



**BRNO UNIVERSITY OF TECHNOLOGY**

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

**FACULTY OF INFORMATION TECHNOLOGY**

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

**DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA**

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

# **AUTOMATIC TRANSLATION AND COMPARISON OF LUMERICAL AND MEEP SIMULATIONS**

AUTOMATICKÝ PŘEKLAD A SROVNÁNÍ SIMULACÍ MEZI MEEP A LUMERICAL

**BACHELOR'S THESIS**

BAKALÁŘSKÁ PRÁCE

**AUTHOR**

AUTOR PRÁCE

**DANIEL MAČURA**

**SUPERVISOR**

VEDOUCÍ PRÁCE

**Ing. TOMÁŠ MILET, Ph.D.**

**BRNO 2025**

---

---

## Abstract

This thesis aims to provide a comparison between Ansys Lumerical and Meep FDTD simulation tools. A transpiler from Lumerical Scripting Language to the Meep Python interface was implemented. FDTD simulation were first automatically results are compared with analytical results.

## Abstrakt

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v českém (slovenském) jazyce.

## Keywords

CEM, FDTD, Meep, Ansys Lumerical, transpiler,

## Klíčová slova

Sem budou zapsána jednotlivá klíčová slova v českém (slovenském) jazyce, oddělená čárkami.

## Reference

MAČURA, Daniel. *Automatic translation and comparison of Lumerical and Meep simulations*. Brno, 2025. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Tomáš Milet, Ph.D.

---

---

---

# Automatic translation and comparison of Lumerical and Meep simulations

## Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana X... Další informace mi poskytli... Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....  
Daniel Mačura  
December 22, 2024

## Acknowledgements

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant apod.).

---

---

# Contents

<b>1</b>	<b>Note to the Reader</b>	<b>3</b>
<b>2</b>	<b>Introduction</b>	<b>4</b>
<b>3</b>	<b>Review of Relevant Literature</b>	<b>5</b>
3.1	Computational Electromagnetics . . . . .	5
3.2	Formal Language Theory . . . . .	5
3.2.1	Alphabets and Languages . . . . .	5
3.2.2	Grammars . . . . .	6
3.2.3	Regular Languages . . . . .	8
3.3	Compilers . . . . .	8
<b>4</b>	<b>Transpiler Implementation</b>	<b>10</b>
<b>5</b>	<b>Evaluation of Results</b>	<b>11</b>
<b>6</b>	<b>Conclusion</b>	<b>12</b>
	<b>Bibliography</b>	<b>13</b>

---

---

# List of Figures

3.1	Description of Chomsky hierarchy. . . . .	8
3.2	Description of a compiler as a single unit. . . . .	9
3.3	Overview of relevant compiler stages. . . . .	9

---

---

# Chapter 1

## Note to the Reader

Through the thesis, the expression „iff“ refers to „if and only if“.

Except if explicitly stated otherwise, the following notation conventions apply throughout the thesis. In the context of regular languages, lower case latin alphabet letters, such as  $a, b, c, \dots$ , refer to terminal symbols. Upper case latin alphabet letters, such as  $A, B, C, \dots$ , denote nonterminal symbols.

---

---

## Chapter 2

# Introduction

[[Add motivation for thesis, talk about importance of optics simulations, talk about uses, show examples. Talk about the need to compare simulation software.]]

---

---

## Chapter 3

# Review of Relevant Literature

This chapter aims to give the reader a broad understanding of the subjects of matter. It provides an overview of the physics behind the aforementioned simulations and an introduction to the basic concepts of compiler design and the underlying formal language theory. This chapter will not delve into the specifics of each subject at hand but will provide the reader with the foundations necessary to comprehend the rest of this thesis.

### 3.1 Computational Electromagnetics

Curl theorem

Divergence theorem

Maxwell's equations

[[assuming magnetic monopoles dont exist]] [[show all 4 equations]] [[derive partial form from integral form]]

Finite Difference Time Domain

### 3.2 Formal Language Theory

We use languages like English or Slovak every day to communicate information, these languages are referred to as *natural languages*. The Oxford English Dictionary defines language as a system of spoken or written communication used by a particular country, people, community. However a more rigorous definition of languages and the tools to operate on them is required. That is what the notion of formal language theory will introduce in the following sections.

