UDC 519.713:004.8
DOI https://doi.org/10.32782/2521-6643-2026-2-72.3

Sopronyuk T. M., Candidate of Physical and Mathematical Sciences,
Associate Professor,
Associate Professor at the Department of Applied Mathematics
and Information Technologies
Yuriy Fedkovych Chernivtsi National University
ORCID: 0000-0002-7031-9880

AUTOMATED WORK WITH FINITE AUTOMATA: A TOOL FOR TEACHING IN THE ERA OF ARTIFICIAL INTELLIGENCE

The article addresses the problem of maintaining objectivity in student assessment in the context of widespread access to artificial intelligence tools, particularly large language models such as ChatGPT. The study focuses on the discipline of formal languages and automata theory, where traditional task formats (transition tables) are easily solved by modern AI models. The purpose of the work is to develop software for automated generation, determinization, and visualization of finite automata, enabling rapid creation of individual assignment variants in visual format (transition diagrams). The system includes a random NFA generator with guaranteed state reachability and controlled transition density, a visualization module with customizable parameters, a pairwise automaton display function for concatenation and union operations, an assignment variant generator (20 variants with concatenation, union, and iteration tasks), and an NFA determinization algorithm that reads automata from text files and visualizes the resulting DFA. Experimental results show that visual format tasks significantly reduce AI solution success, while student testing confirms reduced AI usage. The developed software can be used in teaching courses on automata theory and language processors.

Key words: *finite automata, nondeterministic automaton, determinization, automaton generation, visualization, artificial intelligence, ChatGPT, educational process, Python, Jupyter Notebook, graphviz.*

Сопронюк Т. М. Автоматизована робота зі скінченними автоматами: інструментарій для навчання в епоху штучного інтелекту

Стаття присвячена проблемі збереження об'єктивності оцінювання знань студентів в умовах широкого доступу до інструментів штучного інтелекту, зокрема великих мовних моделей типу ChatGPT. Дослідження зосереджене на дисципліні «Теорія формальних мов та автоматів», де традиційні форми завдань (таблиці переходів) легко розв'язуються сучасними ШІ-моделями. Метою роботи є розробка програмного забезпечення для автоматизованої генерації, детермінізації та візуалізації скінченних автоматів, яке дозволяє швидко створювати індивідуальні варіанти завдань у візуальному форматі (діаграми переходів). Система включає генератор випадкових НКА з гарантованою досяжністю станів та контрольованою щільністю переходів, модуль візуалізації з налаштованими параметрами, функцію парного відображення автоматів для операцій конкатенації та альтернативи, генератор варіантів завдань (20 варіантів із завданнями на конкатенацію, альтернативу та ітерацію) та алгоритм детермінізації НКА, що зчитує автомати з текстових файлів і візуалізує отриманий ДКА. Експериментальні результати показують, що завдання у візуальному форматі значно знижують успішність розв'язування ШІ, а тестування на студентах підтверджує зниження використання. Розроблене програмне забезпечення може бути використане у викладанні курсів з теорії автоматів та мовних процесорів.

Ключові слова: *скінченні автомати, недетермінований автомат, детермінізація, генерація автоматів, візуалізація, штучний інтелект, ChatGPT, навчальний процес, Python, Jupyter Notebook, graphviz.*

Formulation of the problem. In modern educational processes, artificial intelligence (AI) tools, particularly large language models (LLMs) such as ChatGPT, Claude, Gemini, and others, are becoming increasingly widespread [7; 8]. These tools demonstrate remarkable capabilities in solving typical academic tasks, creating a serious challenge for student knowledge assessment systems. This problem is particularly acute in technical disciplines where tasks are often formalized and can be easily automated [9].

The theory of formal languages and automata is a fundamental discipline in computer science education [1; 4]. Traditional tasks in this course include constructing nondeterministic finite automata (NFA) for given regular expressions, performing concatenation, union, and iteration operations on automata, as well as NFA determinization [5]. Research shows that modern AI models can effectively solve such tasks when presented in formalized form, such as transition tables.

One promising approach to maintaining assessment objectivity is shifting from tabular to visual format (transition diagrams), which requires students not only to know algorithms but also to possess skills in interpreting

graphical structures. However, manually creating large numbers of individual assignments with visual automaton representations is extremely time-consuming for instructors. Thus, the development of tools for automated generation of such assignments, enabling rapid creation of numerous unique variants, automaton visualization, determinization demonstration, and pairwise comparison of automata for various operations, is highly relevant.

