SWRLCLI: A Program for Displaying and Debugging SWRL Rules Embedded in an OWL Ontology

Damion Dooley[1,*], William Hsiao[1]

[1]Centre for Infectious Disease Genomics and One Health, Simon Fraser University, 8888 University Drive, Burnaby, BC, Canada

Abstract

An aspirational promise made by OWL ontology advocates has been that the power of axiomatized classes and rule engine reasoning could be brought together to add intelligence to many kinds of problem domain that can be formalized in ontology. Food recipe preparation, a kind of process modelling that spans both home cooking and industrial food formulation processing plant contexts, is a problem space where we aim to demonstrate the fruits of the ontology class and rule inference intelligence approach. However, our attempt to develop SWRL rules to demonstrate intelligence about ingredients ran into the awkwardness of using available SWRL tools within Protege, our open source ontology curation platform of choice. There, the "OWLTab" and "rules" plugins make it difficult to determine if rules are firing, and provide no debugging information on them if they are not. We introduce SWRLCLI, a rapidly Claude AI-coded Java program based on SWRLAPI, that runs on the command line rather than inside Protege or as a stand-alone GUI. It offers a variety of features, including the ability to load a given ontology and its import files and instance data therein, test individual or sets of rules with a color-coded debug display in markdown of best but possibly incomplete binding of predicates, and the ability to anchor rules to match specific triples to probe how close rules have come to executing. Editing features include the ability to delete, add to or replace an ontology's rules via another set supplied in a local table, as well as the ability to use relationship labels rather than curies. Altogether this enables much easier experience in rule development for the open source community, and encourages a vision where a better portrayal of rules can be seen as an explanatory AI contribution to a neuro-symbolic logic partnership vision.

Keywords

OWL ontology, SWRL, SWRLTAB, debugging, reasoning

1. Introduction

SWRL is a fairly well known "IF - THEN" rule language which users of the Protege ontology editor can interact with to some extent via its built-in SWRLTab plugin, which is also available as a stand-alone app [1–3]. SWRLTab, which uses SWRLAPI and OWLAPI to interact with reasoners and OWL formats, provides users with an interface for adding, updating, deleting, and executing one or more SWRL rules directly in the current ontology loaded in Protege [4]. Its rule syntax is relatively intuitive to understand, with an "IF" antecedent or premise part that outlines the structural pieces of graph instance data that must be in place, and a "THEN" consequent or head part that details one or more new instances of triples to add. Since OWL is monotonic, SWRL rules can add but never take away such triples. Through the use of variables to represent entity instances, and with the ability to match object and data properties emanating from those variables, SWRL goes beyond the limitations of property chains in recognizing specified graph parts, and attaching new object properties across or calculating and adding new data properties on such content. Note that SWRL also has a SQWRL query language which has the same SWRL antecedent expressivity, along with a query select and result-shaping functions [5].

Having opened an ontology with a set of rules and a model of instance triples in Protege, and with the Pellet reasoner started (which SWRL and its built-in functions is compatible with) one may find under the right circumstances that one or more SWRL rules have been satisfied, which causes one or more instances of triples - either a new object or data property - to be added to the ontology in memory.

Proceedings of FOIS 2026 Satellite events co-located with the 16th International Conference on Formal Ontology in Information Systems (FOIS 2026), September 21-25, 2026, Vitória, Brazil.

*Corresponding author.

damion_dooley@sfu.ca (D. Dooley)

0000-0002-8844-9165 (D. Dooley); 0000-0002-1342-4043 (W. Hsiao)

© 2026 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 ISSN 1613-0073

published 2026-11-27

CEUR-WS.org/Vol-4342/short1.pdf

Users then have the option of saving newly generated triples into the ontology file. Deducing which new triples in Protege have been added by SWRL rules, apart from all the other triples added by regular Pellet reasoning, is challenging, since they are all blended together in Protege's "Individuals by Class" window.

Our tool mainly aims to improve on SWRLTab's ability to display individual rules in the context of Pellet ontology inferences, followed by SWRL rule processing. SWRLTab is unforgiving about how predicates are displayed in Protege, so while one can name a rule, its object and data property labels are only human-friendly if they happen to have URIs that contain their readable names in CamelCase or snake_case, such as "hasBrother". In OBO Foundry the Relations Ontology as well as other ontologies are encouraged to have their predicate URIs as numeric identifiers, yielding the display of rules as shown in Fig 1, where RO:0000053 translates to "has characteristic". There is an alternative Protege "rules" plugin, shown in Fig 2, which defaults to using rdfs:label for rule class and predicate labelling if found, but it has its quirks, too, in that it cannot reference SWRL math and other built-in functions in the same way as SWRLTab does. In either view, the layout of a rule provides no indication of predicate dependencies arising from shared variables.

