
COMP20121 The Implementation and Power of Computer Languages *'Power' Part*

<http://www.cs.man.ac.uk/~petera/2121/index.html>

Peter Aczel

room: CS2.52, tel: 56155

email: `petera@cs.man.ac.uk`

Department of Computer Science, University of Manchester

Main Objectives

In this part of the course you will learn about

- the underlying principles of the techniques and tools used in the other part of the course; e.g. regular expressions, grammars, parsing programs
- and the power and limitations of computers and programming languages

Theory of Computation

- Roughly speaking all programming languages have the same power.
- Some problems cannot be handled in any programming language.

Introduction 1

What is a computation?

Introduction 1

What is a computation?

What can be computed?

Introduction 1

What is a computation?

What can be computed?

What cannot be computed?

Introduction 1

What is a computation?

What can be computed?

What cannot be computed?

Abstract machines:

- Finite state automata
- Pushdown automata
- Turing Machines

Introduction 2

Section 1: Regular Languages and Finite Automata.

Introduction 2

Section 1: Regular Languages and Finite Automata.

Section 2: Context-free Grammars
and
Pushdown Automata

Introduction 2

Section 1: Regular Languages and Finite Automata.

Section 2: Context-free Grammars
and
Pushdown Automata

Section 3: Turing machines

Introduction 2

Section 1: Regular Languages and Finite Automata.

Section 2: Context-free Grammars
and
Pushdown Automata

Section 3: Turing machines

Section 4: Computability

Resources

- Lectures

Resources

- Lectures
- Examples classes

Resources

- Lectures
- Textbooks
- Examples classes

Resources

- Lectures
- Textbooks
- Examples classes
- Web page:

<http://www.cs.man.ac.uk/~petera/2121/index.html>

Resources

- Lectures
- Examples classes
- Textbooks
- Web page:

<http://www.cs.man.ac.uk/~petera/2121/index.html>

- Notes: These
 - outline the material covered
 - define what is examinable
 - include the exercises

Resources

- Lectures
- Examples classes
- Textbooks
- Web page:

<http://www.cs.man.ac.uk/~petera/2121/index.html>

- Notes: These
 - outline the material covered
 - define what is examinable
 - include the exercises

Exercises: **DO THEM!**

Assessment

- labs (20% of total marks)

for the *other* part of the course

Assessment

- labs (20% of total marks)
for the *other* part of the course
- examples classes (10% of total marks) for *this* part of the course

Assessment

- labs (20% of total marks)
for the *other* part of the course
- examples classes (10% of total marks) for *this* part of the course
- exam (70% of total marks)
for *both* parts of the course

LECTURE ONE: Regular languages

- We focus on a simple syntax of **patterns** (regular expressions) such as $(ab)^*$ and the words that match a pattern; e.g. words *abab* and *ababab* match the pattern $(ab)^*$, but the word *aba* does not.

LECTURE ONE: Regular languages

- We focus on a simple syntax of **patterns** (**regular expressions**) such as $(ab)^*$ and the words that match a pattern; e.g. words *abab* and *ababab* match the pattern $(ab)^*$, but the word *aba* does not.
- The set $\{\epsilon, ab, abab, ababab, \dots\}$ is an example of a **regular language**.

Programs that use patterns

e.g.

egrep echoes each line in a file that matches a pattern.

lex produces a **lex analyser** which, when run, consumes text, performing actions depending on which patterns are matched.

A basic matching program

A program **bm** written by Pete Jinks: On the command

```
>bm <pattern> <word>
```

the response is

```
>yes (if <word> matches <pattern>)
```

```
>no (if <word> does not match <pattern>)
```

Alphabets

Definition 1 An *alphabet*, Σ , is a finite set of characters (primitive indecomposable symbols) called the *letters*.

Alphabets

Definition 1 An *alphabet*, Σ , is a finite set of characters (primitive indecomposable symbols) called the *letters*.

We typically use x , y and z for unspecified elements of an alphabet.

Alphabets

Definition 1 An *alphabet*, Σ , is a finite set of characters (primitive indecomposable symbols) called the *letters*.

We typically use x , y and z for unspecified elements of an alphabet. Some symbols have a reserved meaning. These may not make sense until later; e.g.

ϵ , ϵ , \emptyset , τ , $|$, $*$, $($, $)$, EOF, EOS, \perp .

Words, 1

Definition 2 A *word* over an alphabet Σ is a finite string $x_1x_2 \cdots x_n$ where the x_i are letters from Σ .

This includes the empty string ϵ .

(made up of 0 letters, so $n = 0$ in this case.)

Words,2

We use α , β , γ to refer to unspecified words over a given alphabet.

