Preface

These slides are a summary of the the book's Preface:

- the subject of formal language theory;
- the Forlan Project;
- overview of book.

Background

Since the 1930s, the subject of formal language theory has been developed by computer scientists, linguists and mathematicians.

Background

Since the 1930s, the subject of formal language theory has been developed by computer scientists, linguists and mathematicians.

Formal languages are sets of strings over finite sets of symbols, called alphabets,

Background

Since the 1930s, the subject of formal language theory has been developed by computer scientists, linguists and mathematicians.

Formal languages are sets of strings over finite sets of symbols, called alphabets, and languages can be described by

- regular expressions (which "generate" languages),
- finite automata (which "accept" languages),
- grammars (which "generate" languages) and
- Turing machines (which "accept" languages).

Background

Since the 1930s, the subject of formal language theory has been developed by computer scientists, linguists and mathematicians.

Formal languages are sets of strings over finite sets of symbols, called alphabets, and languages can be described by

- regular expressions (which "generate" languages),
- finite automata (which "accept" languages),
- grammars (which "generate" languages) and
- Turing machines (which "accept" languages).

The set of identifiers of a programming language is a formal language—one that can be described by a regular expression or a finite automaton.

Background

Since the 1930s, the subject of formal language theory has been developed by computer scientists, linguists and mathematicians.

Formal languages are sets of strings over finite sets of symbols, called alphabets, and languages can be described by

- regular expressions (which "generate" languages),
- finite automata (which "accept" languages),
- grammars (which "generate" languages) and
- Turing machines (which "accept" languages).

The set of identifiers of a programming language is a formal language—one that can be described by a regular expression or a finite automaton.

The set of all strings of tokens that are generated by a programming language's grammar is another example of a formal language.

Background (Cont.)

Because of its applications, computer science programs typically offer courses in this subject.

Background (Cont.)

Because of its applications, computer science programs typically offer courses in this subject.

- Applications to compiler construction:
  - Regular expressions and finite automata used when specifying and implementing lexical analyzers;
  - Grammars used to specify and implement parsers.

Background (Cont.)

Because of its applications, computer science programs typically offer courses in this subject.

- Applications to compiler construction:
  - Regular expressions and finite automata used when specifying and implementing lexical analyzers;
  - Grammars used to specify and implement parsers.
- Finite automata used when designing hardware and network protocols.

Background (Cont.)

Because of its applications, computer science programs typically offer courses in this subject.

- Applications to compiler construction:
  - Regular expressions and finite automata used when specifying and implementing lexical analyzers;
  - Grammars used to specify and implement parsers.
- Finite automata used when designing hardware and network protocols.
- Turing machines used to formalize the notion of algorithm—enabling study of what is, and is not, computable.

Background (Cont.)

Formal language theory largely concerned with algorithms, both ones that are explicitly presented, and ones implicit in theorems that are proved constructively.

Background (Cont.)

Formal language theory largely concerned with algorithms, both ones that are explicitly presented, and ones implicit in theorems that are proved constructively.

Typically, students apply these algorithms to toy examples by hand, and learn how they are used in applications.

Background (Cont.)

Formal language theory largely concerned with algorithms, both ones that are explicitly presented, and ones implicit in theorems that are proved constructively.

Typically, students apply these algorithms to toy examples by hand, and learn how they are used in applications.

Students would obtain a deeper understanding if they could experiment with the algorithms using computer tools.

Background (Cont.)

Consider, e.g., an exercise in which students are asked to synthesize a deterministic finite automaton that accepts some language, L.

Background (Cont.)

Consider, e.g., an exercise in which students are asked to synthesize a deterministic finite automaton that accepts some language, L.

- Conventionally, a student builds machine by hand, then proves it correct.

Background (Cont.)

Consider, e.g., an exercise in which students are asked to synthesize a deterministic finite automaton that accepts some language, L.

- Conventionally, a student builds machine by hand, then proves it correct.
- But given right computer tools, another approach possible.

Background (Cont.)

Consider, e.g., an exercise in which students are asked to synthesize a deterministic finite automaton that accepts some language, L.

