MIT Computational Law Report

Blawx: Rules as Code Demonstration

Jason Morris

Published on: Aug 14, 2020

0.0 What is Rules as Code?

> *Rules as code is an interdisciplinary, technology-enabled approach to the development of rules that is designed to improve policy outcomes. Rules as Code features multidisciplinary teams of lawyers, policy experts, computer programmers, and others who draft rules in natural language and code at the same time.*

Since the 1980s, experts in law and computing science have sought to automate legal reasoning by digitizing the meaning of laws.[1] In the last several years this idea has been renewed in a movement known as "Rules as Code." Governments including New Zealand, Australia, Canada, and France have been working to discover whether and how the delivery of public services can be enhanced by digitizing the rules to which those services must adhere.[2] An international conversation is growing. The article provides an introduction to Rules as Code, its anticipated benefits, and its potential impact on the legal services industry. Then, the article introduces Blawx, an open source, user-friendly tool for Rules as Code, and continues with some example demonstrations that use Blawx to provide legal reasoning capabilities for a web application.

0.1 How did Rules as Code Emerge?

The recent discussion of Rules as Code emerged from work done in New Zealand's Service Innovation Lab, led by Pia Andrews.[3] The motivation of their experiments was to find a way of integrating various Government of New Zealand digital services. Their hypothesis was that laws and regulations are typically difficult to automate in consistent and accurate ways, making them more difficult to integrate, in part because they are not drafted with computers as an intended audience. To test this hypothesis, Andrews' team ran experiments that brought computer programmers into mock legislative drafting processes. The participants, including programmers, subject matter experts, service delivery experts, and facilitators, went through a process to agree on an abstract design for the law. The legislation was then drafted in both natural language and in computer language, a process called "co-drafting."

Co-drafting reveals ambiguities in natural language drafting. This is because drafting in code requires adherence to stricter semantics than natural language. While intentional vagueness can still be expressed, it is more difficult to unintentionally rely on assumptions. For example, we may presume that everyone knows what a "week" is. But the word "week" hides a lot of ambiguity that becomes clear when you try to express it in code. You will need to answer questions like "what day does a week start on?", and "can a week include both December 31 and January 1 of the next year?" Unnecessary and unintentional vagueness in the drafting of rules are eliminated as a result.[4]

0.2 How does Rules as Code Benefit the Policy Process

The results of the co-drafting experiments suggest significant potential benefits for public policy. From the perspective of public service delivery experts, the resulting law was much easier to automate. From the perspective of the legislative drafters, the resulting law was also better drafted.[5] In addition to representing the rule itself, the version of the rules that is written in computer code also can also serve as a first step toward building models that predict how a rule could work, tuning a rule in order to achieve better outcomes, and adapting a rule based on how it is performing in the real world.[6] This new ability to test the effects of a rule against the intent of a rule provides a new method by which to cover the error space between legislative design and legislative drafting, and is similar to the computing science practice of test-driven development. But the encoded law could also be used to predict its policy effects, covering the space between policy intent and legislative design.

0.3 What are the Benefits to Automating Legal Services?

Some of the most astonishing possible advantages of Rules as Code come from using the resulting encoded legislation to create automated legal services, either in government or in the private sector.

0.3.1 Increased Quality of Legal Applications

Encoding legal logic separately from other aspects of software products is expected to improve the quality of applications that automate legal services. As noted by Lauritsen and Steenhuis, substantive legal quality assurance is difficult with most legal software.[7] Often, the legal logic, application logic, and interface logic are intermingled.[8]

Consider a legal application designed to help someone draft a will. Presume a person who is under the age of 18 is not entitled to draft a will. An example of what typically happens now in the development of legal automation software is that this legal requirement is expressed in a section of code that decides what screen to display to the user. In addition to legal requirements such as age, that section of code will also deal with application concerns, such as whether it needs to ask about witnesses, and interface concerns, such as whether the software needs to ask the user's age or if this information has already been submitted somewhere else in the application .

As a result, asking a lawyer, even one who is familiar with the programming language used, to analyze source code of an application to determine whether the legal requirements have been correctly implemented, is a difficult task. That sort of quality assurance can only be performed through testing the application by using it. That sort of testing addresses only one fact scenario at a time, and is slow and expensive.

Rules as Code supports quality assurance by separating the code implementing the legal rules from the rest of the software. In doing this, a rule can be read, analyzed, and tested separately using simpler

and more efficient quality assurance techniques. This makes it easier for a subject matter expert to review and test that code by itself, to confirm that the legal rules the software implements are correct.

0.3.2 Rapid Legal Application Development

Once the high-quality encoding of the legislation is developed, the task of building applications is greatly simplified.

> *If we use Rules as Code, it becomes possible to write a brand-new web application that provides a user-friendly one-question-at-a-time interview and provides a substantive answer to a complicated legal question along with a human-readable explanation of how that answer was reached,* just by representing the question in code.

The resulting application would be little more than a prototype, but can be built in a fraction of the time that it would usually take.

0.3.3 Simplified Maintenance

One of the big challenges with ensuring quality in legal applications is maintaining legal accuracy over time as things change. Maintenance of the legal reasoning in a legal service application becomes easier with Rules as Code, particularly if the technology used for the encoding shares a paradigm and structure with the way the law was drafted in natural language. When (not if) a part of the rules change, only the corresponding part of the encoding needs to change to match. The applications that use that encoding do not need to change.Like an update to the operating system of your computer, all of the applications relying on the rule's encoding are updated for the new environment and in real time.

1.0 Why Should Lawyers Care?

The transformation from rules as documents to Rules as Code is one that has many implications for legal service delivery and lawyers in particular. Critically, Rules as Code supports a realistic approach to improve access to justice by automating legal services; however, Rules as Code also promises to transform the practice of law in more fundamental and positive ways.

1.1 Access to Justice

Lawyers have demonstrated that we, alone, are unable to solve the access to justice crisis. The Legal Services Corporation estimates that more than 80% of the impoverished, and more than 50% of the middle class, lack access to legal services in the United States.[9] This problem is perpetuated by protectionist regulatory regimes from which lawyers personally benefit. As a profession, we are morally obliged to do better.

Economics teaches that increased efficiency in production can help with a market facing supply shortage. In the legal services market, automation of legal services could go some way to serving those who we do not currently serve.

Imagine the government has adopted Rules as Code for certain pieces of legislation. They draft a law and release the open source encoded version of it at the same time it is signed into law. The encoding is in a standards-compliant format that can be used in free, open source software tools.

Now imagine clients need help applying for relief from eviction. The encoding of the landlord-tenant law can be used as the source code of an application to help tenants make applications for relief. The application will collect relevant information, and provide answers, with explanation for how those answers were obtained, and references to the relevant legislation. And it can be built quickly, and easily, and with confidence that it is consistent with the law. And if the law changes, the application may not need to be changed at all in order to continue to work perfectly.

Rules as Code enhances the ability to automate legal services, and provides a viable strategy for significantly mitigating the access to justice crisis.

1.2 The Future of Law

The future of law will be digital. The economic opportunity alone ensures that. What that digital future looks like for lawyers is difficult to predict. But we can learn from the history of the profession of accounting.

Forty years ago, accountants did not use computers to do the math involved in making financial projections. They could have, but the cost required to learn the programming language needed to make these financial projections was too high.

Within 25 years of VisiCalc's release in the 1970s, the use of electronic spreadsheets became a basic competence required of the accounting profession. So how did digitization change the accounting profession? The use of technology empowered accountants to do their work faster and more accurately, all while handling increasingly complex tasks. Consequently, demand for their services increased. Accounting software and spreadsheets did not automate accountants out of existence. Instead, these tools transformed accountants into more efficient and capable service providers.

The practice of law has been digitized to an extent. Where we used to deal with paper, we now (mostly) deal with electronic documents. The use of word processors is ubiquitous, and the use of automated document generation is growing. But lawyers are not document experts. Our expertise is rules. The digitization of rules in the legal profession is barely visible, with the growing interest in smart contracts a notable exception.

When the full digitization of rules occurs, we can expect the same effects on the legal profession as were seen in the profession of accounting. Lawyers will work faster, more accurately, and will be able to deal with more complicated tasks. Services that were previously only possible to obtain from a lawyer will be possible to obtain, at low cost, from a website. The costs of services generally will go down, and their value will go up, increasing the demand for legal services, and increasing the size of the profession.

1.3 Is Rules as Code About Law?

Yes, and no. The Rules as Code movement comes from the public sector. In the public sector, legislation, regulation, and policy are the guideposts. Rules as Code certainly has massive implications for lawyers if the project comes to fruition. So Rules as Code does have a lot of implications on the way that law will be practiced in the future.

At the most basic level, the Rules as Code argument is one that applies to all rules, not only laws. It could be used to great effect for any rule for which greater clarity and ease of automation and testing is valuable. The principles of Rules as Code could - and I argue *should* - be applied to contracts, organizational policies, or even the rules of your favorite board game.

1.4 Does this Mean Lawyers Should Code?

This is the wrong question. The more useful question is this: Is there a category of tool for a given task so obviously worth learning that every lawyer ought to be expert in it?

For accountants, and the task of financial modeling, the answer is obviously spreadsheets. For both lawyers and accountants, and the task of document creation, the answer is word processors. For lawyers, and the task of automating legal reasoning, there is no good answer. At present, the tools available are either too difficult to learn or designed only to solve a small part of the problem.

Have accountants learned to code? In one sense, yes. Electronic spreadsheets are a domain-specific declarative programming tool for math. In another sense, no. Using Excel doesn't *feel* like coding. But whether being an expert Excel user qualifies as being a "coder" is irrelevant.

Programming languages and applications like Excel are automation tools. Automation tools have a cost involved in learning them, and a benefit associated with using them. When the balance is right, people will adopt them. For accountants, the balance for automating financial modeling wasn't right until spreadsheets reduced the cost of learning.

We know the same is true for lawyers. Take, for example, the ubiquitous task of searching for case law. If you are a lawyer you may recognize "liab* /p zoo*" as a query that asks a research tool to find a document with a word starting with "liab" that occurs in the same paragraph as a word starting with

"zoo". If you can read that query, congratulations: You are a coder. Because that query is written in a domain-specific programming language for legal information retrieval.

People who say lawyers should learn to code believe lawyers should be able to do what it is possible to do when you have learned a programming language. People who say lawyers should not learn to code believe that programming in general-purpose programming languages is too complicated a skillset to add to what lawyers are already expected to know. Everyone is right.

As soon as "coding" is worth the time it takes to learn and master a new tool, every lawyer will become a coder, whether that's the name they use for it, or not.

The onus is not on the lawyers to become programmers. The onus is on the programmers to provide tools that lawyers can realistically use. But the situation is improving, and rapidly. While expert systems have been around since the 1980s, they have not had the same sort of adoption as spreadsheets.[10] So far, the options for legal reasoning are lacking. Most are still too difficult to learn or focused on too small a part of the problem.

Docassemble has made amazing strides in the development of an open source architecture for legal expert systems, and teams like Community Lawyer and Documate have been creating even more accessible versions of that tool.

The most important function of this paper is to help the reader appreciate how much further it is possible to go. Blawx was created as a tool to help demonstrate that potential.

2.0 What is Blawx?

In this section, I introduce an open source project that I have developed called Blawx and explore how it works.

My LLM thesis focused on declarative logic programming tools. This category of programming tools is relatively unpopular, but has long been recognized as particularly useful for encoding legal rules.[11] In my thesis, I wanted to learn what features these tools would require in order to have the greatest possible impact on the delivery of legal services.[12] The conclusion I reached is that we need tools that are open source, purpose-built, and above all easy to use.

