CTE Workshop Proceedings, 2025, Vol. **12**, pp. 399–425 https://doi.org/10.55056/cte.838

Teaching logic programming: a review

Serhiy O. Semerikov[1,2,3,4,5], Iryna S. Mintii[6,3,1,4,7,8,5] and Natalia V. Moiseienko[1]

[1]*Kryvyi Rih State Pedagogical University, 54 Universytetskyi Ave., Kryvyi Rih, 50086, Ukraine*
[2]*Kryvyi Rih National University, 11 Vitalii Matusevych Str., Kryvyi Rih, 50027, Ukraine*
[3]*Institute for Digitalisation of Education of the NAES of Ukraine, 9 M. Berlynskoho Str., Kyiv, 04060, Ukraine*
[4]*Zhytomyr Polytechnic State University, 103 Chudnivsyka Str., Zhytomyr, 10005, Ukraine*
[5]*Academy of Cognitive and Natural Sciences, 54 Universytetskyi Ave., Kryvyi Rih, 50086, Ukraine*
[6]*University of Łódź, 68 Gabriela Narutowicza Str., 90-136 Łódź, Poland*
[7]*Lviv Polytechnic National University, 12 Stepana Bandery Str., Lviv, 79000, Ukraine*
[8]*Kremenchuk Mykhailo Ostrohradskyi National University, 20 University Str., Kremenchuk, 39600, Ukraine*

Abstract. Logic programming constitutes a significant paradigm within computer science, offering a unique approach to programming based on formal logic rather than conventional imperative instructions. This review examines contemporary methodologies for teaching logic programming, spanning various paradigms including Prolog, Answer Set Programming (ASP), Datalog, and Constraint Logic Programming (CLP). Through a comprehensive analysis of literature, we identify effective pedagogical strategies, common obstacles faced by educators and students, and emerging trends in instructional techniques. Our findings reveal that visualization tools, problem-based learning, integration with other programming paradigms, and contextual application-based approaches demonstrate the most promise for enhancing student comprehension and engagement. Additionally, we explore cognitive challenges specific to declarative thinking, educational challenges related to curriculum integration, and motivational issues that affect student learning outcomes. This review offers evidence-based recommendations for practitioners and identifies promising directions for future research in logic programming education.

Keywords: logic programming, computer science education, Prolog, Answer Set Programming, Datalog, constraint logic programming, declarative programming, teaching methods

1. Introduction

Logic programming represents a distinct paradigm within the landscape of computer science education, diverging significantly from the imperative and object-oriented approaches that dominate mainstream curricula. Founded on principles of mathematical logic, particularly first-order predicate calculus, logic programming enables problem-solving through declaration of knowledge rather than specification of control flow. This fundamental shift in perspective presents unique challenges and opportunities for both educators and learners.

The distinct nature of logic programming warrants careful consideration of teaching methodologies. As Hanus [22] notes, the declarative approach requires students to develop alternative ways of computational thinking – focusing on what needs to

0000-0003-0789-0272 (S. O. Semerikov); 0000-0003-3586-4311 (I. S. Mintii); 0000-0002-3559-6081 (N. V. Moiseienko)
semerikov@gmail.com (S. O. Semerikov); mintii@iitlt.gov.ua (I. S. Mintii); n.v.moiseenko@gmail.com (N. V. Moiseienko)
https://kdpu.edu.ua/semerikov (S. O. Semerikov); https://acnsci.org/mintii/ (I. S. Mintii); https://kdpu.edu.ua/personal/nvmoiseienko.html (N. V. Moiseienko)

be accomplished rather than how to accomplish it. This conceptual reorientation demands specialized pedagogical strategies.

Despite its theoretical elegance and practical applications in artificial intelligence, knowledge representation, and constraint solving, logic programming often occupies a marginal position in computer science curricula. Hynek [25] observes that while logic programming acquired a firm place within computer science programs decades ago, the rapid development of new programming languages has pushed it away from the forefront of widely utilized programming paradigms. Consequently, questions arise regarding effective approaches to teaching logic programming and its place within modern computing education.

This review examines current approaches to teaching logic programming across its major paradigms: Prolog, Answer Set Programming (ASP), Datalog, and Constraint Logic Programming (CLP). We analyze pedagogical strategies, technological tools, curriculum integration approaches, and assessment methods. Additionally, we explore cognitive, educational, and motivational challenges that impact the effectiveness of logic programming instruction.

Our analysis is guided by several research questions:

1. What are the predominant approaches to teaching logic programming across different paradigms?
2. Which teaching methods demonstrate empirical evidence of effectiveness?
3. What are the primary obstacles to student comprehension and engagement?
4. How do teaching approaches differ across educational levels and contexts?
5. What emerging trends and innovations show promise for improving logic programming education?

2. Background and context

2.1. Definition and historical context

Logic programming emerged in the 1970s from debates concerning procedural versus declarative representations of knowledge in artificial intelligence [31]. As Ligeza [35] explains, the fundamental idea, originally proposed by Robert Kowalski, consists of applying a subset of First-Order Logic for declarative encoding of knowledge and employing specific resolution theorem proving strategies for inference. This combination of declarative specification with automated reasoning created a powerful programming paradigm.

The development of Prolog (Programming in Logic) by Alain Colmerauer and Philippe Roussel at the University of Aix-Marseille in the early 1970s marked the first practical implementation of logic programming principles. Since then, logic programming has evolved into a diverse family of languages and systems, each emphasizing different aspects of declarative computation.

2.2. Major logic programming paradigms

Contemporary logic programming encompasses several distinct paradigms, each with unique characteristics and applications:

1. *Prolog and its variants* remains the most widely taught and utilized logic programming language. Based on Horn clause logic, it employs a backward-chaining inference mechanism. According to Gelfond et al. [19], the advent of Prolog in 1972 started a revolution that "enabled us to think previously impossible thoughts, and ushered in both logic programming and the declarative programming paradigm". Modern Prolog systems like SWI-Prolog, SICStus, and Ciao extend basic logic programming with features such as constraints, modules, and higher-order programming.

2. *Answer Set Programming (ASP)* evolved from research on the semantics of negation in logic programs. As Schaub and Woltran [52] explains, ASP offers "a high-level modeling language paired with high-performance solving technology", making it particularly suitable for knowledge representation and combinatorial optimization problems. Unlike Prolog's backward-chaining approach, ASP employs a more declarative paradigm where solutions are represented as answer sets, computed through sophisticated constraint solving techniques.
3. *Constraint Logic Programming (CLP)* extends logic programming with constraint solving capabilities. According to Gavanelli and Rossi [17], CLP has been "one of the most successful branches of Logic Programming", attracting interest from both theoreticians and practitioners. By integrating constraints over specific domains (e.g., real numbers, finite domains), CLP enables efficient solving of complex problems in scheduling, planning, and optimization.
4. *Datalog* combines logic programming with database query languages. With its foundation in deductive databases, Datalog has experienced renewed interest in applications such as program analysis, information extraction, and knowledge graphs. Sáenz-Pérez [51] describes Datalog as a deductive database system that supports both Datalog and SQL query languages, enabling educational exploration of both paradigms within a unified framework.

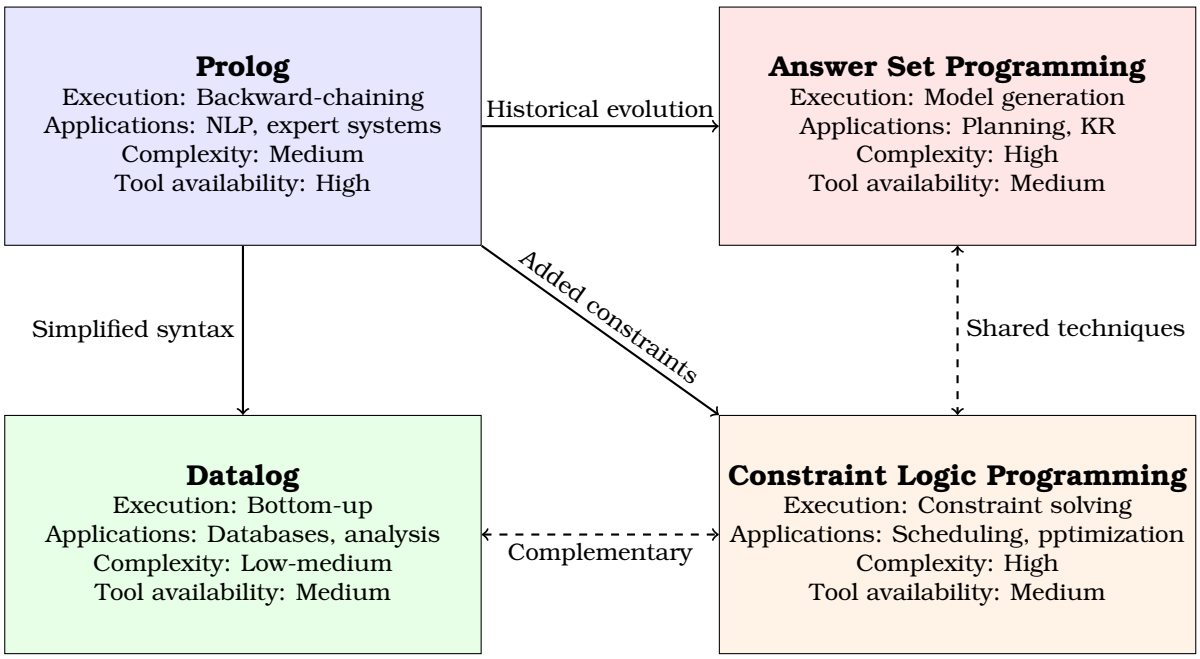

Figure 1: Comparison of major logic programming paradigms, illustrating their key characteristics, applications, and relationships. The arrows indicate evolutionary relationships and conceptual connections between paradigms.

2.3. Current position in computing curricula

Despite its theoretical significance and practical applications, logic programming occupies an inconsistent position within computing curricula globally. Hynek [25] raises the question of "whether there are still some good and justified reasons to keep Logic Programming in the present ICT specialists curricula", noting that while logic programming was once prominent in European and Asian universities, it has been marginalized as newer programming languages gained prominence.

The teaching of logic programming often occurs within specialized courses on programming languages, artificial intelligence, or formal methods. Villadsen and

Jacobsen [59] describes two university courses where logic programming is taught: a bachelor-level course on logical systems and logic programming, and a master's course on automated reasoning. This positioning reflects both the specialized nature of logic programming and its connections to broader areas of computer science.

Recent initiatives have also explored introducing logic programming at earlier educational stages. Rodríguez and Cecchi [50] advocates for the early integration of programming within compulsory education, presenting a methodological proposal called "Metagame" that facilitates the effective integration of logic programming in primary education through collaborative and gamifying strategies.

