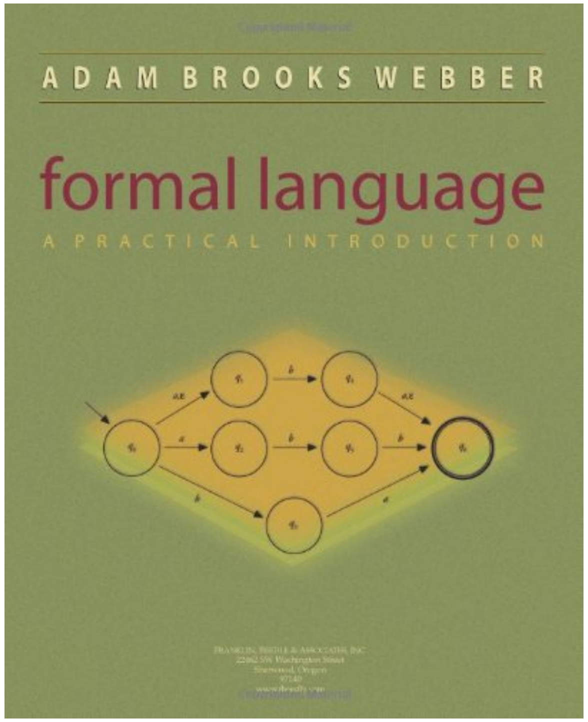
Welcome!

- CSC445 – Models Of Computation
- Dr. Lutz Hamel
- Tyler 251
- hamel@cs.uri.edu



https://fbeedle.com/content/formal-language-practical-introduction-0

The Course

- Text: “Formal Language”, by Adam Book Webber, Franklin, Beedle & Associates, 2007.
- Special Features:
  - Course website with lecture notes
  - Online gradebook (email me for access code)

Introduction and Chapter One: Fundamentals

No one who loves language can take much pleasure in the prospect of studying a subject called formal language. It sounds suspiciously abstract and reductionistic. It sounds as if all the transcendent beauty of language will be burned away, fired under a dry heat of definitions and theorems and proofs, until nothing is left but an ash of syntax. It sounds abstract—and it is, undeniably. Yet from this abstraction arise some of the most beautiful and enduring ideas in all of computer science.

Why Study Formal Language?

- Connected...
  - ...to many other branches of knowledge
- Rigorous...
  - ...mathematics with many open questions at the frontiers
- Useful...
  - ...with many applications in computer systems, particularly in programming languages and compilers
- Accessible...
  - ...no advanced mathematics required
- Stable...
  - ...the basics have not changed much in the last thirty years

*Algebraists use the words **group**, **ring**, and **field** in technical ways, while entomologists have precise definitions for common words like **bug** and **fly**. Although it can be slightly confusing to overload ordinary words like this, it's usually better than the alternative, which is to invent new words. So most specialized fields of study make the same choice, adding crisp, rigorous definitions for words whose common meaning is fuzzy and intuitive.*

*The study of formal language is no exception. We use crisp, rigorous definitions for basic terms such as **alphabet**, **string**, and **language**.*

Outline

- 1.1 Alphabets
- 1.2 Strings
- 1.3 Languages

Alphabets

- An *alphabet* is any finite set of symbols
  - {0,1}: binary alphabet
  - {0,1,2,3,4,5,6,7,8,9}: decimal alphabet
  - ASCII, Unicode: machine-text alphabets
  - Or just {a,b}: enough for many examples
  - {}: a legal but not usually interesting alphabet
- We will usually use Σ as the name of the alphabet we're considering, as in $\Sigma = \{a,b\}$

Alphabets Uninterpreted

- Informally, we often describe languages interpretively
  - "the set of even binary numbers"
- But our goal is to describe them rigorously, and that means avoiding interpretations
  - "the set of strings of 0s and 1s that end in 0"
- We don't define what a *symbol* is, and we don't ascribe meaning to symbols

Outline

Strings

- A *string* is a finite sequence of zero or more symbols
- Length of a string: $|abbb| = 4$
- *A string over the alphabet* Σ means *a string all of whose symbols are in* Σ
  - The set of all strings of length 2 over the alphabet $\{a,b\}$ is $\{aa, ab, ba, bb\}$

Empty String

- The empty string is written as ε
- Like "" in some programming languages
- $|\varepsilon| = 0$
- Don't confuse empty set and empty string:
  - $\{\} \neq \varepsilon$
  - $\{\} \neq \{\varepsilon\}$

Symbols And Variables

- Sometimes we will use variables that stand for strings: x = *abbb*
- In programming languages, syntax helps distinguish symbols from variables
  - `String x = "abbb";`
- In formal language, we rely on context and naming conventions to tell them apart
- We'll use the first letters, like *a*, *b*, and *c*, as symbols
- The last few, like *x*, *y,* and *z*, will be string variables

Concatenation

- The *concatenation* of two strings x and y is the string containing all the symbols of x in order, followed by all the symbols of y in order
- We show concatenation just by writing the strings next to each other
- If $x = abc$ and $y = def$, then $xy = abcdef$
- For any x, $\varepsilon x = x\varepsilon = x$

Numbers

- We use $\mathcal{N}$ to denote the set of natural numbers: $\mathcal{N} = \{0, 1, \ldots\}$

Exponents

- Exponent *n* concatenates a string with itself *n* times
  - If $x = ab$, then
    - $x^0 = \varepsilon$
    - $x^1 = x = ab$
    - $x^2 = xx = abab$, etc.
  - We use parentheses for grouping exponentiations (assuming that Σ does not contain the parentheses)
    - $(ab)^7 = ababababababab$

Outline

- 1.1 Alphabets
- 1.2 Strings
- 1.3 Languages

Languages

- A *language* is a set of strings over some fixed alphabet
- *Not* restricted to finite sets: in fact, finite sets are not usually interesting languages
- All our alphabets are finite, and all our strings are finite, but most of the languages we're interested in are infinite

Kleene Star

- The Kleene closure of an alphabet Σ, written as Σ*, is the language of all strings over Σ
  - {*a*}* is the set of all strings of zero or more *a*s: {ε, *a*, *aa*, *aaa*, …}
  - {*a*,*b*}* is the set of all strings of zero or more symbols, each of which is either *a* or *b* = {ε, *a*, *b*, *aa*, *bb*, *ab*, *ba*, *aaa*, …}
  - $x \in \Sigma^*$ means *x* is a string over Σ
- Unless Σ = {}, Σ* is infinite
- If Σ = {} then what is Σ*?

Set Formers

- A set written with extra constraints or conditions limiting the elements of the set:

$\{x \in \{a, b\}^* \mid |x| \leq 2\} = \{\varepsilon, a, b, aa, bb, ab, ba\}$

$\{xy \mid x \in \{a, aa\} \text{ and } y \in \{b, bb\}\} = \{ab, abb, aab, aabb\}$

$\{x \in \{a, b\}^* \mid x \text{ contains one } a \text{ and two } b\text{s}\} = \{abb, bab, bba\}$

$\{a^n b^n \mid n \geq 1\} = \{ab, aabb, aaabbb, aaaabbbb, \ldots\}$

The Quest

- Using set formers to describe complex languages is challenging
- They can often be vague, ambiguous, or self-contradictory
- A big part of our quest in the study of formal language is to develop better tools for defining languages