Q1: Monkey & Bananas problem

There are three classes of objects involved in this problem, Monkey class, Box class and Bananas class. They all have two attributes: location and height. Monkey class has a special attribute: has, which is a list of objects that monkey already got. Monkey class also has five methods: Walk, Push, Climb-Up, Climb-Down and Get. These operations (methods) are defined as the following statements:

Walk updates a monkey's location:

```
def Walk(self, newLocation):
    self.location = newLocation
```

Push takes one Box object, updates the box's location and the monkey's location:

```
def Push(self, box, newLocation):
    …check conditions, whether they have same locations…
    self.location = newLocation
    box.location = newLocation
```

Climb-Up and ***Climb-Down*** take one Box object, update the monkey's height:

```
def Climb-Up(self, box):
…check conditions, whether they have same locations, whether monkey is already on the box…
    self.height += box.height

def Climb-Down(self, box):
    …check conditions…
    self.height -= box.height
```

Get takes one Bananas object. It first checks whether the monkey already got the object, if not then checks whether they (the monkey and bananas) have same locations and heights.

```
def Get(self, bananas):
    if bananas in self.has:
        return True
    elif self.location == bananas.location and self.height == bananas.height:
        self.has.append(bananas)
        return True
    else:
        return False
```

After inventing these operators, the next step is using **Means-Ends Analysis** to help the monkey to reach the goal. The strategy of Means-Ends Analysis is quite simple, that if we cannot solve a problem in one step, why not solve the problem in multiple steps and each step solve a small proportion of the problem.

Let's first create one Monkey object (m), one Box object (bo) and one Bananas objects by passing them locations and heights. *m = Monkey('A', 'X'), bo = Box('B', 'X'), ba = Bananas('C', 'Y').* Table 1 describes the initial state and the goal. The only difference between these two states is that the monkey finally got the bananas (*m.Get(ba) == True*). The difference between states is simplified as a quantitative value after cumulating each small difference values. The

differences of locations and heights are signed as 1, while the difference that whether the monkey got the bananas gets a big difference value as 7. We assume that getting the bananas is the monkey's major goal, and teasing scientist is a minor goal.

Table 1:

initial state:	***transition state:***	**goal:**
m.location == 'A'(**1**) m.height == 'X' (**1**) bo.location == 'B' (**1**) m.Get(ba) == False (**0**)	*m.location == 'C' **(0)*** *m.height == 'Y' **(0)*** *bo.location == 'C' **(0)*** *m.Get(ba) == True**(7)*	m.location == 'A' (**1**) m.height == 'X' (**1**) bo.location == 'B' (**1**) m.Get(ba) == True (**7**)
Score = 3	***Score = 7***	**Score = 10**

After analysis of the Get operation, there are some conflicts in the goal state. To get the bananas, the monkey has to update its location and height, to update its height, it has to update the box's location, and thus it cannot keep them at their original places. To solve the problem, the monkey generated a middle goal (transition state in Table 1). The original problem changes into two small problems (1. get the bananas and 2. restore locations and heights). Means-ends analysis can be used to solve these two small problems and details are listed in Table 2 (problem 1) and Table 3 (problem 2). The difference values are signed differently, since now there are different goals. The first problem (Table 2) has initial score 0, and the score increases after the monkey makes right operations. After one Walk, one Push, one Climb-Up and one Get operation, the final score reaches 5. The second problem (Table 3) has initial score 2, and after one Climb-Down, one Push and one Walk operation reaches final goal (score 5).

Table 2:

initial state:	***goal:***	**operations (scores):**
m.location == 'A' (**0**) m.height == 'X' (**0**) bo.location == 'B' (**0**) m.Get(ba) == False (**0**)	*m.location == 'C' **(1)*** *m.height == 'Y' **(1)*** *bo.location == 'C' **(1)*** *m.Get(ba) == True**(2)*	m.Walk('B') (+**0**) m.Push(bo, 'C') (+**2**) m.Climb-Up(bo) (**+1**) m.Get(ba) (+**2**)
Score = 0	***Score = 5***	**Score = 5**

Table 3:

initial state:	**goal:**	**operations (scores):**
*m.location == 'C' **(0)*** *m.height == 'Y' **(0)*** *bo.location == 'C' **(0)*** *m.Get(ba) == True **(2)***	m.location == 'A' (**1**) m.height == 'X' (**1**) bo.location == 'B' (**1**) m.Get(ba) == True (**2**)	m.Climb-Down(bo) (**+1**) m.Push(bo, 'B') (**+1**) m.Walk('A') (**+1**)
Score = 2	**Score = 5**	**Score = 5**

Q2: the concept of an apple

The concepts are built on knowledge and experiences. For us, with abundant knowledge and experiences, we can deliver the concept of an apple in much more details. We know the apple is kind of fruit (knowledge of fruit), which tastes delicious (knowledge of delicious). And based on our experiences, we know it has various colors, shapes, sizes. We can even tell whether an apple is fresh or not just by looking its color and texture. Martian's knowledge background is quite limited, so it will not give the concept of an apple as accurate as humans.