3. Teaching approaches by paradigm

3.1. Prolog-based instruction

3.1.1. Traditional approaches: syntax-first vs. problem-first

Two principal approaches dominate Prolog instruction: syntax-first and problem-first methodologies. The syntax-first approach begins with language constructs and gradually introduces problem-solving techniques, while the problem-first approach starts with engaging challenges and introduces syntax as needed.

Wick and Stevenson [63] observes that "the traditional way to teach Prolog is to have students start by writing recursive mathematical definitions of the problems they are trying to solve". However, this approach may not integrate well with broader programming language curricula. As an alternative, they propose introducing Prolog through a demonstration language like Scheme, finding this approach equally effective for novice learners.

The problem-first approach has gained traction due to evidence suggesting improved student engagement. Bellström and Thorén [6] argues that through visualization, "the logic component in the learning process can be empowered to facilitate an application perspective to achieve syntax". This represents a fundamental shift from traditional syntax-first methodologies.

Figure 2 illustrates the comparative learning trajectories of these approaches. While problem-first approaches typically result in faster initial comprehension, both methodologies tend to converge as students reach advanced proficiency levels.

3.1.2. Visual learning tools and execution tracers

Visualization plays a crucial role in enhancing students' understanding of Prolog's execution model, particularly the challenging concepts of unification, backtracking, and resolution. Vosinakis, Anastassakis and Koutsabasis [60] notes that "the limited availability of visual teaching aids for LP can lead to low motivation for learning", highlighting the importance of visual tools in sustaining student engagement.

Several visual learning environments have demonstrated effectiveness. The MeLoISE platform presented by Vosinakis, Koutsabasis and Anastassakis [61] provides a collaborative visual interface to Prolog programming within virtual worlds. Their evaluation revealed that "students performed very well and were enthusiastic with the new environment", suggesting substantial benefits from visual learning approaches.

Similarly, Prolog visualization systems using Logichart diagrams can help students understand execution flow. As Adachi [1] explains, these tools allow students to "visually trace Prolog execution (goal calling, success, and failure)" and observe "dynamic change in a Prolog program by calling extra-logical predicates".

3.1.3. Problem-based learning methods

Problem-based learning (PBL) has emerged as a particularly effective approach for teaching Prolog. This methodology situates learning in complex tasks that require

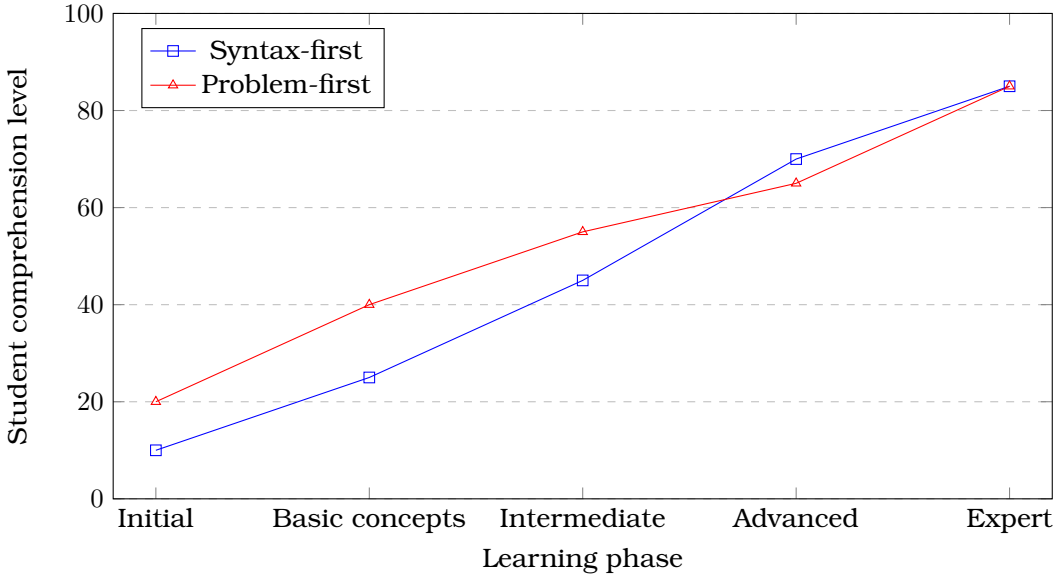

Figure 2: Comparative student comprehension levels across learning phases for syntax-first versus problem-first teaching approaches in Prolog instruction. The problem-first approach shows stronger initial and early-phase comprehension, while both approaches converge at advanced levels.

scaffolding to help students engage in sense-making and manage their problem-solving processes.

Wu and Lo [64] found that incorporating worked examples into problem-based learning activities significantly enhanced students' logic problem-solving performance. Their approach involved "a series of geometric logic problems developed and tested in a pilot study", demonstrating how structured PBL can develop logical thinking skills.

The effectiveness of PBL in Prolog instruction is further supported by Souza and Bittencourt [55], who analyzed the impact of using a problem-based learning approach on student motivation and engagement in an introductory programming course. They discovered that "the PBL approach relates to students' attention and relevance", though they caution that "there must be careful problem design and planning in order to positively relate to student motivation and engagement".

3.1.4. Integration with AI curriculum

Prolog instruction frequently occurs within broader artificial intelligence curricula, where its logical foundations serve as a bridge between theoretical AI concepts and practical implementation.

Zhang et al. [65] demonstrate the integration of logic programming with science education, developing "two modules: one for chemistry and the other for chemistry and physics" implemented in an elective course for 8th graders. Their findings indicate that a "Logic Programming based approach is accessible to the students", suggesting potential for broader integration across educational contexts.

At university level, Page [44] argues for computational logic in the undergraduate curriculum, noting that "logic provides the mathematical basis for hardware design and software development". The integration of logic programming within computer science programs enables students to develop reasoning skills applicable across various domains.

3.2. Answer Set Programming pedagogy

3.2.1. Modeling-first approaches

ASP pedagogy frequently employs modeling-first approaches, emphasizing problem representation over syntactic details. This approach aligns with ASP's declarative nature, where problem specification takes precedence over execution mechanisms.

Paramonov et al. [45] introduced Sketched Answer Set Programming (SkASP), a novel method aimed at facilitating ASP modeling. In this approach, "the user writes partial ASP programs, in which uncertain parts are left open and marked with question marks" while providing examples of desired program behavior. The system then synthesizes a complete ASP program, reducing the complexity of model development for novices.

The effectiveness of modeling-first approaches is reinforced by Schaub and Woltran [52], who notes that ASP's "declarative approach allows a user to concentrate on a problem's specification rather than the computational means to solve it". This makes ASP particularly suitable for teaching key AI techniques through concise and elaboration-tolerant representations.

3.2.2. Tools for ASP education

Several specialized tools support ASP education. Bertagnon and Gavanelli [7] presents ASPECT (Answer Set rePresentation as vEctor graphiCs in laTex), a sub-language of ASP that enables users to "directly define, in an intuitive and declarative way, a graphical representation of the answer set". This visualization capability helps students interpret the often complex and verbose output of ASP solvers.

Another notable educational tool is onlineSPARC, described by Marcopoulos and Zhang [39] as "an online ASP environment with a self-contained file system and a simple interface". This platform allows students to edit logic programs and perform various tasks, including querying programs and producing visualizations based on answer sets. By eliminating the complexity of downloading and installing specialized software, onlineSPARC reduces barriers to ASP adoption in educational settings.

Experimental environments like asprilo, presented by Gebser et al. [18], provide domain-specific frameworks for teaching ASP concepts. This system supports "experimental studies of approaches addressing complex dynamic applications", with a focus on robotic intra-logistics, combining multi-agent planning, reasoning about action, and resource management within a unified framework.

3.2.3. Integration with knowledge representation courses

ASP is frequently taught within the context of knowledge representation courses, where its formal semantics and expressive power make it particularly suitable for modeling complex domains.

Genesereth [20] identifies General Game Playing (GGP) as a "killer app for Logic Programming", finding it "especially well-suited to LP" for educational purposes. By analyzing the components of GGP – game description, game playing, and metagaming – educators can illustrate how logic programming techniques apply to engaging real-world problems.

The integration of ASP with other reasoning paradigms is exemplified by El-Khatib, Pontelli and Son [14], who presents ASP-PROLOG, "a system which provides a tight and well-defined integration of a multi-paradigm logic programming system (CIAO Prolog) and Answer Set Programming". This combined approach enables students to explore different reasoning mechanisms within a unified framework, enhancing their understanding of knowledge representation techniques.

3.3. Constraint Logic Programming

3.3.1. Teaching approaches

Constraint Logic Programming (CLP) combines logic programming with constraint solving, requiring specialized pedagogical approaches that address both paradigms.

Szeredi [56] describes experiences of teaching constraints through logic puzzles at the Budapest University of Technology and Economics. This approach uses puzzles to illustrate constraint programming techniques, making abstract concepts concrete through engaging, game-like problems.

A comprehensive approach to teaching CLP is presented by Apt and Wallace [3] in their work on constraint logic programming using Eclipse. They propose a systematic introduction through carefully-chosen examples that guide students through the language and demonstrate its expressive power, versatility, and utility.

Fages [16] advocates using Prolog with its constraint-solving libraries as "a unique language to teach all aspects of constraint programming". By adding higher-level mathematical modeling language constructs to Prolog libraries, educators can create an integrated environment for teaching constraint-based modeling, search programming, and constraint solving.

3.3.2. Application-based learning

Application-based learning approaches in CLP emphasize practical problem-solving in domains such as scheduling, planning, and optimization.

Kerdprasop and Kerdprasop [28] demonstrates the use of constraint logic programming for frequent pattern discovery in knowledge discovery tasks. Their implementation on the ECLiPSe constraint system provides students with hands-on experience applying CLP techniques to data mining problems.

Another practical application is presented by Bădică, Bădică and Ivanović [5], who uses Constraint Logic Programming to model block-structured scheduling processes. This approach allows students to explore the relationship between business process modeling and project scheduling using CLP as an integrative framework.

Dal Palù et al. [11] describes using CLP for exploring protein fragment assembly, illustrating how constraint programming can address complex bioinformatics challenges. This type of domain-specific application helps students understand the practical utility of CLP in scientific research.

3.4. Datalog and deductive databases

3.4.1. Current teaching practices

Datalog, with its foundations in deductive databases, occupies a unique position between logic programming and database management systems.

Sáenz-Pérez [51] presents the Datalog Educational System (DES), "a deductive database which supports both Datalog and SQL as query languages". This dual-language approach enables students to explore the relationship between logic programming and relational database concepts within a unified framework.

Teaching Datalog often involves compiler-based approaches. Cuteri and Ricca [10] describes a compiler for stratified Datalog programs that transforms a given program into a problem-specific executable implementation. This technique helps students understand the relationship between declarative specifications and efficient execution mechanisms.

3.4.2. Tools and platforms

Several specialized tools support Datalog education. Tran, Kato and Hu [58] presents a counterexample-guided debugger for non-recursive Datalog, which helps students understand the behavior of their programs through concrete counterexamples when expectations are not met.

