# Inference Engines Performance in Reasoning Tasks for Intelligent Tutoring Systems

Oleg A. Sychev, Anton Anikin(✉), and Mikhail Denisov

Volgograd State Technical University, Lenin Ave, 28, Volgograd 400005, Russia
anton.anikin@vstu.ru

**Abstract.** The use of formal knowledge representation models in intelligent tutoring systems often requires logical reasoning on these models by predefined rules. This process can be time and memory consuming, so finding effective software reasoners for different applications is an important research field. This problem is relevant for cognitive and constraint-based intelligent tutoring systems. We performed a comparative study of various software reasoners (Pellet, Apache Jena inference subsystem, Apache Jena SPARQL query processor, SWI-Prolog with semweb package, Closed World Machine, and Answer Set Programming solvers Clingo and DLV) for solving tasks specific to intelligent tutoring systems using three formal models with different properties and corresponding rule sets created for intelligent tutoring systems in introductory programming education domain. We compared features of rule-definition formalisms for different approaches and measured run and wall time, average CPU load, and peak RAM usage based on the count of inferred RDF triples. The experiments show that Apache Jena infers the solution quicker than other reasoners on the majority of tasks but consumes a significant amount of memory, while Clingo performs significantly better for combinatorial problems.

**Keywords:** Intelligent tutoring systems · Constraint-based ITS · Inference engines · Performance · Resource description framework · Answer set programming

## 1 Introduction and Related Work

E-learning is an application of computational science where technologies improve the learning process. It concerns developing software tools for students' automated assessment, students' self-assessment, and teacher's self-evaluation [18]. Using artificial intelligence is a promising approach in e-learning systems. Intelligent models and algorithms allow providing immediate feedback, based on the learning analytics, during online learning [13,16,17], checking the correctness of the learner answers [30], generating the learning tasks, and managing the learning process in an adaptive manner [11,14,22].

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12950, pp. 471–482, 2021.
https://doi.org/10.1007/978-3-030-86960-1_33

The intelligent tutoring systems (ITS) development implies the use of knowledge modeling for these systems. The use of formal knowledge-representation models in modern cognitive, rule-based, and constraint-based intelligent tutoring systems requires logical inference. The rules and facts for it can be implemented as OWL/RDF ontologies using OWL/SWRL rules, allowing using reasoners like Pellet and Jena [25]. So, this combination of models and reasoning mechanisms can be applied to different tasks in the ITS implementation. E.g., in [29] authors use a set of SWRL rules to find an order of evaluation of C-language expressions and capture fault reason for incorrect students' answers. A similar approach was applied to verify algorithm execution tracings [28] in ITS for programming learning. Other approaches include using Prolog language [31] and Answer Set Programming [9] (ASP) [10,21]. For example, authors of AFFLOG [15] use ASP for learning course planning according to task difficulty and student affection.

There are ontology benchmarks, such as the recent OWL2Bench [26]. These benchmarks, however, focus on reasoning within the semantics of the various OWL/OWL2 profiles without considering arbitrary, user-defined rules. Also, they usually do not investigate problems specific to intelligent tutoring systems.

Research investigating production rule engines seldom deal with RDF or OWL. The OpenRuleBench study [24] examines eight rule engines, only one of which (Jena) handles RDF/OWL. Again, this test examines the performance of three features specific to this type of reasoners, each involving several rules (using both real-world and synthetic datasets). The results of this study are outdated; since then, the approaches have evolved, and new tools such as Clingo have emerged.

In [25], the authors investigate three reasoners (including Jena) on six rulesets for forward-chaining reasoning tasks with data represented as RDF. The scope of the rules is real-world (products and recommendations for customers), and rulesets range in size from 1 to 24 rules. The subject of the study is close to our own and includes, in addition to performance, reasoning strategies, supported rule languages and their expressiveness, and the use of built-in functions.

This study aims to cover more reasoners dealing with RDF and also emerging problem solvers like Clingo, and test them on problems related to intelligent tutoring systems that are often more complex than regular semantic-web problems, having up to a hundred rules within a problem domain. So, the reasoning engine used in ITS should perform complex reasoning while grading students' answers in real time which can be a time- and memory-consuming task.

