Computational Lexicology, Morphology and Syntax

Diana Trandabăț

Academic year 2022-2023

Today's Topics

- Finite State Technology
- Regular Languages and Relations
- Review of Set Theory
- Understand the mathematical operations that can be performed on such Languages.
- Understand how Languages, Relations, Regular Expressions, and Networks are interrelated.

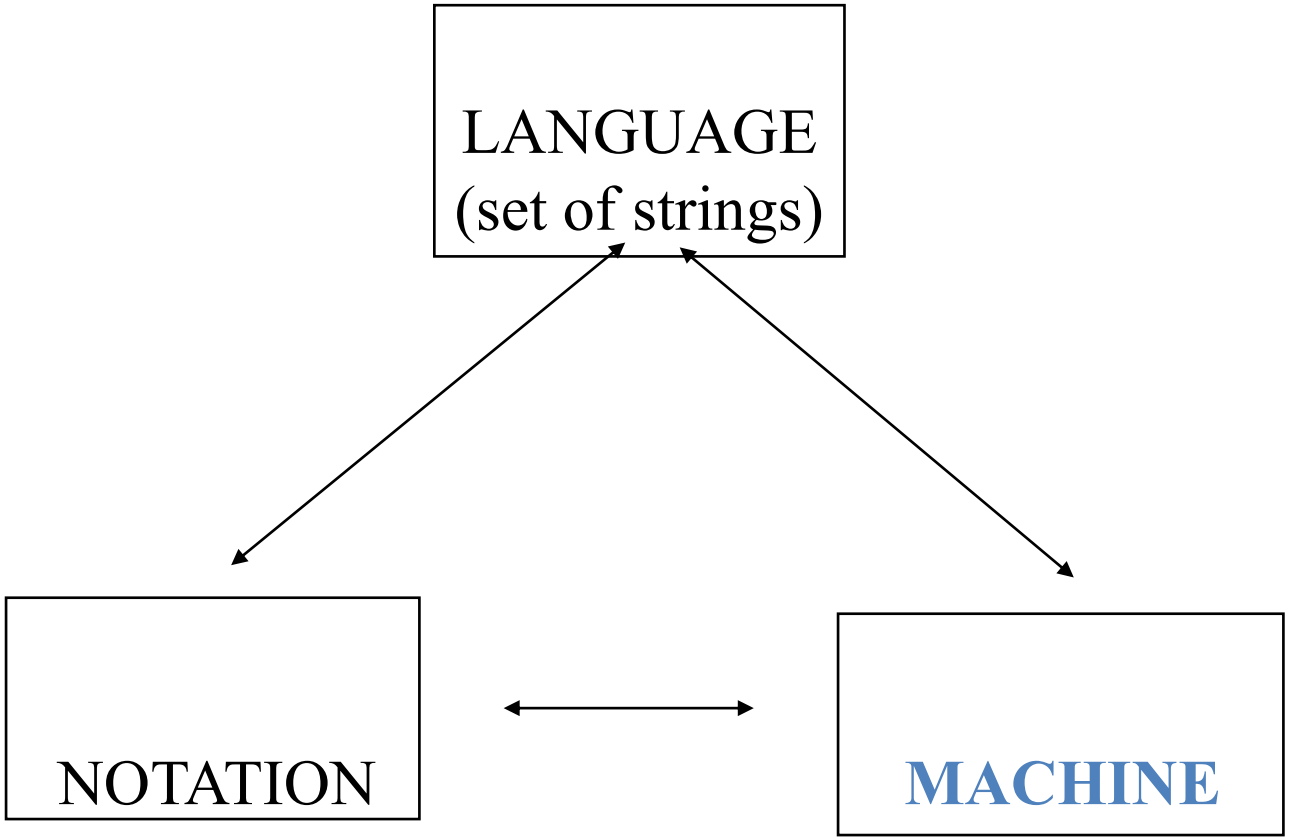
What is Finite State Technology?

- Finite State Technology refers to a collection of techniques for application of Finite State Automata (FSA) to a range of linguistically motivated problems.
- Such Techniques include
  - Design of user languages for specifying FSA
  - Compilation of such languages into efficient transition networks.
  - Development environments and runtime systems

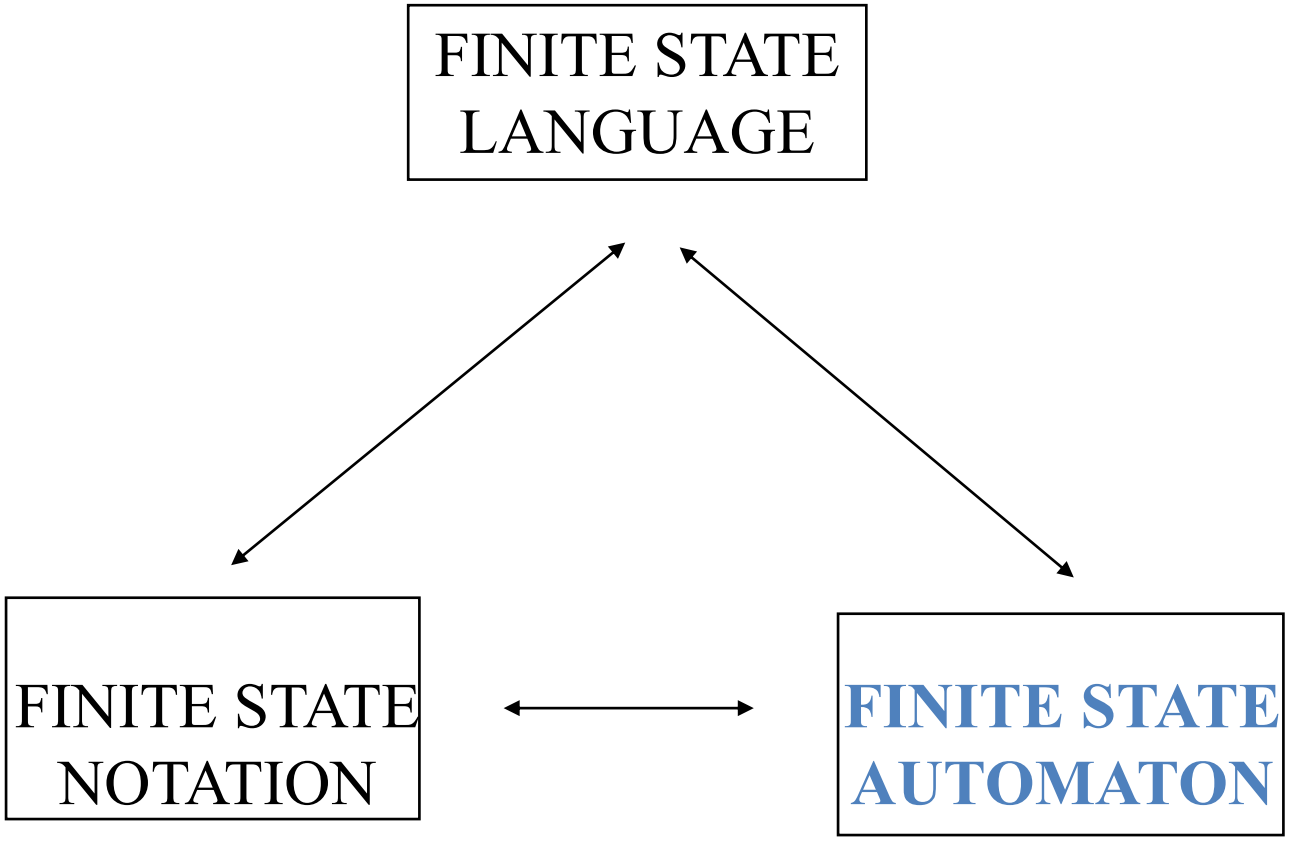
What is Finite-State Technology Good For?

