Re-learning Prolog

2013-11-18 BY GENE

Frankly, I think Prolog is the sexiest computer language ever. Programming with a free-form propositional logic? How cool is that?

tl;dr: https://github.com/ology/Logical-Music-Studio

When I was a freshman, my CS prof assigned me independent study in AI. I discovered Prolog and fell in love.

Once that year, my hand-held computer went missing. As a poor CS/Math student I was devastated. But as an out-of-the-closet-geek, I naturally wrote a program to help – in Prolog – called “detective” that considered suspects, means, motive and opportunity.

As I was composing the code, I realized what its answer would be. When I finished, it said that I was the most likely culprit. The next day I found my computer in a drawer with my unused, off-season clothes.

Ever since then, I have burned with a secret desire to re-embrace logic programming. So, last week I installed swi-prolog on my Mac and Ubuntu boxes. And for the first time in a long time, I feel the excitement of a kid learning something powerful and mysterious. But this time the kid is a professional software engineer…

So, I already did (and re-did) the classic genealogy example. I have coded up other academic programs like “20 Questions” and graphs with nodes and edges. But I am an engineer, and it’s time to make something useful and original – otherwise what is the point?

The ambitious part of me wants to make a “constraint based expert system”, but I decided upon a simpler musical track. I thought about, then planned a program I have imagined, off and on, for a while now: A music studio setup helper. One that can deduce what cables are needed to hook up a series of devices by simply knowing what devices I have.

First thing: Declare the cables used in an audio studio. So, with a bit of “constraint magic” I can now ask things like,

```
1 ?- ensure_loaded('cables.pro').
% cables.pro compiled 0.00 sec, 34 clauses
true.
2 ?- full_cable(trs, trs).
true.
3 ?- full_cable(trs, midi).
false.
```

That is, a TRS <=> TRS cable is ubiquitous – true! But a TRS <=> MIDI cord? No way. False.

Here is an example (recursive) rule, that knows about devices, their ports and the cables that can connect:

```
/* Rule: Devices connect their ports with cables */
/* device_cable/2 */
device_cable(Device, Cable) :-
    device(Device, Ports),
    device_cable(Device, Cable, Ports).
/* device_cable/3 */
device_cable(_, Cable, [Head|_]) :-
```

```
    port(Head, Cable, _, Size),
    full_cable(Cable, male, Size, _).
device_cable(Device, Cable, [_|Tail]) :-
    device_cable(Device, Cable, Tail).
```

That's it! Prolog is succinct (if not inscrutable).

This means something like, "Given the list of device ports, if that device can connect with a cable, that cable is male of the same size (e.g. dimensions, diameter)."

This rule can compute lots of things. You can ask questions like "Does my Behringer mixer have a MIDI port?" Or "What cables can connect with my Digitech processor?" Or what devices connect with RCA cords?"

But this is just an intermediate means to an end. What I am working out next is making a device_device() connection rule that will be able to answer "What cords do I need to connect these two devices?" Which is the "practical" matter, for me.

It's a work in progress, but I have made progress. Learning something potentially powerful is exciting. Feeling like a university freshman again is unexpected. I feel like I should be studying for a stats 101 test…

FILED UNDER: EDUCATION, SOFTWARE
TAGGED WITH: LOGIC PROGRAMMING, PROLOG

Epistemologist-at-large