University of Portsmouth
BSc (Hons) Computer Science
Third Year

Theoretical Computer Science (THEOC)

M21276

September 2025 - January 2026

20 Credits

Thomas Boxall
thomas.boxall1@myport.ac.uk

Contents

Page 1

Lecture - A1: Introduction to Languages

2025-09-29 14:00 Janka

There are two useful decks of slides on Moodle: Introduction to THEOCs and Overview of THEOCs. There is also a *Worksheet 0* which recaps the key concepts from year 1's Architecture & Operating Systems (Maths) and 2nd year's Discrete Maths and Functional Programming.

1.1 Introduction

Languages are a system of communication. The languages we commonly use are built for communicating and passing along instructions to other humans or computers. Depending on the context in which a language is used, will vary the precision which must exist within the language. For example, a language to convey "pub tonight?" to a friend can be as simple as that, where the human can add context clues to fill in the blanks; however to convey `print('hello, world')` to a computer - the language must be precise as it is not designed to interpret sloppy writing.

Languages are defined in terms of the set of symbols (called it's *alphabet*), which get combined into acceptable *strings*, which happens based on rules of sensible combination called *grammar*.

We can take this definition and see it in practice for the English Language:

Alphabet The alphabet for the English Language is Latin: $A = \{a, b, c, d, e, \ldots, x, y, z\}$

Strings (words) Strings are formed from A, for example 'fun', 'mathematics'. The English vocabulary defines which are really strings (for example which appear in the Oxford English Dictionary)

Grammar From the collection of words, we can build sentences using the English rules of *grammar*

Language The set of possible sentences that make up the English Language

Whilst this is an example around a tangible, understandable example - the elements of a formal language are exactly the same however they must be defined without any ambiguity. For example, programming languages have to be defined with a precise description of the syntax used.

1.2 Formalising Language Definitions

Definitions

Alphabet A finite, nonempty set of symbols. For example: $\Sigma = \{a, b, c\}$
String A finite sequence of symbols from the alphabet (placed next to each other in juxtaposition). For example: abc, aaa, bb are examples of strings on Σ
Empty String A string which has no symbols (therefore zero length), denoted Λ.
Language Where Σ is an alphabet, then a language over Σ is a set of strings (including empty string Λ) whose symbols come from Σ

For example, if $\Sigma = \{a, b\}$, then $L = \{ab, aaab, abbb, a\}$ is an example of a language over Σ.

Languages are not finite and they may or may not contain an empty string.

If Σ is an alphabet, then Σ^* denotes the infinite set of all strings made up from Σ - including an empty string. For example, if $\Sigma = \{a, b\}$ then $\Sigma^* = \{\Lambda, a, b, ab, aab, aaab, bba, \ldots\}$. We can therefore say that when looking at Σ^*, a language over Σ is any subset of Σ^*.

Example: Languages

For a given alphabet, Σ, it is possible to have multiple languages. For example:
- $\emptyset$ - an empty language
- $\{\Lambda\}$ - a language containing only an empty string (silly language)
- Σ - the alphabet itself
- Σ^* - the infinite set of all strings made up from the alphabet

Alternatively, we can make this slightly more tangible:
Where $\Sigma = \{a\}$:
- $\emptyset$
- $\{\Lambda\}$
- $\{a\}$
- $\{\Lambda, a, aa, aaa, aaaa, \ldots\}$

1.3 Combining Languages

It is possible to combine languages together to create a new language.

1.3.1 Union and Intersection

As languages are just sets of strings, we can use the standard set operations for Union and Intersection to combine the languages together.

Example: Union and Intersection

Where $L = \{aa, bb, ab\}$ and $M = \{ab, aabb\}$
Intersection (common elements between the two sets): $L \cap M = \{ab\}$
Union (all elements from each set): $L \cup M = \{aa, bb, ab, aabb\}$

1.3.2 Product

The product of two languages is based around concatenation of strings...