- Finite-state techniques cannot handle central embedding
  - the man the dog the cat bit followed ate.
- They are well suited to "lower-level" natural language processing such as
  - Tokenization – what is the next word?
  - Spelling error detection: does the next word belong to a list?
  - Morphological/phonological analysis/generation
  - Shallow syntactic parsing and "chunking"

Languages, Notations and Machines



Languages, Notations and Machines

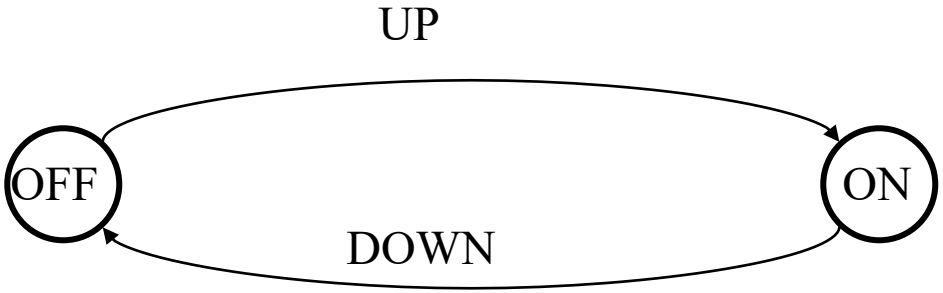


FINITE STATE AUTOMATA: preliminary definition

- A finite state automaton includes:
  - A finite set of states
  - A finite set of labelled transitions between states

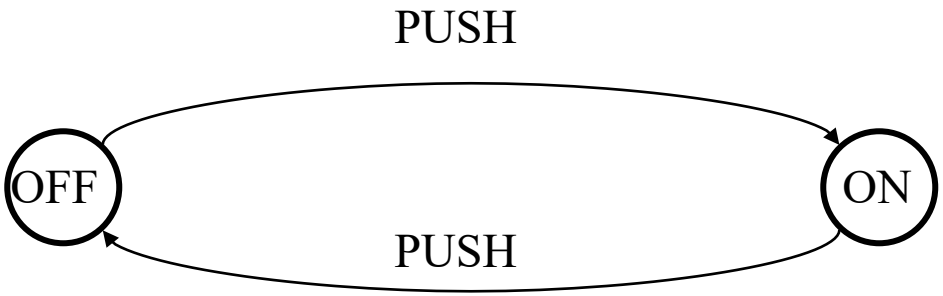
Physical Machines with Finite States

- The Lightswitch Machine



Physical Machines with Finite States

- The Lightswitch Toggle Machine



The Five Cent Machine

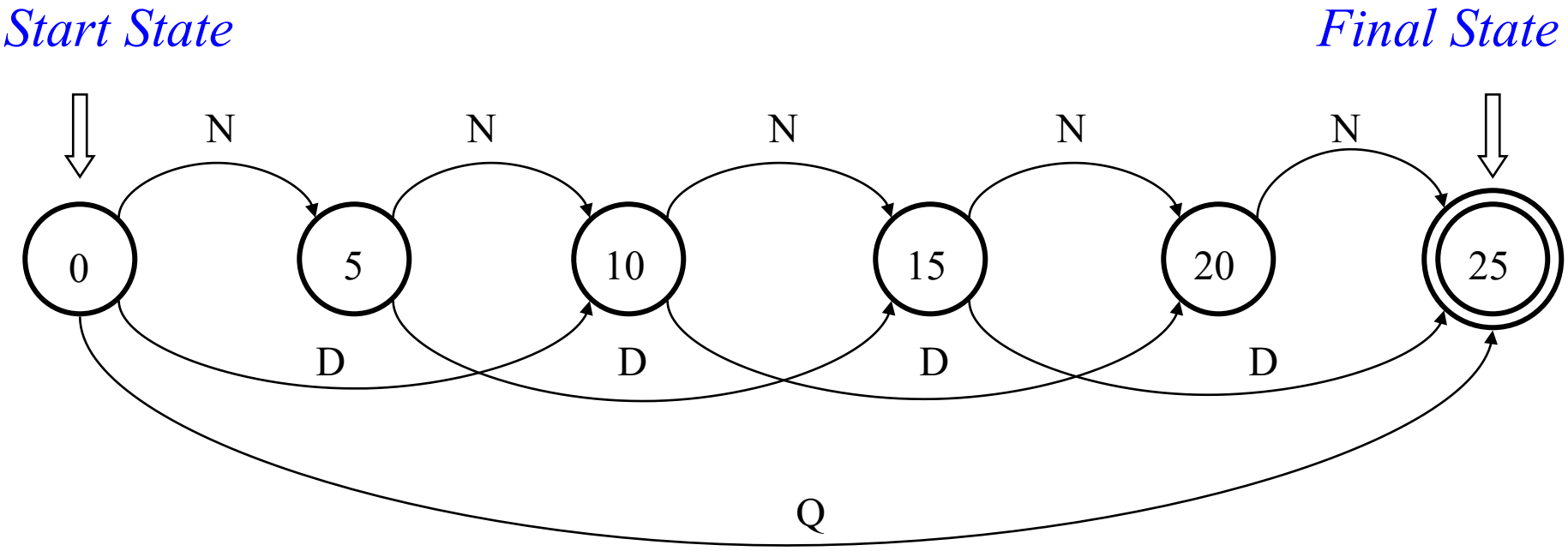
- Problem:
  - Assume you have one, two, and five cent pieces
  - Design a finite state automaton which accepts exactly 5 cents.

The Cola Machine

- Need to enter 25 cents (USA) to get a drink
- Accepts the following coins:
  - Nickel = 5 cents
  - Dime = 10 cents
  - Quarter = 25 cents
- For simplicity, our machine needs exact change
- We will model only the coin-accepting mechanism

Physical Machines with Finite States

- The Cola Machine



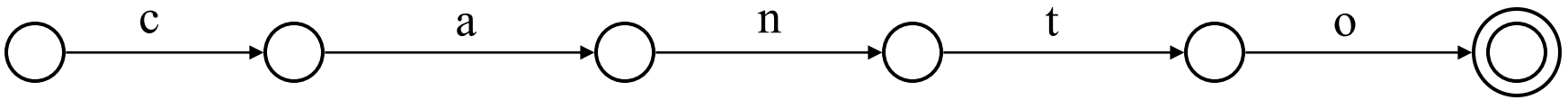
The Cola Machine Language

- List of all the sequences of coins accepted:
  - { Q, DDN, DND, NDD, DNNN, NDNN, NNDNNNND, NNNNN }
- Think of the coins as SYMBOLS or CHARACTERS
- The set of symbols accepted is the ALPHABET of the machine
- Think of sequences of coins as WORDS or “strings”
- The set of words accepted by the machine is its LANGUAGE