Another educational approach involves constraining Datalog programs. Skvortsov et al. [54] presents Logica, "an open-source logic programming language" that extends Datalog with support for numerical computations and integrates with data science toolchains. This modern implementation connects traditional Datalog concepts with contemporary data science practices, enhancing relevance for students.

The timeline in figure 3 illustrates the shift from syntax-focused methods to interactive, visualization-based, and integrated approaches, alongside the development of increasingly sophisticated educational tools.

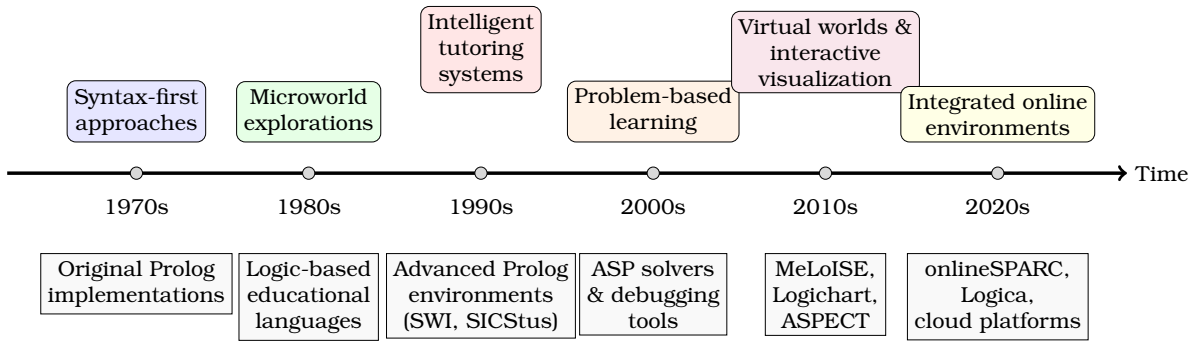

Figure 3: Evolution of pedagogical approaches and tools for teaching logic programming from the 1970s to the present day.

4. Cross-cutting pedagogical themes

4.1. Visualization approaches

Visualization plays a critical role in teaching abstract concepts across all logic programming paradigms. Effective visualization techniques help students understand execution models, develop intuition for declarative thinking, and interpret program results.

4.1.1. Types of visualizations

Logic programming visualizations span a spectrum from execution tracers to conceptual representations. Malandrino et al. [38] presents a diagrammatic language for visualizing logic based on spatial, graphical, and symbolic notations. Their V-Logic approach supports "different visual representation schemes" to provide "greater insight than any alone".

Eppler [15] compares various visualization formats – concept maps, mind maps, conceptual diagrams, and visual metaphors – finding that their combination "can play to the strength of each one". When applied to logic programming concepts, these complementary visualization techniques can enhance motivation, attention, understanding, and recall.

Program execution visualization is particularly important in logic programming education. Loboda and Brusilovsky [36] evaluates user-adaptive explanatory program visualization, finding that "explanatory visualization allows students to substantially increase the understanding of a new programming topic". Their eye-tracking studies revealed that "adaptive visualization captivates attention more than its non-personalized counterpart and is more interesting to students".

4.1.2. Evidence of effectiveness

Research consistently demonstrates the effectiveness of visualization in logic programming education. Awasekar [4] conducted an experimental study comparing program visualization with traditional teaching approaches, finding "a statistically significant higher performance on a post-test for the program visualization group

compared to the traditional group". Notably, this effect was observed "regardless of students' visual/verbal learning style", suggesting broad applicability.

The impact of visualization tools extends beyond test performance to motivation and engagement. Vosinakis, Anastassakis and Koutsabasis [60] found that platforms for teaching logic programming in virtual worlds yielded "encouraging results regarding group learning performance and user experience". By enabling visual interpretation and verification of program results, these environments make abstract concepts concrete and accessible.

Visualization effectiveness depends on thoughtful design. Jones et al. [27] demonstrates that applying established visualization principles to models can increase accuracy, perceived message credibility, and aesthetic appeal while decreasing mental effort and review time. These findings suggest that carefully designed visualizations can significantly enhance the learning experience for logic programming students.

Table 1 summarizes key visualization approaches used in logic programming education, highlighting their benefits and limitations. This diversity of approaches allows educators to select visualization techniques appropriate to specific learning objectives and student needs.

Table 1
Comparison of visualization approaches in logic programming education.

Approach	Description	Benefits	Limitations
Execution tracers	Visualize program execution, including unification steps, variable bindings, and backtracking	Enhances understanding of execution model; makes abstract processes concrete	Technical complexity; may overwhelm beginners
Virtual worlds	3D environments where program execution is represented through object interactions	Engaging; promotes collaborative learning; provides intuitive metaphors	Resource-intensive; may distract from core concepts
Conceptual diagrams	Visual representations of logical relationships and structures	Promotes abstract thinking; aids in problem formulation	Less effective for understanding dynamic execution
Answer set visualization	Graphical representation of answer sets for ASP programs	Simplifies interpretation of complex results; facilitates debugging	Language-specific; limited to certain problem types
Constraint visualizers	Visual representation of constraint propagation and solution spaces	Helps understand constraint solving process; illustrates pruning mechanisms	Requires prior understanding of constraint concepts

4.2. Problem-based learning

Problem-based learning (PBL) approaches situate logic programming instruction within engaging, authentic problem contexts. This methodology has proven particularly effective for teaching declarative programming paradigms.

4.2.1. Types of Problems and Domains

Logic programming education employs diverse problem domains, each offering unique pedagogical advantages. Wu and Lo [64] explores geometric logic problems, demonstrating how worked examples can enhance students' problem-solving performance. Their approach combines structured guidance with opportunities for independent problem-solving.

Puzzle-based problems feature prominently in logic programming education. Szeredi [56] describes using logic puzzles to teach constraint programming, while Perez-Lancho et al. [47] presents the MAFIA tool, which "helps students to understand the basic concepts of logic in an interactive way" through engaging case-based scenarios.

Real-world domains provide authentic contexts for learning. Genesereth [20] advocates using General Game Playing as an application area "especially well-suited to LP", while Zhang et al. [65] demonstrates integrating logic programming with chemistry and physics education. These domain-specific applications help students appreciate the practical utility of logic programming techniques.

4.2.2. Scaffolding approaches

Effective PBL in logic programming requires thoughtful scaffolding to manage cognitive load and support progressive skill development.

Choo [9] discusses scaffolding in problem-based learning, distinguishing between "hard scaffolds" (static supports developed based on learner difficulties) and "soft scaffolds" (teacher actions responding to specific learner needs). Both types play important roles in supporting students as they navigate the challenges of logic programming.

Tiantong and Teemuangsai [57] presents four scaffolding modules for collaborative problem-based learning: metacognitive scaffolding, conceptual scaffolding, strategic scaffolding, and procedural scaffolding. These complementary approaches address different aspects of the learning process, from problem comprehension to solution implementation.

Tiered scaffolding approaches can be particularly effective. Lape [33] describes a "tiered scaffolding of Problem-Based Learning techniques" ranging from highly guided to entirely unformed problems. This progressive approach helps students develop independence while managing cognitive load appropriately.

4.3. Integrated curriculum approaches

Integration of logic programming with broader curriculum elements enhances relevance and promotes transfer of knowledge across domains.

4.3.1. Logic programming in different course contexts

Logic programming appears in diverse curricular contexts, from specialized programming courses to broader computer science and STEM education.

Boluk [8] advocates for integrated curriculum design as "an empowering and engaging pedagogical approach" that unites core courses into a coherent whole. This approach can position logic programming within a broader context of computing skills and concepts.

At the primary and secondary education levels, Rodríguez and Cecchi [50] presents Metagame, a methodological proposal for integrating logic programming in primary education through "collaborative and gamifying strategies". This early introduction can develop foundational computational thinking skills that support later learning.

In university settings, Villadsen and Jacobsen [59] describes using the proof assistant Isabelle in two courses on logic and automated reasoning, demonstrating how logic programming concepts can be integrated with formal methods education.

4.3.2. Cross-paradigm teaching

Teaching logic programming alongside other programming paradigms presents both challenges and opportunities.

Ragonis and Haberman [48] explores linking different programming paradigms, noting that "educators recommend that students become acquainted with different programming paradigms in order to acquire alternative ways of computational thinking". They specifically examine object-oriented programming and logic programming

in the context of using inheritance for knowledge representation and problem solving.

A more comprehensive approach is presented by Oprea [43], who developed Onto-DeclarProg, "an educational ontology on declarative programming" that integrates ontologies on logic programming and functional programming. This framework helps students understand the relationships between different paradigms within a unified conceptual structure.

Hanus [23] discusses functional logic programming, presenting how languages like Curry combine "the most important declarative programming paradigms". This multi-paradigm approach provides "a common platform for the research, teaching, and application of integrated functional logic languages".

Figure 4 illustrates connections between logic programming and related disciplines, application domains, core curriculum elements, and integrated teaching approaches.

