

Blawx: Web-based user-friendly Rules as Code

Jason Morris¹

¹*Lexpedite Legal Technology Ltd., Sherwood Park, Alberta, Canada*

Abstract

This paper describes Blawx, a prototype web-based user-friendly Rules as Code tool, powered by a goal-directed answer set programming. The paper briefly describes Rules as Code, and introduces desirable qualities for Rules as Code tools. It provides justifications for Blawx's implementation of the Google Blockly library, and the s(CASP) reasoning system. It then provides a step-by-step tour of how Blawx allows a user to generate an answer set program representing their understanding of a statute, and use that encoding to power an application. The paper concludes with a brief discussion of the current short term and anticipated long-term development objectives for Blawx.

Keywords

Rules as Code, s(CASP), Legal Technology, Explainable AI, Expert Systems

1. Rules as Code

Rules as Code¹ is a proposed methodology of public administration that calls ideally for rules be drafted in natural language, and represented in executable languages at the same time, or as soon as possible thereafter. The anticipated benefits of this approach include improvements to legislative drafting quality, social policy design, and simplifying the development of legally compliant automations for service delivery and compliance.

2. Blawx

Blawx², which is a portmanteau of "law" and "blocks", is a project I have been working on since December of 2017. It is my attempt to create a platform for Rules as Code that satisfies a number of critical requirements, so as to advance the development and adoption of Rules as Code. Those requirements include:

- open source, and freely available, so as to encourage adoption, experimentation, and to ensure transparency
- easy for subject matter experts to use, to facilitate adoption and avoid the knowledge acquisition bottleneck


GDEASP '22: Workshop on Goal-Directed Execution of Answer Set Programs, August 1, 2022, Haifa, Israel

✉ jason@lexpedite.ca (J. Morris)

🌐 <https://www.lexpedite.ca/> (J. Morris)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

¹See, e.g. Mohun, J. and A. Roberts (2020), "Cracking the code: Rulemaking for humans and machines", OECD Working Papers on Public Governance, No. 42, OECD Publishing, Paris, <https://doi.org/10.1787/3afe6ba5-en>.

²Available at <https://github.com/lexpedite/blawx>

- sophisticated reasoning features, to ensure the representations are as useful as possible
- natural language explanations for conclusions linked to legislative source material, to earn trust of experts, developers, and users
- ease of integration with other technologies, to increase the value of encodings and promote adoption

While there are a number of commercial and open source offerings with some of these qualities, several of which have inspired aspects of this work³, Blawx is to my knowledge the only offering with all of them.

Since late 2021, development of Blawx has been guided primarily by the needs of the Rules as Code Programme at the Canada School of Public Service, which is supporting departments across the Government of Canada interested in exploring the Rules as Code methodology.

2.1. Blockly

Blawx features a drag-and-drop interface in which an answer set program is represented by "blocks" which connect to one another in a way that is visually similar to connecting pieces of a jigsaw puzzle.

Blawx's visual interface is developed using Google's Blockly library⁴, which is based in research on teaching programming to children, and utilized in very popular pedagogical tools such as Scratch⁵. The puzzle-piece interface reduces the types of syntax errors that are possible compared to a traditional programming environment, and provides more immediate feedback with regard to others. This minimizes the cognitive load on beginning users who typically struggle to differentiate between syntactic and semantic errors.

2.2. s(CASP)

Work began in December of 2021 on re-implementing Blawx using s(CASP)⁶ as the back-end reasoner, moving away from Flora-2⁷. The decision to re-implement in s(CASP) was taken after investigations into s(CASP) that took place in part at the Singapore Management University Centre of Computational Law and continued inside the Digital Experience and Client Data division of Service Canada's Benefits Delivery Modernization programme. Those experiments demonstrated its feasibility as a tool for diagnosing and correcting legislative drafting errors⁸, for powering expert systems⁹, and for dealing with the data types most commonly used in our target use cases¹⁰.

³See, in particular, AustLII's Datalex at <https://www.datalex.org>

⁴Available at <https://github.com/Google/Blockly>

⁵Available at <https://scratch.mit.edu>

⁶Arias, Joaquin, et al. "Constraint answer set programming without grounding." *Theory and Practice of Logic Programming* 18.3-4 (2018): 337-354.