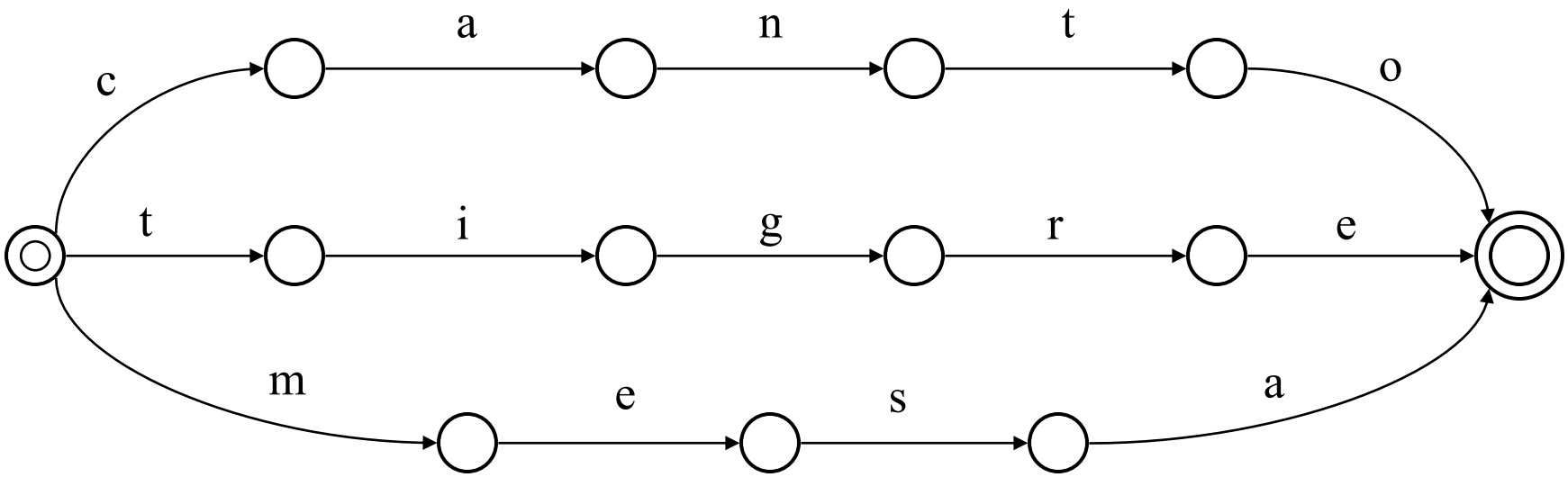
FINITE STATE AUTOMATA: better definition

- A finite state automaton includes:
- A finite set of states
  - Initial State
  - Final State (s)
- A finite set of labelled transitions between states
- Labels are symbols from an alphabet
- Recognises a language
- Generates a language as well!

A Network that Accepts a One Word Language



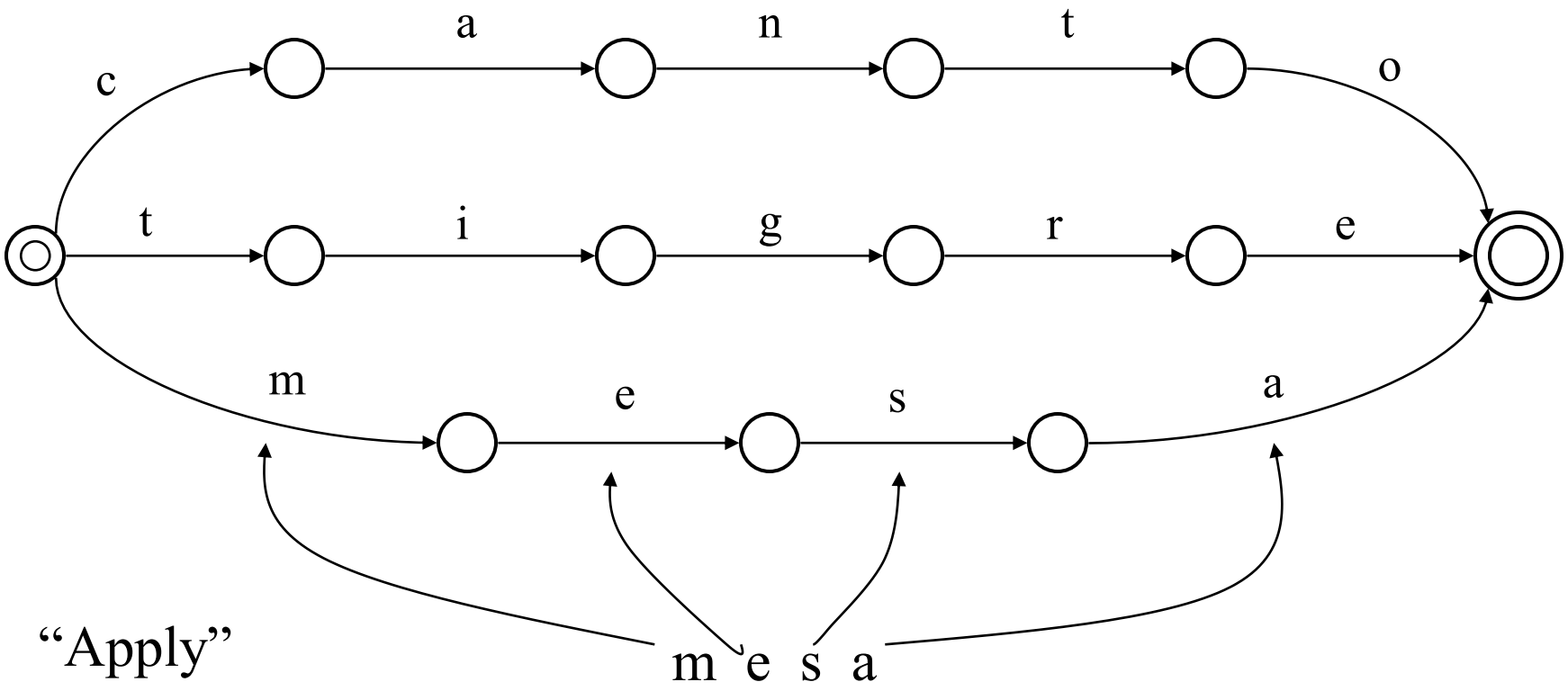
A Network that Accepts a Three Word Language



Scaling Up the Network

- Imagine the same network expanded to handle three million words, all of them corresponding to valid words of a given language.
- We supply a word and ‘apply’ it to the network. If it is accepted by the network, then it is a valid word. Otherwise it does not belong to the language
- This is the basis for a Spanish spelling error detector.

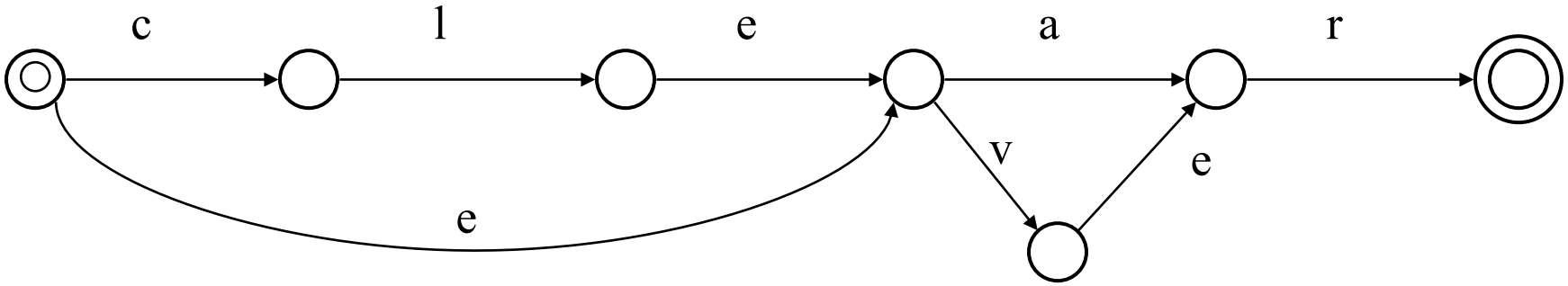
Looking Up a Word



Lookup Failure

- Lookup succeeds when all input is consumed **and** final state is reached. Lookup can fail because:
- Not all input is consumed ("libro", "tigra")
- Input is fully consumed but state is not final ("cant")
- Final state is reached but there is still unconsumed output ("mesas")

