Probabilistic Reasoning

Mahesh Devalla

November 15, 2016

1 Certain and uncertain reasoning

1.1 Crossing the street in traffic

We cross the street when walk sign is turned on. But this is not always true, in some special cases like when we are in a hurry, we dont wait for the walk sign. We just look in both the directions and conclude that the cars are pretty far away to reach you and we run across the road before the cars arrive. Here come a few questions, some among them are:

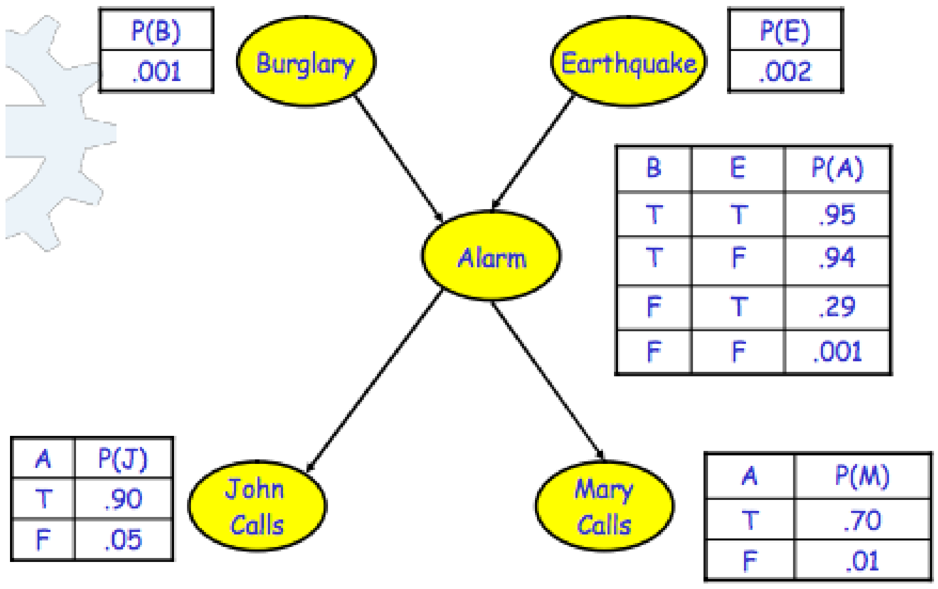
- Did we know surely that the cars we saw at a distance were not moving fast to hit us? How did we conclude this? Consider if we got our logic or prediction wrong and we may end up meeting with an accident. We have a little bit of uncertainty and certainty here.
- Can this reasoning provide any knowledge us to understand how a person could make a risky decision like crossing the road in traffic, without having any perfect logic(knowledge) on all the remaining relevant factors?

This is one of the examples to show that making a wrong decision can have weird consequences upon our actions, which we take without absolute certainty. In general, we usually take pretty much reasonably confident decisions and most of the times they become correct. But this certain and uncertain reasoning needs some explanation especially when we use artificial intelligence to make some decisions, which we call as *Probabilistic Reasoning.*

2 Probabilistic Reasoning

Probabilistic Reasoning includes or uses *probability theory* to solve the uncertainty in a problem with the help logic that can be deduced to exploit structure of formal argument. Probabilistic Reasoning finds a solution from traditional logical truth tables, and the results are derived from probabilistic expressions. Probabilistic Reasoning multiply the computational complexities of their probabilistic and logics, which is a bit difficult because they produce counter-intuitive results in some cases. There are many kinds of probabilistic reasonings. One of the important one is *Markov logic networks* that attempt to make a probabilistic extension to logical entailment try to solve the problems of uncertainty and also that lack evidence. *Markov Models* can be used to find the approximate solutions to complex problems.

For example if an alarm has been installed at home to detect threats like burglary,earth quakes etc.Below is the logical and probabilistic reasoning picture at a glance.