Even though the dramatic difference on background knowledge, Humans and Martian will learn concepts in a similar way. For humans, one object's concept generates simultaneously when we see the first positive example. The new formed concept will be examined by more and more positive and negative examples. If we find positive examples that the concept does not include, we will generalize our concept to cover the new positive examples, while if we find negative examples that match the current concept, we will specialize our current concept to exclude the negative examples. The refinement procedure happens every day/time, even on some well know concepts. For example, when I first saw apple pears in my local Costco store, I was so curious and cannot control myself to buy and taste them. Apple pear blurs the line between apple and pear and forces me to refine the concepts of apple and pear. These near miss examples are especially important for concepts learning.

Following similar procedures, Martian will learn the concept of an apple using the provided positive and negative examples. Table 1 shows the concept changes after providing three positive examples. Example1 is a fresh red delicious apple, which is sphere in shape and weights 200 g. With the first positive example, Martian quickly builds the concept of an apple as a red, smooth and sphere object weights 200 g. Example 2 is a large green apple and has many spots on its surface. Martian generalized its apple concept, that apple's color can be red and green, texture various between smooth and spotty. This generalization method is called ***Chunking*** or ***Enlarge-set*** heuristic. Apple's weight ranges between 200-350 g, that generalization method is called ***Close-interval***. The third example is a bad apple with wrinkly surface and un-regular shape. Martian further updates its apple concept, that an apple can have rounded shape. This is a ***Climb-tree*** heuristic method.

Table 1:

	Ex 1 (+) "red delicious"	**Concept**	**Ex 2 (+)** "green apple"	**Concept**	**Ex 3 (+)** "bad apple"	**Concept**
color	red	red	green	red, green	red	red, green
weight	200 g	200 g	350 g	200-350 g	150 g	150-350 g
texture	smooth	smooth	spotty	smooth, spotty	wrinkly	smooth, spotty wrinkly
shape	sphere	sphere	sphere	sphere	rounded	rounded

After three positive examples, Martian is provided three negative examples. The first negative example is a ping-pong ball, which is white, light, smooth and sphere. It does not fit Martian's apple concept, thus Martine will do nothing. The second negative example is an orange, which is yellow, smooth and rounded. The orange does not fit Martian's apple concept, thus Martian will do nothing. The third negative example is a green lemon, which is green, smooth and weights 250 g. The only difference is that its shape is ellipsoid. To exclude lemons from apples, Martian may have to update its apple concept to change shape from rounded back to sphere. Negative examples are especially important for concepts refinements. But considering Martian's limited knowledge, further refinements are difficult.

Table 2:

	Ex 1 (-) "ping-pong ball"	**Concept**	**Ex 2 (-)** "orange"	**Concept**	**Ex 3 (-)** "lemon"	**Concept (final)**
color	white	red, green	yellow	red, green	green	red, green
weight	50 g	150-350g	200 g	150-350g	250 g	150-350g
texture	smooth	smooth, spotty wrinkly	smooth	smooth, spotty wrinkly	smooth	smooth, spotty wrinkly
shape	sphere	rounded	rounded	rounded	ellipsoid	sphere

Q3: robot training

The goal is to design robots that are capable of baby-sitting. Instead of pre-installing all knowledge or skills, such robots should be able to learn various skills, and how to handle different situations directly from their human trainers. We can assume that such robots already have the abilities to walk (move around in a room), handle things (move objects), see (detect objects' shapes, colors and motions through a laser detector) and hear (ability to analyze sounds or even simple languages) from their surrounding environments.

The challenges:

If we can train a five-year old baby to do lots of things, what are the difficulties to train a robot? Most times, the difficulties of a problem show up only when you start to solve the problem. Let's first see a simple implementation of robot training, which of course has lots of shortcomings. By analyzing these shortcomings, we will have much better understanding of the difficulties of robot training.