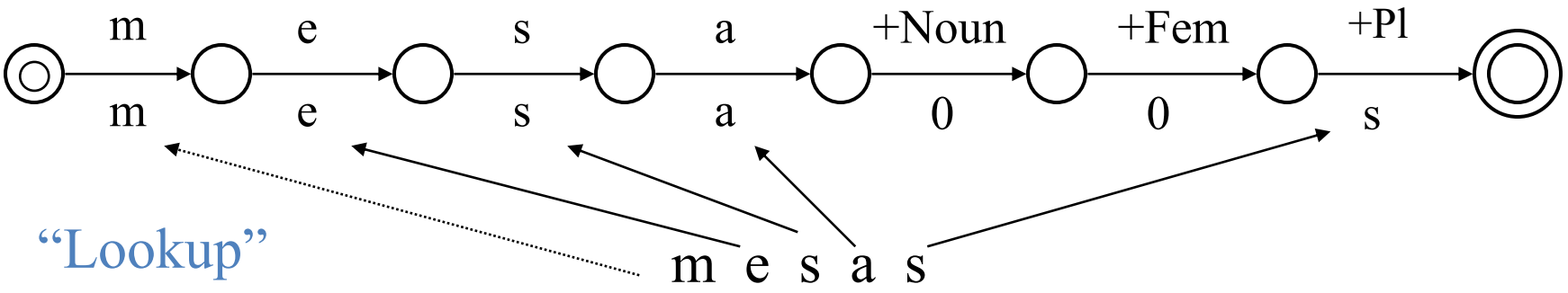
Shared Structure



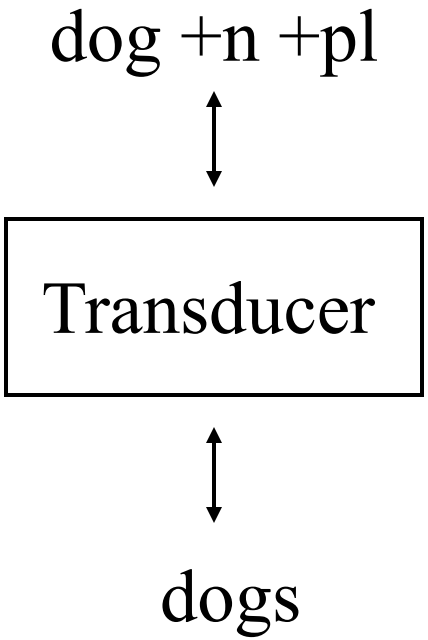
Transducers

"Lookdown"

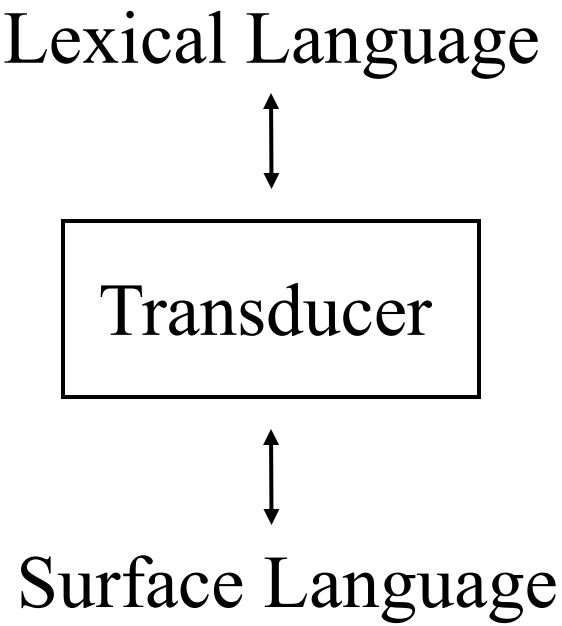
mesa+Noun+Fem+Pl



A Morphological Analyzer

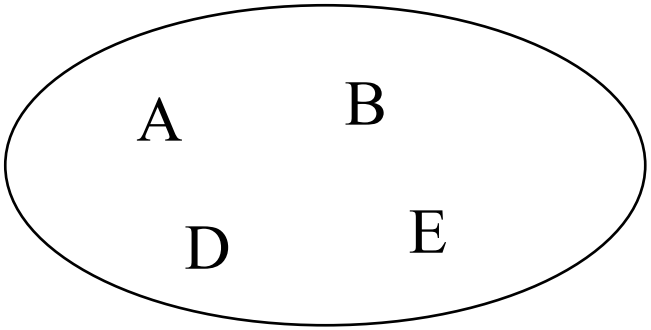


A Morphological Analyzer



A Quick Review of Set Theory

- A set is a collection of objects.

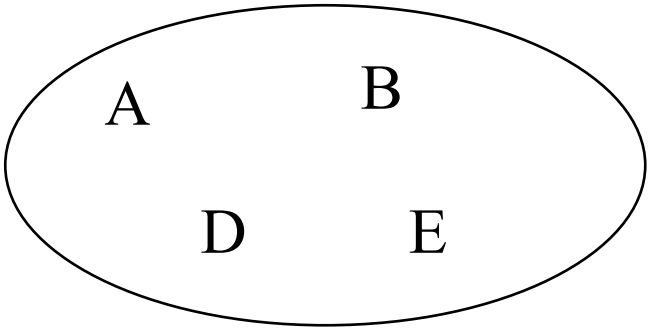


We can enumerate the “members” or “elements” of finite sets: { A, D, B, E }.

There is no significant order in a set, so { A, D, B, E } is the same set as { E, A, D, B }, etc.

Uniqueness of Elements

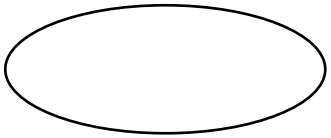
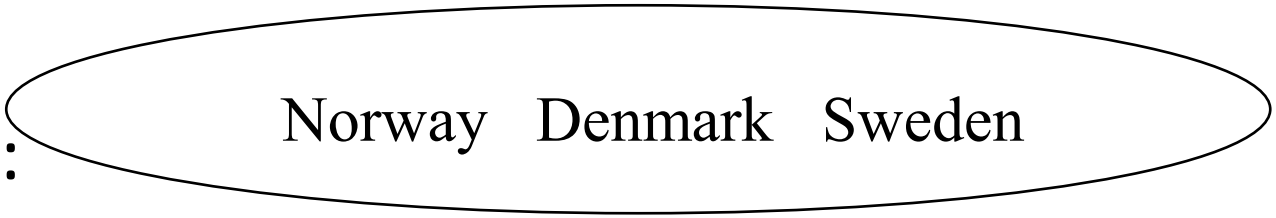
- You cannot have two or more 'A' elements in the same set



{ A, A, D, B, E} is just a redundant specification of the set

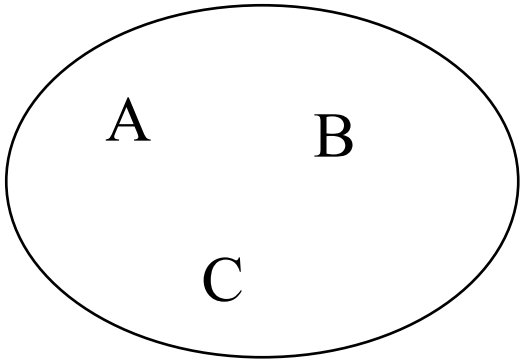
{ A, D, B, E }.