⁷Available at <http://flora.sourceforge.net>

⁸Morris, Jason. "Constraint answer set programming as a tool to improve legislative drafting: a rules as code experiment." *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Law*. 2021.

⁹Docassemble-L4 is available at <https://github.com/smuclaw/docassemble-l4>

¹⁰Presented at Programming Languages and the Law 2022 workshop as "Prevalence of Expression Types in Legislative Text", see <https://popl22.sigplan.org/details/prolala-2022-papers/11/Prevalence-of-Expression-Types-in-Legislative-Text>

These experiments revealed several potentially important benefits from using s(CASP) that were not anticipated. In addition to the critically important feature of natural language explanations for conclusions, experience showed that multiple stable models, dual programs, and abducibility provide the legal knowledge engineer with a vastly better understanding of what their encoding actually means in the development phase, and in turn much greater confidence that the semantic meaning of the code is consistent with their interpretation of the rules.

3. Rules as Code in Blawx

The user experience of using Blawx for Rules as Code involves providing the legislative text, encoding the legislative text in a structurally-isomorphic way, testing those encodings, and then integrating those encodings into other systems.

3.1. Step 1: Getting the Law Into Blawx

In order for encodings to be structurally isomorphic to the law from which they are derived, Blawx needs to be aware of the structure of the source law. Akoma Ntoso¹¹ (also known as LegalDocML), is used to represent legislative texts inside the Blawx system. Open source, user-friendly editors for LegalDocML are rare, and those that do exist use styles of legislative drafting unfamiliar in the Canadian context.

For that reason, I created CLEAN¹² (Canadian Legislative Enactments in Akoma Ntoso), which is a markdown-like plain text syntax for legislative text, with tools for converting it into Akoma Ntoso. Blawx allows the user to specify legislation in CLEAN. The interface is shown in Figure 1.

3.2. Step 2: Encoding the Law

Once the law has been entered, the user is presented with a side-by-side coding environment with the text of the law on the left, and the Blawx coding environment on the right. The text of the law is annotated with radio button selectors, which are used to select the hierarchical section of the law for which the user wants to generate code. If the user wants to encode parts smaller than a numbered section of the law, CLEAN allows for a span syntax to divide a block of text into arbitrarily deeply nested named parts, which Blawx will treat as though they were numbered sections of the law. This interface is seen in Figure 2.

The user selects one section of the law, and represents that section of law in the coding window. Ontology (categories, attributes, and objects) declared in any section of the law become available for use in all sections of the law, eliminating the need to redefine terms that have been defined elsewhere.

When sections of the law are expressed in a non-monotonic way, such as defaults and exceptions, the user has access to language elements that allow them to attribute a conclusion to a section of law, specify which conclusions conflict, and specify how to resolve those conflicts as

¹¹Palmirani, Monica, and Fabio Vitali. "Akoma-Ntoso for legal documents." *Legislative XML for the semantic Web*. Springer, Dordrecht, 2011. 75-100.

¹²Available at <https://github.com/lexpedite/CLEAN>

Figure 1: The user interface for describing legislation in Blawx

Ruledoc name:

Rule text:

Bird Act

1. A Penguin is a Bird.
2. If a thing is a bird, it flies, [penguin]
{unless it is a penguin}.

Figure 2: The user interface for selecting a section of law to encode

- ▾ Bird Act
 - 1 A Penguin is a Bird.
 - 2 If a thing is a bird, it flies, unless it is a penguin .

between two rules. In this way, the user is able to maintain structural isomorphism regardless of whether in the legislative text the conflict is explicitly resolved in the default (e.g. "subject to following sections"), or in the exception (e.g. "despite section 3 above"). These statements are resolved using a very simple argumentation theory inspired by the approach taken in Flora-2¹³. An example of an encoding using defaults and exceptions is shown in figure 3.

