Annual International Scientific Conference
"Competitiveness and Innovation in the Knowledge Economy"
September 26-27, 2025
Chisinau, Republic of Moldova

DOI: https://doi.org/10.24818/cike2025.64

LOGIC AND CONSTRAINTS IN PROGRAMMING: A PATH TO PROFESSIONALISM AND SAFE AI

PELIN NICOLAE
Department of Information Technology and Information Management
Academy of Economic Studies of Moldova
e-mail: nicolae.ion.pelin@gmail.com
ORCID ID: 0009-0007-2792-355X

Abstract: Modern programming languages often provide developers with wide freedom of action under minimal logical constraints. While this accelerates development, it simultaneously increases the likelihood of errors, reduces reliability, and complicates system maintenance. This paper advances the thesis that programmer professionalism is shaped not by unrestricted freedom, but by structure and logical constraints that guide thinking and architectural decisions.

Particular attention is given to logic-declarative approaches and the use of logic-based programming languages (such as Prolog and constraint-oriented languages) for knowledge formalization. It is demonstrated that limiting freedom through formal rules does not suppress creativity but, on the contrary, serves as a condition for responsible reasoning and sustainable design.

The research methodology includes a literature review, analysis of programming paradigms, and conceptual modeling of logic-based systems. The results show that logic-declarative methods enable knowledge formalization in a form that ensures transparency and safety of artificial intelligence (AI) systems. This approach contributes not only to technical reliability but also to a higher level of trust in intelligent systems applied in socially sensitive domains.

The study highlights the importance of logic-declarative methods for education, where they foster a culture of precision and responsibility, as well as for law and AI ethics, where formalized, verifiable, and transparent representations of knowledge are required. Thus, logical constraints in programming should be seen not as a limitation but as a foundation of professional training and a key condition for the safe application of artificial intelligence technologies.

Keywords: formalization, AI safety, Prolog, education, programmer professionalism
JEL Classification: C88, D83, I23

1. Introduction

The relevance of this study stems from the growing problem of explainability and uncontrollability of artificial intelligence (AI) systems. As AI increasingly participates in critical decision-making across economics, medicine, and governance, its autonomy is advancing faster than the development of effective mechanisms for control and safety assurance. The opacity of deep-learning algorithms — the so-called *"black box"* — raises serious concerns about ethical and legal accountability for AI-driven decisions. Therefore, ensuring AI safety requires not only technological or legal constraints but also a conceptual rethinking of its foundations.

Problem statement. The current AI learning paradigm, based on the statistical analysis of big data, produces systems capable of prediction but not of causal reasoning. This reveals the absence of a logical structure as the foundation for transparency and predictability in AI behavior.

Key idea. The central argument of this study is that control and safety in AI are possible only when the system is built on logical principles — a formal discipline that ensures the transparency and explainability of inference processes. Logic serves as a normative framework for reasoning, analogous to law within systems of social regulation, defining the boundaries of responsibility and predictability.

Aim, Hypothesis, and Objectives.

The aim of the study is to substantiate the critical role of the programmer's logical culture as a normative and professional foundation for designing safe and explainable AI systems.

Research hypothesis: the safety and explainability of artificial intelligence are directly proportional to the level of logical culture of its creator.

To achieve this goal, the following objectives are set:

1. To define the place and role of logic as the foundation of professional culture in computer science.
2. To analyze the relationship between knowledge formalization (through logical models) and code verifiability.
3. To demonstrate that the logical education of future programmers is an integral part of the general strategy for ensuring the safety and resilience of AI systems.

The article explores the interrelation between logic, professional reasoning, and the constraints that ensure safety in AI development. Thus, the task of cultivating logical thinking among future programmers becomes a key component of the overall strategy for artificial intelligence safety.

The article examines the interrelations between logic, professional thinking, and the constraints that ensure safety in the context of AI development. Hence, the task of logical training for future programmers becomes part of the general strategy for ensuring artificial intelligence safety.

2. Logic as the Foundation of Thinking and Professionalism

When addressing the question of what defines a programmer's professionalism, it should be emphasized that it begins not with the mastery of programming languages or technical skills, but with the ability to think logically. Logic is the invisible framework that transforms a collection of knowledge into a coherent system, and individual actions into a conscious process. It may thus be regarded as the foundation of professional culture — the basis upon which competence, responsibility, and the capacity for rational problem-solving are built.

