

# Ontology Reasoning for Explanatory Feedback Generation to Teach How Algorithms Work

Anton ANIKIN <sup>a,b</sup>, Oleg SYCHEV <sup>a</sup>, and Mikhail DENISOV <sup>a</sup>

<sup>a</sup> *Volgograd State Technical university, Russia*

<sup>b</sup> *Software Engineering School, Volgograd, Russia*

**Abstract.** Developing algorithms using control structures and understanding their building blocks are essential skills in mastering programming. Ontologies and software reasoning is a promising method for developing intelligent tutoring systems in well-defined domains (like programming languages and algorithms); it can be used for many kinds of teaching tasks. In this work, we used a formal model consisting of production rules for Apache Jena reasoner as a basis for developing a constraint-based tutor for introductory programming domain. The tutor can determine fault reasons for any incorrect answer that a student can enter. The problem the student should solve is building an execution trace for the given algorithm. The problem is a closed-ended question that requires arranging given actions in the (unique) correct order; some actions can be used several times, while others can be omitted. Using formal reasoning to check domain constraints allowed us to provide explanatory feedback for all kinds of errors students can make.

**Keywords.** Ontology reasoning, Constraint-based tutoring systems, Program execution trace, Error detection

## 1. Introduction

Ontology models and formal logic reasoning are used for knowledge representation and processing in different domains for a wide range of tasks. E.g., the Ontology Driven Software Engineering (ODSE) approach [1] implies using ontology models for various aspects of software engineering: modeling different parts of software systems, products, modules, and algorithms. In [2] the ontology model is used for declarative program analysis in software development. Most of these aspects are important in introductory programming courses as well, where ontologies are widely used for domain modeling [3,4].

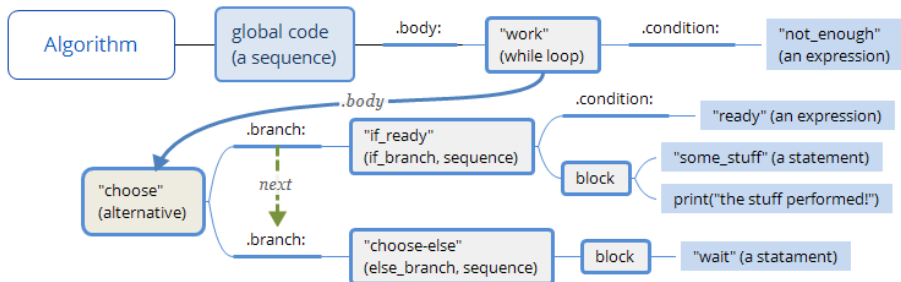
One of the efficient approaches to introducing new learners to algorithms analysis and synthesis is the trace-based teaching approach that allows to decrease the dropout and grade failures by 25.49% and 8.51% respectively [5,6,7]. According to the structured programming approach, any algorithm can be represented as a tree of control structures. In the introductory programming teaching on the Problem Formulation step [6] the algorithmic reasoning skills ("a pool of abilities that are connected to constructing and understanding algorithms: to analyze given problems; to specify a problem precisely; to find basic actions that are adequate to the given problem; to construct a correct algorithm to

a given problem using the basic actions” [8]) improvement is important. On the Solution Expression step, when the problem is formulated, and students should express a solution through programming structures, selecting the appropriate structures for solving the task is the main difficulty [9]. Finally, on the Solution Execution and Evaluation step, students should test and analyse the code to identify and correct problems, and code tracing activities are appropriate tasks on this step [5,10]. Explanatory feedback (e.g., error-flagging feedback [11] as well as other forms of explanations of student errors) has a significant effect on learning efficiency and results.

So, automated algorithm trace generation and analysis with explanatory feedback is an important task in introductory programming learning that can be solved using ontology domain models and formal logic reasoning. The reasoning rules allow to set the domain constraints and use these rules not only for the execution trace check for correctness but for the particular errors detection and corresponding explanation providing as well at the same time.

