RESEARCH

ROBOT: A tool for managing and releasing Open Biological Ontologies

James Overton[??*]
, James Balhoff[??]
, David Osumi-Sutherland[??]
and Christopher John Mungall[??]

[*]Correspondence: james@overton.ca
[??]Knocean, TODO, Toronto, Canada
Full list of author information is available at the end of the article

Abstract
ROBOT is a tool for working with Open Biomedical Ontologies

Keywords: sample; article; author

1 Background

Making ontologies is hard!

1.1 Conventions

Many ontology projects use an "edit" file for development. Editors modify this file to add terms and fix bugs, often using Protégé. When ready, the edit file is processed and packaged for release. ROBOT provides a range of commands to help with the release process.

We'll use the `edit.owl` file as our running example. It contains a fragment of Uberon, a cross-species anatomy ontology with rich logical axioms. You can use Protégé to look around.

What follows is a series of example commands that can be used to process `edit.owl` in various ways. The expected results are also provided in files that you can inspect. The example commands will create new files in a new directory, but with similar names, so that you can compare the results you get with the expected results. We use this system for testing ROBOT.

1.2 Converting

Ontologies are shared in different formats. The default format used by ROBOT is RDF/XML, but there are other OWL formats, RDF formats, and also the OBO file format, as well as the new OBO JSON format [CITE]

```
robot convert --input annotated.owl --output results/annotated.obo
```

1.3 Diffing two ontologies

To compare two text files and check for differences, you can use the Unix `diff` command on Linux or Mac OS X:

```
diff file1.txt file2.txt
```

or on Windows the `FC` command:

```
FC file1.txt file2.txt
```

Any number of graphical diff tools are also available, for example FileMerge is part of Apple's free XCode tools.

Although OWL ontology files are text, it's possible to have two identical ontologies saved to two files with very different structure. ROBOT provides a `diff` command that compares two ontologies while ignoring these differences:

```
robot diff --left edit.owl --right release.owl
```

If an `output` is provided then a report will be written with any differences between the two ontologies:

```
robot diff --left edit.owl --right release.owl --output results/release-diff.txt
```

See `release-diff.txt` for an example.

1.4 Merging

OWL ontologies are often divided into several `.owl` files, with `owl:imports` statements to bring them together. Sometimes you want to take all those imports and merge them into a single ontology with a single `.owl` file.

```
robot merge --input edit.owl --output results/merged.owl
```

You don't need `owl:import` statements: you can merge any number of ontologies by using multiple `--input` arguments. All the ontologies and their imports are merged into the first ontology.

```
robot merge --input edit.owl --input foo.owl --output results/merged2.owl
```

1.5 Filtering

Some ontologies contain more axioms than you want to use. You can use the `filter` command to keep only those axioms with ObjectProperties that you specify. For example, Uberon contains rich logical axioms, but sometimes you only want to keep the 'part of' and 'has part' relations. Here we start with a fragment of Uberon and filter for parthood relations:

```
robot filter --input uberon_fragment.owl --term obo:BFO_0000050 --term obo:BFO_0000051 --output
```

1.6 Module Extraction

The reuse of ontology terms creates links between data, making the ontology and the data more valuable. But often you want to reuse just a subset of terms from a target ontology, not the whole thing. Here we take the filtered ontology from the previous step and extract a "STAR" module for the term 'adrenal cortex' and its supporting terms:

```
robot extract --method STAR
    --input filtered.owl \
    --term-file uberon_module.txt \
    --output results/uberon_module.owl
```

The `--method` options fall into two groups: Minimal Information for Reuse of External Ontology Term (MIREOT) and Syntactic Locality Module Extractor (SLME).