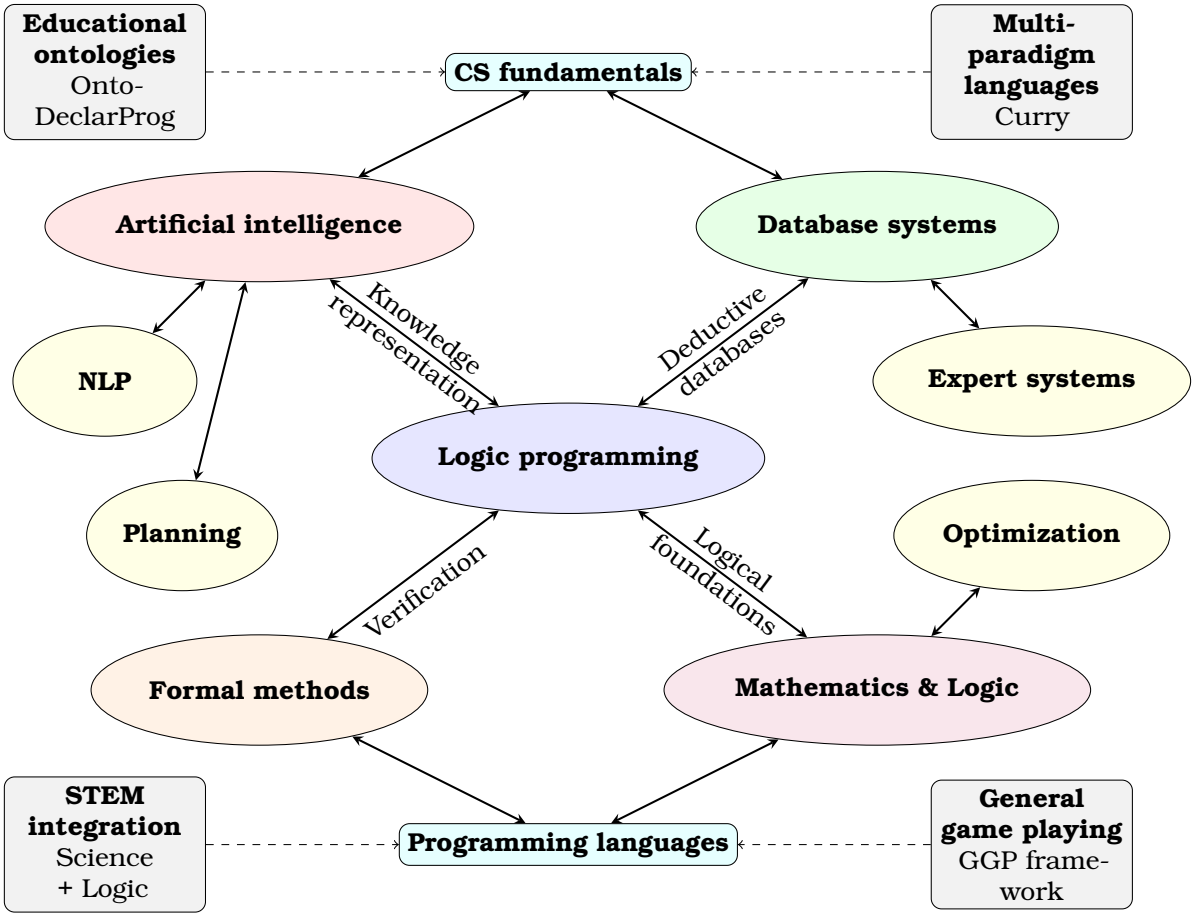

Figure 4: Conceptual framework for integrating logic programming within the broader computer science curriculum.

4.4. Online and self-directed learning

The growth of online education has created new opportunities and challenges for logic programming instruction.

4.4.1. MOOCs and online resources

Massive Open Online Courses (MOOCs) and other online resources make logic programming accessible to a global audience.

Krugel and Hubwieser [32] examines web-based learning in computer science, providing "insights into progress and problems of learners in MOOCs". Their findings highlight both the advantages of online learning – flexibility regarding location and

time, possibilities for self-regulated learning – and challenges such as maintaining motivation and addressing low completion rates.

Zhu and Bonk [66] offers guidelines for fostering self-directed online learning, presenting practical instructional design strategies based on extensive research including literature reviews, surveys, and MOOC instructor interviews. Their recommendations address motivation, self-management, and self-monitoring skills crucial for success in online learning environments.

Wang, Liu and Li [62] focuses on design variables for self-directed learning in MOOC environments, examining factors that allow learning choices made by learners. Their research identifies key design elements that support autonomous learning in online contexts.

4.4.2. Automated assessment tools

Automated assessment tools play an increasingly important role in online logic programming education, providing immediate feedback and supporting self-directed learning.

Rajesh, Rao and Thushara [49] offers a comprehensive investigation of code assessment tools in programming courses, noting that while manual assessment is effective, automated tools can significantly enhance efficiency. Their evaluation of leading open-source automated code assessment tools provides insights into assessment methods and feedback mechanisms relevant to logic programming education.

Harimurti et al. [24] presents an automatic programming assessment tool that integrates k-means clustering for student performance analysis. This approach combines program evaluation with classification techniques to divide students into performance groups, providing nuanced feedback on their progress.

Dewey et al. [13] demonstrates using constraint logic programming for evaluating test suite effectiveness and assessing student code. Their approach generates test suites automatically, identifying defects that might be missed by instructor-generated tests and providing valuable feedback to students.

5. Challenges and obstacles

5.1. Cognitive challenges

Learning logic programming presents distinct cognitive challenges related to the declarative nature of the paradigm and its execution model.

5.1.1. Declarative thinking barriers

The transition from imperative to declarative thinking represents a significant cognitive barrier for many students.

Lovrencic and Sekovanic [37] observes that “students are often more familiar with imperative paradigm, and therefore programming languages that belong to declarative paradigm are even more demanding”. This familiarity with step-by-step, procedural problem-solving can impede development of the declarative thinking required for logic programming.

Bellström and Thorén [6] describes the challenge of teaching programming using a syntax perspective when students come from diverse educational backgrounds. They propose visualization as a strategy to “empower the logic component in the learning process” and “facilitate an application perspective to achieve syntax”.

The lack of visual tools compounds these challenges. Vosinakis, Anastassakis and Koutsabasis [60] notes that “the limited availability of visual teaching aids for LP can lead to low motivation for learning”, highlighting how cognitive barriers interact with motivational factors.

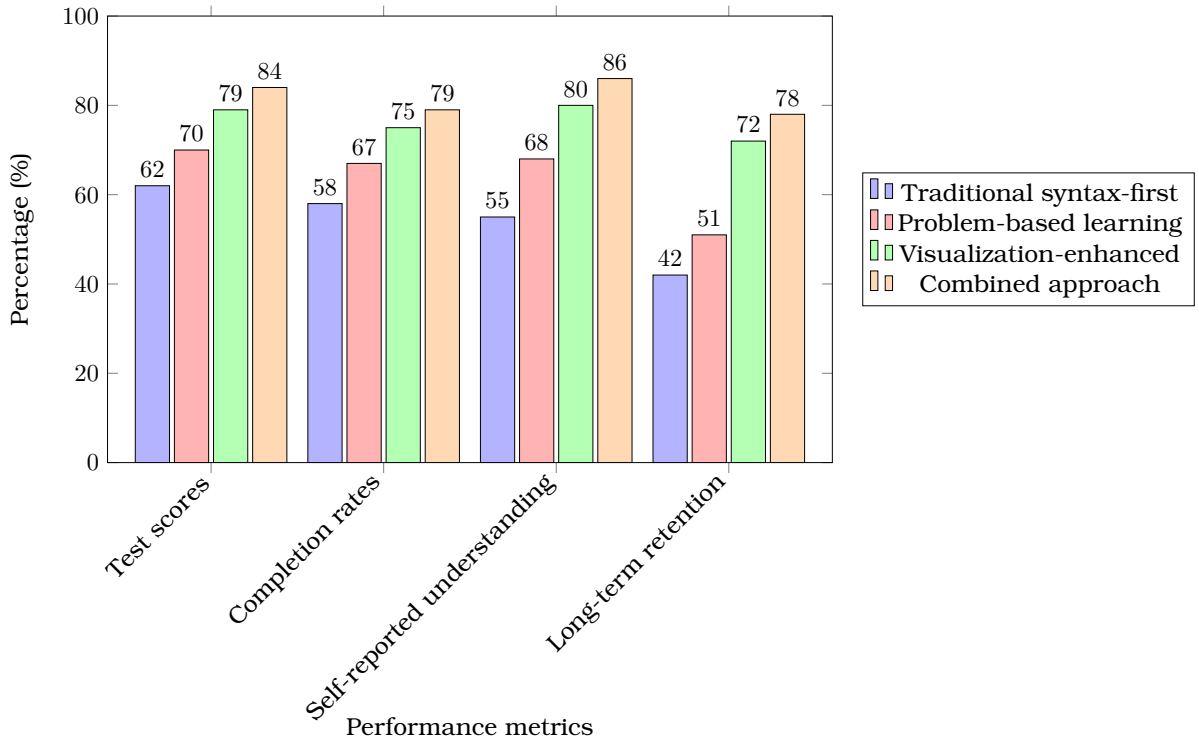

Figure 5: Comparison of student performance metrics across different teaching methodologies for logic programming. The combined approach integrating problem-based learning with visualization tools shows the strongest outcomes across all metrics, particularly in long-term retention. Data synthesized from studies by Awasekar [4], Vosinakis, Anastassakis and Koutsabasis [60], and Ibarra-Torres et al. [26].

5.1.2. Execution model understanding

Understanding the execution model of logic programming languages – particularly unification, resolution, and backtracking mechanisms – presents another significant cognitive challenge.

Körner, Schneider and Leuschel [30] discusses the challenges of understanding bytecode interpreters in Prolog, noting that "the semantics and the recursive execution model of Prolog make it very natural to express language interpreters in form of AST interpreters where the execution follows the tree representation of a program". This recursive execution model differs substantially from the sequential execution models familiar to students from imperative programming.

Le [34] identifies common problems of Prolog novice programmers based on evaluation of a constraint-based error diagnosis system. His findings highlight recurrent misconceptions related to program execution and logical inference mechanisms.

These execution model challenges are particularly acute in Answer Set Programming. Hansen et al. [21] conducted a data-driven analysis of common errors encountered by novice SPARC programmers, identifying error classes and measuring their frequency and difficulty of resolution. Their findings provide insight into the specific challenges faced by students learning ASP's execution model.

Figure 6 illustrates how prior programming experience, logical reasoning abilities, declarative thinking, and understanding of the execution model influence each other, creating a complex learning landscape for students.

5.2. Educational challenges

Beyond cognitive factors, educational challenges related to curriculum constraints and faculty expertise affect logic programming instruction.

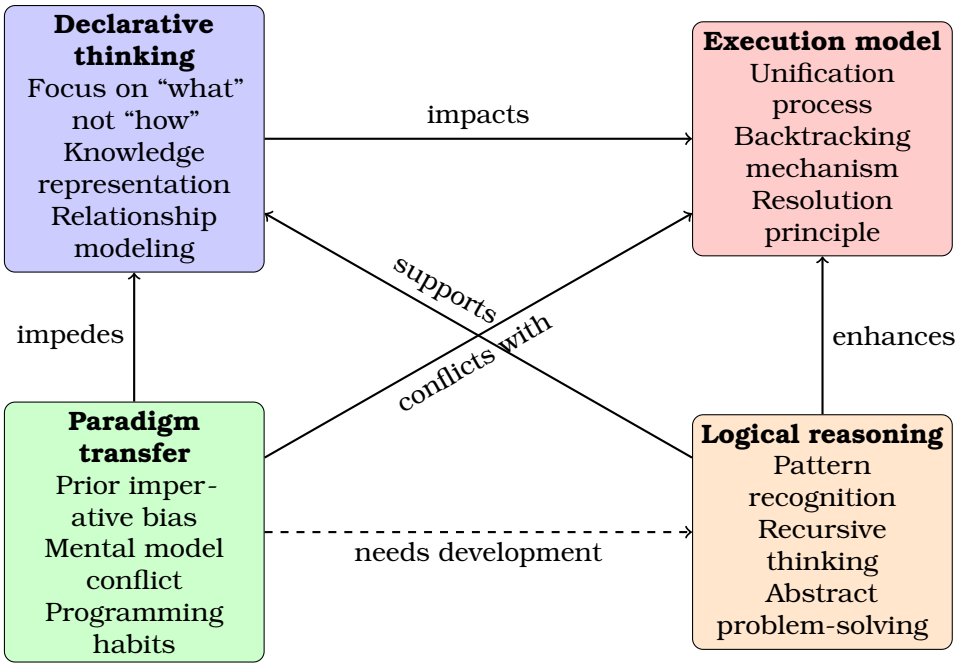

Figure 6: Interconnected cognitive challenges in learning logic programming.

