Introduction to (Python) ACT-R

Semantics Seminar: Computing Dynamic Meanings

Adrian Brasoveanu*

April 20, 2015

Contents

*This intro to (Python) ACT-R is partly based on Kam Kwok's *Python ACT-R User Guide version 1.0* (12/21/2009) (`https://sites.google.com/site/kamkwokcogsci/cogsci-papers-and-projects/pACTRUserGuideReleasdv1.pdf`), on Robert West's tutorials available at `https://sites.google.com/site/pythonactr/`, on chapters 11 and 12 of Stewart 2007, and on some of the tutorials that are part of the Python ACT-R / CCMSuite library itself. Python ACT-R has been created and is maintained by Terrence Stewart (see Stewart 2007, Stewart and West 2007). The github repo is available here `https://github.com/tcstewar/ccmsuite`. This document has been created with PythonTex (Poore, 2013).

1 Installing Python 2.7 and Python ACT-R

The first step is to install Python 2.7 on your machine. The Anaconda platform is available for all major operating systems (Windows, Mac, Linux) here:

`http://continuum.io/downloads`

This platform includes all the necessary scientific libraries you will likely need out of the box – in addition to Python 2.7. Do not install Python 3.∗ – Python ACT-R and various scientific computation libraries will work only with Python 2 (most likely, 2.6 and above).

If you need to get up to speed with Python, you can try these resources (or any other resources you find on the web that seem like they would be useful to you – and there are many):

- Think Python – How to Think Like a Computer Scientist: `http://www.greenteapress.com/thinkpython/`
- Python 2 tutorial: `http://www.python-course.eu/course.php`
- The official Python 2 tutorial: `https://docs.python.org/2/tutorial/`
- Learn Python the Hard Way: `http://learnpythonthehardway.org/`
- Python Scientific Lecture Notes: `https://scipy-lectures.github.io/`

Once the Anaconda platform is installed, download Python ACT-R (aka the CCM suite):

`https://github.com/tcstewar/ccmsuite/archive/master.zip`

Unzip the archive, find the `ccm` subfolder and copy-paste it in whatever folder your Python installation stores its libraries. That is, install the `ccm` library manually.

To test that your Python ACT-R installation works, launch Anaconda (it comes with a built-in editor) or your favorite text editor[1] and type the following code exactly as it is displayed below. Note that all the indentation levels are exactly 4 spaces; do not use tabs to indent lines.

(1) File **test_Python_ACT-R_installation.py**:

```
import ccm
from ccm.lib.actr import *


class Testing(ACTR):
    goal = Buffer()
```

[1]If you don't have a good editor yet, install Gedit from here: `https://wiki.gnome.org/Apps/Gedit`. You shouldn't use MS Word / Libre Office / Open Office for coding; you could, but you most probably shouldn't.

```
        def init():
            goal.set("test")
        def test(goal="test"):
            print "Python ACT-R installation up and running"
            print "The hard part is over :-)"
            goal.set("stop")
        def stop_production(goal="stop"):
            self.stop()


    class MyEnvironment(ccm.Model):
        pass


    if __name__ == "__main__":
        testing = Testing()
        empty_environment = MyEnvironment()
        empty_environment.agent = testing
        ccm.log_everything(empty_environment)
        empty_environment.run()
        ccm.finished()
```

Once you are done typing the code, save the file as `test_Python_ACT-R_installation.py` in a folder, open a terminal in the folder in which you saved this file, and run the python script in the terminal with the command below:

```
$ python test_Python_ACT-R_works.py
```

If you are running Anaconda on Windows, just click the `Run` button in the menu.
If everything is installed correctly, you will get the following output:

```
   0.000 agent.production_threshold None
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_time 0.05
   0.000 agent.goal.chunk None
   0.000 agent.goal.chunk test
   0.000 agent.production test
   0.050 agent.production None
Python ACT-R installation up and running
The hard part is over :-)
   0.050 agent.goal.chunk stop
   0.050 agent.production stop_production
   0.100 agent.production None
```

If you don't get this output, you need to get help specifically for your operating system and the Python distribution / platform you are using. Google will know; or the above tutorials; or a computer-savvy friend.