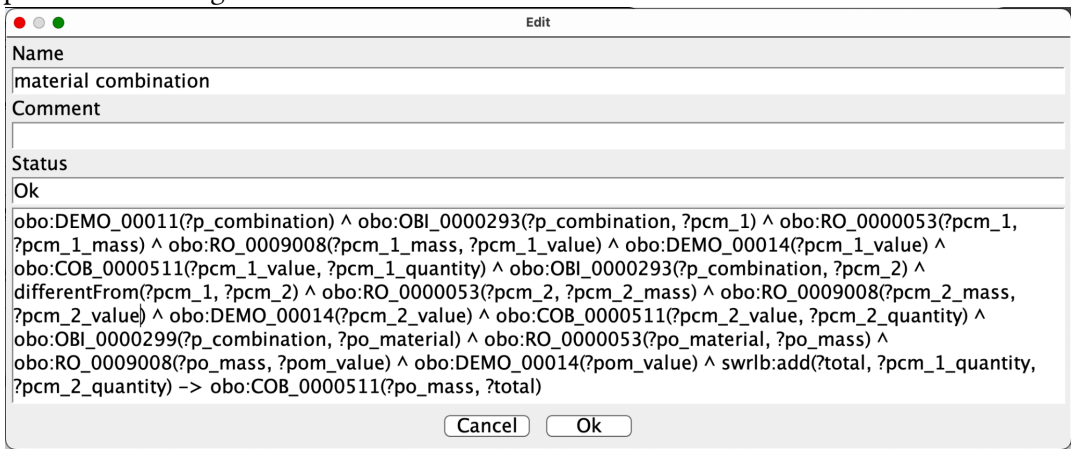

Figure 1: Display of a rule in SWRLTab's rule editor modal popup.

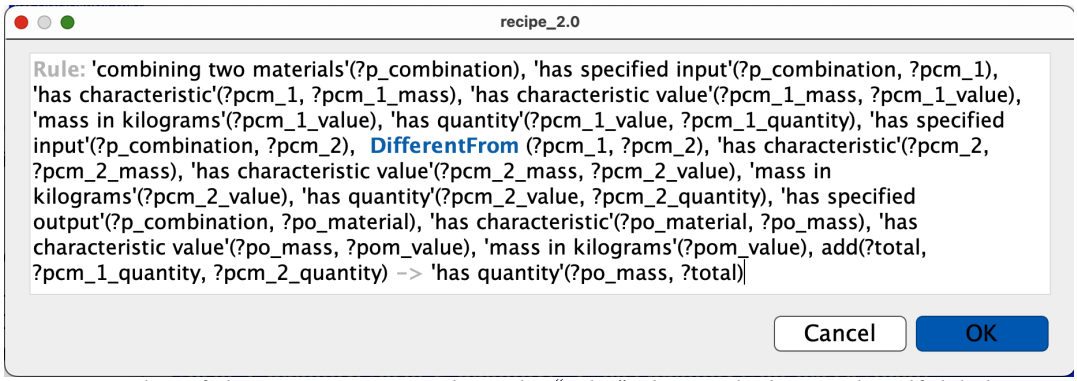

Figure 2: Display of the same Figure 1 rule in the "rules" plugin which consults rdfs:label to provide more readable predicates.

Other caveats exist while trying to edit rules in these windows, a number of which are covered by Michael DeBellis's tutorial [6]. For example, when editing rules in Protege in the "rules" tab, omit the "swrl:" or "swrlb:" prefixes for built-ins like "DifferentFrom" and "add", otherwise a syntax error occurs; Meanwhile SQWRL "sqwrl:sum" etc. functions are not accepted in "rules" tab, but are accepted in the "SWRLTab" and SQWRLTab" with prefix. SWRL has a few fundamental logical limitations too, like being unable to qualify the cardinality count of a property and being incompatible with some reasoners and datetime and numeric format functions; nevertheless the rule functionality SWRL does support motivated a workaround to solve the syntactic frustration mentioned here. As well, should a switch to another more powerful rule platform eventually become necessary, rule conversion should be trivial.