Analysis of recent research and publications. The impact of artificial intelligence on education has become the subject of intensive research in 2023–2026. Lo [7] conducted a large-scale study of ChatGPT's capabilities in solving academic tasks across various disciplines. The author demonstrates that the model successfully performs tasks in programming, mathematics, and engineering disciplines, achieving levels comparable to top-performing students. Bommasani et al. [8] explore the ethical aspects of AI use in education, emphasizing the need to reconsider assessment methods. The authors propose focusing on tasks requiring critical thinking, creative approaches, and deep understanding of the material, rather than merely formal application of algorithms.

In the context of teaching automata theory, the research by Rodger and Finley [10] is significant; they developed the JFLAP educational environment that allows students to interactively work with automata. JFLAP supports automaton visualization, concatenation, union, and iteration operations, as well as determinization. However, JFLAP does not offer functionality for automated generation of individual tasks – each automaton must be entered manually, significantly limiting the ability to create numerous unique variants.

The PySimpleAutomata library [11] provides basic tools for working with automata in Python, including determinization algorithm implementation. However, it lacks functionality for generating random automata, and its visualization is limited to DOT format, requiring additional tools for conversion to images. The automata-lib library [12] implements basic algorithms, but its visualization is even more limited.

The work [3] examines the phases of language processor construction based on automata theory, presenting practical implementation aspects. The textbook [4] thoroughly covers the theoretical foundations of language processors and formal languages, providing practical examples of finite automaton implementation. The theses [5] describe an educational simulator for operations with nondeterministic finite automata, demonstrating visualization capabilities and interactive work with automata.

Special mention should be made of the textbook [6], which applied the approach of automated generation of random individual assignment variants for students – in the context of solving differential equations with impulse action. This approach to creating individual assignments by generating random problem parameters proved effective and has been adapted in this work for the needs of automata theory.

The system developed in this article extends the ideas from our previous works [3; 4; 5; 6] by adding automated generation and pairwise visualization.

The problem of automated generation of random automata was studied by Tabakov and Vardi [13], Almeida et al. [14], who proposed methods for ensuring state reachability and controlling transition density. A recent work by Baburin and Cotterell [2] provides a detailed analysis of the subset construction algorithm used for determinization, exploring when it leads to exponential state growth. However, these studies focus on theoretical aspects of generation and algorithm analysis rather than creating practical educational tools.

Thus, there is a need for the development of tools that combine random NFA generation, their visualization, automated task variant formation, and determinization, while also providing pairwise automaton comparison for concatenation and union operations.

The purpose of the article is to develop software for automated generation, determinization, and visualization of finite automata that enables:

1) generating random nondeterministic finite automata with controlled parameters (number of states, alphabet size, transition density);

2) visualizing automata as transition diagrams with customizable parameters suitable for inclusion in educational materials;

3) providing pairwise display of two automata side by side for concatenation and union operations;

4) automatically forming individual assignment variants for students (20 or more variants) with concatenation, union, and iteration operations;

5) performing NFA determinization with visualization of the resulting DFA;

6) investigating the effectiveness of visual automaton representation as a means of countering automatic task solving by artificial intelligence.

Presenting main material. To achieve the stated purpose, a software module was developed in Python using the graphviz library [15] for visualization and IPython.display for displaying results in the Jupyter Notebook environment. The program consists of four main components: random NFA generator, visualization module, assignment variant generator, and determinization module.

2.1. Formal definition of finite automata

A finite automaton (FA) is a mathematical model of a discrete system that has a finite number of states and transitions between them under the influence of input symbols. Formally, an NFA is defined as a quintuple [1; 4]:

$$A = (Q, \Sigma, \delta, q_0, F),$$

where:

- Q – finite set of states;
- Σ – finite alphabet (set of input symbols);
- $\delta: Q \times \Sigma \to 2^Q$ – transition function mapping (state, symbol) to a set of possible next states;
- $q_0 \in Q$ – initial state;
- $F \subseteq Q$ – set of final (accepting) states.

Unlike a deterministic finite automaton (DFA), in an NFA the transition function can return multiple states (including the empty set), reflecting behavioral ambiguity. A DFA is a special case of an NFA where, for each pair (state, symbol), the transition function returns at most one state, i.e., $\delta: Q \times \Sigma \to Q$.

2.2. Random NFA generator

The developed method for generating random NFAs is based on the principle of ensuring reachability of all states from the initial state. This is an important property for educational tasks, since unreachable states do not affect the automaton's language but may confuse students.