Words,2

We use α , β , γ to refer to unspecified words over a given alphabet.

- We will later be forced to come up with a way of marking the end of a word.
- We will use the symbol EOF.

So we sometimes write a word as

$$x_1 x_2 \cdots x_n \text{EOF.}$$

Words, 3

For example, $ababa$ is a word over the alphabet $\{a, b\}$.

Words, 3

For example, $aaba$ is a word over the alphabet $\{a, b\}$.

- It is also a word over the alphabet $\{a, b, c, d\}$!

Words, 3

For example, $abab$ is a word over the alphabet $\{a, b\}$.

- It is also a word over the alphabet $\{a, b, c, d\}$!
- It may also be written $abab$ EOF.

Concatenation, 1

One way of forming new words from existing ones is by **concatenation**, that is, sticking two words together. From

$$\alpha = x_1x_2 \cdots x_m \text{ and } \beta = y_1y_2 \cdots y_n$$

we thus get the word

$$\alpha\beta = x_1x_2 \cdots x_my_1y_2 \cdots y_n.$$

Concatenation,2

Note that concatenation with the empty word has no effect, that is

$$\epsilon\alpha = \alpha = \alpha\epsilon,$$

Concatenation,2

Note that concatenation with the empty word has no effect, that is

$$\epsilon\alpha = \alpha = \alpha\epsilon,$$

and so also

$$\alpha\epsilon\beta = \alpha\beta.$$

Languages, 1

Σ^* is the set of all words over Σ .

Languages, 1

Σ^* is the set of all words over Σ .

Definition 3 A *language* over an alphabet

Σ is any subset of the set Σ^* .

Languages, 1

Σ^* is the set of all words over Σ .

Definition 3 A *language* over an alphabet Σ is any subset of the set Σ^* .

The largest language over Σ is all of Σ^* .

The smallest one is the empty set \emptyset .

Languages,2

Note: Every word over an alphabet has finite length, but there are **infinitely** many words for any non-empty alphabet.

Languages,2

Note: Every word over an alphabet has finite length, but there are **infinitely** many words for any non-empty alphabet.

- So languages can be infinite; e.g. The language $\{a\}^*$ has the words $\epsilon, a, aa, aaa,$ etc ... in it.

Use of placeholders

We reserve certain symbols to be placeholders for certain things.

Use of placeholders

We reserve certain symbols to be placeholders for certain things.

Σ an alphabet

Use of placeholders

We reserve certain symbols to be placeholders for certain things.

Σ an alphabet

x, y, z letters from the alphabet Σ

Use of placeholders

We reserve certain symbols to be placeholders for certain things.

Σ an alphabet

x, y, z letters from the alphabet Σ

α, β, γ words over Σ

Use of placeholders

We reserve certain symbols to be placeholders for certain things.

Σ an alphabet

x, y, z letters from the alphabet Σ

α, β, γ words over Σ

L, L' languages over Σ

Use of placeholders

We reserve certain symbols to be placeholders for certain things.

Σ an alphabet

x, y, z letters from the alphabet Σ

α, β, γ words over Σ

L, L' languages over Σ

p, p' patterns

Creating new languages, 1

Given languages L_1 over Σ_1 and L_2 over Σ_2 we can define new languages over

$\Sigma_1 \cup \Sigma_2$:

$$L_1 \cup L_2 = \{\alpha \mid \alpha \in L_1 \text{ or } \alpha \in L_2\},$$

$$L_1 \cap L_2 = \{\alpha \mid \alpha \in L_1 \text{ and } \alpha \in L_2\},$$

$$L_1 - L_2 = \{\alpha \mid \alpha \in L_1 \text{ and } \alpha \notin L_2\},$$

Creating new languages,2

and also the language

$$L_1 \cdot L_2 = \{\alpha_1\alpha_2 \mid \alpha_1 \in L_1, \alpha_2 \in L_2\}.$$

Creating new languages,3

Also, given a language L , let

$$L^n = \{ \alpha_1 \cdots \alpha_n \mid \alpha_1, \dots, \alpha_n \in L \}$$

Creating new languages,3

Also, given a language L , let

$$L^n = \{ \alpha_1 \cdots \alpha_n \mid \alpha_1, \dots, \alpha_n \in L \}$$

$$L^* = L^0 \cup L^1 \cup \dots \cup L^n \cup \dots$$

Creating new languages,3

Also, given a language L , let

$$L^n = \{a_1 \cdots a_n \mid a_1, \dots, a_n \in L\}$$

$$L^* = L^0 \cup L^1 \cup \dots \cup L^n \cup \dots$$

$$L^R = \{x_n x_{n-1} \cdots x_1 \mid x_1 x_2 \cdots x_n \in L\}.$$

Describing languages

In order to describe a language, we could, of course, merely list all its words.

