Building Blawx

Jason Morris[1]

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

Abstract
Blawx was introduced as a user-friendly graphical environment for statutory legal knowledge representation by non-programmers using goal-directed execution of constraint answer set programming at GDE '22.

Development of Blawx has continued, and this presentation will showcase significant changes made in the last year, including: * a re-implementation of defeasibility, * a new event reasoning method, * a friendly user interface for specifying open-world knowledge (and uncertainty) using user-defined predicates, * improvements to natural language generation in explanations, * a constraint-based implementation of dates, times, and durations, * lists and aggregates, * a revision to how boolean attributes are encoded, * user-defined predicates of arity > 2, and * integration with LLM for summary of explanations.

The presentation will also briefly discuss the experience thus far of training non-programmers to use Blawx inside the government of Canada, the challenges that have been faced in integrating Blawx encodings with other systems, the types of encodings users are generating, and the uses to which those encodings are being put.

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

1. Blawx

Blawx is a prototypical open source web-based development environment for statutory knowledge representation, designed for ease of use for non-programmers, and using the s(CASP) goal-directed constraint answer set programming language[1]. The user interface is built using the Blockly visual development environment library[2], and is a drag-and-drop block-style interface, with an associated web application.

Blawx[3] is intended to serve as a learning tool for experiments into "Rules as Code"[4], which is an approach to public service that calls for the encoding of the semantic content of statutory, regulatory, and other normative texts as soon as possible in their life-cycle so as to get benefits in policy design, legislative drafting, public administration, and compliance.

GDE 2023: 3rd International Workshop on Goal-directed Execution of Answer Set Programs, London, UK, July 10, 2023
jason@lexpedite.ca (J. Morris)
https://www.lexpedite.ca/ (J. Morris)

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

[2]Available at https://github.com/Google/Blockly

[3]Available at https://github.com/lexpedite/blawx

[4]See, 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.

Work to re-implement Blawx using the s(CASP) language began in late 2021, and since being introduced in August of 2022[5] there have been significant changes to the tool's reasoning features and user interface. Blawx has also seen significant use inside the Canadian federal government, which we report on briefly below.

2. Building Blawx's Reasoning Features

Several improvements have been made the Blawx visual language, to take better advantages of s(CASP)'s reasoning methods, or to add or re-implement higher-level reasoning features.

2.1. Booleans

The initial implementation of Blawx treated boolean attributes of objects as a binary predicate over an object and a boolean value, with booleans treated as a separate data type in the coding interface. This caused problems with allowing s(CASP) to predict when statements were contradictory. Our experience also indicated that users had difficult ascertaining what the difference was between saying that it was "logically false that the value of an attribute was true", and saying "the value of an attribute is false." This also confused efforts to generate user interfaces for specifying facts. Booleans in Blawx have therefore been re-implemented as unary predicates on objects, so that logical negation is the only way to express falsehood, and the boolean data type has been removed in favour of using predicate statements, and logical negation.

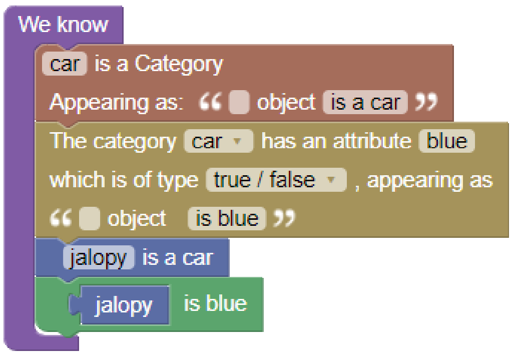

We know
car is a Category
Appearing as: “ object is a car ”
The category car has an attribute blue
which is of type true / false , appearing as
“ object is blue ”
jalopy is a car
jalopy is blue

Figure 1: The Revised User Interface for Boolean Attributes

2.2. Constraint Date Math

The initial implementation of date math in Blawx used a mathematical approach (using Prolog's "is" operator), which limited opportunity to take advantage of s(CASP)'s reasoning over numerical constraints in the context of dates. That implementation has been replaced with an

[5]See the GDEASP 2022 presentation at http://platon.etsii.urjc.es/ jarias/GDE-2022/GDE-04.pdf

implementation in which dates are represented using timestamps, and date calculations are performed using constraint math. This improves Blawx's ability to reason over dates, but the method in which it was implemented required sacrificing Blawx's ability to deal with durations of irregular size (i.e. months and years). Code generation takes data of those types and generates numerical values in a term such as `datetime(12345)`, and that term is reformatted for display to the user in explanations.