2 Why do we care about ACT-R, and cognitive architectures and modeling in general

Linguistics is part of the larger field of cognitive science. So the answer to this question is one that applies to cog sci in general. Here's one recent version of the argument, taken from chapter 1 of Lewandowsky and Farrell 2010. The argument is an argument for *process* models as the proper scientific target to aim for (roughly, models of human language performance), rather than *characterization* models (roughly, models of human language competence).

Both of them are better than simply *descriptive* models, "whose sole purpose is to replace the intricacies of a full data set with a simpler representation in terms of the model's parameters. Although those models themselves have no psychological content, they may well have compelling psychological implications. [Both characterization and process models] seek to illuminate the workings of the mind, rather than data, but do so to a greatly varying extent. Models that characterize processes identify and measure cognitive stages, but they are neutral with respect to the exact mechanics of those stages. [Process] models, by contrast, describe all cognitive processes in great detail and leave nothing within their scope unspecified. Other distinctions between models are possible and have been proposed [...], and we make no claim that our classification is better than other accounts. Unlike other accounts, however, **our three classes of models map into three distinct tasks that confront cognitive scientists. Do we want to describe data? Do we want to identify and characterize broad stages of processing? Do we want to explain how exactly a set of postulated cognitive processes interact to produce the behavior of interest?**" (Lewandowsky and Farrell, 2010, p. 25)

In a bit more detail: "Like characterization models, [the power of process models] rests on hypothetical cognitive constructs, but by providing a detailed explanation of those constructs, they are no longer neutral. [...] At first glance, one might wonder why not every model belongs to this class. After all, if one can specify a process, why not do that rather than just identify and characterize it? The answer is twofold. First, it is not always possible to specify a presumed process at the level of detail required for [a process] model [...] Second, there are cases in which a coarse characterization may be preferable to a detailed specification. For example, it is vastly more important for a weatherman to know whether it is raining or snowing, rather than being confronted with the exact details of the water molecules' Brownian motion. Likewise, in psychology [and linguistics!], modeling at this level has allowed theorists to identify common principles across seemingly disparate areas. That said, we believe that in most instances, cognitive scientists would ultimately prefer an explanatory process model over mere characterization." (Lewandowsky and Farrell, 2010, p. 19)

3 The very basics: Defining an agent, a model / environment, and running the model

If you want to create an agent for your model, the first step is to define its class. The following example defines a user agent class called `MyAgent` (a 'class' is a template for creating an object).

The lines below are `import` statements for including the `ccm` library, which provides all the necessary Python ACT-R classes for modeling.

(2) File **example_1.py**:

```
import ccm # import ccm module library for Python ACT-R classes
from ccm.lib.actr import *
```

We then declare a user defined agent class called `MyAgent`, which is inherited from the CCMSuite `ACTR` class. A class is a type. To get a token, i.e., an object of that class, the class must be instantiated. An agent is created with a production module by default. The production module is a part of the ACTR class we're inheriting from. So the user can define production rules as a part of the agent without any additional setup.

(3) File **example_1.py** (ctd.):

```
class MyAgent(ACTR):
    goal = Buffer() # Creating the goal buffer for the agent
    def init(): # this rule fires when the agent is instantiated.
        goal.set("sandwich bread") # set goal buffer to direct program flow
    def bread_bottom(goal="sandwich bread"): # if goal="sandwich bread" , fire rule
        print "I have a piece of bread"
        goal.set("stop") # set goal buffer to direct program flow
    def stop_production(goal="stop"):
        self.stop() # stop the agent
```

Let's examine our class in detail. The line `goal=Buffer()` creates the goal buffer for the agent. (The goal buffer is sometimes referred to as the focus buffer in the Python implementation of ACT-R.) Buffers are created by calling the `Buffer()` class. These are simple ACT-R buffers which are able to store a single chunk. They are the main system for communicating between modules. You can create as many as you like and give them any names you like, e.g., `goal = Buffer()`, `retrieval = Buffer()`, `imaginal = Buffer()`. Traditionally, ACT-R models have included a goal buffer, a retrieval buffer, and a visual buffer. More recently this has been expanded to include an imaginal buffer as well. (Apparently, there's a tradition for ACT-R buffers to end in "-al".)

