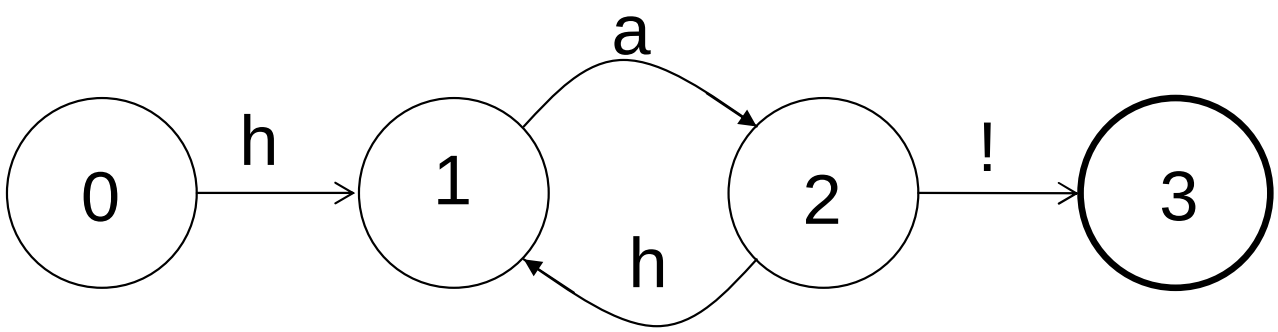
If we cannot reach a final state, the input string is not valid (well-formed formula, *wff*).

If we have a choice, we can come back (backtracking) and try the alternative way later.

Example

Example: A finite-state automaton to recognize the laughing language (ha!, haha!, hahaha!, etc.)

If you find a path, you find a valid string

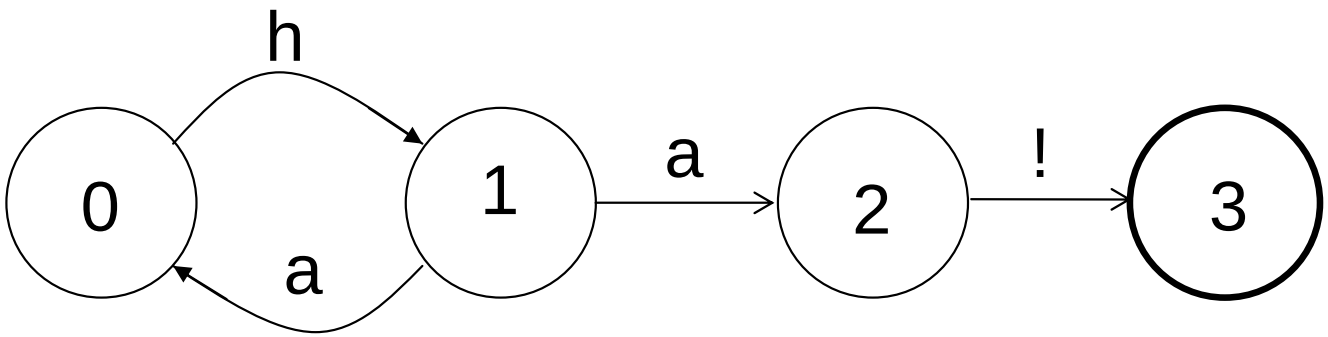


The final state will be placed on the right (in bold).

At any state, you have no real choice (deterministic automaton). Is this the single solution?

Deterministic & non-deterministic

Example of a non-deterministic automaton



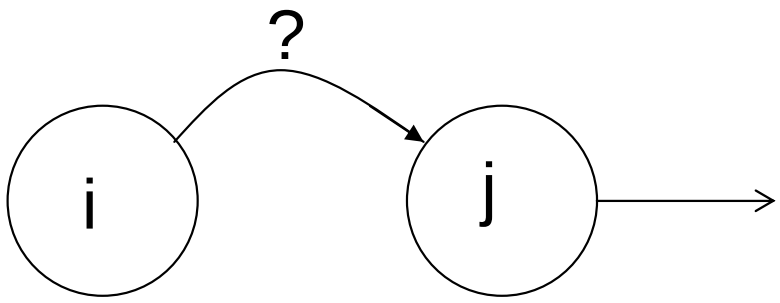
Do we recognize the same strings?

Why non-deterministic?

