3.1.2 Agency and computational theory

Even though it seems reasonable to explore biological and cognitive sciences for insights in intelligence, how can we compare such different systems: carbon and silicon "life" forms? One powerful means of conceptualizing the different systems is to think of an abstract intelligent system. Consider something we'll call an *agent*. The agent is self-contained and independent. It has its own "brains" and can interact with the world to make changes or to sense what is happening. It has self-awareness. Under this definition, a person is an agent. Likewise, a dog or a cat or a frog is an agent. More importantly, an intelligent robot would be an agent, even certain kinds of web search engines which continue to look for new items of interest to appear, even after the user has logged off. Agency is a concept in artificial intelligence that allows researchers to discuss the properties of intelligence without discussing the details of how the intelligence got in the particular agent. In OOP terms, "agent" is the superclass and the classes of "person" and "robot" are derived from it.

AGENT

Of course, just referring to animals, robots, and intelligent software packages as "agents" doesn't make the correspondences between intelligence any clearer. One helpful way of seeing correspondences is to decide the level at which these entities have something in common. The set of levels of commonality lead to what is often called a *computational theory*[88] after David Marr. Marr was a neurophysiologist who tried to recast biological vision processes into new techniques for computer vision. The levels in a computational theory can be greatly simplified as:

COMPUTATIONAL THEORY

Level 1: Existence proof of what can/should be done. Suppose a roboticist is interested in building a robot to search for survivors trapped in a building after an earthquake. The roboticist might consider animals which seek out humans. As anyone who has been camping knows, mosquitoes are very good at finding people. Mosquitoes provide an existence proof that it is possible for a computationally simple agent to find a human being using heat. At Level 1, agents can share a commonality of purpose or functionality.

Level 2: Decomposition of "what" into inputs, outputs, and transformations. This level can be thought of as creating a flow chart of "black boxes." Each box represents a transformation of an input into an output. Returning to the example of a mosquito, the roboticist might realize from biology that the mosquito finds humans by homing on the heat of a human (or any warm blooded animal). If the mosquito senses a hot area, it flies toward it. The roboticist can model this process as: `input=thermal image`, `output=steering command`. The "black box" is how the mosquito transforms the input into the output. One good guess might be to take the centroid of the thermal image (the centroid weighted by the heat in each area of the image) and steer to that. If the hot patch moves, the thermal image will change with the next sensory update, and a new steering command will be generated. This might not be exactly how the mosquito actually steers, but it presents an idea of how a robot could duplicate the functionality. Also notice that by focusing on the process rather than the implementation, a roboticist doesn't have to worry about mosquitoes flying, while a search and rescue robot might have wheels. At Level 2, agents can exhibit common processes.

Level 3: How to implement the process. This level of the computational theory focuses on describing how each transformation, or black box, is implemented. For example, in a mosquito, the steering commands might be implemented with a special type of neural network, while in a robot, it might be implemented with an algorithm which computes the angle between the centroid of heat and where the robot is currently pointing. Likewise, a researcher interested in thermal sensing might examine the mosquito to see how it is able to detect temperature differences in such a small package; electro-mechanical thermal sensors weigh close to a pound! At Level 3, agents may have little or no commonality in their implementation.

It should be clear that Levels 1 and 2 are abstract enough to apply to any agent. It is only at Level 3 that the differences between a robotic agent and a biological agent really emerge. Some roboticists actively attempt to emulate biology, reproducing the physiology and neural mechanisms. (Most roboticists are familiar with biology and ethology, but don't try to make exact duplicates of nature.) Fig. 3.1 shows work at Case Western Reserve's Bio-Bot Laboratory under the direction of Roger Quinn, reproducing a cockroach on a neural level.

In general, it may not be possible, or even desirable, to duplicate how a biological agent does something. Most roboticists do not strive to precisely replicate animal intelligence, even though many build creatures which resemble animals, as seen by the insect-like Genghis in Fig. 3.2. But by focusing on Level 2 of a computational theory of intelligence, roboticists can gain insights into how to organize intelligence.