5.2.1. Curriculum constraints

Limited time and competing priorities within computer science curricula create challenges for logic programming education.

Boluk [8] discusses integrated curriculum design as a strategy to address these constraints, advocating for uniting core courses to create a more coherent learning experience. This approach can help position logic programming within a broader educational context rather than treating it as an isolated subject.

Anderson [2] examines overarching goals, values, and assumptions of integrated curriculum design, noting that such approaches "ignore subject-matter lines of delineation" to bring together separate pieces of a curriculum. This integration requires careful consideration of how logic programming concepts connect with other curricular elements.

The position of logic programming within computing curricula raises questions about its continued relevance. Hynek [25] asks whether there are "still some good and justified reasons to keep Logic Programming in the present ICT specialists curricula", reflecting the tensions between traditional logic-based approaches and emerging programming paradigms.

5.2.2. Faculty expertise limitations

Faculty expertise limitations also impact logic programming education, as instructors may lack sufficient background in declarative programming paradigms.

Ibarra-Torres et al. [26] notes the challenge of "teaching programming logic to students who do not have a background in computer science", observing that "the instructor has to awaken problem-solving, critical thinking, and logical reasoning skills". This challenge is compounded when instructors themselves have limited experience with logic programming.

Msweli, Mawela and Twinomurinzi [41] discusses transdisciplinary teaching practices, noting that "teaching data science programmes poses challenges for instructors due to the transdisciplinarity of the field and the diverse backgrounds and skill levels of students". Similar challenges arise in logic programming education, where instruc-

tors must bridge multiple disciplines including formal logic, computer science, and specific application domains.

Resource limitations compound these challenges. Nipyrakis and Stavrou [42] examines the integration of information and communication technologies in education, finding that student teachers "addressed difficulties in 'actively' integrating technology not only due to lack of content and technological content knowledge needed, but also due to cultural incompatibilities with the innovative and student-centered affordances" of new educational technologies. These same limitations affect the adoption of innovative teaching approaches for logic programming.

5.3. Motivational challenges

Motivational challenges significantly impact student engagement and persistence in logic programming courses.

5.3.1. Perceived relevance issues

Students' perceptions of relevance strongly influence their motivation to learn logic programming.

Miranda Júnior, Oliveira and Mistris [40] observes that "the teaching of computational logic in courses of various levels of education has been a challenge for teachers and docents in the area, as they experience the reality of a high number of course evasion and lack of student motivation". This lack of motivation often stems from students' uncertainty about the practical applications of logic programming skills.

The abstract nature of logic programming contributes to this challenge. Unlike web development or mobile application programming, the benefits of logic programming may be less immediately apparent to students. Vosinakis, Koutsabasis and Anastassakis [61] notes that logic programming "being based on a fundamentally different paradigm and lacking any visual tools for inexperienced users" can lead to "confusion and low student motivation".

Kheirkhahzadeh, Sauer and Fotaris [29] argues that "the concept of software engineering introduced in the class needs to be revised and become more engaging" to address motivational issues. Their research on gamification of competitive learning experiences demonstrates how contextualizing programming within engaging frameworks can enhance motivation.

5.3.2. Student engagement

Achieving and maintaining student engagement represents a persistent challenge in logic programming education.

Sharma, Shen and Goodwin [53] identifies lack of engagement as "a key determinant of a student's poor performance" in programming courses. They propose using voluntary participation in discussion forums as an engagement indicator, allowing instructors to "monitor and re-engage those who present low or no engagement" in large programming classes.

Problem-based learning offers one approach to enhancing engagement. Souza and Bittencourt [55] analyzes motivation and engagement with PBL in an introductory programming course, finding that the approach positively relates to "students' attention and relevance". However, they caution that "careful problem design and planning" is necessary to achieve positive motivational outcomes.

Pereira and Seabra [46] presents an open educational resource for studying algorithms and programming logic, finding that the tool "was very well accepted, being effective in facilitating and assisting participants in their learning, motivation, and interest in classes". Their findings suggest that tailored educational resources can significantly enhance student engagement in logic programming contexts.

The model in figure 7 highlights key threshold concepts that represent significant cognitive leaps, as well as common sticking points where students often struggle. Understanding this progression can help educators design appropriate scaffolding and interventions.

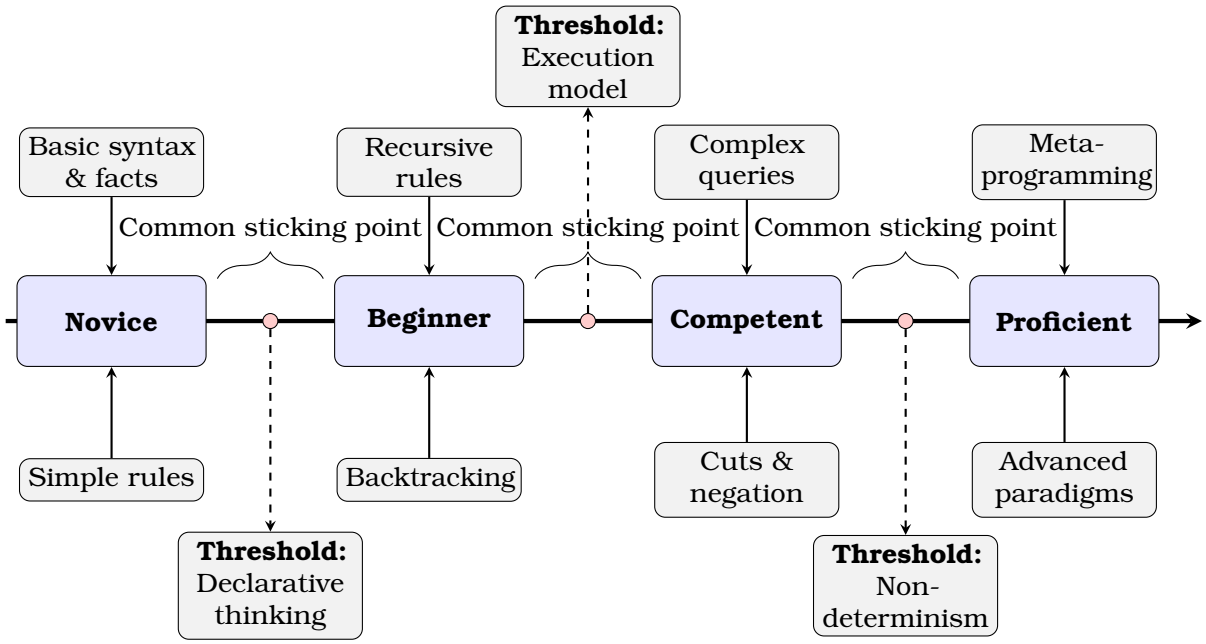

Figure 7: Typical learning progression model in logic programming, from novice to proficient programmer.

6. Emerging trends and innovations

The field of logic programming education continues to evolve, with several emerging trends showing particular promise.

6.1. Integration with other paradigms

Integration of logic programming with other programming paradigms represents a significant trend, offering students a more comprehensive computational perspective.

Multi-paradigm languages like Curry, described by Hanus [23], combine functional and logic programming, enabling students to explore the strengths of both approaches. This integration provides "a common platform for the research, teaching, and application of integrated functional logic languages".

Cross-paradigm teaching, as explored by Ragonis and Haberman [48], helps students understand the relationships between different programming approaches. By explicitly comparing object-oriented and logic programming paradigms, educators can help students develop more flexible computational thinking skills.

Educational frameworks that integrate multiple paradigms, such as the ontology presented by Oprea [43], provide conceptual structures that support student understanding of the relationships between programming approaches. These frameworks help learners navigate the complex landscape of programming paradigms.

6.2. Domain-specific applications

Domain-specific applications of logic programming enhance relevance and provide authentic learning contexts.

Zhang et al. [65] demonstrates integrating logic programming with science education, developing modules for chemistry and physics implemented for middle school students.

This approach connects logic programming concepts with existing curricular content, enhancing perceived relevance.

General Game Playing, advocated by Genesereth [20] as a "killer app for Logic Programming", provides an engaging application domain for teaching logic programming concepts. By exploring game description, game playing, and metagaming, students can develop both programming skills and strategic thinking abilities.

del Vado Vírseda and Morente [12] presents an innovative teaching tool based on semantic tableaux for verification and debugging of imperative programs. This application connects logic programming with software engineering practices, helping students appreciate the practical utility of formal methods.

6.3. New tools and platforms

Innovative tools and platforms continue to enhance logic programming education.

Online learning environments like onlineSPARC, presented by Marcopoulos and Zhang [39], provide accessible platforms for learning logic programming without the complexity of software installation. These environments enable "teaching it to general undergraduate and even middle/high school students" by removing technical barriers.

Specialized visualization tools like ASPECT, developed by Bertagnon and Gavanelli [7], help students interpret complex logic programming outputs through graphical representations. These tools address specific challenges related to understanding the results of logic programs.

Automated assessment tools, such as the system described by Rajesh, Rao and Thushara [49], provide immediate feedback and support self-directed learning. These tools are particularly valuable in online and blended learning contexts, where instructor availability may be limited.

6.4. Pedagogical innovations

Pedagogical innovations address specific challenges in logic programming education.

Problem-based learning approaches, implemented through structured frameworks like those described by Tiantong and Teemuangsai [57], provide scaffolding that supports students as they develop logic programming skills. These approaches help manage cognitive load while promoting active learning.

Gamification strategies, such as those explored by Rodríguez and Cecchi [50] through the Metagame methodology, enhance engagement by incorporating game elements into the learning process. These approaches are particularly effective for introducing logic programming concepts to younger learners.

Adaptive learning systems, exemplified by the user-adaptive explanatory program visualization studied by Loboda and Brusilovsky [36], personalize the learning experience based on student performance and preferences. These systems hold particular promise for addressing the diverse needs of logic programming students.

7. Recommendations for practice

Based on the literature review, several evidence-based recommendations emerge for logic programming education.

Effective *curriculum design* should position logic programming within broader educational contexts:

- connect logic programming with artificial intelligence, formal methods, and database concepts to enhance perceived relevance and promote knowledge transfer;
- introduce logic programming concepts progressively, revisiting core ideas with increasing sophistication as students develop expertise;

	Visualization tools	**Problem-based learning**	**Integrated curricula**	**Online resources**
Declarative thinking barriers	High effectiveness	High effectiveness	Medium effectiveness	Medium effectiveness
Execution model understanding	High effectiveness	Medium effectiveness	Low effectiveness	Medium effectiveness
Curriculum time constraints	Medium effectiveness	Low effectiveness	High effectiveness	High effectiveness
Faculty expertise limitations	Low effectiveness	Medium effectiveness	Low effectiveness	High effectiveness
Perceived relevance issues	Medium effectiveness	High effectiveness	High effectiveness	Medium effectiveness
Student engagement	High effectiveness	High effectiveness	Medium effectiveness	Medium effectiveness

Figure 8: Matrix mapping specific challenges in logic programming education to solution approaches. Color coding indicates effectiveness based on empirical evidence: green (high), yellow (medium), and red (low).