- MIREOT: extract a simple hierarchy of terms
- STAR: use the SLME to extract a fixpoint-nested module
- TOP: use the SLME to extract a top module
- BOT: use the SLME to extract a bottom module

For MIREOT, both “upper” (ancestor) and “lower” (descendant) limits must be specified, like this:

```
robot extract --method MIREOT
    --input uberon_fragment.owl \
    --upper-term "obo:UBERON_0000465" \
    --lower-term "obo:UBERON_0001017" \
    --lower-term "obo:UBERON_0002369" \
    --output results/uberon_mireot.owl
```

For more details see:

- MIREOT
- SLME
- ModuleType

1.7 Reasoning

One of the main benefits of working with OWL is the availability of powerful automated reasoners. There are several reasoners available, and each has different capabilities and characteristics. For this example we’ll be using ELK, a very fast reasoner that supports the EL subset of OWL 2.

```
robot reason --reasoner ELK --input ribosome.owl --output results/reasoned.owl
```

It’s also possible to place the new inferences in their own ontology:

```
robot reason --reasoner ELK --create-new-ontology true --input ribosome.owl --output results/ne
```

1.8 Relaxing equivalence axioms

Robot can be used to relax Equivalence Axioms to weaker SubClassOf axioms. The resulting axioms will be redundant with the stronger equivalence axioms, but may be useful for applications that only consume SubClassOf axioms

```
robot relax  --input ribosome.owl --output results/relaxed.owl
```

1.9 Reducing Graph

Robot can be used to ‘reduce’ (i.e. remove redundant subClassOf axioms), independent of this previous step.

```
robot reduce --reasoner ELK --input ribosome.owl --output results/reduced.owl
```

1.10 Materialization

Robot can materialize all parent superclass and superclass expressions using the expression materializing reasoner, which wraps an existing OWL Reasoner

```
robot materialize --reasoner ELK --input emr_example.obo --term obo:BFO_0000050  --output resul
```

This operation is similar to the reason command, but will also assert parents of the form `P some D`, for all P in the set passed in via `--property`

This can be combined with filter and reduce to create an ontology subset that is complete

```
robot materialize --reasoner ELK --input emr_example.obo --term BFO:0000050 filter -t BFO:0000
```

1.11 Annotating

It's important to add metadata to an ontology before releasing it, and to update the ontology version IRI.

```
robot annotate --input edit.owl \
  --ontology-iri "https://github.com/ontodev/robot/examples/annotated.owl" \
  --version-iri "https://github.com/ontodev/robot/examples/annotated-1.owl" \
  --annotation rdfs:comment "Comment" \
  --annotation rdfs:label "Label" \
  --annotation-file annotations.ttl \
  --output results/annotated.owl
```

1.12 Mirroring

Many ontologies make use of `owl:imports` to bring in other ontologies, or portions of other ontologies. Large import chains involving multiple large ontologies are more prone to run-time failure due to network errors or latency. It can therefore be beneficial to mirror or cache an external ontology's import chain locally. This can be though of as analogous to what happens with standard dependency management tools for software development.

The following command will mirror a local

```
robot mirror -i test.owl -d results/my-cache -o results/my-catalog.xml
```

This will generate a directory `results/my-cache/purl.obolibrary.org/obo/ro/` with the imported ontologies as files. The file my-catalog.xml is a generated XML catalog mapping the source URIs to local files.

1.13 Templating

ROBOT can also create ontology files from templates. See template.md for details. Here's a quick example:

```
robot template --template template.csv \
  --prefix "ex: http://example.com/" \
  --ontology-iri "https://github.com/ontodev/robot/examples/template.owl" \
  --output results/template.owl
```

1.14 Validating Profiles

OWL 2 has a number of profiles that strike different balances between expressive power and reasoning efficiency. ROBOT can validate an input ontology against a profile (EL, DL, RL, QL, and Full) and generate a report. For example:

```
robot validate-profile -profile EL \
  --input merged.owl \
  --output results/merged-validation.txt
```

1.15 Chaining

On Unix platforms it's common to "chain" a series of commands, creating "pipeline" that combines several simple commands to accomplish a complex task. This works because most Unix tools communicate with streams of text. Unfortunately this doesn't work as well for OWL ontologies, because they cannot be streamed between commands, but we can achieve a similar result within ROBOT.