Algorithm 1. Random NFA Generation

Input: number of states $n \in \{3, 4, 5\}$, alphabet size $|\Sigma| \in \{2, 3\}$, transition density $p = 0.2$

Output: NFA $A = (Q, \Sigma, \delta, q_0, F)$

1. $Q \leftarrow \{q_1, q_2, \ldots, q_n\}$
2. $q_0 \leftarrow q_1$
3. $F \leftarrow$ random selection of m states from Q, where $m \in [2, [n/2]]$
4. $\Sigma \leftarrow$ random selection of 2–3 symbols from $\{a, b, c, d, f, x, y, z\}$
5. Initialize $\delta(q, a) \leftarrow \varnothing$ for all $q \in Q$, $a \in \Sigma$
6. for $i = 2$ to n:
7. $q_j \leftarrow$ random state from $\{q_1, \ldots, q_{i-1}\}$
8. $a \leftarrow$ random symbol from Σ
9. $\delta(q_j, a) \leftarrow \delta(q_j, a) \cup \{q_i\}$
10. for each $q \in Q$ and $a \in \Sigma$:
11. for each $t \in Q$, $t \neq q_0$:
12. if random() < p:
13. $\delta(q, a) \leftarrow \delta(q, a) \cup \{t\}$
14. return $(Q, \Sigma, \delta, q_0, F)$

The key feature of the algorithm is guaranteeing reachability of all states (steps 6–9): for each state q_i ($i \geq 2$), a transition is added from some previous state q_j ($j < i$), ensuring a path from q_0 to each state. This eliminates the need for an additional cleaning stage to remove unreachable states, which is often required in general generation methods.

The choice of parameters $n \in \{3, 4, 5\}$ and $|\Sigma| \in \{2, 3\}$ is determined by educational goals: automata with 3–5 states are sufficiently complex to demonstrate core concepts but not too cumbersome for students to construct manually. Transition density $p = 0.2$ provides a moderate amount of nondeterminism – interesting enough for analysis but not overly complex.

Figure 1 shows an example of a generated NFA with parameters $n = 4$, $\Sigma = \{a, b\}$.

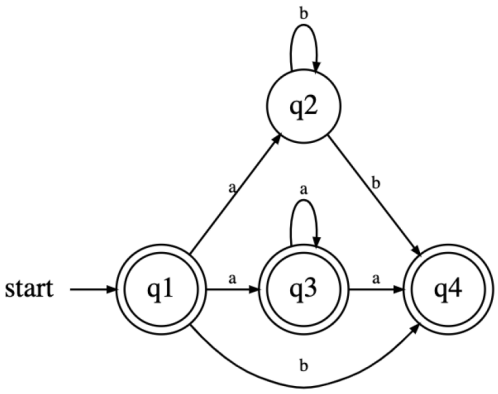

Fig. 1. Example of a generated nondeterministic finite automaton (NFA) with four states and alphabet $\{a, b\}$

Source: developed by the author

2.3. Visualization module

For automaton visualization, the graphviz library [15] is used, which allows generating graphs in DOT format and saving them as images. The developed draw_nfa_for_print() function implements compact automaton display with customizable parameters:

- Image size: 6 × 3 inches – optimal for insertion into text and placing two automata side by side on A4 paper;
- Node distances: nodesep = 0.3, ranksep = 0.3 – ensuring compact layout without element overlap;
- Font sizes: 9 pt for edge labels, 12 pt for states – ensuring readability at reduced image size;
- Node shapes: initial state denoted by an invisible vertex with an arrow; final states – double circles; ordinary states – single circles.

2.3.1. Pairwise automaton display

An important feature of the developed system is the ability to display two automata side by side, which is necessary for visualizing concatenation and union operations. For this purpose, the display_two_nfas() function is implemented, performing the following actions:

1. Alphabet alignment: uses the first automaton's alphabet for the second, ensuring correctness of subsequent operations. If automata have different alphabets, concatenation and union operations require the same alphabet, making this alignment critically important.

2. Image generation: calls draw_nfa_for_print() for each automaton separately with appropriate labels ("A", "B", "C", "D", "M^*", "M^+", etc.).

3. Row placement: uses CSS flexbox rules to place two images in a single row with equal spacing (gap: 30 px). Each image occupies 50 % of container width, ensuring symmetric display.

The implementation uses SVG format for images, guaranteeing high quality when scaling and maintaining sharpness when printing. The CSS rule page-break-inside: avoid prevents splitting the automaton pair between pages when printing, which is important for maintaining task integrity.

Figure 2 shows an example of pairwise display of two automata with parameters n □ {3, 4}, Σ = {f, y}.

