Prototyping Logic-Based AI Services with LogicUS

Victor Ramos-González, Joaquin Borrego-Díaz, Fernando Sancho-Caparrini

Department of Computer Science and Artificial Intelligence – University of Seville, Seville, Spain
vramos1@us.es, jborrego@us.es, fsancho@us.es

Abstract

Currently, there is renewed interest in logic-related solutions for AI and Computer Science. The availability of software tools to support the realization of such studies (both as powerful and versatile prototyping tools and as teaching tools) has become a necessity. Intending to contribute to this field, we present a tool that allows the unification of different logic tasks, focused on Computer Logic but adaptable to the treatment in several subfields, contexts, and abstraction levels (*LogicUS-LIB*, *LogicUS-NB*, *LogicUS-GUI*).

The tool provides a sound framework for two activity fields. On the one hand, in the topic of logic-based systems research, prototyping is facilitated in a relatively fast, simple, and highly adaptable way. On the other hand, in Education, by allowing the student abstract from low-level execution of algorithms whilst preserving the conceptual structures and procedural methodologies underlying the logical foundations.

Introduction

The increase in computational capacity and the development of different areas have produced powerful results in many Computer Science subfields, particularly in those related to Logic: Mathematical/Computational (Automated/Interactive Theorem Proving, Automated Formalization, etc.), Logistics and Optimization (CSPs, Automated Reasoning & Planning, etc.), System Modeling and Verification (MAS, Model Checking, etc.), and Intelligent Systems (Neuro-Symbolic approaches, dMARS, XAI, etc.)[1]. These trends highlight the need for intuitive and versatile tools -at the academic and research levels- that enable abstracting the user from the ground logic process and focusing on prototyping AI applications (e.g. (Huang et al. 2017) and (Borrego-Díaz and Galán-Páez 2022) in the field of Data Science).

Our work aims to provide a framework that meets such needs. Currently, this is focused on Classical Logic (but adaptable to others) and oriented both to the academic (as a teaching tool[2]) and the research use as a prototyping tool.

LogicUS[3] provides a web-based tool allowing to work with Propositional (PL) and First-Order Logic FOL (representation structures and operational procedures) as the essential layer over which to build sophisticated and logic-based procedures. It is open to the development of other logics, by adding the specific structures and mechanisms.

[1]See e.g. (Calegari et al. 2020) for a recent overview.

[2]It was sketched in ISSAC congress in an unpublished demonstration, https://www.issac-conference.org/2022/software.php

[3]https://logicus-es.github.io

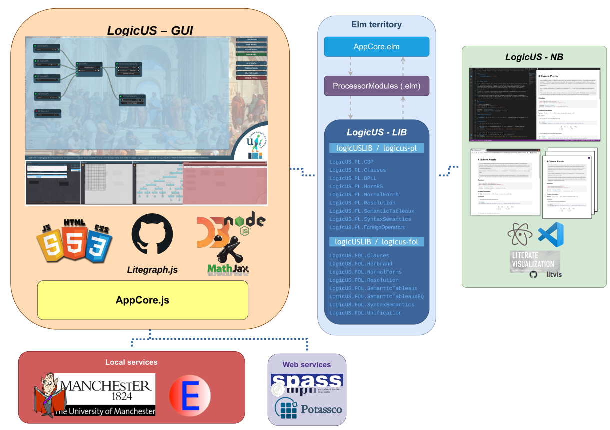

Figure 1: Architecture of *LogicUS* (its three main components LIB, NB and GUI). Elm is base calculus language, *Litegraph.js* it the technology for user interaction, *litvis* for document generation, and *JS*, *Node* for third-party services.

The *LogicUS* System

LogicUS system consists of three main tools providing different levels of abstraction: *LogicUS-LIB* (functional engine; functional libraries programmed in Elm), *LogicUS-NB* (an integration with *litvis* to generate executable documents) and *LogicUS-GUI* (an interface GUI oriented to web and executable flows for visualization and integration).