In everyday perception, logic is often seen merely as a tool for reasoning or proof. However, in the professional activity of a programmer, it serves a far deeper function — it structures thought itself. Logic establishes the rules of one's internal dialogue, both with oneself and with the machine, transforming a chaotic stream of ideas into an ordered system of concepts, relations, and dependencies. Without it, coding becomes a mechanical act, and design degenerates into an intuitive sequence of random choices.

Logic does not constrain thought; it directs it. It functions like grammar in language — creating a structure that makes thought articulate and comprehensible. Just as grammar enables the free use of language, logic enables the free use of reasoning. Without it, one cannot build a clear argument, explain a solution, or formalize knowledge (Kowalski &Sergot, 1990).

Just as law ensures freedom in a legal society through rules, logic ensures freedom of thought through structure. It does not suppress creativity but renders it meaningful. In this sense, logic is not a restriction but an instrument of liberation: it protects the professional from the arbitrariness of intuition, from perceptual errors, and from cognitive biases.

The professionalism of a programmer is expressed not in the volume of memorized syntactic constructions, but in the ability to see the system of interrelations between facts and formulas. The deeper a programmer understands logic, the higher the level of abstraction they can achieve, the more accurately they can model reality, and the more reliable and secure the systems they create become.

It is logic that transforms a set of algorithms into an intellectual system, and a programmer into a thinker — one capable not merely of writing code, but of creating meaning. Hence, the formation of logical thinking should be a primary goal in the education of programmers. Without it, one cannot speak either of professional culture or of the safety of artificial intelligence.

Understanding logic as the foundation of professionalism naturally leads to the next question — the relationship between language and logic. After all, it is language, whether natural or artificial, that serves as the principal instrument for expressing thought. If logic defines the structure of thinking, language renders that structure visible, turning inner reasoning into a form that can be shared, analyzed, and processed.

For a programmer, language is not merely a means of communicating with a machine, but a form of dialogue with logic itself. Every construct, every syntactic rule, reflects a particular model of reasoning. To understand a language is to understand how logic is embodied within it. Thus, the path from language to logic is the path from the external form of expression to the internal structure of meaning.

3. From Language to Logic

Language is a bridge between thought and action, between inner understanding and outward expression. It unites both cognitive and social functions: on the one hand, language allows a person to structure their own thinking; on the other, it enables the transmission of meaning to others. It is through language that a collective worldview is formed — the foundation of knowledge, culture, and science. A schematic representation of the basic elements of a text (facts, rules, and queries) and the process of extracting an answer from it can be useful as well (Pelin, 2023, p. 567).

For a programmer, language has yet another dimension: it is an instrument for the formalization of thought. Whereas natural language allows for ambiguity and contextual meanings, programming languages demand precision and unambiguity. In this difference lies the key role of logic — it transforms language from a means of communication into a means of controlling processes and knowledge.

In informatics, the transition from language to logic can be viewed as a movement from a narrative to a declarative form. Natural speech describes *what happens*; logic and programming languages answer *why* and *under what conditions* it must happen. Where natural language remains limited to intuitive judgments, logic requires explicit inference rules, and programming languages implement these rules in formal constructions.

An example from practice illustrates this distinction. In ordinary speech, we say: "If it rains, the street is wet."

For a machine, this is insufficient — it must explicitly define the structure of the relationship between facts: "rain(x) → wet(street)."

Here, language is transformed into a logical form, and the logical form — into code that can be interpreted. Thus, programming becomes not so much the art of writing as the art of translation — the translation of human meanings into the language of logic.

In this sense, language can be regarded as the surface of logic. It reveals logic's manifestations but does not replace its structure of reasoning. A programmer working with language inevitably encounters the limits of grammar, and it is precisely at this boundary that logic begins to act as a universal mechanism of semantic control. It connects language and reality, ensuring the consistency and predictability of a program's behavior.