ROBOT allows several commands to be chained by using the output ontology as the input to the next step. Here's an example of a full release pipeline using chained commands:

```
robot \
  merge --input edit.owl \
  reason --reasoner ELK \
  annotate --annotation-file annotations.ttl --output results/example.owl \
  convert --output results/example.obo
```

Each command has been put on its own line, for clarity. Only the first command has an explicit `--input` argument. The following commands use the output of the previous command as their input. Also notice that the first two commands do not specify an `--output` file. Their output is not saved to the filesystem, only sent to the next command. But the last two commands both specify `--output` files, and their results are saved to different files.

Chained commands are powerful but can be tedious to write out. Consider putting them in a `Makefile`.

1.16 Prefixes

Terms in OBO and OWL are identified using IRIs (Internationalized Resource Identifiers), which generalize the familiar addresses for web pages. IRIs have many advantages, but one of their disadvantages is that they can be pretty long. So we have standard ways to abbreviate IRIs in a particular context by specifying **prefixes**. For example, Turtle files start with `@prefix` statements, SPARQL queries start with `PREFIX` statements, and JSON-LD data includes a `@context` with prefixes.

For robot we use the JSON-LD format. See `robot-core/src/main/resources/obo_context.jsonld` for the JSON-LD context that is used by default. It includes common, general linked-data prefixes, and prefixes for all the OBO library projects.

If you do not want to use the defaults, you can use the `--noprefixes` option. If you want to replace the defaults, use the `--prefixes` option and specify your JSON-LD file. Whatever your choice, you can add more prefixes using the `--prefix` option, as many times as you like. Finally, you can print or save the current prefixes using the `export-prefixes` command. Here are some examples:

```
robot --noprefixes --prefix "foo: http://foo#" \
  export-prefixes --output results/foo.json

robot --prefixes foo.json -p "bar: http://bar#" -p "baz: http://baz#" \
  export-prefixes
```

The various prefix options can be used with any command. When chaining commands, you usually want to specify all the prefix options first, so that they are used "globally" by all commands. But you can also use prefix options for single commands. Here's a silly example with a global prefix "foo" and a local prefix "bar". The first export includes both the global and local prefixes, while the second export includes only the global prefix.

```
robot --noprefixes --prefix "foo: http://foo#" \
  export-prefixes --prefix "bar: http://bar#" \
  export-prefixes
```

2 Results and Discussion

2.1 Use as part of Continuous Integration

Travis

Jenkins

[CITE]

2.2 Releases with GitHub

2.3 Ontology Starter Kit

2.4 Releasing to OBO

2.5 Use in Relation Ontology

We extended the concept of an incoherent ontology to include incoherent RBoxes

2.6 Use in GO

2.7 Templating

DOSDPs, QTT

2.8 Use of SPARQL

2.9 Use of Makefiles

On Unix platforms (including Mac OS X and Linux) you can use the venerable Make tool to string together multiple `robot` commands. Make can also handle dependencies between build tasks.

`TODO make`

`TODO make release`

When working with Makefiles, be careful with whitespace! Make expects tabs in some places and spaces in others, and mixing them up will lead to unexpected results. Configure your text editor to indicate when tabs or spaces are being used.

BioMake[CITE]

2.10 Missing features

Here are some other commands we should provide examples for:

- import, update imports
- add metadata
- package for release
- get term hierarchy
- convert formats

blah

2.11 foo

dd

- ROBOT: ROBOT is an OBO Tool
- OWL: Web Ontology Language
- RDF: Resource Description Format
- DOSDP: Dead Simple OWL Design Patterns

Competing interests

The authors declare that they have no competing interests.

Author's contributions
JO designed and implemented the framework. CJM provided requirements and implemented the materialize and reduce operations. DOS provided requirements and specifications and performed testing for GO.

Acknowledgements
Text for this section . . .

Figures

Figure 1 Sample figure title. A short description of the figure content should go here.

Figure 2 Sample figure title. Figure legend text.

Tables

Table 1 Sample table title. This is where the description of the table should go.

	B1	B2	B3
A1	0.1	0.2	0.3
A2	...	..	.
A3	..	.	.