2.3. Defeasibility

Ease of maintenance and explainability of encodings in Blawx are dependent on structural isomorphism, a strong one-to-one relationship between the smallest possible sections of legal text and corresponding sections of code. Maintaining structural isomorphism requires allowing the sections of code to interact with one another in a way analogous to the manner in which sections of law defeat one another in statutory text. The original implementation of defeasibility was found to be computationally inefficient.

To address this problem, the defeasibility library was re-implemented in such a way as to flatten the higher-order structure of the predicates that were used to describe the defeating relationships. For example, what was previously encoded as `according_to(section_1,plays(bob,baseball))` is now encoded as `according_to(section_1,plays,bob,baseball)`, with only minimal changes to the user interface.

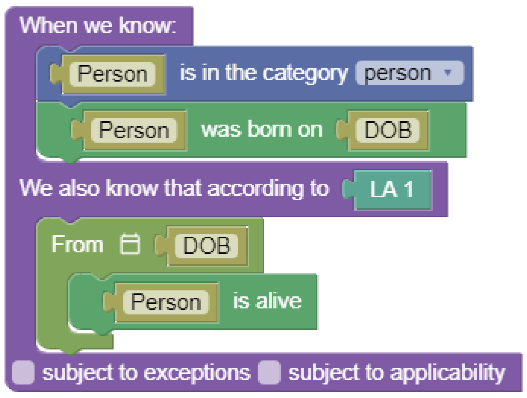
The block language also has features to allow the user to attribute a conclusion to the current section (by default), to exclude or include certain conclusions from checking for defeaters, and to decide whether to check for "applicability" for any objects defined as members of a category. The exception option is a small sacrifice of structural isomorphism, as if the law is amended to an an exception to a rule that previously did not have one, it may now be necessary to enable defeasibility in the new defeated rule in order to implement that change.

2.4. Event Reasoning

Most recently, a simple version of event reasoning has been added to Blawx, taking advantage of s(CASP)'s ability to do numerical constraint calculations over dates. The interface includes three predicates which are used to specify events in time ("initially", "from", and "ultimately"), and two predicates that are used to query the truth value of statements at a given point in time ("as of" and "between"). This feature is still under active development, and we hope to be able to ascertain its effectiveness over the coming months.

2.5. Ternary+ Predicates

Blawx previously allowed only for the creation of unary predicates (which represented "Categories", and after modification, boolean attributes of objects), and binary predicates (which represented attributes of an object). This limitation was originally intended to provide a limited structure for navigation in user interfaces using the Blawx API. However, with the changes to the Scenario Editor (described below) and corresponding changes to the API, this motivation was no longer relevant. As such we have added the ability to declare predicates of between 3 and 10 parameters, referred to as "relationships". This has had the effect of making the modelling of

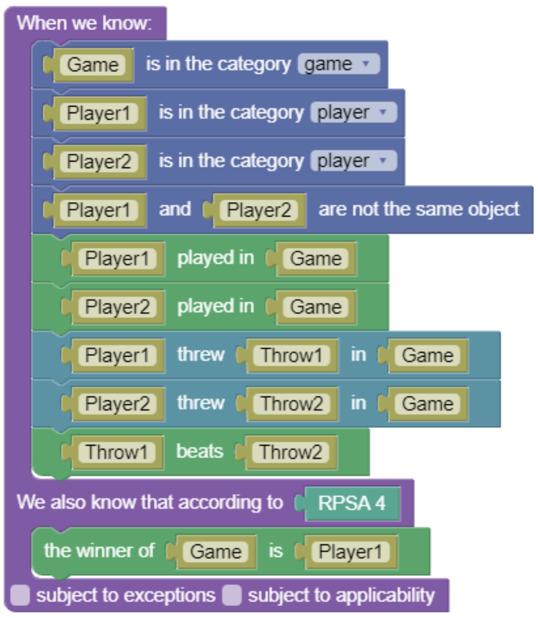


```
When we know:
  Person is in the category person
  Person was born on DOB
We also know that according to LA 1
  From DOB
    Person is alive
subject to exceptions  subject to applicability
```