The operation of *concatenation of strings* places two strings in juxtaposition. For example, the concatenation of the two strings *aab* and *ba* is the string *aabba*. We use the name *cat* to denote this operation: cat(aab, ba) = aabba. We can combine two languages L and M by forming the set of all concatenations of strings in L with strings in M, which is called the product of two languages.

Definitions

Product of two languages If L and M are languages, then the new language called the product of L and M is defined as $L \cdot M$ (or just LM). This can be seen in set notation below:

$$L \cdot M = \{cat(s, t) : s \in L \text{ and } t \in M\}$$

The product of a language, L, with the language containing only an empty string returns L:

$$L \cdot \{\Lambda\} = \{\Lambda\} \cdot L = L$$

The product of a language, L, with an empty set returns an empty set:

$$L \cdot \emptyset = \emptyset \cdot L = \emptyset$$

The operation of concatenation is not commutative - meaning the order of the two languages matters. For two languages, it's usually true that:

$$L \cdot M \neq M \cdot L$$

Example: Commutativity Laws of Concatenation

For example, if we take two languages: $L = \{ab, ac\}$ and $M = \{a, bc, abc\}$

$$\begin{aligned} L \cdot M &= \{aba, abbc, ababc, aca, acbc, acabc\} \\ M \cdot L &= \{aab, aac, bcab, bcac, abcab, abcac\} \end{aligned}$$

They have no strings in common!

The operation of concatenation is associative. Which means that if L, M, and N are languages:

$$L \cdot (M \cdot N) = (L \cdot M) \cdot N$$

Example: Associativity Laws of Concatenation

For example, if $L = \{a, b\}$, $M = \{a, aa\}$ and $N = \{c, cd\}$ then:

$$\begin{aligned} L \cdot (M \cdot N) &= L \cdot \{ac, acd, aac, aacd\} \\ &= \{aac, aacd, aaac, aaacd, bac, bacd, baac, baacd\} \\ \text{which is the same as} & \\ (L \cdot M) \cdot N &= \{aa, aaa, ba, baa\} \cdot N \\ &= \{aac, aacd, aaac, aaacd, bac, bacd, baac, baacd\} \end{aligned}$$

1.3.3 Powers of a Language

For a language, L, the product $L \cdot L$ is denoted by L^2.

The language product L^n for ever $n \in \{0, 1, 2, \ldots\}$ is defined as follows:

$$\begin{aligned} L^0 &= \{\Lambda\} \\ L^n &= L \cdot L^{n-1}, \text{if } n > 0 \end{aligned}$$

Example: Powers of Languages

If we take $L = \{a, bb\}$:

$$\begin{aligned} L^0 &= \{\Lambda\} \\ L^1 &= L = \{a, bb\} \\ L^2 &= L \cdot L = \{aa, abb, bba, bbbb\} \\ L^3 &= L \cdot L^2 = \{aaa, aabb, abba, abbbb, bbaa, bbabb, bbbba, bbbbbb\} \end{aligned}$$

1.4 Closure of a Language

Attempt to extricate a better definition of closure out of Janka

The *closure* of a language is an operation which is applied to a language.

If L is a language over Σ (for example $L \subset \Sigma^*$) then the closure of L is the language denoted by L^* and is defined as follows:

$$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$$

The *Positive Closure* of L is the language denoted by L^+ and is defined as follows:

$$L^+ = L^1 \cup L^2 \cup L^3 \cup \ldots$$

So from this we can derive that $L^* = L^+ \cup \{\Lambda\}$. However - it's not necessarily true that $L^+ = L^* - \{\Lambda\}$.

For example, if we take our alphabet as $\Sigma = \{a\}$ and our language to be $L = \{\Lambda, a\}$ then $L^+ = L^*$.