Example of the transition from a natural statement to a logical and programmatic form:

a) **Natural statement (unstructured):**

A father is always older than his child. (1)

b) **Structuring step (identifying logical roles):**

If someone is a father, then he is older than his child. (2)

Here the phrase "if ... then" introduces an implication — a fundamental construct of logic.

c) **Formalization step (predicate logic):**

Predicates *father(x, y)* and *older(x, y)* appear, and the implication becomes a logical rule:

$\forall x, y\ (\text{father}(x, y) \rightarrow \text{older}(x, y))$ (3)

d) **Transition to program form (Prolog):**

In logic programming, the implication is implemented as an inference rule:

older(X, Y) :- father(X, Y). (4)

Here the operator :- (read as "if") expresses the reverse direction of implication:

older(X, Y) is true if father(X, Y) is true. (5)

Such a transition requires prior structuring of the statement. We must first identify the entities (X, Y), define their relations (*father*, *older*), and then establish the logical link between them. This process is an essential part of a programmer's thinking — treating information not merely as data, but as logical structures.

The transition from language to logic is not just a technical operation but a cognitive step that distinguishes the performer from the thinker. For a programmer, it means developing the ability to see beyond syntax to structure, and beyond structure to meaning. Where language ends, logic begins — and it is at this point that professional thinking is born.

4. Logic, Predicates, and Chomsky's Grammar

Logical programming does not proceed from the syntax of a language but from a logical model of reality, formalized in terms of **facts**, **rules**, and **queries**. In all three cases, the bearer of information is the **predicate** (or predicate form) — an atomic formula of predicate calculus expressing a relationship between objects and their properties.

It is precisely predicates that ensure the transition from natural to formal language: any statement, rule, or question expressed in natural language can be transformed into a predicate form suitable for logical inference. Without this step, modeling the world and extracting information from natural-language texts becomes impossible.

As R. Kowalski emphasized (Kovalski,1979), logical programming is *programming in logic*, where a program is understood as a set of logical propositions and execution is understood as a process of proof:

Algorithm = Logic + Control (6)

Logic programming = Logic without Control (7)

Thus, the logical model is the core, and the **Prolog** language is merely an instrument for its implementation. Prolog allows knowledge to be expressed in predicate form, preserving the logical structure of reasoning and minimizing the likelihood of errors associated with arbitrary control constructions typical of imperative languages.

The works of Noam Chomsky (Chomsky, 1965) marked a turning point in understanding the relationship between language and thought. He demonstrated that behind any language lies a **universal grammar** — a system of rules that enables the generation of an infinite number of sentences from a finite set of elements. This idea became not only a linguistic revolution but also the theoretical foundation for the subsequent development of artificial intelligence and programming languages.

In logical programming, Chomsky's idea manifests itself in the **rewrite-rule mechanism** (Chomsky, 1965). Grammar is not merely a set of syntactic constraints but a formal means of transitioning from natural language to logic, from an utterance to a formula. The programmer does not specify *how* to verify a fact, but only *what* must be true if certain conditions hold. This distinction is the key to professionalism: logic enables the separation of problem description from the mechanism of its solution.

Logical programming shows that **structuring** and **formalization** are two stages of a single process. First, the programmer identifies semantic elements (structuring); then expresses them in logical terms (formalization); afterwards, the structure can be refined or reorganized depending on the task context (restructuring).

This iterative scheme — "rough structure → formalization → structure refinement" — describes the natural path of thought from observation to model.

It is important to note that logical constraints serve as the *rails* along which thought moves. In imperative languages, such constraints are weaker, and the programmer is often free to act arbitrarily — increasing the risk of errors. A well-known example is the **Ariane 5 Flight 501** (European Space Agency, 1996 год), where a software module written in Ada inherited an incorrect data-conversion routine from Ariane 4. A numeric-type overflow (attempting to store a 64-bit value in a 16-bit field) led to the failure of the navigation system and the loss of the rocket.

This case is often cited as an illustration of excessive programmer freedom — when the system allows erroneous assumptions and lacks formal restrictions that could have prevented them. Logical programming, by contrast, is founded on the principle that *all* program behavior must follow from logical laws rather than arbitrary instructions. Consequently, the probability of error is reduced not only technically but also conceptually: the programmer thinks within a strict logical model rather than subjective procedures.

In this sense, logical programming is a form of **engineering self-control**. It encourages thinking not in terms of operations but in terms of truths — a mindset directly related to the professionalization of the programmer's reasoning. If imperative programming demands the ability to manage **process complexity**, then logical programming demands the ability to manage **semantic complexity**.

