# Fall 2018
# CS 591 S2
# Formal Language Theory: Integrating Experimentation and Proof
**TR 12:30-1:45pm CAS 324**

**Alley Stoughton**

`https://alleystoughton.us/591-s2`

`stough@bu.edu`

Unless you've attended seminars at BU's Hariri Institute, you probably don't know anything about me. So let me start by saying that I did my doctoral work at the University of Edinburgh in Scotland, taught for several years at the University of Sussex in England, and then was an associate professor at Kansas State University for seventeen years. Subsequently to moving to Boston in 2010, I've worked at MIT Lincoln Laboratory, Madrid's IMDEA Software Institute (mostly remotely) and BU's Hariri Institute. My background is in programming languages and formal methods, but in recent years I've been working at the intersection of programming languages/formal methods and security, including cryptography. I'm also interested in the pedagogy of formal language theory—and thus this course.

Formal language theory—also known as automata theory—is concerned with various ways of describing formal languages, which are sets of strings over alphabets. For example, the set of identifiers of a programming language is a formal language.

Regular expressions and finite automata describe the regular languages, which is the most restrictive level in what's called the Chomsky hierarchy, after MIT linguist Noam Chomsky. Regular expressions are used for many purposes, including searching for patterns in text, and specifying lexical analyzers, the phase of a compiler that groups characters into tokens. Finite automata are used to implement lexical analyzers, as well as when specifying hardware and network protocols, among many other purposes.

Next in the hierarchy of languages comes the context-free languages, which are described by context-free grammars. Grammars are used to specify the syntax of programming languages and many other tree-structured notational systems. And parsers implement grammars, turning lists of tokens into parse trees.

Finally, the recursive and recursively enumerable languages are defined using Turing machines—or other universal programming languages. These languages are used to study what is—and is not—computable.

Formal language theory is traditionally taught as a paper-and-pencil mathematics course, even though it is largely concerned with algorithms on regular expressions, finite automata, grammars and Turing machines. But about fifteen years ago I set out to change that, by building a toolset—which I came to call Forlan—implementing these algorithms, and by using the toolset in my teaching, so as to balance proof with experimentation. Forlan's embedded in the functional programming language Standard ML, a language whose notation and concepts are similar to those of mathematics. It's strongly typed and interactive, properties that make sophisticated experimentation robust and enjoyable.

The prerequisites of this course are CS 112 (Introduction to Computer Science 2) and CS 131 (Combinatoric Structures), or equivalent, as we'll be doing both programming, using Forlan, and proving, using informal mathematics. If you are scared of doing proofs, the experimental aspect of my course will ease you into the theory.

You'll presumably be happy to hear that the textbook for my course won't cost you anything. We'll be using a draft of the book I've been writing, which integrates experimentation using Forlan with proof.

In terms of assessment, the focus will be on practical work—there will be seven problem sets, some of them involving use of Forlan. But there will also be a cumulative final exam. To help you learn, I'll be offering weekly problem solving sessions, in which I'll solve problems at the blackboard.

Thanks for reading my pitch for the course. If you have any questions, I'll be happy to answer them. Please email me at `stough@bu.edu`.