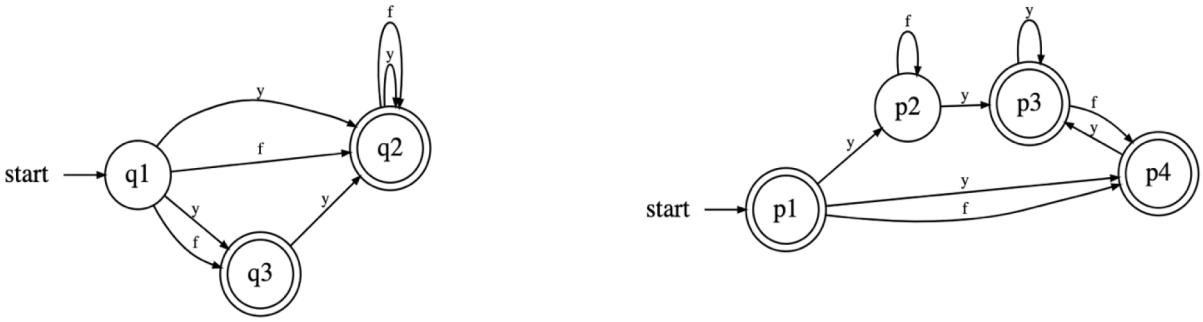

Fig. 2. Example of pairwise display of two automata

Source: developed by the author

2.4. Assignment variant generator for students

The main functionality of the developed system, focused on practical application in the educational process, is the automated generation of individual assignment variants. The generate_variant() function creates one variant containing three types of tasks:

1. **Concatenation (A · B):** students are asked to construct an NFA for concatenation of languages given by automata A and B, presented as transition diagrams. Concatenation is one of the fundamental operations on regular languages, and the ability to construct an automaton for concatenation from automata for the factors is an important skill.

2. **Union (C ∪ D):** students are asked to construct an NFA for the union of languages given by automata C and D. Union is another fundamental operation that, together with concatenation and iteration, forms the basis for constructing regular expressions.

3. **Iteration (M^* and M^+):** students are asked to separately construct NFAs for Kleene star (M^*) and positive iteration (M^+) for a given automaton. These operations allow describing language repetitions and are key to understanding regular expressions.

For each task type, new random automata with the same alphabet are generated, ensuring correctness of operations. The alphabet for each task is generated independently, preventing cheating between different task types within a single variant – the student receiving a variant has a unique set of automata for each task.

The program generates 20 variants, each placed on a separate page using CSS rules page-break-before: always and page-break-after: always. This allows directly printing the resulting HTML document as a collection of individual assignments. If necessary, the number of variants can be easily increased by changing the loop parameter.

Figure 3 shows a fragment of the generated assignment collection.

