

Scott Renshaw
Heath Reineke
Knowledge and Representation
03/13/18
Term Project

1. An overview of your project, including your motivation for tackling it and how you scoped it.

Our project focused on building a knowledge base and rules for classification of a bill becoming a law in the United State's government for Companion. Our motivation for this problem is that even though this is a process that affects all of us who live in the US a lot of people do not know this process and it can be quite confusing with having many different rules and regulations. We also saw the ability for a reasoner like the one present in Companion to bridge this gap by allowing users to query a knowledge base and receive a bills status instantly. Users can then delve more into that specific bill by clicking on it and viewing the attributes which led to that bill being of its status, which can be a very educational experience for users.

We scoped our project by creating a set of attributes and rules (horn clauses) for Companion and then scraping data from the United States Government websites that list bills and their status to create a knowledge base of bills to test our rules and attributes on.

2. A description of what you built. What reasoner did you use, what knowledge representation resources did you use? What did you get off the shelf, and what did you build?

We built a logic reasoner for these rules in Companion as well as a scraper and parser to make the set of bills and laws for our rules to be tested on. Companion already had the collections Bill-ProposedLaw, KilledBill-ProposedLaw and Law so we did not have to create these collections. We did create all the attributes for the items to describe the state of the Bill as well as the functions to classify these items. These were the most important part, as the horn clauses we wrote for classification depend completely on these attributes.

The attributes we had were in the form of predicates in Companion, and they were the following:

- writtenSponsored*: every bill we analyzed had this predicate, as all bills that we scraped were written and sponsored by some committee
- houseVoteResult*, *senateVoteResult*: after a bill is written and sponsored, it goes to a vote. These predicates took a bill as the first argument and a truth value as the second (either True or False, depending on whether or not the vote passed)
- presidentVeto*: if a bill was vetoed, it was given this predicate
- vetoOverturn*: this took a bill and a truth value, with true corresponding to an overturn and false corresponding to a failed overturn (or a successful veto)
- presidentSigned*: bills that become law must be signed by the president, so all laws have this predicate

We also created the collection of Bill-Idea to represent ideas people have for bills, since all bills are first before anything ideas. As an example, for a bill to become law, it must first be a Bill-Idea, then it must be written and then sponsored by some committee, then it must be voted on in the sector of congress in which it started (either the House or the Senate), then voted on in the other sector of congress, then signed by the president. We were able to just give bills the aforementioned attributes, then, through some horn clauses, we were able to let the reasoner handle the rest of the classification into either Bill-Idea, Bill-ProposedLaw, KilledBill-ProposedLaw, or Law.

This would have been interesting all on its own, and we could have simply manually inputted a few bills as examples. However, we thought it much more interesting to try to load a large number of bills into the knowledge base programmatically to give the system a bit more real world applicability. So, we also built a scraper and parser to scrape the United States Government websites for bill information as well as making them into a meld file format to be able to use in the knowledge base and query them. In all, we loaded all the bills from the 2011-2012 congress session (112th congress) and an additional ~300 bills from other sessions (to give us more evenly distributed bills amongst our categories).

3. Examples of your system in action, including screenshots and real output.

Output from scraping data for bills and saving it to create a base of examples

```

1 (isa s1379-112 Bill-Idea)
2 (writtenSponsored s1379-112)
3 (houseVoteResult s1379-112 True)
4 (senateVoteResult s1379-112 True)
5 (presidentSigned s1379-112)
6
7 (isa s1977-112 Bill-Idea)
8 (writtenSponsored s1977-112)
9
10 (isa s3278-112 Bill-Idea)
11 (writtenSponsored s3278-112)
12
13 (isa s2198-112 Bill-Idea)
14 (writtenSponsored s2198-112)
15
16 (isa s912-112 Bill-Idea)
17 (writtenSponsored s912-112)
18
19 (isa s2950-112 Bill-Idea)
20 (writtenSponsored s2950-112)
21
22 (isa s778-112 Bill-Idea)
23 (writtenSponsored s778-112)
24
25 (isa s2508-112 Bill-Idea)
26 (writtenSponsored s2508-112)
27
28 (isa s1983-112 Bill-Idea)
29 (writtenSponsored s1983-112)
30
31 (isa s3075-112 Bill-Idea)