Describing languages

In order to describe a language, we could, of course, merely list all its words.

However, if the language in question is infinite, this is a problem!

Describing languages

In order to describe a language, we could, of course, merely list all its words.

However, if the language in question is infinite, this is a problem!

It would be nice to have a concise way to describe languages.

Patterns, 1

Definition 4 *Let Σ be an alphabet. A **pattern** or **regular expression over Σ** is any word over $\Sigma^{\text{pat}} = \Sigma \cup \{\emptyset, \epsilon, |, *, (,)\}$ generated by the following inductive definition.*

Patterns,2

Empty pattern The character \emptyset ;

Patterns,2

Empty pattern The character \emptyset ;

Empty word the character ϵ ;

Patterns,2

Empty pattern The character \emptyset ;

Empty word the character ϵ ;

Base case every letter from Σ ;

Patterns,3

If p_1 and p_2 are patterns then so are

Concatenation (p_1p_2) ;

Patterns,3

If p_1 and p_2 are patterns then so are

Concatenation (p_1p_2) ;

Alternative $(p_1|p_2)$;

Patterns,3

If p_1 and p_2 are patterns then so are

Concatenation (p_1p_2) ;

Alternative $(p_1|p_2)$;

If p is a pattern then so is

Kleene star (p^*) .

Parsing, 1

In order to make patterns easier to read,
we may **leave out some brackets.**

Parsing, 1

In order to make patterns easier to read,
we may **leave out some brackets.**

- Kleene star binds the strongest.

Parsing, 1

In order to make patterns easier to read, we may **leave out some brackets**.

- Kleene star binds the strongest.
- Concatenation binds next strongest. For example, the correct bracketing for $a*b|cda$ is

Parsing, 1

In order to make patterns easier to read, we may **leave out some brackets**.

- Kleene star binds the strongest.
- Concatenation binds next strongest. For

example, the correct bracketing for $a*b|cda$ is $((((a*)b)|(cda)))$.

Parsing,2

But $((((a*)b)|(cda)))$ still does not contain all brackets required in the definition!

Parsing,2

But $((((a*)b)|(cda)))$ still does not contain all brackets required in the definition!

- Since concatenation is clearly associative, we don't bother with the distinction between $(ab)c$ and $a(bc)$.

When does word α match pattern $p?$, 1

Definition 5

When does word α match pattern $p?$, 1

Definition 5

Empty word empty word ϵ matches
pattern ϵ ;

When does word α match pattern $p?$, 1

Definition 5

Empty word empty word ϵ matches
pattern ϵ ;

Base case letter x matches pattern x ;

When does word α match pattern p ?, 2

Definition 5 (continued)

Concatenation concatenation $(\alpha_1\alpha_2)$
matches (p_1p_2) when α_1
matches p_1 and α_2
matches p_2 ;

When does word α match pattern $p?$, 3

Definition 5 (continued)

Alternative word α matches $(p_1|p_2)$
when α matches p_1 or p_2
(or both);

When does word α match pattern $p?$, 4

Definition 5 (continued)

Kleene star concatenation

$\alpha_1\alpha_2 \cdots \alpha_n$ matches (q^*)

whenever each of $\alpha_1, \alpha_2,$

\dots, α_n match q ; this

includes the empty word ϵ ,

when $n = 0$.

Example for pattern matching

Consider the pattern $a*b|cda$, for example. Which of the following words match it?

<i>b</i>	<i>ab</i>	<i>aabb</i>	<i>cd</i>	<i>abcda</i>	<i>aab</i>	<i>aaab</i>	<i>cda</i>	<i>abc</i>

Example for pattern matching

Consider the pattern $a*b|cda$, for example. Which of the following words match it?

<i>b</i>	<i>ab</i>	<i>aabb</i>	<i>cd</i>	<i>abcda</i>	<i>aab</i>	<i>aaab</i>	<i>cda</i>	<i>abc</i>
✓								

Example for pattern matching

Consider the pattern $a*b|cda$, for example. Which of the following words match it?

<i>b</i>	<i>ab</i>	<i>aabb</i>	<i>cd</i>	<i>abcda</i>	<i>aab</i>	<i>aaab</i>	<i>cda</i>	<i>abc</i>
✓	✓							

Example for pattern matching

Consider the pattern $a*b|cda$, for example. Which of the following words match it?

<i>b</i>	<i>ab</i>	<i>aabb</i>	<i>cd</i>	<i>abcda</i>	<i>aab</i>	<i>aaab</i>	<i>cda</i>	<i>abc</i>
✓	✓	X						

Example for pattern matching

Consider the pattern $a*b|cda$, for example. Which of the following words match it?