I wrote Blawx to demonstrate the potential of the declarative logic approach for Rules as Code. Blawx is a free, open source web application for encoding rules, and for answering questions using those rules. It may be used through a user interface, like a spreadsheet, or it can be connected to another app that runs on the web over API, like a database. Technically, it is a combination of Flora-2, aka ErgoLite, an open source declarative logic programming language, and Blockly, a Google platform for

developing visual coding environments. Blockly is a descendent of Scratch, a tool designed to teach kids how to code.

One of the core ideas of the Blockly interface is that the parts of the programming language are represented as pieces of a puzzle. If the user attempts to place a piece of code where it does not belong, the piece won't "fit". If there is something missing, you will see a "hole". Different pieces that are related to one another appear in similar colours. In this way, the interface provides information that is not visible when programming in a text interface. If you make a mistake, you get visual feedback immediately, and learning to write the language is therefore easier.

As a strong believer in the principle of "release early, release often, and release before it's ready," Blawx is alpha software. It is buggy. If you look under the hood, it is held together with duct tape and string. All that being said, Blawx works. And it demonstrates what the future of drafting Rules as Code might look like.

At the time of writing, it is in version 0.2.3-alpha. A live demonstration is available at app.blawx.com/blawx.html, and the source code is available on GitHub if you would like to try running it locally or contribute to the code.

Blawx has a combination of features that are not, to my knowledge, available elsewhere:

- Blawx uses a visual programming environment designed at MIT for effectiveness of introducing non-programmers to programming.
- Blawx uses a fully open source declarative logic programming language, Flora-2.
- Blawx offers the ability to code rules as rules, not as procedures or functions or objects, increasing the one-to-one relationship between the structure of the rules and the code.
- Blawx offers the ability to code rules that disagree with one another, increasing the one-to-one relationship between the structure of the rules and the code.
- Blawx, itself, is free, open source software that has been released under the MIT license.
- Blawx features a module that allows users to create guided interviews in Docassemble that use rule encodings from Blawx.

2.1 What is it Like to Code in Blawx?

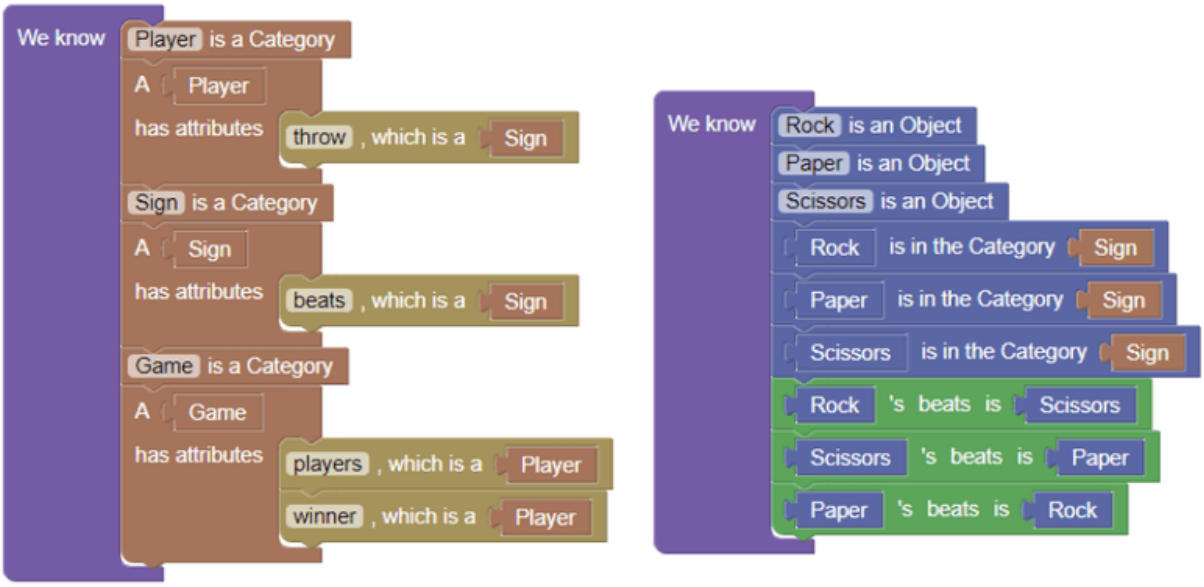
The programming process in Blawx consists of a few phases. First, you set out the ontology: the vocabulary of categories of objects, their properties, and the relationships these objects have with one another. Second, you write the rules using that vocabulary. Finally, you provide facts and ask questions of the system.

2.1.1 Defining an Ontology

Most people are familiar with the game Rock, Paper, Scissors. Let's say that you want to encode the rules of Rock, Paper, Scissors in Blawx. If we were co-drafting, the ontology part of the natural language rules might read like this:

> *A game consists of one or more players. One of the players may be the winner of the game. Rock, Paper, and Scissors are the only available signs to throw. Rock beats Scissors; Scissors beats Paper; Paper beats Rock.*

Ontologies are very often implicit in legislation and regulation. Forcing you to be explicit about them is one of the ways that Rules as Code improves drafting. To encode this ontology in Blawx, you would start by using the drag and drop interface to identify the categories and objects that you want to be able to refer to in the rules. Here's what that would look like:



The intent is that by reading this code you would have a clear understanding of its meaning. You can see that some things that are implicit in the natural language version above have been made explicit in these rules, such as the idea that a sign has a relationship called "beats" to other objects that are also signs. That is a simple example of the way in which Rules as Code forces the people drafting rules to be more explicit.

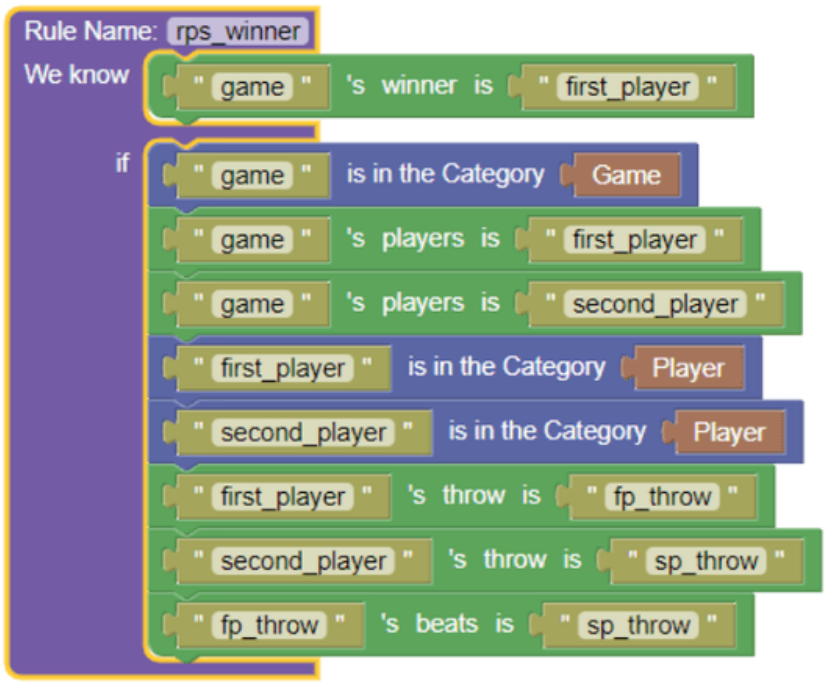
What is unique about the Blawx interface is that it changes as you go. For example, the green block that says "____'s beats is _____" did not exist until the attribute "beats" was created in the "Sign" object. Once you have created the attribute, you do not need to type the name in again (and risk

spelling it wrong); there is a block that can be reused, dragged, and dropped to refer to other objects, categories, or properties.

In this way, Blawx allows users to develop a more technical vocabulary. The items in that vocabulary, including categories, objects, and attributes, become visual elements that can be dragged and dropped, and that can only be placed in the code where they would be syntactically correct. Another way this feature is reinforced is through the way that blocks are stacked vertically to imply a connection with an "and" operation.

2.1.2 Translating the Rules from Text to Blawx

The second phase of the process is expressing the rules in Blawx. If you were drafting the rules to Rock, Paper, Scissors, how would you describe the rule for who won? Assuming there are no ties and only two-player games are played, the rule may be described as, "The winner is the player who throws a sign that beat the sign thrown by the other player." Here is how that rule is created in Blawx:



This is a rule block that has two sections. The top section is the conclusion, and the bottom section is the conditions that must be true for the conclusion to be true. In declarative logic programming languages conclusions are usually stated first, so in its current version Blawx follows that tradition.

The brown blocks are variables. The names are insignificant, except that anywhere the same name is used it refers to the same object. What we want the rule to determine is the winner of the game. But a game has players, a player has a throw, and throws may or may not beat one another. So the conditions

of the rule need to navigate through that ontology to find the pieces of information the rule wants to compare.

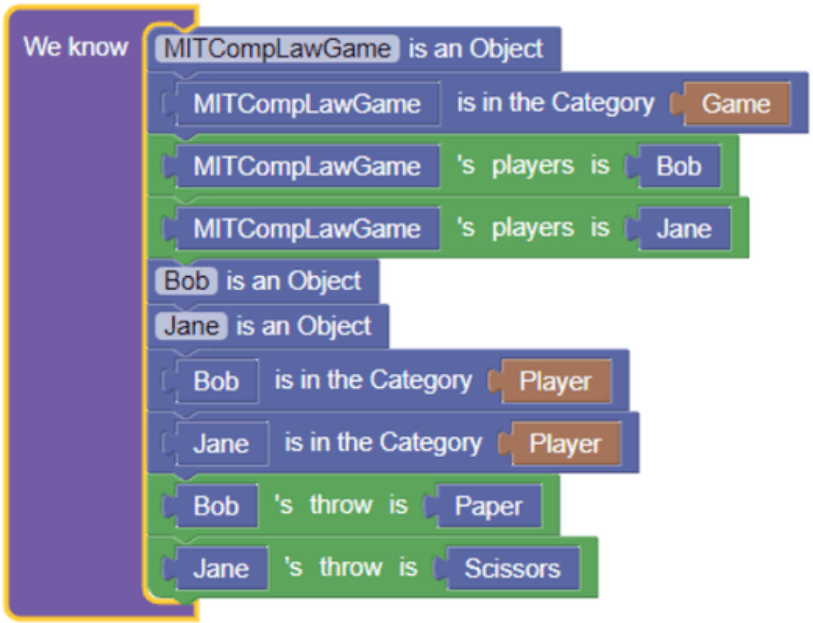
The encoding of the rule can be read aloud as follows:

> We know an object called 'game' has a winner called 'first_player' if:
>
> - The object called game is in the category Game, and
> - The object called game has two players, called 'first_player' and 'second_player', and
> - The players are in the category 'Player,' and
> - The 'first_player' has a throw called 'fp_throw', and
> - The 'second_player' has a throw called 'sp_throw', and
> - 'fp_throw' beats 'sp_throw'

This is a much more verbose description of the rule, but remains relatively readable. All attributes in Blawx are lists, so it is possible to assign more than one value to the same attribute. An attribute called "players," for example, could have more than one value. In a small ruleset, checking for category membership is not necessary. However, in more complicated rulesets, checking for categories can dramatically increase the speed of the code, make troubleshooting easier, or avoid future problems when an attribute name is used by more than one category of object.

2.1.3 Encoding Facts and Queries

Having described the rules, next we give the system facts and pose queries. Below is the encoding for "there is a game between Bob, who throws Paper, and Jane, who throws Scissors."



And here is the encoding of the question “did Jane win the game?”