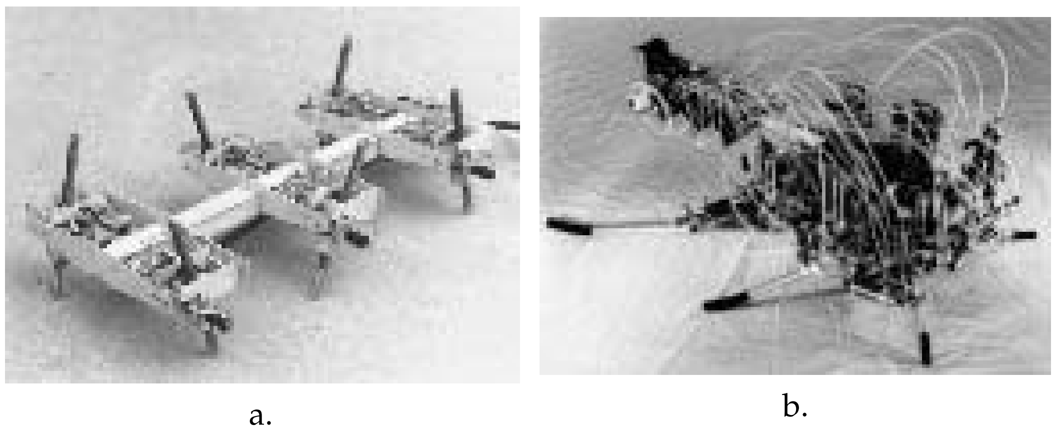

Figure 3.1 Robots built at the Bio-Bot Laboratory at Case Western Reserve University which imitate cockroaches at Level 3: a.) Robot I, an earlier version, and b.) Robot III. (Photographs courtesy of Roger Quinn.)

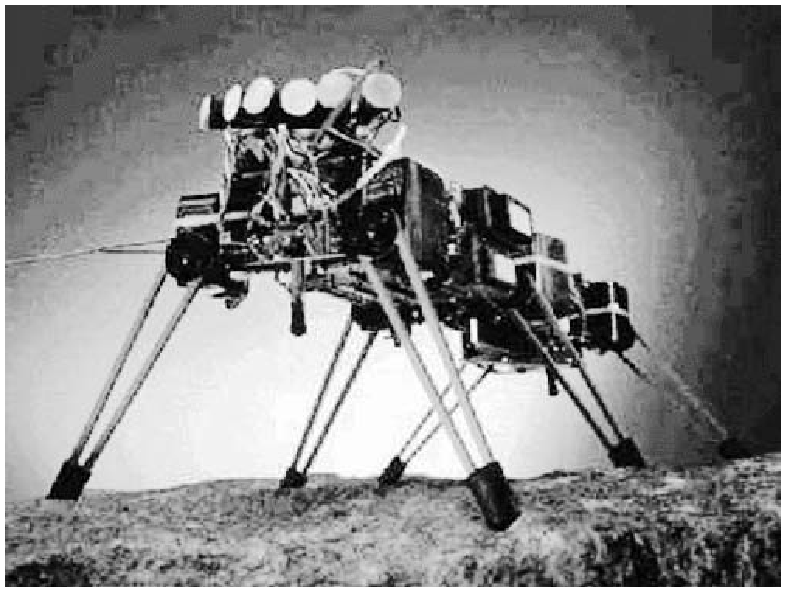

Figure 3.2 Genghis, a legged robot built by Colin Angle, IS Robotics, which imitates an insect at Levels 1 and 2. (Photograph courtesy of the National Aeronautics and Space Administration.)

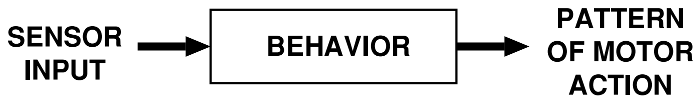

Figure 3.3 Graphical definition of a behavior.

3.2 What Are Animal Behaviors?

Scientists believe the fundamental building block of natural intelligence is a behavior. A *behavior* is a mapping of sensory inputs to a pattern of motor actions which then are used to achieve a task. For example, if a horse sees a predator, it flattens its ears, lowers its head, and paws the ground. In this case, the sensory input of a predator triggers a recognizable pattern of a defense behavior. The defensive motions make up a pattern because the actions and sequence is always the same, regardless of details which vary each episode (e.g., how many times the horse paws the ground). See Fig. 3.3.

BEHAVIOR

Scientists who study animal behaviors are called *ethologists*. They often spend years in the field studying a species to identify its behaviors. Often the pattern of motor actions is easy to ascertain; the challenging part is to discover the sensory inputs for the behavior and why the behavior furthers the species survival.