Cardinality of Sets

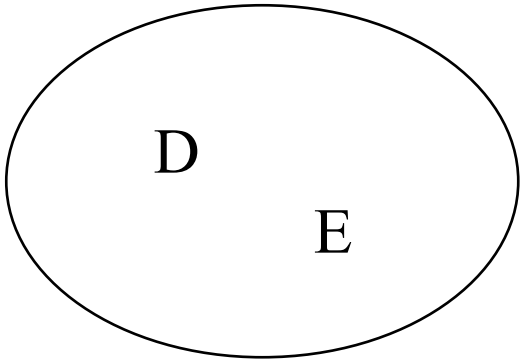
- The Empty Set: 
- A Finite Set: 

- An Infinite Set: e.g. The Set of all Positive Integers

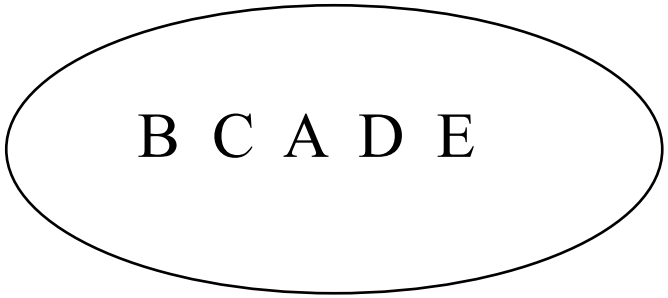
Simple Operations on Sets: Union



Set 1

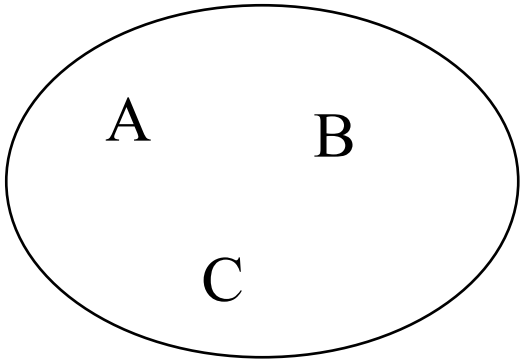


Set 2

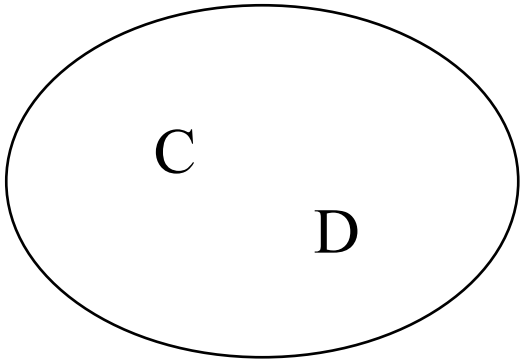


Union of Set1 and Set 2

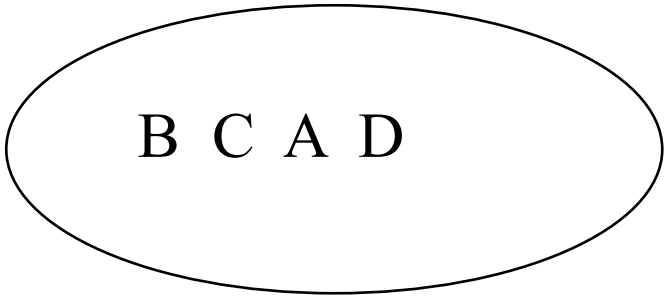
Simple Operations on Sets (2): Union



Set 1

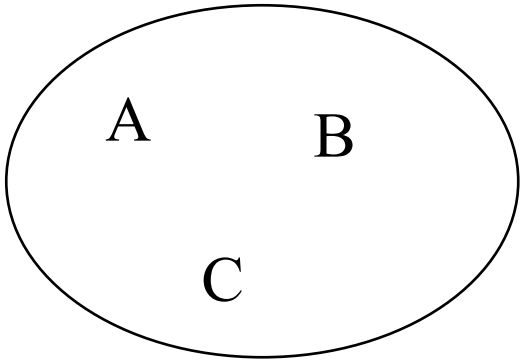


Set 2

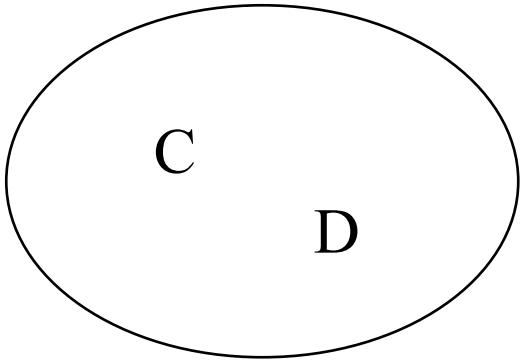


Union of Set1 and Set 2

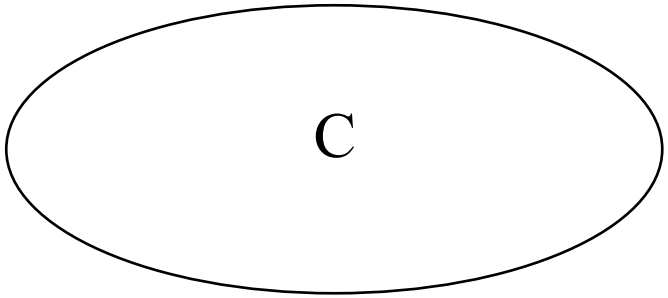
Simple Operations on Sets (3): Intersection



Set 1

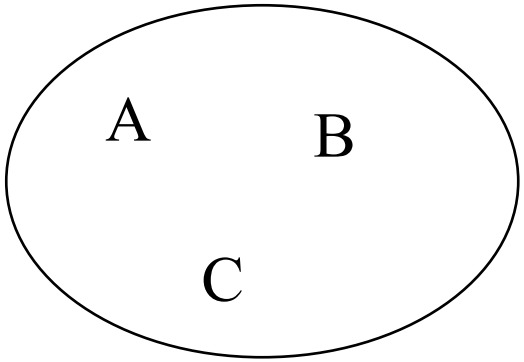


Set 2

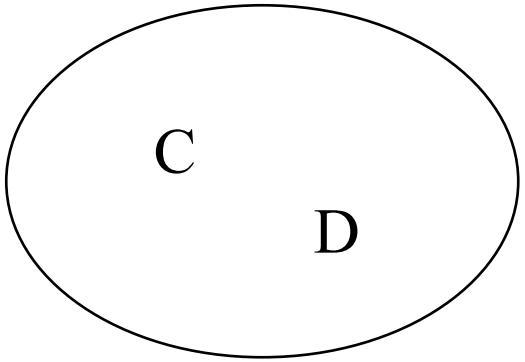


Intersection of Set1 and Set 2

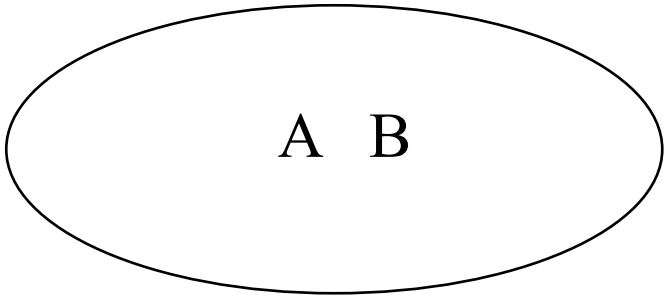
Simple Operations on Sets (4): Subtraction



Set 1



Set 2



Set 1 minus Set 2

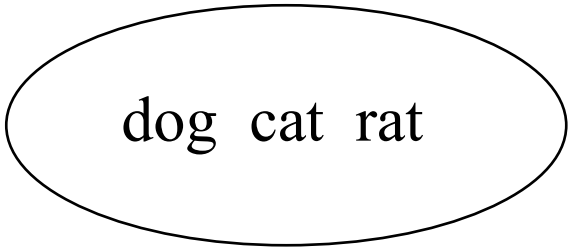
Formal Languages

Very Important Concept in Formal Language Theory:

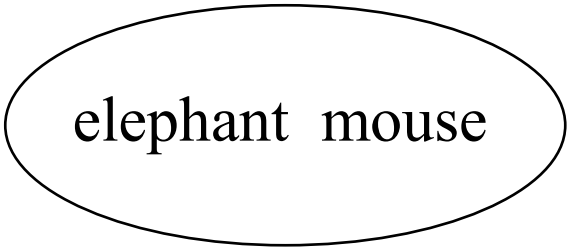
A Language is just a Set of Words.