The simple implementation:

Suppose I can itemize everything in the house, just like a python dictionary *{baby: object 1, cradle: object 2, milk carton: object 3, microwave oven: object 4, breastmilk bottle: object 5,…}.* When I prepare milk to feed my baby, I pour milk from a carton to a breastmilk bottle, put the bottle in a microwave oven, microwave the milk a few minutes, take baby out of cradle, and feed the baby. The robot views my actions like: tilt object 3 above object 5, move object 5 into object 4, move object 5 out of object 4, move object 1 out of object 2, connect object 1 and object 5, move object 1 back to object 2, a sequence like 355454121512. I can further itemize different parts of one object (e.g. the door handle, different push buttons of microwave oven) to make the sequence more accurate. Using the same approach, the robot will record and store my actions as different sequences of numbers. And if I add a trigger before each sequence, for example by adding a baby's crying sound before the previous sequence, the robot will automatically heat the milk and feed the baby when it hears the baby crying sound. In this implementation, the robot has no knowledge of any objects in the house, and everything is as simple as a movable object.

Obviously, you won't like this implementation. It is more close to mechanical engineering than artificial intelligence. First, since the robot views everything same as one object, you will feel unsecure to use such a robot for baby-sitting. Second, during the training process, more and more sequences will be generated and stored, and these sequences will be retrieved at different situations. The sequences storage and searching will be a big problem later on. Third, since the robot has little intelligence, it cannot handle situations that not pre-stored, or slightly different with pre-stored situations. If the milk carton is empty, and even though there is a full gallon of milk just sits next the carton, the robot will still pour milk from the empty carton, microwave the empty bottle, and feed the baby with the empty bottle.

The difficulties of robot training mostly locate at two aspects: **1, knowledge learning during the training process.** Humans learn and increment their knowledge during training. For

example, when we train a teenage to heat milk using a microwave oven, and suppose that he/she has no knowledge of a microwave oven. He will quickly learn that what is a microwave oven, and what is the function of a microwave oven and how to use a microwave oven to heat milk. **2, reactions to different situations.** For example, if the baby is crying, we may shake the cradle to calm the baby. If the baby is crying for hungry, we may start to heat milk and feed the baby. If the baby keeps crying, and has a little fever, we may need to call a doctor or send the baby to hospital. How can a robot learn such reactions from humans during its training? Here, I present how to use **Frames** for robots to learn knowledge and how to use **Production System** for robots to learn humans' responses to different situations.

Using frames for knowledge learning:

A Frame is an artificial intelligence data structure that used to store or represent knowledge. It use slots to store values. In my implementation, robots will use frames to store the knowledge that they learn during the training process. When the robot first see a microwave oven, it will create a frame of "microwave oven" with its shape, size and color values filled (Figure 1a). That is all information of a microwave oven the robot currently knows. When the human trainer opens the microwave oven door and puts in a bottle of milk, the robot will quickly update the microwave oven frame with new information as shown in Figure 1b, that a microwave oven has an inside chamber and can move in a bottle of milk. If the robot can detect temperature changes through its laser detector, it will also found out the temperature increase of the milk after operation. Thus it may further update the frame by adding extra values (Figure 1c), that the temperature of object is low before object moves in and the temperature of object turns into high after object moves out the microwave oven. The robot will repeat this procedure again and again to create new frames, or update slots / values of old frames. The robot will create a frame for every object, e.g. the baby, the cradle, the breastmilk bottle, and the milk, et al.

Figure 1: the changes of the frame of microwave oven during robot training.

a

Slot	Value
Microwave oven	-
Shape	rectangular
Size (inch)	10x10x20
Color	black

b

Slot	Value
Microwave oven	-
Shape	rectangular
Size (inch)	10x10x20
Color	black
Has-inside chamber	true
Take object	a bottle of milk

c

Slot	Value
Microwave oven	-
Shape	rectangular
Size (inch)	10x10x20
Color	black
Has-inside chamber	true
Take object	a bottle of milk
Temperature of object (before)	low
Temperature of object (after)	high

Besides the frames for isolated objects, the robot will also create frames for skills. For example, the procedure of "heating" will be stored as a frame like Figure 2. In current frame of "heating", the value of slot: object is set as a bottle of milk. In the future, when the robot

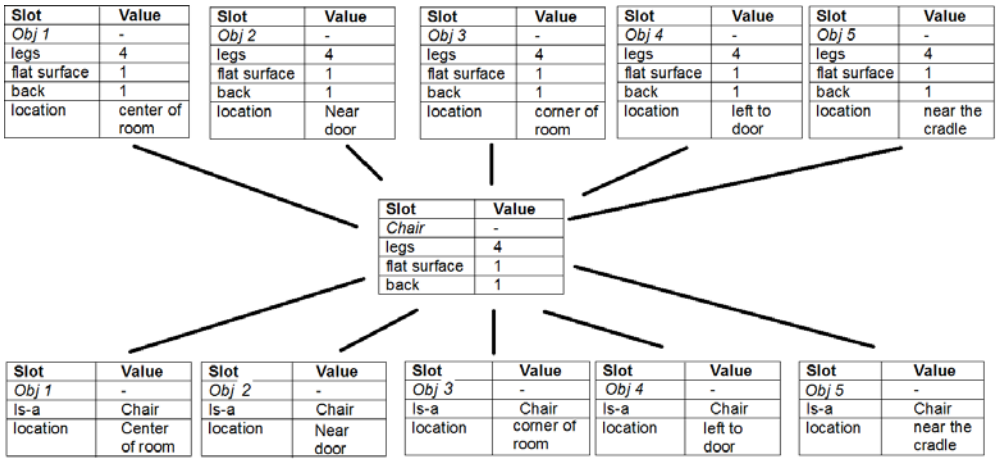
observes the trainer puts more and more objects into microwave oven, it will generalize the value of slot: “object” to include more potential objects, e.g. any beverage or food.