Hence, logical languages can be viewed not only as tools for writing programs but also as instruments for cultivating **metalogical thinking** — the ability to reflect on the structure of one's own reasoning.

The parallel between **Chomskyan linguistics** and **logical programming** is evident. In both systems, meaning arises from structure. Chomsky's grammar describes which sentences are permissible, while logic determines which statements are true. If grammar governs *form*, logic governs *content*. Their union forms the foundation for formal knowledge systems, where language becomes an instrument of reasoning.

One may say that **logic is the grammar of thought**, and **grammar is the logic of language**. Such a perspective dissolves the boundary between the humanities and the technical sciences: they appear as different manifestations of a single intellectual discipline — the art of structuring meaning.

The formalization of knowledge based on logic and predicates thus opens the path toward knowledge modeling in **artificial intelligence**. However, caution is needed regarding formulations such as *"Facts, rules, and predicates form the basis of any expert inference"* (often found in ChatGPT, Gemini, and similar LLM outputs). From a pedagogical standpoint, the juxtaposition of *facts, rules,* and *predicates* on the same level is misleading, as it confuses distinct representational layers:

1. **Conceptual level (semantics):** facts and rules — what we know.
2. **Syntactic level (form):** predicates — how it is represented.

Thus:

Knowledge (facts + rules) + Representation language (predicates) ⇒ Knowledge base.

AI systems and large language models like Gemini and ChatGPT often use generalized formulations that have become common jargon in the field. The absence of a reliable primary reference confirms that these are widespread simplifications rather than precise academic statements.

Hence, the **logical model of knowledge** can be regarded as a natural continuation of Chomsky's ideas — transferred from the domain of language to that of **thinking** and **machine intelligence**.

5. The Cost of Imperative Thinking

The **Ariane 5 Flight 501** disaster (1996) was not merely a technical malfunction but a symptom of a deeper problem in software engineering. According to the European Space Agency

report (European Space Agency, 1996), an error converting a 64-bit floating-point value into a 16-bit integer led to the destruction of a rocket worth about **$370 million**. The program, written in Ada, had worked correctly in previous versions (Ariane 4) but failed to account for new trajectory parameters.The error was not *logical* but *behavioral*—a consequence of the absence of a formal analysis of system invariants.

Imperative programming languages—such as **C, Ada, C++, Java, and Python**—require the programmer to manage states step by step rather than to verify the truth of propositions. Errors arise not at the level of the model's logic but at the level of procedural execution: *"invalid type conversion," "uninitialized variable," "incorrect call order."*All of these reflect **imperative thinking**, in which the programmer acts as an operator rather than as a logician.

By contrast, **logical programming** relies on declarative knowledge representation—stating *what* must be true, not *how* to achieve it.Errors here manifest as contradictions within the model rather than failures of behavior. The programmer works not with instructions but with **predicates** and **inference rules**, guided by the formal laws of logic.

Similar catastrophes have occurred in other domains. During the **Mars Climate Orbiter** mission (Mars Climate Orbiter Mishap Investigation Board, 1999), the spacecraft was lost due to a mismatch of measurement units between two software modules—one used pound-force, the other newtons. This error was, in essence, a violation of the logical invariant of the unit system, which could have been automatically checked in a system with declarative type constraints.

In the case of the **Therac-25** medical accelerator (1985–1987), investigated by **Nancy Leveson** (Leveson & Turner, 1993), improper management of concurrent processes led to multiple fatal radiation overdoses.The underlying cause was the absence of a formal specification of system behavior.

These tragic examples reveal a pattern: where **imperative programming** forces engineers to *debug behavior*, **logical programming** compels them to *refine knowledge*. Only logically grounded models make it possible to design systems whose reliability does not depend on the *intuition of the programmer*.

Here professionalism appears in its deepest sense: the alignment of human thought with the laws of logic, rather than with the habits of algorithmic routine.

6. The Role of Logic in Safe AI

The development of artificial intelligence has demonstrated that computational power and algorithmic complexity alone do not guarantee the reliability or predictability of a system's behavior. Without logical constraints, AI becomes akin to a living organism without reason—it can act efficiently, yet unconsciously. Logic, however, renders a system's behavior explainable and its decisions verifiable and transparent. Freedom without structure is chaos. This principle holds not only in society but also in computational environments. Just as law regulates human behavior in a legal state, logic regulates the behavior of artificial intelligence in the digital space. It defines the boundaries of what is permissible, establishes criteria of truth, and provides a mechanism of control.

