EECS 3401: Report 5 Documentation

Camillo John (CJ) D'Alimonte
(212754396 - cjdal34)
&
Dinesh Kalia
(213273420 - dinesh49)

December 7, 2015

Contents

1 Abstract

This report summarizes the methodologies used to complete the series of tasks provided in the *Report 5 Specification.* It provides a detailed analysis as to the reasoning behind certain chosen heuristics, the successful implementation of various predicates as well as any difficulties encountered throughout the assigned tasks.

2 Introduction

The assigned series of exercises were geared towards achieving the following learning objectives.

- The frame representations of expert/knowledge based systems
- The computation of evidence sets for Bayesian networks
- The computation of Bayesian network probabilities
- The representation of Boolean and Multistate Bayesian networks

Both practical (programming-based) as well as theoretical (explanatory-based) exercises were given in an attempt to meet these learning outcomes.

3 Exercise 1: Expert Systems

Given the inheritance structure for various figures, as shown in Figure 1, and the attribute-value pairs, as shown in Table 1, the goal was to construct a frame representation for the figures and their attributes by creating the indicated instances of the leaf types. The appropriate **side_count**, **boundary_length**, and **area** predicates had to be utilized. Each instance of the four-sided polygons needed a length specified for each of the sides, as shown in Figure 2.

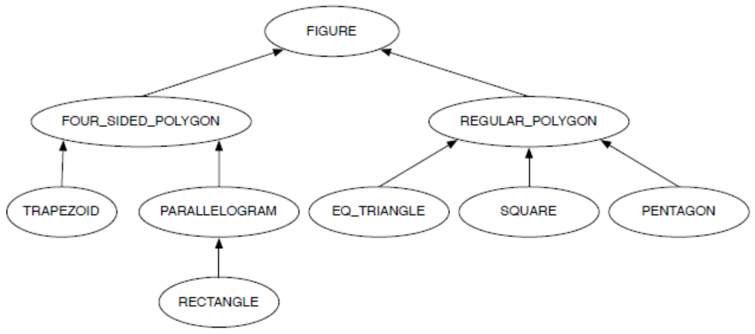


Figure 1: Inheritance diagrams for a collection of figures

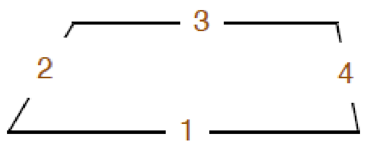
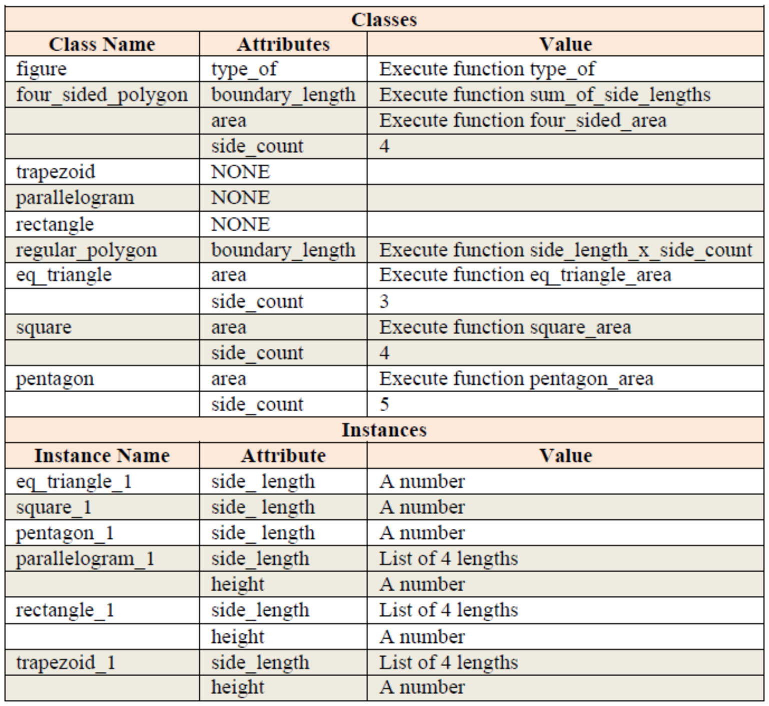


Figure 2: Order of side lengths for four-sided polygons. Sides 1 and 3 are parallel lines.



Classes		
Class Name	**Attributes**	**Value**
figure	type_of	Execute function type_of
four_sided_polygon	boundary_length	Execute function sum_of_side_lengths
	area	Execute function four_sided_area
	side_count	4
trapezoid	NONE	
parallelogram	NONE	
rectangle	NONE	
regular_polygon	boundary_length	Execute function side_length_x_side_count
eq_triangle	area	Execute function eq_triangle_area
	side_count	3
square	area	Execute function square_area
	side_count	4
pentagon	area	Execute function pentagon_area
	side_count	5
Instances		
Instance Name	**Attribute**	**Value**
eq_triangle_1	side_ length	A number
square_1	side_ length	A number
pentagon_1	side_ length	A number
parallelogram_1	side_length	List of 4 lengths
	height	A number
rectangle_1	side_length	List of 4 lengths
	height	A number
trapezoid_1	side_length	List of 4 lengths
	height	A number

Table 1: Attribute-value pairs. Instances are to have the given names

3.1 Assumptions

There were no assumptions made for this particular exercise. All instructions were clear.

3.2 Design & Implementation

The implementation process for this specific exercise was rather routine. The only misunderstanding was the fact that there was an assumption that **execute** was a predicate when in fact it is actually a functor of a compound term used to distinguish a literal slot value and a value that needed to be computed.

The areas of the **four_sided_polygon** were calculated in one predicate using an or statement with the continuity of the program dependent on the use of cut. Given the formulas to calculate these areas, as shown below, programming these calculations proved to be simple. The code for such calculations is shown below.

Area of Rectangle: $A = lw$
Area of Trapezoid: $A = \frac{a+b}{2}h$
Area of Parallelogram: $A = bh$

```
four_sided_area(Object, Value) :-
        (       parent(Object, P),
                (P = 'rectangle' ; P = 'parallelogram'),
                value(Object, height, Height),
                value(Object, side_length, [Side_1|_]),
                Value is Height*Side_1
        ),!
        ;
        (       parent(Object, P),
                P = 'trapezoid',
                value(Object, height, Height),
                value(Object, side_length, [Side_1, _, Side_3|_]),
                Value is (Side_3+Side_1)*Height/2
        ).
```

Listing 1: **four_sided_area** predicate

The calculations for the areas of **regular_polygon** figures were divided into three different predicates; one for each specific feature. This was done as none of the shapes shared a similar feature, unlike the relationship between parallelograms and rectangles. The formulas for the areas of these figures are provided below as well as one example of an area predicate.

Area of Square: $A = a^2$
Area of Pentagon: $A = \frac{1}{4}\sqrt{5(5+2\sqrt{5})}a^2$
Area of Equilateral Triangle: $A = \frac{\sqrt{3}}{4}a^2$

The calculation of the area of each **regular_polygon** was similar to the **pentagon_area** predicate as shown below.

```
pentagon_area(Object, Value) :-
        value(Object, side_length, Side_length),
        Value is sqrt(5*(5+2*sqrt(5)))*Side_length*Side_length/4.
```

Listing 2: **pentagon_area** predicate

The side lengths and heights of the shapes were randomly chosen. In this case, the **side_length** of the pentagon was 9. Overall, the hard-coding of the various features and attributes was straightforward and there was no real algorithmic reasoning behind the program. Rather, the work was tedious and self-explanatory.

3.3 Testing

Some basic test cases were used to determine whether or not the areas of each of the 6 different polygons were indeed calculated correctly. As with the case of developing the program, there was no considerable algorithmic challenge to writing these test cases. It was all a matter of determining if Prolog calculated the same values as the ones obtained through manual calculation. All test cases passed as expected.

```
:- begin_tests(e1).
test(square) :- value(square_1, area, 16).
test(equilateraltriangle) :- value(eq_triangle_1, area, 15.588457268119896).
test(pentagon) :- value(pentagon_1, area, 139.35866944770632).
test(parallelogram) :- value(parallelogram_1, area, 20).
test(rectangle) :- value(rectangle_1, area, 24).
test(trapezoid) :- value(trapezoid_1, area, 24).
:- end_tests(e1).
```

Listing 3: Test Cases for Exercise 1

3.4 Difficulties Encountered

There were no major difficulties encountered during this exercise. Although extra care had to be used to ensure the calculations were done correctly, the process was in itself routine.

