Summarizing Verification Results for Common Logic Ontologies

Nathaniel Swan
University of Maine

This document serves as an overview for an undergraduate senior project that focuses on increasing usability of Macleod, Python software written by Torsten Hahmann that automates consistency and competency checking of ontologies specified in Common Logic (CL) syntax. The central component of the project involves designing a user interface from the ground up enabling users with no command prompt experience to process ontologies conveniently. Included in the user interface is a readable summary of results that are generated from Macleod's pre-existing Python scripts. Proposed work employs automating Macleod's current trying installation process, and designing a user interface where users select desired tasks to run against ontologies, processes them, and concluding with a formatted summary based on findings.

Categories and Subject Descriptors: K.4.3 [**Organizational Impacts**]: Automation—*Macleod*; D.2.2 [**Design Tools and Techniques**]: User interfaces—*Tkinter*; D.3.1 [**Formal Definitions and Theory**]: Syntax—*Common Logic, LADR, TPTP*

General Terms: Design, Management

Additional Key Words and Phrases: Python 2.7, Macleod, Tkinter, automation, task management, user interface

ACM Reference Format:

Nathaniel Swan - 2014. Summarization of Verification Results for Common Logic Ontologies

Compiled reports reflect analysis of these provers. An ideal finished product offers users an interface that tracks log data, directly links to erroneous lines within CL files, and facilitates the rerunning of tasks (i.e., consistency checks and theorem proofs) after any corrections are made.

1. INTRODUCTION

Macleod currently offers hardship to users interested in using the software simply due to the amount of dependencies required. Macleod requires installing Python, modifying multiple path variables, installing various theorem provers (some only currently available for Windows), and editing scripts within the package depending on where Macleod is installed. Lacking any sort of automation, this manual process can impose a major time sink.

When Macleod tasks are run, loosely formatted results are printed to a command line interface (CLI). The central part of this project improves usability by moving away from a CLI, and wraps readable summary data from ontology processing within a user interface. To be useful, the final product guides the user through the processing of ontologies and displays an interactive tree structure, that identifies major characteristics about the input ontology.

The interactive component revolves around the ability to click on a given node

From this, we summarize project goals:

(1) Automate installation of Macleod and its dependencies through an installer
(2) Provide a user interface that intuitively facilitates ontology processing
(3) Increase output readability - generate a tabular summary of results based on user selected tasks
    – Consistency checking
    – Non-trivial consistency checking
    – Theorem proving
(4) *(Optional)* - Push revised ontologies to Google Code repositories

1.1 The Onion Model

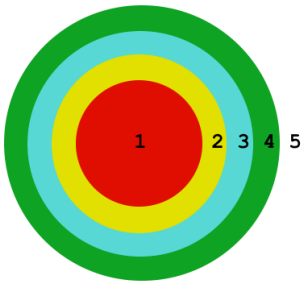

Fig. 1. The Onion Model - representing overall scope of the project. Core functionality begins in the center; each level outward adds features but simultaneous reduces probability of implementation in the completed project based on time constraints.

2. RELATED WORK

2.1 Structure and Use of Ontologies

2.2 Artificial Intelligence

In the field of artificial intelligence (AI), two main types of theories exist, mechanism and content. Ontologies fall under the content category. Philosophy defines an ontology as "the study of the kinds of things that exists", that is, an object, its properties, and the relations between objects within a specified domain[**?**]. In the AI community, some mechanism theories are proposed as the key ingredient when developing machines considered to be *intelligent*. These mechanisms can sometimes create over-excitement, for a mechanism is only as good as the content theory of the domain for which the mechanism is intended[**?**].

Nathaniel Swan acknowledges University of Maine professors Sudarshan Chawathe and Torsten Hahmann. Chawathe offered assistance in documentation and presentation while Hahmann provided full project design and software support. Inquiries regarding the content of this document may be requested from Nathaniel Swan, 174 Brunswick Street Apt. 5, Old Town, ME, 04468, 207.385.3767, or nathanieljswan@gmail.com.

2.3

3. MATERIALS AND METHODS

3.1 Python

Because Macleod is written in Python, *Tkinter* seems to fit best as it is Python's de-facto standard for user interface design[**?**]. More specifically *ttk* (themed tk), extends Tkinter in ways that allow creating widgets that exhibit themes native to the operating system that Python is running from.

3.2 Theorem Provers

3.3 Version Control

3.4 Packaging

One of the biggest factors in designing software is the method of distribution for the final product. For Macleod, it made perfect sense to utilize Git as it offers a quick and easy 'installation' capability called *clone*. Any user of the software will be able to retrieve Macleod and all associated dependencies (add figure of dependencies to refer to) from its remote repository by typing the command *git clone ¡repository link¿*, where the repository link can be found on Macleod's GitHub web page **LINK TO GITHUB PAGE**.

The only requirement of the user prior to being fully self-sufficient with Macleod is having Python 2.7 installed.

4. MACLEOD

Macleod software performs a series of stages when processing ontologies. The figure below chronologically illustrates these stages.