In imperative programming languages, the developer is fully responsible for the sequence of actions, which increases the likelihood of errors. In logical languages, the focus shifts from *how* to *what*: the system itself searches for solutions based on logically specified conditions. This approach reduces risks arising from human error and enhances the system's resilience.

Moreover, logic ensures traceability of decisions—the ability to reconstruct the chain of reasoning leading to a specific conclusion. This feature becomes especially important in the era of neural networks, where most reasoning occurs within a "black box." Integrating logical layers into AI architectures restores transparency and interpretability, two essential requirements for safe systems.

If AI is considered as a collection of models of the world, logic functions as its internal law. It performs the same role as societal laws: determining which actions are permissible, which are prohibited, and on what basis decisions are made. Thus, logic is not a limitation but a guarantee of freedom that does not devolve into arbitrariness. Without logic, intelligence remains empirical and reactive, not deliberate. Logical structure provides measure, transforming AI into a tool capable of responsibility and predictability. In this sense, logic is not only a method for programming but also a moral mechanism within AI architecture.

From Unstructured Reality to Information

The transition from unstructured reality to information is a fundamental process underpinning not only artificial intelligence but also cognition itself. In the surrounding world, data exist as events, properties, and relations that have no form until they are perceived, selected, and interpreted.

Structuring is the first step toward understanding. It begins when a subject discerns stable connections within the chaos of observations and transforms them into knowledge. In the classical sense, information is not merely data but data placed within a context. Context defines structure, and structure enables interpretation.

As noted by Noam Chomsky (Chomsky, 1965), grammar functions as a mechanism that generates all admissible structures of language. Logic, in turn, endows these structures with meaning. Within this interaction lies the foundation of what may be called intellectual information processing.

Technologically, this process can be represented as the following chain:

environment → sensor / expert → data → information → natural language → structuring → representation (in logic) → Chomsky grammar → Turing machine (processing).

The generalized technological sequence unfolds as follows:

- **Environment** — the source of data and stimuli. ↳ Everything begins with the external world—physical, social, or informational.
- **Sensor / expert** — the act of perception or interpretation. ↳ In humans, these are the senses and consciousness; in machines, sensors and perception models.
- **Information → knowledge** — the transition from raw data to meaningful relations. ↳ At this stage, facts, regularities, and connections are formed—elements that can be transmitted and reasoned about.
- **Natural language** — the medium of recording and exchanging knowledge. ↳ It makes knowledge communicable but remains inherently informal.
- **Structuring** — the first step toward formalization. ↳ Roles, relations, and categories are defined (originating with Aristotle and Socrates).
- **Representation in logic** — giving knowledge a formal structure. ↳ Using predicates, rules, and implications—i.e., the language of logical relations.
- **Chomsky grammar** — the transitional level between language and logic. ↳ Formal grammars enable algorithmic manipulation of syntactic structures.
- **Turing machine (processing)** — a universal model for executing formalized procedures.

This sequence represents a technology of information processing—from perception to understanding and further to automated action. Logic serves as the central *hinge* of this chain, linking human meaning (natural language) with machine executability (Turing computation).

The sequence illustrates the progression from perception to understanding, from the empirical to the rational. At each stage, uncertainty decreases while the level of organization increases. The environment produces signals; the sensor or expert captures them as data; language gives them form; logic imparts meaning; and the Turing machine enables computation.

Thus, the logical form of knowledge functions not only as an analytical tool but also as a technology of information processing, transforming cognition into computation and enabling the transition from subjective understanding to formalized knowledge.

For this reason, structuring information is a key step in the creation of artificial intelligence systems. Without logical representation, knowledge cannot be processed; without structuring, it cannot be understood. Logic and structure act as a filter between the chaos of reality and the order of thought.

7. Designing a Curriculum for Future Programmers

Training the programmer of the future requires not only knowledge of technologies but also the ability to think logically, systemically, and responsibly. Technical skills quickly become outdated, whereas the ability to structure information and formulate tasks remains the foundation of professionalism.