4 Exercise 2: Evidence Sets and D-Separation

Given the Bayesian network in Figure 1, the goal was to find all evidence sets that d-separate node A and node H. A justification based on the three conditions that block the connection path between nodes was needed for each set.

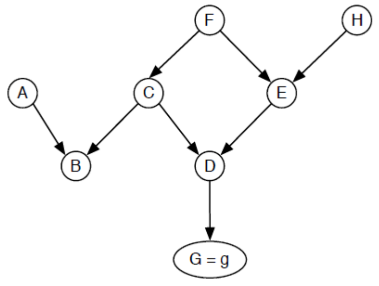


Figure 3: Bayesian network for the evidence set exercise

4.1 Assumptions

The assumption was made that the statement "all evidence sets" refers only to *non-trivial* sets of evidence nodes. Without this assumption, there would exist far too many sets to reasonably justify and the use of these "extra" nodes would not assist in solving this exercise in any way. An assumption was also made to exclude nodes A and H in the sets since it would be meaningless to include them.

4.2 Solution

The evidence sets that d-separate node A and node H are the following:

- $\{\mathbf{B}, \mathbf{C}\}$
- $\{\mathbf{B}, \mathbf{F}, \mathbf{E}\}$

4.3 Justification

$\{\mathbf{B}, \mathbf{C}\}$: Evidence is entered at A and it affects B. Since B is part of a converging d-connection triad of nodes with A and C as parents, according to the common effect condition, any evidence at B results in evidence transmitted to its parents, A and C. Therefore the evidence propagated from A to B influences C. C is a part of a diverging d-connection triad of nodes where B and D

share a common parent in C. According to the common cause condition, the evidence from B to D is blocked when C has hard evidence. Although C has not been explicitly hard-coded, it is in a serial connection with D and G. According to the casual trial condition, a serial connection from C to G is blocked when there is hard evidence along the path. Since G is hard-coded, the path to C becomes blocked. As C implicitly becomes hard evidence, the divergence connection between B and C fails and thus the connection is blocked. Therefore, the d-connection is broken and there exists a d-separation from node A and node H along the evidence set $\{\mathbf{B}, \mathbf{C}\}$.

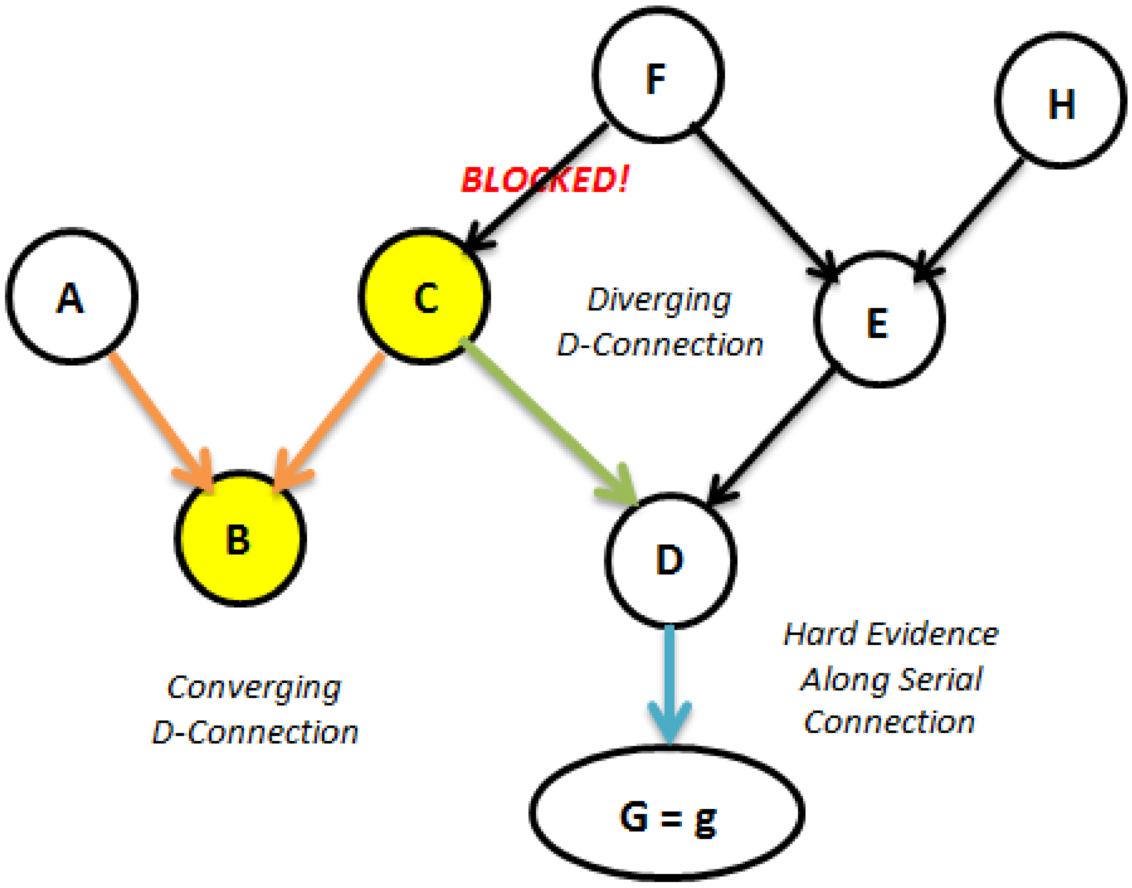


Figure 4: Diagram for the $\{\mathbf{B}, \mathbf{C}\}$ Evidence Set

$\{\mathbf{B}, \mathbf{F}, \mathbf{E}\}$: Evidence is entered at A and it affects B. Since B is part of a converging d-connection triad of nodes with A and C as parents, according to the common effect condition, any evidence at B results in evidence transmitted to its parents, A and C. Therefore the evidence propagated from A to B influences C. C is in a serial connection with F to form a path from B to F via C. According to the causal trial condition, a serial connection is only blocked if there exists hard evidence at C. This is not the case. F becomes influenced by C. F forms a diverging d-connection triad of nodes with C and E as descendants. According to the common cause condition, C and E are only blocked if F has hard evidence which in this case, it does not. Therefore E becomes influenced by F. D forms a converging d-connection triad of nodes with C and E and thus, according to common effect condition, can only maintain a path if evidence of D is found. Although G is hard-coded, according to the casual trial condition, the serial connection between C and G breaks and thus the convergence connection must be blocked as well. Since no d-connection exists, there is a d-separation from A to H.