#### 3.2.1 Alphabets and Languages

[[add text about languages and alphabets, use English as example]]

**Definition 3.2.1** (Alphabet). Let  $\Sigma$  be an *alphabet*, a finite nonempty set of symbols, letters. Then  $\Sigma^*$  defines the set of all sequences  $w$ :

$$w = a_1 a_2 a_3 \dots a_{n-1} a_n, \in \Sigma \text{ for } n \in \mathbb{N}$$



The sequence of symbols  $w$  is called a *word*. The length of a word is given by the number of symbols  $a$ , symbolically  $|w| = n$ . The word with a length of 0 is called an *empty word* denoted as  $\epsilon$ .

**Definition 3.2.2** (Language). The set  $L$  where  $L \subseteq \Sigma^*$  is known as the *formal language* over the alphabet  $\Sigma$ .

The words  $L = \{\epsilon, a, b, aa, ab, bb\}$  are some words of a language  $L$  over the alphabet  $\Sigma = \{a, b\}$ . Other examples for languages over the alphabet  $\Sigma = \{a, b\}$  might include:

- $L_1 = \{\epsilon\}$
- $L_2 = \{a\}$
- $L_3 = \{aaa\}$
- $L_4 = \{a^i, b^i; i \in \mathbb{N}\}$

### Notation conventions

- Iteration

Let  $a^i$ ;  $a \in \Sigma$  and  $i \in \mathbb{Z}$  be the *iteration* of a character  $a$ , where  $|a^i| = i$ .

Example below:

- $a^0 = \epsilon$
- $a^1 = a$
- $a^2 = aa$
- $a^i = a_0a_1a_2 \dots a_i; i \in \mathbb{N}$

- Concatenation

Let  $w \cdot w'$ ;  $w, w' \in \Sigma^*$  be the *concatenation* of the words  $w$  and  $w'$ .

$w = a_1a_2a_3 \dots a_n; w' = a'_1a'_2a'_3 \dots a'_m; n, m \in \mathbb{Z}$  then

$w \cdot w' = ww' = a_1a_2a_3 \dots a_na'_1a'_2a'_3 \dots a'_m$

- **[[Kleene star]]**

- Symbol count

The number of occurrences of  $a$  in  $w$ , where  $a \in \Sigma, w \in \Sigma^*$ , noted as  $|w|_a$ .

### 3.2.2 Grammars

Linguists refer to grammar as a set of rules that specify how a natural language is formed. Due to the polysemous and vague nature of natural languages, it is not suited for the description of unambiguous systems. The inherent need to strictly describe languages introduced various language defining mechanisms. Many of these mechanisms are interchangeable, and may describe the same languages. However, not all mechanisms are able to describe languages formed by other mechanisms.

The concept of a grammar is a powerful tool for describing languages[1, p. 52]. A simple sentence in English consists of a single independent clause. A clause is typically formed by a subject and a predicate. This can be written as follows.

$\langle clause \rangle \rightarrow \langle subject \rangle \langle predicate \rangle$

We can further define the  $\langle subject \rangle$  and  $\langle predicate \rangle$ . One of the possible subjects is a noun phrase, and the predicate may be a simple verb.

$$\begin{aligned}\langle subject \rangle &\rightarrow \langle determiner \rangle \langle premodifier \rangle \langle noun \rangle \langle postmodifier \rangle \\ \langle predicate \rangle &\rightarrow \langle verb \rangle\end{aligned}$$

Associating the  $\langle determiner \rangle$  with the article „the“,  $\langle premodifier \rangle$  with „fast“ or „slow“,  $\langle noun \rangle$  with „athlete“,  $\langle postmodifier \rangle$  to „from England“ or to an empty string and finally associating the  $\langle verb \rangle$  to „won“ or „lost“ allows us to write multiple sentences like „The fast athlete from England won“ or „The slow athlete lost“. These sentences are considered to be *well formed*, as they resulted from following the grammatical rules.

The premise is consecutively replacing  $\langle clause \rangle$  until only irreducible blocks of the language remain. Generalizing this idea brings about the concept formal grammars.