Based on what we now know, there's some interesting properties of closure we can derive. Let L and M be languages over the alphabet Σ:

- $\{\Lambda\}^* = \emptyset^* = \{\Lambda\}$
- $L^* = L^* \cdot L^* = (L^*)^*$
- $\Lambda \in L$ if and only if $L^+ = L^*$
- $(L^* \cdot M^*)^* = (L^* \cup M^*)^* = (L \cup M)^*$
- $L \cdot (M \cdot L)^* = (L \cdot M)^* \cdot L$

These will be explored more in the coming Tutorials

1.5 Closure of an Alphabet

As we saw earlier, Σ^* is the infinite set of all strings made up from Σ. The closure of Σ coincides with our definition of Σ^* as the set of all strings over Σ. In other words, it is a nice representation of Σ^* as follows:

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$$

From this, we can see that Σ^k represents the set of strings of length k, for each their symbols are in Σ.

Page 2

Lecture - A2: Grammars

2025-09-29 15:00 Janka

As we saw in the previous lecture, languages can be defined through giving a set of strings or combining from the existing languages using operations such as productions, unions, etc. Alternatively, we can use a grammar to define a language.

To describe a grammar for a language - two collections of alphabets (symbols) are necessary.

Definitions

Terminal Symbols from which all strings in the language are made. They are symbols of a ‘given’ alphabet for generated language. Usually represented using lower case letters

Non-Terminal Temporary Symbols (different to terminals) used to define the grammar replacement rules within the production rules. They must be replaced by terminals before the production can successfully make a valid string of the language. Usually represented using upper case letters.

Now we know what terminals & non-terminals are - we need to know how to produce terminals from non-terminals. This is where the *Production Rules* come into play. Productions take the form:

$$\alpha \to \beta$$

where α and β are strings of symbols taken from the set of terminals and non-terminals.

A grammar rule can be read in any of several ways:

- “replace α by β”
- “α produces β”
- “α rewrites to β”
- “α reduces to β”

We can now define the grammar.

Definitions

Grammar A set of rules used to define a language - the structure of the strings in the language. There are four key components of a grammar:

1. An alphabet T of symbols called *terminals* which are identical to the alphabet of the resulting language
2. An alphabet N of grammar symbols called *non-terminals* which are used in the production rules
3. A specific non-terminal called the *start symbol* which is usually S
4. A finite set of *productions* of the form $\alpha \to \beta$ where α and β are strings over the alphabet $N \cup T$

2.1 Using a Grammar to Generate a Language

Every grammar has a special non-terminal symbol called a *start symbol* and there must be at least one production with left-side consisting of only the start symbol. Starting from the production rules with the start symbol, we can step-by-step generate all strings belonging to the language described by a given grammar.

As we begin converting from Non-Terminal to Terminal containing strings, we introduce the *Sentential Form*. As we continue to generate strings, we introduce *derivation*.

Definitions

Sentential Form A string made up of terminal and non-terminal symbols.
Derivation Where x and y are sentential forms and $\alpha \to \beta$ is a production, then the replacement of α with β in $x\alpha y$ is called a derivation. We denote this by writing:

$$x\alpha y \Rightarrow x\beta y$$

During our derivations, there are three symbols we may come across:

- $\Rightarrow$ derives in one step
- $\Rightarrow^+$ derives in one or more steps
- $\Rightarrow^*$ derives in zero or more steps

Finally, we can define $L(G)$: the Language defined by the given Grammar.

Definitions

$L(G)$ If G is a grammar with start symbol S and set of terminals T, then the language generated by G is the following set:

$$L(G) = \{s | s \in T^* and S \Rightarrow^+ s\}$$

Great - now we've seen the theory, lets put it into an example.