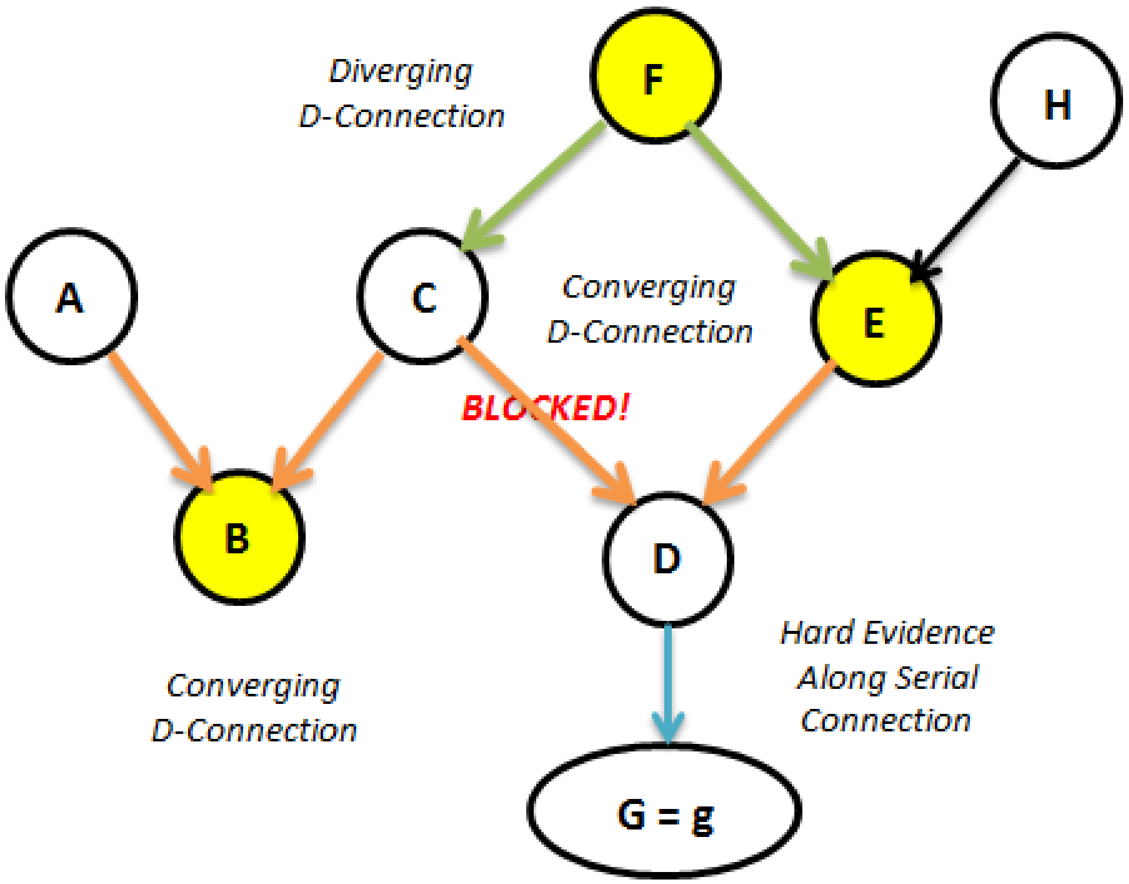


Figure 5: Diagram for the $\{\mathbf{B}, \mathbf{F}, \mathbf{E}\}$ Evidence Set

4.4 Difficulties Encountered

This exercise was both ambiguous and quite difficult as this was a relatively new concept recently learned and there had not been too many examples discussed in both the class lectures or textbook.

5 Exercise 3: Medical Diagnosis Bayesian Network

Given the Bayesian network shown in Figure 6, the goal was to determines the probability of a patient having tuberculosis, lung cancer or bronchitis. There were two causal factors — smoking and whether the patient had been to Asia recently. There were two additional pieces of evidence available at our disposal — whether the patient was suffering from shortness of breath (dyspnoea) or whether a positive, or negative X-ray test result was available.

The overall objective of this exercise was to compute the following probabilities in Prolog and by hand using the node probability tables as shown in Figure 7.

$P(dyspnoea) =?$
$P(smoker \mid tuberculosis\_or\_cancer) =?$

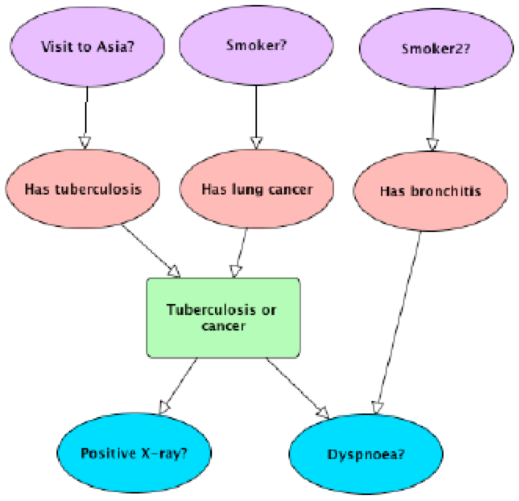


Figure 6: Bayesian network describing tuberculosis or cancer diagnosis

5.1 Design & Implementation

The first part in implementing this exercise was the need to establish the parent and child relationship from the given Bayesian network. Using the **parent** predicate, the relationships were hard-coded in as follows.

```
parent(asia, tuberculosis).
parent(smoker, lungCancer).
parent(smoker2, bronchitis).
parent(tuberculosis, tuberculosisOrCancer).
parent(lungCancer, tuberculosisOrCancer).
parent(tuberculosisOrCancer, positiveXray).
parent(tuberculosisOrCancer, dyspnoea).
parent(bronchitis, dyspnoea).
```

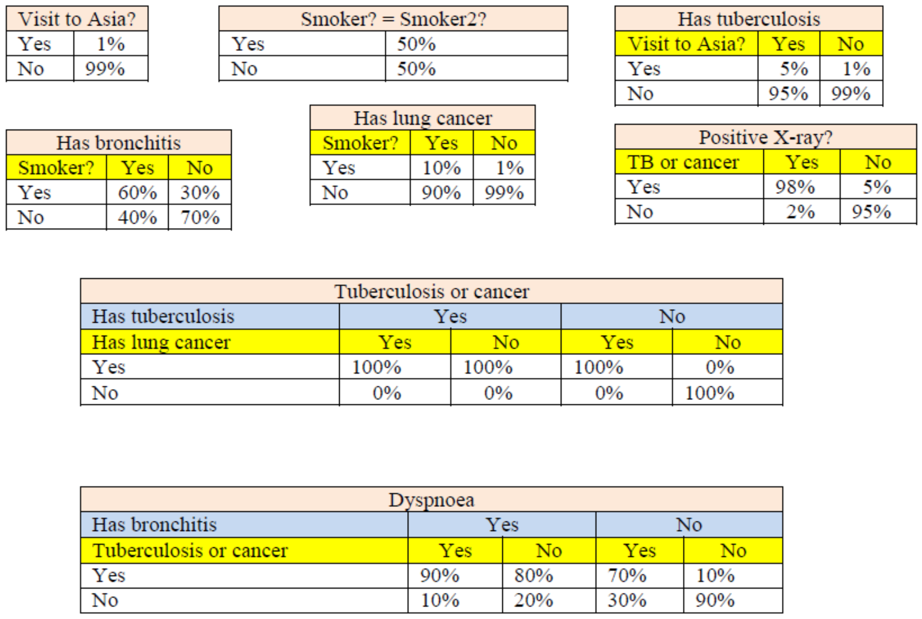
Listing 4: Parent and Descendant Relationships

Once the relationships were established, the probability of each specific event had to be hard-coded into the program. Using the provided node probability tables, as shown in Figure 7, the correct corresponding probability was assigned to each event. The probabilities were calculated directly from the table provided or in some cases, carefully deduced from related facts. The hard-coding of these facts is seen below.

```
p(asia, 0.01).
p(smoker, 0.5).
p(smoker2, 0.5).
p(tuberculosis, [asia], 0.05).
p(tuberculosis, [~asia], 0.01).
```

```
p(lungCancer, [smoker], 0.1).
p(lungCancer, [~smoker], 0.01).
p(bronchitis, [smoker2], 0.6).
p(bronchitis, [~smoker2], 0.3).
p(tuberculosisOrCancer, [tuberculosis, lungCancer], 1).
p(tuberculosisOrCancer, [tuberculosis, ~lungCancer], 1).
p(tuberculosisOrCancer, [~tuberculosis, lungCancer], 1).
p(tuberculosisOrCancer, [~tuberculosis, ~lungCancer], 0.0).
p(positiveXray, [tuberculosisOrCancer], 0.98).
p(positiveXray, [~tuberculosisOrCancer], 0.05).
p(dyspnoea, [tuberculosisOrCancer, bronchitis], 0.9).
p(dyspnoea, [tuberculosisOrCancer, ~bronchitis], 0.7).
p(dyspnoea, [~tuberculosisOrCancer, bronchitis], 0.8).
p(dyspnoea, [~tuberculosisOrCancer, ~bronchitis], 0.1).
```

Listing 5: Probabilities of Certain Events



Visit to Asia?	
Yes	1%
No	99%

Smoker? = Smoker2?	
Yes	50%
No	50%

Has tuberculosis		
Visit to Asia?	Yes	No
Yes	5%	1%
No	95%	99%

Has bronchitis		
Smoker?	Yes	No
Yes	60%	30%
No	40%	70%

Has lung cancer		
Smoker?	Yes	No
Yes	10%	1%
No	90%	99%

Positive X-ray?		
TB or cancer	Yes	No
Yes	98%	5%
No	2%	95%

Tuberculosis or cancer				
Has tuberculosis	Yes		No	
Has lung cancer	Yes	No	Yes	No
Yes	100%	100%	100%	0%
No	0%	0%	0%	100%

Dyspnoea				
Has bronchitis	Yes		No	
Tuberculosis or cancer	Yes	No	Yes	No
Yes	90%	80%	70%	10%
No	10%	20%	30%	90%