We considered seven inferences engines.

1. `Pellet` 2.3 [27] is an OWL reasoner supporting SWRL [6] built-ins.
2. Apache `Jena` [3] - RDF/OWL framework for Java including rule reasoner.
3. Apache Jena ARQ [1] - a `SPARQL` query processor capable running batches of SPARQL Update [5].
4. SWI-`Prolog` [31] with the semweb (rdf11) package [4] for handling RDF data alongside prolog native predicates representing the domain rules.
5. `CWM` (Closed World Machine) [8] - ancient reasoner for RDF written in Python 2 with its own rule syntax supporting some useful built-ins.

6. `Clingo` [20] - Answer Set Programming (ASP) tool that combines `gringo` grounder and `clasp` solver [19] (handles data as ASP facts, no native support for RDF).
7. `DLV` [2,7] - another ASP solver using a dialect of ASP slightly different from Clingo's one.

`CWM` (Closed World Machine) and `DLV` (an ASP solver) performed poorly during the first evaluation and were excluded from the following experiments. The extensive study included five software reasoners.

We aimed to compare reasoners in terms of their suitability for constraint-based intelligent tutoring systems. To do it, we used three sets of rules scaling the size of 5 datasets representing inference tasks common for intelligent tutoring systems:

- **Program Traces** domain [28] with low inter-connectivity (about 6 triples per subject), originally written in SWRL: builds the correct trace according to the given algorithm and determines errors in the given trace (3 datasets of traces of 5–20 lines).
- **Expressions** [29] with moderate inter-connectivity (about 40 triples per subject), originally in SWRL: finds errors in the order of evaluation of a C programming language expression and constructs a correct partial order DAG (the dataset includes a concatenation of expressions with 2–13 operands).
- **DEC** (Discrete Event Calculus) [15] with high inter-connectivity (about 75 triples per subject), originally in ASP: calculates the state of each fluent (changeable entity) at each point in time provided that events occur (in the generated dataset, each fluent maps to two events).

## 2 Method

For this study, we converted three selected rule sets, representing real-word ITS problems, to the rule syntax of all studied reasoners. The input facts were represented as RDF triples. For ASP reasoner, these were emulated by a ternary predicate `t`, while RDF terms - as a string or term literals. Inferred facts were converted back to RDF triples to extend the initial ontology. For each of the selected reasoning methods, it was necessary to convert the original SWRL rules to the native format since no reasoner other than Pellet supports SWRL with built-ins. The conversion of rules between different formalisms mainly required writing rules in different syntaxes. Some built-ins of the original rule language like regexp matching may require polyfill as syntax extension. Because of the disjunctive nature of a rule head in ASP, each rule with multiple fact assertions in its head breaks down into multiple rules asserting one fact per rule.

While the SWRL semantics assume basic RDFS inference rules, the rdf11 SWI-Prolog library supports this, while the simple Jena rules, SPARQL, and ASP solvers do not. Two of the were rules added to prepared rulesets, namely the `subClass` and `subProperty` propagation.

The solution method for each domain implemented in SWRL, is effective, but not optimal due to the limitations of SWRL. SWRL follows Open World Assumption, which is why it does not support default negation and does not provide the means to modify (update/delete) an ontology (incremental approach). Most of the compared reasoners have more rule-language capabilities than SWRL. However, for a more fair comparison of reasoners, the original solution was not changed, i.e. the semantics of the rules have not changed. This also made it possible to convert the rules programmatically, taking into account only the differences in the syntax of the equivalent constructs but not the semantics. When making comparisons, we nevertheless must keep in mind that the specific capabilities of different resonators may give them an edge on some tasks.

The semantics of the original rulesets are consistent with the output, i.e., using the deduced facts in the reasoning process. Reasoning ends when there is nothing more to display (forward chaining mechanism).

Two reasoners - Prolog (native backward chaining) and SPARQL (no immediate access to just inserted triples) do not support forward chaining. For these, we iteratively ran a batch of all the available rules (without regard to actual applicability to the data) until the size of triple store stopped increasing (thus, the order of the rules may affect the number of complete iterations).