Figure 3: An Encoding of the selected section, using defeasibility features

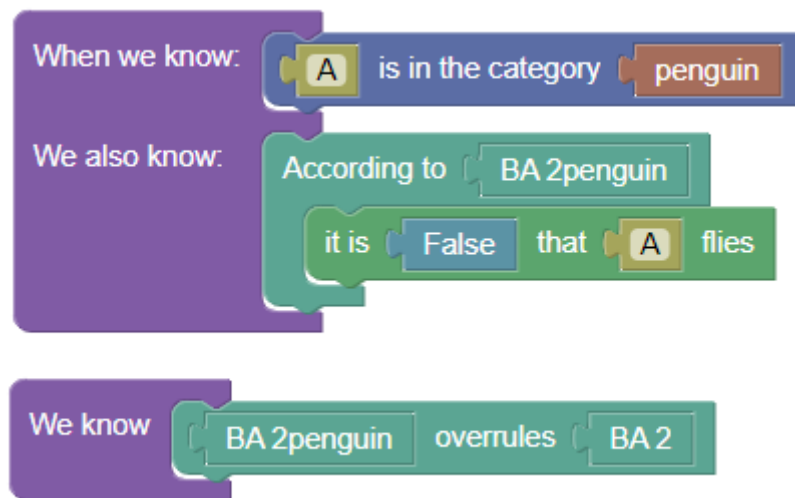
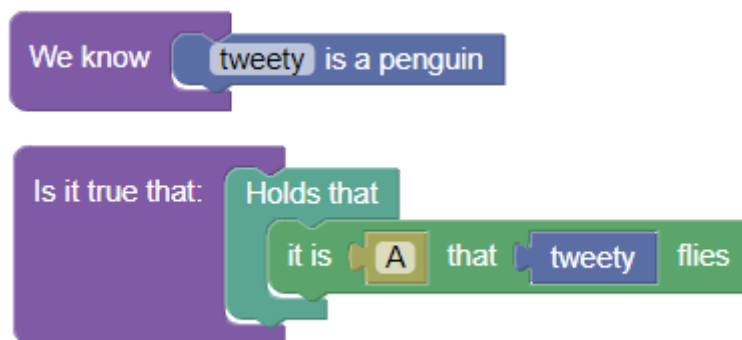


Figure 4: A Test of the Bird Act



3.3. Step 3: Testing the Encoding

Once the law has been encoded, the user is able to create "tests", which are another coding environment with access to all of the ontology declared in the encoding of the law itself. Here, the user can create a fact scenario and a query, have the reasoner run that query, and display the results. If the conclusion is one to which the defeasible argumentation theory applies, there is a version of the query that allows the user to ask whether a given legal conclusion holds, despite being defeasible. The user can also ask whether a conclusion held according to any section of the law. An example test is shown in Figure 4.

¹³Wan, Hui, et al. "Logic programming with defaults and argumentation theories." International Conference on Logic Programming. Springer, Berlin, Heidelberg, 2009.

s(CASP) code is generated from the Blawx code, and saved to the server. When the test is run, the server collects all of the s(CASP) code for the sections of the law, adds the s(CASP) code generated for the test, and adds libraries that are used for defeasibility and date calculations. The query is then sent to SWI-Prolog¹⁴ using the s(CASP) library¹⁵, and the results are displayed to the user. Models with the same bindings are grouped into an "answer". Each "answer" shows its bindings (if applicable) and one or more "explanations", which are a collapsible tree view of the natural language justification provided by the s(CASP) reasoner. When the defeasibility elements of the encoding refer to sections of the source law, those sections are highlighted in the text of the explanation. Hovering over the highlighted text display the text of the legislative section to which they refer. An example of the answer display is shown in Figure 5.

Figure 5: The Results from Running a Test

Output Problems Code

Answer #1 ^

- A: false

Explanation #1 ^

- ∇ it legally holds that it is false that tweety flies, because
 - ∇ according to [section 2 span penguin](#), it is false that tweety flies, because
 - tweety is a penguin
 - ▷ there is no evidence that the conclusion in section 2 span penguin is defeated, because