Figure 2: An example of event reasoning in Blawx

legal concepts much easier, as well as improving the natural language generated by Blawx in those models.

It is anticipated that in future versions of Blawx, Attributes (both unary and binary) may be deprecated in favour of ”relationships” of between 1 and 10 terms. The separate identification of ”Categories” remains valuable for identifying valid responses in user interfaces.

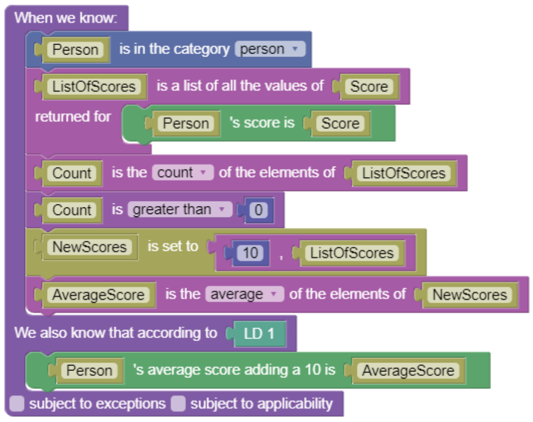


```
When we know:
  Game is in the category game
  Player1 is in the category player
  Player2 is in the category player
  Player1 and Player2 are not the same object
  Player1 played in Game
  Player2 played in Game
  Player1 threw Throw1 in Game
  Player2 threw Throw2 in Game
  Throw1 beats Throw2
We also know that according to RPSA 4
  the winner of Game is Player1
subject to exceptions  subject to applicability
```

Figure 3: A rule using ternary ”relationship” blocks

2.6. Lists and Aggregates

With the addition of access inside s(CASP) to the `forall` predicate, it became possible to give the Blawx user access to the collection of bindings into list structures, and aggregation over those list structures. As such list and aggregation elements have been added to the language. The impure nature of the `forall` predicate means that rules which use aggregation cannot be negated over, but the value of having access to numerical aggregates has thus far proved more important in the usability of the tool.



```
When we know:
  Person is in the category person
  ListOfScores is a list of all the values of Score
  returned for
    Person 's score is Score
  Count is the count of the elements of ListOfScores
  Count is greater than 0
  NewScores is set to 10 , ListOfScores
  AverageScore is the average of the elements of NewScores
We also know that according to LD 1
  Person 's average score adding a 10 is AverageScore
subject to exceptions  subject to applicability
```