Using the “Run Blawx Code” command, the system responds:

```
The answer is: Yes
```

In addition to “yes or no” questions, Blawx can ask “search” questions. So if you wanted to ask about the winners of any games, you could encode that query as follows:

When that code is run, the system will reply:

```
The answer is: Yes
game: MITCompLawGame, winner: Jane
```

2.1.4 Additional Examples

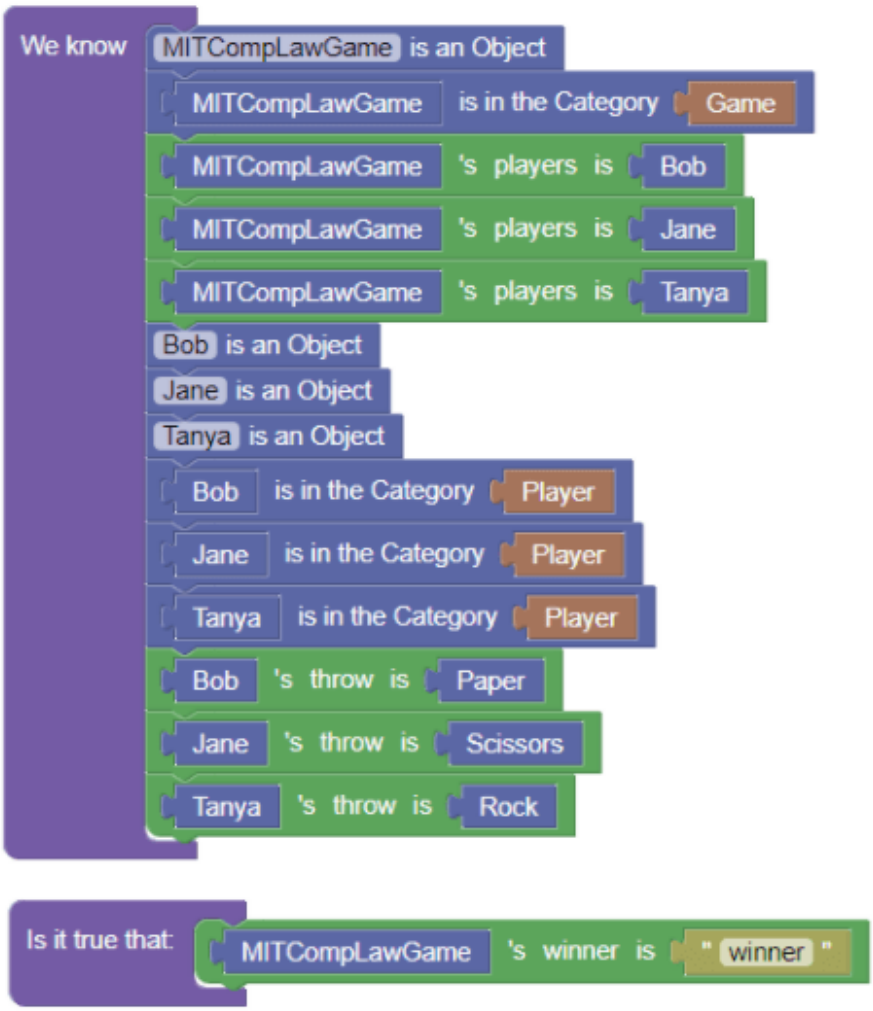
If you would like see a video of the actual user experience, here is a video of me drafting and testing these rules from scratch.

Visit the web version of this article to view interactive content.

Encoding Rock Paper Scissors in Blawx.com

Here is the link for an instance of the live demo of Blawx with the above Rock, Paper, Scissors rules pre-loaded; the demo is also embedded just below. Play with it. See if you can create a game with three players, Bob, Jane, and Tanya. Have Tanya throw Rock. Then ask for a list of winners of the game. What do you think the answer will be, and why?

Visit the web version of this article to view interactive content.




```
The answer is: Yes

winner: Bob
winner: Jane
winner: Tanya
```

All three of them are winners because we have encoded the winner as anyone whose throw beat someone else's throw. In this example, that is true for everyone. It is left as an exercise for the reader to see if you can rewrite the rule to exclude circular ties.

2.2 Using Blawx to Power a Web App

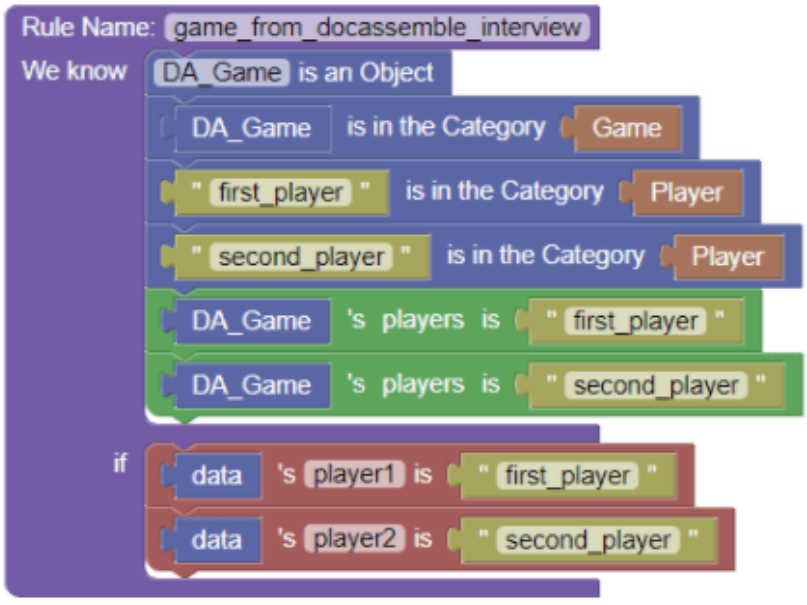
Blawx uses an open API, allowing other applications to be built on top of Blawx over the web. In order to use Blawx code to power a web application, that application must send the "facts" to the Blawx API

in JSON format, and the rules to be used as a Blawx file. The Blawx file must include code to translate the data into the ontology that the rules use, and the query the application wants answered.

Docassemble-blawx, an integration between Docassemble and Blawx, is one example of an integration with Blawx using its API. The process of modifying the Blawx code to allow it to be used in the Docassemble integration is described below.

In order to connect the data produced from the Docassemble interview with Blawx, it is necessary to create a root object called "data."

Then, the data must be mapped to the ontology that has been set up in Blawx. In this case, I created a Docassemble interview that would generate two objects called "player1" and "player2." Because these objects have not been previously defined, the first step is to create a rule that will build a game, categorize it, categorize the "players" and add them to the game. Here is what such a rule looks like:

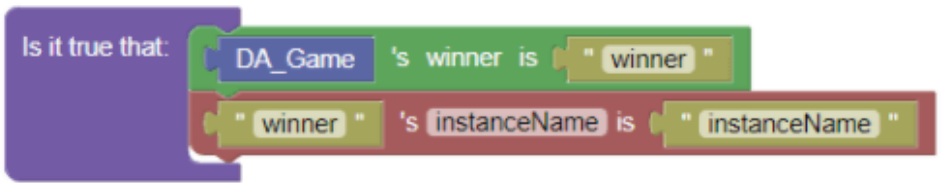


The red blocks in the condition of the rule are Blawx's "data" blocks. They enable us to describe relationships that have not been declared in the code, but which the user knows will exist in data provided from an outside source.

The Docassemble interview records the "throw" as a string of text like "Rock," as opposed to a reference to the object named "Rock" in the Blawx Code. Three rules are needed to translate those text

descriptions into object references. Those rules look like this:

The last step is to formulate a question that will be useful for the Docassemble server. Docassemble uses the “instanceName” property to uniquely identify attributes in its data. That data is then sent to Blawx in the JSON version of the interview data. So, if we find out who the winner of the game is, and then send the instanceName of that winner, Docassemble will know exactly which player we mean. Here is the query we use to generate that data. Note that we are using the red data attribute block to collect instanceName. This is because we know instanceName will exist, but that property has not yet been declared in our code.



The Blawx workspace can now be saved to a file, and that file uploaded to a Docassemble server, and used in an interview. If you would like to see the Blawx code for this app in the live demo, you can click here.

You can also try the interview right here or in the embed below.

Visit the web version of this article to view interactive content.

You will be asked to log in. You can create your own account, or use the email address demo@demo.com, and the password Demo2020. This demonstration interview, including the .blawx file that powers it, is available for download from GithHub at this link.

2.3 Using Blawx to Build Rules for Obtaining a COVID-19 Test

Having seen how it works with a small example, in this section I will demonstrate the use of Blawx to implement a more complicated set of rules, and power an application based on that encoding. This will allow us to see some of the challenges in drafting that Rules as Code can help avoid, and will also demonstrate some of Blawx's more advanced features.

In my home province of Alberta, one of the most salient public rule-sets is the rules for testing and isolation with regard to COVID-19. The government has changed these rules often. Those changes need to be reflected in online screening tools provided for members of the public, and as quickly as possible. Below is a demonstration of how to use a combination of Blawx and Docassemble to accomplish this goal.

2.3.1 The Rules

Here's a summary of the Rules for Obtaining a COVID-19 Test, as I understood them on May 5, 2020:

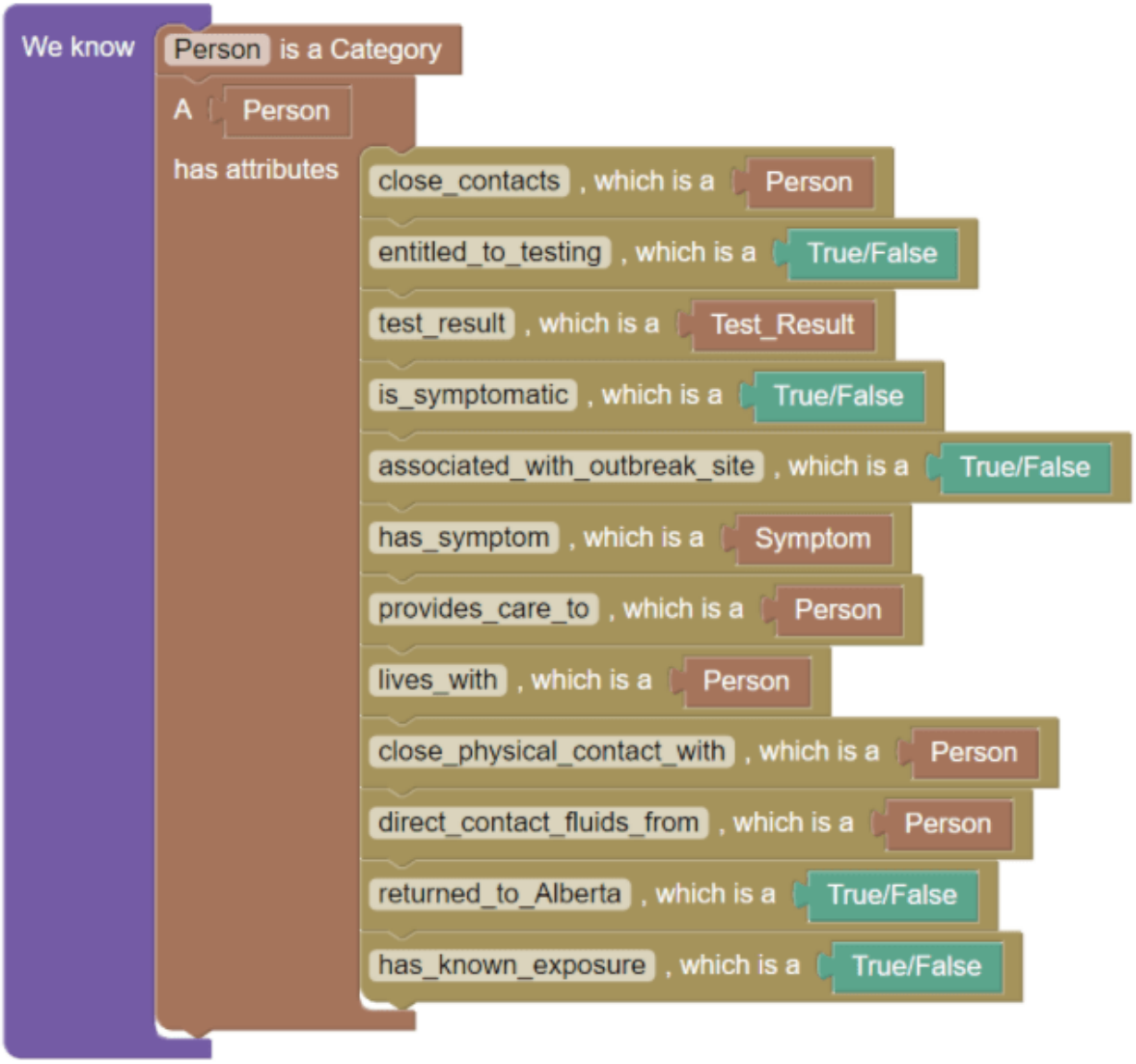
> *You can get tested for Covid-19 if you are symptomatic, you are a close contact of a person who has tested positive, or you are a worker or resident at an outbreak site.*
>
> *You are symptomatic if you have one of the following symptoms:*
>
> - *Fever*
> - *Cough (new cough or worsening chronic cough)*
> - *Shortness of breath or difficulty breathing (new or worsening)*
> - *Runny nose*
> - *Stuffy nose*
> - *Sore throat*
> - *Painful swallowing*
> - *Headache*
> - *Chills*
> - *Muscle or joint aches*