B	E	P(A)
T	T	.95
T	F	.94
F	T	.29
F	F	.001

A	P(J)
T	.90
F	.05

A	P(M)
T	.70
F	.01

Figure 1: Home Alarm Prototype

Here in this assignment a robot exists at a random location in the maze, where each position will have a specific colors like red, green, blue and yellow.Here, I implemented the Hidden Markov Models, Forward Filtering and the Viterbi Algorithms for the robot localization problem.

We perform all our tests on the following maze :

```
rryg
bbyy
rybr
gryg
```

Note:This is a very simple maze, and a new maze file could be given as input by changing in the file *LoadMaze.java*, or edit the file *Maze.txt*.

3 Hidden Markov Models

Hidden Markov model (HMM) is a statistical Markov model which has hidden states. As these states are not visible we rely on output which is visible. Here we define each state with a specific probability and it is distributed over the possible output tokens. Actually a HMM can be seen as an amalgamation of hidden variables that can control states to be selected for each sequence. The sequences generated will provide some knowledge about the states.

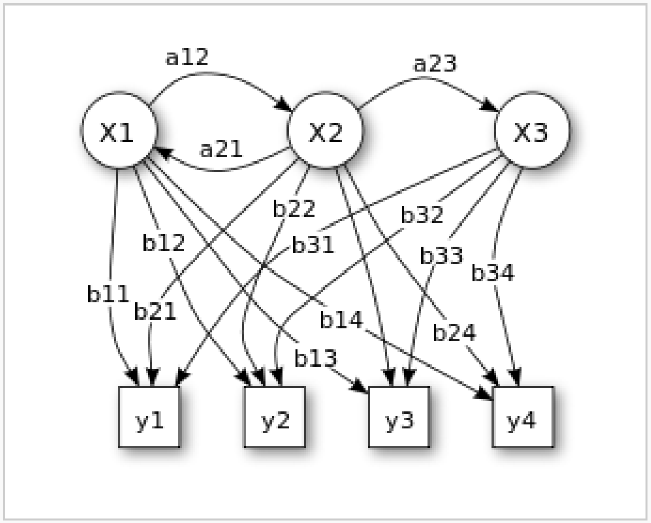


Figure 1. Probabilistic parameters of a hidden Markov model (example)
X — states
y — possible observations
a — state transition probabilities
b — output probabilities

Hidden Markov models has its applications in many fields, some of them are

- Speech recognition
- Gesture recognition
- Tagging
- Bioinformatics etc.

In this assignment Hidden Markov Models(HMM) allow us to combine previous results, along with the current calculation to make inferences, that gives the location of the robot.

In this algorithm, the location of the robot at time t depends on the location of the robot at time $t-1$ and the current calculation e_t.

At every step, we shall calculate X_t making use of X_{t-1} making use of the state transition model and update the values using a sensor model.

In the forward-filter method, we compute it recursively from the starting point X_0.

3.1 State Transition Model

As we are not sure about the location of the robot, So at any time(instance) t the robot moves from one location to the other location based upon calculations.

We compute a transition state matrix T where $T_{i,j} = P(X_{t+1} = j|X_t = i)$. This probability can be said to be equivalent to ,

$$T_{i,j} = \frac{1}{N(i)} if j \in Neighbors(i) else 0. \tag{1}$$

In the above equation, the function $N(i)$ is the number of neighbors of i, and $Neighbours(i)$ would be the set of all neighbors of i.

We have a column vector e_t, which is the current evidence. We need to take a transpose of the matrix T and multiply it with e_t, which is given by.

$$e_{t+1} = \sum_i P(X_{t+1} = i | X_t = j) P(j) \tag{2}$$

As a result we get a column matrix, we can compute probability for each next position from here