Figure 7: Node probability tables for the model in Figure 6

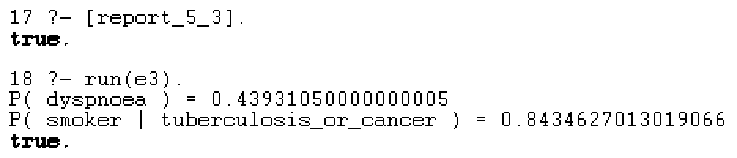
5.2 Prolog Results

Since the values were correctly stored, the probabilities could be calculated. In order to obtain the results from Prolog, a simple **run** predicate was built to output the findings. The **run** predicate is included below:

```
run(e3) :-
prob([ dyspnoea], [ ], Probability1),
prob([ smoker], [ tuberculosisOrCancer], Probability2),
write('P( dyspnoea ) = '), write(Probability1), nl,
write('P( smoker | tuberculosis_or_cancer ) = '), write(Probability2).
```

Listing 6: **run** predicate

The resulting output is shown below. Therefore, according to the Prolog program:



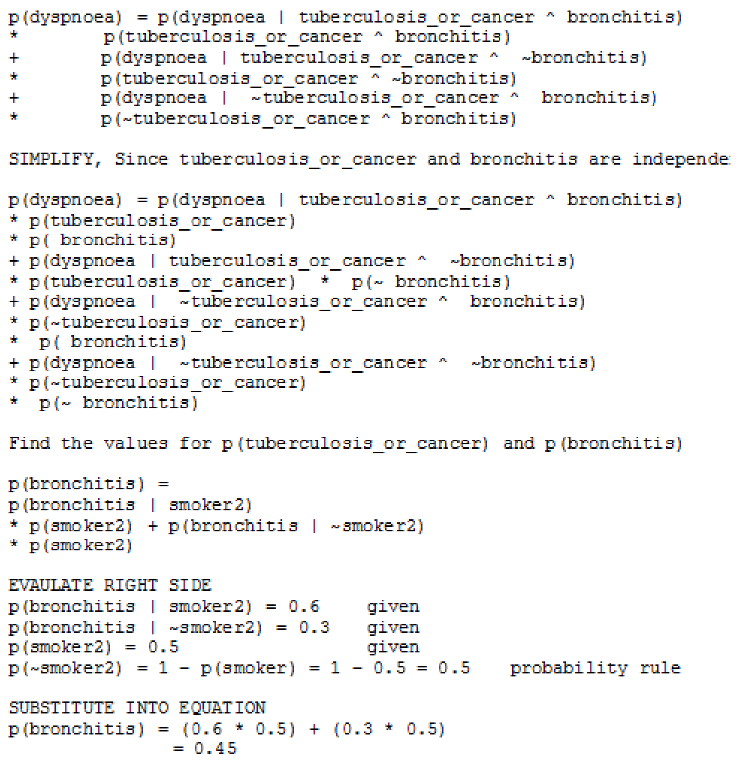
```
17 ?- [report_5_3].
true.

18 ?- run(e3).
P( dyspnoea ) = 0.43931050000000005
P( smoker | tuberculosis_or_cancer ) = 0.8434627013019066
true.
```

Figure 8: Terminal Output

$P(dyspnoea) = 0.43931050$
$P(smoker\,|\,tuberculosis\_or\_cancer) = 0.8434627013019066$