Figure 4: A rule using lists and aggregation

3. Building Blawx's Interface Features

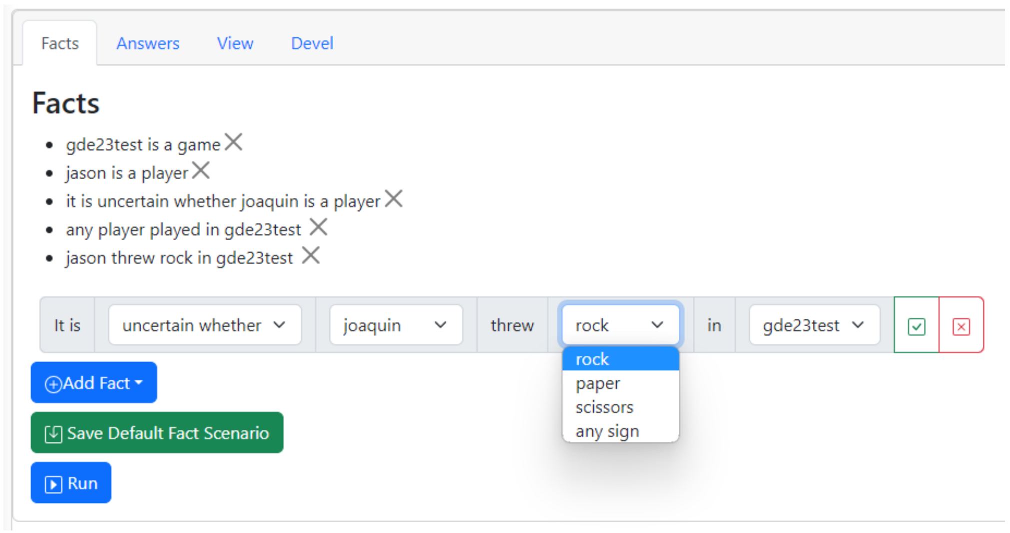
In addition to significant changes to the Blawx language, we have made two notable changes to the user interface: the addition of the scenario editor, a revised method of natural langauge generation to improve explanations, and integration with LLM for explanation summaries.

3.1. Scenario Editor

A previous attempt at generating a friendly user-interface for generating and testing the outcomes of fact scenarios used a chatbot style interface, but user testing revealed that the interface was too slow and inflexible for practical use in exploring the meaning of Blawx encodings. That work was abandoned in favour of a "scenario editor" interface, which has gone through a number of revisions to reach its current state.

The scenario editor is aimed at subject matter experts who are interested in validating that the behaviour of an encoding of a rule is consistent with their own interpretation of the rule, but who want to explore the behaviour by generating fact scenarios and reviewing explanations, as opposed to reviewing code.

The scenario editor takes advantage of the ontological information kept in the predicates defined by the user, offering only those suggestions for inputs that are of the correct category in each parameter of the predicate. It also takes advantage of s(CASP)'s ability to allow the user to specify known falsehoods, and to specify statements that s(CASP) should treat as abducible. It also allows the user to create ungrounded fact statements, which use the types for the parameters to generate natural langauge representations such as "any person".



Facts

- gde23test is a game
- jason is a player
- it is uncertain whether joaquin is a player
- any player played in gde23test
- jason threw rock in gde23test