> - *Feeling unwell in general, or new fatigue or severe exhaustion*
> - *Gastrointestinal symptoms (nausea, vomiting, diarrhea or unexplained loss of appetite)*
> - *Loss of sense of smell or taste*
> - *Conjunctivitis, commonly known as pink eye*
>
> *If you have Fever, Cough, Shortness of Breath or difficulty breathing (new or worsening), a runny nose, or a sore throat, you are legally required to isolate for 10 days or until the symptoms resolve, if it takes longer.*
>
> *If you have tested positive for Covid-19, you are required to isolate for 10 days or until the symptoms resolve, if it takes longer.*
>
> *If you have symptoms, have tested negative, but you have known exposure, you are required to isolate for 14 days.*
>
> *If you have symptoms, have tested negative, and you do not have known exposure to the virus, you are not required to isolate.*
>
> *If you are a close contact of a person who has tested positive for Covid-19, you are required to isolate for 14 days, or for 10 days after you get symptoms, or until the symptoms resolve, whichever is longer.*
>
> *If you return to Alberta from elsewhere, you are required to isolate for 14 days, or for 10 days after you get symptoms, or until the symptoms resolve, whichever is longer.*
>
> *You are a close contact of someone if you provide care, live with or have close physical contact without appropriate use of personal protective equipment, or come into direct contact with infectious body fluids from that person.*

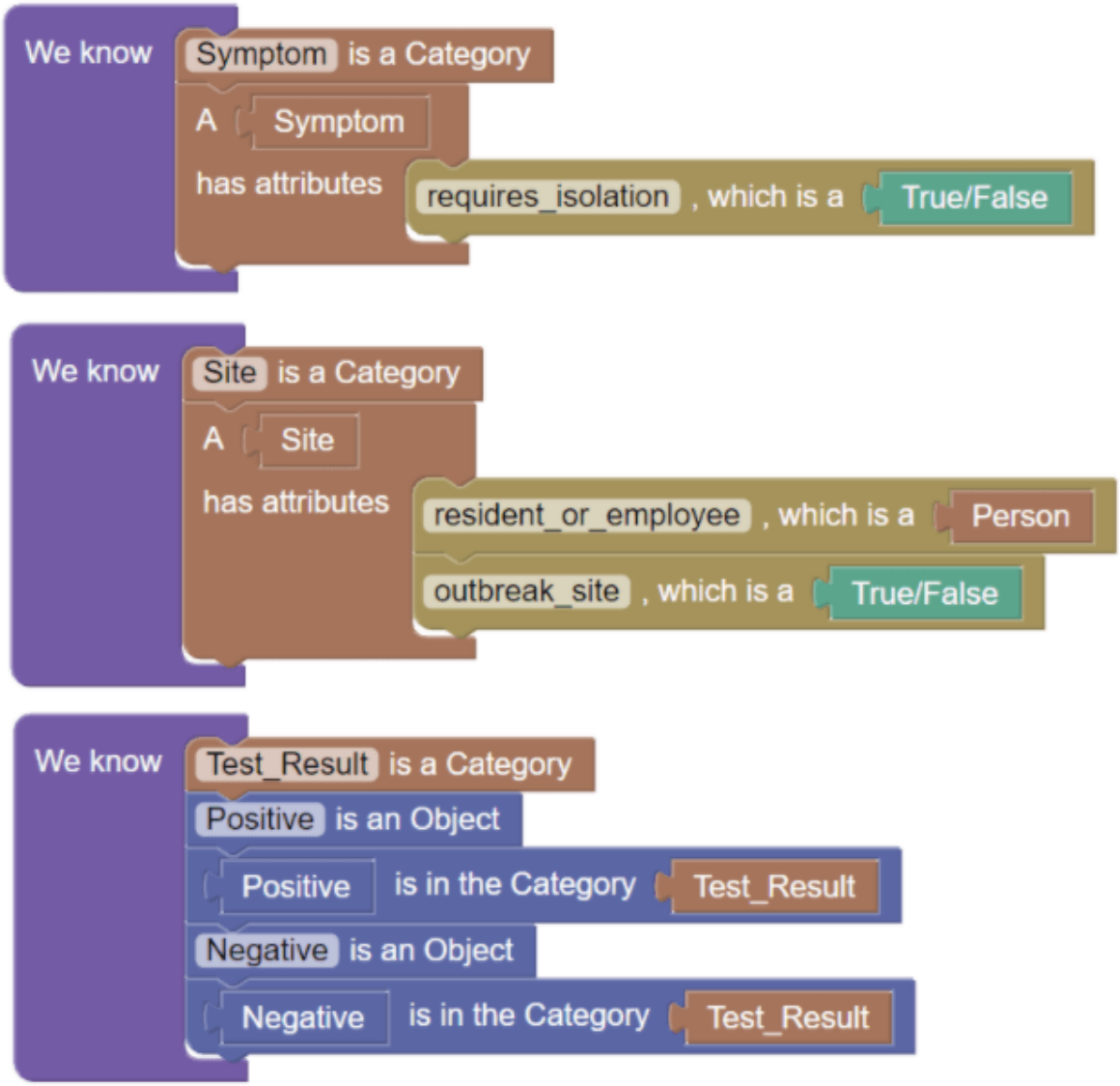
2.3.2 Defining an Ontology

Again, the process of encoding in Blawx typically follows three steps. The first is to set out the ontology that will be used to describe the categories, attributes, and objects about which you want to be able to write rules and queries. In this case, people, symptoms, sites, and test results are all categories that will need to be defined.

Here is what the ontology for a Person category might look like:



The rest of the ontology is defined below.



When we encoded the rules of Rock, Paper, Scissors in the ontology phase, we described the abstract concept of a "Sign," and we were also able to talk about the concrete instances of "Rock," "Paper," and "Scissors" as well. This was because we know those elements will always be relevant.

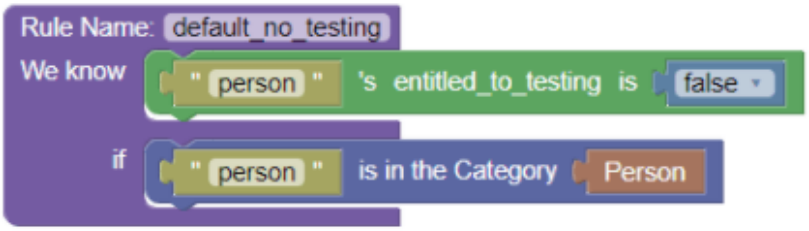
In the same way, in the encoding above, "Positive" and "Negative" are defined as objects in the category "Test Results." The same can be done for the category of Symptoms and the properties of the objects in that category. A portion of that code looks like this:

2.3.3 Translating the Rules from Text to Blawx

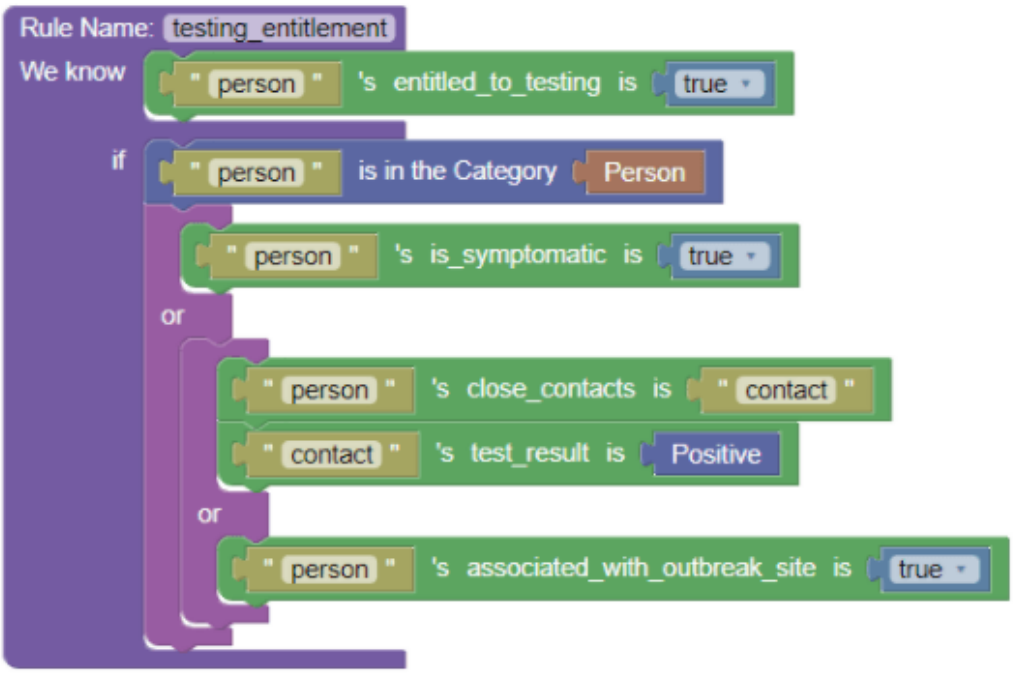
Now that we have the ontology, it is possible to start reconstructing the textual rules into Blawx. Let's begin with the rule that defines whether a person is entitled to testing.

2.3.3.1 Testing Entitlement

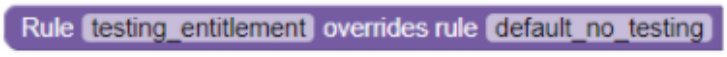
By default, an individual would not be entitled to testing, but the entitlement changes under certain circumstances. So we will use Blawx's ability to make rules that are exceptions to other rules.



We can now make a rule that sets out that a person is entitled to testing if they are symptomatic, if they have close contact with a person who has tested positive, or if they are associated with an outbreak site.