More Complex Transition

Draw a FSA to accept from state i to state j …

a) the symbol "*a*"
b) any symbol
c) the symbols "*ed*"
d) with, at least, once the symbol "*b*"
e) with 0, 1, or *n* occurrences of the symbol "*c*"



We can define a metachar (e.g., *)

More Complex Examples

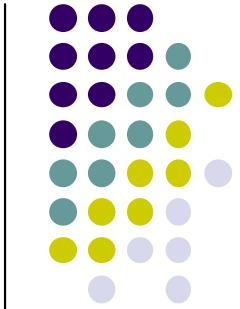
Draw the FSA to accept numbers…

a) Real numbers (e.g., 123.45)
b) With the scientific notations (e.g., 3.14E+02)
c) A simple English grammar

Hint: Replace a set of symbols by a LABEL

LABEL: set of symbols

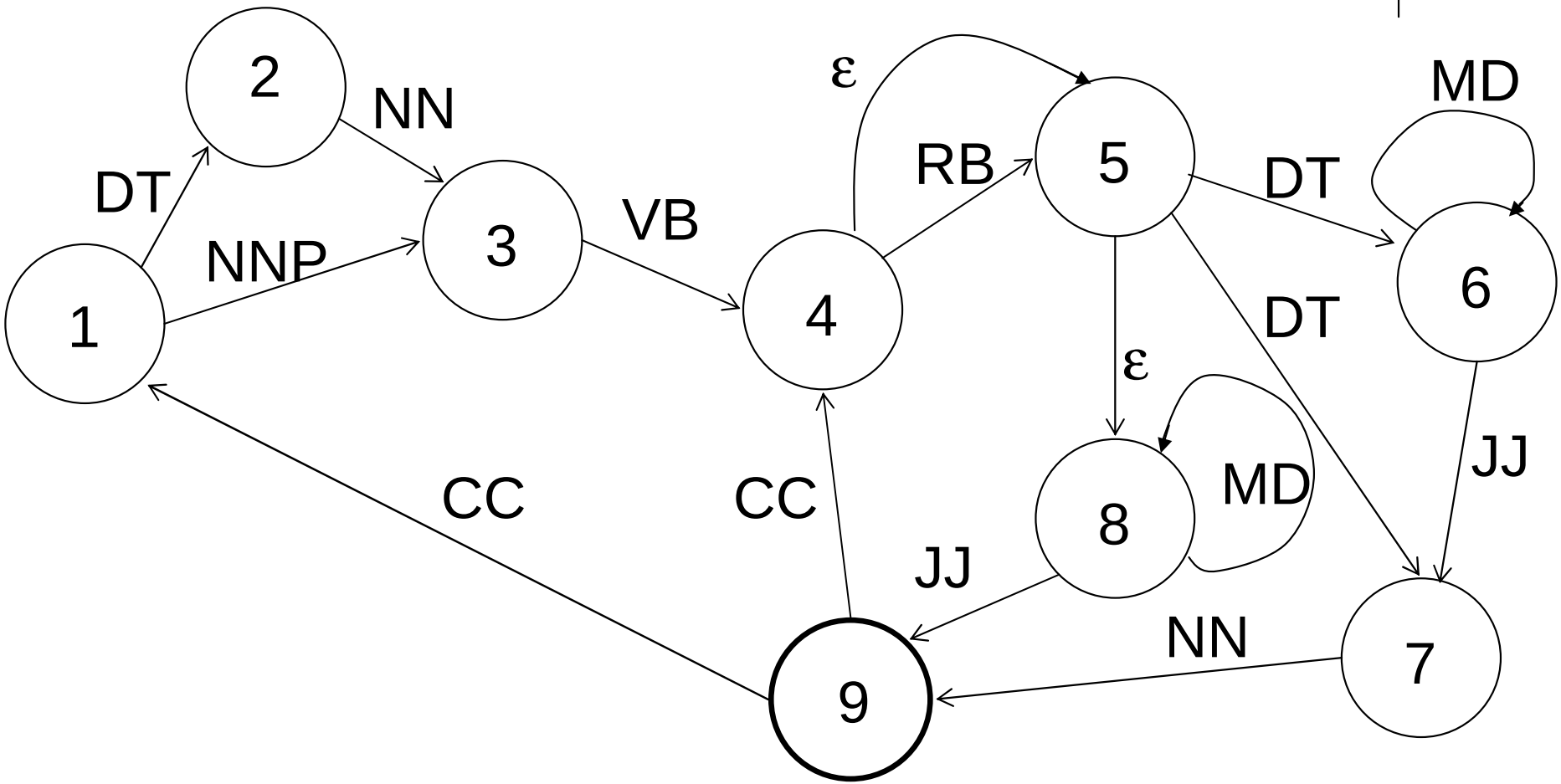
Simple Grammar



We have the following lexicon:
NNP (proper noun): Kim, Mary, Ann.
DT (determiner): a, the, her.
NN (noun): consumer, man, woman.
VB (verb): is, was
JJ (adjective): happy, stupid.
MD (modal): very
RB (adverb): often, always, sometimes.
CC (conjunction): and, or.