3.1.1 Implementation of HMM

Here we just iterate through all the positions in the maze(maze.txt), and compute each of their respective probabilities. Below is the code for its implementation.

```
              tModel = new double[area][area];
    for (int mazecord = 0; mazecord < area; mazecord++){
      currmove = 0;
      nextprob = 0.0;
      position = LoadMaze.stateXY(mazecord);
      for (int[] tempmove : LoadMaze.move)
        if (loadmaze.isValid(position[0] + tempmove[0], position[1] + tempmove[1]))
          currmove += 1;
      nextprob=(currmove == 0)?0.0:(1.0/currmove);

      for (int[] tempmove : LoadMaze.move) {
        if (loadmaze.isValid(position[0] + tempmove[0], position[1] + tempmove[1])) {
          nextcord = LoadMaze.xyState(position[0] + tempmove[0], position[1] + tempmove[1]);
          tModel[mazecord][nextcord] = nextprob;
        }
      }
    }
```

3.2 Sensor Model

By having the state transition model, we can't surely say the possible location of a robot. For example if we don't have any sensor information, then the distribution might be uniform. Additionally these sensor matrices are only diagonal matrices and it is given by.

$$O_t = P(e_t | X_t) \tag{3}$$

Where O_t is a diagonal matrix.

3.2.1 Implementation of Sensor Model

The implementation of Sensor Model is given below. The sensor model is constructed within the same method. In the code shown below, as expected, if it is the same color, then we set the probability in the matrix as 0.88, else 0.04 each.

```

  color = 'r';
    maze = new double[this.getArea()][this.getArea()];
    for (int r = 0; r < area; r++) {
      position = LoadMaze.stateXY(r);
      maze[r][r]=(loadmaze.getValue(position[0], position[1]) == color)?0.88:0.04;
    }
    red = maze;

    color = 'b';
    maze = new double[area][area];
    for (int b = 0; b < area; b++){
      position = LoadMaze.stateXY(b);
      maze[b][b]=(loadmaze.getValue(position[0], position[1]) == color)?0.88:0.04;
    }
    blue = maze;

    color = 'g';
    maze = new double[area][area];
    for (int g = 0; g < area; g++) {
      position = LoadMaze.stateXY(g);
      maze[g][g]=(loadmaze.getValue(position[0], position[1]) == color)?0.88:0.04;
    }
    green = maze;

    color = 'y';
    maze = new double[area][area];
    for (int y = 0; y < area; y++){
      position = LoadMaze.stateXY(y);
      maze[y][y]=(loadmaze.getValue(position[0], position[1]) == color)?0.88:0.04;
    }
    yellow = maze;
```

3.3 Forward Filtering Algorithm

Forward Filtering algorithm combines both the state transition model and the sensor models. Here we apply transition model to a previous observation and calculate the sensor data. We later update the model after each move. This process ends when all the data is read about the colors.

Probability distribution at each time t is given by.

$$f_{t+1} = \alpha O_t T^t e_t \tag{4}$$

We calculate the normalizing constant at each step. As we know that probability rule we add 1 then calculate the values of e_t.

3.3.1 Implementation of Forward Filtering

Here we loop through all the sensor data and at each state we apply a transition on previous state. We later normalize the data.

```
public void filtering() {
    double[][] sen = null;
    double[] forwardfilter;
    forwardfilter = forward;

    for (int color = 0; color < path.length; color++) {
      if(value[color]=='r')
      {
        sen=red;
      }
      else if(value[color]=='b')
      {
        sen = blue;
      }
      else if(value[color]=='g')
      {
        sen = green;
      }
      else if(value[color]=='y')
      {
        sen = yellow;
      }


      double[] content = calc(sen, calc(tTranspose, getForward()));

      double[] normalizedMatrix = new double[content.length];
      double sum = 0;

      for (double val : content)
        sum += val;

      for (int i = 0; i < content.length; i++)
        normalizedMatrix[i] = content[i] / sum;

      forwardfilter = normalizedMatrix;
      if (color == 0)
        viterbi[0] = forwardfilter;

      System.out.println("\nFiltered Distribuition at the step " + (color + 1) + ": ");
      System.out.println("Current Location is:" + path[color] + "");
      int len = length;
      for (int i = 0; i < forwardfilter.length; i++) {
        if (len == length) {
          System.out.print("\n  ");
          len = 0;
        }
      System.out.print(dformat.format(forwardfilter[i]) + ", ");
        len++;
      }
      System.out.println("");
      printStars();

    }
  }
```

