How to Replace Judges with Robots

Justus Perlwitz

2016-04-25

Contents

The other day, I found out something real fun: Not all bike lanes in Germany need to be used! Since bike lanes are quite dreadful and not at all safe, I wanted to write a handy tool to show me when to use a bicycle lane and when not. The German traffic code (StVO §2 Abs. 4 Satz 2) dictates:

> Radfahrer müssen einzeln hintereinander fahren; nebeneinander dürfen sie nur fahren, wenn dadurch der Verkehr nicht behindert wird. Sie müssen Radwege benutzen, wenn die jeweilige Fahrtrichtung mit Zeichen 237, 240 oder 241 gekennzeichnet ist. Andere rechte Radwege dürfen sie benutzen. Sie dürfen ferner rechte Seitenstreifen benutzen, wenn keine Radwege vorhanden sind und Fußgänger nicht behindert werden. Das gilt auch für Mofas, die durch Treten fortbewegt werden. Außerhalb geschlossener Ortschaften dürfen Mofas Radwege benutzen.

Here is my attempt at translating:

> Cyclists must ride in single file; they maybe only ride next to each other if regular traffic is not impeded. Cyclists must use bike lanes if the respective direction of traffic is marked with sign 237, 240 or 241. Other right bike lanes may be used. Furthermore, they are allowed to use right road shoulders if no bike lanes exist and pedestrians are not obstructed. The same applies to motorbikes that are pedal powered. Motorbikes are allowed to use bike lanesOutside of built-up areas [Br. English!].

Did I tell you I had a brief career as a technical translator? I kid you not.

Now you might wonder about this lawful detour. I wanted an excuse to combine logical programming and civil codes! It has always been a tempting topic, as civil codes seem like a pretty straightforward rule set that has around the same power as first-order logic. So how does one combine the two? Easy! Encode legal code in actual code!

And what language should one use when doing logical programming? While a lot of logical programming languages with very powerful inference engines exist, the most widely known and loved (or hated, if undergrad) programming language must be Prolog. Able to encode first-order logic with its simple, yet powerful horn clauses, it is the perfect candidate to encode some slick legal prowess. If lawyers ever find out about Prolog, they'll all be out of a job soon™ .

Essentially, we are going to create a miniature expert system that understands part of the German traffic code.

Let's encode the first two rules about riding in single file. Since computers do not know what impedes traffic, we need to generate additional rules about what constitutes an impediment. Let's come up with a few examples

```
impedes_traffic(bike, rush_hour).
impedes_traffic(bike, construction_site).
impedes_traffic(bike, narrow_road).
ride_single_file(bike, TrafficSituation) :-
    impedes_traffic(bike, TrafficSituation).
```

In our code, we have declared that bikes impede traffic when the traffic situation involves rush hours, construction sites and narrow roads. Let's ask our runtime system a few questions:

```
?- ride_single_file(bike, rush_hour).
true.
```

```
?- ride_single_file(bike, night).
false.

?- ride_single_file(bike, X).
X = rush_hour ;
X = construction_site ;
X = narrow_road.
```

We can add even more rules:

```
must_use(bike, bike_lane, sign(237)).
must_use(bike, bike_lane, sign(240)).
must_use(bike, bike_lane, sign(241)).
```

and we can state facts, like the sign we are currently seeing and ask questions:

```
?- must_use(bike, bike_lane, sign(237)).
true.
?- must_use(bike, bike_lane, sign(250)).
false.
```

In this case we are asking the system whether we need to use the bike lane when we see 237, whereupon we get a `true.` as an answer. In case sign 250 is visible, we do not have to use the bike lane.

We can also enquire in which cases the bike lane needs to be used.

```
?- findall(X, must_use(bike, bike_lane, X), Answer).
Answer = [sign(237), sign(240), sign(241)].
```

Here, we have found all signs which make bike lane usage compulsory.

1 Summary

While this is only a small example of what encoding civil codes in Prolog can do, we can already see that this language is a powerful tool for AI in legal research. In the coming articles I want to take a closer look at how to resolve more complex dependencies in legal texts and show the strengths of how Prolog allows to do backtracking searches to conveniently find all applicable answers to questions.