It is | uncertain whether | joaquin | threw | rock | in | gde23test

Figure 5: The scenario editor user interface

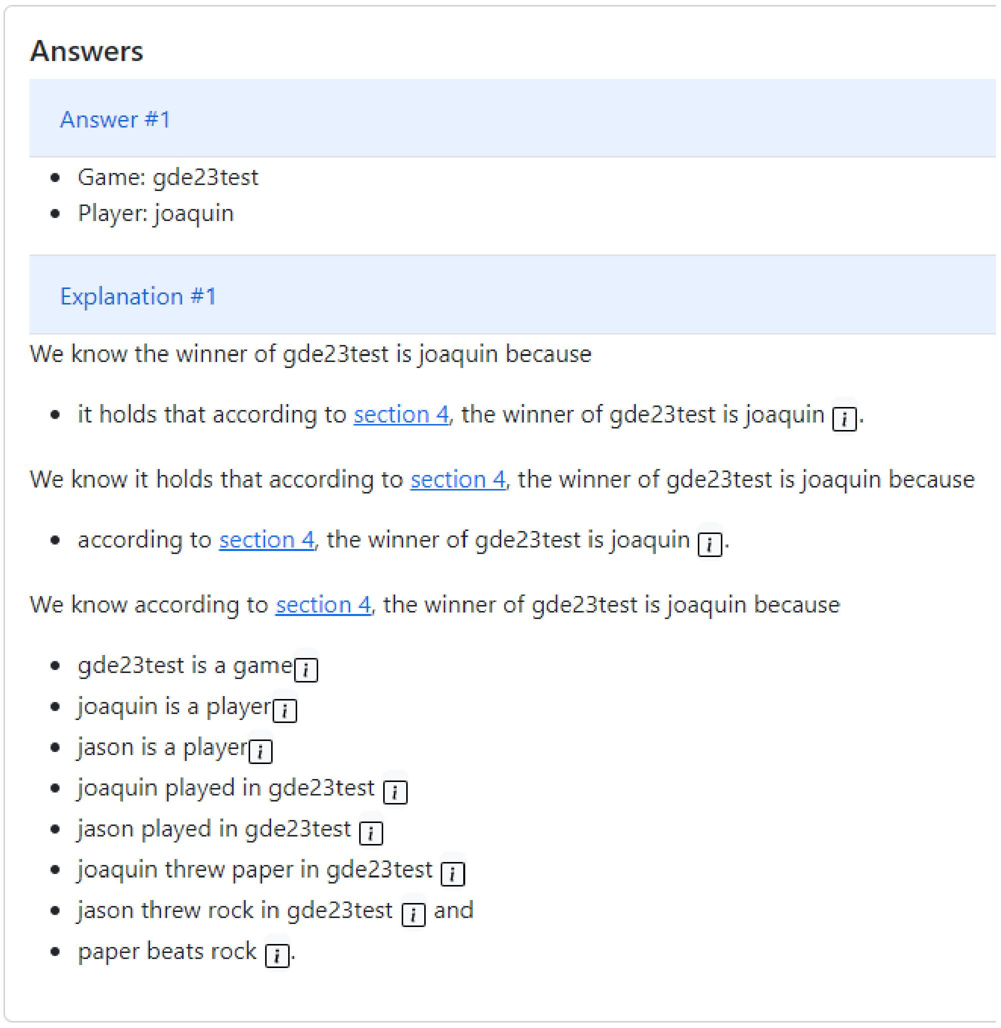
3.2. Natural Language Generation for Explanations

In addition to improved interface for input, the scenario editor uses a revised method of generating natural langauge explanations from the answers received from the s(CASP) reasoner. Instead of using the NLG provided by the s(CASP) reasoner, it uses the NLG information provided by the user about their defined predicates, plus knowledge of the flattening of higher-order predicates used by s(CASP) and Blawx, to provide a NLG system in which nested terms are recursively translated into natural langauge, so that `-according_to(section1,plays,bob,baseball)` will be converted into "it is not the case that according to section 1 bob plays baseball".

The secnario editor also provides the natural language explanation not as a single tree, but as a series of linked paragraphs, allowing the user more control over which portions of the explanation they want to examine in the user interface, and ensuring that each portion of the explanation that is added is a natural language sentence.

3.3. Integration with LLM for Explanation Summarization

Most recently for interface changes, a configuration option has been created that allows the user of Blawx to specify an OpenAI API key. If they do so, the scenario editor will take the text of each explanation generated, and send it to a ChatGPT3.5 endpoint for summarization, and the result will be displayed to the user in place of the explanation, with the option to expand the original explanation for context.


Answers

Answer #1

- Game: gde23test
- Player: joaquin

Explanation #1

We know the winner of gde23test is joaquin because

- it holds that according to section 4, the winner of gde23test is joaquin i.

We know it holds that according to section 4, the winner of gde23test is joaquin because

- according to section 4, the winner of gde23test is joaquin i.

We know according to section 4, the winner of gde23test is joaquin because

- gde23test is a game i
- joaquin is a player i
- jason is a player i
- joaquin played in gde23test i
- jason played in gde23test i
- joaquin threw paper in gde23test i
- jason threw rock in gde23test i and
- paper beats rock i.

Figure 6: An answer display in scenario editor

4. Usage

Blawx began seeing use inside the Canada School of Public Service in roughly August of 2022. The school has been involved in Rules as Code experiments for several years, but experienced some degree of frustration with the large amount of time and effort required by a large number of people in order to generate encodings in their previous approaches.

A legally-trained staff person with CSPS was trained on how to use Blawx for legal knowledge representation, and has been primarily responsible for generating code in several experimental projects. Anecdotally, the experience has indicated that the cost and time required to generate encodings of similar complexity has been significantly reduced. That user has begun training users in other departments to do the same encoding work.