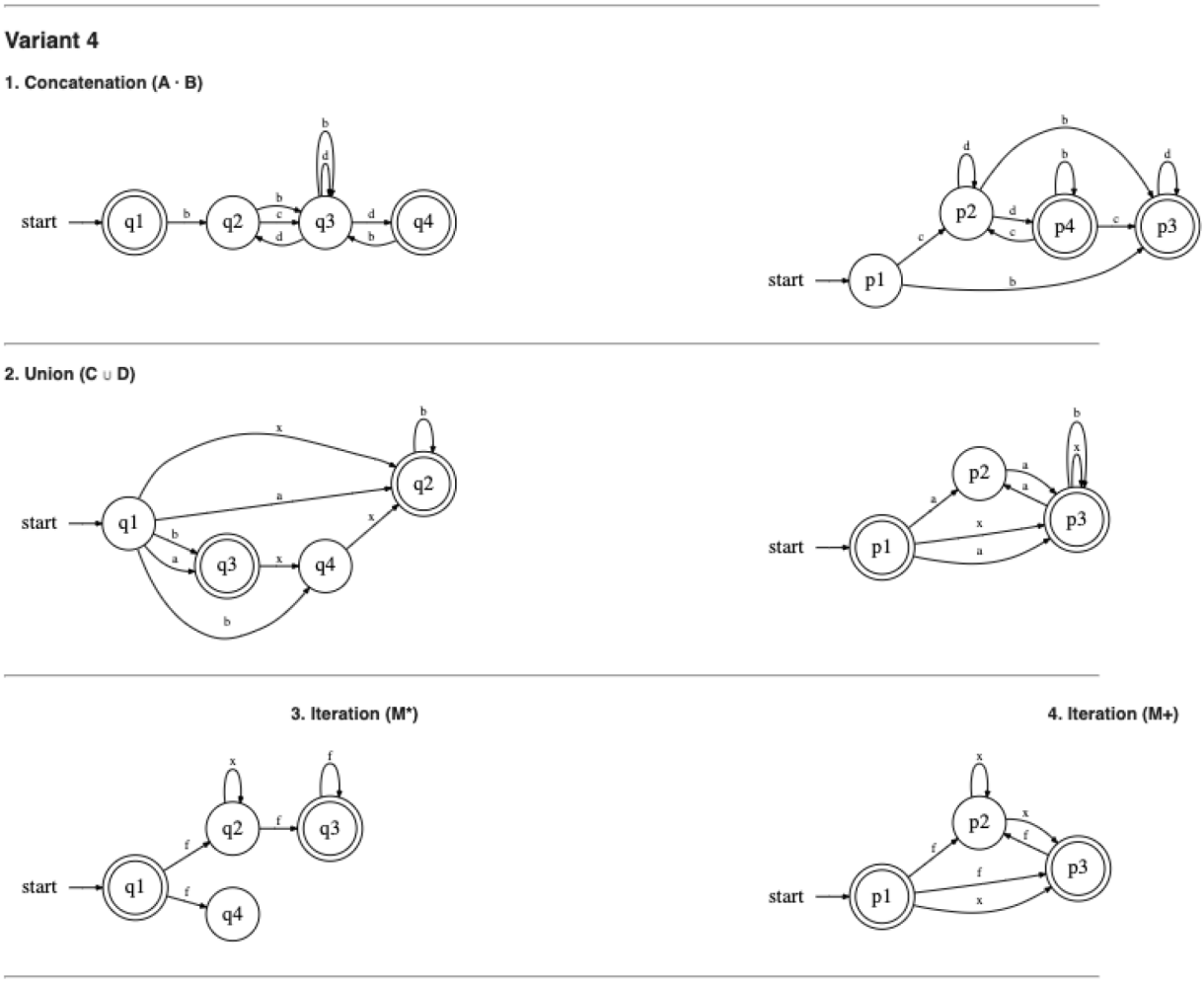

Fig. 3. Fragment of the generated assignment collection (variant 4)

Source: developed by the author

2.5. Determinization algorithm

An important component of the automata theory course is NFA determinization – converting a nondeterministic automaton into an equivalent deterministic one. This process demonstrates a fundamental property: the classes of languages recognized by NFAs and DFAs coincide [1].

The developed system implements a determinization algorithm based on the subset construction method. The software implementation (file NFA-to-DFA.ipynb) performs the following steps:

1. **Reading the input NFA** from a text file in transition table format. The first line of the file contains the alphabet (symbols separated by "|"). Subsequent lines describe transitions for each state. The symbol "–" before a state name denotes the initial state, the symbol "*" denotes a final state. If a state is both initial and final, a combined notation is used.

2. **Building the DFA state set** as subsets of the original NFA states. The initial DFA state is the set containing only the initial NFA state.

3. **Computing transitions** for each newly formed set of states for each alphabet symbol. For a given set of states S and symbol a, the union of sets of states reachable from each state in S by symbol a is computed. If the resulting set of states T is not yet in the DFA state set, it is added as a new state.

4. **Determining DFA final states** – those sets that contain at least one final state of the original NFA.

5. **Visualizing the resulting DFA** using the same visualization module as for the original NFA.

An important feature of the implementation is **identifying reachable states** after building the DFA. The algorithm analyzes which DFA states are reachable from the initial state and excludes unreachable states from the final representation. This allows obtaining a DFA with a minimal number of states equivalent to the original NFA, which is important for demonstration clarity.

Figure 4 shows the NFA and the equivalent DFA obtained after applying the determinization algorithm.

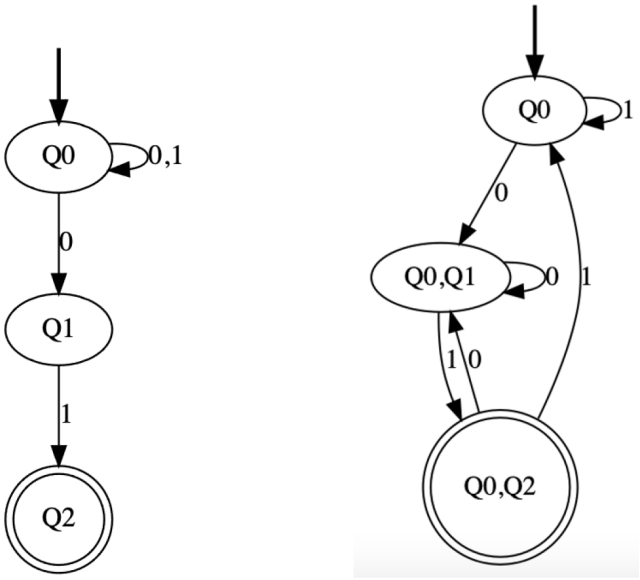

Fig. 4. NFA (left) and equivalent DFA (right) after determinization

Source: developed by the author

2.6. Experimental study

To evaluate the effectiveness of the proposed approach, an experimental study was conducted in two directions: assessment of modern AI models' ability to solve tasks in different formats, and analysis of the impact of representation format on student academic performance.

2.6.1. AI testing on different representation formats

To assess the capabilities of modern AI models in solving automata theory problems, a comparative analysis was conducted using ChatGPT-4 (version from March 2026). A typical task «Construct an NFA for concatenation of languages given by automata A and B» was presented in two ways:

- **Tabular format** – automata presented as transition tables, where rows correspond to states, columns to alphabet symbols, and intersections indicate sets of next states.
- **Visual format** – automata presented as transition diagrams (Fig. 1), where states are depicted as circles and transitions as arrows with symbol labels.