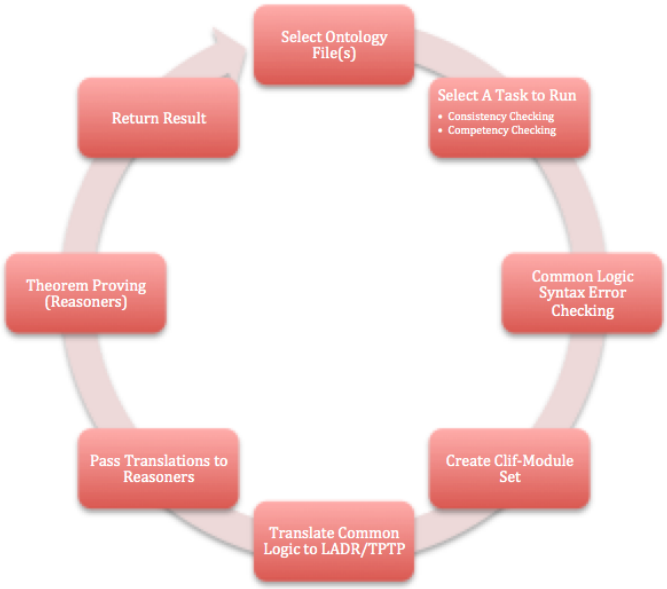

Fig. 2. Stages of Macleod During Ontology Processing

4.1 Selecting an ontology

The details for selecting an ontology, or better yet, ClifModule, is straight forward. Upon opening the application, the user chooses a *.clif* file or a directory that contains multiple *.clif* files.

4.2 Selecting a task

Selecting which task to run is as simple as clicking the appropriate button in the user interface once a file or directory has been chosen for the input. Once the button is pressed, Macleod scripts begin execution, starting first with error checking all *.clif* files.

4.3 Checking for syntax errors

Prior to the creation of the ClifModuleSet as detailed in the next section, Macleod analyzes each module for syntactic correctness. If an error in the *.clif* file, that is, the Common Logic syntax, the task in execution terminates and produces an error to standard out.

4.4 Creating the ClifModuleSet

A ClifModuleSet consists of ClifModules. To better understand, it is easier to think of the ClifModuleSet in terms of a tree structure with the initial *.clif* file selected as the root of the tree. All *.clif* files used by the root, or better stated, imported by the root, are children in the tree; the modules imported by the children, are now the grandchildren of the root. This is a recursive definition, thus, any *.clif* file can be selected and processed by the software.

The following figure is taken directly from Macleod, shows what a typical ontology (or ClifModuleSet) might look like.

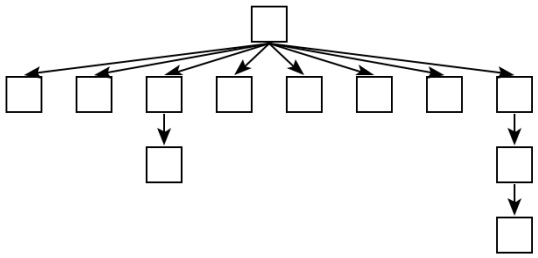

Fig. 3. Visualization of a ClifModuleSet - Thanks to my colleague Robert Powell for implementing the visualization of the ClifModuleSet as he is also working on this code base in terms of visualizing common logic ontology processing.

4.5 LADR/TPTP Translation

Once Macleod finishes developing the import structure, as shown in Fig. 3, the next step is to translate the ClifModules into a format that theorem provers can handle. For Macleod, there are two main formats, LADR and TPTP, each used for specific theorem provers. A list of theorem provers used in Macleod can be found in Fig. *?addthislater?*.

4.6 Theorem Proving and Results

The last stage of the process is validating the translations by passing them to theorem provers, returning formatted verification results to the summary tab in the GUI.

5. RESULTS

Coming soon to a paper near you. This section will very likely include a figure that is a screenshot of the final UI design upon completion. Closer to the end of the semester. For now, pork time.

Bacon ipsum dolor amet ham porchetta shoulder turkey tri-tip chuck capicola picanha jerky doner flank drumstick leberkas shankle beef ribs. Meatloaf chuck beef ribs cow andouille landjaeger pork chop pork jowl shankle venison prosciutto jerky short ribs pancetta. Pancetta beef pork chop shank. Kevin venison fatback picanha strip steak. Corned beef chicken pork chop frankfurter chuck bresaola, ham ground round kevin cupim tail alcatra. Andouille spare ribs pancetta ribeye beef ribs swine ground round.

6. FUTURE WORK

This section is to be completed when all time to work on the features of the project is exhausted. What features made it into the project, what features are half-implemented, and what features could make the software even better/more usable.

7. CONCLUSION

This section will likely be completed in the very near future. I haven't finalized the whole thought for the paper, but this is the section where it will come full-circle. I'll take a look at Macleod's evolution from where it started, to where it has come today, to where it will go. (Probably briefly mentioning where it will go to reiterate what was said in the previous section.) Alas, bacon ipsum.

Fatback porchetta ribeye andouille, pancetta hamburger jowl tail spare ribs bacon ground round chuck turkey corned beef sausage. Pancetta frankfurter leberkas, short ribs kevin shankle corned beef pig tongue flank shank ribeye turkey short loin. Strip steak pork chicken, tenderloin capicola jowl drumstick beef. Spare ribs hamburger beef ribs pastrami turducken, short loin rump shoulder kevin. Salami jerky cupim leberkas. Corned beef prosciutto salami bresaola, t-bone porchetta shoulder. Rump porchetta hamburger, leberkas ball tip frankfurter pork belly strip steak alcatra.

8. ACKNOWLEDGEMENTS

The project would not have been possible without the help of my classmates, advisor Torsten Hahmann, professor Sudarshan Chawathe, and Robert Powell II. Hahmann provided full project design and suggesting methods that offer the best technical advantages for this design. Chawathe reviewed and offered feedback for all iterations of project reports in addition to teaching in-depth technical writing practices. Finally, the project wouldn't be cohesive or useful without the tree drawing capabilities implemented by Powell.