(We have used the Penn Treebank POS tags)

Simple Grammar



Simple Grammar

- Verify your grammar with the following string:

a. Kim is happy

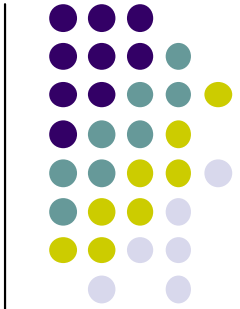
b. Ann is often a consumer

c. Mary is a happy consumer and Ann is happy

d. the consumer is very happy

- What are the strings you can generate?
- Can you add the plural form?
- Can you transform the grammar to admit also interrogative string?

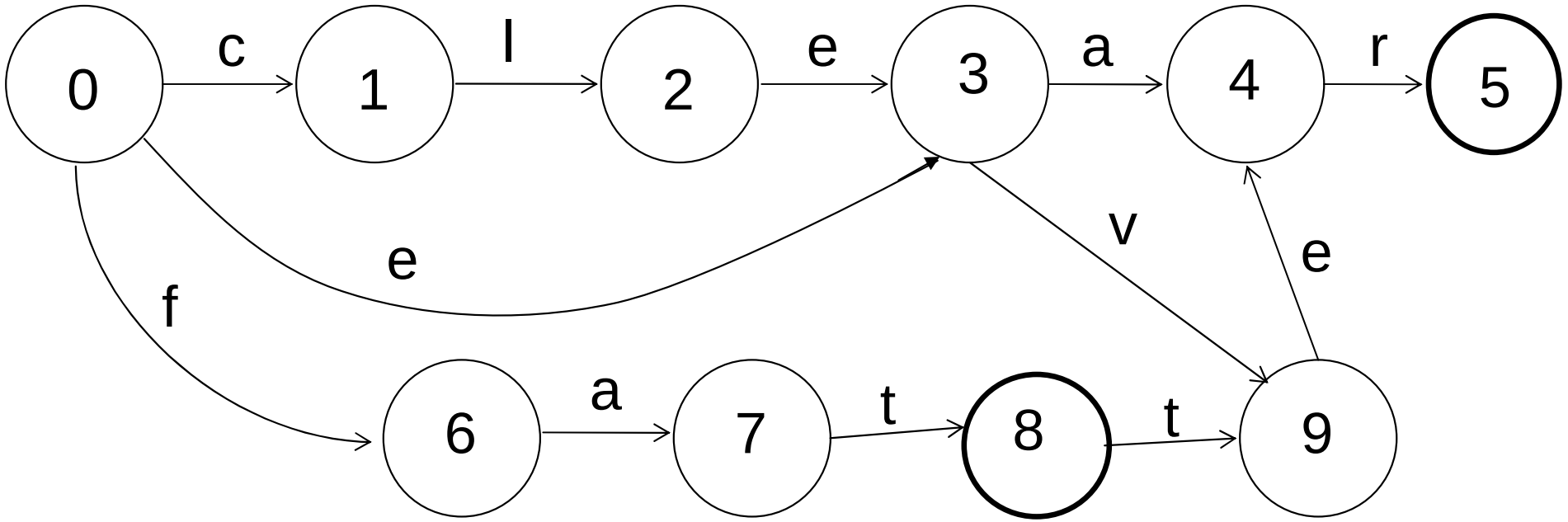
Lexical Analysis



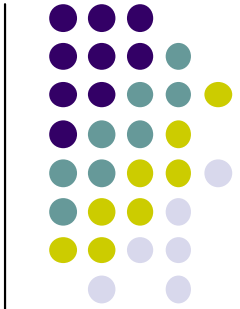
- Other application: Recognize a set of words belonging to a sublanguage.
- Build a finite-state automaton that will recognize the following words
  - Clear
  - Clever
  - Ear
  - Ever
  - Fat
  - Fatter