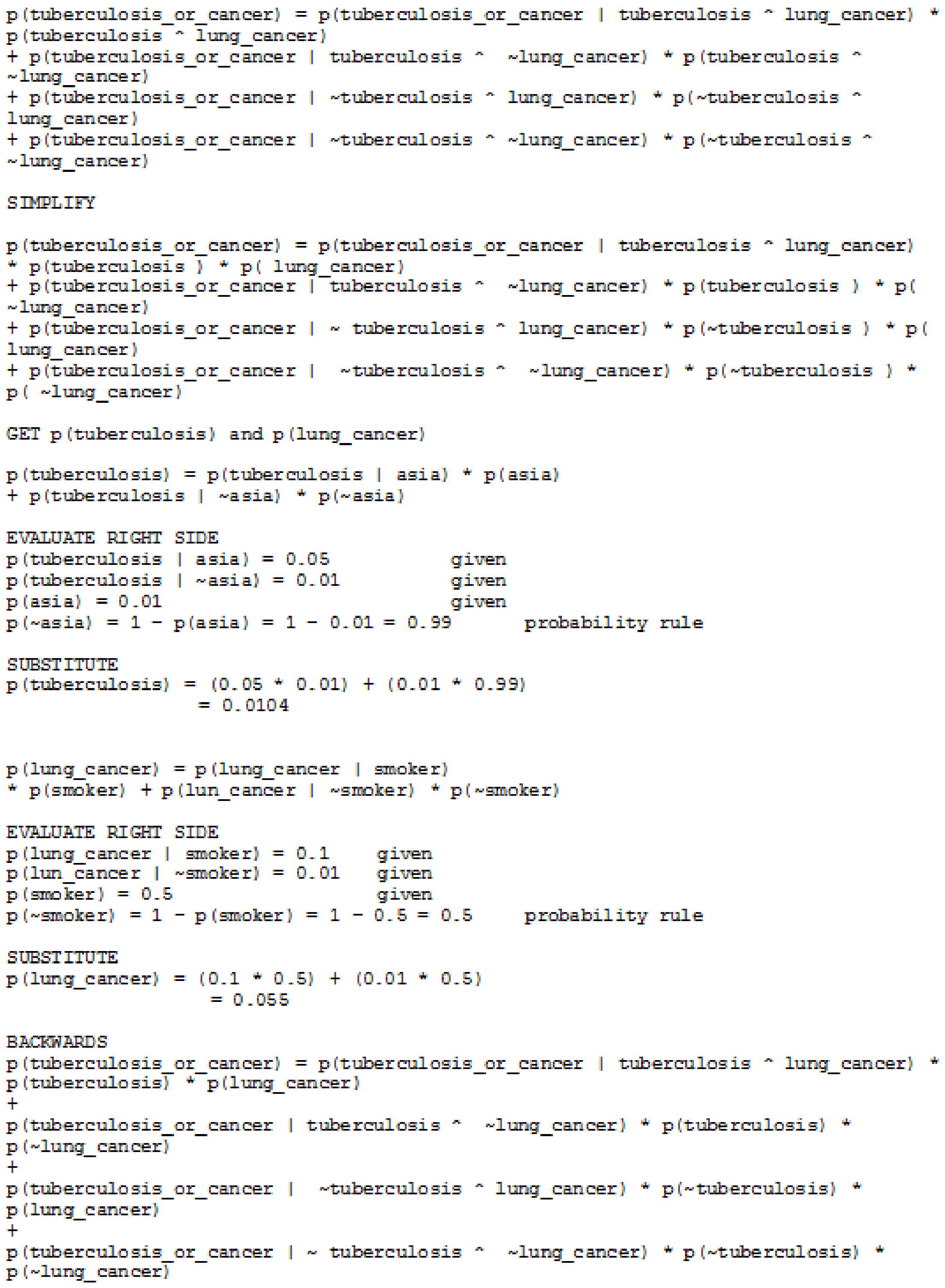
5.3 Manual Calculations

$P(dyspnoea):$



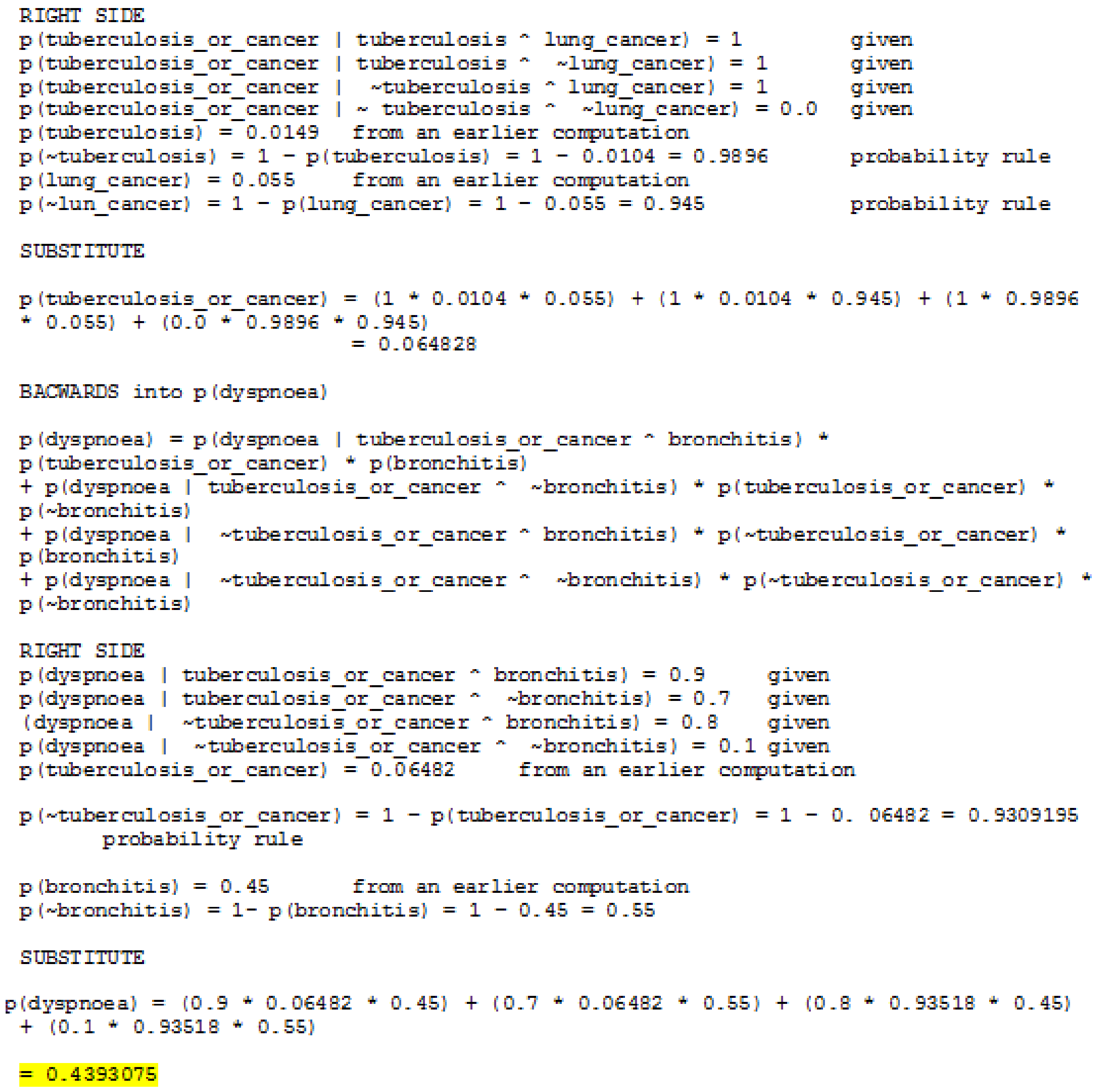
```
p(dyspnoea)  = p(dyspnoea | tuberculosis_or_cancer ^ bronchitis)
*        p(tuberculosis_or_cancer ^ bronchitis)
+        p(dyspnoea | tuberculosis_or_cancer ^  ~bronchitis)
*        p(tuberculosis_or_cancer ^ ~bronchitis)
+        p(dyspnoea |  ~tuberculosis_or_cancer ^  bronchitis)
*        p(~tuberculosis_or_cancer ^ bronchitis)

SIMPLIFY, Since tuberculosis_or_cancer and bronchitis are independe

p(dyspnoea)  = p(dyspnoea | tuberculosis_or_cancer ^ bronchitis)
* p(tuberculosis_or_cancer)
* p( bronchitis)
+ p(dyspnoea | tuberculosis_or_cancer ^  ~bronchitis)
* p(tuberculosis_or_cancer)  *  p(~ bronchitis)
+ p(dyspnoea |  ~tuberculosis_or_cancer ^  bronchitis)
* p(~tuberculosis_or_cancer)
*  p( bronchitis)
+ p(dyspnoea |  ~tuberculosis_or_cancer ^  ~bronchitis)
* p(~tuberculosis_or_cancer)
*  p(~ bronchitis)

Find the values for p(tuberculosis_or_cancer) and p(bronchitis)

p(bronchitis) =
p(bronchitis | smoker2)
* p(smoker2) + p(bronchitis | ~smoker2)
* p(smoker2)

EVAULATE RIGHT SIDE
p(bronchitis | smoker2) = 0.6     given
p(bronchitis | ~smoker2) = 0.3    given
p(smoker2) = 0.5                  given
p(~smoker2)  = 1 - p(smoker) = 1 - 0.5 = 0.5    probability rule

SUBSTITUTE INTO EQUATION
p(bronchitis) = (0.6 * 0.5) + (0.3 * 0.5)
                = 0.45
```

Figure 9: Manual Calculations for $P(dyspnoea)$ - Part 1



```
p(tuberculosis_or_cancer) = p(tuberculosis_or_cancer | tuberculosis ^ lung_cancer) *
p(tuberculosis ^ lung_cancer)
+ p(tuberculosis_or_cancer | tuberculosis ^  ~lung_cancer) * p(tuberculosis ^
~lung_cancer)
+ p(tuberculosis_or_cancer | ~tuberculosis ^ lung_cancer) * p(~tuberculosis ^
lung_cancer)
+ p(tuberculosis_or_cancer | ~tuberculosis ^ ~lung_cancer) * p(~tuberculosis ^
~lung_cancer)

SIMPLIFY

p(tuberculosis_or_cancer) = p(tuberculosis_or_cancer | tuberculosis ^ lung_cancer)
* p(tuberculosis ) * p( lung_cancer)
+ p(tuberculosis_or_cancer | tuberculosis ^  ~lung_cancer) * p(tuberculosis ) * p(
~lung_cancer)
+ p(tuberculosis_or_cancer | ~ tuberculosis ^ lung_cancer) * p(~tuberculosis ) * p(
lung_cancer)
+ p(tuberculosis_or_cancer |  ~tuberculosis ^  ~lung_cancer) * p(~tuberculosis ) *
p( ~lung_cancer)

GET p(tuberculosis) and p(lung_cancer)

p(tuberculosis) = p(tuberculosis | asia) * p(asia)
+ p(tuberculosis | ~asia) * p(~asia)

EVALUATE RIGHT SIDE
p(tuberculosis | asia) = 0.05          given
p(tuberculosis | ~asia) = 0.01         given
p(asia) = 0.01                         given
p(~asia) = 1 - p(asia) = 1 - 0.01 = 0.99      probability rule

SUBSTITUTE
p(tuberculosis) = (0.05 * 0.01) + (0.01 * 0.99)
                 = 0.0104


p(lung_cancer) = p(lung_cancer | smoker)
* p(smoker) + p(lun_cancer | ~smoker) * p(~smoker)

EVALUATE RIGHT SIDE
p(lung_cancer | smoker) = 0.1    given
p(lun_cancer | ~smoker) = 0.01   given
p(smoker) = 0.5                  given
p(~smoker) = 1 - p(smoker) = 1 - 0.5 = 0.5     probability rule

SUBSTITUTE
p(lung_cancer) = (0.1 * 0.5) + (0.01 * 0.5)
                  = 0.055

BACKWARDS
p(tuberculosis_or_cancer) = p(tuberculosis_or_cancer | tuberculosis ^ lung_cancer) *
p(tuberculosis) * p(lung_cancer)
+
p(tuberculosis_or_cancer | tuberculosis ^  ~lung_cancer) * p(tuberculosis) *
p(~lung_cancer)
+
p(tuberculosis_or_cancer |  ~tuberculosis ^ lung_cancer) * p(~tuberculosis) *
p(lung_cancer)
+
p(tuberculosis_or_cancer | ~ tuberculosis ^  ~lung_cancer) * p(~tuberculosis) *
p(~lung_cancer)
```