Example: Using a Grammar to Derive a Language

Let a grammar, G, be defined by:
- the set of terminals $T = \{a, b\}$
- the only non-terminal start symbol S
- the set of production rules: $S \to \Lambda, \quad S \to aSb$
  or in shorthand: $S \to \Lambda | aSb$

Now, beginning the derivations. We have to start with the start symbol S, and we can either derive Λ or aSb. Obviously deriving Λ would end the production and deriving aSb would allow us to keep re-using the production rules:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow \dots$$

Using a combination of the two production rules, we can build up a picture of what strings we can derive from the start symbol:

$$S \Rightarrow \Lambda, S \Rightarrow aSb \Rightarrow ab$$

The second string above we can turn into the following shorthand:

$$S \Rightarrow^* ab$$

Or alternatively, we can use shorthand to jump forward and yet continue the derivation:

$$S \Rightarrow^* aaaSbbb$$

This brings us to the end of the example as we can now define $L(G)$:

$$L(G) = \{\Lambda, ab, aabb, aaabbb, \dots\}$$

Example: Longer Derivation of a String

Let $\Sigma = \{a, b, c\}$ be the set of terminal symbols and S be the only non-terminal symbol. We have four production rules:

- $S \rightarrow \Lambda$
- $S \rightarrow aS$
- $S \rightarrow bS$
- $S \rightarrow cS$

Which can alternatively be represented in Shorthand: $S \rightarrow \Lambda|aS|bS|cS$
To derive the string *aacb* we would undergo the following derivation:

$$S \Rightarrow aS \Rightarrow aaS \Rightarrow aacS \Rightarrow aacbS \Rightarrow aacb\Lambda = aacb$$

Which can be shortened to $S \Rightarrow^* aacb$
Note how we started on the left and worked left-to-right. This makes this derivation a *leftmost* derivation because we produced the leftmost characters first.

2.2 Infinite Languages

As in the previous example, note how there is no bound on the length of the strings in an infinite language. Therefore there is no bound on the number of derivation steps used to derive the strings. If the grammar has n productions, then any derivation consisting of $n+1$ steps must use some production twice.

Where a language is infinite - some of the productions or sequence of productions must be used repeatedly to construct the derivations.

Example: Infinite language

Take the infinite language $\{a^n b | n \geq 0\}$ which can be described by the grammar $S \rightarrow b|aS$.
To derive the string $a^n b$, the production $S \rightarrow aS$ is used repeatedly, n times and then the derivation is stopped by using the production $S \rightarrow b$.

The production $S \rightarrow aS$ allows us to say "If S derives w, then it also derives aw".

2.3 Recursion / Indirect Recursion

A production is called recursive if its left side occurs on it's right side. For example the production $S \rightarrow aS$ is recursive.

A production $A \rightarrow \ldots$ is indirectly recursive if A derives a sentential form that contains A in two or more steps.

Example: Indirect Recursion

If the grammar contains the rules $S \rightarrow b|aA, A \rightarrow c|bS$ then both productions $S \rightarrow aA$ and $A \rightarrow bS$ are indirectly recursive:

$$S \Rightarrow aA \Rightarrow abS$$
$$A \Rightarrow bS \Rightarrow baA$$

A grammar can also be considered recursive where it contains either a recursive production or an indirectly recursive production. We can deduce from this that a grammar for an infinite language must be directly or indirectly recursive.

2.4 Constructing Grammars

Up to now, we've looked at deriving a language from a given grammar. Now we will take the inverse - be given a language and construct a grammar which derives the specified language.

Sometimes it is difficult or even impossible to write down a grammar for a given language. Unsurprisingly, a language may have more than one grammar which is correct and valid.

2.4.1 Finite Languages

If the number of strings in a language is finite, then a grammar can consist of all productions of the form $S \to w$ for each string w in the language.

Example: Finite Language

The finite language $\{a, ba\}$ can be described by the grammar:

$$S \to a|ba$$

2.4.2 Infinite Languages

To find the grammar for a language where the number of strings is infinite is a considerably bigger challenge. There is no universal method for finding a grammar for an infinite language, however the method of *combining grammars* can prove useful.