Example

FSA for the lexical analysis



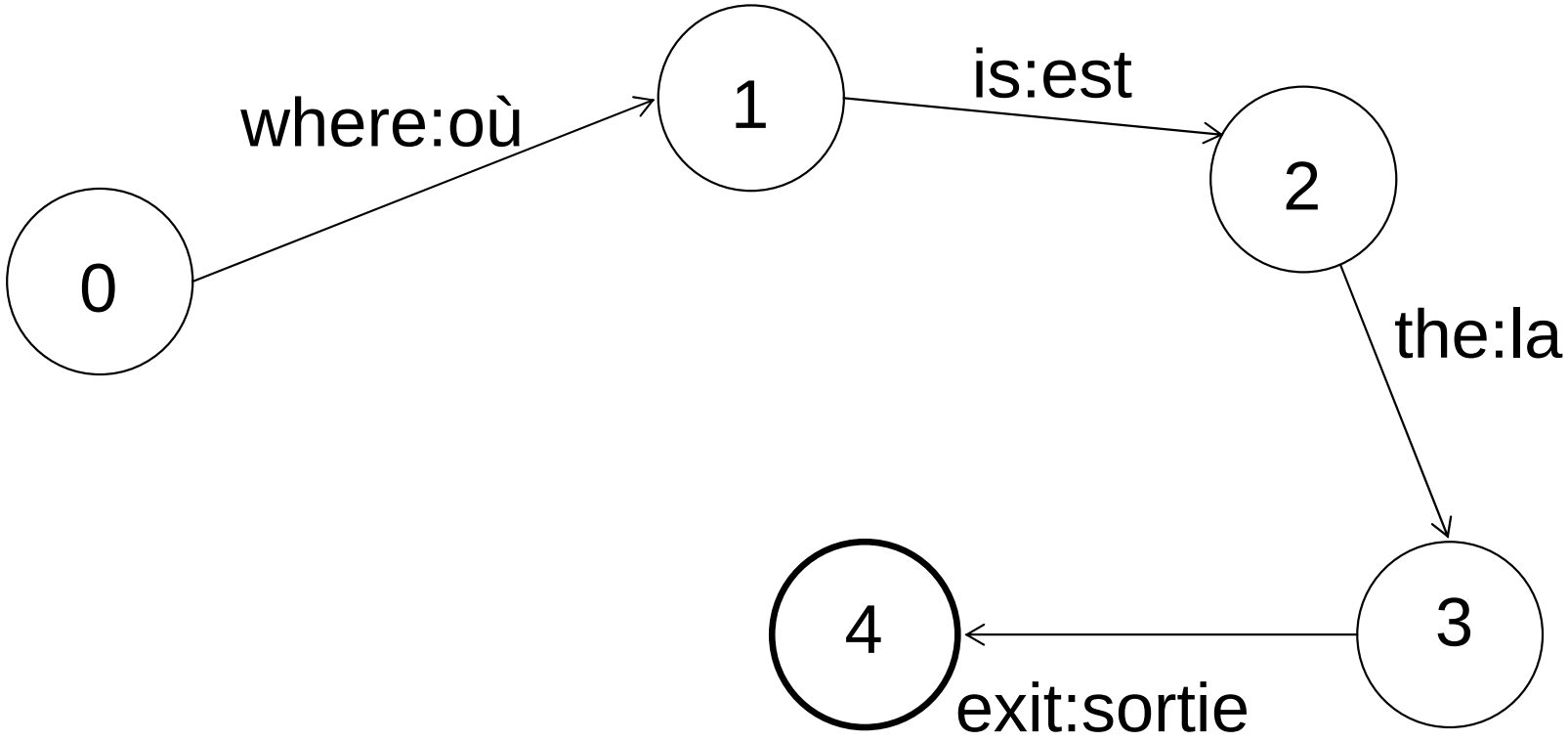
Transducers



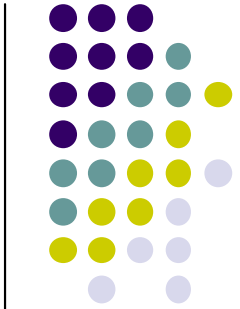
- Instead of verifying the string (valid or not), we may transform it (more precisely, return a new output string if the input string is valid).
- Use the relation definition (e.g., {(a,bb), (cd,ε)}) between states (but one could be the empty symbol ε).
- We may change our name from Finite-State Automaton to Finite-State Transducer (FST).
- You need to propose a FST to produce the translation for the string (where is the exit) into (où est la sortie).