These three components are designed according to efficiency and usability standards as driving forces. Thus, the tool provides a particularly useful system in both fields, research (through a relatively fast, simple, and highly adaptable way for logic-based prototyping), and education (through a framework that makes low-level execution of algorithms while preserving the conceptual structures and procedural methodologies underlying the logical foundations).

LogicUS-LIB: The Functional Core

This is the computing engine of the system. It contains several packages: (i) the implementation (in Elm, a pure functional web-oriented language) of all the necessary types for the different logical elements (formulas, clauses, interpretations, substitutions, etc.), (ii) the functions required for their treatment, and (iii) decision algorithms that allow working with both PL (LogicUS.PL[4]) and FOL (LogicUS.FOL[5]). To facilitate interoperability with rendering interfaces, it also includes some representation mechanisms (to LaTeX, *GraphViz/DOT* and *CNF-DIMACS* notation, for example).

To prevent users rebuild well-known procedures, they can exploit structures and functions already provided (they can be publicly shared).

LogicUS-GUI: A Flow-Design and Web-Based GUI

LogicUS can be used directly from the REPL console provided by Elm (by importing the LogicUS-LIB libraries) and may be an interesting use for some users. However, this approach is far from the visual, convenient, and reusable methods usually required by researchers for their proofs; and even more so from the objectives of standard Logic courses. For this reason, *LogicUS-GUI* provides a complete front-end guided by the principles of accessibility and usability. This tool integrates Elm with web technologies (including HTML, CSS, JavaScript, and LiteGraph as main contributors), providing a high abstraction layer.

The design of Elm as a Web-oriented language and its ability to compile to JavaScript allows (i) the construction of a pure Web interface, which is executable on any machine with a browser regardless of the OS; (ii) the integration of other Web technologies such as Litegraph (as an essential method of user interaction with the system) or D3.js (for the proper visualization of the graphs); (iii) the connection with robust solving systems.

An interesting feature is the use of Litegraph.js as underlying technology for system-user interaction. This belongs to the visual programming paradigm by creating execution graphs, an ideal way to reflect the problem-solving methodology and the possibility of observing the partial-step results, a point of study in every system. The graphs may include nodes of different types (providing different functionalities), selectable from a menu of available nodes, and connectable to each other, fully interactive with the mouse. Moreover, Litegraph within JS and Elm give us other characteristics including node options, model saving/loading, third-party connection (web by *HTTP Requests* or local by *Node.js*). That creates an nice environment for research prototyping but also, for educational purposes.

LogicUS-NB: Executable Documents for Logic

Lastly, to relieve some limitations of GUI (e.g. L-structures in FOL), an intermediate environment, *LogicUS-NB*, is provided. It is notebook-like system powered by *litvis*[6]. This way, a module to connect Markdown (enhanced with LaTeXand other visualization engines as *Tikz* or *GraphViz*) with the Elm run engine is provided. This allows the embedding of *Elm chunks* containing executable Elm code, that is executed and properly rendered in the document.

The tool, within the representation method provided in *LogicUS-LIB*, allows the user to model a complete problem by using logical formulas and then solve it by applying algorithms implemented in *LogicUS-LIB* (or even using the document itself for coding there the algorithm). All this through a markdown file that supports several options for visualization and different outcome formats (HTML or PDF, among others) by using standard editors (Atom or VSCode).

That provides an ideal environment for developing, simultaneously, the code of the model and the documentation, which is directly useful for researchers (who can, easily, share these documents with all the material embedded) and teachers (who can integrate the theoretical concepts with the practical demonstrations).

Conclusions and Related Work

A concise description of *LogicUs* has been presented. It is focused on its threefold module architecture of increasing abstraction/generalization that enable, in turn, three types of system usage. Its modularity, besides the use of functional programming, allows its easy extension (including new functions in *LogicUs-LIB*) and the exploiting of the existing ones to prototype logic-based reasoning methods.