On the basis of these experiments, knowledge and interest in the Rules as Code approach has reached several critical departments inside the Government of Canada responsible for the drafting and maintenance of statutory and regulatory text. It is anticipated that experiments of this type will continue to become more common.

The purpose of those projects have varied from generating a web application capable of advising regulated entities on what procedural steps are required in certain situations based on encodings of multiple statutes and regulations; to static analysis of proposed regulatory text against typical fact scenarios to determine whether a regulatory enactment is valuable in that context.

We anticipate that these experiments represent world-firsts in a number of categories, and hope to publish in greater detail on them in the future.

5. Challenges

Our experiences over the last year have revealed challenges in using Blawx that we did not anticipate. Primary among them are misapprehensions of the nature of legislation and regulation, and the purpose of statutory knowledge representation, and corresponding misapprehensions of the capabilities of a tool which is limited to an encoding of statutory knowledge.

Both subject matter experts and software developers seem to occasionally be under the impression that statutes contain procedural information about the ”order” in which factors are considered in determining whether or not a legal conclusion can be reached. As such, they presume this information can be used to guide the order of questions that appear in an expert system built on that encoding. This may be a result of an apprehension that the ”numbering” in laws relates to a sequence, as opposed to mere identification.

What Blawx can currently do is advise, using s(CASP)’s abducibility mechanisms, which additional facts would be relevant to finding a concrete answer to the query sent to the Blawx API. But it cannot use the encoding of the law itself to provide any rational reason to place one of those questions before or after another in an interview.

Similarly, there has been difficulty in explaining to users of encodings that the natural language generation capabilities of Blawx are designed to facilitate encoding and validation, and not for the generation of user-facing expert systems in which the use of legal terminology is likely to be counter-productive.

Among the problems we could possibly have, these are unobjectionable. The users of blawx are satisfied with it, and want more. That is perceived as a sign of success.

There have also been some challenges implementing the new event reasoning system recently added, and it is anticipated that this will require modification going forward to interact properly with other of Blawx’s reasoning features, such as defeasibility.

6. Future Work

Development priorities continue to be set primarily by the work of the Canada School of Public Service in it’s Rules as Code experimentation with various departments of the Canadian federal government. We anticipate exploring the computational efficiency of Blawx encodings with larger scale projects, and working to improve the ease of integration with other applications and add features specifically for expert system development.

Preliminary work has begun on using LLM chat tools as a user interface to collect ground, unground, positive, negative, and unknown facts and submit them to Blawx in the appropriate format. These early experiments have been very promising.

We have also done preliminary experiments into the ability of LLM tools such as ChatGPT to generate proposed Blawx encodings of statutory text. These early experiments have had mixed results, but are promising enough to warrant further investigation.

We are also actively working on improving the user interface for specifying legislative text from the markdown-like syntax currently used to a WYSIWYG statutory XML editor that generates a sub-schema of AkomaNtoso[6].

7. Conclusion

Significant improvements have been made to the reasoning and interface of Blawx over the course of the last year, all allowing Blawx users to take better advantage of the reasoning features provided by s(CASP). Blawx has also seen considerable use inside the Government of Canada, with considerable success in experiments in both regulatory drafting and service automation. Work remains to determine whether the encodings generated by Blawx are (or can be made) computationally efficient for real-world use cases. We also need to continue to explore whether advances in LLM technology can be leveraged to improve the user interface and the cost of knowledge acquisition.

I look forward to reporting on progress in those directions in future GDE workshops.

Acknowledgments

My thanks to Chris Allison, Chief Data Officer for the Public Health Agency of Canada, who has been immune to skepticism about the value of Rules as Code. Thank you also to everyone in the Public Sector Experimentation group at the Canada School of Public Service and their various projects for their enthusiasm and collaboration in this work. Far more has been accomplished in the last year than I would have thought possible, and it is to their credit.

In particular, thanks again to the first and best user of Blawx, Marty Perron of CSPS, without whose feedback I would be doomed.

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

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