Translation withTansducer

A simple translation example



Transducer

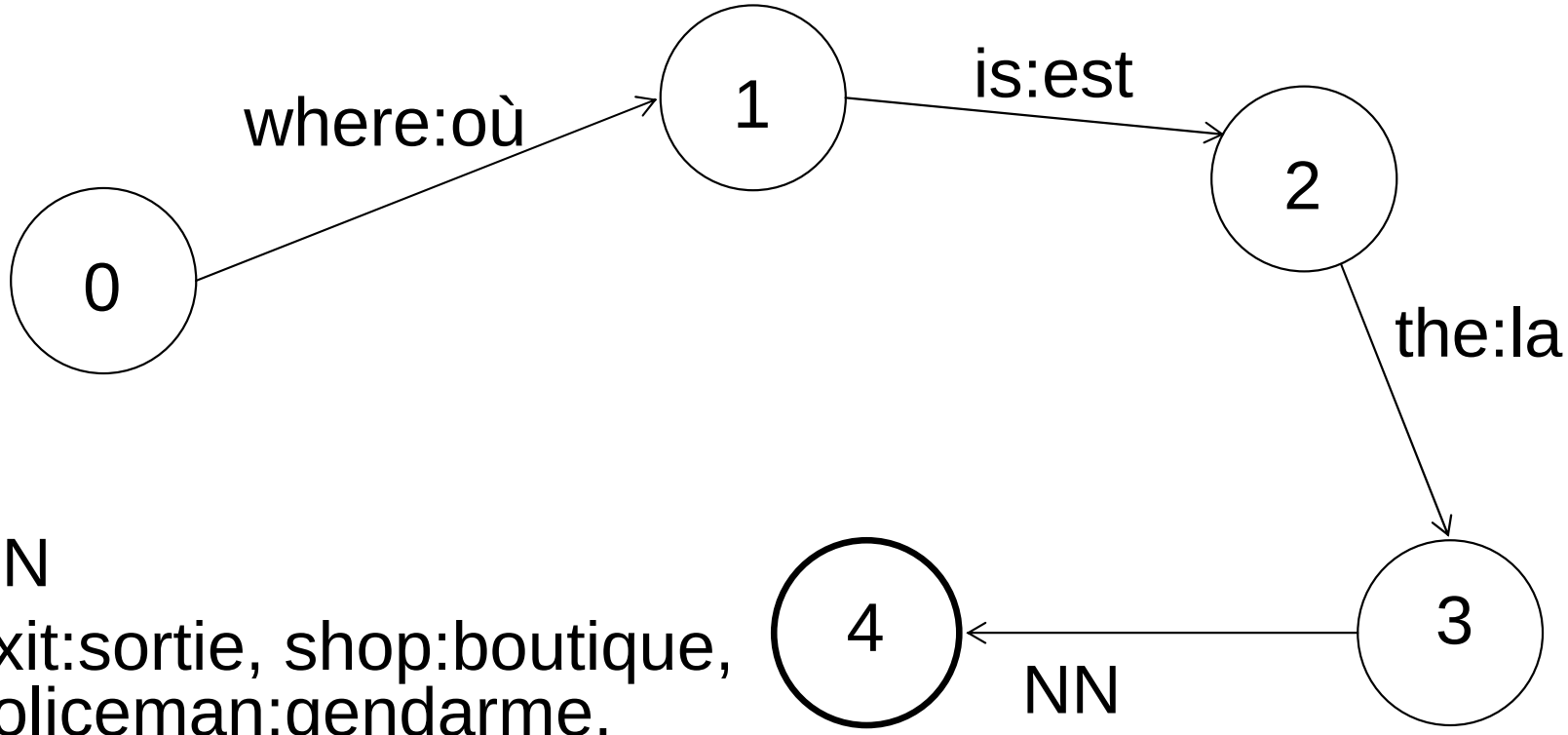


1. Generalize the previous example.

- We may replace a specific symbol pair by a set possible symbols (e.g, NN)
  Example
  NN
  exit:sortie, shop:boutique,
  policeman:gendarme, toilet:toilette