Testing was conducted on a sample of 50 different automaton pairs for each task type (concatenation, union, iteration). Each task was presented to the AI in the corresponding format, and responses were evaluated based on the correctness of constructing the resulting NFA.

Experimental results showed that in tabular format, the AI demonstrated high efficiency: correct responses were obtained for the vast majority of tasks across all operation types. This confirms that modern LLMs handle formalized textual descriptions of automata well.

In visual format, AI success was significantly lower. Even when using image recognition tools integrated into multimodal model versions, correctly recognizing the automaton structure from transition diagrams proved challenging. Success was substantially lower even for the simplest tasks. This is because visual representation requires not only recognition of graphical elements (circles, arrows, labels) but also interpretation of their semantics as states, transitions, and symbols.

The obtained results confirm the hypothesis that visual automaton representation is an effective means of countering automatic task solving by artificial intelligence. A student receiving a task as transition diagrams cannot simply copy it into ChatGPT – they must first independently recognize the automaton structure, requiring deep understanding of the material.

2.6.2. Student testing

To assess the impact of representation format on academic performance and frequency of AI use, an experiment was conducted involving students studying the course "Theory of Formal Languages and Automata". The experiment involved 24 students, divided into two groups of 12 with comparable previous academic performance:

- **Control group** received tasks in tabular format (traditional presentation).
- **Experimental group** received similar tasks in visual format (transition diagrams), generated by the developed system.

Students were asked to complete three task types (concatenation, union, iteration). After completing the tasks, an anonymous survey was conducted regarding AI tool use and satisfaction with the task format.

Experimental results showed that the average score in the control group was somewhat higher compared to the experimental group. The difference can be explained by the more effective use of AI tools by students in the control group.

Task completion time in the experimental group was somewhat longer, which is also expected since students needed to spend time recognizing automaton structure from diagrams.

Most indicative was the difference in AI use frequency. According to self-report data (confirmed by cross-analysis of response styles), a significant portion of students in the control group admitted to using ChatGPT or similar tools for assistance. In the experimental group, this figure was significantly lower. Students explained this by noting that entering transition diagrams into AI is time-consuming, and the obtained responses often contain errors due to incorrect automaton structure recognition.

The survey also showed that students in the experimental group rated tasks as more interesting and meaningful compared to the control group. This indicates that the visual format not only complicates AI use but also increases student engagement in the learning process.

Thus, the experimental study confirms that using visual automaton representation format is an effective means of reducing AI use frequency among students while maintaining academic performance at an acceptable level.

2.7. Comparison with existing solutions

The developed system was compared with existing tools based on the following criteria: availability of random automaton generation, visualization quality, pairwise display support, capability for automated assignment variant generation, determinization implementation, protection against automatic AI solving, and integration with Jupyter Notebook.

JFLAP [10] is a powerful educational environment supporting automaton visualization and interactive operation execution. However, it lacks functionality for automated random automaton generation – each automaton must be entered manually, making creation of large numbers of individual assignment variants extremely labor-intensive. Additionally, JFLAP is a standalone application and does not integrate with Jupyter Notebook environments.

PySimpleAutomata [11] provides a library for working with automata in Python, including determinization implementation. However, its visualization is limited to DOT format, requiring additional tools for conversion to images. The library does not include random automaton generation functionality, limiting its use for creating large sets of test examples. Furthermore, representing automata in text format makes them easily accessible for automatic AI solving.

The developed system stands out with a unique combination of functionality:

- random NFA generation with guaranteed state reachability and controlled transition density;
- high-quality visualization in SVG format with customizable parameters for printing;
- pairwise automaton display for concatenation and union operations;
- automated generation of 20 or more individual assignment variants;
- determinization implementation with reachable state identification and result visualization;
- high level of protection against automatic AI solving due to visual format use;
- full integration with Jupyter Notebook, allowing system use directly in the educational environment.

Conclusions. The following scientific and practical results were obtained during this work:

1. A method for generating random NFAs with guaranteed reachability of all states and controlled transition density was developed. The method's feature is ensuring reachability by adding transitions from previous states, eliminating the need for an additional automaton cleaning stage. This method enables creating diverse educational tasks and experimental samples.

2. A visualization module for automata as transition diagrams with customizable parameters (size 6×3 inches, node distances 0.3, font size 9 pt for edges) was implemented, providing compact display suitable for inclusion in educational materials and printing on A4 format.

3. A pairwise display function display_two_nfas() was created, providing placement of two automata side by side with equal spacing and prevention of page breaks. This is critically important for visualizing concatenation and union operations.