- We use the terms "word" and "string" interchangeably.
- A Language can be empty, have finite cardinality, or be infinite in size.
- You can union, intersect and subtract languages, just like any other sets.

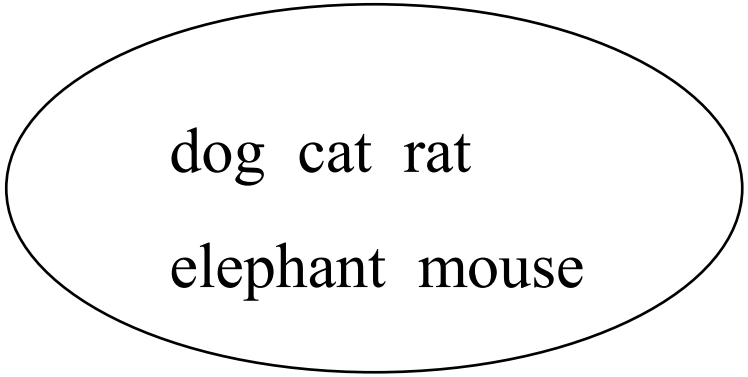
Union of Languages (Sets)



Language 1

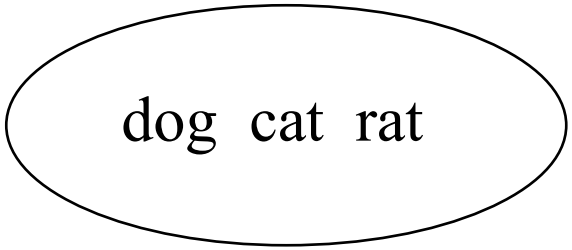


Language 2

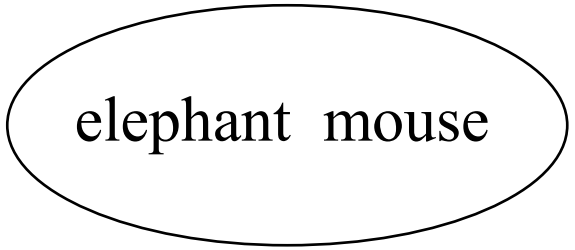


Union of Language 1 and Language 2

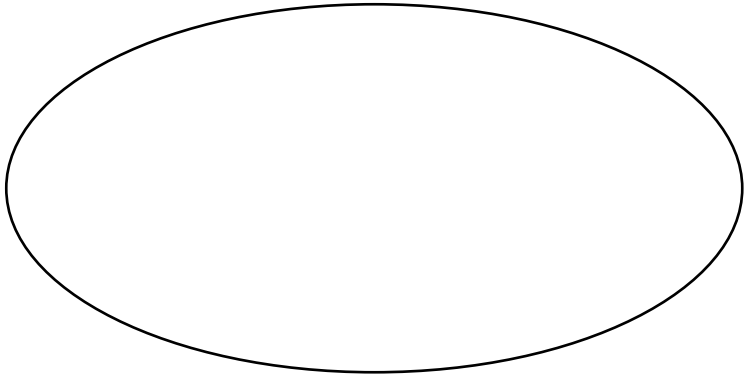
Intersection of Languages (Sets)



Language 1

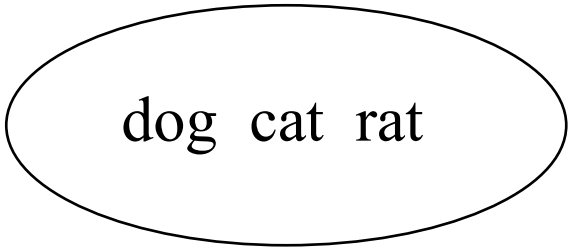


Language 2

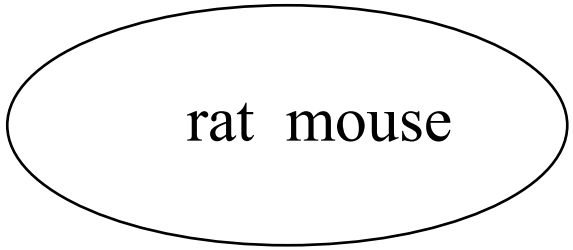


Intersection of Language 1 and Language 2

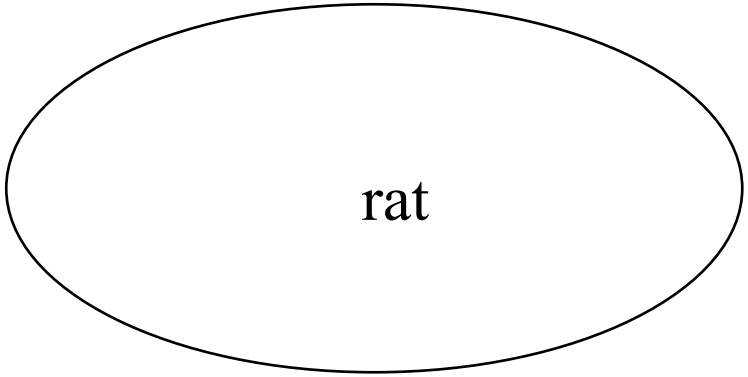
Intersection of Languages (Sets)



Language 1

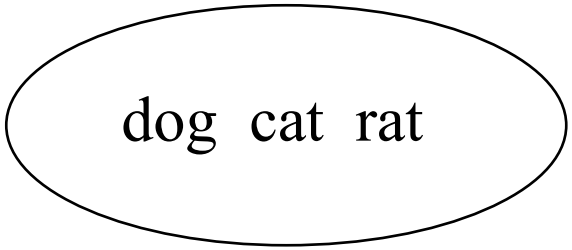


Language 2

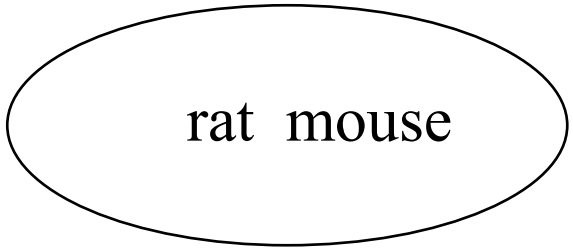


Intersection of Language 1 and Language 2

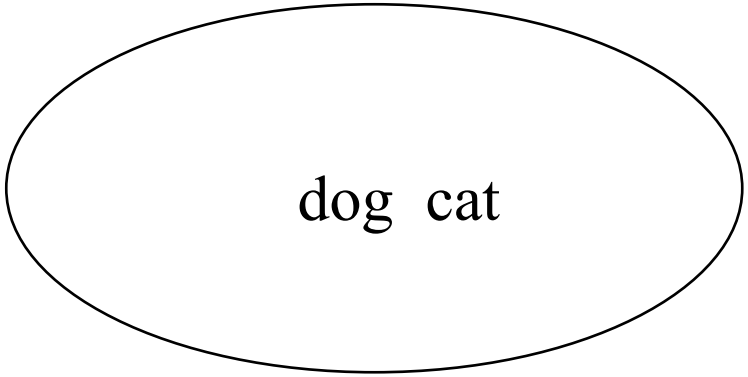
Subtraction of Languages (Sets)