## 2. Intelligent Application to Teach Algorithms

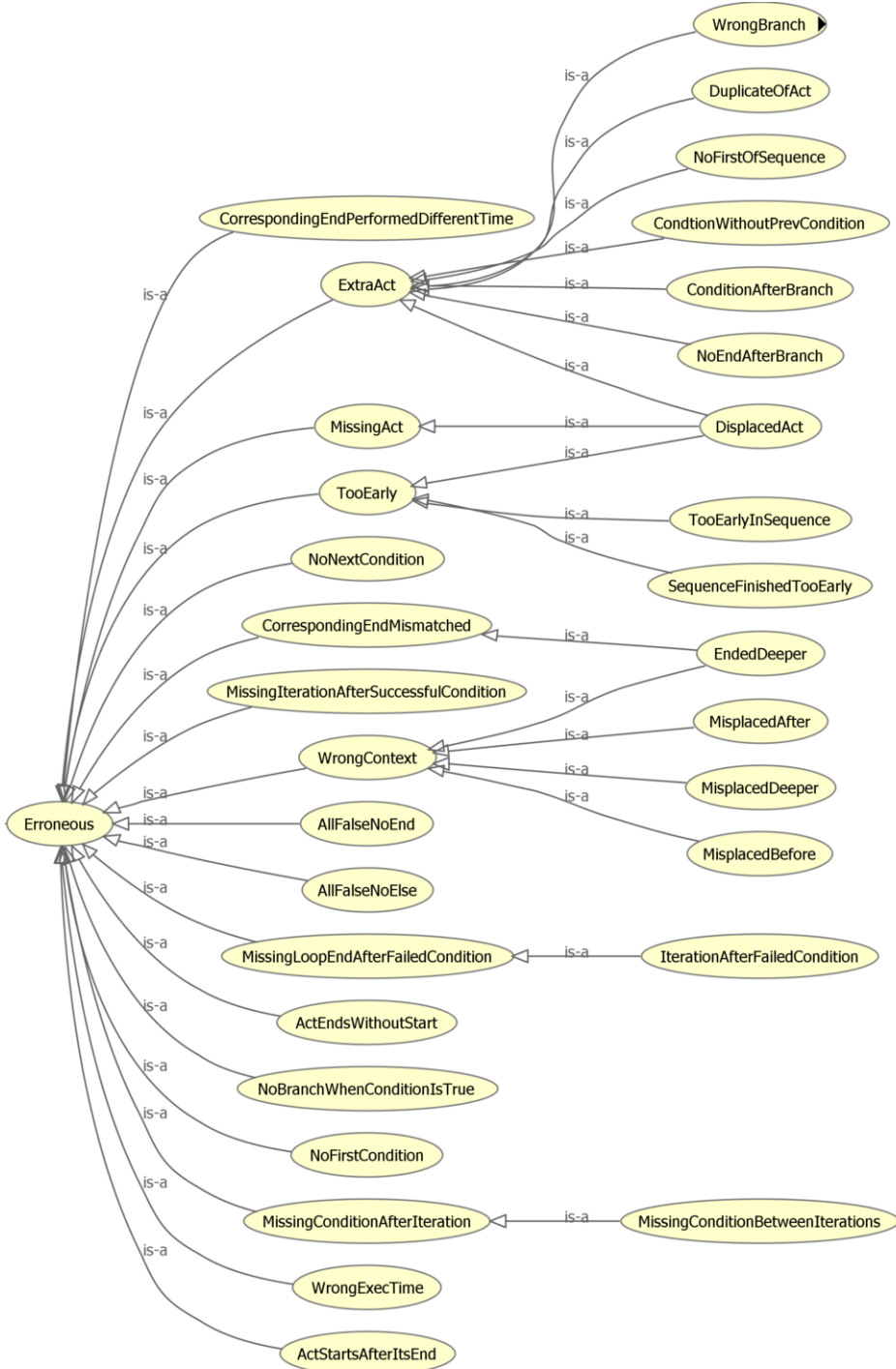
We developed an online tutoring tool *How It Works: Algorithms*<sup>1</sup> using ontology reasoning to grade students’ answers and generate explanatory feedback about their errors. Its input consists of an algorithm, represented as a tree of basic control structures – sequences, alternatives, and loops (see Fig. 1) – and the values of control conditions. The reasoner also receives the student-built trace as a sequence of control-statement execution acts. For complex control structures, the beginning and the end of their execution constitute separate execution acts to represent the nesting of control statements in the trace.



**Figure 1.** Fragment of algorithm represented as an abstract syntax tree

To generate explanatory feedback, we classified all the possible errors in execution-trace building creating 33 concepts to represent them (Fig. 2). The reasoning engine determines the error class and the additional information about the individuals related to the error for feedback generation.

<sup>1</sup><https://howitworks.app/algorithms>



**Figure 2.** Concepts for classifying errors

To select a language for the reasoning rules description and appropriate reasoning engine, we performed a study of software reasoners to find the best one for our domain, comparing Pellet, Apache Jena, Apache Jena SPARQL query processor, SWI-Prolog with semweb package, and ASP (Answer Set Programming) solvers Clingo and DLV. The results show that Apache Jena performs inference quicker than other reasoners on most of the domain-specific tasks.

