Prolog lecture 2

Go to:

http://etc.ch/xVkG

Or scan the barcode

Today's discussion

Videos:

- Solving a logic puzzle
- Prolog rules
- Lists

Agenda

1) Voting/quiz questions from the videos
2) Answer the questions you asked on sli.do
3) Programming challenge

Which of these are true statements

- _ unifies with anything
- 1+1 unifies with 2
- prolog unifies with prolog
- prolog unifies with java

http://etc.ch/xVkG

What's the result of unifying cons(1,cons(X)) with cons(1,cons(2,cons(3)))

- False: they don't unify
- True: they unify
- True: X is now cons(2,cons(3))
- True: X is now cons(1,cons(2,cons(3)))

http://etc.ch/xVkG

Which of these is a list containing the numbers 1,2,3

- [1 , 2 , 3]
- [1 | [2 , 3]]
- [1 | 2 , 3]
- [1 , 2 | 3]
- [1 , 2 | [3]]
- [1 , 2 , 3 | []]

http://etc.ch/xVkG

Q: In the Zebra puzzle, why isn't the `rightOf` fact used help define the `nextTo` fact (e.g. `nextTo(A, B, rightOf(A, B)). nextTo(A, B, rightOf(B, A)).`)

Q: In the Zebra puzzle, why isn't the `rightOf` fact used help define the `nextTo` fact (e.g. `nextTo(A, B, rightOf(A, B)). nextTo(A, B, rightOf(B, A)).`)

A: You could easily define nextTo in terms of rightOf etc. (there's a supervision question on it). It's done without rules in the video because we've not covered rules at that point.

Q: Why is the last lecture still using the "bounds" library? Comment on its website: deprecated - No longer maintained. Please use clpfd.pl

Q: Why is the last lecture still using the "bounds" library? Comment on its website: deprecated - No longer maintained. Please use clpfd.pl

A: The bounds library still works so I have not changed it: this leaves me time for 'other things'...

(Please keep your questions to the videos for the current session.)

Q: I often write logically-correct code which doesn't terminate. What heuristics can I apply to see if this will happen without running the code?

Q: I often write logically-correct code which doesn't terminate. What heuristics can I apply to see if this will happen without running the code?

A: Its quite hard to do this without using things like arithmetic (Thursday) but let's look at some examples now and then some more next time.

Does this program terminate?

```
a(X) :- a(X).
```

Does this program terminate?

```
a(X) :- a(X).
```

Yes! Trick question. This program doesn't have any queries in it...

Does this program terminate?

```
a(X) :- a(X).

:- a(1).
```

Does this program terminate?

```
a([]).

a([_|T]) :- a(T).

:- X = <any_finite_list>, a(X).
```

Does this program terminate?

```
a([],R) :- a(R,[]).

a([H|T],R) :- a(T,[H|R]).

:- X = <any_finite_list>, a(X,[]).
```

What does this print?

```
a([],R) :- print(R), a(R,[]).

a([H|T],R) :- a(T,[H|R]).

:- a([1,2,3],[]).
```

Does this terminate?

```
a([]) :- a([1|X]).

:- a([]).
```

Write a program which runs out of stack as quickly as possible

Today's programming challenge - Map colouring

Colour the regions shown below using four different colours so that no touching regions have the same colour.

```
+----+--------+-------+
|    |   C2   |   C3  |
|    +----+---+--+----+
| C1 | C4 | C7   |    |
|    +----+------+    |
|    |           | C8 |
+----+    C6     |    |
| C5 |           |    |
+----+-----------+----+
```

Useful trick: testing your code

```
last([X],X).

last([_|T],R) :- last(T,R).

:- last([1,2,3],X), X=3.
```

This is better than

```
:- last([1,2,3],3).
```