Assignment 4

Task 1:
S — constant — Shadow (there is a dog named Shadow)
J — constant — John
M — constant — Mary
P — constant — Smartphone
L — constant — Laptop
x, y, z — variables — Used in the following two statements.
isMale(y) — predicate — Checks if the Object y is male or female (will be used with Shadow). If male, return true. If female, return false.
Gives(x, y, z) — function — Person x gives Object y to Person z

Instead of using the quantifier symbols (unavailable on my keyboard), I will simply say FOR EVERY and THERE EXISTS (for convenience's sake).

With the above constants and functions defined, a first-order logic knowledge base can now be constructed:
1. THERE EXISTS S — constant — There is a dog called Shadow.
2. Gives(J, S, M) — function — John gave Shadow to Mary
3. isMale(S) => Gives(M, P, J) — predicate — If Shadow is male, Mary gives a smartphone to John.
4. NOT(isMale(S)) => Gives(M, L, J) — predicate — If Shadow is female, Mary gives John a laptop.
5. FOR EVERY x: Gives(J, x, M) => isMale(x) — predicate — John only gives male dogs to people.
6. Gives(M, L, J) — function — Mary gave John a laptop.

Task 2:
J — John — Same as FOL KB
B — Bill — Same as FOL KB
JTaller — John is taller than Bill — This is equivalent to "taller(John, Bill)"

Assuming there are a total number of n people:
P1, P2, P3, …, Pn — person 1, person 2, person 3, …, person n — This is meant to take the place of the x variable present in the second statement.
P1Taller, P2Taller, P3Taller, … , PnTaller — Person 1 is taller than Bill, Person 2 … , Person n is taller than Bill — Taken altogether, this is equivalent to taller(x, Bill).
P1Tall, P2Tall, P3Tall, …, PnTall — Person 1 is considered tall, Person 2 is considered tall, …, Person n is considered tall — Taken altogether, this is equivalent to tall(x)

With the above defined, we can define statements for the propositional logic version of the knowledge base:
1. JTaller — John is taller than Bill.

2. P1Taller => P1Tall — If Person 1 is taller than Bill, they must be considered tall
P2Taller => P2Tall — If Person 2 is taller than Bill, they must be considered tall
P3Taller => P3Tall — If Person 3 is taller than Bill, they must be considered tall
…
…
…
PnTaller => PnTall — If Person n is taller than Bill, they must be considered tall.

The above group of statements as a whole is equivalent to (FOR EVERY x) taller(x, Bill) => tall(x).

Task 3:
Programming assignment

Task 4:
Below is what would go in the facts file (initial state/preconds, goal/effects):

```
(adult1 Object)
(adult2 Object)
(child1 Object)
(child2 Object)
(leftSide Object)
(rightSide Object)
(boat Object)

(preconds
 (on adult1 leftSide)
 (on adult2 leftSide)
 (on child1 leftSide)
 (on child2 leftSide)
 (on boat leftSide)
 (clear rightSide))

(effects
 (on adult1 rightSide)
 (on adult2 rightSide)
 (on child1 rightSide)
 (on child2 rightSide)
 (on boat rightSide)
 (clear leftSide))
```

Below is what would go in the facts file (comments are added and are denoted by //). There are essentially only 5 different movements that are needed to solve this problem. I have translated

these different movements into 5 “rules”, with each rule represented as an operator. Each rule is described in a comment above the operator. I also made it so that the parameter is arbitrary. I only did this since I wasn’t sure if I could create an operator without any operators.

```
// Rule 1: If there are two children on left side, move both to right side. Parameter is arbitrary here.
(operator
 move2C
  (params
    (<c> Object))
  (preconds
     (on child1 leftSide) (on child2 leftSide) (on boat leftSide))
  (effects
     (del on child1 leftSide) (del on child2 leftSide) (del on boat leftSide) (on child1 rightSide) (on child2 rightSide) (on boat rightSide))
 )

// Rule 2: If there are two children on right side, move one (child1) to left side. Parameter is arbitrary.
(operator
  move1Ca
   (params
     (<c> Object))
   (preconds
    (on child1 rightSide) (on child2 rightSide) (on boat rightSide))
   (effects
    (del on child1 rightSide) (del on boat rightSide) (on child1 leftSide) (on boat leftSide))
   )

// Rule 3: If there is one child (child2) on the right side, move to the left side. Parameter is arbitrary.
(operator
  move1Cb
   (params
    (<c> Object))
   (preconds
    (on child1 leftSide) (on child2 rightSide) (on boat rightSide))
   (effects
    (del on child2 rightSide) (del on boat rightSide) (on child2 leftSide) (on boat leftSide))
   )
```