2. SWRLCLI

Our work mainly focuses on using SWRLAPI for iterative round-trip rule load/edit/test/save interaction, readability, and debug information improvements. Our use case is detailed recipe modelling, where ingredients are inputs and even outputs, and intermediary processes, such combining two materials, have the possibility of automatically calculating temperature, mass, volume, specific heat capacity, etc. about materials being combined, separated or shaped. To this end we have created a SWRLCLI command line application based on the SWRLAPI (and its OWLAPI dependency) located on GitHub with help information provided by the separate README.md markdown document [7]. For a given ontology (provided in any format that OWLAPI can work with), SWRLCLI executes one or a subset or all rules present in the ontology and/or in an input file of rules.

Ontology: recipe-edit.ofn (6419 axiom(s), 4 local import(s) from catalog-v001.xml)

Legend: process · material · characteristic · characteristic_value · information · error
PASS = matching assertion found in ontology
FAIL = no matching assertion found (predicate looked up, specific fact absent)
SKIP = not evaluated (required variable(s) unbound)
Quoted labels: combining two materials · has specified input · has characteristic · has characteristic value · mass in kilograms · has quantity · has specified output · has reference characteristic · characteristic value of · identity target · part of · member of
Bare predicates: temperature · over · changed

Rule: material combination

row	status	predicate	variables	match	notes
1	PASS	combining two materials	?p_combination	(r1.s2.add carrots to water)	
2	PASS	has specified input	?p_combination, ?pcm_1	(r1.s2.add carrots to water, r1.s1.water)	
3	PASS	has characteristic	?pcm_1, ?pcm_1_mass	(r1.s1.water, r1.s1.water mass)	
4	PASS	has characteristic value	?pcm_1_mass, ?pcm_1_value	(r1.s1.water mass, r1.s1.water mass value)	
5	PASS	mass in kilograms	?pcm_1_value	(r1.s1.water mass value)	
6	PASS	has quantity	?pcm_1_value, ?pcm_1_quantity	(r1.s1.water mass value, 1.0)	
7	PASS	has specified input	?p_combination, ?pcm_2	(r1.s2.add carrots to water, r1.s1.baby carrots)	
8	PASS	differentFrom	?pcm_1, ?pcm_2	(r1.s1.water, r1.s1.baby carrots)	
9	PASS	has characteristic	?pcm_2, ?pcm_2_mass	(r1.s1.baby carrots, r1.s1.baby carrots mass)	
10	PASS	has characteristic value	?pcm_2_mass, ?pcm_2_value	(r1.s1.baby carrots mass, r1.s1.baby carrots mass value)	
11	PASS	mass in kilograms	?pcm_2_value	(r1.s1.baby carrots mass value)	
12	PASS	has quantity	?pcm_2_value, ?pcm_2_quantity	(r1.s1.baby carrots mass value, 0.454)	
13	PASS	has specified output	?p_combination, ?po_material	(r1.s2.add carrots to water, r1.s3.mixture)	
14	PASS	has characteristic	?po_material, ?po_mass	(r1.s3.mixture, r1.s3.mixture mass)	
15	PASS	has characteristic value	?po_mass, ?pom_value	(r1.s3.mixture mass, r1.s3.mixture mass value)	
16	PASS	mass in kilograms	?pom_value	(r1.s3.mixture mass value)	
17	PASS	add	?total, ?pcm_1_quantity, ?pcm_2_quantity	(1.454, 1.0, 0.454)	
18	PASS	→ has quantity	?po_mass, ?total	(r1.s3.mixture mass, 1.454)	

Score: 17/17 body atoms satisfied — rule should have fired

Figure 3: A screenshot of the top of a SWRLAPI execution of an ontology's ruleset, where each rule is displayed as an exemplar binding of rule predicates to instance triples.

When a user specifies reasoning on an ontology with parameters "**--infer –debug --format markdown**", a markdown report will list each rule with the most complete binding of a rule's predicates to the ontology's pre-reasoned or inferred triples. Figure 3 reveals a number of features to improve comprehension, illustrated in the first rule, named "**material combination**" which finds two input materials that have mass quantities, and adds them up to yield an output material's mass. The rule

name is either the rdfs:label attached to a rule or a default “S[1…N]” name pattern. This is followed by a description if any, which is stored in the ontology as an rdfs:comment.