In particular, Apache Jena infers the correct trace and student's errors 2.4–2.9 times quicker than SWRL Pellet reasoner. Jena rules are also more expressive than SWRL, having full CRUD operations support (e.g., creation of concepts and individuals), negation support, and relation retraction.

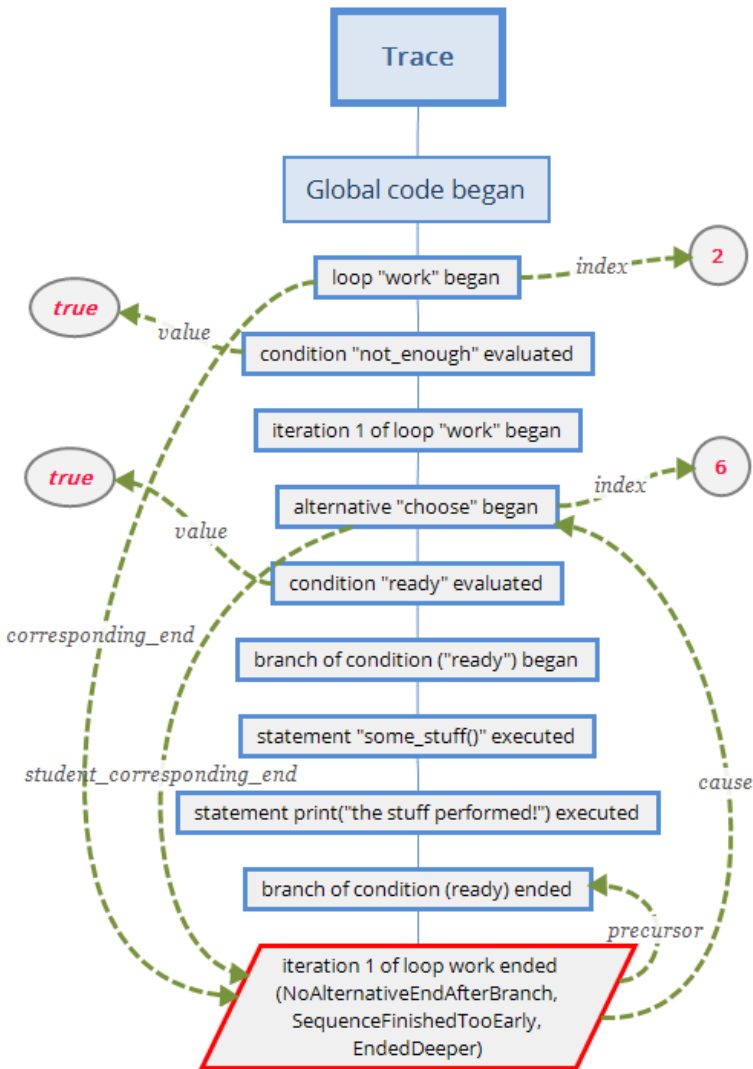



Figure 3. Execution trace processing example

So, the developed ontology contains about 30 concepts for algorithm elements, 7 trace acts, 29 kinds of errors, and 27 explanations for correct acts; 30 roles, and about 100 positive, negative, and helper rules. Using these rules, the execution trace can be generated based on the given algorithm, determine the domain constraints violated by the student, and provide all the necessary information about errors to generate explanatory feedback (Fig. 3).

The implemented software tool allows students to input the user execution trace for the teacher-defined algorithm (stored and available by URL for easy access) by clicking buttons inside the algorithm and show detailed explanatory feedback for errors made using the ontology reasoning described above (Fig. 4).

|   |
|---|
| program began   |
| loop <u>work</u> began 1st time   |
| condition <u>not enough</u> evaluated 1st time - true   |
| iteration 1 of loop <u>work</u> began   |
| alternative <u>choose</u> began 1st time  |
| condition <u>ready</u> evaluated 1st time - true  |
| branch of condition ( <u>ready</u> ) began 1st time   |
| stmt <u>some stuff()</u> executed 1st time  |
| stmt <u>print("the stuff performed!")</u> executed 1st time   |
| branch of condition ( <u>ready</u> ) ended 1st time   |
|  iteration 1 of loop <u>work</u> ended |

**You should pay attention**

- **SequenceFinishedTooEarly:** A sequence performs all its actions from the first through the last, so it's too early to finish the sequence of the body of the loop 'work', because not all the actions of the sequence have completed (ex. alternative 'choose').
- **NoAlternativeEndAfterBranch:** Each alternative performs no more than one alternative action and terminates. The alternative 'choose' has executed the 'if-ready' branch and should finish.
- **EndedDeeper:** Every act ends exactly when all its nested acts have ended, so the act of the body of the loop 'work' cannot end until the end of the act of the alternative 'choose' (the alternative 'choose' is included in the body of the loop 'work').