In Prolog, a rule batch is a predicate with N definitions for N rules, ending with the `fail` command each. In SPARQL, a rule batch for N rules is a big query of N semicolon-concatenated INSERT..SELECT queries. This implementation is principally close to breadth-first search and shows results comparable to the hybrid RETE algorithm (Jena) and ASP (Clingo).

### 2.1 Measurement

The following metrics were recorded with each reasoner invoked as a standalone process:

1. Input problem size.
2. The numbers of input, inferred, and output triples.
3. `Wall time` – the running time from the start of the reasoner's application process to its completion, including i/o, rules and data parsing, and results serialization (in the case of Clingo called using the `clyngor` Python library, the time of grammar-based parsing of the printout is also included).
4. `Exclusive time` – the pure reasoning time excluding any preparation and input/output (this metric is relevant when the calling code is in the same environment – e.g., calling Jena from java code).
5. `CPU load` – average CPU usage in percent per core (similarly to Linux's `top` utility).
6. `Memory` – a peak RAM consumption defined as the average between RSS (working set) and VMS (pagefile, private bytes). The interval for collecting the indicators from the running process was 0.1 s.

The Python code organising the evaluation (using Owlready2 library [23] to handle RDF data) was not included in time measurements. All measurements

were performed on a computer with 4-cores CPU AMD A10-5745M, 8 Gb RAM, no CUDA, Windows 8.1; power profile - high applications performance.

## 3 Results and Discussion

### 3.1 Rule Definition Formalisms and Reasoners Capabilities

The inference engines (reasoners) below were studied: Pellet 2.3 (supports SWRL formalism), Apache Jena inference engine (custom rule syntax), SWI-Prolog with the semweb (rdf11) package (Prolog), Apache Jena ARQ (SPARQL Update), Clingo (Answer Set Programming).

All considered rule formalisms except SWRL follow Closed World Assumption that is sensible for local reasoning [12]. All the reasoners allow triple matching and inferring new relations and types of individuals, although SWRL and ASP do not allow retracting and modifying existing ones. Also, SWRL does not allow creating new classes and individuals.

In RDF, a data unit is represented as a triple, while Prolog and ASP support native nesting of data structures.

Rule composition is not implied by the rule formalisms, but SPARQL allows SELECT queries to be nested, and Prolog predicates are reusable.

Besides, Jena supports a kind of rules that create other rules intended to speed up the inference.

Count, sum, and other aggregates are available in SPARQL and ASP; Prolog allows implementing them. A some sort of optimized transitivity checking is presented in SPARQL as “property paths”, and Prolog can utilize recursion. The concept of integrity constraints (applied to any answer produced) exists only in ASP.

It is not possible to deactivate or prioritize rules, although it is possible to order rules in synthetic iterative approaches for Prolog and SPARQL.

Finally, all the rule languages except SPARQL provide a way to introduce custom built-ins.

Table 1 gives a summary of comparison of rule languages (formalisms) used by the studied inference engines.

### 3.2 Reasoners Performance Tests

For CWM, we ran two tests with the minimal problems, which took 10–20 times longer than using Pellet, one of the slowest reasoners.

For DLV, five tests were conducted, showing very low RAM consumption (about 4 MB) and the sharp increase in the time to solve a problem (6–10 times with a 2-fold increase in the problem size). It was clear that DLV is not scalable to big problems.

So, these two reasoners were discarded from further research because of too poor performance.

**Table 1.** Features of rule definition formalisms

| Feature | SWRL | JENA rules | Prolog | SPARQL Update | ASP (Clingo) |
|---|---|---|---|---|---|
| Open/Closed World Assumption | OWA | CWA | CWA | CWA | CWA |
| Arithmetic built-ins | Yes | Yes | Yes | Yes | Yes |
| Assert relations | Yes | Yes | Yes | Yes | Yes |
| Retract relations | No | Yes | Yes | Yes | No |
| Creating individuals | No | Yes | Yes | Yes | Yes |
| Creating classes | No | Yes | Yes | Yes | Yes |
| Data structures | Plain | Plain | Nested | Plain | Nested |
| Rule composition | No | No | Yes | Nested SELECTs | No |
| Negation support | No | Yes | Yes | Yes | Yes |
| Aggregates | No | No | User-defined | Built-in | Built-in |
| Integrity constraints | No | No | No | No | Yes |
| Transitivity check | Assert all | Assert all | Recursion | Property path | Assert all |
| Custom builtins | Yes (Java) | Yes (Java) | Yes (Prolog) | No | Yes (Lua) |
| Prioritize rules | No | No | Ordering only | Ordering only | No |