4 Forward Backward Smoothing(Bonus:Extra Credit)

In this algorithm, we use the results produced by the forward-filtering phase. We save all the results over the entire sequence. We later run the backward recursion all the way back from t to 1, This computes the smoothed estimate for each step from the backward messages $b_{k+1:t}$ and forward message $f_{1:k}$.

4.1 Implementation of Forward Backward Smoothing

Below is the implementation of the Forward Backward Smoothing.

```
public void fbSmoothingAlgo() {
    double[] forw,back,dis;
    double[][] sen = null;
    double[][] forwvalues = new double[path.length][area];
    forw = getForward();
    back = getBackward();
    for (int color = 0; color < path.length; color++) {
      if(value[color]=='r')
      {
        sen=red;
      }
      else if(value[color]=='b')
      {
        sen = blue;
      }
      else if(value[color]=='g')
      {
        sen = green;
      }
      else if(value[color]=='y')
      {
        sen = yellow;
      }

      double[] message = calc(sen, calc(tTranspose, forw));
      double[] normMatrix = new double[message.length];
      double sum = 0;

      for (double val : message)
        sum += val;

      for (int i = 0; i < message.length; i++)
        normMatrix[i] = message[i] / sum;
      forw = normMatrix;
      forwvalues[color] = forw;
    }

    for (int color = path.length - 1; color >= 0; color--) {
      double[] message = multiply(forwvalues[color], back);
      double[] normMatrix = new double[message.length];
      double sum = 0;

      for (double val : message)
        sum += val;

      for (int i = 0; i < message.length; i++)
        normMatrix[i] = message[i] / sum;
      dis = normMatrix;
      System.out.println("\nSmoothed Distribuition at step " + (color + 1) + ": ");
      System.out.println("Current Location is : " + path[color]);
      int len = length;
      for (int i = 0; i < dis.length; i++) {
        if (len == length) {
          System.out.print("\n  ");
          len = 0;
        }
        System.out.print(dformat.format(dis[i]) + ", ");
        len++;
      }
      System.out.println();
      printStars();
      /*
       * Pick the relevant sensor model, and multiply it.
       */
      if(value[color]=='r')
      {
        sen=red;
      }
      else if(value[color]=='b')
      {
        sen = blue;
      }
      else if(value[color]=='g')
      {
        sen = green;
      }
      else if(value[color]=='y')
      {
        sen = yellow;
      }
      back = calc(tModel, calc(sen, back));
    }
  }
```