**Figure 4.** Explanatory feedback provided by the tutoring tool for an error in the trace

### 3. Conclusion and Future Work

In this study, we present an ontology with a set of reasoning rules that is able to build execution traces for a given algorithm, find errors in students' traces, and provide the necessary information to generate explanatory feedback about the violated constraints representing subject domain laws. The approach was implemented in a software tool, using Apache Jena inference engine for ontology reasoning. The usage of forward chaining RETE algorithm and Jena rules and reasoning allowed us to implement domain-specific rules with adequate performance to grade students' traces in real time step-by-step, showing feedback messages right after adding an erroneous line.

The software tool can be used as a basis for developing intelligent tutoring systems for improvement of algorithmic reasoning skills and developing understanding of program execution during introductory programming courses. The future work includes ex-

panding the set of supported programming languages (C++, Python and Java are implemented at this moment), supporting recursive functions in the algorithms, and developing a constraint-based intelligent tutoring system based on the proposed approach for complex exercises implementation by adding learner's model and intelligent exercise selection.

## Acknowledgements

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] D. Strmečki, I. Magdalenić and D. Kermek, An Overview on the use of Ontologies in Software Engineering, *Journal of Computer Science* **12**(12) (2016), 597–610. doi:10.3844/jcssp.2016.597.610.
- [2] Y. Zhao, G. Chen, C. Liao and X. Shen, Towards Ontology-Based Program Analysis, in: *30th European Conference on Object-Oriented Programming (ECOOP 2016)*, S. Krishnamurthi and B.S. Lerner, eds, Leibniz International Proceedings in Informatics (LIPIcs), Vol. 56, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2016, pp. 26:1–26:25. doi:10.4230/LIPIcs.ECOOP.2016.26.
- [3] M. Kultsova, A. Anikin, I. Zhukova and A. Dvoryankin, Ontology-based learning content management system in programming languages domain, *Communications in Computer and Information Science* **535** (2015), 767–777. doi:10.1007/978-3-319-23766-4\_61.
- [4] X.-T. Pham, T.-V. Tran, V.-T. Nguyen-Le, V.T. Pham and H.D. Nguyen, Build a search engine for the knowledge of the course about Introduction to Programming based on ontology Rela-model, in: *2020 12th International Conference on Knowledge and Systems Engineering (KSE)*, IEEE, 2020. doi:10.1109/kse50997.2020.9287775.
- [5] M. Hertz and M. Jump, Trace-based teaching in early programming courses, in: *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*, ACM Press, 2013. doi:10.1145/2445196.2445364.
- [6] R.P. Medeiros, G.L. Ramalho and T.P. Falcao, A Systematic Literature Review on Teaching and Learning Introductory Programming in Higher Education, *IEEE Transactions on Education* **62**(2) (2019), 77–90. doi:10.1109/te.2018.2864133.
- [7] O. Sychev, M. Denisov and A. Anikin, Ways To Write Algorithms And Their Execution Traces For Teaching Programming, in: *EpSBS - Volume 102 - NININS 2020*, European Publisher, 2021, pp. 1029–1039. doi:10.15405/epsbs.2021.02.02.127.
- [8] T. Wang, X. Su, P. Ma, Y. Wang and K. Wang, Ability-training-oriented automated assessment in introductory programming course, *Computers & Education* **56**(1) (2011), 220–226. doi:10.1016/j.compedu.2010.08.003.
- [9] S. Xinogalos, Designing and deploying programming courses: Strategies, tools, difficulties and pedagogy, *Education and Information Technologies* **21**(3) (2014), 559–588. doi:10.1007/s10639-014-9341-9.
- [10] A.N. Kumar, Solving Code-tracing Problems and its Effect on Code-writing Skills Pertaining to Program Semantics, in: *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ACM, 2015. doi:10.1145/2729094.2742587.
- [11] A.N. Kumar, Allowing Revisions While Providing Error-Flagging Support: Is More Better?, in: *Lecture Notes in Computer Science*, Springer International Publishing, 2020, pp. 147–151. doi:10.1007/978-3-030-52240-7\_27.