The initialization production rule `init()` is the first production rule that an agent will execute when the agent object is instantiated (created). In this example, the action of the rule is to set the goal buffer to `"sandwich bread"` using the buffer's `set` method. In general, a buffer can be set by its `set` method. A method is a function that is associated with / part of an object.

The main purpose of the goal buffer is to represent the model's central executive attention function and it is used to guide or direct the flow of the cognitive task. Typically, inside a production rule, the goal buffer is set to the value of the buffer condition for the next step in the cognitive task.

For example, the `init` rule sets the goal buffer to `"sandwich bread"`, which effectively directs the agent to the `bread_bottom` production rule below that has its (pre)condition for firing that the goal buffer is equal to `"sandwich bread"`. The actions of this rule are to print the string `"I have a piece of bread"` to the console, and then to set the goal buffer to the condition of the next rule that should be applied in the cognitive task. In this very simple case, the task is basically over: the `stop` production rule can fire now, which stops the agent.

Every ACT-R agent automatically has a procedural system / module. This is what causes productions which match the current situation to 'fire', i.e., to perform actions by passing messages to modules or changing buffer values. The default procedural system is a very simple one that does not perform any learning to adjust the utility of each production, so if more than one production can fire at a given time, one will be chosen at random. By default, productions require 0.05 seconds (50 ms) to fire. This can be adjusted by setting the `production_time` value, as shown in the example below. We can also set `production_time_sd` to adjust the standard deviation of the production times (the default is 0). Finally, we can also set the `production_threshold`, which indicates that a production must have a utility above this threshold before it will fire. There are a variety of ways to change how the production system works; we'll talk more about this, and about production utilities and utility learning in a later section.

(4)
```
class SimpleModel(ACTR):
    production_time = 0.05
    production_sd = 0.01
    production_threshold = -20
```

Let's see how this works. We instantiate the class, i.e., we create a specific agent `tim`:

```
[py1] >>> from example_1 import * # we import our agent and environment classes
      >>> tim = MyAgent()
```

Python ACT-R always requires an environment / model in which the agents needs to be placed, but in this case we will not be using anything in the environment so we 'pass' on putting things in there.

(5) File **example_1.py** (ctd.):

```
class MyEnvironment(ccm.Model):
    pass
```

We instantiate the environment and put the agent `tim` in it:

```
[py2] >>> empty_environment = MyEnvironment()
      >>> empty_environment.agent = tim
```

We log / print out everything that happens in the environment:

```
[py3] >>> ccm.log_everything(empty_environment)
         0.000 agent.production_threshold None
         0.000 agent.production_time_sd None
         0.000 agent.production_match_delay 0
         0.000 agent.production_time 0.05
         0.000 agent.goal.chunk None
```

We now run the environment / simulation:

```
[py4] >>> empty_environment.run()
         0.000 agent.goal.chunk sandwich bread
         0.000 agent.production bread_bottom
         0.050 agent.production None
      I have a piece of bread
         0.050 agent.goal.chunk stop
         0.050 agent.production stop_production
         0.100 agent.production None
```

Note how modules – in this example, the procedural module, which is the only module built into any Python ACT-R agent by default – are activated instantaneously when conditions satisfy their use; instantaneously in simulated time (this obviously requires some machine time to compute).

Once activated, modules calculate the total cost in time for the action they are scheduled to do. This is the reason for the third line in the above output `0.050 agent.production None`, which is emitted at time 0.000 s, but lists the projected action for the immediately following time period when the procedural module could schedule another action. Given the current context (i.e., the current contents of all the buffers the agent has) and the production rules the agent possesses, nothing can actually be scheduled for the next time period.

Once the 50 ms required for the `bread_bottom` rule to fire elapse, the actions associated with this rule are performed instantaneously (in simulated time), i.e., we print `"I have a piece of bread"` and update the chunk in the goal / goal buffer to `"stop"`. The production rule `stop_production` can now be activated – and it is, also instantaneously, i.e., at the same time step 0.050 s. Since a new production is activated, the procedural module calculates the necessary time to perform it (50 ms) and lists what production rule can be performed once this time has elapsed: `0.100 agent.production None`. Just as before, no production can be triggered given the current context.