4.2 Results

This algorithm is implemented on the maze provided in "Maze.txt". The random steps taken by the robot, and the inference made by the robot as a result of the algorithm are listed below.

```

Filtered Distribution at the step 1:
Current Location is:14

  0.007, 0.008, 0.179, 0.007,
  0.008, 0.190, 0.009, 0.008,
  0.008, 0.009, 0.190, 0.179,
  0.007, 0.008, 0.179, 0.007,
**************************************************************************************************************************
**************************************************************************************************************************

Filtered Distribution at the step 2:
Current Location is:14

  0.007, 0.008, 0.179, 0.007,
  0.008, 0.190, 0.009, 0.008,
  0.008, 0.009, 0.190, 0.179,
  0.007, 0.008, 0.179, 0.007,
**************************************************************************************************************************
**************************************************************************************************************************

Filtered Distribution at the step 3:
Current Location is:10

  0.007, 0.008, 0.179, 0.007,
  0.008, 0.190, 0.009, 0.008,
  0.008, 0.009, 0.190, 0.179,
  0.007, 0.008, 0.179, 0.007,
**************************************************************************************************************************
**************************************************************************************************************************

Filtered Distribution at the step 4:
Current Location is:14

  0.007, 0.189, 0.009, 0.007,
  0.189, 0.009, 0.009, 0.189,
  0.009, 0.009, 0.009, 0.009,
  0.152, 0.189, 0.009, 0.007,
**************************************************************************************************************************
**************************************************************************************************************************
```

```

Filtered Distribution at the step 5:
Current Location is:13

  0.007, 0.189, 0.009, 0.007,
  0.189, 0.009, 0.009, 0.189,
  0.009, 0.009, 0.009, 0.009,
  0.152, 0.189, 0.009, 0.007,
*********************************************************************************************
    ****************************
*********************************************************************************************
    ****************************

Filtered Distribution at the step 6:
Current Location is:14

  0.007, 0.008, 0.179, 0.007,
  0.008, 0.190, 0.009, 0.008,
  0.008, 0.009, 0.190, 0.179,
  0.007, 0.008, 0.179, 0.007,
*********************************************************************************************
    ****************************
*********************************************************************************************
    ****************************

Filtered Distribution at the step 7:
Current Location is:15

  0.268, 0.015, 0.015, 0.268,
  0.015, 0.016, 0.016, 0.015,
  0.015, 0.016, 0.016, 0.015,
  0.012, 0.015, 0.015, 0.268,
*********************************************************************************************
    ****************************
*********************************************************************************************
    ****************************

Filtered Distribution at the step 8:
Current Location is:15

  0.007, 0.189, 0.009, 0.007,
  0.189, 0.009, 0.009, 0.189,
  0.009, 0.009, 0.009, 0.009,
  0.152, 0.189, 0.009, 0.007,
*********************************************************************************************
    ****************************
*********************************************************************************************
    ****************************

Filtered Distribution at the step 9:
Current Location is:15

  0.268, 0.015, 0.015, 0.268,
  0.015, 0.016, 0.016, 0.015,
  0.015, 0.016, 0.016, 0.015,
  0.012, 0.015, 0.015, 0.268,
*********************************************************************************************
    ****************************
*********************************************************************************************
    ****************************

Filtered Distribution at the step 10:
Current Location is:11

  0.007, 0.008, 0.179, 0.007,
0.008, 0.190, 0.009, 0.008,
0.008, 0.009, 0.190, 0.179,
0.007, 0.008, 0.179, 0.007,
```

5 Viterbi Algorithm(Bonus:Extra Credit)

Viterbi Algorithm calculates the steps for which the probability is maximum. The equation is given by as follows:

$$P(X_{t+1=i}|P_{t=j})P(j) \text{ } is \text{ } maximal. \quad (5)$$

We keep record of all the maximum values j and for every i. Here we find the position with the maximum probability and backtrack to find the path for the sensor readings.The difference between this approach and the Forward-Filtering algorithm is we take and consider the maximum probability instead of taking a accumulated value of the probabilities.