2. Can you take account for the gender (the:la) and (the:le)?

Translations with Transducer

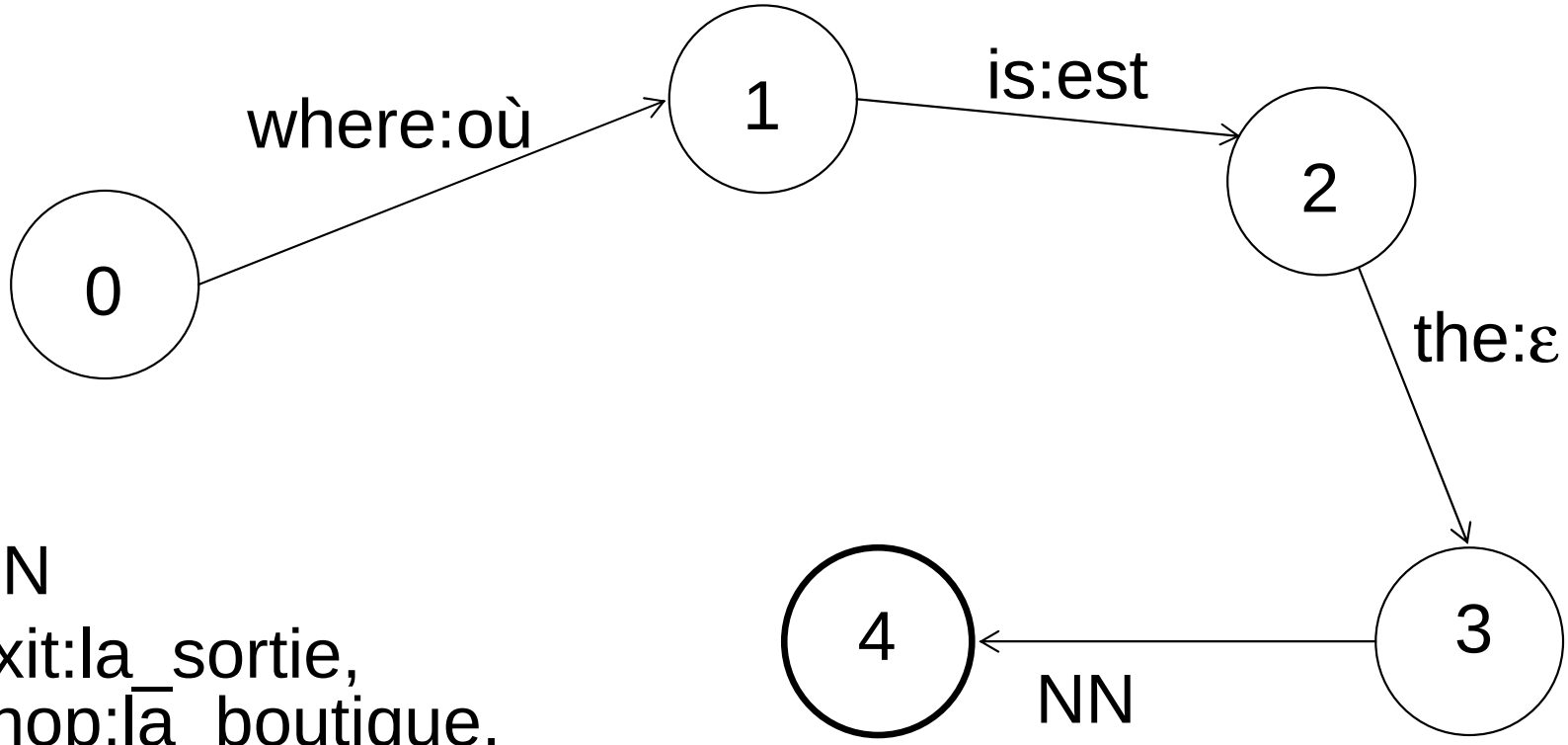


NN

exit:sortie, shop:boutique, policeman:gendarme, toilet:toilette

But “où est la gendarme”!