- Conventionally, a student builds machine by hand, then proves it correct.
- But given right computer tools, another approach possible.
- First, express L in terms of simpler languages, making use of various language operations.

Background (Cont.)

Consider, e.g., an exercise in which students are asked to synthesize a deterministic finite automaton that accepts some language, L.

- Conventionally, a student builds machine by hand, then proves it correct.
- But given right computer tools, another approach possible.
- First, express L in terms of simpler languages, making use of various language operations.
- Next synthesize automata for those languages, and combine machines using corresponding operations.

Background (Cont.)

Consider, e.g., an exercise in which students are asked to synthesize a deterministic finite automaton that accepts some language, L.

- Conventionally, a student builds machine by hand, then proves it correct.
- But given right computer tools, another approach possible.
- First, express L in terms of simpler languages, making use of various language operations.
- Next synthesize automata for those languages, and combine machines using corresponding operations.
- Finally, minimize resulting machine.

Integrating Experimentation and Proof

To support experimentation with formal languages, I designed and implemented a computer toolset called Forlan.

Integrating Experimentation and Proof

To support experimentation with formal languages, I designed and implemented a computer toolset called Forlan.

Forlan is implemented in the functional programming language Standard ML (SML), a language whose notation and concepts are similar to those of mathematics.

Integrating Experimentation and Proof

To support experimentation with formal languages, I designed and implemented a computer toolset called Forlan.

Forlan is implemented in the functional programming language Standard ML (SML), a language whose notation and concepts are similar to those of mathematics.

Forlan is a library on top of the Standard ML of New Jersey (SML/NJ) implementation of SML.

Integrating Experimentation and Proof

To support experimentation with formal languages, I designed and implemented a computer toolset called Forlan.

Forlan is implemented in the functional programming language Standard ML (SML), a language whose notation and concepts are similar to those of mathematics.

Forlan is a library on top of the Standard ML of New Jersey (SML/NJ) implementation of SML.

It's used interactively, and users are able to extend Forlan by defining SML functions.

Integrating Experimentation and Proof (Cont.)

In Forlan, the objects of formal language theory—finite automata, regular expressions, etc.—are defined as abstract types, and have concrete syntax.

Integrating Experimentation and Proof (Cont.)

In Forlan, the objects of formal language theory—finite automata, regular expressions, etc.—are defined as abstract types, and have concrete syntax.

Instead of Turing machines, Forlan implements a simple functional programming language of equivalent power but that's easier to program in.

Integrating Experimentation and Proof (Cont.)

In Forlan, the objects of formal language theory—finite automata, regular expressions, etc.—are defined as abstract types, and have concrete syntax.

Instead of Turing machines, Forlan implements a simple functional programming language of equivalent power but that's easier to program in.

Forlan includes the Java program JForlan, a graphical editor for finite automata and regular expression, parse and program trees.

Integrating Experimentation and Proof (Cont.)

Forlan implements:

- conversions between regular expressions and different kinds of automata,

Integrating Experimentation and Proof (Cont.)

Forlan implements:

- conversions between regular expressions and different kinds of automata,
- the usual operations (e.g., union) on regular expressions, automata and grammars,

Integrating Experimentation and Proof (Cont.)

Forlan implements:

- conversions between regular expressions and different kinds of automata,
- the usual operations (e.g., union) on regular expressions, automata and grammars,
- equivalence testing and minimization of deterministic finite automata,

Integrating Experimentation and Proof (Cont.)

Forlan implements:

- conversions between regular expressions and different kinds of automata,
- the usual operations (e.g., union) on regular expressions, automata and grammars,
- equivalence testing and minimization of deterministic finite automata,
- a general parser for grammars,

Integrating Experimentation and Proof (Cont.)

Forlan implements:

- conversions between regular expressions and different kinds of automata,
- the usual operations (e.g., union) on regular expressions, automata and grammars,
- equivalence testing and minimization of deterministic finite automata,
- a general parser for grammars,
- tentative algorithms for simplifying regular expressions,

Integrating Experimentation and Proof (Cont.)

Forlan implements:

- conversions between regular expressions and different kinds of automata,
- the usual operations (e.g., union) on regular expressions, automata and grammars,
- equivalence testing and minimization of deterministic finite automata,
- a general parser for grammars,
- tentative algorithms for simplifying regular expressions,
- the functional programming language used instead of Turing machines.