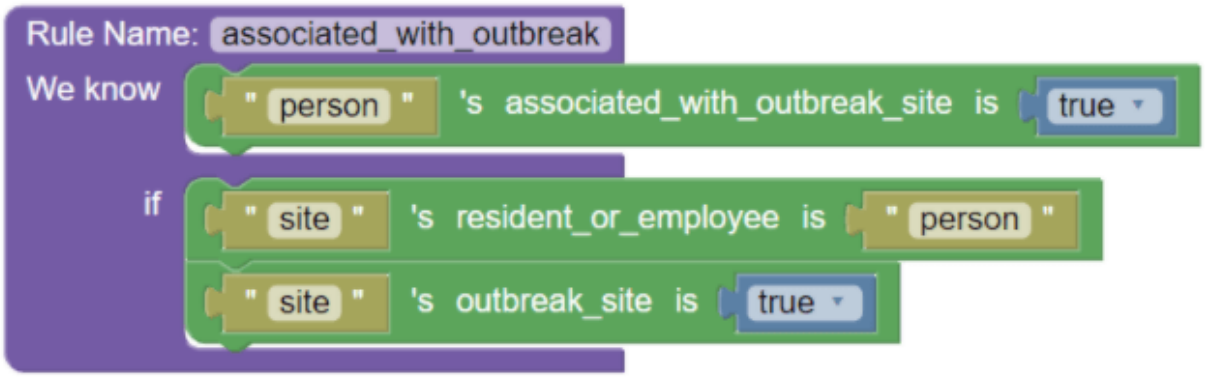
Next, we need to tell Blawx that if the results of these two rules conflict, the "testing_entitlement" rule takes precedence. We do that using the override block.



The above two rules test whether the object "person" is in the category "Person." As discussed, those types of category checks are good practice, but to keep the rules shorter and easier to read, they will be left out for the remainder of the example code.

2.3.3.2 Association with Outbreak Site

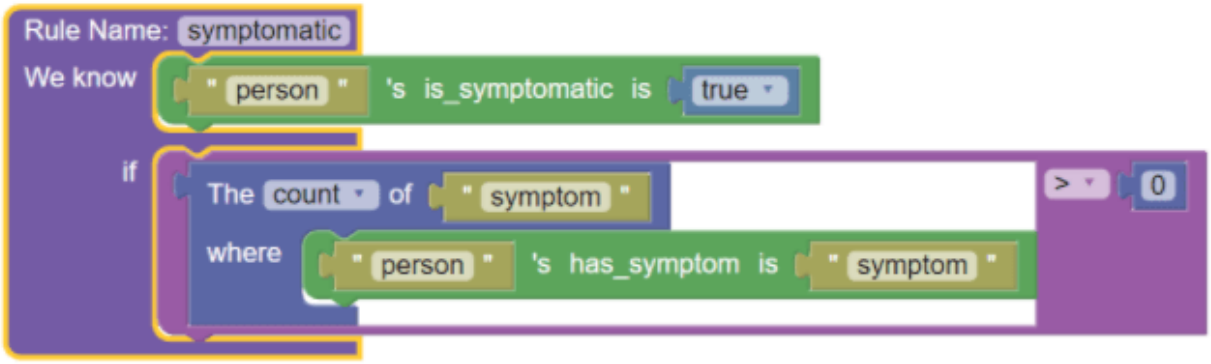
Next we can create a rule for "associated with an outbreak site."



If we were ever going to be interested in proving, with certainty, that a person was not associated with an outbreak site, we could create a default, and use associated_with_outbreak to override that default. But non-association with an outbreak site is not likely to be important, so we can leave the negative result implied.

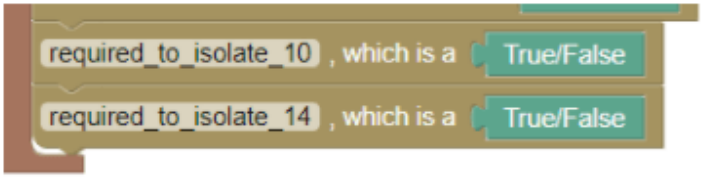
2.3.3.3 Symptomaticity

Now we need a rule that will define if a person is symptomatic. An aggregate function may be used to get a count of all the symptoms a person has, and if the number is greater than zero, they are symptomatic.

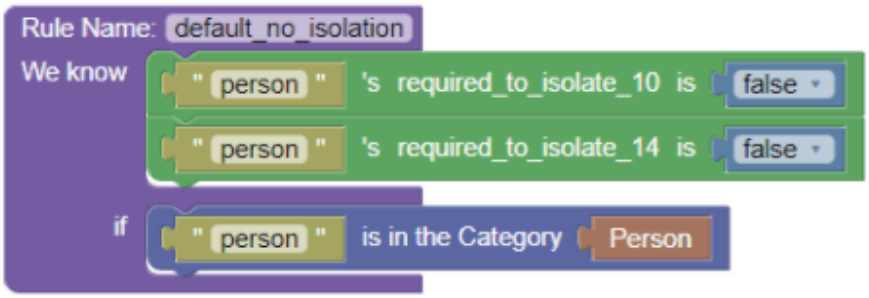


2.3.3.4 Isolation Requirements

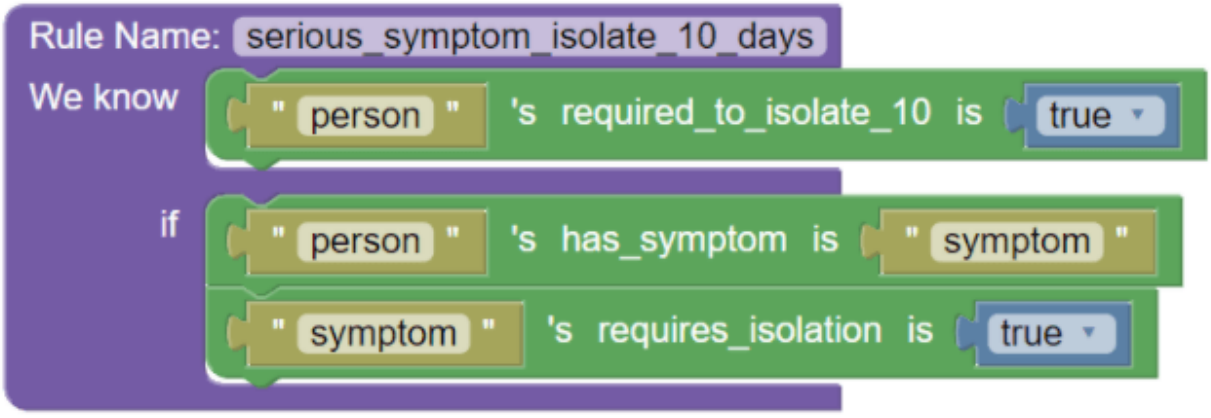
Now we can start writing rules about when a person would be required to isolate. There are two kinds of isolation required: (1) isolation for 10 days (or until symptoms resolve); and (2) isolation for 14 days in the absence of symptoms. This requires us to add two attributes to the “person” object to reflect these two requirements.



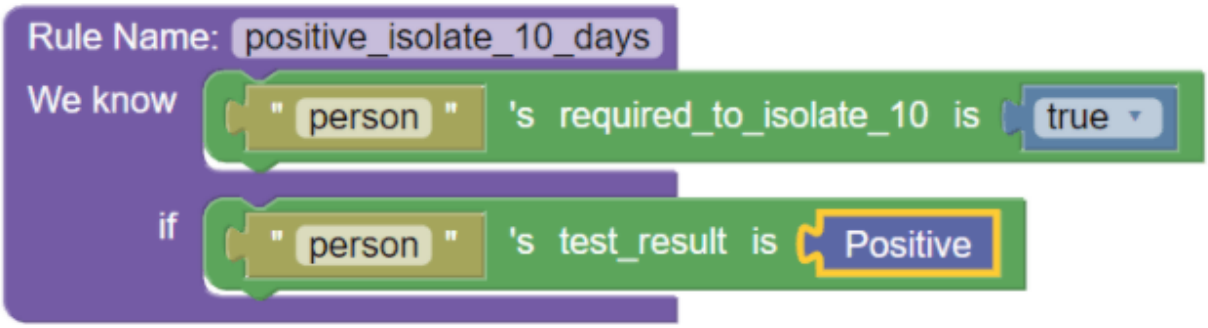
People are going to want to be able to get "no" as an answer to the question of "are you required to isolate." So it is not enough for us to encode the rules for when they are required to isolate; we also need to encode the rules for when people are not required to isolate. This is because Blawx distinguishes between "not knowing that a value is true" and "knowing that a value is false." Here, we can use Blawx's override feature to make "no isolation required" the default, and then override that default with the specific exceptions for when isolation is required.



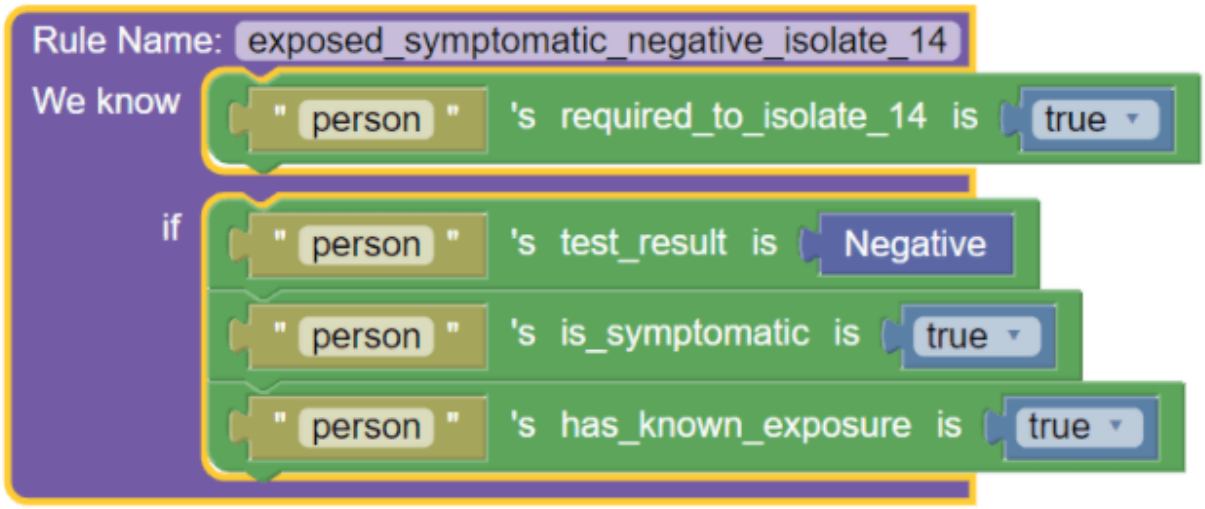
Here is an encoding of the rule that requires isolation for 10 days or until those symptoms resolve. In the original rule, the symptoms are listed. Here, we will instead use the "requires_isolation" attribute of each symptom to determine whether or not it is on the list. This may make it easier to change the list of relevant symptoms without the need to rewrite the rules.



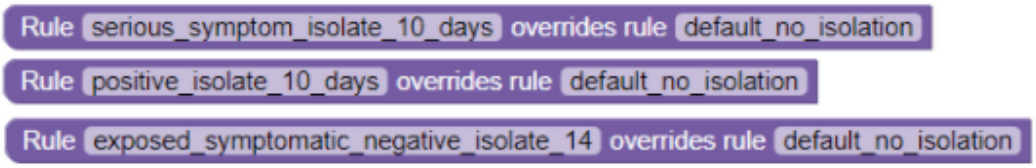
Here is a rule that specifies what must happen if an individual has tested positive for one of the symptoms that requires isolation for 10 days or until symptoms resolve.



Here is an encoding of the rule that exposed symptomatic people who have tested negative still must isolate for 14 days.