Language 1



Language 2

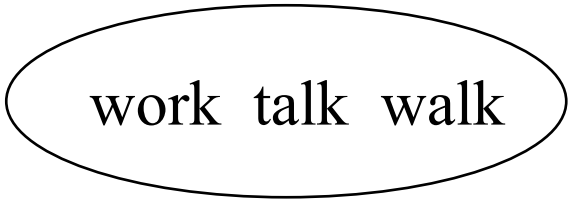


Language 1 minus Language 2

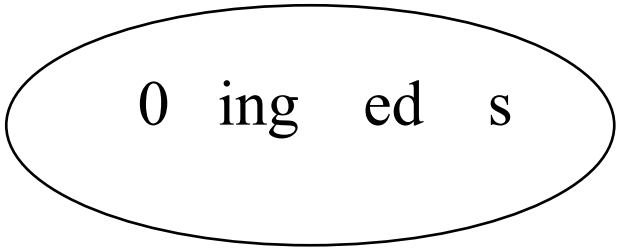
Languages

- A language is a set of words (=strings).
- Words (strings) are composed of symbols (letters) that are "concatenated" together.
- At another level, words are composed of "morphemes".
- In most natural languages, we concatenate morphemes together to form whole words.
- **For sets consisting of words (i.e. for Languages), the operation of concatenation is very important.**

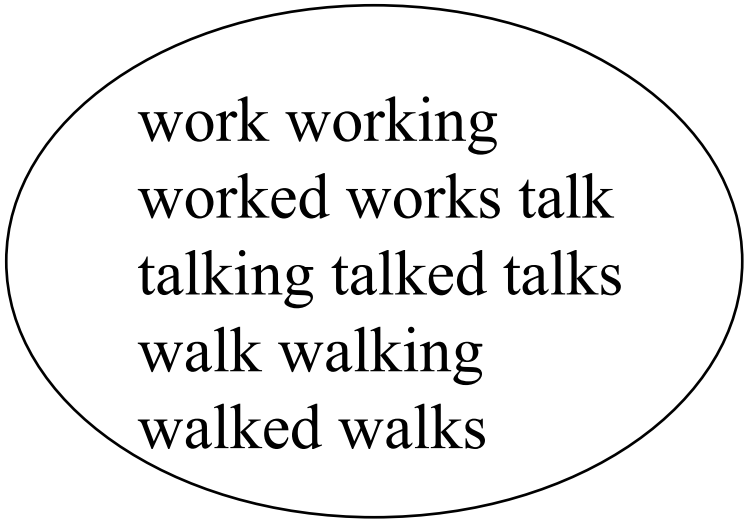
Concatenation of Languages



Root Language

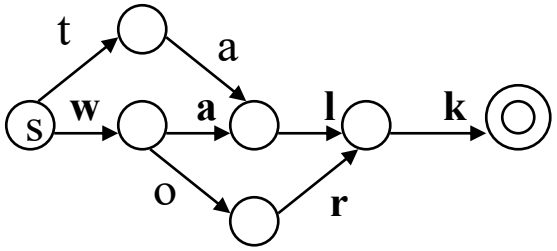


Suffix Language

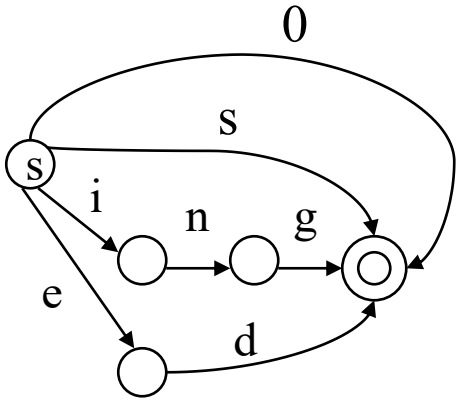


The concatenation of the Suffix language after the Root language.