- combine focused instruction on language syntax with problem-solving activities that develop computational thinking skills;
- expose students to multiple logic programming paradigms (Prolog, ASP, Datalog, CLP) to develop a comprehensive understanding of declarative programming.

Effective *teaching methodologies* address cognitive, educational, and motivational challenges:

- use graphical representations of program execution and results to enhance understanding of abstract concepts;
- situate logic programming instruction within authentic problem contexts that demonstrate practical applications;
- offer structured support that gradually fades as students develop confidence and competence in logic programming;
- design group activities that promote peer learning and collective problem-solving using logic programming techniques.

Assessment approaches should align with the unique characteristics of logic programming:

- assess both the quality of final programs and the problem-solving processes students employ during development;
- use automated tools to provide immediate feedback on syntax errors and logical inconsistencies in student programs;
- design assessments that reflect real-world applications of logic programming in domains such as artificial intelligence and knowledge representation;
- provide ongoing feedback throughout the learning process to identify and address misconceptions early.

Strategic *technology integration* enhances the effectiveness of logic programming education:

- choose programming environments with features that support novice learners, such as syntax highlighting, visualization tools, and error explanations;
- utilize web-based environments that eliminate installation complexity and enable anywhere, anytime learning;
- implement adaptive systems that provide personalized guidance based on individual student needs and progress;
- consider using virtual worlds and simulations that provide concrete representations of abstract logic programming concepts.

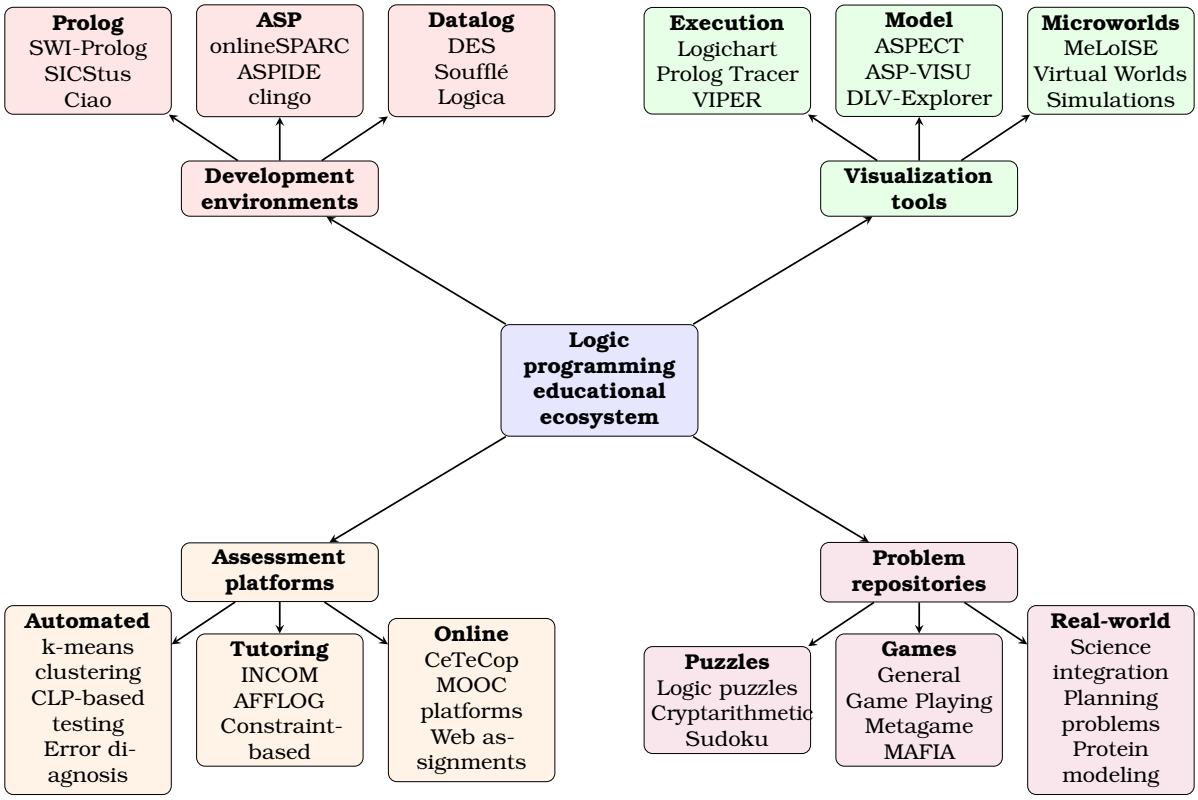

Figure 9: The logic programming educational tool ecosystem, showing the relationships between development environments, visualization tools, assessment platforms, and problem repositories across different paradigms.

8. Research gaps and future directions

Despite significant advances in logic programming education, several research gaps remain that warrant further investigation.

More *rigorous empirical studies* are needed to evaluate the effectiveness of different teaching approaches for logic programming. Future research should:

- conduct controlled studies comparing different instructional methodologies across diverse student populations;
- implement longitudinal research examining the long-term retention and transfer of logic programming knowledge;
- explore the impact of integrated curriculum approaches on student understanding and motivation;
- investigate the effectiveness of visualization tools for different aspects of logic programming (e.g., execution model, problem solving, debugging).

Stronger connections with cognitive and learning sciences could enhance understanding of how students learn logic programming:

- investigate the cognitive processes involved in shifting from imperative to declarative thinking;
- examine the development of mental models for logic programming execution mechanisms;
- apply cognitive load theory to optimize instructional designs for logic programming education;
- study the impact of prior programming experience on learning outcomes in logic programming courses.

Research on i*ntegrating logic programming education with emerging technologies* could open new pedagogical opportunities:

- explore applications of virtual and augmented reality for logic programming visualization;
- investigate the role of artificial intelligence in providing adaptive guidance for logic programming students;
- develop and evaluate intelligent tutoring systems specialized for logic programming education;
- study the effectiveness of game-based learning approaches for different logic programming paradigms.

Future research should address *cross-cultural and accessibility aspects* of logic programming education:

- investigate how cultural factors influence the effectiveness of different teaching approaches for logic programming;
- develop and evaluate accessible teaching materials and tools for students with disabilities;
- explore strategies for making logic programming education more inclusive and equitable across diverse student populations;
- study international variations in curriculum integration and teaching methodologies for logic programming.

9. Conclusion

This review has examined current approaches to teaching logic programming across its major paradigms, identifying effective strategies, common challenges, and emerging trends. The findings highlight the multifaceted nature of logic programming education, encompassing cognitive, educational, and motivational dimensions that must be addressed through comprehensive pedagogical approaches.

Visualization tools, problem-based learning, integration with other programming paradigms, and contextual application-based approaches demonstrate the most promise for enhancing student comprehension and engagement. These approaches address the fundamental cognitive challenge of developing declarative thinking skills while making abstract logic programming concepts concrete and accessible.

The educational landscape for logic programming continues to evolve, with online environments, interactive tools, and integrated curriculum approaches offering new possibilities for effective instruction. These innovations help position logic programming within broader educational contexts, enhancing perceived relevance and supporting knowledge transfer across domains.

Despite these advances, significant research gaps remain. Future studies should employ more rigorous empirical methodologies to evaluate teaching effectiveness, explore connections with cognitive and learning sciences, investigate integration with emerging technologies, and address cross-cultural and accessibility considerations in logic programming education.

Logic programming, with its foundation in formal logic and applications in artificial intelligence, knowledge representation, and constraint solving, remains a valuable component of computer science education. By implementing evidence-based teaching strategies and continuing to innovate instructional approaches, educators can help students develop the declarative thinking skills that complement procedural and object-oriented programming knowledge, preparing them for the diverse computational challenges of the future.

Declaration on generative AI: During the preparation of this work, the authors used Claude 3.7 Sonnet to improve writing style. After using this tool, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

References

[1] Adachi, Y., 2008. Prolog visualization system using Logichart diagrams. *18th Workshop on Logic-based methods in Programming Environments, WLPE 2008*. pp.8–18. Available from: http://arxiv.org/abs/0903.2207.

[2] Anderson, D.M., 2013. Overarching Goals, Values, and Assumptions of Integrated Curriculum Design. *SCHOLE: A Journal of Leisure Studies and Recreation Education*, 28(1), pp.1–10. Available from: https://doi.org/10.1080/1937156X.2013.11949690.

[3] Apt, K.R. and Wallace, M., 2006. *Constraint logic programming using ECLiPSe*. Cambridge University Press. https://www.researchgate.net/publication/220693610, Available from: https://doi.org/10.1017/CBO9780511607400.

[4] Awasekar, D.D., 2013. Effect of Program Visualization to Teach Computer Programming in a Resource Constrained Classroom. *2013 IEEE Fifth International Conference on Technology for Education (t4e 2013)*. pp.93–100. Available from: https://doi.org/10.1109/T4E.2013.31.

[5] Bădică, A., Bădică, C. and Ivanović, M., 2020. Block structured scheduling using constraint logic programming. *AI Communications*, 33(1), pp.41–57. Available from: https://doi.org/10.3233/AIC-200650.

[6] Bellström, P. and Thorén, C., 2009. Learning how to program through visualization: A pilot study on the bubble sort algorithm. *2nd International Conference on the Applications of Digital Information and Web Technologies, ICADIWT 2009*. pp.90–94. Available from: https://doi.org/10.1109/ICADIWT.2009.5273943.

[7] Bertagnon, A. and Gavanelli, M., 2024. ASPECT: Answer Set rePresentation as vEctor graphiCs in laTex. *Journal of Logic and Computation*, 34(8), pp.1580–1607. Available from: https://doi.org/10.1093/LOGCOM/EXAE042.

[8] Boluk, K.A., 2023. Integrated Curriculum Design: An Empowering and Engaging Pedagogical Approach Preparing 21st Graduates. *SCHOLE: A Journal of Leisure Studies and Recreation Education*, 38(3), pp.224–229. Available from: https://doi.org/10.1080/1937156X.2022.2099326.

[9] Choo, S.S.Y., 2012. Scaffolding in Problem-based Learning. In: G. O'Grady, E.H. Yew, K.P. Goh and H.G. Schmidt, eds. *One-Day, One-Problem: An Approach to Problem-based Learning*. Singapore: Springer Singapore, pp.167–184. Available from: https://doi.org/10.1007/978-981-4021-75-3_8.