LogicUS-GUI enhances its didactic facet with module that allows the student who has taken Computational Logic to study logic-based AI methods by designing them by connecting modules that implement well-known algorithms. The idea of the system meets some of the prerrequisites for materials for AI (Stadelmann et al. 2021). It also shares some design ideas with (de Oliveira, Oliveira, and da Rocha Falcão 2018) and (Alonso, Aranda, and Martín-Mateos 2007)[7]. Our design decisions contrasts with the use, common in undergraduate courses, of automated theorem provers (e.g. (Kyrilov and Noelle 2015)), Prolog/ASP[8] (Ribeiro, Simões, and Ferreira 2009), or ad hoc implementations of selected reasoning and representation methods: Without aiming for completeness: (del Vado Vírseda and Morente 2011), *Tree Proof Generator*[9], *DPLL Demo*[10], SLI[11], Matita (Asperti et al. 2007) *SATRennesPa*[12], *ProofTools*[13] and PIE[14].

Acknowledgments

This work is supported by Spanish State Investigation Agency (Agencia Estatal de Investigación), Project PID2019-109152GB-I00/AEI/10.13039/501100011033.

[4]package.elm-lang.org/packages/logicUSLIB/logicus-pl

[5]package.elm-lang.org/packages/logicUSLIB/logicus-fol

[6]https://github.com/gicentre/litvis

[7]https://www.glc.us.es/apli2/login/

[8]e.g. https://www.cs.utexas.edu/users/vl/teaching/lbai/

[9]https://www.umsu.de/trees/

[10]http://people.irisa.fr/Francois.Schwarzentruber/dpll_demo/

[11]https://www.infor.uva.es/~arancha/SLI.html

[12]*http://satrennespa.irisa.fr/WebContent/*

[13]https://creativeandcritical.net/prooftools

[14]http://cs.christophwernhard.com/pie/

References

Alonso, J. A.; Aranda, G. A.; and Martín-Mateos, F. J. 2007. KRRT: Knowledge Representation and Reasoning Tutor System. In Moreno Díaz, R.; Pichler, F.; and Quesada Arencibia, A., eds., *Computer Aided Systems Theory – EUROCAST 2007*, 400–407. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-75867-9.

Asperti, A.; Coen, C. S.; Tassi, E.; and Zacchiroli, S. 2007. User Interaction with the Matita Proof Assistant. *J. Autom. Reason.*, 39(2): 109–139.

Borrego-Díaz, J.; and Galán-Páez, J. 2022. Explainable Artificial Intelligence in Data Science: From Foundational Issues Towards Socio-Technical Considerations. *Minds and Machines*, 32(3): 485–531.

Calegari, R.; Ciatto, G.; Denti, E.; and Omicini, A. 2020. Logic-Based Technologies for Intelligent Systems: State of the Art and Perspectives. *Information*, 11(3).

de Oliveira, N. I.; Oliveira, M. V. M.; and da Rocha Falcão, J. T. 2018. An Automated Environment for Teaching Programming Logic on Distance Learning IT Courses. In McLaren, B. M.; Reilly, R.; Zvacek, S.; and Uhomoibhi, J. O., eds., *Proceedings of the 10th International Conference on Computer Supported Education, CSEDU 2018, Funchal, Madeira, Portugal, March 15-17, 2018, Volume 1*, 267–274. SciTePress.

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: 1907–1916. Proceedings of the International Conference on Computational Science, ICCS 2011.

Huang, X.; Kwiatkowska, M.; Wang, S.; and Wu, M. 2017. Safety Verification of Deep Neural Networks. In Majumdar, R.; and Kunčak, V., eds., *Computer Aided Verification*, 3–29. Cham: Springer International Publishing. ISBN 978-3-319-63387-9.

Kyrilov, A.; and Noelle, D. C. 2015. Using Automated Theorem Provers to Teach Knowledge Representation in First-Order Logic. *CoRR*, abs/1507.03670.

Ribeiro, P.; Simões, H.; and Ferreira, M. 2009. Teaching Artificial Intelligence and Logic Programming in a Competitive Environment. *Informatics in Education*, 8(1): 85–100.

Stadelmann, T.; Keuzenkamp, J.; Grabner, H.; and Würsch, C. 2021. The AI-Atlas: Didactics for Teaching AI and Machine Learning On-Site, Online, and Hybrid. *Education Sciences*, 11(7).