CPU load shows that Clingo and Prolog engines used only one core, while Java-based engines (Pellet, Jena, and SPARQL) used multithreading, but their memory consumption was significantly higher.

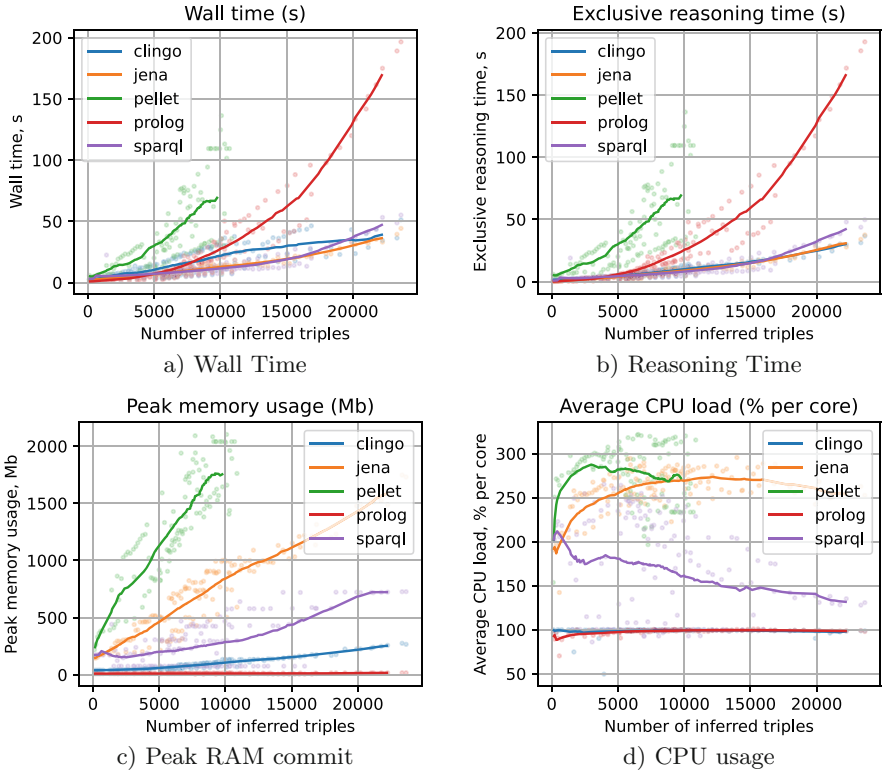
The two groups of reasoners showed significantly different results: the engines based on logic programming (Clingo, Prolog) versus Java engines with Rete algorithm or ontology querying (Pellet, Jena, SPARQL). The reasoners from the first group run on a single CPU core and spend less RAM (Fig. 4, 5). The second group of reasoners uses multithreading (2..3 cores on 4-cored CPU) and consumes a lot of memory even for tiny problems.

Prolog (running iteratively) shows low memory consumption on all tasks (Fig. 4). It has the fastest dry start and best time on light load tasks (up to 3k..5k inferred triples on Traces domain) (Fig. 3). Then its running time increases faster than that of Jena and Clingo, however, in the Expressions domain it outperformed SPARQL on large tasks as well.

Pellet reasoner consistently performed worse than all the others in terms of time and memory on all rule sets. Prolog demonstrates low memory usage regardless of the task size (in triples). It is the best reasoner for small reasoning tasks (up to about 4000 of inferred triples in the Traces domain, see Fig. 1), but it scales poorly with the task size, though it still outperforms SPARQL in the Expressions domain.