// Rule 4: If there are two adults and one child (child1) on left side, move one adult (adult1) to right side. Parameter is arbitrary.

```
(operator
   move1Aa
   (params
    (<a> Object))
   (preconds
    (on child1 leftSide) (on adult1 leftSide) (on adult2 leftSide) (on child2 rightSide) (on boat
leftSide))
   (effects
    (del on adult1 leftSide) (del on boat leftSide) (on adult1 rightSide) (on boat rightSide))
  )
```

// Rule 5: If there is one child (child1) and one adult (adult2) on the left side, move adult (adult2) to right side. Parameter is arbitrary.

```
(operator
  move1Ab
   (params
    (<a> Object))
   (preconds
    (on child1 leftSide) (on adult2 leftSide) (on adult1 rightSide) (on child2 rightSide) (on boat
leftSide))
   (effects
     (del on adult2 leftSide) (del on boat leftSide) (on adult2 rightSide) (on boat rightSide))
)
```

// Rule 6: If there are two children on left side and two adults on right side, move both children to right-side (will always be the final move). Parameter is arbitrary.

```
(operator
  finalMove
   (params
    (<c> Object))
   (preconds
    (on child1 leftSide) (on child2 leftSide) (on adult1 rightSide) (on adult2 rightSide) (on boat
leftSide))
   (effects
    (del on child1 leftSide) (del on child2 leftSide) (del on boat leftSide) (del clear rightSide) (on
child1 rightSide) (on child2 rightSide) (clear leftSide))
)
```

This is the end of the PDDL code.

Task 5:

The resultant state is given below:

(A ttt1)
(B ttt1)
(C ttt1)
(ppp1 B C)
(ppp2 A)
(ppp2 B)
(ppp3 C)
(eee1 A C)
(eee3 A)
(eee1 B C)
(eee2 B)

The removed lines were (eee2 C) and (eee3 C). Two lines were added: (eee1 B C) and (eee2 B). Otherwise, the rest of the state remains intact, as it wasn't changed by the effects of aaa.

Task 6:
P1, P2, P3, P4 — predicates
X1, X2, X3, X4, X5 — constants
I am assuming that the constants can be used as the parameters for the predicates.
I am also assuming that the order of parameters matters, meaning that something like (P1 X1 X2) is not considered equivalent to (P1 X2 X1).

At the lower bound, none of the predicates would take any arguments (meaning they would always be true). That means there could only be a single unique state consisting of the 5 predicates (with no arguments) and nothing else.

At the upper bound, each predicate would take three arguments. Since I assumed that the order of parameters matters, a permutation must be calculated to find the number of unique combinations of parameters that a single predicate can have. The permutation can be calculated as such (choosing 3 out of 5): 5!/(5-3)!. This comes out as 60. That means that, for a single predicate with three parameters, there are 60 unique sets of three parameters.

At this point, there are four predicates, each with 60 different sets of three parameters. We now have to calculate the number of unique combinations of predicates, with each combination of predicates constituting a state. The number of unique states can then be calculated using the Rule of Product (60 ways of using P1, 60 ways of using P2, and so on). So, according to the Rule of Product, the number of unique states = 60*60*60*60 = 12960000.

In conclusion, given a domain (JUNGLE) with 4 predicates (each with, at most, 3 parameters) and 5 constants, the number of unique states ranges from 1 (no parameters) to 12960000 (3 parameters), inclusive.