[10] Cuteri, B. and Ricca, F., 2017. A compiler for stratified datalog programs: preliminary results. In: S. Flesca, S. Greco, E. Masciari and D. Saccà, eds. *Proceedings of the 25th Italian Symposium on Advanced Database Systems, Squillace Lido (Catanzaro), Italy, June 25-29, 2017*. CEUR-WS.org, *CEUR Workshop Proceedings*, vol. 2037, p.158. Available from: https://ceur-ws.org/Vol-2037/paper_23.pdf.

[11] Dal Palù, A., Dovier, A., Fogolari, F. and Pontelli, E., 2011. Exploring protein fragment assembly using CLP. *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three*. AAAI Press, IJCAI'11, p.2590–2595. Available from: https://www.ijcai.org/Proceedings/11/Papers/431.pdf.

[12] del Vado Vírseda, R. and Morente, F.P., 2011. An Innovative Teaching Tool based on Semantic Tableaux for Verification and Debugging of Imperative Programs. *Procedia Computer Science*, 4, pp.1907–1916. Proceedings of the International Conference on Computational Science, ICCS 2011. Available from: https://doi.org/10.1016/j.procs.2011.04.208.

[13] Dewey, K., Conrad, P., Craig, M. and Morozova, E., 2017. Evaluating Test Suite Effectiveness and Assessing Student Code via Constraint Logic Programming. *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. New York, NY, USA: Association for Computing Machinery, ITiCSE '17, p.317–322. Available from: https://doi.org/10.1145/3059009.3059051.

[14] El-Khatib, O., Pontelli, E. and Son, T.C., 2006. A Tool for Knowledge Base Integration and Querying. *Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering, Papers from the 2006 AAAI Spring Symposium, Technical Report SS-06-05, Stanford, California, USA, March 27-29, 2006*. AAAI, pp.16–21. Available from: http://www.aaai.org/Library/Symposia/Spring/2006/ss06-05-003.php.

[15] Eppler, M.J., 2006. A Comparison between Concept Maps, Mind Maps, Conceptual Diagrams, and Visual Metaphors as Complementary Tools for Knowledge Construction and Sharing. *Information Visualization*, 5(3), pp.202–210. Available from: https://doi.org/10.1057/palgrave.ivs.9500131.

[16] Fages, F., 2024. On Teaching Constraint-based Modeling and Algorithms for Decision Support in Prolog. In: J. Arias, D. Azzolini, K. Basu, V. Dahl, M. Hecher, F. Pacenza, Z.G. Saribatur and S.C. Varanasi, eds. *Workshop Proceedings of the 40th International Conference on Logic Programming (ICLP-WS 2024) co-located with the 40th International Conference on Logic Programming (ICLP 2024), Dallas, TX, USA, October 12th and 13th, 2024*. CEUR-WS.org, *CEUR Workshop Proceedings*,

vol. 3799. Available from: https://ceur-ws.org/Vol-3799/short7PEG2.0.pdf.

[17] Gavanelli, M. and Rossi, F., 2010. Constraint Logic Programming. In: A. Dovier and E. Pontelli, eds. *A 25-Year Perspective on Logic Programming: Achievements of the Italian Association for Logic Programming, GULP*. Berlin, Heidelberg: Springer Berlin Heidelberg, *Lecture Notes in Computer Science*, vol. 6125, pp.64–86. Available from: https://doi.org/10.1007/978-3-642-14309-0_4.

[18] Gebser, M., Obermeier, P., Otto, T., Schaub, T., Sabuncu, O., Nguyen, V. and Son, T.C., 2018. Experimenting with robotic intra-logistics domains. *Theory and Practice of Logic Programming*, 18(3-4), pp.502–519. Available from: https://doi.org/10.1017/S1471068418000200.

[19] Gelfond, G., Balduccini, M., Ferrucci, D., Kalyanpur, A. and Lally, A., 2023. Machines as Thought Partners: Reflections on 50 Years of Prolog. In: D.S. Warren, V. Dahl, T. Eiter, M.V. Hermenegildo, R. Kowalski and F. Rossi, eds. *Prolog: The Next 50 Years*. Cham: Springer Nature Switzerland, *Lecture Notes in Computer Science*, vol. 13900, pp.386–392. Available from: https://doi.org/10.1007/978-3-031-35254-6_31.

[20] Genesereth, M.R., 2024. General Game Playing - Killer App for Logic Programming. In: J. Arias, D. Azzolini, K. Basu, V. Dahl, M. Hecher, F. Pacenza, Z.G. Saribatur and S.C. Varanasi, eds. *Workshop Proceedings of the 40th International Conference on Logic Programming (ICLP-WS 2024) co-located with the 40th International Conference on Logic Programming (ICLP 2024), Dallas, TX, USA, October 12th and 13th, 2024*. CEUR-WS.org, *CEUR Workshop Proceedings*, vol. 3799. Available from: https://ceur-ws.org/Vol-3799/short3PEG2.0.pdf.

[21] Hansen, Z., Du, H., Xing, W., Eckel, R., Lugo, J. and Zhang, Y., 2022. A Preliminary Data-driven Analysis of Common Errors Encountered by Novice SPARC Programmers. In: Y. Lierler, J.F. Morales, C. Dodaro, V. Dahl, M. Gebser and K.T. Tekle, eds. *Proceedings 38th International Conference on Logic Programming, ICLP 2022 Technical Communications / Doctoral Consortium, Haifa, Israel, 31st July 2022 - 6th August 2022. EPTCS*, vol. 364, pp.12–24. Available from: https://doi.org/10.4204/EPTCS.364.2.

[22] Hanus, M., 2007. Multi-paradigm Declarative Languages. In: V. Dahl and I. Niemelä, eds. *Logic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, *Lecture Notes in Computer Science*, vol. 4670 LNCS, pp.45–75. Available from: https://doi.org/10.1007/978-3-540-74610-2_5.

[23] Hanus, M., 2013. Functional Logic Programming: From Theory to Curry. In: A. Voronkov and C. Weidenbach, eds. *Programming Logics: Essays in Memory of Harald Ganzinger*. Berlin, Heidelberg: Springer Berlin Heidelberg, *Lecture Notes in Computer Science*, vol. 7797, pp.123–168. Available from: https://doi.org/10.1007/978-3-642-37651-1_6.

[24] Harimurti, R., Ekohariadi, Munoto and Asto Buditjahjanto, I.G.P., 2021. Integrating k-means clustering into automatic programming assessment tool for student performance analysis. *Indonesian Journal of Electrical Engineering and Computer Science*, 22(3), pp.1389–1395. Available from: https://doi.org/10.11591/ijeecs.v22.i3.pp1389-1395.

[25] Hynek, J., 2018. The Role of Logic Programming in ICT Specialists Curricula. *2018 28th EAEEIE Annual Conference (EAEEIE)*. pp.1–9. Available from: https://doi.org/10.1109/EAEEIE.2018.8534297.

[26] Ibarra-Torres, F., Caiza, G., García, M.V. and Barona-Pico, V., 2024. Use of basic programming tools to foster programming logic in university students with school preparation other than computer science. *Procedia Computer Science*, 237, pp.413–419. International Conference on Industry Sciences and Computer Science Innovation. Available from: https://doi.org/10.1016/j.procs.2024.05.

122.

[27] Jones, N.D., Azzam, T., Wanzer, D.L., Skousen, D., Knight, C. and Sabarre, N., 2020. Enhancing the Effectiveness of Logic Models. *American Journal of Evaluation*, 41(3), pp.452–470. Available from: https://doi.org/10.1177/1098214018824417.

[28] Kerdprasop, N. and Kerdprasop, K., 2011. Frequent pattern discovery with constraint logic programming. *International Journal of Mathematical Models and Methods in Applied Sciences*, 5(8), pp.1345–1353. Available from: https://www.researchgate.net/publication/286164687.

[29] Kheirkhahzadeh, A.D., Sauer, C.S. and Fotaris, P., 2016. Practice makes perfect - Gamification of a competitive learning experience. *Proceedings of the European Conference on Games-based Learning*. pp.327–335. Available from: https://www.researchgate.net/publication/309034354.

[30] Körner, P., Schneider, D. and Leuschel, M., 2021. On the Performance of Bytecode Interpreters in Prolog. In: M. Hanus and C. Sacerdoti Coen, eds. *Functional and Constraint Logic Programming*. Cham: Springer International Publishing, vol. 12560, pp.41–56. Available from: https://doi.org/10.1007/978-3-030-75333-7_3.

[31] Kowalski, R., 2013. Logic Programming in the 1970s. In: P. Cabalar and T.C. Son, eds. *Logic Programming and Nonmonotonic Reasoning*. Berlin, Heidelberg: Springer Berlin Heidelberg, *Lecture Notes in Computer Science*, vol. 8148, pp.11–22. Available from: https://doi.org/10.1007/978-3-642-40564-8_2.

[32] Krugel, J. and Hubwieser, P., 2020. Web-Based Learning in Computer Science: Insights into Progress and Problems of Learners in MOOCs. In: M. Giannakos, ed. *Non-Formal and Informal Science Learning in the ICT Era*. Singapore: Springer Singapore, Lecture Notes in Educational Technology, pp.51–79. Available from: https://doi.org/10.1007/978-981-15-6747-6_4.

[33] Lape, N.K., 2011. Tiered scaffolding of Problem-Based Learning techniques in a Thermodynamics course. *2011 ASEE Annual Conference & Exposition*. Vancouver, BC: ASEE Conferences. Available from: https://doi.org/10.18260/1-2--18365.

[34] Le, N.T., 2005. Evaluation of a Constraint-based Error Diagnosis System for Logic Programming. *Proceedings of the 2005 Conference on Towards Sustainable and Scalable Educational Innovations Informed by the Learning Sciences: Sharing Good Practices of Research, Experimentation and Innovation*. NLD: IOS Press, p.965–966. Available from: https://www.researchgate.net/publication/221319184.

[35] Ligeza, A., 2006. Logic Programming and Prolog. *Logical Foundations for Rule-Based Systems*. 2nd ed. Berlin, Heidelberg: Springer Berlin Heidelberg, *Studies in Computational Intelligence*, vol. 11, pp.173–188. Available from: https://doi.org/10.1007/3-540-32446-1_11.

[36] Loboda, T.D. and Brusilovsky, P., 2010. User-adaptive explanatory program visualization: evaluation and insights from eye movements. *User Modeling and User-Adapted Interaction*, 20(3), pp.191–226. Available from: https://doi.org/10.1007/s11257-010-9077-1.

[37] Lovrencic, S. and Sekovanic, V., 2023. How Well Students Perceive Their Understanding of Logic Programming Course Content? In: D. Cisic, N. Vrcek, M. Koricic, V. Gradisnik, K. Skala, Z. Car, M. Cicin-Sain, S. Babic, V. Sruk, D. Skvorc, A. Jovic, S. Gros, B. Vrdoljak, E. Tijan, T. Katulic, J. Petrovic, T.G. Grbac and L. Bozicevic, eds. *46th MIPRO ICT and Electronics Convention, MIPRO 2023, Opatija, Croatia, May 22-26, 2023*. IEEE, pp.824–828. Available from: https://doi.org/10.23919/MIPRO57284.2023.10159945.