Jena turned out to be the quickest reasoner for low inter-connectivity domains, but its memory consumption rises quickly with the task size (see Fig. 2a); the reasoning can fail because of insufficient memory. For the models with a high number of links per entity (like DEC domain and, to some extent, Expressions domain), Clingo outperforms other reasoners while maintaining relatively low memory consumption.

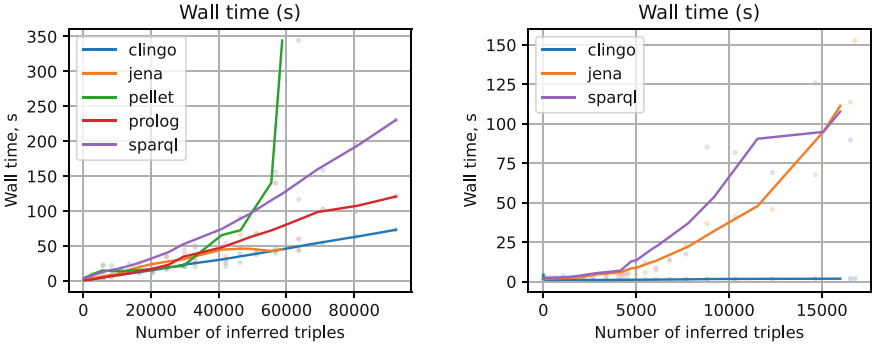
To better assess run time and memory consumption, we plotted them on the same graph (see Fig. 6; the lower value, the better). Colored lines connect the measurements of different reasoners on the same input. The vertex in the lower-left corner of any figure points to the best reasoner for this situation. It shows that in any domain Prolog steeply increases execution time, while Jena steeply increases memory consumption when the task size rise while Clingo becomes the best reasoner for large tasks.




a) Wall Time b) Reasoning Time

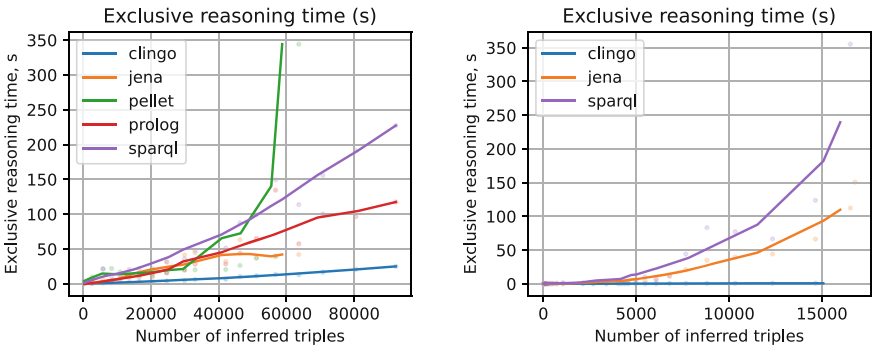
c) Peak RAM commit d) CPU usage

**Fig. 1.** Evaluation results: Traces domain (low inter-connectivity).



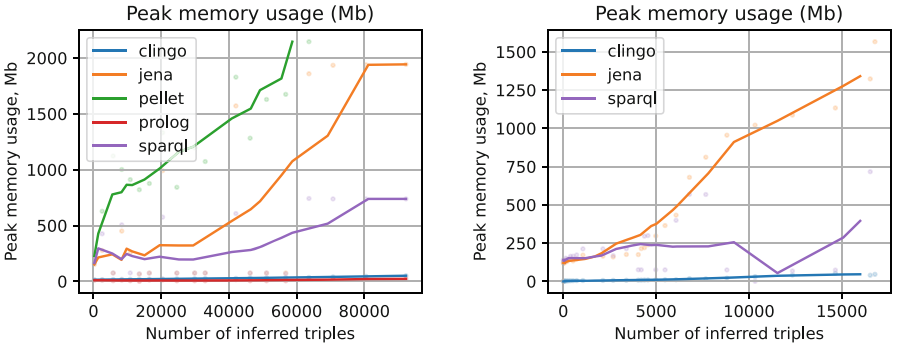

a) Expressions (middle inter-connectivity) b) DEC (higher inter-connectivity)