```

End result was example document of roughly 35,000 lines long holding information from thousands of bills

Results from different queries using companion studio of things to see if they are bills, laws, or killed bills

Query / WM Fact Edit

```
(isa s1379-112 Law)
action = query
context = EverythingPSC; facts = all, env, infer
```

Answers:

in EverythingPSC:

?

A

(ist-Information EverythingPSC (isa s1379-112 Law))

[true]

Return to Query/Edit Page

Query / WM Fact Edit

```
(isa s572-112 Law)
  action = query
  context = EverythingPSC; facts = all, env, infer
```

Answers:

None found.



[Return to Query/Edit Page](#)

Query / WM Fact Edit

```
(isa s572-112 Bill-ProposedLaw)
  action = query
  context = EverythingPSC; facts = all, env, infer
```

Answers:

in EverythingPSC:

  (ist-Information EverythingPSC
(isa s572-112 Bill-ProposedLaw)) [true]



[Return to Query/Edit Page](#)

Query / WM Fact Edit

```
(isa hr1-100 Law)
  action = query
  context = EverythingPSC; facts = all, env, infer
```

Answers:

in EverythingPSC:

  (ist-Information EverythingPSC (isa hr1-100 Law)) [true]



[Return to Query/Edit Page](#)

Query / WM Fact Edit

```
(isa sres78-102 KilledBill-ProposedLaw)
  action = query
  context = EverythingPSC; facts = all, env, infer
```

Answers:

in EverythingPSC:

  (ist-Information EverythingPSC
(isa sres78-102 KilledBill-ProposedLaw)) [true]

[Return to Query/Edit Page](#)

4. A discussion of how you evaluated your system's performance – what are the relevant measures, and how did your system come out with regard to them?

Our main way of testing our system was simply spot checking different bills scraped from the internet with various attributes to see how well the reasoner could classify the bills using the rules we wrote. We were able to get perfect accuracy, which makes sense due to the straightforward nature of the process of bills becoming law and the ease of use with Companions with rule-building.

The auxiliary testing measure was how well the scraped data turned out in our massive meld file, as scraping the data turned out to be the brunt of the work needed to complete our project. This was sort of an iterative testing procedure for us: scrape some data, see how it looks, parse it to the meld file, fix bugs, repeat. The definitive final test for this was simply scanning the final 35,000-line meld file for errors. Once we completed the iterative testing process, we received no errors in the file.

5. An evaluation of the project itself: What worked well, what didn't work well, and what might you have done differently.

The most important part of our project was having good and accurate different attributes for our system, but at the same time keeping the attributes broad and not too specific. By having these good attributes it made it easy to write the rules to classify the system, and by keeping them broad we were able to use them for different situations rather than having to have tons of overlap. The scraper and parser for the government website was extremely helpful because it allowed for us to generate a huge test data set for our rules.

The main thing that didn't necessarily go well was the original plan for scraping all of our data from congress.gov, as the site was pretty unreliable. We spent a good deal of time building a reliable scraper for the site, and trying out different modules in python only to have the site go down twice while trying to scrape data. This unreliable source caused some headaches, but luckily the internet has many sources for getting data on government bills in the United States.

We could have done a few things differently, but we think the thing that would have been most interesting would have been to use data on bill statuses from gpo.gov as they have bulk data with some interesting attributes like who sponsored which bills, which committees they were referred to, etc. These

attributes likely could've been incorporated into our project in some creative way, but the data was difficult to work with as it didn't have concrete bill statuses and key attributes for our rules, so building the basic structure for the reasoner would have been incredibly difficult.