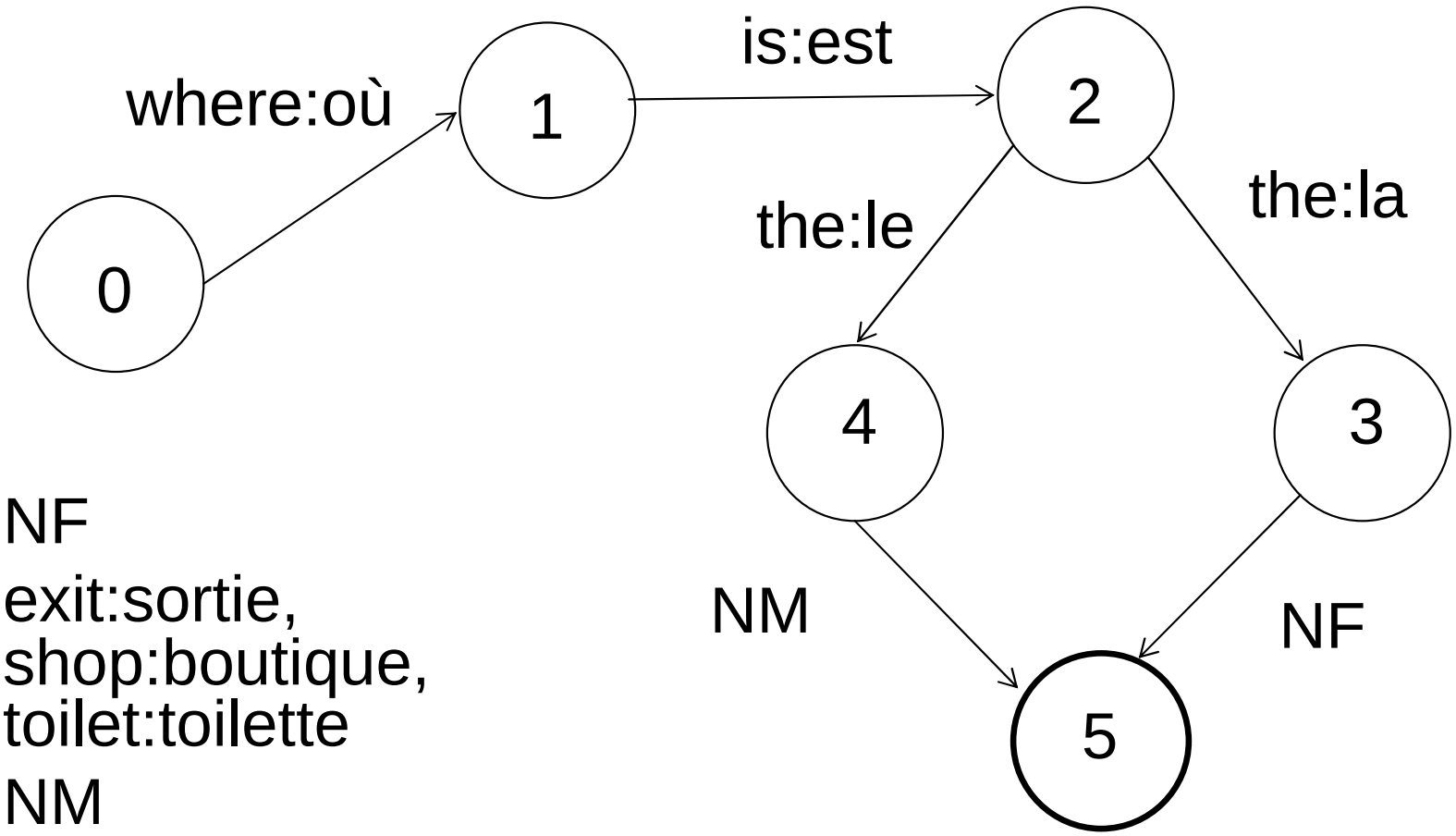
Translations with Gender



NN
exit:la_sortie,
shop:la_boutique,
policeman:le_gendarme,
toilet:la_toilette

Closed association between DT & NN!

Translations with Gender



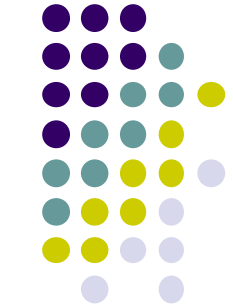
NF

exit:sortie,
shop:boutique,
toilet:toilette

NM

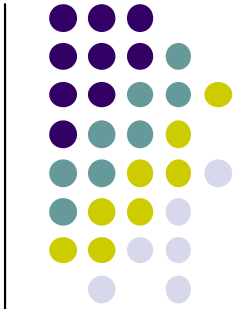
policeman:gendarme

Overview of FSA and FST



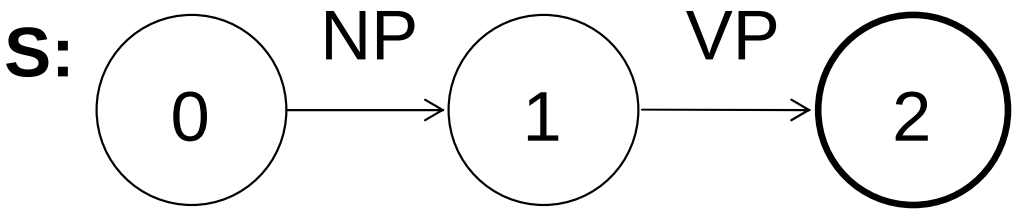
- The simplest approach to NLP
- Of very little use by themselves
- Map one string of symbols into another
- Can be used for sublanguage translation
- Can be used for morphological processing
- Are easy to implement
- Can handle expression like $a^n b^m$ but not $a^n b^n$
- But not sufficient for NLP

