Introduction to
Formal Languages

Martin Fränzle

Informatics and Mathematical Modelling
The Technical University of Denmark
Formal Languages

What is it?
A language is a set of “legal” sentences.
(Formal) Languages

- A language is a set of “legal” sentences.
- A sentence is a sequence of symbols.
- The symbols can be characters, words, punctuation, hieroglyphs, dots and dashes (Morse code), ...
(Formal) Languages

- A language is a set of “legal” sentences.
- A sentence is a sequence of symbols.
- The symbols can be characters, words + punctuation, hieroglyphs, dots and dashes (Morse code), ... 
- A formal language is a language defined by a finite set of unambiguous rules delimiting the legal sentences from the illegal ones.
Why are they interesting?

Because formal languages arise everywhere in computing:

- **Programming languages, query languages, etc.**
- **The inputs expected by programs:**
  - inputs to be keyed in, e.g. numbers in certain format, names, etc.
  - sequences of button clicks in a GUI.
- **Protocols for data exchange between computers** (e.g. SMTP)
Why are they interesting?

Because formal languages arise everywhere in computing:

- **Programming languages, query languages, etc.**
- **The inputs expected by programs:**
  - inputs to be keyed in, e.g. numbers in certain format, names, etc.
  - sequences of button clicks in a GUI.
- **Protocols for data exchange between computers** (e.g. SMTP)

Knowledge of their algorithmic properties permits automatic generation of programs for their manipulation.
Some means of defining formal languages

A finite and unambiguous set of rules can be given by

1. a “computer” with finite memory such that the whole machine can be described finitely,
Some means of defining formal languages

A finite and unambiguous set of rules can be given by

1. a “computer” with finite memory such that the whole machine can be described finitely,

2. a finite set of recursive definitions of the form
   - if \( c \) is a natural number then \( c \) is an arithmetic expression,
   - \ldots,
   - if \( e_1 \) and \( e_2 \) are arithmetic expressions then \( e_1 + e_2 \) is an arithmetic expression,
   - \ldots
Some means of defining formal languages

A finite and unambiguous set of rules can be given by

1. a “computer” with finite memory such that the whole machine can be described finitely,

2. a finite set of recursive definitions of the form
   - if $c$ is a natural number then $c$ is an arithmetic expression,
   - ... ,
   - if $e_1$ and $e_2$ are arithmetic expressions then $e_1 + e_2$ is an arithmetic expression,
   - ...

3. ...
Some means of defining formal languages

A finite and unambiguous set of rules can be given by

1. a “computer” with finite memory such that the whole machine can be described finitely,

   “Regular languages”

2. a finite set of recursive definitions of the form
   - if $c$ is a natural number then $c$ is an arithmetic expression,
   - ..., 
   - if $e_1$ and $e_2$ are arithmetic expressions then $e_1 + e_2$ is an arithmetic expression,
   - ...

   “Context-free languages”

3. ...
Regular languages:

Application domains
Regular languages

arise naturally in

• descriptions of **hardware components**, 

• descriptions of **communication protocols**, 

• **word classification problems**, e.g.: 
  is the next word a hyperlink, or is it to be rendered as text? 

• descriptions of **embedded computer systems**, 


Regular languages

arise naturally in

- descriptions of **hardware components**, because these are computational devices with finite memory,
- descriptions of **communication protocols**, because these engage into an alternation of finitely many different phases,
- **word classification problems**, e.g.: is the next word a hyperlink, or is it to be rendered as text?
- descriptions of **embedded computer systems**, because these are computers with finite memory.
Finite automata as models:

Embedded computer systems
The coffee vending machine — architecture

Vending machine

Financial administr.

Brewer control

cout

coin

canc

req

caf
The coffee vending machine — dynamics

Financial administration

- No pay
  - \((*, *, *, *, \text{caf})\)
  - \((-\text{canc}, *, *, *, \text{caf})\)
  - \((-\text{canc}, *, *, *, \text{caf})\)
  - \((-\text{cin}, *, *, *, \text{caf})\)

- Paid
  - \((*, *, *, *, \text{caf})\)
  - \((\text{cin}, *, *, *, \text{caf})\)
  - \((*, \text{canc}, *, *, \text{caf})\)

Brewer control

- Idle
  - \((-\text{cin}, *, *, *, \text{caf})\)

- Wait
  - \((\text{cin}, *, *, *, \text{caf})\)
  - \((*, \text{canc}, *, *, \text{caf})\)