Contemporary educational programs generally emphasize the syntax of programming languages rather than logic as the substantive basis of computation. As a result, students learn to write code without understanding the deep connection between the form of expression and the structure of the knowledge it conveys.

This gap can be bridged by placing the **logical paradigm** at the core of programmer training. Logic ensures not only universality but also transparency of thought. Within logical frameworks, a programmer cannot "write arbitrarily": each statement, fact, or rule must be meaningful and verifiable. This represents a form of professional responsibility in intellectual work.

The proposed structure of the educational cycle can be outlined as follows:

- **Year 1 — Fundamentals of Logic and Information Structuring:** Introduction to logical thinking, basic forms of inference, propositional and predicate logic, Aristotle's categories as tools for concept analysis.
- **Year 2 — Theory and Practice of Logic Programming:** Mastery of the Prolog language, construction of knowledge bases, the relationship between facts, rules, and queries; transition from natural language to formalized representation.
- **Year 3 — Systems Theory and Systemic Thinking:** Understanding the world as a set of interconnected elements; developing the ability to perceive structure beneath the surface of phenomena.
- **Year 4 — Knowledge Representation and Processing:** Methods for formalizing subject domains, ontologies, and declarative programming in logic as a model for safe AI.

Each course develops not merely technical skills but a **culture of thinking**—the ability to express knowledge formally, correctly, and reproducibly. This ability transforms the programmer into a **knowledge architect** and artificial intelligence into a **tool rather than a threat**.

Such training prepares not just an executor of algorithms but a specialist capable of understanding the meaning of their programs, evaluating the consequences of their operation, and explaining the decisions made. In other words, it enables the creation of ethical and safe AI systems.

Thus, reflections on the nature of logic as the foundation of rational thought lead us to the necessity of understanding its connection with language and freedom—three interconnected dimensions of both human and machine intelligence.

8. Language, Logic, and Freedom

At first glance, language appears merely as a means of communication, and logic as a tool for reasoning. However, in a deeper sense, language is a form of the existence of thought, while logic is the form of its movement. Freedom, in turn, represents the ability to act according to reason rather than against it. Thus, language, logic, and freedom form a unified triad: language records

meaning, logic ensures its consistency, and freedom enables conscious choice among possible courses of action.

Freedom is inconceivable without logic. Deprived of a logical foundation, it degenerates into arbitrariness; conversely, logic without freedom becomes dogma. In both programming and life, it is necessary to seek a balance between strictness and flexibility: excessive rigidity of rules paralyzes creativity, while their absence leads to chaos. Only a logical structure allows one to create freely, but within meaningful bounds.

True freedom does not imply the rejection of constraints; on the contrary, it presupposes their conscious acceptance for the sake of greater effectiveness and responsibility. Just as musical notes constrain a composer yet make music possible, so logic sets the boundaries of thought, allowing it to sound clear and harmonious.

Logic can be called the language of reason, and programming—its modern form. Logic programming is an attempt to speak to a machine in the language of reason. This language requires not only technical literacy but also inner discipline, the ability to think precisely and justly. Without these qualities, even the most advanced artificial intelligence will reproduce not reason itself but only its semblance.

The quality of language determines the quality of freedom. A poor language makes a person vulnerable to manipulation; imprecise logic leads to errors; unreflective freedom leads to self-destruction. Therefore, the development of logical thinking becomes not merely an educational task but a condition for preserving human dignity in the digital age.

9. Logic Programming and Artificial Intelligence

Logic programming is not merely a way of writing programs; it is an attempt to represent knowledge in a form suitable for reasoning. In traditional programming, a person specifies how actions should be carried out; in logic programming, they define what must be true. This paradigm is based on describing facts and rules, from which the machine independently derives consequences. In this way, logic programming is closer to the process of thinking than to a sequence of algorithmic instructions.

In essence, logic programming models rational thought, governed by principles of causality and justification. It naturally became the foundation for research in artificial intelligence, as it allows not only the modeling of behavior but also the formalization of the reasoning process itself.