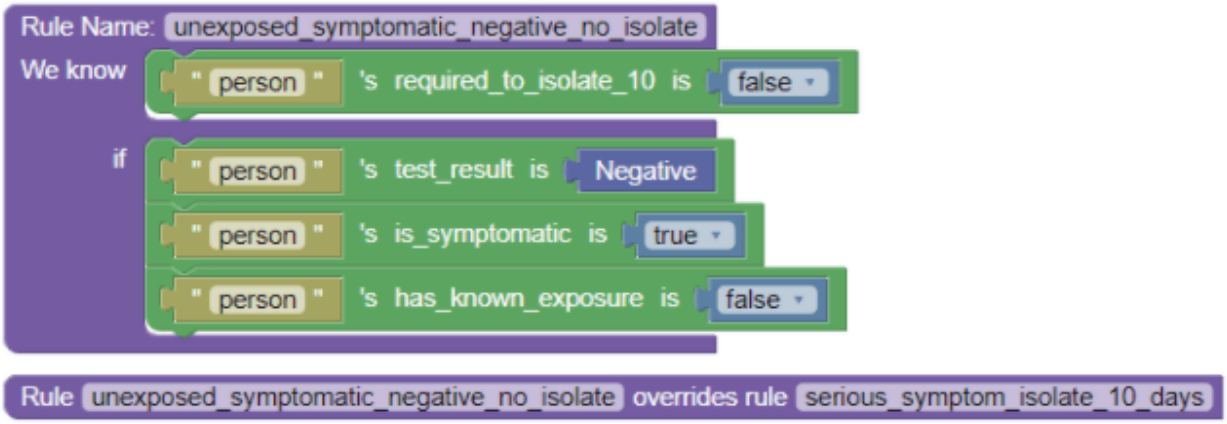
Those are the three circumstances in which you can be required to isolate, so they all override the "default_no_isolation" rule. That is made explicit in the following override blocks:



2.3.3.5 Exceptions to Exceptions

The next of the rules is interesting. It states that if you are symptomatic, but you have no known exposure and have already tested negative, you are not required to isolate. This rule does not contradict the rule about known exposure, because that rule requires the person to have known exposure. As well, it does not contradict the rule about positive tests because it requires a negative test. However, it does contradict the rule about serious symptoms. This is an exception to an

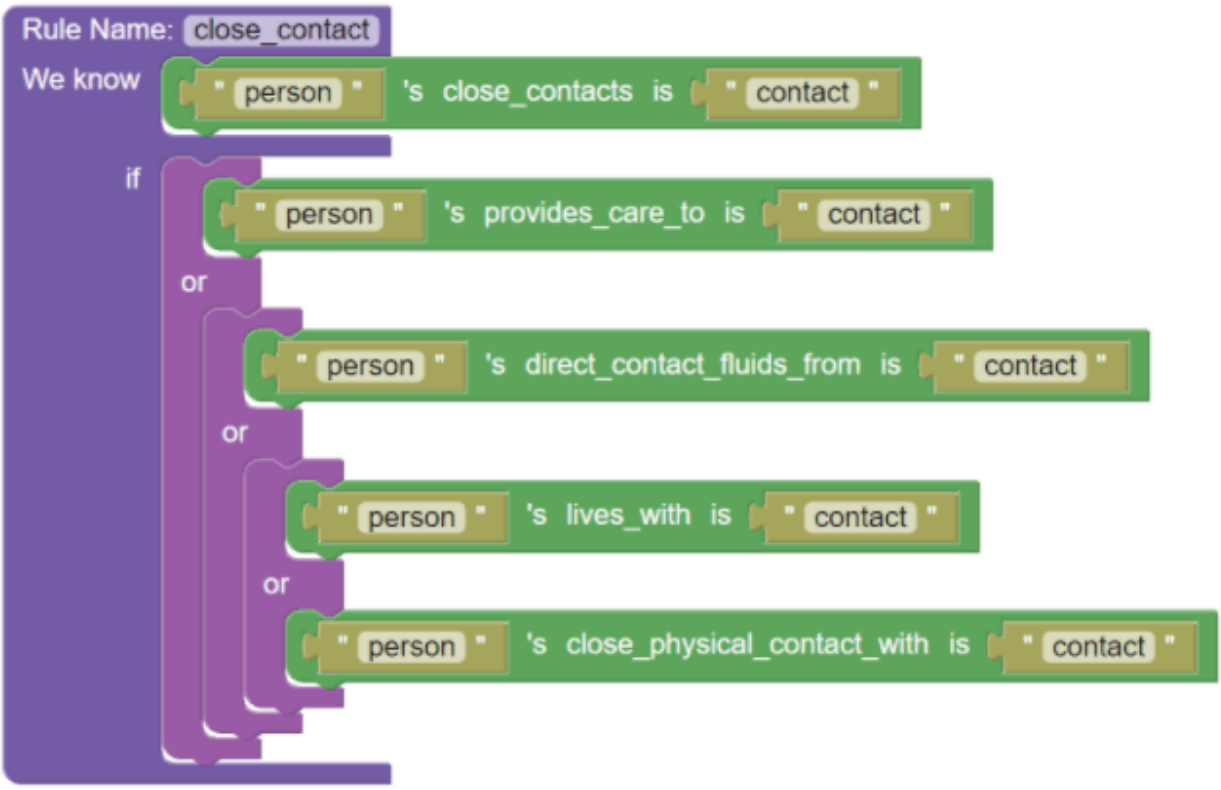
exception. No isolation is required unless you have a serious symptom, in which case it is required; unless you also have no known exposure and a negative test result, in which case it is not required again. So, we will encode this rule as an exception to the requirement to isolate for 10 days if you have serious symptoms.



Note that with an encoding tool that does not allow you to write exceptions (which is most of them), you would need to combine the default, the exceptions to the default, and the exceptions into a single decision tree structure or rule. This reformulation of the source rules will work in obtaining the right answers, but masks certain knowledge the subject matter expert knows about the rules. That masking makes it more difficult to determine if and how the code needs to change later, when the rules change.

2.3.3.6 Close Contacts

We need a rule setting out whether a person is a close contact of another person. Here's that rule:



2.3.3.7 An Interpretation Challenge

Now, we have two rules left. The first is: if you are a close contact of a person who has tested positive for COVID-19, you are either required to isolate for 14 days, or for 10 days after you get symptoms, or until your symptoms resolve, whichever is longer. The second is: if you return to Alberta from elsewhere, you are required to isolate for 14 days, or for 10 days after you get symptoms, or until the symptoms resolve, whichever is longer.

We already have a rule that states if you have shown symptoms, you are required to isolate for 10 days or until the symptoms resolve, whichever is longer. Do the parts of these rules covering symptoms differ from that conclusion? Maybe, if we interpret these two rules as referring to "any symptoms, even symptoms that would not usually require isolation." We will assume that was the intent.

Each rule also has two different possible conclusions. Either you are required to isolate for 14 days, or you are required to isolate for 10. So what do we do?

We have a few options. We can either (1) implement each rule as two rules; or (2) implement both together as three rules. In the latter scenario, the rules would outline two rules for the two different reasons that you might need to isolate for 14 days, and a third for the event that if you have to isolate for 14 days, but then become symptomatic and you need to isolate for 10.

Three would seem better than four for the sake of simplicity, but four is the better option for few reasons. The first is that reformulating into three rules would suggest that the third rule applies to any other reason that people might be required to isolate for 14 days, such as when they are symptomatic but have tested negative and have known contacts. Another reason is that one of these rules could be changed without changing the other one, in which case we would need to break the third encoded rule in half anyway.

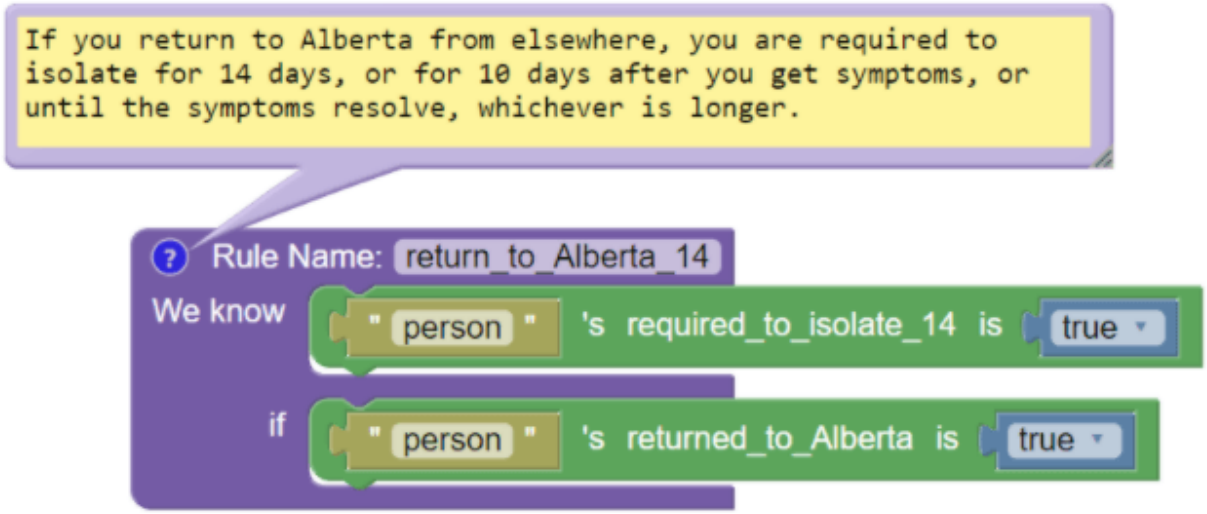
In this case, we are forced to take what looks like one rule and turn it into two. That is rather dangerous to the one-to-one relationship between code and rules that we are trying to maintain. One way to address that concern is by using comments. Comments can be added to the encoded rules to indicate that they refer to the same source rule. In future versions of Blawx, I hope to be able to navigate directly from the text of the rule to relevant encodings of the same section, and back, using a similar technique.

It is also not perfectly clear whether the requirement to isolate for 10 days and the requirement to isolate for 14 days are mutually exclusive. The use of the word "or" seems to suggest only one, but "whichever is longer" might be read as including the 14 days in the options. If you return to Alberta on day 1, you get symptoms on day 2, they resolve on day 5, can you leave isolation on day 11, or do you have to wait for day 14, because it is "longer"?

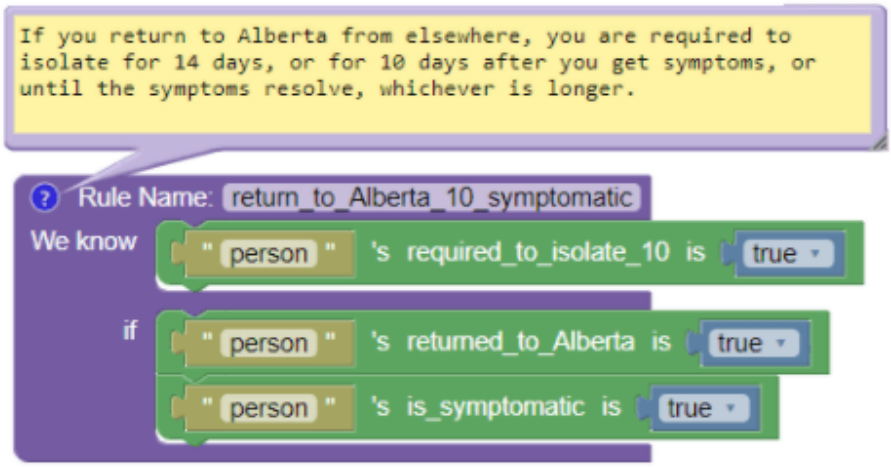
That is exactly the type of problems that encoding rules *as they are being drafted* would reveal. The Rules as Code approach makes rules easier to understand and implement, because they can be clarified early. For the purpose of this demonstration, I will proceed as if both isolation requirements can apply at the same time.