Example: Infinite Language

To find a grammar for the following simple language:

$$\{\Lambda, a, aa, \ldots, a^n, \ldots\} = \{a^n : n \in \mathbb{N}\}$$

We can use the following solution:
- We know the set of terminals: $T = \{a\}$
- We know the only non-terminal start symbol: S
- So therefore we can generate the production rules: $S \to \Lambda, S \to aS$

2.5 Combining Grammars

If we take L and M to be languages which we are able to find the grammars; then there exist simple rules for creating grammars which produce the languages $L \cup M$, $L \cdot M$, and L^*. This therefore means we can describe L and M with grammars having disjoint sets (where neither set has common elements) of non-terminals.

The combination process is started by assigning start symbols for the grammars of L and M to be A and B respectively:

$$L : A \to \ldots, \quad M : B \to \ldots$$

2.5.1 Union Rule

The union of two languages, $L \cup M$ starts with the two productions:

$$S \to A|B$$

which is followed by: the grammar rules of L (start symbol A) and then the grammar rules of M (start symbol B).

Example: Combining Grammars Using Union Rule

If we take the following language:

$$K = \{\Lambda, a, b, aa, bb, aaa, bbb, \ldots, a^n, b^n, \ldots\}$$

Now to find the grammar for it.
Firstly we look at it and see quite clearly there is a pattern, K is a union of the two languages:

$$L = \{a^n | n \in \mathbb{N}\} \text{ and } M = \{b^n | n \in \mathbb{N}\}$$

Therefore we can write a grammar for K as follows:

- $A \to \Lambda | aA$ (grammar for L)
- $B \to \Lambda | bB$ (grammar for M)
- $S \to A | B$ (union rule)

2.5.2 Product Rule

Much the same as the Union Rule, the product of two languages, $L \cdot M$ starts with the production:

$$S \to AB$$

Which is then followed by: the grammar rules of L (start symbol A) and then the grammar rules of M (start symbol B).

Example: Combining Grammars Using Product Rule

If we take the following language:

$$\begin{aligned} K &= \{a^m b^n | m, n \in \mathbb{N}\} \\ &= \{\Lambda, a, b, aa, ab, aaa, bb\} \end{aligned}$$

We can first find out that K is the product of two languages:

$$L = \{a^n | n \in \mathbb{N}\} \text{ and } M = \{b^n | n \in \mathbb{N}\}$$

Therefore we can write a grammar for K as follows:

- $A \to \Lambda | aA$ (grammar for L)
- $B \to \Lambda | bB$ (grammar for M)
- $S \to AB$ (product rule)

2.5.3 Closure Rule

The grammar for the closure of a language, L^*, starts with the production:

$$S \to AS | \Lambda$$

Which is followed by: the grammar rules of L (start symbol A).

Example: Grammar Closure Rule

If we take the problem that we want to construct a language, L, of all possible strings made up from zero or more occurrences of aa or bb:

$$L = \{aa, bb\}^* = M^*$$

Where $M = aa, bb$

Therefore:

$$L = \{\Lambda, aa, bb, aaaa, aabb, bbbb, bbaa, \ldots\}$$

Therefore, we can write a grammar for L as follows:

- $S \to AS|\Lambda$ (closure rule)
- $A \to aa|bb$ (grammar for $\{aa, bb\}$)

2.6 Equivalent Grammars

Grammars are not unique; a given language can have many grammars which could produce it. Grammars can be simplified down to their simplest form.

Example: Simplifying Grammars

If we take the grammar from the previous example:

$$S \to AS|\Lambda, \quad A \to aa|bb$$

We can simplify this:

- Replace the occurrence of A in $S \to AS$ by the right side of $A \to aa$ to obtain the production $S \to aaS$
- Replace A in $S \to AS$ by the right side of $A \to bb$ to obtain the production $S \to bbS$

We can therefore write the grammar in simplified form as:

$$S \to aaS|bbS|\Lambda$$