5.1 Implementation of Viterbi Algorithm

We modify this algorithm in a few ways. Here we find the best move that has the highest probability from the previous state. Later we have to backtrack to the beginning of the path.

```
public void viterbiAlgo() {
  double[][] sen = null;
  int max, prevVal;
  LinkedList<Integer> finalPath = new LinkedList<>();
  double maxVal,tempVal;
  int currVal, finalVal;
  Arrays.fill(prev_viterbi[0], -1);
  for (int color = 1; color < path.length; color++) {
    double[] viterbiVal = new double[area];

    for (currVal = 0; currVal < area; currVal++) {
      max = -1;
      maxVal = 0.0;
      for (finalVal = 0; finalVal < area; finalVal++) {
        tempVal = viterbi[color - 1][finalVal] * tModel[finalVal][currVal];
        if (tempVal > maxVal) {
          maxVal = tempVal;
          max = finalVal;
        }
      }
      viterbiVal[currVal] = maxVal;
      prev_viterbi[color][currVal] = max;
    }

    if(value[color]=='r')
    {
      sen=red;
    }
    else if(value[color]=='b')
    {
      sen = blue;
    }
    else if(value[color]=='g')
    {
      sen = green;
    }
    else if(value[color]=='y')
    {
      sen = yellow;
    }
    double[] message = calc(sen, viterbiVal);
    double[] normalized = new double[message.length];
    double sum = 0;

    for (double val : message)
      sum += val;

    for (int i = 0; i < message.length; i++)
      normalized[i] = message[i] / sum;

    viterbi[color] = normalized;
  }

  max = -1;
  maxVal = -1.0;
  for (int i = 0; i < area; i++) {
    tempVal = viterbi[path.length - 1][i];
    if (tempVal > maxVal) {
      maxVal = tempVal;
      max = i;
    }
  }
  currVal = max;
  finalPath.addFirst(currVal);
  for (int color = path.length - 1; color > 0; color--) {
    prevVal = prev_viterbi[color][currVal];
    finalPath.addFirst(prevVal);
    currVal = prevVal;
  }

  System.out.println("Final path with probability is:" + maxVal);
  System.out.println(finalPath);
}
```

5.2 Results

```
Smoothed Distribuition at step 10:
Current Location is : 11

  0.007, 0.007, 0.461, 0.021,
  0.007, 0.024, 0.002, 0.022,
  0.001, 0.001, 0.041, 0.209,
  0.001, 0.001, 0.187, 0.009,
**************************************************************************************
    **************************
**************************************************************************************
    **************************

Smoothed Distribuition at step 9:
Current Location is : 15

  0.020, 0.007, 0.005, 0.481,
  0.003, 0.005, 0.036, 0.019,
  0.001, 0.004, 0.010, 0.037,
  0.001, 0.006, 0.013, 0.351,
**************************************************************************************
    **************************
**************************************************************************************
    **************************
```

```
Smoothed Distribuition at step 8:
Current Location is : 15

  0.001, 0.022, 0.023, 0.026,
  0.014, 0.001, 0.001, 0.515,
  0.000, 0.001, 0.001, 0.117,
  0.002, 0.011, 0.117, 0.148,
  .....(Note:Please run the code to see the complete output, I intentionally truncated
    output here).
```

6 Results

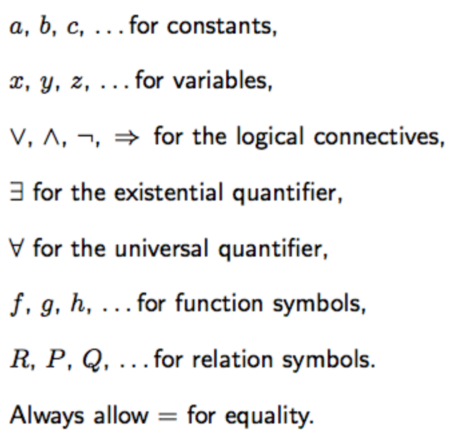
Viterbi algorithm results match with the actual path.The probability of a path depends on color sequences and the colors that are used in *Maze.txt*. Hence the Viterbi algorithm could be perceived as working.

```
Found the path with probability of 0.7773454325115525.
[2, 2, 2, 1, 1, 2, 3, 7, 3, 2]
```

7 First Order Logic

First-order logic like a normal language assumes the world contains Objects, Relations and Functions. Whereas *Propositional Logic* assumes the world contains facts and operations are performed based upon these facts and uses quantifiers. To contrary First-order logic utilizes variables that have some value upon the objects and allows the use of sentences that contain these variables.

Below is the notation used for First Order Logic:



a, b, c, ... for constants,

x, y, z, ... for variables,

$\vee$, $\wedge$, $\neg$, $\Rightarrow$ for the logical connectives,

$\exists$ for the existential quantifier,

$\forall$ for the universal quantifier,

f, g, h, ... for function symbols,

R, P, Q, ... for relation symbols.

Always allow $=$ for equality.