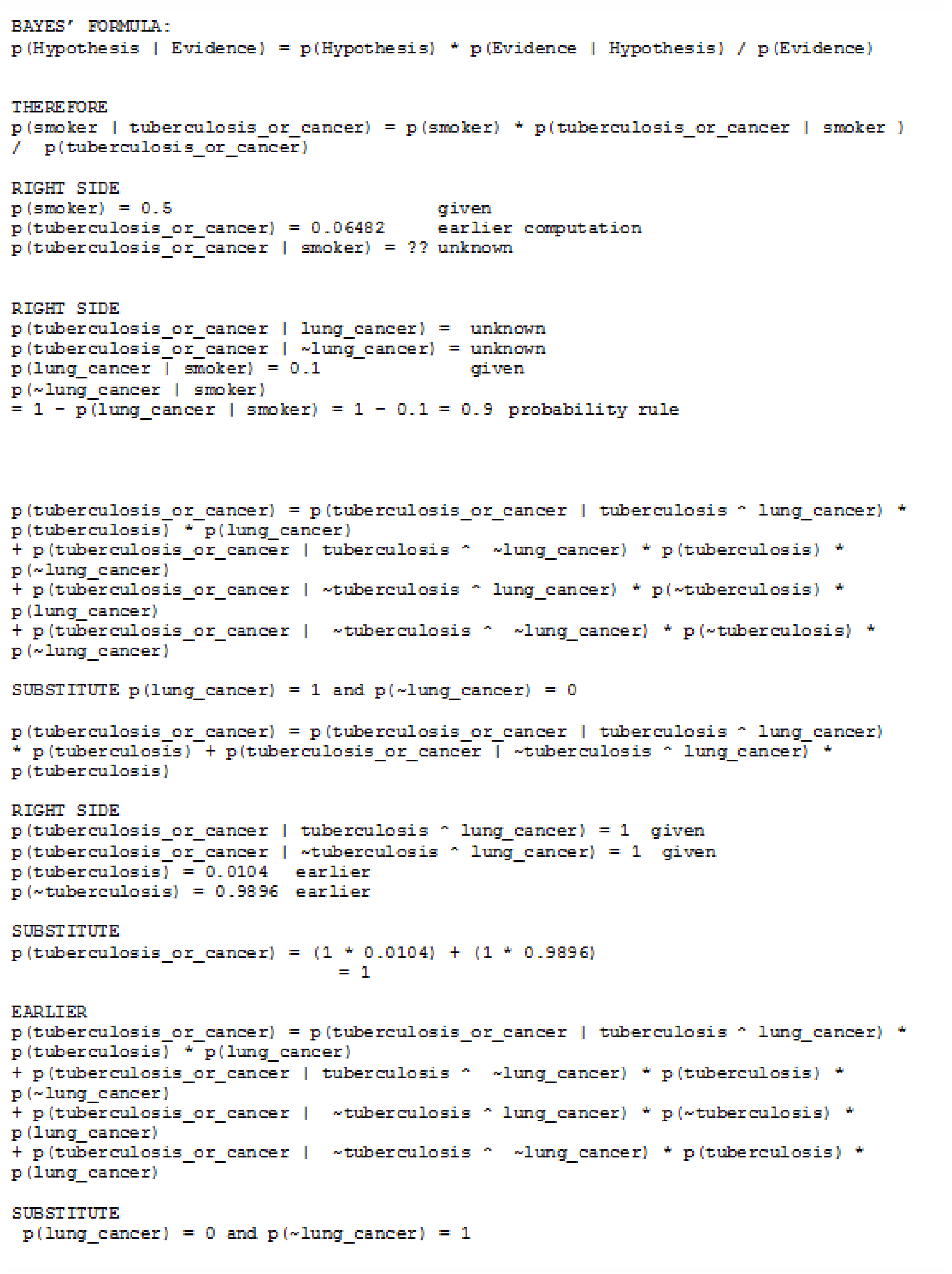
Figure 10: Manual Calculations for $P(dyspnoea)$ - Part 2



```
RIGHT SIDE
p(tuberculosis_or_cancer | tuberculosis ^ lung_cancer) = 1          given
p(tuberculosis_or_cancer | tuberculosis ^  ~lung_cancer) = 1        given
p(tuberculosis_or_cancer |  ~tuberculosis ^ lung_cancer) = 1        given
p(tuberculosis_or_cancer | ~ tuberculosis ^  ~lung_cancer) = 0.0    given
p(tuberculosis) = 0.0149   from an earlier computation
p(~tuberculosis) = 1 - p(tuberculosis) = 1 - 0.0104 = 0.9896        probability rule
p(lung_cancer) = 0.055     from an earlier computation
p(~lun_cancer) = 1 - p(lung_cancer) = 1 - 0.055 = 0.945             probability rule

SUBSTITUTE

p(tuberculosis_or_cancer) = (1 * 0.0104 * 0.055) + (1 * 0.0104 * 0.945) + (1 * 0.9896
* 0.055) + (0.0 * 0.9896 * 0.945)
                           = 0.064828

BACWARDS into p(dyspnoea)

p(dyspnoea) = p(dyspnoea | tuberculosis_or_cancer ^ bronchitis) *
p(tuberculosis_or_cancer) * p(bronchitis)
+ p(dyspnoea | tuberculosis_or_cancer ^  ~bronchitis) * p(tuberculosis_or_cancer) *
p(~bronchitis)
+ p(dyspnoea |  ~tuberculosis_or_cancer ^ bronchitis) * p(~tuberculosis_or_cancer) *
p(bronchitis)
+ p(dyspnoea |  ~tuberculosis_or_cancer ^  ~bronchitis) * p(~tuberculosis_or_cancer) *
p(~bronchitis)

RIGHT SIDE
p(dyspnoea | tuberculosis_or_cancer ^ bronchitis) = 0.9     given
p(dyspnoea | tuberculosis_or_cancer ^  ~bronchitis) = 0.7   given
(dyspnoea |  ~tuberculosis_or_cancer ^ bronchitis) = 0.8    given
p(dyspnoea |  ~tuberculosis_or_cancer ^  ~bronchitis) = 0.1 given
p(tuberculosis_or_cancer) = 0.06482     from an earlier computation

p(~tuberculosis_or_cancer) = 1 - p(tuberculosis_or_cancer) = 1 - 0. 06482 = 0.9309195
       probability rule

p(bronchitis) = 0.45       from an earlier computation
p(~bronchitis) = 1- p(bronchitis) = 1 - 0.45 = 0.55

SUBSTITUTE

p(dyspnoea) = (0.9 * 0.06482 * 0.45) + (0.7 * 0.06482 * 0.55) + (0.8 * 0.93518 * 0.45)
 + (0.1 * 0.93518 * 0.55)

= 0.4393075
```

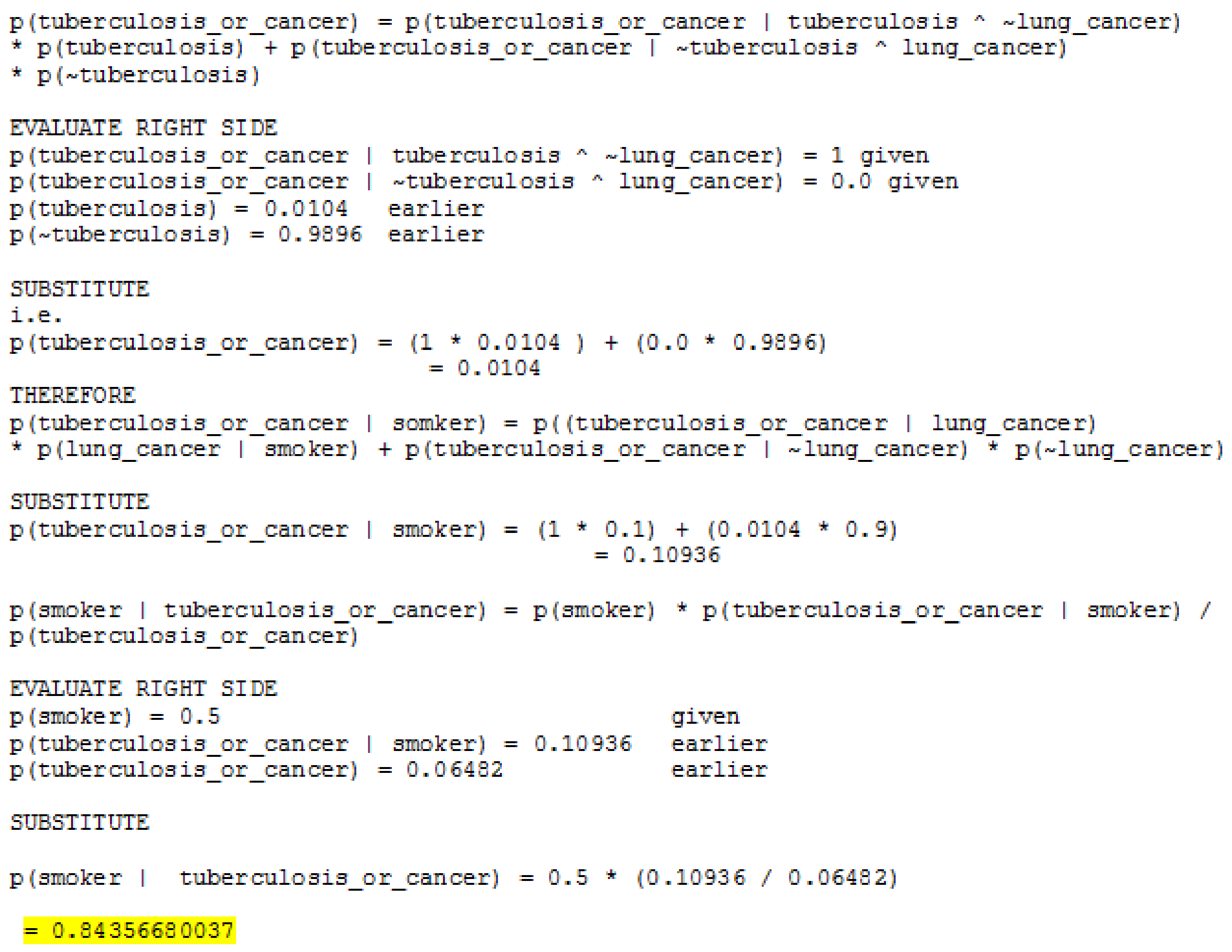
Figure 11: Manual Calculations for $P(dyspnoea)$ - Part 3