Integrating Experimentation and Proof (Cont.)

The textbook on formal language theory *Formal Language Theory: Integrating Experimentation and Proof* is based on Forlan.

Integrating Experimentation and Proof (Cont.)

The textbook on formal language theory *Formal Language Theory: Integrating Experimentation and Proof* is based on Forlan.

I have attempted to keep the conceptual and notational distance between the textbook and toolset as small as possible.

Integrating Experimentation and Proof (Cont.)

The textbook on formal language theory *Formal Language Theory: Integrating Experimentation and Proof* is based on Forlan.

I have attempted to keep the conceptual and notational distance between the textbook and toolset as small as possible.

Book treats concepts and algorithms both theoretically, especially using proof, and through experimentation, using Forlan.

Integrating Experimentation and Proof (Cont.)

The textbook on formal language theory *Formal Language Theory: Integrating Experimentation and Proof* is based on Forlan.

I have attempted to keep the conceptual and notational distance between the textbook and toolset as small as possible.

Book treats concepts and algorithms both theoretically, especially using proof, and through experimentation, using Forlan.

Book provides numerous, fully worked-out examples of how regular expressions, finite automata, grammars and programs can be designed and proved correct.

Integrating Experimentation and Proof (Cont.)

The textbook on formal language theory *Formal Language Theory: Integrating Experimentation and Proof* is based on Forlan.

I have attempted to keep the conceptual and notational distance between the textbook and toolset as small as possible.

Book treats concepts and algorithms both theoretically, especially using proof, and through experimentation, using Forlan.

Book provides numerous, fully worked-out examples of how regular expressions, finite automata, grammars and programs can be designed and proved correct.

I have tried to simplify the subject's foundations, using alternative definitions when needed.

Integrating Experimentation and Proof (Cont.)

Readers of the book are assumed to have significant experience reading and writing informal mathematical proofs.

Integrating Experimentation and Proof (Cont.)

Readers of the book are assumed to have significant experience reading and writing informal mathematical proofs.

The book assumes no previous knowledge of Standard ML.

Integrating Experimentation and Proof (Cont.)

Readers of the book are assumed to have significant experience reading and writing informal mathematical proofs.

The book assumes no previous knowledge of Standard ML.

Drafts of the book have been successfully used at Kansas State University and Boston University.

Outline of the Book

Book consists of five chapters:

- **Chapter 1: Mathematical Background** Set theory, induction and recursion, trees and inductive definitions.

Outline of the Book

Book consists of five chapters:

- **Chapter 1: Mathematical Background** Set theory, induction and recursion, trees and inductive definitions.
- **Chapter 2: Formal Languages** Symbols, strings, alphabets, and (formal) languages. Proving language equalities using induction principles. Introduction to Forlan.

Outline of the Book

Book consists of five chapters:

- **Chapter 1: Mathematical Background** Set theory, induction and recursion, trees and inductive definitions.
- **Chapter 2: Formal Languages** Symbols, strings, alphabets, and (formal) languages. Proving language equalities using induction principles. Introduction to Forlan.
- **Chapter 3: Regular Languages** Regular expressions and languages, five kinds of finite automata, algorithms for processing and converting between regular expressions and finite automata, applications of regular expressions and finite automata to hardware design, searching in text files and lexical analysis.

Outline of the Book (Cont.)

- **Chapter 4: Context-free Languages** Context-free grammars and languages, algorithms for processing grammars and for converting regular expressions and finite automata to grammars, and recursive-descent (top-down) parsing.

Outline of the Book (Cont.)

- **Chapter 4: Context-free Languages** Context-free grammars and languages, algorithms for processing grammars and for converting regular expressions and finite automata to grammars, and recursive-descent (top-down) parsing.
- **Chapter 5: Recursive and Recursively Enumerable Languages** A functional programming language, and the recursive and recursively enumerable languages, which are defined using programs. Algorithms for processing programs and for converting grammars to programs. Problems, like the halting problem (the problem of determining whether a program halts when run on a given input), that can't be solved by programs.