REFLEXIVE BEHAVIOR

STIMULUS-RESPONSE

REACTIVE BEHAVIOR

CONSCIOUS BEHAVIOR

Behaviors can be divided into three broad categories.[10] *Reflexive behaviors* are *stimulus-response (S-R)*, such as when your knee is tapped, it jerks upward. Essentially, reflexive behaviors are "hardwired"; neural circuits ensure that the stimulus is directly connected to the response in order to produce the fastest response time. *Reactive behaviors* are learned, and then consolidated to where they can be executed without conscious thought. Any behavior that involves what is referred to in sports as "muscle memory" is usually a reactive behavior (e.g., riding a bike, skiing). Reactive behaviors can also be changed by conscious thought; a bicyclist riding over a very narrow bridge might "pay attention" to all the movements. *Conscious behaviors* are deliberative (assembling a robot kit, stringing together previously developed behaviors, etc.).

The categorization is worthy of note for several reasons. First, the Reactive Paradigm will make extensive use of reflexive behaviors, to the point that some architectures only call a robot behavior a behavior if it is S-R. Second, the categorization can help a designer determine what type of behavior to use, leading to insights about the appropriate implementation. Third, the use of the word "reactive" in ethology is at odds with the way the word is used in robotics. In ethology, reactive behavior means learned behaviors or a skill; in robotics, it connotes a reflexive behavior. If the reader is unaware of these differences, it may be hard to read either the ethological or AI literature without being confused.

3.2.1 Reflexive behaviors

Reflexive types of behaviors are particularly interesting, since they imply no need for any type of cognition: if you sense it, you do it. For a robot, this would be a hardwired response, eliminating computation and guaranteed to be fast. Indeed, many kit or hobby robots work off of reflexes, represented by circuits.

Reflexive behaviors can be further divided into three categories:[10]

REFLEXES

1. *reflexes*: where the response lasts only as long as the stimulus, and the response is proportional to the intensity of the stimulus.

TAXES

2. *taxes*: where the response is to move to a particular orientation. Baby turtles exhibit *tropotaxis*; they are hatched at night and move to the brightest light. Until recently the brightest light would be the ocean reflecting the moon, but the intrusion of man has changed that. Owners of beach front property in Florida now have to turn off their outdoor lights during hatching season to avoid the lights being a source for tropotaxis. Baby turtles hatch at night, hidden from shore birds who normally eat them. It had been a mystery as to how baby turtles knew which way was the ocean when they hatched. The story goes that a volunteer left a flashlight on the sand while setting up an experiment intended to show that the baby turtles used magnetic fields to orient themselves. The magnetic field theory was abandoned after the volunteers noticed the baby turtles heading for the flashlight! Ants exhibit a particular taxis known as *chemotaxis*; they follow trails of pheromones.

FIXED-ACTION PATTERNS

3. *fixed-action patterns*: where the response continues for a longer duration than the stimulus. This is helpful for fleeing predators. It is important to keep in mind that a taxis can be any orientation relative to a stimulus, not just moving towards.

The above categories are not mutually exclusive. For example, an animal going over rocks or through a forest with trees to block its view might persist (fixed-action patterns) in orienting itself to the last sensed location of a food source (taxis) when it loses sight of it.

The tight coupling of action and perception can often be quantified by mathematical expressions. An example of this is orienting in angelfish. In order to swim upright, an angelfish uses an internal (*idiothetic*) sense of gravity combined with its vision sense (*allothetic*) to see the external percept of the horizon line of the water to swim upright. If the fish is put in a tank with prisms that make the horizon line appear at an angle, the angelfish will swim cockeyed. On closer inspection, the angle that the angelfish swims at is the vector sum of the vector parallel to gravity with the vector perpendicular to the perceived horizon line! The ability to quantify animal behavior suggests that computer programs can be written which do likewise.

IDIOTHETIC

ALLOTHETIC

3.3 Coordination and Control of Behaviors

KONRAD LORENZ
NIKO TINBERGEN

Konrad Lorenz and Niko Tinbergen were the founding fathers of ethology. Each man independently became fascinated not only with individual behaviors of animals, but how animals acquired behaviors and selected or coordinated sets of behaviors. Their work provides some insight into four different ways an animal might acquire and organize behaviors. Lorenz and Tinbergen's work also helps with a computational theory Level 2 understanding of how to make a process out of behaviors.

