Intelligent system programming

Jan Vium Enghoff

March 10, 2016

In collaboration with Søren Harrison.

1 Advantage of backward-chaining

Backward-chaining (BC) is a stronger choice when answering specific questions. If we use the wombat scenario then BC is better when answering questions like "Is there a wombat on the square next to mine?". FC on the other hand is more efficient answering questions like "How many dangerous squares do i know of?".

BC is said to be goal-driven, while FC is said to be data-driven.

2 Pseudo-code

Algorithms can be included using the commands as shown in algorithm 1.

Algorithm 1 Propositional logic backward-chaining entails

```
1:  function PL-BC-ENTAILS?(KB, q) returns true or false
2:      unproven ← Nil                                   ▷ symbols to prove
3:      unproven.push(q)
4:      v ← Nil                                          ▷ the already visited clauses
5:      agenda ← list of proven conclusions
6:
7:      return Entails?(KB, unproven, v)
8:
9:  function ENTAILS?(KB, unproven, v) returns true or false
10:     q ← unproven.pop()
11:     foreach clause in KB where conclusion = q do
12:         unproven′ ← Nil
13:         cProven ← true
14:         foreach symbol in clause.premise do
15:             if symbol not in agenda then
16:                 unproven′.push(symbol)
17:
18:         if unproven′ = ∅ then
19:             agenda.add(q)
20:             return true
21:         else
22:             if v contains clause.premise then
23:                 v.remove(clause.premise)
24:                 return false
25:             else
26:                 v.add(clause)
27:                 unproven.pushAll(unproven′)
28:                 foreach symbol in unproven do
29:                     cProven ← cProven and Entails?(KB, unproven, v)
30:         if cProven = true then
31:             return true
32:     return false
```

3 Putting BC to the test

Running the algorithm from section 2 on the test data provided will result in it terminating by returning false. This is due to the detected cycle in:

$$B \wedge E \wedge G \implies C \text{ and } C \implies G.$$

This can be fixed by adding additional Horn-clauses to the KB, i.e. by determining either G or C.

4 Running time

The algorithm from section 2 will run over every horn clause and symbol which might help prove q until a solution is found or there is not enough knowledge to find one. It's hard determining the worst case runtime since the algorithm is highly dependent on the horn clauses available in the KB. A lot of clauses with q as conclusion will resolve in the algorithm possibly checking the same symbols multiple times, which makes it seems like it's not gonna approach O(n).

A way to improve the average runtime is to implement already visited clauses v using a hash table data structure since the algorithm only will use insertion and lookup for this part.