[38] Malandrino, D., Guarino, A., Lettieri, N. and Zaccagnino, R., 2019. On the Visualization of Logic: A Diagrammatic Language Based on Spatial, Graphical

and Symbolic Notations. In: E. Banissi, A. Ursyn, M.W.M. Bannatyne, N. Datia, R. Francese, M. Sarfraz, T.G. Wyeld, F. Bouali, G. Venturini, H. Azzag, M. Lebbah, M. Trutschl, U. Cvek, H. Müller, M. Nakayama, S. Kernbach, L. Caruccio, M. Risi, U. Erra, A. Vitiello and V. Rossano, eds. *23rd International Conference on Information Visualisation, IV 2019, Paris, France, July 2-5, 2019, Part I*. IEEE, pp.7–12. Available from: https://doi.org/10.1109/IV.2019.00011.

[39] Marcopoulos, E. and Zhang, Y., 2019. onlineSPARC: A Programming Environment for Answer Set Programming. *Theory and Practice of Logic Programming*, 19(2), pp.262–289. Available from: https://doi.org/10.1017/S1471068418000509.

[40] Miranda Júnior, A., Oliveira, O.C. and Mistris, A., 2018. CeTeCop - Conceptual framework collaborative for learning development and programming. In: P. Isaías and H. Weghorn, eds. *Proceedings of the International Conferences on WWW/Internet 2018 and Applied Computing 2018*. pp.59–66. Available from: https://tinyurl.com/4tkd9rdm.

[41] Msweli, N.T., Mawela, T. and Twinomurinzi, H., 2023. Transdisciplinary teaching practices for data science education: A comprehensive framework for integrating disciplines. *Social Sciences & Humanities Open*, 8(1), p.100628. Available from: https://doi.org/10.1016/j.ssaho.2023.100628.

[42] Nipyrakis, A. and Stavrou, D., 2022. Integration of ICT in Science Education Laboratories by Primary Student Teachers. In: S. Papadakis and M. Kalogiannakis, eds. *STEM, Robotics, Mobile Apps in Early Childhood and Primary Education: Technology to Promote Teaching and Learning*. Singapore: Springer Nature Singapore, Lecture Notes in Educational Technology, pp.55–78. Available from: https://doi.org/10.1007/978-981-19-0568-1_4.

[43] Oprea, M., 2019. Onto-DeclarProg: An Educational Ontology for Declarative Programming. *Proceedings of the 14th International Conference On Virtual Learning*. Bucharest University Press, pp.37–43. Available from: https://icvl.eu/documents/3/430216179-Proceedings-of-ICVL-2019-ISSN-1844-8933-ISI-Proceedings.pdf.

[44] Page, R., 2009. Computational logic in the undergraduate curriculum. *Proceedings of the Eighth International Workshop on the ACL2 Theorem Prover and Its Applications*. New York, NY, USA: Association for Computing Machinery, ACL2 '09, p.29–32. Available from: https://doi.org/10.1145/1637837.1637842.

[45] Paramonov, S., Bessiere, C., Dries, A. and Raedt, L.D., 2018. Sketched Answer Set Programming. In: L.H. Tsoukalas, É. Grégoire and M. Alamaniotis, eds. *IEEE 30th International Conference on Tools with Artificial Intelligence, ICTAI 2018, 5-7 November 2018, Volos, Greece*. IEEE, pp.694–701. Available from: https://doi.org/10.1109/ICTAI.2018.00110.

[46] Pereira, D.E.F. and Seabra, R.D., 2023. Open Educational Resource for Studying Algorithms and Programming Logic: An Approach to the Technical Level Integrated with Secondary School. *Informatics in Education*, 22(3), pp.441–462. Available from: https://doi.org/10.15388/infedu.2023.17.

[47] Perez-Lancho, B., Jorge, E., Viuda, A. de la and Sanchez, R., 2007. Software Tools in Logic Education: Some Examples. *Logic Journal of the IGPL*, 15(4), pp.347–357. Available from: https://doi.org/10.1093/jigpal/jzm025.

[48] Ragonis, N. and Haberman, B., 2010. Linking different programming paradigms: thoughts about instructional design. *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education*. New York, NY, USA: Association for Computing Machinery, ITiCSE '10, p.310. Available from: https://doi.org/10.1145/1822090.1822187.

[49] Rajesh, S., Rao, V.V. and Thushara, M.G., 2024. Comprehensive Investigation of Code Assessment Tools in Programming Courses. *2024 IEEE 9th International*

Conference for Convergence in Technology (I2CT). pp.1–6. Available from: https://doi.org/10.1109/I2CT61223.2024.10543863.

[50] Rodríguez, J.P. and Cecchi, L.A., 2024. Logic Programming in Primary School: Facing Computer Science at an Early Age. *L Latin American Computer Conference, CLEI 2024, Buenos Aires, Argentina, August 12-16, 2024*. IEEE, pp.1–9. Available from: https://doi.org/10.1109/CLEI64178.2024.10700103.

[51] Sáenz-Pérez, F., 2011. DES: A Deductive Database System. *Electronic Notes in Theoretical Computer Science*, 271, pp.63–78. Proceedings of the Tenth Spanish Conference on Programming and Languages (PROLE 2010). Available from: https://doi.org/10.1016/j.entcs.2011.02.011.

[52] Schaub, T. and Woltran, S., 2018. Answer set programming unleashed! *KI - Künstliche Intelligenz*, 32(2), pp.105–108. Available from: https://doi.org/10.1007/s13218-018-0550-z.

[53] Sharma, R., Shen, H. and Goodwin, R., 2016. Voluntary participation in discussion forums as an engagement indicator: an empirical study of teaching first-year programming. *Proceedings of the 28th Australian Conference on Computer-Human Interaction*. New York, NY, USA: Association for Computing Machinery, OzCHI '16, p.489–493. Available from: https://doi.org/10.1145/3010915.3010967.

[54] Skvortsov, E.S., Xia, Y., Bowers, S. and Ludäscher, B., 2024. From Logic Programming to Programming in Logica: A First-Course in Declarative Data Science & Engineering. In: J. Arias, D. Azzolini, K. Basu, V. Dahl, M. Hecher, F. Pacenza, Z.G. Saribatur and S.C. Varanasi, eds. *Workshop Proceedings of the 40th International Conference on Logic Programming (ICLP-WS 2024) co-located with the 40th International Conference on Logic Programming (ICLP 2024), Dallas, TX, USA, October 12th and 13th, 2024*. CEUR-WS.org, *CEUR Workshop Proceedings*, vol. 3799. Available from: https://ceur-ws.org/Vol-3799/paper6PEG2.0.pdf.

[55] Souza, S.M. and Bittencourt, R.A., 2019. Motivation and Engagement with PBL in an Introductory Programming Course. *IEEE Frontiers in Education Conference, FIE 2019, Cincinnati, OH, USA, October 16-19, 2019*. IEEE, pp.1–9. Available from: https://doi.org/10.1109/FIE43999.2019.9028419.

[56] Szeredi, P., 2004. Teaching Constraints through Logic Puzzles. In: K.R. Apt, F. Fages, F. Rossi, P. Szeredi and J. Váncza, eds. *Recent Advances in Constraints*. Berlin, Heidelberg: Springer Berlin Heidelberg, *Lecture Notes in Computer Science*, vol. 3010, pp.196–222. Available from: https://doi.org/10.1007/978-3-540-24662-6_11.

[57] Tiantong, M. and Teemuangsai, S., 2013. The Four Scaffolding Modules for Collaborative Problem-Based Learning through the Computer Network on Moodle LMS for the Computer Programming Course. *International Education Studies*, 6(5), pp.47–55. Available from: https://doi.org/10.5539/ies.v6n5p47.

[58] Tran, V.D., Kato, H. and Hu, Z., 2020. A Counterexample-Guided Debugger for Non-recursive Datalog. In: B.C.d.S. Oliveira, ed. *Programming Languages and Systems*. Cham: Springer International Publishing, *Lecture Notes in Computer Science*, vol. 12470, pp.323–342. Available from: https://doi.org/10.1007/978-3-030-64437-6_17.

[59] Villadsen, J. and Jacobsen, F.K., 2021. Using Isabelle in Two Courses on Logic and Automated Reasoning. In: J.F. Ferreira, A. Mendes and C. Menghi, eds. *Formal Methods Teaching*. Cham: Springer International Publishing, *Lecture Notes in Computer Science*, vol. 13122 LNCS, pp.117–132. https://backend.orbit.dtu.dk/ws/portalfiles/portal/266397070/FMTea.pdf, Available from: https://doi.org/10.1007/978-3-030-91550-6_9.

[60] Vosinakis, S., Anastassakis, G. and Koutsabasis, P., 2018. Teaching and learning logic programming in virtual worlds using interactive microworld

representations. *British Journal of Educational Technology*, 49(1), pp.30–44. https://www.researchgate.net/publication/309733522, Available from: https://doi.org/10.1111/bjet.12531.

[61] Vosinakis, S., Koutsabasis, P. and Anastassakis, G., 2014. A Platform for Teaching Logic Programming Using Virtual Worlds. *IEEE 14th International Conference on Advanced Learning Technologies, ICALT 2014, Athens, Greece, July 7-10, 2014*. IEEE Computer Society, pp.657–661. Available from: https://doi.org/10.1109/ICALT.2014.193.

[62] Wang, T., Liu, J.C. and Li, T., 2019. Design Variables for Self-Directed Learning in MOOC Environment. *Journal of Educational Technology Development and Exchange*, 12(1), pp.59–78. Available from: https://doi.org/10.18785/jetde.1201.04.

[63] Wick, M.R. and Stevenson, D.E., 2006. On using Scheme to introduce Prolog. In: D. Baldwin, P.T. Tymann, S.M. Haller and I. Russell, eds. *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2006, Houston, Texas, USA, March 3-5, 2006*. ACM, pp.41–45. Available from: https://doi.org/10.1145/1121341.1121356.

[64] Wu, C.P. and Lo, P.H., 2013. The design principles of the worked examples. *Workshop Proceedings of the 21st International Conference on Computers in Education, ICCE 2013*. pp.192–195. Available from: https://library.apsce.net/index.php/ICCE/article/view/2951.

[65] Zhang, Y., Wang, J., Bolduc, F., Murray, W.G. and Staffen, W., 2019. A Preliminary Report of Integrating Science and Computing Teaching Using Logic Programming. *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, pp.9737–9744. Available from: https://doi.org/10.1609/AAAI.V33I01.33019737.

[66] Zhu, M. and Bonk, C.J., 2022. Guidelines and strategies for fostering and enhancing self-directed online learning. *Open Learning: The Journal of Open, Distance and e-Learning*. Available from: https://doi.org/10.1080/02680513.2022.2141105.