Modern AI systems are primarily built on neural network technologies, which simulate the associative, empirical side of human thinking—the ability to recognize patterns, images, and sounds. Logic programming, in contrast, reproduces the rational side of thought—the ability to explain, prove, and draw conclusions. The neural approach provides intuition, while the logical approach provides understanding. Without a logical component, artificial intelligence remains merely an instinctive mechanism, capable of guessing but unaware of why its decisions are correct or erroneous.

Safe and responsible artificial intelligence is impossible without logic. The logical component is necessary for self-control, analysis of motives, and justification of decisions. Without it, we obtain a tool entirely dependent on the context and intentions of its developers. Introducing logical constraints, on the other hand, enables accountability—for both machines and the humans who create them.

Logic programming can be seen as a method of teaching AI to reason rather than merely compute. It directs the system to seek explanations rather than execute commands. In this sense, logic programming serves not only as an engineering tool but also as an educational model, demonstrating how to think structurally, honestly, and consistently.

This approach should also form the foundation of future education. It is important to teach not just how to write code but how to formulate meaning in logical forms. Programming, in this context, becomes an expression of professionalism and freedom. A person capable of clearly

articulating what is true and why is protected from manipulation, errors, and mindless obedience to machines.

10. Education and the Logic of Freedom

Education has always been a dialogue between freedom and necessity. On one side lies the need to master knowledge and discipline; on the other — the desire for creativity and independent thought. Logic serves as the bridge between these two dimensions, transforming discipline into a path toward genuine freedom.

A logically trained mind is capable of distinguishing meaning from noise, essence from appearance. It resists manipulation and can construct its own system of values and arguments. Therefore, teaching logic should not be limited to philosophy or mathematics — it must become the foundation of all education, especially in the digital age.

The ability to formalize information — to express it in logical and unambiguous forms — becomes a new form of literacy. Just as in earlier centuries people learned to read and write to participate in cultural life, today they must learn to think logically in order to remain free in a world governed by algorithms.

Programming, when taught as logical modeling, becomes a tool for cultivating responsibility and intellectual autonomy. It reveals the connection between thought and consequence, between the precision of expression and the safety of decisions. A student who understands how logic constrains and liberates thought learns to respect both the limits of reason and the dignity of freedom.

True education is not about accumulating information but about mastering the structures of reasoning that allow one to use information wisely. Logical thinking thus becomes not only a professional competence but also a moral orientation: to act rationally is to act responsibly.

In this sense, logic is not a constraint on freedom but its precondition. To think logically is to act consciously — and therefore freely.

11. Conclusion

Logic forms the foundation of professional thinking for programmers and underpins the development of safe and explainable AI. By structuring knowledge and reasoning, it ensures transparency, traceability, and accountability, transforming freedom into responsible decision-making.

Educating future programmers within a logical paradigm fosters professional competence, moral responsibility, and the ability to design resilient AI systems. Integrating logic, language, and structured reasoning enables the creation of intelligent systems where freedom, responsibility, and safety are inseparably linked.

References:

1. Chomsky, N., 1965. Aspects of the theory of syntax. Cambridge, Massachusetts: MIT Press.
2. European Space Agency, 1996. Report of the inquiry board on the Ariane 5 Flight 501 failure. Paris: ESA. Available at: https://www.esa.int/Newsroom/Press_Releases/Ariane_501_-_Presentation_of_Inquiry_Board_ report (Accessed: 30 September 2025).
3. Kowalski, R.A., 1979. Algorithm = logic + control. Communications of the ACM, 22(7), pp.424-436.
4. Kowalski, R. and Sergot, M., 1990. The use of logical models in legal problem solving. Ratio Juris, 3(2), pp.201-218.
5. Leveson &. Turner. An Investigation of the Therac-25 Accidents. IEEE Computer Society. July, 1993,Vol. 26, No. 7, pp. 18–41.
6. Mars Climate Orbiter Mishap Investigation Board. (1999). Mars Climate Orbiter Mishap Investigation Board Phase I Report: November 10, 1999. Washington, D.C.: NASA.
7. Pelin, N., 2023. Искусственный интеллект, основанный на логике и право. В: Omul, criminologia, ştiinţa. 2-е изд. Кишинэу: Institutul de ştiinţe penale şi criminologie aplicată, Т. 1, с. 562–568. Доступно по адресу: https://ibn.idsi.md/sites/default/files/imag_file/562-568.pdf (Дата обращения: 30.09.2025).