**Fig. 2.** Evaluation results: Wall Time in two domains.



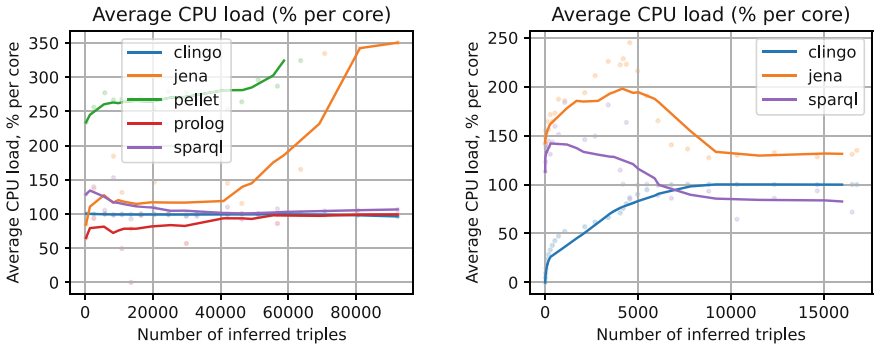

a) Expressions (middle inter-connectivity) b) DEC (higher inter-connectivity)

**Fig. 3.** Evaluation results: Exclusive Reasoning Time in two domains.



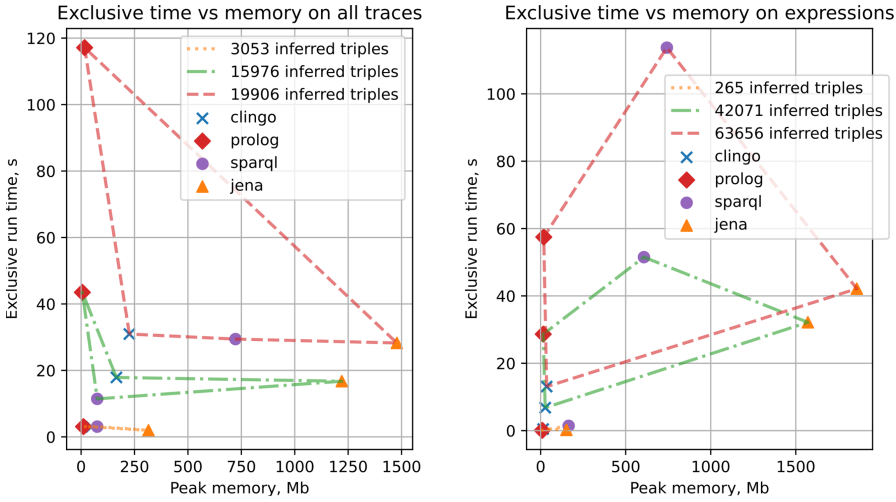

a) Expressions (middle inter-connectivity) b) DEC (higher inter-connectivity)

**Fig. 4.** Evaluation results: Peak RAM commit in two domains.




a) Expressions (middle inter-connectivity) b) DEC (higher inter-connectivity)

**Fig. 5.** Evaluation results: CPU usage in two domains.




a) Traces domain b) Expressions domain

**Fig. 6.** Time and memory consumption in two domains.

## 4 Conclusion

We compared several semantic rule engines – RDF reasoners (Pellet, Jena inference engine, SPARQL, and SWI-Prolog with rdf11) and ASP solver Clingo – as candidates for ITS backend using three different rule sets used in modern Intelligent Tutoring Systems.

CWM, released in 2000 and last updated in 2008, is no more relevant. The latest DLV (released in 2012) showed low suitability for something bigger than trivial problems.

We evaluated five rule engines on three real-world formal models related to Intelligent Tutoring Systems: Program Traces, Programming Language Expressions, and Discrete Event Calculus.

Prolog provides the fastest solution for small problems, but its efficiency drops quickly with the problem size.

Jena reasoner performs better for rule sets with low inter-connectivity but consumes much memory.