Recursive Transition Network

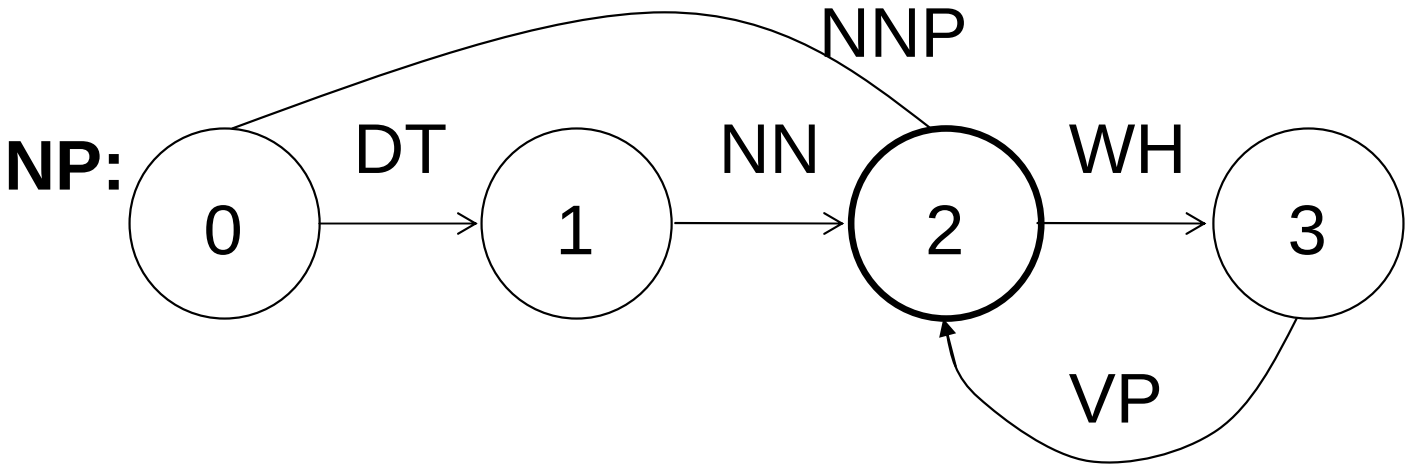


- We could name a network (for future use) and thus we may simplify the design process (we may reuse existing networks)
  (Recursive Transition Network, RTN)
- We have:
  NN: woman, house, table, mouse, man, genius, …
  NNP: Mary, John, Washington, Ben, …
  DT: a, the, that, …
  VB: sees, hits, sings, loves, saw, …
  WH: who, which, that, …
- A valid Sentence is a string with a NounPhrase follows by a VerbPhrase.

Recursive Transition Network

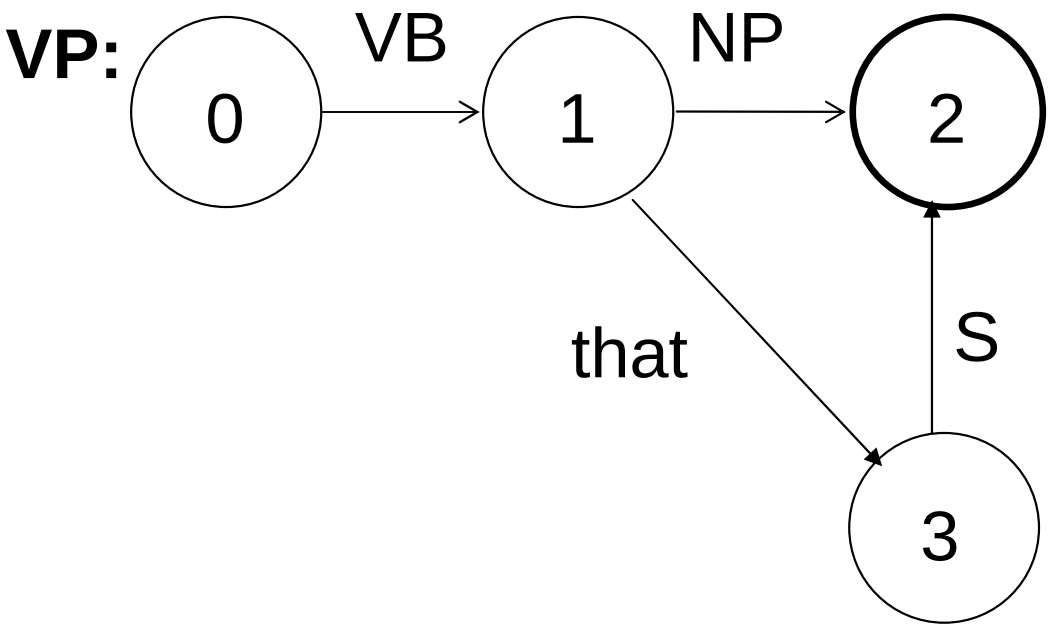


- A valid NounPhrase is a string composed of a determinant follows by a noun and possibly follows by a WH and VerbPhrase



Recursive Transition Network

- And for the VerbPhrase network (VP):

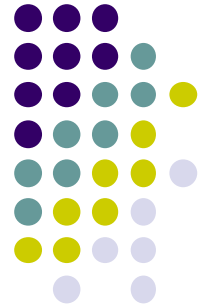


- Can you recognize the sentences:
  John sees the house, Mary loves John,
  John says that Mary says that Washington is a genius

Recursive Transition Network

- We clearly take account for recursive patterns in the language
- Very common in all natural languages
- But with limited use
- “The man who the woman sings sees Ben” .
  “The man who the woman who the boy plays sings hits Washington” .
- Useful to named a network
- Handle recursive expressions easily
  “The rapidity that the motion that the wing that the hummingbird has has has is remarkable”
- Can recognize expression such as $a^n b^n$ but not $a^n b^n c^n$

Exercice



- Using FSA
  can you represent the Latin morphology
  E.g., the first declension
  nominative, vocative, accusative, genitive, dative, ablative
  rosa, rosa, rosam, rosae, rosae, rosae;
  rosae, rosae, rosas, rosarum, rosis, rosis.
- Using the RTN method, can you recognize a grammar having the following rule
  S: a S b
  and that can generate ab, aabb, aaabbb, ..., $a^n b^n$