**Definition 3.2.3** (Grammar). [2] Let an ordered quadruple  $G$  define a grammar such that:  $G = (N, \Sigma, P, S)$ , where:

1.  $N$  is a finite set of *nonterminal* symbols
2.  $\Sigma$  is an alphabet finite set of *terminal* symbols, such that  $N \cap \Sigma = \emptyset$
3.  $P$  is a finite set of rewriting rules known as *productions*, ordered pairs  $(\alpha, \beta)$ .  $P$  is a subset of the cartesian product of  $\alpha = (N \cup \Sigma)^* N (N \cup \Sigma)^*$  and  $\beta = (N \cup \Sigma)^*$

Productions are denoted as  $\alpha \rightarrow \beta$ . If there are multiple productions with the same left hand side ( $\alpha$ ), we can group their right hand sides ( $\beta$ ).

$$\alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2 \text{ may be written as } \alpha \rightarrow \beta_1 | \beta_2$$

4.  $S$  is the starting symbol of the grammar  $G$ , where  $S \in N$

Productions  $\alpha \rightarrow \beta$  symbolize, that given the words  $V, W, x, y \in (N \cup \Sigma)^*$ , the word  $V$  can be rewritten as follows:  $V \Rightarrow W$  iff there are words, which satisfy the following condition,  $V = x\alpha y \wedge W = x\beta y$  and  $\alpha \rightarrow \beta \in P$ .

**Definition 3.2.4** (Derivation).  $V \xRightarrow{*} W$  iff there is a finite set of words

$$v_0, v_1, v_2, \dots, v_z; z \in \mathbb{Z}$$

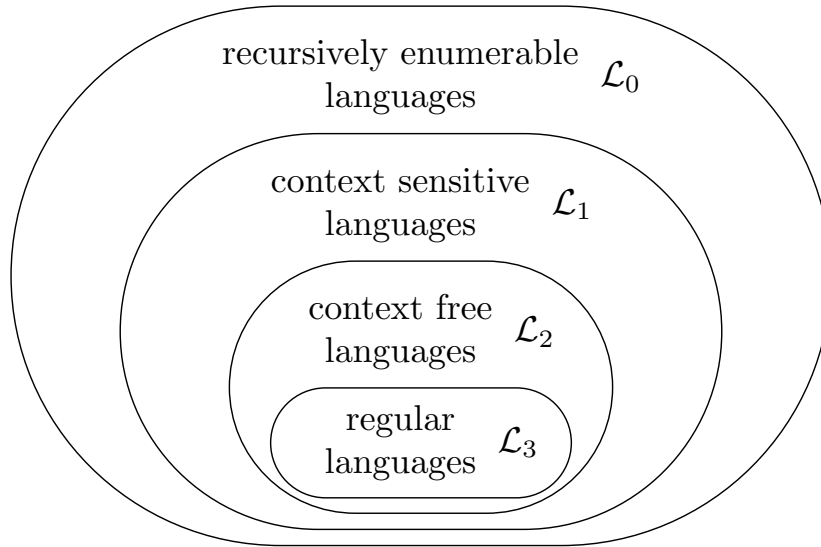
such that  $v_0 = V$  and  $v_z = W$  where each is rewritten from the previous word. Such a sequence of applications of productions is called a derivation. The length of a derivation is given by  $z$ .

Grammars are often represented using formalisms, these formalisms often set restrictions on the left and right hand sides of productions. These restrictions further impose limits on the set of languages a grammar may produce, this is known as the *expressive power* of a grammar.

**Chomsky hierarchy** When working with formal grammars, the need to compare their expressive power arose. Linguist Noam Chomsky introduced the now so called *Chomsky hierarchy*[3]. A set of four classes, each more expressive than the previous, see fig. 3.1.

The intricacies of each class are not necessary in the context of this thesis, how ever regular languages will be used heavily throughout the rest of this thesis, so they will be explained more in depth.

Figure 3.1: Description of Chomsky hierarchy.



Grammar	Language	Restrictions
$\mathcal{L}_3$	Regular	
$\mathcal{L}_2$	Context Free	
$\mathcal{L}_1$	Context Sensitive	
$\mathcal{L}_0$	Recursively Enumerable	

Table 3.1: An overview of the Chomsky hierarchy classes.