Clingo outperforms other reasoners for high inter-connectivity models, especially those implying combinatorial search while keeping memory consumption low. Its disadvantages are the lack of ability to modify the existing facts, poor RDF support, and using only one CPU core per task. Developing a multithreaded version of Clingo can make it the best reasoner for most tasks.

The options shown most stable behaviour concerning time/memory ratio are Clingo and iterative SPARQL approach.

For well-parallelable problems (especially implying combinatorial search) like DEC, Clingo shows promising speed keeping low memory consumption, although it does not provide RDF interface and disallow facts modification.

In the case of limited hardware resources, Prolog and ASP (Clingo) can be considered, because they require less space both in installation (single executable vs. java virtual machine with JARs) and in operation.

Our results can be used for selecting an inference engine for ITS according to its formal model properties. For example, they were used to choose the inference engine for the intelligent tutor "How it Works: Algorithms" [30]: Jena reasoner was chosen as the best-performing reasoner for the programming traces domain. Complex systems may use different reasoners for different models. Generally, Clingo and Jena reasoner are the best choices among modern inference engines.

**Acknowledgment.** The reported study was funded by RFBR, project number 20-07-00764 "Conceptual modeling of the knowledge domain on the comprehension level for intelligent decision-making systems in the learning".

## References

1. ARQ - A SPARQL Processor for Jena. https://jena.apache.org/documentation//query/. Accessed 30 Apr 2021
2. DLV System. http://www.dlvsystem.com/dlv/. Accessed 30 Apr 2021
3. Jena - a free and open source Java framework for building Semantic Web and Linked Data applications. https://jena.apache.org. Accessed 30 Apr 2021
4. library(semweb/rdf11): The RDF database. https://www.swi-prolog.org/pldoc/man?section=semweb-rdf11. Accessed 30 Apr 2021
5. SPARQL Update. A language for updating RDF graphs. W3C Member Submission 15 July 2008. https://www.w3.org/Submission/SPARQL-Update/. Accessed 30 Apr 2021
6. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission 21 May 2004. https://www.w3.org/Submission/SWRL/. Accessed 30 Apr 2021

7. Adrian, W.T., et al.: The ASP System DLV: Advancements and Applications. KI - Künstliche Intelligenz, pp. 177–179 (2018). https://doi.org/10.1007/s13218-018-0533-0
8. Berners-Lee, T.: Cwm: General-purpose data processor for the semantic web. http://www.w3.org/2000/10/swap/doc/cwm (2000). Accessed 30 Apr 2021
9. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Commun. ACM **54**(12), 93–103 (2011). https://doi.org/10.1145/2043174.2043195
10. Calegari, R., Ciatto, G., Mascardi, V., Omicini, A.: Logic-based technologies for multi-agent systems: a systematic literature review. Autonomous Agents Multi-Agent Syst. **35**(1), 1–67 (2020). https://doi.org/10.1007/s10458-020-09478-3
11. Chang, M., D'Aniello, G., Gaeta, M., Orciuoli, F., Sampson, D., Simonelli, C.: Building ontology-driven tutoring models for intelligent tutoring systems using data mining. IEEE Access **8**, 48151–48162 (2020). https://doi.org/10.1109/access.2020.2979281
12. Cortés-Calabuig, A., Denecker, M., Arieli, O., Van Nuffelen, B., Bruynooghe, M.: On the local closed-world assumption of data-sources. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS (LNAI), vol. 3662, pp. 145–157. Springer, Heidelberg (2005). https://doi.org/10.1007/11546207_12
13. Demaidi, M.N., Gaber, M.M., Filer, N.: OntoPeFeGe: ontology-based personalized feedback generator. IEEE Access **6**, 31644–31664 (2018)
14. Dermeval, D., Albuquerque, J., Bittencourt, I.I., Isotani, S., Silva, A.P., Vassileva, J.: GaTO: An ontological model to apply gamification in intelligent tutoring systems. Frontiers Artif. Intell. **2**, July 2019. https://doi.org/10.3389/frai.2019.00013. https://doi.org/10.3389/frai.2019.00013
15. Dougalis, A., Plexousakis, D.: AFFLOG: A Logic Based Affective Tutoring System. In: Kumar, V., Troussas, C. (eds.) ITS 2020. LNCS, vol. 12149, pp. 270–274. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49663-0_31
16. Franzoni, V., Biondi, G., Milani, A.: Emotional sounds of crowds: spectrogram-based analysis using deep learning. Multimed. Tools Appl. **79**(47–48), 36063–36075 (2020)
17. Franzoni, V., Milani, A., Mengoni, P., Piccinato, F.: Artificial intelligence visual metaphors in e-learning interfaces for learning analytics. Appl. Sci. **10**(20), 7195 (2020)
18. Franzoni, V., Pallottelli, S., Milani, A.: Reshaping higher education with e-studium, a 10-years capstone in academic computing. In: Gervasi, O., et al. (eds.) ICCSA 2020. LNCS, vol. 12250, pp. 293–303. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58802-1_22
19. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: *clasp*: a conflict-driven answer set solver. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS (LNAI), vol. 4483, pp. 260–265. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72200-7_23
20. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-shot ASP solving with clingo. CoRR abs/1705.09811 (2017)
21. Janhunen, T.: Cross-Translating Answer Set Programs Using the ASPTOOLS Collection. KI - Künstliche Intelligenz **32**(2-3), 183–184 (2018). https://doi.org/10.1007/s13218-018-0529-9
22. Kultsova, M., Anikin, A., Zhukova, I., Dvoryankin, A.: Ontology-based learning content management system in programming languages domain. Commun. Comput. Inf. Sci. **535**, 767–777 (2015). https://doi.org/10.1007/978-3-319-23766-4_61