Figure 2: Notation of First Order Logic

7.1 Language for First Order Logic

Below are the rules for the language followed by FOL.

- All the symbols in the vocabulary are non-logical symbols of language.
- A finite(may be infinite) collection of variables like x, y, z, w etc.
- The Boolean connectives (negation), (conjunction), (disjunction),and (implication).
- The quantifiers (the universal quantifier) and (the existential quantifier).
- The round brackets) and (and the comma. (These are essentially punctuation marks; they are used to group symbols.)

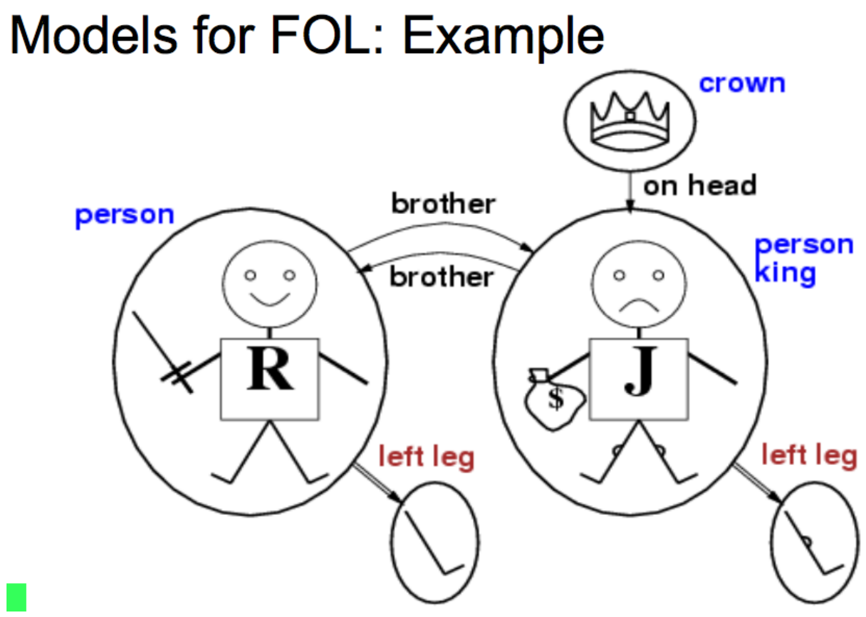


Figure 3: A Model of First Order Logic

7.2 Implementation of Tower of Hanoi in FOL

We can represent many models in FOL one of them is Tower of Hanoi. We know about the *tower of hanoi* problem, which needs no explanation. Below is the pseudo algorithm for the problem.

```
To move n discs from peg A to peg C:
1) Move n  1 discs from A to B. This leaves disc n alone on peg A
2) Move disc n from A to C
3) Move n  1 discs from B to C so they sit on disc n
```

This algorithm is easy to represent in programming languages like PROLOG. But in First Order Logic it is a bit different.

7.2.1 Problem

Problem: Move all disks (one at a time) from 1st peg to 3rd peg without putting a larger disk on a smaller disk.

7.2.2 Objects

- Disks: 1, 2, 3, 4, 5
- Pegs: A, B, C

7.2.3 Predicates for Sorting

- Sorting: DISK, PEG
- DISK(A) is FALSE
- PEG(A) is TRUE

7.2.4 Predicates for Comparison

- SMALLER(1,2) is TRUE.

Axiom: ∀XYZ.(SMALLER(X,Y) ∧ (SMALLER(Y,Z)) → SMALLER(X,Z) Premise: SMALLER(1,2) ∧ SMALLER(2,3) Situational constant(S): Identifies state of system after a series of moves. More predicates:

- Vertical relationship(ON) :ON(X,Y,S) asserts that disc X is on disk Y in situation S.
- Nothing on top of disk(FREE): FREE(X,S) indicates that no disc is on X.

7.2.5 Second axiom

∀ X S.FREE(X,S) ≡ ¬ ∃Y.(ON(Y,X,S))

- For all disks X and situation S, X is free in situation S if and only if there does not exist a disk Y such that Y is ON X in situation S.