### 3.2.3 Regular Languages

#### Automata

#### Regular Expressions

[[show equivalence DFA = NFA =  $\mathcal{L}_3$ ]]

**Definition 3.2.5** (Finite State Machine).

**Definition 3.2.6** (Regular Expressions).

## 3.3 Compilers

Today's world undeniably relies on software more than ever before. It is imperative for developers to be able to efficiently write software. Software today is written in programming languages, high-level human-readable notations for defining how a program should run. However, before a program can be run on a system, it has to be translated (or compiled) into low-level machine code, which computers can run. The computer program that facilitates this translation is called a *compiler*. See fig. 3.2.

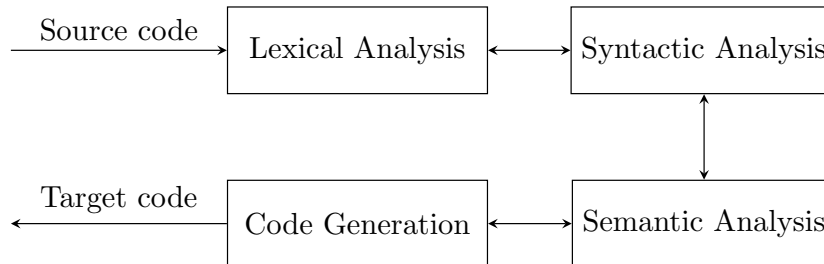
Compilers are complex programs. It is helpful to break them down into parts, each handling different tasks, which are chained together to form a compiler. Modern compilers

Figure 3.2: Description of a compiler as a single unit.



are composed of many phases, such as the Lexical Analyzer, Syntax Analyzer, Semantic Analyzer, Intermediate Code Generator, Machine-Independent Code Optimizer, Code Generator, Machine-Dependent Code Optimizer [4, p. 5], however this chapter covers the components relevant to this thesis. See relevant stages in fig. 3.3.

Figure 3.3: Overview of relevant compiler stages.



## Lexical Analysis

The first stage of a compiler is lexical analysis, also known as a *lexer* or *tokenizer*. For the remainder of this thesis, *lexer* will refer to the lexical analysis stage of a compiler. The lexer consumes a stream of characters, the *source code*, and returns a stream of *tokens*. A *token* is a lexically indivisible unit, for example, the Python keyword `return`, you cannot divide it further, for example, into `re` `turn`. Each token is comprised of characters. The rule that defines which combination of characters constitutes a given token is called a *pattern*. The sequence of characters matching a pattern is called a *lexeme*, the *lexeme* is stored along with the token as a value.

## Regular Expressions

A compact way to represent the patterns accepting tokens are *regular expressions*, which were introduced in definition 3.2.6. Regular expressions are an algebraic definition of patterns, they specify *regular languages*,  $\mathcal{L}_3$

## Syntactic Analysis

### LL Parser

## Semantic Analysis

## Code Generation

---

---

## Chapter 4

# Transpiler Implementation

[[mention lexer regex implementation]] [[mention LL table calculation according to theory]] [[write about transformation of ast]]

---

---

## Chapter 5

# Evaluation of Results

[[talk about methodology]] [[test examples analytical vs FDTD]]

---

---

## Chapter 6

# Conclusion

[[compare lumerical and meep]] [[test result TBD]]

---

---

# Bibliography

- [1] PETER, L. *An Introduction to Formal Languages and Automata*. Jones & Bartlett Publishers, january 2016. ISBN 9781284077247. Available at:  
<[https://books.google.com/books/about/An\\_Introduction\\_to\\_Formal\\_Languages\\_and.html?hl=&id=pDaUCwAAQBAJ](https://books.google.com/books/about/An_Introduction_to_Formal_Languages_and.html?hl=&id=pDaUCwAAQBAJ)>.
- [2] SALOMAA, A. *Formal languages*. Boston: Academic Press, 1987. ISBN 0126157502. Available at: <[https://openlibrary.org/books/OL2373365M/Formal\\_languages](https://openlibrary.org/books/OL2373365M/Formal_languages)>.
- [3] CHOMSKY, N. Three models for the description of language. *IRE Transactions on information theory*. IEEE, 1956, vol. 2, no. 3, p. 113–124.
- [4] AHO, A. V.; LAM, M. S.; SETHI, R. and ULLMAN, J. D. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. 2.th ed. USA: Addison-Wesley Longman Publishing Co., Inc., 2006. ISBN 0321486811.