The four ways to acquire a behavior are:

INNATE

1. to be born with a behavior (*innate*). An example is the feeding behavior in baby arctic terns. Arctic terns, as the name implies, live in the Arctic where the terrain is largely shades of black and white. However, the Arctic tern has a bright reddish beak. When babies are hatched and are hungry, they peck at the beak of their parents. The pecking triggers a regurgitation reflex in the parent, who literally coughs up food for the babies to eat. It turns out that the babies do not recognize their parents, per se. Instead, they are born with a behavior that says: if hungry, peck at the largest red blob you see. Notice that the only red blobs in the field of vision should be the beaks of adult Arctic terns. The largest blob should be the nearest parent (the closer objects are, the bigger they appear). This is a simple, effective, and computationally inexpensive strategy.

SEQUENCE OF INNATE BEHAVIORS

2. to be born with a *sequence of innate behaviors*. The animal is born with a sequence of behaviors. An example is the mating cycle in digger wasps.

A female digger wasp mates with a male, then builds a nest. Once it sees the nest, the female lays eggs. The sequence is logical, but the important point is the role of stimulus in triggering the next step. The nest isn't built until the female mates; that is a change in internal state. The eggs aren't laid until the nest is built; the nest is a visual stimulus releasing the next step. Notice that the wasp doesn't have to "know" or understand the sequence. Each step is triggered by the combination of internal state and the environment. This is very similar to Finite State Machines in computer science programming, and will be discussed later in Ch. 5.

INNATE WITH MEMORY

3. to be born with behaviors that need some initialization (*innate with memory*). An animal can be born with innate behaviors that need customizing based on the situation the animal is born in. An example of this is bees. Bees are born in hives. The location of a hive is something that isn't innate; a baby bee has to learn what its hive looks like and how to navigate to and from it. It is believed that the curious behavior exhibited by baby bees (which is innate) allows them to learn this critical information. A new bee will fly out of the hive for a short distance, then turn around and come back. This will get repeated, with the bee going a bit farther along the straight line each time. After a time, the bee will repeat the behavior but at an angle from the opening to the hive. Eventually, the bee will have circumnavigated the hive. Why? Well, the conjecture is that the bee is learning what the hive looks like from all possible approach angles. Furthermore, the bee can associate a view of the hive with a motor command ("fly left and down") to get the bee to the opening. The behavior of zooming around the hive is innate; what is learned about the appearance of the hive and where the opening is requires memory.

LEARN

4. to *learn* a set of behaviors. Behaviors are not necessarily innate. In mammals and especially primates, babies must spend a great deal of time learning. An example of learned behaviors is hunting in lions. Lion cubs are not born with any hunting behaviors. If they are not taught by their mothers over a period of years, they show no ability to fend for themselves. At first it might seem strange that something as fundamental as hunting for food would be learned, not innate. However, consider the complexity of hunting for food. Hunting is composed of many sub-behaviors, such as searching for food, stalking, chasing, and so on. Hunting may also require teamwork with other members of the pride. It requires great sensitivity to the type of the animal being hunted and the terrain. Imagine trying to write a program to cover all the possibilities!

> While the learned behaviors are very complex, they can still be represented by innate releasing mechanisms. It is just that the releasers and actions are learned; the animal creates the program itself.

Note that the number of categories suggests that a roboticist will have a spectrum of choices as to how a robot can acquire one or more behaviors: from being pre-programmed with behaviors (innate) to somehow learning them (learned). It also suggests that behaviors can be innate but require memory. The lesson here is that while S-R types of behaviors are simple to pre-program or hardwire, robot designers certainly shouldn't exclude the use of memory. But as will be seen in Chapter 4, this is a common constraint placed on many robot systems. This is especially true in a popular style of hobby robot building called BEAM robotics (biology, electronics, aesthetics, and mechanics), espoused by Mark Tilden. Numerous BEAM robot web sites guide adherents through construction of circuits which duplicate memory-less innate reflexes and taxes.

INTERNAL STATE
MOTIVATION

An important lesson that can be extracted from Lorenz and Tinbergen's work is that the internal state and/or motivation of an agent may play a role in releasing a behavior. Being hungry is a stimulus, equivalent to the pain introduced by a sharp object in the robot's environment. Another way of looking at it is that motivation serves as a stimulus for behavior. Motivations can stem from survival conditions (like being hungry) or more abstract goals (e.g., need to check the mail). One of the most exciting insights is that behaviors can be sequenced to create complex behaviors. Something as complicated as mating and building a nest can be decomposed into primitives or certainly more simple behaviors. This has an appeal to the software engineering side of robotics.