4. An assignment variant generator generate_variant() was developed, which automatically forms individual variants for students. Each variant contains three task types: concatenation, union, and iteration (M^* and M^+). The program generates 20 variants with automatic placement on separate pages.

5. An NFA determinization algorithm based on the subset construction method was implemented, with an additional stage of identifying reachable states. The software implementation (file NFA-to-DFA.ipynb) allows reading an NFA from a text file in transition table format, building an equivalent DFA, and visualizing the result with automatic state renaming for ease of perception.

6. An experimental study confirmed that:

– modern AI models (ChatGPT-4) demonstrate high success rates when solving tasks in tabular format;

– in visual format, AI success is significantly lower, making it an effective countermeasure;

– using visual format reduces AI use frequency among students with minimal impact on academic performance;

– students rate visual format tasks as more interesting and meaningful.

The scientific novelty of the obtained results lies in:

• development of a method for generating random NFAs with guaranteed state reachability adapted for educational purposes;

• creation of a comprehensive toolkit for automated generation of individual tasks, pairwise visualization, and NFA determinization;

• experimental confirmation of visual representation effectiveness as a means of countering AI use in the educational process.

The practical significance of the work lies in creating software that enables instructors to effectively prepare individual assignments for students in conditions of widespread access to AI tools, ensuring objectivity of knowledge assessment. The developed modules can be used in the educational process when teaching the courses "Theory of Formal Languages and Automata" and "Language Processors" [3; 4; 5]. The software implementation of the described algorithms is available in an open repository [16].

Prospects for further research in this direction include:

1. Expanding functionality to support other task types (DFA minimization, regular expression conversion, automaton intersection and complement).

2. Integration with distance learning systems (Moodle, Google Classroom) for automatic variant distribution, answer collection, and automated verification.

3. Development of methods for automatic verification of completed tasks presented in visual format using computer vision techniques.

4. Investigation of AI adaptation possibilities for recognizing transition diagrams and development of corresponding countermeasures to maintain the effectiveness of the proposed approach.

5. Adding export capability to various formats (PDF, LaTeX, HTML) for integration with various document management systems and publishing platforms.

Bibliography:

1. Hopcroft J. E., Motwani R., Ullman J. D. Introduction to Automata Theory, Languages, and Computation. 3rd ed. Boston : Pearson Education, 2006. 535 p. ISBN 978-0-321-45536-9

2. Baburin I., Cotterell R. A Close Analysis of the Subset Construction // arXiv preprint. 2025. arXiv:2407.09891v5. DOI: https://doi.org/10.48550/arXiv.2407.09891

3. Sopronyuk T. M., Sopronyuk A. Yu., Drobot A. V. Phases of language processor construction for.NET platform // Буковинський математичний журнал. 2023. Т. 11, № 2. С. 71–84. DOI: https://doi.org/10.31861/bmj2023.02.07

4. Сопронюк Т.М. Мовні процесори та формальні мови: від теорії до практики : навч. посібник. Чернівці : Чернівец. нац. ун-т ім. Ю. Федьковича, 2025. 198 с. ISBN 978-966-423-942-1. URL: https://archer.chnu.edu.ua/handle/123456789/12079

5. Паранчич М. Ю., Сопронюк Т.М. Навчальний тренажер для операцій з недетермінованими скінченними автоматами // Матеріали Міжнародної наукової інтернет-конференції «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення», випуск 96 (м. Тернопіль, Україна, м. Ополе, Польща, 11–12 лютого 2025 р.). 2025. С. 34–36. URL: http://www.konferenciaonline.org.ua/ua/article/id-2090/

6. Петришин Р. І., Сопронюк Т. М. Наближені методи розв'язування диференціальних рівнянь з імпульсною дією : навч. посібник. Чернівці : Чернівецький національний університет, 2010. 200 с. ISBN 978-966-423-113-5

7. Lo C. K. What Is the Impact of ChatGPT on Education? A Rapid Review of the Literature // Education Sciences. 2023. Vol. 13, No. 4. P. 410. DOI: https://doi.org/10.3390/educsci13040410

8. Bommasani R., Hudson D. A., Adeli E., et al. On the Opportunities and Risks of Foundation Models // arXiv preprint. 2021. arXiv:2108.07258. DOI: https://doi.org/10.48550/arXiv.2108.07258

9. Eaton S. E. Postplagiarism: transdisciplinary ethics and integrity in the age of artificial intelligence and neurotechnology // International Journal for Educational Integrity. 2023. Vol. 19, No. 1. DOI: https://doi.org/10.1007/s40979-023-00144-1

10. Rodger S. H., Finley T. W. JFLAP: An Interactive Formal Languages and Automata Package. Sudbury : Jones & Bartlett Learning, 2006. 192 p. ISBN 978-0-7637-3834-1

11. PySimpleAutomata documentation. URL: https://pysimpleautomata.readthedocs.io/ (дата звернення: 22.03.2026).

12. automata-lib documentation. URL: https://github.com/caleb531/automata (дата звернення: 22.03.2026).

