Example of rule-based reasoning for the Lisp Expert System

Christopher Buckingham

April 8, 2014

1 Starting situation

```
(setf *rules '
 (R1 (mammal ((hair y)(give-milk y)))
  R2 (bird ((feathers y)(lay-eggs y)))
  R3 (carnivore ((mammal y) (eats-meat y)(pointed-teeth y) (forward-eyes y)))
  R4 (carnivore ((mammal y)(eats-meat y)(claws y)))
  R5 (ungulate ((mammal y)(hoofs y)))
  R6 (ungulate ((mammal y) (chew-cud y)))
  R7 (cheetah ((mammal y) (carnivore y) (tawney y) (dark-spots y)))
  R8 (tiger   ((mammal y) (carnivore y) (tawney y) (black-stripes y)))
  R9 (giraffe ((ungulate y) (long-neck y) (long-legs y) (dark-spots y)))
  R10 (zebra ((ungulate y)(black-stripes y)))
  R11 (ostrich ((bird y) (fly n) (long-neck y) (long-legs y)
              (black-and-white-colour y)))
  R12 (penguin ((bird y) (fly n) (swim y) (black-and-white-colour y)))
  R13 (albatross ((bird y) (fly-well y)))))

(setf *facts '((give-milk y) (hair y)(black-stripes y)))
```

Task: prove that Zeb is a zebra.

2 Forward chaining proof that fails

See whether any of the facts are a goal. For example, if (zebra y) was in the fact base, then you would know that Zeb was a zebra because zebra is a goal (as are cheetah, tiger, giraffe, ostrich, penguin, and albatross). If there is no goal fact, then you have to use the current facts and match them against the conditions of the rules to see if any of the rules produce some new facts. This is called forward chaining: going from the facts to the conclusions, where any proved rule conclusions become new facts added to the fact base.

As new facts are added to the fact base, the matching with conditions of rules is repeated. The iteration (cycle) continues until either a fact is produced that matches a goal or there are no new rules that match the fact base.

2.1 Worked example for Zeb

1. Triggered rules: R1;

2. Fire R1: add (mammal y) to facts.

3. *facts = ((mammal y) (give-milk y) (hair y) (black-stripes y)))

4. Triggered rules: R1 but its fact already exists in the fact base so no point in firing it again.

5. No other rule is triggered, so forward chaining fails.

3 Forward chaining proof that succeeds

Let's add a fact to the working memory (*facts) that will cause forward chaining to prove the query "Is Zeb a zebra?".

```
   (setf *facts '((give-milk y) (hair y)(black-stripes y) (hoofs y)))
```

1. CYCLE 1

   (a) Triggered rules: R1
   (b) Fire R1: add (mammal y) to facts:
   (c) *facts = ((mammal y) (give-milk y) (hair y) (black-stripes y) (hoofs y)))
   (d) mammal is not a goal so continue forward chaining

2. CYCLE 2

   (a) Triggered rules: R1, R5
   (b) R1 already fired so fire R5: add (ungulate y) to facts
   (c) *facts = ((ungulate y) (mammal y) (give-milk y) (hair y) (black-stripes y) (hoofs y)))
   (d) ungulate is not a goal so continue forward chaining

3. CYCLE 3

   (a) Triggered rules: R1, R5, R10
   (b) R1, R5 already fired: fire R10: add (zebra y) to facts.
   (c) *facts = ((zebra y)(ungulate y) (mammal y) (give-milk y) (hair y) (black-stripes y) (hoofs y)))
   (d) zebra is a goal so forward chaining succeeds and the animal is identified.

4. Output: ZEB IS A ZEBRA!

4 Backward chaining

```
(setf *facts '((hoofs y) (give-milk y) (hair y)(black-stripes y)))
```

Try to prove the animal is a zebra. The general approach is to see whether Zebra is a known fact. If not, find a rule that has Zebra as the conclusion and try to prove the rule's conditions. It is backward chaining because you are starting with the conclusion and trying to prove the conditions.

4.1 Worked example

1. Is Zebra a fact?
2. No so find the rule that matches the conclusion zebra.
3. Rule 10 has a matching conclusion so try to prove R10
4. R10 is true if its conditions are true.
5. Put conditions on queue to prove: ((ungulate y) (black-stripes y))
   (a) Prove each element of the queue.
   (b) When the queue is empty, the rule has been proved.
   (c) If there are no more rules to try and the queue is not empty, then the rule cannot be proved.
6. First: prove ungulate
7. Is ungulate a fact?
8. No, so it can only be true if it is the conclusion of a rule and we can prove the rule.
9. Is ungulate the conclusion of a rule?
10. Yes, it is the conclusion of R5 and R6.
11. Both rules are put on the list of rules to prove
12. Current proof list: ((R5 or R6) (black-stripes y))
13. Try to prove R5 (because it is first in order, but there are other sorting mechanisms such as choose the most specific rule, with most conditions, because it is the best match to the current situation)
14. R5 is true if Zeb is a mammal and has hoofs so replace R5 with these conditions
15. Current proof list: (((mammal y) (hoofs y)) or R6) (black-stripes y))
16. Is mammal a fact?
17. No, so check whether it is the conclusion of a rule.
18. Yes, it is the conclusion of R1
19. Replace mammal with conditions of R1 on the proof list
20. Current proof list: (((hair y)(give-milk y)(hoofs y)) or R6) (black-stripes y))
21. Is (hair y) in the fact base?
22. Yes, so this condition is proved.
23. Current proof list: (((give-milk y)(hoofs y)) or R6) (black-stripes y))
24. Is (give-milk y) in the fact base?
25. Yes so this condition is proved.

26. Current proof list: (((hoofs y)) or R6) (black-stripes y))
27. Is (hoofs y) in the fact base? Yes, so R5 has been proved (mammal and hoofs both true).
28. R6 not needed because the proof list for ungulate to be proved was R5 or R6 and R5 has been proved.
29. Current proof list: ((black-stripes y))
30. Is (black-stripes y) in the fact list?
31. Yes, so all conditions on the proof list have been proved and so R10 has been proved.
32. Output: ZEB IS A ZEBRA!

This is a recursive algorithm, which you can see from the backward-chaining function in the Lisp code. The more complex function produces the rule chain used to prove the query (Is Zeb a Zebra?) and provides a much better explanation of the reasoning than forward chaining. Read the example output given for the Lisp code and try running it yourself to see what is happening.