The debug option searches through all available instance triples to find a combination to display that satisfies a given rule. It ranks partial matches to find the best incomplete solution as well simply by ranking the count of satisfied triples for each rule. One can further constrain candidate solutions by providing a set of individual URIs to constrain which solution set is studied. A PASS/FAIL/SKIP status column indicates whether a match was successful or not. This is followed by a rule’s predicate column where predicates are indented according to the dependency they have on a preceding predicate’s variable(s). This yields an easier view on relatively independent blocks of graph structure being matched to.

A legend indicates a color code for kinds of entity (for example blue = process entity, green = material entity, etc.) which are matched by markdown-displayable CSS styling rules that colorize via a lookup table of predicate types and the type of term that inhabit their parameter position(s), for example that “combining two materials” is a class predicate taking one variable that matches a process instance. The “variables” and “matched” instance columns have corresponding coloration. The css coloration approach is not specific to COB / BFO material entity, characteristic, process etc. but rather is based on a swrlcli_config.yaml file containing user-definable predicates (the README.md indicates that one can place this file next to the ontology one is working with, among other options). The example yaml below shows the coloration used in above figures:

entity_styles:

process: ”span {color: blue;}”

material: ”span {color: green;}”

characteristic: ”span {color: tan;}”

characteristic_value: ”span {color: saddlebrown;}”

information: ”span {color: dimgray;}”

error: ”span {color: red;}” # unbound variables — override to change highlight

predicate_styles:

”has specified input”: [process, material]

”has specified output”: [process, material]

”has characteristic”: [material, characteristic]

”has characteristic value”: [characteristic, characteristic_value]

”has quantity”: [characteristic_value, information]

”mass in kilograms”: [characteristic_value]

”combining two materials”: [process]

”swrlb:add”: [information]

”temperature”: [characteristic]

”characteristic value of”: [characteristic_value, characteristic]

A close inspection of the matched instances in Fig. 3 reveals an additional naming convention about the various processes, characteristics and ingredients involved, namely that they are qualified by recipe (r1) and by state (s1, s2, s3) within the preparation of that recipe. One can see the inputs to the process are in state s1, the process itself occupies s2, and the output mixture exists in state 3. We aim to show how state-based identifiers go hand in hand with more complex chained processes and SWRL rules that make calculations along the way. It should become easier to analyze this rule, a “combining materials” as echoing a process model involving an initial state of materials and their characteristics, a transforming process - “add carrots to water” - and resulting materials and their characteristics. (A

side-note is that the “has characteristic value” relation is tentative and has not made it into the RO ontology yet.)

If no exemplar rule binding exists, a partly bound rule with the highest number of variable bindings will be shown instead. This helps the detective work to determine which triples are missing from the reasoned pool of triple instances (Figure. 4)

Rule: S1

row	status	predicate	variables	match	notes
1	FAIL	has reference characteristic	?material, ?ref_temperature	(?material, r2.s1.garlic clove temperature)	*no ObjectPropertyAssertion(RO_0009010, ?, r2.s1.garlic_clove_temperature)*
2	PASS	temperature	?ref_temperature	(r2.s1.garlic clove temperature)	
3	PASS	has characteristic	?material, ?temperature	(r2.s1.garlic clove, r2.s1.garlic clove temperature)	
4	PASS	temperature	?temperature	(r2.s1.garlic clove temperature)	
5	FAIL	has characteristic value	?ref_temperature, ?ref_value	(r2.s1.garlic clove temperature, ?ref_value)	*no ObjectPropertyAssertion(RO_0009008, r2.s1.garlic_clove_temperature, ?)*
6	SKIP	→ characteristic value of	?ref_value, ?temperature	(?ref_value, r2.s1.garlic clove temperature)	

Score: 3/5 body atoms satisfied — rule likely did not fire

Figure 4: A rule with incomplete antecedent predicates, highlighted in red.