<i>b</i>	<i>ab</i>	<i>aabb</i>	<i>cd</i>	<i>abcda</i>	<i>aab</i>	<i>aaab</i>	<i>cda</i>	<i>abc</i>
✓	✓	X	X					

Example for pattern matching

Consider the pattern $a*b|cda$, for example. Which of the following words match it?

<i>b</i>	<i>ab</i>	<i>aabb</i>	<i>cd</i>	<i>abcda</i>	<i>aab</i>	<i>aaab</i>	<i>cda</i>	<i>abc</i>
✓	✓	X	X	X				

Example for pattern matching

Consider the pattern $a*b|cda$, for example. Which of the following words match it?

<i>b</i>	<i>ab</i>	<i>aabb</i>	<i>cd</i>	<i>abcda</i>	<i>aab</i>	<i>aaab</i>	<i>cda</i>	<i>abc</i>
✓	✓	X	X	X	✓			

Example for pattern matching

Consider the pattern $a*b|cda$, for example. Which of the following words match it?

<i>b</i>	<i>ab</i>	<i>aabb</i>	<i>cd</i>	<i>abcda</i>	<i>aab</i>	<i>aaab</i>	<i>cda</i>	<i>abc</i>
✓	✓	X	X	X	✓	✓		

Example for pattern matching

Consider the pattern $a*b|cda$, for example. Which of the following words match it?

<i>b</i>	<i>ab</i>	<i>aabb</i>	<i>cd</i>	<i>abcda</i>	<i>aab</i>	<i>aaab</i>	<i>cda</i>	<i>abc</i>
✓	✓	X	X	X	✓	✓	✓	

Example for pattern matching

Consider the pattern $a*b|cda$, for example. Which of the following words match it?

<i>b</i>	<i>ab</i>	<i>aabb</i>	<i>cd</i>	<i>abcda</i>	<i>aab</i>	<i>aaab</i>	<i>cda</i>	<i>abc</i>
✓	✓	X	X	X	✓	✓	✓	X

Regular languages

Every pattern p defines the language

$$L(p) = \{\alpha \in \Sigma^* \mid \alpha \text{ matches } p\}.$$

Regular languages

Every pattern p defines the language

$$L(p) = \{\alpha \in \Sigma^* \mid \alpha \text{ matches } p\}.$$

Definition 6 *A language L is **regular** if it is the set of all words matching some regular expression, that is, if there is a pattern p such that $L = L(p)$.*

Some languages defined by patterns

Here are some examples for languages defined by patterns.

Some languages defined by patterns

Here are some examples for languages defined by patterns.

- $L(\emptyset) = \emptyset$;

Some languages defined by patterns

Here are some examples for languages defined by patterns.

- $L(\emptyset) = \emptyset$;
- $L(\epsilon) = \{\epsilon\}$;

Some languages defined by patterns

Here are some examples for languages defined by patterns.

- $L(\emptyset) = \emptyset$;
- $L(\epsilon) = \{\epsilon\}$;
- $L(p_1|p_2) =$

Some languages defined by patterns

Here are some examples for languages defined by patterns.

- $L(\emptyset) = \emptyset$; • $L(\epsilon) = \{\epsilon\}$;
- $L(p_1|p_2) = L(p_1) \cup L(p_2)$;

Some languages defined by patterns

Here are some examples for languages defined by patterns.

- $L(\emptyset) = \emptyset$; • $L(\epsilon) = \{\epsilon\}$;
- $L(p_1|p_2) = L(p_1) \cup L(p_2)$;
- $L(p_1p_2) =$

Some languages defined by patterns

Here are some examples for languages defined by patterns.

- $L(\emptyset) = \emptyset$; • $L(\epsilon) = \{\epsilon\}$;
- $L(p_1|p_2) = L(p_1) \cup L(p_2)$;
- $L(p_1p_2) = L(p_1) \cdot L(p_2)$;

Some languages defined by patterns

Here are some examples for languages defined by patterns.

- $L(\emptyset) = \emptyset$; • $L(\epsilon) = \{\epsilon\}$;
- $L(p_1|p_2) = L(p_1) \cup L(p_2)$;
- $L(p_1p_2) = L(p_1) \cdot L(p_2)$;
- $L(p^*) =$

Some languages defined by patterns

Here are some examples for languages defined by patterns.

- $L(\emptyset) = \emptyset$; • $L(\epsilon) = \{\epsilon\}$;
- $L(p_1|p_2) = L(p_1) \cup L(p_2)$;
- $L(p_1p_2) = L(p_1) \cdot L(p_2)$;
- $L(p^*) = L(p)^*$.