$P(smoker\,|\,tuberculosis\_or\_cancer)$:



```
BAYES' FORMULA:
p(Hypothesis | Evidence) = p(Hypothesis) * p(Evidence | Hypothesis) / p(Evidence)


THEREFORE
p(smoker | tuberculosis_or_cancer) = p(smoker) * p(tuberculosis_or_cancer | smoker )
/  p(tuberculosis_or_cancer)

RIGHT SIDE
p(smoker) = 0.5                         given
p(tuberculosis_or_cancer) = 0.06482     earlier computation
p(tuberculosis_or_cancer | smoker) = ?? unknown


RIGHT SIDE
p(tuberculosis_or_cancer | lung_cancer) =  unknown
p(tuberculosis_or_cancer | ~lung_cancer) = unknown
p(lung_cancer | smoker) = 0.1              given
p(~lung_cancer | smoker)
= 1 - p(lung_cancer | smoker) = 1 - 0.1 = 0.9  probability rule




p(tuberculosis_or_cancer) = p(tuberculosis_or_cancer | tuberculosis ^ lung_cancer) *
p(tuberculosis) * p(lung_cancer)
+ p(tuberculosis_or_cancer | tuberculosis ^  ~lung_cancer) * p(tuberculosis) *
p(~lung_cancer)
+ p(tuberculosis_or_cancer | ~tuberculosis ^ lung_cancer) * p(~tuberculosis) *
p(lung_cancer)
+ p(tuberculosis_or_cancer |  ~tuberculosis ^  ~lung_cancer) * p(~tuberculosis) *
p(~lung_cancer)

SUBSTITUTE p(lung_cancer) = 1 and p(~lung_cancer) = 0

p(tuberculosis_or_cancer) = p(tuberculosis_or_cancer | tuberculosis ^ lung_cancer)
* p(tuberculosis) + p(tuberculosis_or_cancer | ~tuberculosis ^ lung_cancer) *
p(tuberculosis)

RIGHT SIDE
p(tuberculosis_or_cancer | tuberculosis ^ lung_cancer) = 1  given
p(tuberculosis_or_cancer | ~tuberculosis ^ lung_cancer) = 1  given
p(tuberculosis) = 0.0104   earlier
p(~tuberculosis) = 0.9896  earlier

SUBSTITUTE
p(tuberculosis_or_cancer) = (1 * 0.0104) + (1 * 0.9896)
                              = 1

EARLIER
p(tuberculosis_or_cancer) = p(tuberculosis_or_cancer | tuberculosis ^ lung_cancer) *
p(tuberculosis) * p(lung_cancer)
+ p(tuberculosis_or_cancer | tuberculosis ^  ~lung_cancer) * p(tuberculosis) *
p(~lung_cancer)
+ p(tuberculosis_or_cancer |  ~tuberculosis ^ lung_cancer) * p(~tuberculosis) *
p(lung_cancer)
+ p(tuberculosis_or_cancer |  ~tuberculosis ^  ~lung_cancer) * p(tuberculosis) *
p(lung_cancer)

SUBSTITUTE
 p(lung_cancer) = 0 and p(~lung_cancer) = 1
```

Figure 12: Manual Calculations for $P(smoker\,|\,tuberculosis\_or\_cancer)$ - Part 1



```
p(tuberculosis_or_cancer) = p(tuberculosis_or_cancer | tuberculosis ^ ~lung_cancer)
* p(tuberculosis) + p(tuberculosis_or_cancer | ~tuberculosis ^ lung_cancer)
* p(~tuberculosis)

EVALUATE RIGHT SIDE
p(tuberculosis_or_cancer | tuberculosis ^ ~lung_cancer) = 1 given
p(tuberculosis_or_cancer | ~tuberculosis ^ lung_cancer) = 0.0 given
p(tuberculosis) = 0.0104   earlier
p(~tuberculosis) = 0.9896  earlier

SUBSTITUTE
i.e.
p(tuberculosis_or_cancer) = (1 * 0.0104 ) + (0.0 * 0.9896)
                             = 0.0104
THEREFORE
p(tuberculosis_or_cancer | somker) = p((tuberculosis_or_cancer | lung_cancer)
* p(lung_cancer | smoker) + p(tuberculosis_or_cancer | ~lung_cancer) * p(~lung_cancer)

SUBSTITUTE
p(tuberculosis_or_cancer | smoker) = (1 * 0.1) + (0.0104 * 0.9)
                                         = 0.10936

p(smoker | tuberculosis_or_cancer) = p(smoker) * p(tuberculosis_or_cancer | smoker) /
p(tuberculosis_or_cancer)

EVALUATE RIGHT SIDE
p(smoker) = 0.5                             given
p(tuberculosis_or_cancer | smoker) = 0.10936   earlier
p(tuberculosis_or_cancer) = 0.06482            earlier

SUBSTITUTE

p(smoker |  tuberculosis_or_cancer) = 0.5 * (0.10936 / 0.06482)

 = 0.84356680037
```

Figure 13: Manual Calculations for $P(smoker | tuberculosis\_or\_cancer)$ - Part 2

6 Exercise 4: Monty Hall Dilemma Bayesian Network

Similar to that of Exercise 3, a Bayesian network, as shown in Figure 14, was given in an attempt to solve a simplified version of the Monty Hall dilemma. The goal was to calculate the probability of $P(prize\_door = green | door\_shown = blue, picked\_door = red)$.

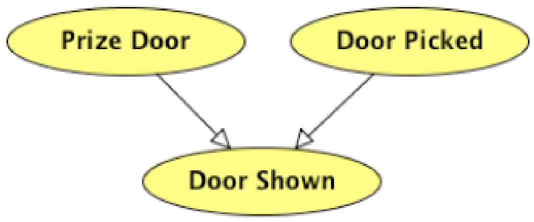


Figure 14: Monty Hall Dilemma Bayesian Network

6.1 Design & Implementation

This exercise reinforced the programming concepts utilized in the previous exercise. The only difference in this particular question was the considering of multiple states. The difficultly in writing the code remained the same and the overall process to which the results were obtained did not change from Exercise 3 to Exercise 4.

The first part in implementing this exercise was the need to establish the parent and child relationship from the given Bayesian network. Using the **parent** predicate, the relationships were hard-coded in as follows.

```
parent(prizeDoor, doorShown).
parent(doorPicked, doorShown).
```

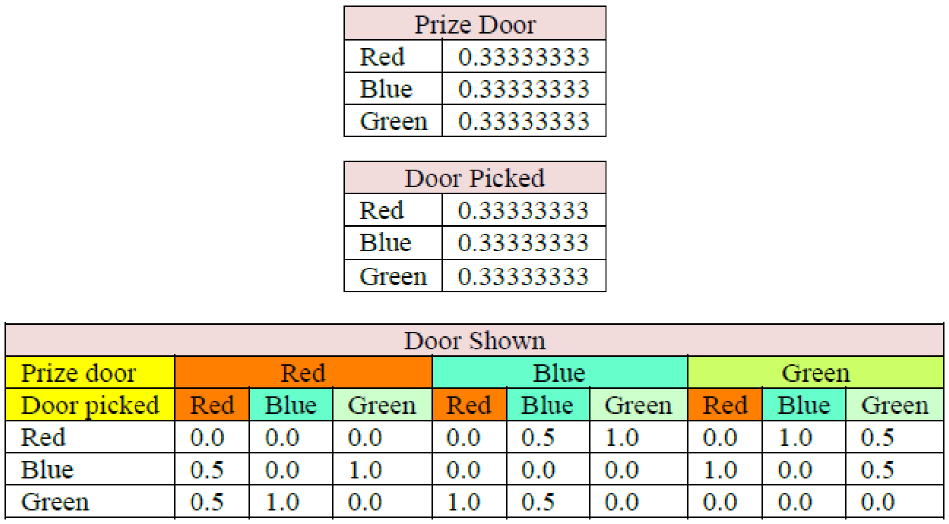
Listing 7: Parent and Descendant Relationships