Figure 2: the frame to represent the skill of “heating”

Slot	Value
heating	-
apparatus	a microwave oven
object	a bottle of milk
time	2 min
procedure	1, open apparatus door 2, put in object 3, set time 4, move out object

The generalization step is also quite challenge for robots. If there are five chairs in the room, then the robot will create five different frames to represent each chair. For humans, we may quickly realize that they are the same things (chair) at different locations, but how can a robot to summarize these five isolated frames into one parent frame. The possible implementation would be set a threshold number e.g. 100, then after the robot created every 100 new frames, it will automatically start to compare all frames by their slot names and slot values to generate a list of parent frames. For example, in the 100 new frames, the robot detected that 5 frames (obj1 –obj5) that all have four legs, one back and one flat surface. The robot then may create the parent frame “chair” with specific slots and values, and add a new “Is-a” slot into obj1-obj5 frames that with the value of the parent frame “chair” (Figure 3).

Figure 3: the generation of parent frame “chair” after comparing five new frames’ slots and values.



The knowledge learning process is very complicated, and even more challenge if only through observation. If the robot can understand simple languages, the learning process will be much simpler. Human trainer may give direct instructions to the robots, thus the robot can create frames based on instructions instead of its own exploration.

Using production system to learn responses at different situations:

As discussed earlier, that a human baby-sitter will take different reactions at different situations. For example, if he/she hears the baby’s crying, he/she may first shake the cradle to

calm the baby. If that fails, or if the baby is crying for hungry, he/she may start to heat milk and feed the baby. If that fails again and he/she notices that the baby has a fever, he/she may call a doctor or send the baby to hospital. The robot will learn such responses through a production system.

A production system is a system based on IF… THEN... rules, and it consists three parts:

1, a set of rules (IF…, THEN…)

For a robot baby-sitter, the set of rules is more like:

IF baby is crying THEN shake the cradle for 10 min

IF baby is crying THEN heat milk and feed the baby

IF baby is crying and baby has a fever THEN call a doctor or send the baby to hospital

After previous training, the robot already stored knowledge about objects like baby, cradle, milk, … in frames. The robot also masters various skills like shake cradle, heat milk, feed baby, using frames.

During the training, the set of rules is incremental. The robot will add new IF…THEN… through observing human trainer's behavior or directly through following human trainer's instructions. For example, the robot observed that every time when the baby crawls out of the cradle, the human trainer put the baby back to the cradle. The robot will then add a new rule to the set of rules as:

IF baby moves out of cradle THEN put baby back to cradle

2, working memory

The working memory usually includes a detailed description of the current state. For a baby-sitter robot, it monitors the location of the baby object, and whether it makes sounds similar to pre-recorded crying sounds. The robot puts such information in working memory. The robot also puts its past actions in the working memory, that whether it already shook the cradle or fed the baby.

3, the recognize-act cycle

The robot keeps monitoring the baby object and updates its information in the working memory. At the same time, the robot is continuously searching through the working memory to see whether it contains specific patterns that listed in the set of rules (IF…, THEN…). Once it detected some behaviors, e.g. it finds that the location of baby is out of the cradle, it then triggers the IF…THEN… reaction, the robot will follows the rule to put the baby back to the cradle. If the robot detects baby crying sound, it also triggers IF…THEN… reactions. And since the baby crying sound will trigger a few rules, the robot need to examine the working memory for its past actions. If the working memory shows that the robot had already shaken the cradle, the robot will then start the second reaction to heat milk and to feed the baby. And at same time writes down its action in working memory.

Limitations:

The new implementation is much better compared with the original one. It is more intelligent. It learns knowledge during the training process. Through creating parent frames and using different generalization process, it obtains kinds of flexibilities when it faces situations that not pre-stored. For example, when it starts to heat milk, and found the original milk carton is empty, it is capable to identify other milk cartons in refrigerator, since they have the same parent frame “milk carton”. The limitation of this implementation is that unlike humans that can use logic or experience to handle new situations, while the current robot will stick on the set of rules (IF…, THEN…).