7.2.6 More Predicates

- Acceptable (legal) move: LEGAL (X,Y,S)
- Act of moving disk: MOVE(X,Y,S)

7.3 Formula for tower of hanoi in FOL

We define a predicate symbol Move(n, x, y, z) of arity 4 with intended meaning: It is possible to move n discs from peg x to peg y using peg z for temporary storage.

- F_1 = ∀x∀y∀z Move(*0, x, y, z*).
- F_2 = ∀n∀x∀y∀z ((Move(n, x, z, y) ∧ Move(n, z, y, x)) → Move($s(n)$, x, y, z)
- Does F1, F2 ⊨ Move(*s 4(0)*, *first, second, third*) hold?
- Fact clauses are positive one-literal clauses:{*Sum(0, x, x)*}, $\{Move(0, x, y, z)\}$
- Procedural clauses are of the form $\{P, \neg Q_1 \ldots.., \neg Q_n\}$. P is the head and Q_1, . . . , Q_n the body.
  {¬*Sum(x, y, z)*,*Sum(s(x), y, s(z))*}
  {¬*Move(n,x,z, y)*,¬ *Move(n,z, y, x)*,*Move(s(n), x, y,z)*}

- Goal clauses are negative one-literal clauses. Note: They can also be represented in languages like prolog. It is mostly associated with artificial intelligence and computational linguistics. These days prolog supports graphical user interfaces,administrative and networked applications too.

7.4 Conclusion and Limitations

The problems like Tower of Hanoi and some logical problems in artificial intelligence can be solved with First Order Logic.To the contrary First Order Logic can't be expressed in some of the cases like natural languages. For an example first-order logic is undecidable in most of the cases, which tells us that a complete and terminating decision algorithm is impossible. This led scientists to study first-order logic with two variables and the counting quantifiers.

8 References

- *http : //tutorials.jenkov.com/java − internationalization/decimalformat.html*
- *http : //web.cse.ohio − state.edu/ dwang/teaching/cis730/Belief − net.pdf*
- *http : //www.cs.tut.fi/ elomaa/teach/AI − 2011 − 9.pdf*
- *http : //www.cs.mcgill.ca/ dprecup/courses/Prob/Lectures/prob − lecture01.pdf*
- *http : //web.stanford.edu/ danlass/NASSLLI − coursenotes − combined.pdf*
- *https : //en.wikipedia.org/wiki/Probabilistic*$_l$*ogic*
- *http : //groups.engin.umd.umich.edu/CIS/course.des/cis479/lectures/uncert.html*
- *http : //www.cs.dartmouth.edu/ devin/cs76/06*$_h$*mm − logic/assignment*$_m$*arkov.html*
- *http : //mlg.eng.cam.ac.uk/zoubin/papers/ijprai.pdf*
- *http : //aries.ektf.hu/ gkusper/ArtificialIntelligence*$_L$*ectureNotes.v.1.0.4.pdf*
- *http : //www.sdsc.edu/ tbailey/teaching/cse151/lectures/chap07a.html*
- *https : //www.ismll.uni − hildesheim.de/lehre/ai − 08s/skript/ai − 2up − 05 − fol.pdf*
- *https : //www.techfak.uni−bielefeld.de/ags/wbski/lehre/digiSA/WS0506/MDKI/Vorlesung/vl07*$_F$*OL.pdf*
- *https : //en.wikipedia.org/wiki/First − order*$_l$*ogic#Automated*$_t$*heorem*$_p$*roving*$_a$*nd*$_f$*ormal*$_m$*ethods*
- *http : //www.let.rug.nl/bos/lot2013/BB1.pdf*
- *http : //pages.cs.wisc.edu/ dyer/cs540/notes/fopc.html*
- *http : //www.cs.cmu.edu/ cdm/pdf/AutomLogic1 − 6up.pdf*

References