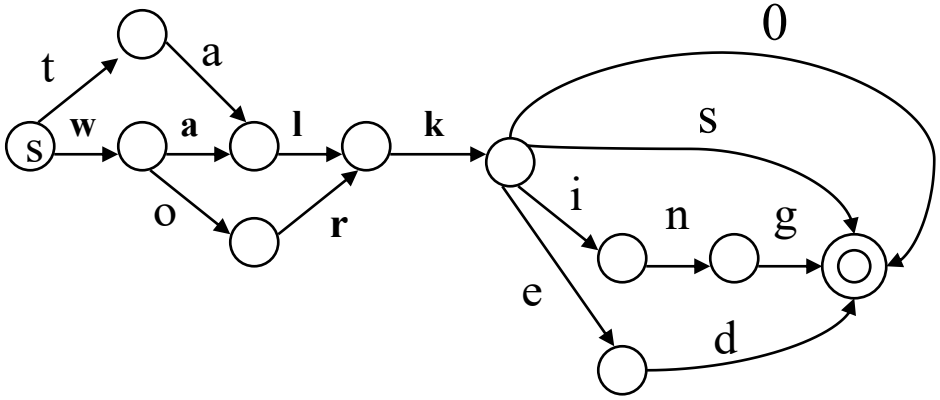
Languages and Networks



Network/Language 1



Network/Language 2

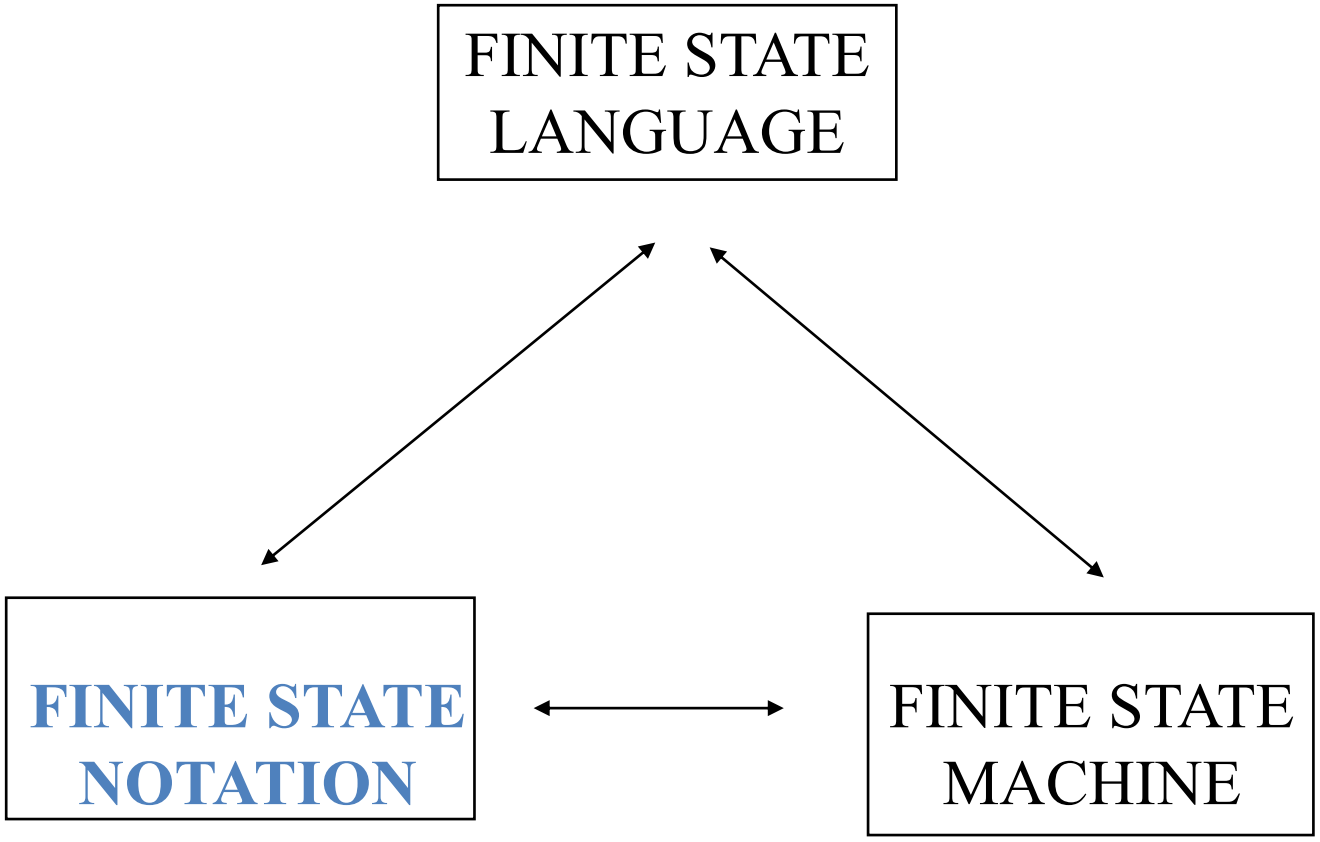


The concatenation of
Network 1 and Network 2

Why is “Finite State” Computing so Interesting?

- Finite-state systems are mathematically elegant, easily manipulated and modifiable.
- Computationally efficient. Usually very compact.
- The programming linguists do is declarative, describing facts of our natural language; i.e. we write grammars. We do not hack ad hoc code.
- Finite-state systems are inherently bidirectional: we can use the same system to analyze and to generate.

Languages, Notations and Machines



Regular Expressions
Abstract Definition

- 0 is a regular expression
- ε is a regular expression
- if $\alpha \in \Sigma$ is a letter then α is a regular expression
- if Ψ and Φ are regular expressions then so are (Ψ + Φ) and (Ψ . Φ)
- if Φ is a regular expression then so is (Φ)*
- Nothing else is a regular expression

Searching Text

- Analysis of written texts often involves searching for (and subsequent processing of):
  - a particular word
  - a particular phrase
  - a particular pattern of words involving gaps
- How can we specify the things we are searching for?

Regular Expressions and Matching

- A Regular Expression is a special notation used to specify the things we want to search for.
- We use regular expressions to define patterns.
- We then implement a matching operation m(<pattern>,<text>) which tries to match the pattern against the text.

Simple Regular Expressions

- Most ordinary characters match themselves.
- For example, the pattern `sing` exactly matches the string `sing`.
- In addition, regular expressions provide us with a set of *special characters*

The Wildcard Symbol

- The “.” symbol is called a *wildcard*: it matches any single character.
- For example, the expression *s.ng* matches *sang*, *sing*, *song*, and *sung*.
- Note that "." will match not only alphabetic characters, but also numeric and whitespace characters.
- Consequently, s.ng will also match non-words such as *s3ng*

Assignment

- Draw the FSM which corresponds to `s.ng`

Repeated Wildcards

- We can also use the wildcard symbol for counting characters. For instance ....zy matches six-letter strings that end in zy.
- The pattern t... will match, among others, the words *that* and *term*
- It will also match the word sequence *to a* (since the third "." in the pattern can match the space character).

Optionality

- The “?” symbol indicates that the immediately preceding regular expression is optional. The regular
- expression colou?r matches both British and American spellings, colour and color.

Repetition

- The "+" symbol indicates that the immediately preceding expression is repeatable at least once
- For example, the regular expression "coo+l" matches cool, coool, and so on.
- This symbol is particularly effective when combined with the . symbol. For example,
- f .+ f matches all strings of length greater than two, that begin and end with the letter f (e.g foolproof).

Repetition 2

- The “*” symbol indicates that the immediately preceding expression is both optional and repeatable.
- For example .*gnt.* matches all strings that contain gnt.

Character Class

- The [ ] notation enumerates the set of characters to be matched is called a *character class*.
- For example, we can match any English vowel, but no consonant, using [aeiou].
- We can combine the [] notation with our notation for repeatability.
- For example, expression p[aeiou]+t matches peat, poet, and pout.

The Choice Operator

- Often the choices we want to describe cannot be expressed at the level of individual characters.
- In such cases we use the choice operator "|" to indicate the alternate choices.
- The operands can be any expression.
- For instance, jack | gill will match either jack or gill.

Choice Operator 2

- Note that the choice operator has wide scope, so that abc|def is a choice between abc and def, and not between abcef and abdef.
- The latter choice must be written using parentheses: ab(c|d)ef

Ranges

- The [ ] notation is used to express a set of choices between individual characters.
- Instead of listing each character, it is also possible to express a *range* of characters, using the - operator.
- For example, [a-z] matches any lowercase letter

Exercise

Write regular expressions matching

- All 1 digit numbers
- All 2 digit numbers
- All date expressions such as 12/12/1950

Ranges II

- Ranges can be combined with other operators.
- For example [A-Z][a-z]* matches words that have an initial capital letter followed by any number of lowercase letters.
- Ranges can be combined as in [A-Za-z] which matches any alphabetical character.

Assignment

- What does the following expression match?
- [b-df-hj-np-tv-z]+

Complementation

- the character class [b-df-hj-np-tv-z] allows us to match consonants.
- However, this expression is quite cumbersome.
- A better alternative is to say: let's match anything which isn't a vowel.
- To do this, we need a way of expressing *complementation*.

Complementation 2

- We do this using the symbol “^” as the first character within the class expression [ ].
- [^aeiou] is just like our earlier character class, except now the set of vowels is preceded by ^.
- The expression as a whole is interpreted as matching anything which *fails* to match [aeiou]
- In other words, it matches all lowercase consonants (plus all uppercase letters and non-alphabetic

Complementation 3

- As another example, suppose we want to match any string which is enclosed by the HTML tags for boldface, namely <B> and </B>, We might try something like this: <B>.*</B>.
- This would successfully match *<B>important</B>*, but would also match *<B>important</B> and <B>urgent</B>*, since the <.*y subpattern will happily match all the characters from the end of important to the end of urgent.

Complementation 4

- One way of ensuring that we only look at matched pairs of tags would be to use the expression <B>[^<]*</B>, where the character class matches anything other than a left angle bracket.
- Finally, note that character class complementation also works with ranges. Thus [^a-z] matches anything other than the lower case alphabetic characters a through z.

Other Special Symbols

- Two important symbols in this are “^” and “$” which are used to *anchor* matches to the beginnings or ends of lines in a file.
- **Note:** “^” has two quite distinct uses: it is interpreted as complementation when it occurs as the first symbol within a character class, and as matching the beginning of lines when it occurs elsewhere in a pattern.

Special Symbols 2

- As an example, [a-z]*s$ will match words ending in s that occur at the end of a line.
- Finally, consider the pattern ^$; this matches strings where no character occurs between the beginning and the end of a line — in other words, empty lines.

Special Symbols 3

- Special characters like “.”, “*”, “+” and “$” give us powerful means to generalise over character strings.
- Suppose we wanted to match against a string which itself contains one or more special characters?
- An example would be the arithmetic statement $5.00 * ($3.05 + $0.85).
- In this case, we need to resort to the so-called *escape* character “\” (“backslash”).
- For example, to match a dollar amount, we might use \$[1-9][0-9]*\.[0-9][0-9]

Summary

- Regular Expressions are a special notation.
- Regular expressions describe patterns which can be matched in text.
- A particular regular expression E stands for **a set of strings**. We can thus say that E describes a **language**.

Great!

See you next time!