Automata, Nets and Applications
in Software Engineering.

We will start with studying the automata theory to the extent needed to develop your first translator/compiler. While automata theory possesses its own beauty, we will study it with a practical goal in mind. You will develop your own small programming language or select a part of an existing language (Java, Lisp, ML, SETL or suggested by you). Then you will implement your own interpreter/compiler for it.

You may ask why? Why do I need to implement yet another language? One answer is that in the software industry a need in small “in-house” languages appears quite often. It is very often needed to use textual representation of configurations, models, intermediate results in a big software system. CAD systems, servers of any kind (web, network, radio) are full of script, command, and markup languages. Network protocols are languages and contain other small sublanguages. Another answer is that it is just fun to create the interpreter of your own language (or the part of your favourite language), isn’t it?

After the implementation of a compiler we will move to the world of concurrent computations. The modern computational paradigm shifts from traditional centralized computing on workstations, servers, or groups of servers, to distributed, decentralized, loosely coupled computing (cloud, ubiquitous, edge, utility computing). More simply, “computations” are spread between all computational nodes of a distributed system (peer-to-peer networks, wireless sensor networks), rather than concentrated in the dedicated “computational core” of the system. Classical tools for construction and analysis of traditional sequential computational systems are state machines (finite automata), stacks (pushdown automata) and universal computers (Turing machines, $λ$-calculus). But construction and understanding of concurrent systems demand rather different models and approaches. The bunch of different approaches were developed to address these issues –communicating automata, process algebras, logical calculus and many different graph models.

We will study one of them – Petri nets – a classical tool used in the development of distributed systems. The formalism is not the only choice among others, but it has a nice graphical notation that easier the understanding and development of distributed systems. Another benefit of Petri nets is that this formalism can be found in the vast amount of real-world software tools and systems. When we learn Petri nets basics, we will develop a distributed system. Each student will develop his own part with local behaviour. At the end of the course, we will combine our parts to obtain a big distributed system.

# Course Objective

The main practical goal of the course is to teach students the basics of automata and net theories with application in the fields of translators and distributed systems development. As the result students will learn how to systematically design and implement translators, and they will develop their first compiler and distributed system. Automata and net theories have many application in other fields of computer science. We have chosen translators and distributed systems as they were always considered as the black art of programming. The students will see how beautiful theoretical constructions enable them to construct serious industrial software. The course will provide a basis for more advanced courses as:

* Model-driven development (the OMG MDA software development methodology);
* Formal methods in Software Engineering;
* Business processes management systems;
* Compiler construction;
* Network and distributed protocols.

# The position of the course in the structure of the educational program

The course length is 40 academic hours of audience classes divided into 19 lecture hours and 21 seminar hours. Academic control forms are two home assignments, one test, and one written exam.

## Prerequisites of the course:

The common undergraduate knowledge of programming, algorithms and data structures.

* CS101 – Programming Fundamentals
* CS103 – Algorithms and Data structures

# Topic-Wise Curricula Plan

|  |  |  |  |
| --- | --- | --- | --- |
| № | Topic name | Course hours, total | Audience hours |
| Lectures | Practicalstudies |
|  | **Part 1 (20 hrs.)** |  |  |  |
|  | Motivation lecture.Compiler dragons. Modelling languages (SDL/MSC, UML, IDEF). Model checking. Network protocols (TCP/IP, faster packets processing). Business processes. Whole map of the underturing world. People and their influence: J. Ullman, Rabin, Buchi, C.A. Petri, K. Jensen.Tools: JFLAP, CPNtools, NPNtools. Only pictures – no formalisms/theorems. |  | 1 | 0 |
|  | Finite automata (DFA/NFA): informal and formal definitions, basic terminology, operational semantics. Categories of FA. From FA to languages and back ($FA\rightarrow L(FA)$, $L\rightarrow FA$). |  | 2 | 2 |
|  | Grammars (game introduction). $G\rightarrow L\left(G\right)$. $L\rightarrow G$. Noam Chomsky: idea, failure and success. Chomsky hierarchy. Generators vs. recognizers.  |  | 2 | 2 |
|  | Regular world: regular languages, regular grammars (right/left linear), regular expressions, Kleene algebra. Transformations: $FA\leftrightarrow RE, FA\leftrightarrow RG(r/l)$. |  | 2 | 2 |
|  | Push down automata and context free languages. Informal/formal definitions. PDA/NPDA, stack, counters (2-counters machines). Context free grammars. $PDA\rightarrow CFG, CFG\rightarrow PDA$. Normalization. |  | 2 | 2 |
|  | Parsing and translation: Normal Chomsky form. CYK algorithm. LL, LR grammars. Attributed trees. Compiler-compilers: YACC, BISON, ANTLR. |  | 2 | 3 |
|  | **Part 1, totally:** | 22 | 11 | 11 |
|  | **Part 2 (20 hrs.)** |  |  |  |
|  | Petri nets (PN): informal/formal definitions, basic terminology, operational semantics. Petri nets hierarchy. Modeling distributed systems with Petri nets. |  | 2 |  |
|  | Petri nets analysis. Checking structural and behavioral properties (invariants, traps, siphons, SCC, hammocks etc). |  | 2 | 2 |
|  | On the implementation of Petri nets. Distributed resource allocation and implementation of a PN as a distributed system. |  | 4 | 8 |
|  | **Part 2, totally:** | 18 | 8 | 10 |
|  | **TOTAL:** | 40 | 19 | 21 |

1. Hopcroft, John E.; Motwani, Rajeev; Ullman, Jeffrey D. (2006). Introduction to Automata Theory, Languages, and Computation (3rd ed.). Addison-Wesley. ISBN 81-7808-347-7
2. Elaine Rich (2008). Automata, Computability and Complexity: Theory and Applications. Pearson. ISBN 0-13-228806-0.
3. Jacques Sakarovitch, Elements of Automata Theory, Cambridge University Press, 758 pp. 2009, ISBN 0521844258, 9780521844253.
4. Aho, Alfred V., Lam, Monica S., Sethi, Ravi and Ullman, Jeffrey D.. Compilers : Principles, Techniques, & Tools, Second Edition. Second Edited by Michal Hirsch, Matt Goldstein, Katherine Harutunian and Jefferey Holocumb. : Addison-Wesley, 2007.
5. Dirk Taubner: On the Implementation of Petri Nets. European Workshop on Applications and Theory of Petri Nets 1987: 418-434
6. Petri Nets: Properties, Analysis and Applications, by Tadao Murata, in: Proceedings of the IEEE, vol. 77, no. 4, April 1989. (pp. 541-580)
7. Wolfgang Reisig, Understanding Petri Nets. Modeling Techniques, Analysis Methods, Case Studies, 2013, 230 p. ISBN 978-3-642-33278-4.
8. The Petri Nets World <http://www.informatik.uni-hamburg.de/TGI/PetriNets/>