2.3.3.8 Isolation upon Return to Alberta

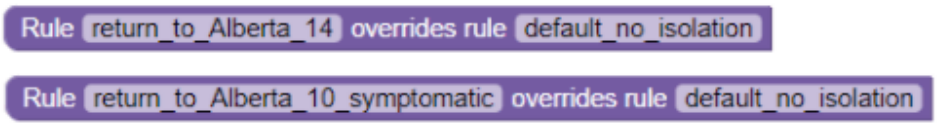
The first half of the "returning to Alberta" rule looks like this:



This rule does not include a requirement that the person not be symptomatic, because that would make the two rules mutually exclusive. Here's what the second half of the rule looks like:

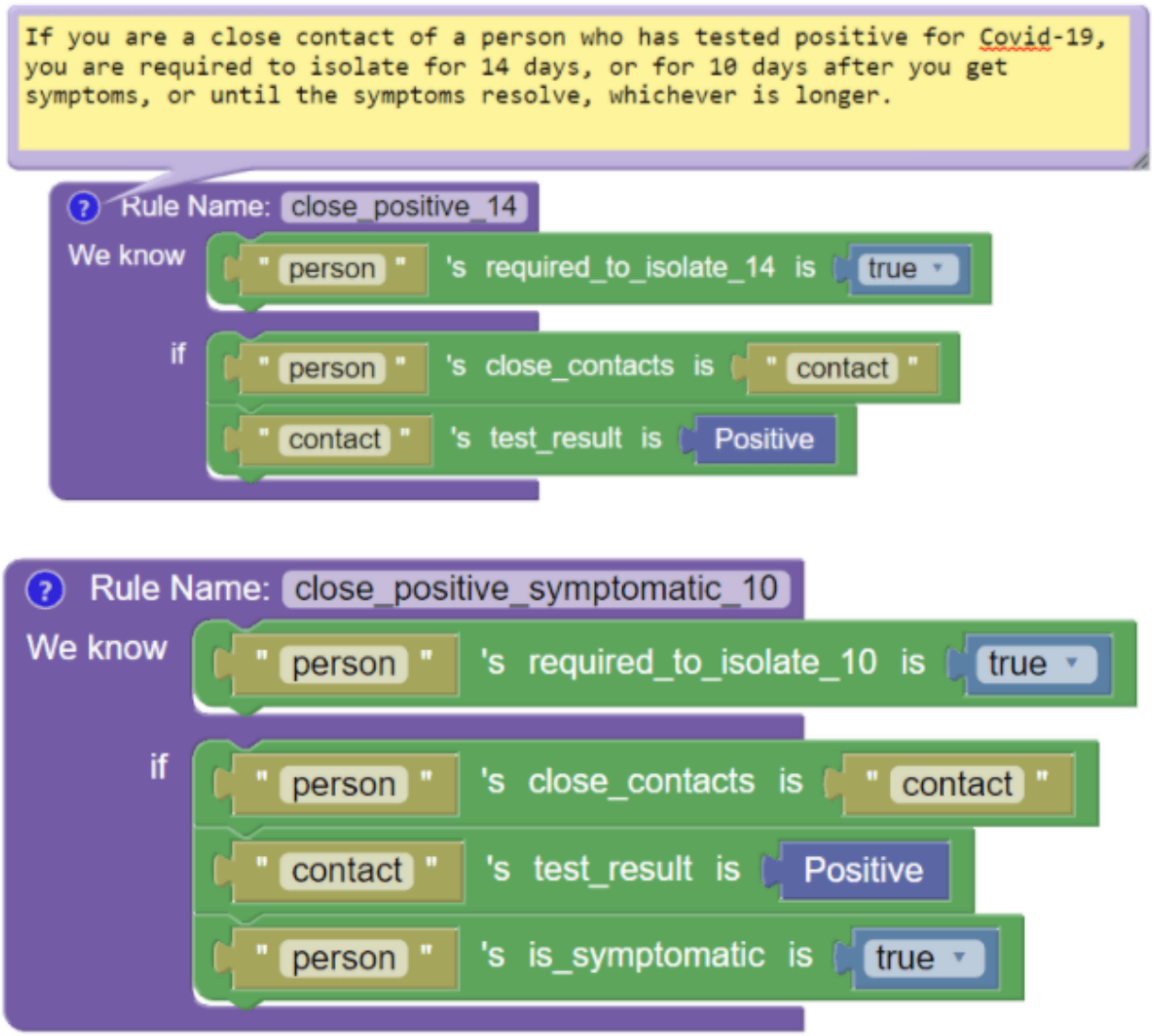


Both rules override the default that no isolation is required. So, we will add in the override blocks to make that explicit.



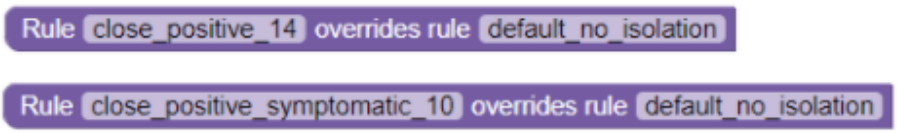
2.3.3.9 Isolation on Positive Test in Close Contact

Similarly, here are the two rules implementing the requirements about close contacts.



On this last rule you can see what the comment looks like while minimized.

As both rules override the default rule that no isolation is required, we will explicitly identify them as exceptions to the default.



2.3.3.10 Missing Pieces

If you were paying close attention, you may have noticed two things that have not been addressed in the rule encoding. There is no definition for what counts as a “known contact,” and there are some unresolved conflicts, specifically related to negative conflicts after traveling.

First, it is possible that a "known contact" may be defined elsewhere in the source rules, and it was simply missed during the review process. So this may not be an error on the part of the drafters of the rule and may be an error in the research undertaken in reviewing the rule. In any case, if it is a drafting error, it is just the type of drafting error that the Rules as Code approach makes more obvious.

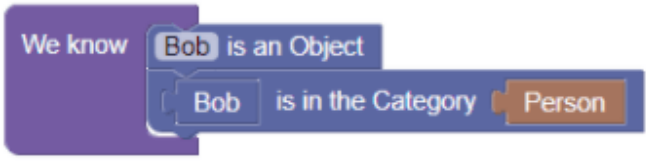
Second, in relation to unresolved conflicts, there is a default (no isolation required), which can be overridden by having serious symptoms (10 days isolation required), which can in turn be overridden by having a negative test result (no isolation required). On the other hand, we also have two rules that stipulate that people who have either traveled outside of Alberta or have had known contacts and become symptomatic, isolation of 10 days is required. That rule clearly overrides the default, but it does not say whether it is superior or inferior to the rule about a negative test result.

We will not try to resolve that issue in our encoding, but count it among the things we might have noticed had we been using Rules as Code to draft these requirements.

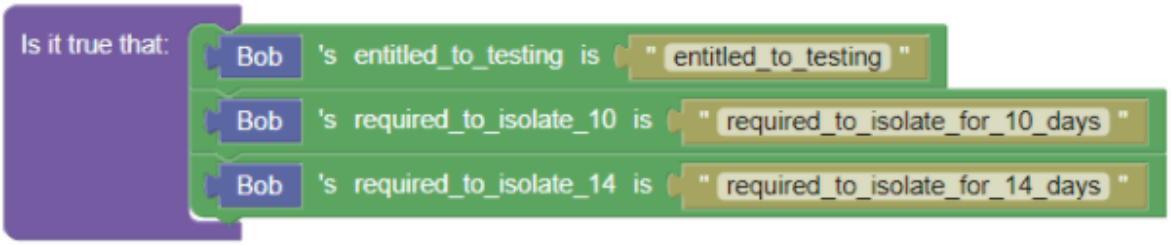
2.3.4 Encoding Facts and Queries

Now that we have written the code, we can encode some fact scenarios and determine whether or not the code is working as it should. Let's start with the simplest case. There is a person. If we know nothing else, is that person entitled to testing, or required to isolate?

We start with setting out the facts.



Now, we need to ask whether Bob is entitled to testing, or required to isolate.



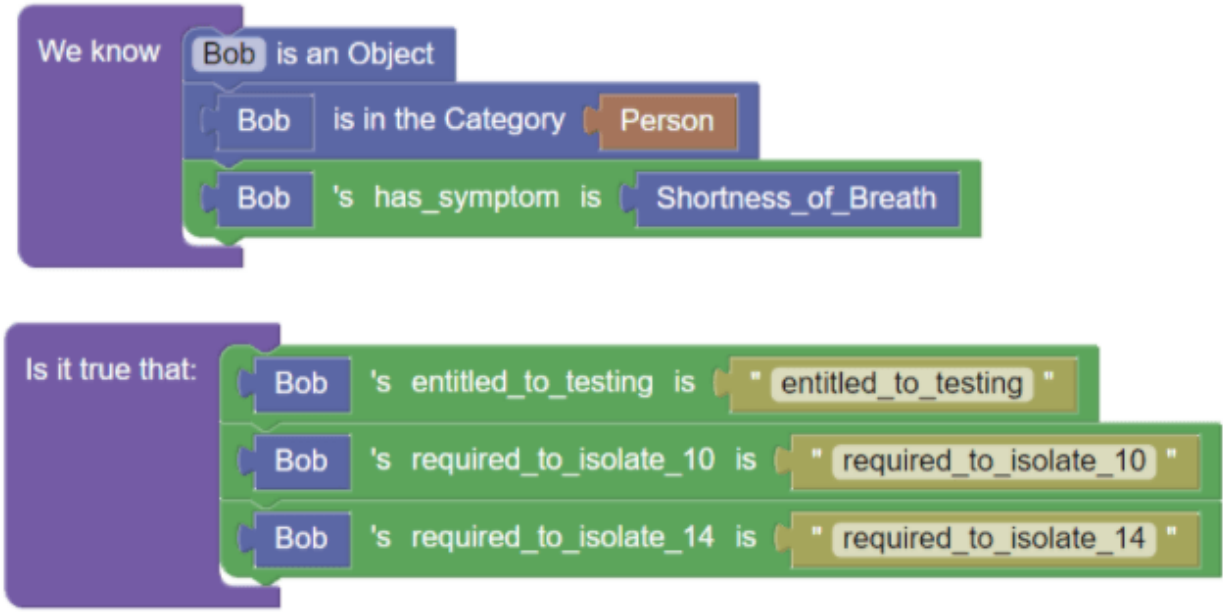
If you use "Run Blawx Code" command, you will get the following answer:

```
The answer is: Yes

entitled_to_testing: false, required_to_isolate_10: false,
required_to_isolate_14: false
```

Remember that for a search query, the "answer" is always to the question "do any records exist that satisfy this search," and then it provides the details. So the "yes" on the top line means only that Blawx can answer the question. The actual answer is below. The details are that Bob is not entitled to testing, and he is not required to isolate, so the defaults seem to be working fine.

Let's see if we can make Bob a person who is entitled to testing because he is short of breath.