Once the relationships were established, the probability of each specific event had to be hard-coded into the program. Notice how there are many more probabilities to consider in this exercise compared to the last exercise. This is a direct result of the use of multiple states. Using the provided node probability tables, as shown in Figure 15, the correct corresponding probability was assigned to each event. The probabilities were calculated directly from the table provided or in some cases, carefully deduced from related facts. The hard-coding of these facts is seen below.

```
p(prizeDoor=red, 0.333).
p(prizeDoor=blue, 0.333).
p(prizeDoor=green, 0.333).
p(doorPicked=red, 0.333).
p(doorPicked=blue, 0.333).
p(doorPicked=green, 0.333).

p(doorShown=red, [prizeDoor=red, doorPicked=red], 0.0).
p(doorShown=red, [prizeDoor=red, doorPicked=blue],  0.0).
p(doorShown=red, [prizeDoor=red,  doorPicked=green], 0.0).
p(doorShown=blue, [prizeDoor=red, doorPicked=red], 0.5).
p(doorShown=blue, [prizeDoor=red, doorPicked=blue],  0.0).
p(doorShown=blue, [prizeDoor=red,  doorPicked=green], 0.0).
p(doorShown=green, [prizeDoor=red, doorPicked=red], 0.5).
p(doorShown=green, [prizeDoor=red, doorPicked=blue],  1.0).
p(doorShown=green, [prizeDoor=red,  doorPicked=green], 0.0).

p(doorShown=red, [prizeDoor=blue, doorPicked=red], 0.0).
p(doorShown=red, [prizeDoor=blue, doorPicked=blue],  0.5).
p(doorShown=red, [prizeDoor=blue,  doorPicked=green], 1.0).
p(doorShown=blue, [prizeDoor=blue, doorPicked=red], 0.0).
p(doorShown=blue, [prizeDoor=blue, doorPicked=blue],  0.0).
p(doorShown=blue, [prizeDoor=blue,  doorPicked=green], 0.0).
p(doorShown=green, [prizeDoor=blue, doorPicked=red], 1.0).
p(doorShown=green, [prizeDoor=blue, doorPicked=blue],  0.5).
p(doorShown=green, [prizeDoor=blue,  doorPicked=green], 0.0).
```

```
p(doorShown=red, [prizeDoor=green, doorPicked=red], 0.0).
p(doorShown=red, [prizeDoor=green, doorPicked=blue],  1.0).
p(doorShown=red, [prizeDoor=green,  doorPicked=green], 0.5).
p(doorShown=blue, [prizeDoor=green, doorPicked=red], 1.0).
p(doorShown=blue, [prizeDoor=green, doorPicked=blue],  0.0).
p(doorShown=blue, [prizeDoor=green,  doorPicked=green], 0.5).
p(doorShown=green, [prizeDoor=green, doorPicked=red], 0.0).
p(doorShown=green, [prizeDoor=green, doorPicked=blue],  0.0).
p(doorShown=green, [prizeDoor=green,  doorPicked=green], 0.0).
```

Listing 8: Probabilities of Certain Events



Prize Door	
Red	0.33333333
Blue	0.33333333
Green	0.33333333

Door Picked	
Red	0.33333333
Blue	0.33333333
Green	0.33333333

Door Shown									
Prize door	Red			Blue			Green		
Door picked	Red	Blue	Green	Red	Blue	Green	Red	Blue	Green
Red	0.0	0.0	0.0	0.0	0.5	1.0	0.0	1.0	0.5
Blue	0.5	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.5
Green	0.5	1.0	0.0	1.0	0.5	0.0	0.0	0.0	0.0

Figure 15: Node probability tables for the model in Figure 9

6.2 Prolog Results

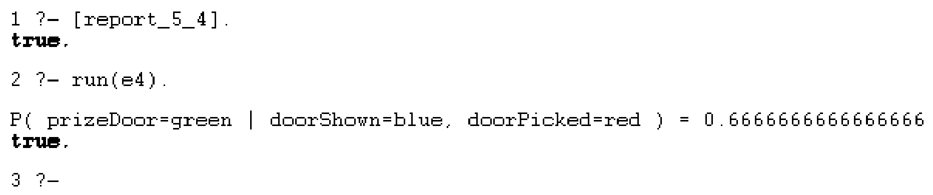
Since the values were correctly stored, the probabilities could be calculated. In order to obtain the results from Prolog, a simple **run** predicate was built to output the findings. The **run** predicate is included below:

```
run(e4) :-
prob(prizeDoor=green, [ doorShown=blue, doorPicked=red], Prob1), nl,
write('P( prizeDoor=green | doorShown=blue, doorPicked=red ) = '), write(Prob1).
```

Listing 9: **run** predicate

The resulting output is shown below. Therefore, according to the Prolog program:

$$P(prize\_door = green \,|\, door\_shown = blue, picked\_door = red) = 0.6666666666666$$



```
1 ?- [report_5_4].
true.

2 ?- run(e4).

P( prizeDoor=green | doorShown=blue, doorPicked=red ) = 0.6666666666666666
true.

3 ?-
```

Figure 16: Terminal Output

6.3 Manual Calculations

Unable to complete...the formulas became a "mess" and were extremely difficult to follow.

7 Exercise 5: Boolean and Multistate Algorithm Comparison

The following is a list of differences between the Boolean and Multistate Bayesian Algorithms. A reasoning behind each difference is included.

- There is a substantial difference in how the two files conclude that the Cond implies X is false. In Listing 10, the program checks to see if the negation of X is a member of the network and if so, cuts. The probability of ˜X is simply calculated as 1-(the probability of X). This simple process works in this case because there is only 1 state of the node to consider. On the other hand, in Listing 11, multiple states have to be considered and hence, a compound term X is formed. The compound term determines if state K, regardless of other states before or after it, is included in the network. K's state is opposite of X's and thus, Cond implies that X is false.

  ```
  prob(X, Cond, 0) :-
      member(~ X, Cond), !.

  prob(~ X, Cond, P) :- !,
      prob(X, Cond, P0),
      P is 1 - P0.
  ```

  Listing 10: f16_4_BayesianNet.pl

  ```
  prob(X, Cond, 0) :-
      X =.. [_, K, _] ,
      hasNode(K, Cond) , !.
  ```

  Listing 11: bayesianNet_multistate.pl

- The following code was added to the original definition of **predecessor** from **f16_4_BayesianNet.pl** as there needed to be a way to extract the node name from the state, variable=value, pair.

  ```
  predecessor(X, Y) :-
      X =.. [_,Xf,_], Y =.. [_,Yf, _],
      parent(Xf, Yf).

  predecessor(X, Z) :-
  ```

```
          X =.. [_,Xf,_], Z =.. [_,Zf, _],
          parent(Xf, Y),
          predecessor(Y, Zf).
```

Listing 12: bayesianNet_multistate.pl

- There is a minor difference between the two programs in their use of Bayes' rule if the condition involves a descendant of X. The **bayesianNet_multistate.pl** program has an extra clasuse that determines that if the probability of Py = 0 and **predecessor(X,Y)** is true, then the probability of P = 1. If this is not the case, then the calculation of the probability of P is the same in both programs, regardless of the number of states.

```
prob(X, Cond0, P) :-
     mydelete(Y, Cond0, Cond),
     predecessor(X, Y), !,
     prob(X, Cond, Px),
     prob(Y, [X | Cond], PyGivenX),
     prob(Y, Cond, Py),
     P is Px * PyGivenX / Py.
```

Listing 13: f16_4_BayesianNet.pl

```
prob(X, Cond0, P) :-
     mydelete(Y, Cond0, Cond),
     predecessor(X, Y), !,
     prob(X, Cond, Px),
     prob(Y, [X | Cond], PyGivenX),
     prob(Y, Cond, Py),
     ( Py = 0, P = 1, !
     ;
       P is Px * PyGivenX / Py
     ).
```

Listing 14: bayesianNet_multistate.pl

8 Conclusion

The majority of the learning objectives were met through the completion of the assigned exercises although there were specific parts that lead to some struggle. Both the second and fifth questions were difficult and the manual calculations proved to be tedious and prone to error. Overall, however, much was learned about Bayesian networks and Expert systems.

9 Appendix

This report was jointly authored. Both members worked on the outline of each question together as a pair. The programming of the predicates was jointly done. Dinesh wrote the documentation and comments for each predicate while CJ wrote the test cases for each exercise.