CSC 222: AUTOMATA THEORY

Lecture II: Languages in the Abstract

Building blocks of Language

- Letters
- Words
- Sentences
- Paragraphs
- Stories…

Levels of Description

- Alphabet (character set)
- Lexicon / dictionary
- Syntax / grammar
- Semantics / meaning
- Pragmatics

Language Structure

- General language theory needed => universal language structure
- Decision as to whether a given string of units constitutes a valid larger unit is explicit and does not rely on guesswork
- Letters: alphabet lookup
- Words: dictionary lookup
- Sentences: rules of grammar…

Challenges

- Hard for Natural Languages – why?

Challenges

- Rules can be broken: idiom, dialect, slang, poetic metaphor, automatic correction
- Lookup cannot deal with structures of infinite size and infinite variety
  - Show that sentences in any natural language are infinite?

How do we represent language?

- One of two ways:
  - An alphabet and the exhaustive list of all valid words
  - An alphabet and a set of rules defining the acceptable words (set of rules = grammar)

A sentence rule?

Alphabet = {n o m a w i}

If it is proven that God exists then language
PERSON = {man}

If it is never proven that God exists then language
PERSON = {woman}

Is woman in PERSON?

Acceptable Grammar Rules

- Must enable us to decide, in a finite amount of time, whether a given string of alphabet letters is or is not a word/sentence in the language.
- Note: it is not a requirement that all the letters in the alphabet appear in the words selected for the language.

Formal Language Specification

- Use simplified, formal languages
- Define the alphabet
- Define the words
- Manipulate the language using strict rules

Basic Concepts in Formal Languages

- $\Sigma = \{x\}$
- $L_1 = \{x\ xx\ xxx\ xxxx\ \ldots\}$
- $L_1 = \{\boldsymbol{x^n}$ *for* $n = 1\ 2\ 3\ \ldots\}$
- Λ or the null string can also be part of a language (but it is not part of L_1 in this case)

Defining a Function

- Concatenation (joining two strings to form a new longer string)
- Concatenating *xx* with *xxx* gives the string *xxxxx*
- Another way of putting it:

x^n concatenated with x^m is the string x^{n+m}

Properties of Functions

- Concatenating two words in a language may or may not produce a string in the language
- Example L_{odd} = {x xxx xxxxx …}
- What can we say about the string formed by concatenating x^n and x^m when both strings are members of L_{odd}?

Properties of Functions

- Concatenation is not always a symmetric function (i.e. concatenating string x^n to x^m may not always give the same result as concatenating string x^m to x^n)
- Example, think of the English strings “paint” and “house”

Properties of Functions

- Σ = {0 1 2 3 4 5 6 7 8 9}
- L_3 = {Any finite string of alphabet letters from Σ that does not start with the letter 0}
- What is another way of describing language L_3?
- If we wanted to include the string (word) 0 in L_3, how would you define L_3?

Definitions and Proofs

- Definitions and proofs are the building blocks of mathematical descriptions of computation.
- Definitions set the boundaries while
- Proofs enable us to extend knowledge by showing rigorously that certain facts can be relied upon for further processing.
- A definition is: a statement giving the precise meaning of an entity
- A proof is: fact or evidence or argument sufficing to establish a fact or truth

The Length Function

- Definition: the length of a string is the number of letters in the string.
- Examples:
  - If a = xxxx, length(a) = 4
  - If c = 428, length(c) = 3
  - length(xxxxx) = 5
  - length(Λ) = 0

Facilities

- Defining functions gives us new ways to define languages!
- Example
  - L_3 = {any finite string of alphabet letters that, if it has length more than one, does not start with a zero}

The Reverse Function

- Definition: if a is a string in some language L, then reverse(a) is the same string of letters spelled backwards, called the reverse of L.
- Note that like concatenation, reverse does not necessarily produce a string that belongs to L.
- Examples:
  - Reverse(*xxx*) = *xxx*
  - Reverse(*xxxxx*) = *xxxxx*
  - Reverse(432) = 234
- Example:
  - Reverse(240) = 042 which is not a string in L_3

The Palindrome Function

- Definition: Given an alphabet Σ, the language palindrome is the language of all strings, y, over the alphabet Σ where reverse(y) = y
- Example:
- Σ = {a b}
- Palindrome = {Λ a b aa bb aaa aba bab bbb …}

The Closure Operation

- Definition: Given an alphabet Σ, the null string and any string of letters that can be formed by concatenating letters from Σ, are strings in a language called the CLOSURE of the alphabet.
- The notation for the closure is Σ^* (This star is called the Kleene star)
- The Kleene star is an operation that makes an infinite language of strings of letters from an alphabet
- Example:
- Σ = {*x*}, Σ^* = {*Λ x xx xxx xxxx xxxxx*…}
- Σ = {0 1}, Σ^* = {Λ 0 1 00 01 10 11 000 001 010 011 100…}
- Σ = {*a b c*}, Σ^* = {*Λ a b c aa ab ac ba bb bc*…}
- What do you notice about these closures?