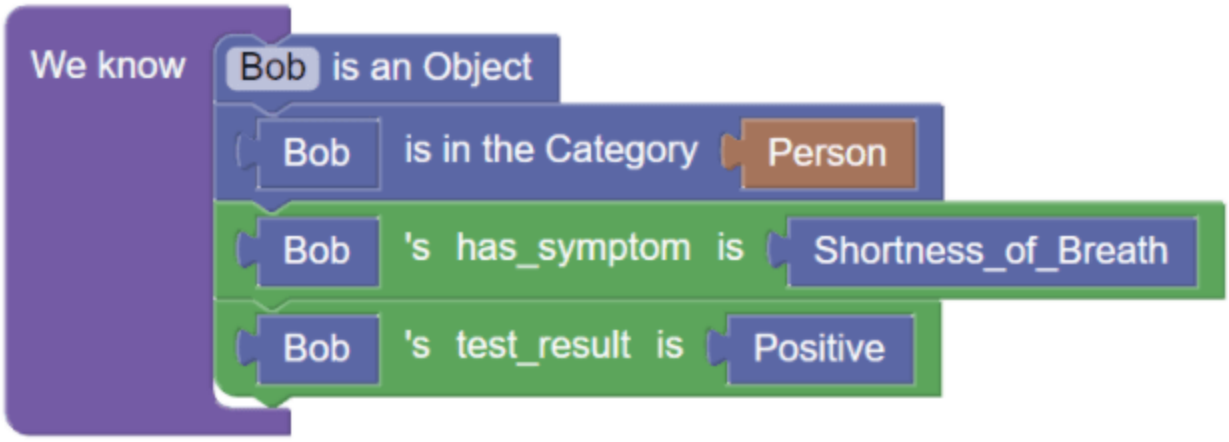


Running this code, we get the answer:

```
The answer is: Yes

entitled_to_testing: true, required_to_isolate_10: false,
required_to_isolate_14: false
```

Bob is now entitled to get testing. Let's assume that he does and tests positive.



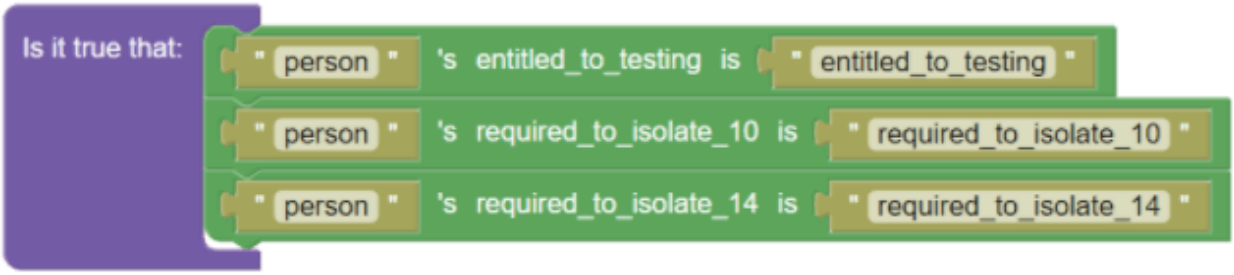
The answer is:

```
The answer is: Yes

entitled_to_testing: true, required_to_isolate_10: true,
required_to_isolate_14: false
```

As he has tested positive, the exception to the default rule applies and Bob is now required to isolate for 10 days. Now, let's try something more difficult. Let's say that Bob lives with Jane, Bob just came from out of province, Jane provides care to Ted, who has tested positive, and Jane has a stuffy nose (a minor symptom not requiring isolation on its own). Finally, Bob and Jane both live with Austin. Let's ask what conclusions the system draws.

We modify the question block to look like this:



What this means, in effect, is that you are asking Blawx to find all combinations of a "person" object and the relevant three values that it knows based on information in the database. When we run the code, we get the following answers:

```
The answer is: Yes

person: Austin, entitled_to_testing: false, required_to_isolate_10:
false, required_to_isolate_14: false
person: Bob, entitled_to_testing: false, required_to_isolate_10:
false, required_to_isolate_14: true
person: Jane, entitled_to_testing: true, required_to_isolate_10: true,
required_to_isolate_14: true
person: Ted, entitled_to_testing: false, required_to_isolate_10: true,
required_to_isolate_14: false
```

We can see Austin is not entitled to testing, and does not need to isolate. Bob is not entitled to testing, but because he has returned from out of province, he is required to isolate for 14 days. Jane is entitled to testing because she has had close contact with a person who has tested positive and has exhibited symptoms herself. She is also required to isolate 10 days because she is symptomatic (even though the symptoms alone do not require isolation) and has had close contact with a person who tested positive. This is one of the examples in which both requirements might apply at the same time. Ted is not entitled to testing, but he is required to isolate for 10 days because he tested positive.

2.3.5 Try the Demo

If you would like to play with the above rules in the live demo of Blawx and build your own fact scenarios and run your own queries, it is embedded below or you can click this link for the full-screen version.

Visit the web version of this article to view interactive content.

Using the above rules, and Docassemble-blawx, I have also created a prototype Docassemble interview that will ask you for the relevant information about an arbitrary number of people, and then tell you which of those people are entitled to testing or required to isolate, and for how long.

You can try that interview here. You will be asked to log in. You can create your own account, or use the email address demo@demo.com, and the password "Demo2020".

Visit the web version of this article to view interactive content.

Docassemble-blawx prototype

If you would like to see the source code for the docassemble interview, it is available on GitHub.

If you would like to see the version of the rules used to generate the interview (including code to "translate" the data from Docassemble) in the live demo of Blawx, you can click here.

3.0 Future Work

While Blawx already has a combination of features for Rule as Code applications that are not found elsewhere, there is much more we can do. Some of the short-term objectives include:

- Co-Drafting features that allow the user to draft and encode a rule in the same workspace, and link sections of code to sections of natural language rules.
- Allowing the user to specify and test for cardinality restraints (for example, a game can have multiple players, but only one winner)
- Giving the user an easy way to get warnings about easily-detected errors
- Expanding the available data types and functions to include dates, times, and durations.
- Adding the ability to explain answers and generate natural language versions of those explanations.
- More and better tutorials, documentation, and examples.
- Better capabilities for automatically generating test fact scenarios.

In addition to what we can do *to* it, there is the important work we can do *with* it.

Blawx provides an opportunity for important research to be done into some hard, unanswered questions about the use of declarative logic programming in law. For instance, in tools of comparable complexity, is there an efficiency advantage in encoding legal rules with a declarative tool over an imperative programming tool? How complicated does a rule set need to be before you gain those advantages? Are there efficiency gains to using a tool with defeasibility over a tool without? How difficult is it to teach a legal subject matter expert to encode their own knowledge with such a tool? What can be done to make that teaching more effective?

The answers to these types of questions would determine whether the value of declarative logic tools are as promising as I suspect. Until now it has not even been possible to do those sorts of experiments, because a user-friendly declarative logic tool with defeasibility did not exist. Blawx makes those experiments possible.

If you are interested in contributing to its development by writing code, trying the tool and providing feedback, writing tutorials, examples, or documentation, I am actively looking for collaborators.

4.0 Conclusion

The Rules as Code movement is a forward-looking approach that requires us to grapple with digitizing rules, not only the documents in which those rules are recorded. I believe declarative logic programming is the best technology for digitizing rules. It has been around a long time but has never seen widespread adoption in the legal services industry. This may be due to the fact that it has not been made accessible to people who are not programmers.

Blawx is a first effort at creating that accessibility to enable the legal profession to explore the possibilities. As the examples above demonstrate, it is possible to use Blawx to encode rules quickly and in a style that is similar to how they are drafted in natural language. Doing so helps to identify vague language, missing logic, and creates software tools that can be used by applications on the web.

If we continue to create tools that make it realistic for non-programmers to digitize rules and use digitized rules to automate services, it will fundamentally change the way legal services are provided. It is very difficult to overstate the potential of this shift. The legal services industry currently operates on the basis of asymmetric information: the lawyer has knowledge that the client does not. When it becomes possible to encode some significant portion of what the lawyer knows, when that knowledge can be used by clients without the direct involvement of lawyers, that promises to shift the balance of power in favor of the client.

At the same time, this type of technology has the potential to transform the work of lawyers in incredibly exciting ways. Could a lawyer evaluate combinations of multiple different versions of a contract for potentially mutually-incompatible obligations using hundreds of randomly-generated fact scenarios? Yes. Could they do it in mere minutes? Soon, with the help of tools like Blawx, that will be a realistic possibility.

Are there ways in which legal tasks could be automated with this technology in the future? Absolutely, yes, though not many. Most of the tasks that are easiest to automate are below the threshold of services a lawyer can economically provide. They are too small, and without automation the lawyer cannot achieve a profitable volume. The far more important change is that this type of technology will

make legal service providers better, faster, and more valuable, and increase the scope of the services that are possible.

However, these outcomes are only possible if talent and resources are invested in designing tools that lawyers will actually use.

Jason Morris is a dad, husband, and occasional dungeon master. He is also a lawyer and proprietor of Round Table Law, a virtual law firm in Sherwood Park, Alberta, Canada. He holds an interdisciplinary LLM in Computational Law from the University of Alberta Faculty of Law and Department of Computing Science. In 2018 he was awarded an ABA Innovation Fellowship for his work on tools to generate case-based reasoning systems for subjective legal issues. He is a sessional instructor of "Coding the Law" at the University of Alberta Faculty of Law. Most recently, he has been appointed as a Senior Researcher in Symbolic Artificial Intelligence at the Singapore Management University Centre for Computational Law. If you'd like to reach out to Jason, he's available on Twitter, via email, and on GitHub.

Footnotes

1. MM. J. Sergot, F. Sadri, A Kowalski, F. Kriwaczek, P. Hammond, and H. T. Cory, "The British Nationality Act as a Logic Program" Communications of the ACM Vol. 29-5, 370-386 (1986). Available at: http://www.doc.ic.ac.uk/~rak/papers/British%20Nationality%20Act.pdf. ↩
2. See generally, Anoush Darabi, "New Zealand explores machine-readable laws to transform government." apolitical, May 11, 2018. Available at: https://apolitical.co/en/solution_article/new-zealand-explores-machine-readable-laws-to-transform-government; Better rules - better outcomes, New Zealand Ministry of Business, Innovation & Employment. Available at: https://www.mbie.govt.nz/business-and-employment/business/support-for-business/better-for-business/better-rules-better-outcomes/. ↩
3. *Id.* ↩
4. Tim de Sousa, Pia Andrews, "When we code the rules on which our society runs, we can create better results and new opportunities for the public and regulators, and companies looking to make compliance easier" The Mandarin. October 1, 2019. Available at: https://www.themandarin.com.au/116681-when-machines-are-coding-the-rules-on-which-our-society-runs-we-get-better-results-new-opportunities-for-the-public-and-regulators-and-companies-looking-to-make-compliance-easier/. ↩
5. Jason Morris, "Rules as Code" Law Practice Today. December 13, 2019. Available at: https://www.lawpracticetoday.org/article/rules-code/. ↩

6. Alex “Sandy” Pentland, “A Perspective on Legal Algorithms” MIT Computational Law Report. Available at: https://law.mit.edu/pub/aperspectiveonlegalalgorithms. ↩

7. Marc Lauritsen and Quinten Steenhuis, “Substantive Legal Software Quality: A Gathering Storm?” Proceedings of the Seventeenth International Conference on Artificial Intelligence and Law (ICAIL ’19). Association for Computing Machinery, New York, NY, USA, 52–62. DOI: https://doi.org/10.1145/3322640.3326706. ↩

8. *Id.* ↩

9. Linna, Daniel, The Future of Law and Computational Technologies: Two Sides of the Same Coin. MIT Computational Law Report. Retrieved from https://law.mit.edu/pub/thefutureoflawandcomputationaltechnologies; *See also* The Unmet Need for Legal Aid. Legal Services Corporation. Last viewed August 3, 2020. Available at: https://www.lsc.gov/what-legal-aid/unmet-need-legal-aid ↩

10. Susskind, R. E. (1991). EXPERT SYSTEMS IN LAW: A JURISPRUDENTIAL INQUIRY. ↩

11. M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammong, H. T. Cory, “The British Nationality Act as a logic program” Communications of the ACM, 1986. DOI: https://doi.org/10.1145/5689.5920. ↩

12. Morris, Jason, Spreadsheets for Legal Reasoning: The Continued Promise of Declarative Logic Programming in Law (April 15, 2020). Available at SSRN: https://ssrn.com/abstract=3577239 or http://dx.doi.org/10.2139/ssrn.3577239. ↩