13. Tabakov D., Vardi M. Y. Experimental evaluation of classical automata constructions // International Conference on Logic for Programming Artificial Intelligence and Reasoning. 2005. P. 396–411. DOI: https://doi.org/10.1007/11591191_28

14. Almeida M., Moreira N., Reis R. Enumeration and generation with a string automata representation // Theoretical Computer Science. 2007. Vol. 387, No. 2. P. 93–102. DOI: https://doi.org/10.1016/j.tcs.2007.07.029

15. Graphviz – Graph Visualization Software. URL: https://graphviz.org/ (дата звернення: 22.03.2026).

16. Сопронюк Т. М. NFA Automata Tools: програмне забезпечення для генерації, візуалізації та детермінізації скінченних автоматів. GitHub. URL: https://github.com/tsopronyuk/nfa-automata-tools (дата звернення: 23.03.2026).

References:

1. Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation* (3rd ed.). Boston: Pearson Education.

2. Baburin, I., & Cotterell, R. (2025). A Close Analysis of the Subset Construction. arXiv preprint. https://doi.org/10.48550/arXiv.2407.09891

3. Sopronyuk, T. M., Sopronyuk, A. Yu., & Drobot, A. V. (2023). Phases of language processor construction for.NET platform. *Bukovinian Mathematical Journal, 11* (2), 71–84. https://doi.org/10.31861/bmj2023.02.07

4. Sopronyuk, T. M. (2025). *Language Processors and Formal Languages: From Theory to Practice*. Chernivtsi: Yuriy Fedkovych Chernivtsi National University. Retrieved from https://archer.chnu.edu.ua/handle/123456789/12079 (in Ukrainian).

5. Paranchych, M. Yu., & Sopronyuk, T. M. (2025). Educational simulator for operations with nondeterministic finite automata. In *Proceedings of the International Scientific Internet Conference "Information Society: Technological, Economic and Technical Aspects of Formation"*, Issue 96 (pp. 34–36). Ternopil–Opole. Retrieved from http://www.konferenciaonline.org.ua/ua/article/id-2090/ (in Ukrainian).

6. Petryshyn, R. I., & Sopronyuk, T. M. (2010). *Approximate methods for solving differential equations with impulse action*. Chernivtsi: Yuriy Fedkovych Chernivtsi National University. (in Ukrainian).

7. Lo, C. K. (2023). What Is the Impact of ChatGPT on Education? A Rapid Review of the Literature. *Education Sciences, 13*(4), 410. https://doi.org/10.3390/educsci13040410

8. Bommasani, R., Hudson, D. A., Adeli, E., et al. (2021). On the Opportunities and Risks of Foundation Models. *arXiv preprint*. https://doi.org/10.48550/arXiv.2108.07258 (preprint)

9. Eaton, S. E. (2023). Postplagiarism: transdisciplinary ethics and integrity in the age of artificial intelligence and neurotechnology. *International Journal for Educational Integrity, 19*(1). https://doi.org/10.1007/s40979-023-00144-1

10. Rodger, S. H., & Finley, T. W. (2006). *JFLAP: An Interactive Formal Languages and Automata Package*. Sudbury: Jones & Bartlett Learning.

11. *PySimpleAutomata documentation*. (n.d.). Retrieved March 22, 2026, from https://pysimpleautomata.readthedocs.io/

12. *automata-lib documentation*. (n.d.). Retrieved March 22, 2026, from https://github.com/caleb531/automata

13. Tabakov, D., & Vardi, M. Y. (2005). Experimental evaluation of classical automata constructions. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning* (pp. 396–411). https://doi.org/10.1007/11591191_28

14. Almeida, M., Moreira, N., & Reis, R. (2007). Enumeration and generation with a string automata representation. *Theoretical Computer Science, 387*(2), 93–102. https://doi.org/10.1016/j.tcs.2007.07.029

15. *Graphviz – Graph Visualization Software*. (n.d.). Retrieved March 22, 2026, from https://graphviz.org/

16. Sopronyuk, T. M. (2026). NFA Automata Tools: software for generating, visualizing, and determinizing finite automata. GitHub. Retrieved March 23, 2026, from https://github.com/tsopronyuk/nfa-automata-tools

Дата першого надходження статті до видання: 24.03.2026
Дата прийняття статті до друку після рецензування: 20.04.2026
Дата публікації (оприлюднення) статті: 30.05.2026