The Closure Operation

- A closure makes an infinite language out of a finite alphabet (however, note that each string in the infinite language has finite length)
- The closure always begins with Λ
- The closure always contains the alphabet
- It is easier to write the set of strings in a sequence that begins with the shortest first (lexicographic ordering)

Generalized Closure

- The same operation can be used with a set of strings
- Definition: If S is a set of strings (words), then S^* is the null string and the set of all finite strings formed by concatenating words from S, where any word may be used as often as we like.

Example of Generalized Closure

- S = {a ab}
- S^* = {Λ a aa ab aaa aab aba aaaa aaab …}
- Now that the closure is more complex we need a method of proving that a word belongs in the closure
- We can do this by enumerating all the words in the closure up to the suspect word. An easier way is by factoring…

Factoring

- Definition: factoring means dividing up a string into substrings that belong to the closure.
- For example: abaab belongs to the language above because it can be factored into the substrings:

(ab) (a) (ab)

- Since these three factors are all in the set S, therefore their concatenation is in S*.
- Unique factoring = when there is only one way to factor a string
- Does ababba belong to the language defined above?

Factoring

- Note that the parentheses '(' and ')' are not letters in the alphabet and are being used solely to factor.
- If, in some language, the parentheses are letters of the alphabet then it is important to define them as such.

Proof by Constructive Algorithm

Consider S = {xx, xxx}.

S* = {Λ and all strings of more than one x}

= {x^n for n= 0, 2, 3, 4, 5,..}

= {Λ xx xxx xxxx xxxxx ...}

is xxxxxxx in this closure?

Proof by Constructive Algorithm

- Define the problem clearly
- Solve the problem for the simple elements
- Restate the problem as a search for the first counterexample
- Demonstrate that no counterexample can be found
- State the solution as a fact
- END OF PROOF

Proof by Constructive Algorithm

- Suppose you want to prove mathematically that set S^* contains all x^n for $n \neq 1$ (this is the first step, a clear definition of the problem)
- You would begin by showing that this is true for the first few members of the set i.e. x^2, x^3, x^4, x^5, x^6 etc.
- You would then try to **disprove** the theorem by proving that ***there is some x^i that is not generated by S^****
- For ease you would then try to identify the ***first x^i that is not in S^****
- you can imagine that you were building strings of x's sequentially and you come to the first one that you cannot generate in S^*

Proof by Constructive Algorithm

- For example you may have generated up to x^{353} and when you get to x^{354} you find that you cannot generate it
- However, a quick glance shows you that by concatenating x^{352} and x^2 (both previous members of S^*) it is possible to generate x^{354}!
- Therefore there is no such x^i
- Therefore, if it is not possible to identify a counterexample, then the hypothesis that set S* contains all x^n for $n \neq 1$ must be true
- END OF PROOF

Proof by Constructive Algorithm

- This type of proof is attractive to Computer Scientists because it is a proof that says 'If I can build it then it is true.'
- We will be seeing more of this type of proof so let us go through it's essential elements once again…

Proof by Constructive Algorithm

- Define the problem clearly
- Solve the problem for the simple elements
- Restate the problem as a search for the first counterexample
- Demonstrate that no counterexample can be found
- State the solution as a fact
- END OF PROOF

Recap

- If $\Sigma = \{\}$ then $\Sigma^* = \Lambda$
- If $S = \{\Lambda\}$ then $S^* = \Lambda$
- If $\Sigma = \{x\}$ then $\Sigma^+ = \{x\ xx\ xxx\ \ldots\}$ (L_1)
  Note: + is referred to as positive closure
- Generalized string closures:

  $S = \{w_1\ w_2\ w_3\}$
  $S^+ = \{w_1, w_2, w_3, w_1w_1, w_1w_2, w_1w_3, w_2w_1, w_2w_2, w_2w_3, \ldots\}$

Exercise

- Consider the language S*, where S = {a, b}. How many words does this language have of length 2? Of length 3? Of length n?
- Prove that for any set S of strings, S* = S**
  - Descriptively
  - By constructive algorithm