23. Lamy, J.B.: Owlready: ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. Artif. Intell. Med. **80** (2017). https://doi.org/10.1016/j.artmed.2017.07.002
24. Liang, S., Fodor, P., Wan, H., Kifer, M.: OpenRuleBench. In: Proceedings of the 18th International Conference on World Wide Web - WWW 2009. ACM Press (2009). https://doi.org/10.1145/1526709.1526790
25. Rattanasawad, T., Buranarach, M., Saikaew, K.R., Supnithi, T.: A comparative study of rule-based inference engines for the semantic web. IEICE Trans. Inf. Syst. E101.D(1), 82–89 (2018). https://doi.org/10.1587/transinf.2017swp0004. https://doi.org/10.1587/transinf.2017swp0004
26. Singh, G., Bhatia, S., Mutharaju, R.: OWL2Bench: a benchmark for OWL 2 reasoners. In: Pan, J.Z., Tamma, V., d'Amato, C., Janowicz, K., Fu, B., Polleres, A., Seneviratne, O., Kagal, L. (eds.) ISWC 2020. LNCS, vol. 12507, pp. 81–96. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-62466-8_6
27. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: a practical OWL-DL reasoner. J. Web Semantics **5**(2), 51–53 (2007). https://doi.org/10.1016/j.websem.2007.03.004. https://www.sciencedirect.com/science/article/pii/S1570826807000169, software Engineering and the Semantic Web
28. Sychev, O., Denisov, M., Anikin, A.: Verifying algorithm traces and fault reason determining using ontology reasoning. In: 19th International Semantic Web Conference on Demos and Industry Tracks: From Novel Ideas to Industrial Practice, ISWC-Posters 2020, vol. 2721, pp. 49–53 (2020). http://ceur-ws.org/Vol-2721/paper495.pdf
29. Sychev, O., Penskoy, N.: Ontology-based determining of evaluation order of c expressions and the fault reason for incorrect answers. In: 19th International Semantic Web Conference on Demos and Industry Tracks: From Novel Ideas to Industrial Practice, ISWC-Posters 2020, vol. 2721, pp. 44–48 (2020). http://ceur-ws.org/Vol-2721/paper494.pdf
30. Sychev, O., Denisov, M., Terekhov, G.: How it works: Algorithms - a tool for developing an understanding of control structures. In: Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 2. ACM, June 2021. https://doi.org/10.1145/3456565.3460032
31. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: Swi-prolog. Theory and Practice of Logic Programming **12**(1–2), 67–96 (2012). https://doi.org/10.1017/S1471068411000494