3.3.1 Innate releasing mechanisms

INNATE RELEASING
MECHANISMS

RELEASER

Lorenz and Tinbergen attempted to clarify their work in how behaviors are coordinated and controlled by giving it a special name *innate releasing mechanisms* (IRM). An IRM presupposes that there is a specific stimulus (either internal or external) which releases, or triggers, the stereotypical pattern of action. The IRM activates the behavior. A *releaser* is a latch or a Boolean variable that has to be set. One way to think of IRMs is as a process of behaviors. In a computational theory of intelligence using IRMs, the basic black boxes of the process would be behaviors. Recall that behaviors take sensory input and produce motor actions. But IRMs go further and specify when a behav-

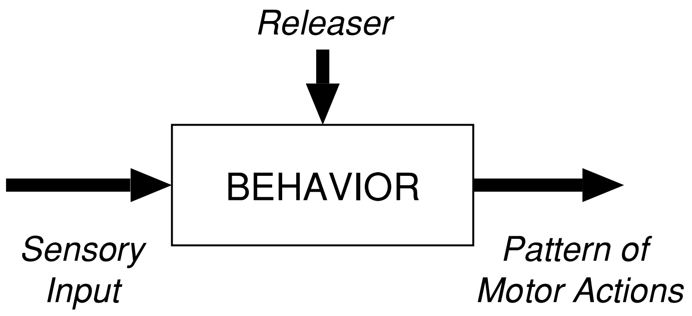

Figure 3.4 Innate Releasing Mechanism as a process with behaviors.

ior gets turned on and off. The releaser acts as a control signal to activate a behavior. If a behavior is not released, it does not respond to sensory inputs and does not produce motor outputs. For example, if a baby arctic tern isn't hungry, it doesn't peck at red, even if there is a red beak nearby.

Another way to think of IRMs is as a simple computer program. Imagine the agent running a C program with a continuous `while` loop. Each execution through the loop would cause the agent to move for one second, then the loop would repeat.

```
enum            Releaser={PRESENT, NOT_PRESENT};
Releaser        predator;
while (TRUE)
{
   predator = sensePredators();
   if (predator == PRESENT)
     flee();
}
```

In this example, the agent does only two things: sense the world and then flees if it senses a predator. Only one behavior is possible: `flee`. `flee` is released by the presence of a predator. A predator is of type `Releaser` and has only two possible values: it is either present or it is not. If the agent does not sense the releaser for the behavior, the agent does nothing. There is no "default" behavior.

This example also shows filtering of perception. In the above example, the agent only looks for predators with a dedicated detection function, `sense-Predators()`. The dedicated predator detection function could be a specialized sense (e.g., retina is sensitive to the frequency of motions associated with predator movement) or a group of neurons which do the equivalent of a computer algorithm.

COMPOUND RELEASERS

Another important point about IRMs is that the releaser can be a *compound of releasers*. Furthermore, the releaser can be a combination of either external (from the environment) or internal (motivation). If the releaser in the compound isn't satisfied, the behavior isn't triggered. The pseudo-code below shows a compound releaser.

```
enum            Releaser={PRESENT, NOT_PRESENT};
Releaser        food;
while (TRUE)
{
   food = senseFood();
   hungry = checkState();
      if (food == PRESENT && hungry==PRESENT)
      feed();
}
```

IMPLICIT CHAINING

The next example below shows what happens in a sequence of behaviors, where the agent eats, then nurses its young, then sleeps, and repeats the sequence. The behaviors are implicitly chained together by their releasers. Once the initial releaser is encountered, the first behavior occurs. It executes for one second (one "movement" interval), then control passes to the next statement. If the behavior isn't finished, the releasers remain unchanged and no other behavior is triggered. The program then loops to the top and the original behavior executes again. When the original behavior has completed, the internal state of the animal may have changed or the state of the environment may have been changed as a result of the action. When the motivation and environment match the stimulus for the releaser, the second behavior is triggered, and so on.

```
enum            Releaser={PRESENT, NOT_PRESENT};
Releaser        food, hungry, nursed;
while (TRUE) {
   food = sense();
   hungry = checkStateHunger();
   child = checkStateChild();
   if (hungry==PRESENT)
      searchForFood(); //sets food = PRESENT when done
   if (hungry==PRESENT && food==PRESENT)
      feed(); // sets hungry = NOT_PRESENT when done
   if (hungry== NOT_PRESENT && parent==PRESENT)
      nurse(); // set nursed = PRESENT when done
   if (nursed ==PRESENT)
      sleep();
}
```

The example also reinforces the nature of behaviors. If the agent sleeps and wakes up, but isn't hungry, what will it do? According to the releasers created above, the agent will just sit there until it gets hungry.