3.4. Step 4: Deploying the Encoding

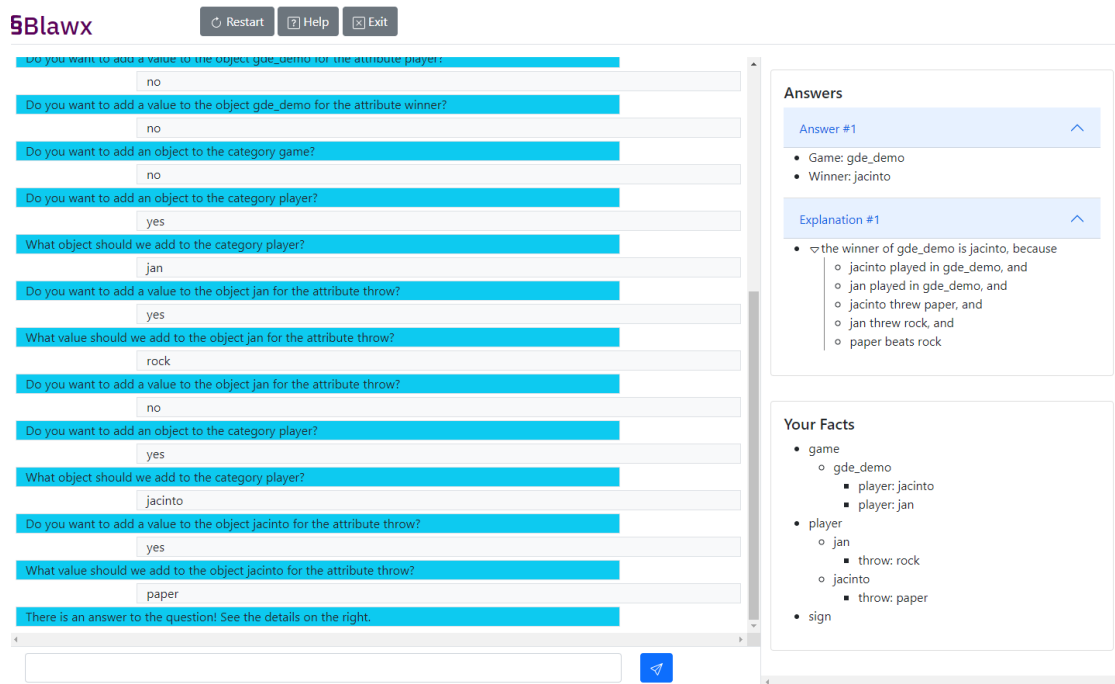
Each test that is created in Blawx automatically generates three web API endpoints designed to allow the user to collect information about the ontology defined in a given test, and to run the included queries with or without information about what additional inputs might be relevant to

¹⁴Available at <https://swi-prolog.org>

¹⁵Available at <https://github.com/JanWielemaker/sCASP>

finding stable models for the query. These interfaces are designed to allow the Blawx encoding to be able to power expert systems and other tools elsewhere on the web. As a demonstration of the use of these interfaces, an example expert system called "BlawxBot" is available. Running BlawxBot displays a chatbot-style user interface where the user is prompted to provide the details of a relevant fact scenario. As soon as a stable model exists the interview is terminated and the result is displayed to the user. An example of the BlawxBot interface is shown in Figure 6.

Figure 6: The BlawxBot Interface



4. Future Work

Work on Blawx in the near term will continue to be guided by the needs of the Canada School of Public Service and its client departments. Currently, our priorities are:

- Complete BlawxBot expert system chatbot interface
- Enhance documentation, examples, and training materials

We then hope to test Blawx extensively with real-use world cases, and use feedback from the users to guide future development.

5. Conclusion

The Rules as Code movement suggests that legal knowledge representation has great potential benefits for policy design, legislative drafting, public administration, compliance, and other endeavours. Our experiments have shown that s(CASP) has great potential as a tool for statutory legal knowledge representation. Blawx is an attempt to facilitate the use of s(CASP) by non-programmer subject matter experts for Rules as Code tasks, to explore how we might unlock that potential. Recent months have seen a focus on developing a prototype which is now almost complete. Work will continue inside the Government of Canada testing that prototype against real-world use cases to improve both the tool and our understanding of the Rules as Code methodology generally.

Acknowledgments

My thanks to Pia Andrews, and to the DECD team at the Benefits Delivery Modernization Programme in Service Canada. Thanks also to the visionary leaders at the Canada School of Public Service who are leading the world in commitment to exploring the Rules as Code approach in the public sector. In particular, thanks to Marty Perron of CSPS, who has provided valuable direct feedback on Blawx's development.

Thanks also to the leadership and staff of the Singapore Management University Centre for Computational Law.

Thank you also the developers of s(CASP) for their support and assistance, and to the organizers of this conference for their interest in my work in the legal domain.

Research having occurred at the Singapore Management University Centre for Computational Law was supported by the National Research Foundation (NRF), Singapore, under its Industry Alignment Fund – Pre-Positioning Programme, as the Research Programme in Computational Law. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.