The report legend includes a list of predicates by label which are hyperlinked to their online definitions; as well as “bare” predicates that don’t need to be quoted. This detail is significant because if the script is invoked as “./swrlcli --list-rules --format tsv [path to ontology file]”, it will display plain .swrl file syntax that can be redirected straight to a text file, edited, and fed back in with --file, as documented in the readme file. Also, when debug is requested in markdown format, a “temp.md” file is created which can be loaded in a browser to display as rendered markdown, which is convenient for editing and rerunning rules, with results visibly changing without the need to refresh (depending on browser capabilities). An extract from the top of this output is shown in Fig. 5.

```
rule material combination
'combining two materials'(?p_combination)
  ^ 'has specified input'(?p_combination, ?pcm_1)
    ^ 'has characteristic'(?pcm_1, ?pcm_1_mass)
      ^ 'has characteristic value'(?pcm_1_mass, ?pcm_1_value)
        ^ 'mass in kilograms'(?pcm_1_value)
        ^ 'has quantity'(?pcm_1_value, ?pcm_1_quantity)
  ^ 'has specified input'(?p_combination, ?pcm_2)
    ^ differentFrom(?pcm_1, ?pcm_2)
    ^ 'has characteristic'(?pcm_2, ?pcm_2_mass)
      ^ 'has characteristic value'(?pcm_2_mass, ?pcm_2_value)
        ^ 'mass in kilograms'(?pcm_2_value)
        ^ 'has quantity'(?pcm_2_value, ?pcm_2_quantity)
  ^ 'has specified output'(?p_combination, ?po_material)
    ^ 'has characteristic'(?po_material, ?po_mass)
      ^ 'has characteristic value'(?po_mass, ?pom_value)
        ^ 'mass in kilograms'(?pom_value)
          ^ swrlb:add(?total, ?pcm_1_quantity, ?pcm_2_quantity)
-> 'has quantity'(?po_mass, ?total)

rule S1
'has reference characteristic'(?material, ?ref_temperature)
  ^ temperature(?ref_temperature)
  ^ 'has characteristic'(?material, ?temperature)
    ^ temperature(?temperature)
  ^ 'has characteristic value'(?ref_temperature, ?ref_value)
-> 'characteristic value of'(?ref_value, ?temperature)
```

Figure 5: The markdown listing of an ontology's rule set, which is also available without coloration via the "--format tsv" option, can be saved or pasted into a text file and fed into SWRLCLI verbatim.

The SWRLCLI command is available via a "**./swrlcli [parameters]**" shell command for convenience; it accesses a compiled swrlcli-1.0.0-jar-with-dependencies.jar file. The application was developed predominantly by Claude Code v2.1.47 interacting with the claude-sonnet-4-6 model. The application has been tested on an Apple Macbook on a draft recipe ontology model, and is available at: https://github.com/cidgoh/swrlcli.

3. Conclusion

OWL logic has a variety of powers, ranging from recognition (determining that an entity possesses the necessary and sufficient conditions to be a member of a given class), contradiction detection (where, after inferences about disjoint relations, a predicate and its negation are found in a model), and synthesis (having recognized some structure of entity instances exist, new object or data properties are created that attach to existing or new instances). SWRL allows us to synthesizes new relationships and literal values, but in lieu of existing Protege SWRL management options, SWRLCLI has been developed to enable development of a visually insightful library of rules so that projects like our state-based recipe process modeling can deliver on their ontology impact vision. Color coding the relational parts of a rule, organizing rule triples in a hierarchy, enabling rules to have fixed value parts for testing purposes, with partial matches revealed, all help to advance OWL ontology prowess and reduce the amount of effort to do so.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] M.A. Musen, Protégé Team, The protégé project, AI Matters 1 (2015) 4–12.

[2] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, SWRL: A Semantic Web Rule Language Combining OWL and RuleML, (2004). https://www.w3.org/submissions/SWRL/ (accessed August 21, 2025).

[3] M. O'Connor, SWRLTab plugin, Github, 2014. https://github.com/protegeproject/swrltab-plugin (accessed June 17, 2026).

[4] M. O'Connor, swrlapi, Stanford, 2016. https://github.com/protegeproject/swrlapi (accessed June 17, 2026).

[5] M. O'Connor, A. Das, SQWRL: a Query Language for OWL, in: R.H.A.P. Patel-Schneider (Ed.), 2009. https://ceur-ws.org/Vol-529/owled2009_submission_42.pdf (accessed June 17, 2026).

[6] M. DeBellis, Semantic Web Rule Language (SWRL) Tutorial, MichaelDeBellis (2019). https://www.michaeldebellis.com/post/swrl_tutorial (accessed August 22, 2025).

[7] D. Dooley, SWRLCLI, Centre for Infectious Disease Genomics and One Health, 2026. https://github.com/cidgoh/swrlcli (accessed June 17, 2026).