In the previous example, the agent's behaviors allowed it to feed and enable the survival of its young, but the set of behaviors did not include fleeing or fighting predators. Fleeing from predators could be added to the program as follows:

```
enum            Releaser={PRESENT, NOT_PRESENT};
Releaser        food, hungry, nursed, predator;
while (TRUE) {
   predator = sensePredator();
   if (predator==PRESENT)
      flee();
   food = senseFood();
   hungry = checkStateHunger();
   parent = checkStateParent();
   if (hungry==PRESENT)
      searchForFood();
   if (hungry==PRESENT && food==PRESENT)
      feed();
   if (hungry== NOT_PRESENT && parent==PRESENT)
      nurse();
   if (nursed ==PRESENT)
      sleep();
}
```

Notice that this arrangement allowed the agent to flee the predator regardless of where it was in the sequence of feeding, nursing, and sleeping because predator is checked for first. But fleeing is temporary, because it did not change the agent's internal state (except possibly to make it more hungry which will show up on the next iteration). The code may cause the agent to flee for one second, then feed for one second.

One way around this is to *inhibit*, or turn off, any other behavior until fleeing is completed. This could be done with an `if-else` statement:

```
while (TRUE) {
   predator = sensePredator();
   if (predator==PRESENT)
      flee();
   else {
      food = senseFood();
      hungry = checkStateHunger();
      ...
   }
}
```

The addition of the `if-else` prevents other, less important behaviors from executing. It doesn't solve the problem with the predator releaser disappearing as the agents runs away. If the agent turns and the predator is out of view (say, behind the agent), the value of `predator` will go to `NOT_PRESENT` in the next update. The agent will go back to foraging, feeding, nursing, or sleeping. Fleeing should be a fixed-pattern action behavior which persists for some period of time, `T`. The fixed-pattern action effect can be accomplished with:

```
#define T LONG_TIME
while (TRUE) {
   predator = sensePredator();
   if (predator==PRESENT)
      for(time = T; time > 0; time--)
         flee();
   else {
      food = senseFood();
      ...
   }
}
```

The C code examples were implemented as an implicit sequence, where the order of execution depended on the value of the releasers. An implementation of the same behaviors with an explicit sequence would be:

```
Releaser         food, hungry, nursed, predator;
while (TRUE) {

   predator = sensePredator();
   if (predator==PRESENT)
```

```
      flee();
   food = senseFood();
   hungry = checkStateHunger();
   parent = checkStateParent();
   if (hungry==PRESENT)
      searchForFood();
   feed();
   nurse();
   sleep();
}
```

The explicit sequence at first may be more appealing. It is less cluttered and the compound releasers are hidden. But this implementation is not equivalent. It assumes that instead of the loop executing every second and the behaviors acting incrementally, each behavior takes control and runs to completion. Note that the agent cannot react to a predator until it has finished the sequence of behaviors. Calls to the fleeing behavior could be inserted between each behavior or fleeing could be processed on an interrupt basis. But every "fix" makes the program less general purpose and harder to add and maintain.

The main point here is: *simple behaviors operating independently can lead to what an outside observer would view as a complex sequence of actions.*

3.3.2 Concurrent behaviors

An important point from the examples with the IRMs is that behaviors can, and often do, execute concurrently and independently. What appears to be a fixed sequence may be the result of a normal series of events. However, some behaviors may violate or ignore the implicit sequence when the environment presents conflicting stimuli. In the case of the parent agent, fleeing a predator was mutually exclusive of the feeding, nursing, and sleeping behaviors.

Interesting things can happen if two (or more) behaviors are released that usually are not executed at the same time. It appears that the strange interactions fall into the following categories:

EQUILIBRIUM

- *Equilibrium (the behaviors seem to balance each other out):* Consider feeding versus fleeing in a squirrel when the food is just close enough to a person on a park bench. A squirrel will often appear to be visibly undecided as to whether to go for the food or to stay away.