- Brew
  - \((*, *, *, \text{req}, \text{caf})\)
  - \((*, *, *, \text{req}, \text{caf})\)
An example run

Financial administration

- \((*, *, \neg cout, *, caf)\)
- \((*, \neg canc, *, \neg req, \neg caf)\)
- \((*, \neg canc, \neg cout, *, \neg caf)\)
- \((*, *, *, *, caf)\)

Brewer control

- \((*, *, *, *, caf)\)
- \((*, *, *, *, \neg caf)\)
- \((*, canc, *, \neg req, \neg caf)\)
- \((*, \neg canc, *, \neg req, \neg caf)\)
An example run

Financial administration

- Brewidle wait
  - no pay
  - (*, *, *, req, −caf)(cin, *, *, *, −caf)
  - (*, −canc, *, −req, −caf)
  - (*, canc, *, −req, −caf)
  - (*, *, *, *, caf)

Brewer control

- Brewidle wait
  - no pay
  - (*, *, *, req, −caf)
  - (*, −canc, *, −req, −caf)
  - (*, *, *, *, caf)

Financial administration

- Brewidle wait
  - no pay
  - (*, *, *, req, −caf)
  - (*, −canc, *, −req, −caf)
  - (*, *, *, *, caf)

Brewer control

- Brewidle wait
  - no pay
  - (*, *, *, *, caf)
An example run

Financial administration

Brewer control

02140 Languages and Parsing, MF, Fall 2003 – p.11/16
An example run

Financial administration

no pay

paid

Brewer control

idle

wait

brew

02140 Languages and Parsing, MF, Fall 2003 – p.11/16
An example run

Financial administration

Brewer control

02140 Languages and Parsing, MF, Fall 2003 – p.11/16
An example run

Financial administration

Brewer control

idle

wait

brew
An example run

**Financial administration**

- `(*, *, −cout, *, caf)
- `(*, canc, −cout, *, −caf)
- `(*, canc, *, −req, −caf)
- `(*, *, *, *, caf)

**Brewer control**

- `(*, −canc, *, −req, −caf)
- `(*, *, *, *, caf)
- `(*, −canc, *, −req, −caf)
- `(*, *, *, *, caf)
An example run

Financial administration

- 

Brewer control

- 

02140 Languages and Parsing, MF, Fall 2003 – p.11/16
The coffee vending machine


idle
no pay

wait
no pay

brew
no pay

paid

idle
paid

wait
paid

Free coffee!

Unreachable states!
The coffee vending machine


Unreachable states!
Free coffee!
What you’ll learn

1. The model of finite automata,
2. methods for expressing properties like “free coffee”, namely regular expressions,
3. algorithms for manipulating regular languages,
4. how to plug these algorithms together such that they automatically check properties like “free coffee”.

What you’ll learn

1. The model of finite automata,
2. methods for expressing properties like “free coffee”, namely regular expressions,
3. algorithms for manipulating regular languages,
4. how to plug these algorithms together such that they automatically check properties like “free coffee”.

I.e., you will build programs which analyze programs!
Embedded systems in-the-large
Context-free languages:

Application domains
Context-free languages arise naturally in

- in the definition of programming languages, query lang., etc.:
  - if \( c \) is a natural number then \( c \) is an arithmetic expression,
  - if \( e_1 \) and \( e_2 \) are arithmetic expressions then \( e_1 + e_2 \) is an arithmetic expression,
  - ...

- descriptions of expected document structures, e.g. DTDs in XML
Context-free languages

arise naturally in

• in the definition of programming languages, query lang., etc.:
  • if \( c \) is a natural number then \( c \) is an arithmetic expression,
  • if \( e_1 \) and \( e_2 \) are arithmetic expressions then \( e_1 + e_2 \) is an arithmetic expression,
  • ...

⇒ formal definition permits tool support by compiler-compilers,

• descriptions of expected document structures, e.g. DTDs in XML

⇒ formal def. facilitates conformance check and automatic decomposition.
Context-free languages

arise naturally in

- in the definition of *programming languages*, *query lang.*, etc.:
  - if $c$ is a natural number then $c$ is an arithmetic expression,
  - if $e_1$ and $e_2$ are arithmetic expressions then $e_1 + e_2$ is an arithmetic expression,
  - ...

$\Rightarrow$ formal definition permits tool support by compiler-compilers,

- descriptions of *expected document structures*, e.g. DTDs in XML

$\Rightarrow$ formal def. facilitates conformance check and automatic decomposition.

You’ll learn

- to formally define context-free languages,
- to use compiler-compilers based on CFLs.