The `stop_production` rule instructs the agent to stop, so the current simulation run terminates at time step 0.100 s.This is not explicitly listed, it is implicit given the schedule for the next time step, namely `0.100 agent.production None`, provided by the procedural module.

So now we can stop the whole environment / simulation:

```
[py5] >>> ccm.finished()
```

One note about the way chunks – i.e., sets of slot-value pairs, or in non-ACT-R but more standard terminology, attribute-value matrices (AVMs) – are represented in Python ACT-R. The full representation for the chunk `"sandwich bread"` is in fact something like `"goal:sandwich object:bread"`, where `"goal:... object:..."` are the slots / features / attributes, and `"...:sandwich ...:bread"` are the values. When the chunk is represented simply in terms of the sequence of values, i.e., `"sandwich bread"`, the attributes are implicitly introduced as 'attribute 1', 'attribute 2', etc., i.e., the chunk is treated as a partial variable assignment with the variables distinguished by their position in the sequence. Chunks / AVMs are really dictionaries (in the Python sense, i.e., hash tables), which are really just souped up partial variable assignments (mathematically).

In ACT-R, chunks have a limited number of slots ($7 + / - 2$), so they are rather small partial variable assignments. The values of their slots can be other chunks though, and this is how complex structures (e.g., syntactic trees) can be represented – very much like in GPSG and HPSG.

4 Central Executive Control

Let's understand a bit better the output of the model.

The central executive control of an agent is one of the core components of the ACT-R theory. The main function of the central executive control is to represent 'consciousness' (in a somewhat technical term of monitoring the contents of all buffers) and intentionality in the agent.

When an agent is created, the central executive control function starts to operate as the agent's awareness. It continuously examines the buffers and the module states to determine if a particular production rule is to be activated.

A production rule is "fired" or activated when the conditions of the rule are completely matched, i.e., when all the buffer values are matched with the conditions stated in the condition-part of a production rule. For example, the production rule `bread_bottom` in the above example has as its only condition `goal="sandwich bread"`, and its actions are printing a message and resetting the goal / goal buffer.

Production rules are activated one rule at a time and each firing consumes 50 ms (0.05 s). For more complex models, multiple rules could be activated in parallel in different agents or modules.

If no productions match, then the production system does nothing. However, as soon as the content of a buffer or a module state changes (see below for more discussion of module states wrt the Declarative Memory module) such that a production rule could fire, the production system notices this and instantaneously (in simulated time) readies that production to fire 50 ms later.

Thus, the production system performs its search for matching patterns every time there is a change in the cognitive context, i.e., in the buffers, or in module states, and it is not currently waiting to fire a previously scheduled production.

The production system also checks that the contents of the relevant buffers and module states that were preconditions for the currently scheduled production rule are still a valid match after the 50 ms delay. That is, even if the contents of the buffers have changed since the beginning of the 50 ms delay, the changes should not affect the relevant condition pattern matches or the production could be inappropriate. If this change occurs in Python ACT-R, the production selection process is restarted after the

50 ms without firing the previously scheduled rule.[2]

The central executive control and the production rules together are responsible for the algorithm of the program/agent. In Example 1, the rules are fired or executed according to the sequence set by the goal buffer.

5 Declarative Memory

Declarative memory refers to memories that can be consciously recalled and discussed.[3] Declarative memory is subject to forgetting: memories have an activation level that decays over time. Both the probability and the latency of retrieving / recalling a declarative memory depend on its activation value.

In ACT-R, declarative memories are represented / formalized as "chunks", which are attribute-value matrices (AVMs, or feature-value matrices) of the kind used in Generalized Phrase Structure Grammar (GPSG), Lexical Function Grammar (LFG), or Head-driven Phrase Structure Grammar (HPSG).

Chunks are used to communicate / transfer information between modules through buffers. Buffers are the 'interfaces' of the various modules we conjecture the human mind has; e.g., the goal buffer is the interface of the procedural memory module (the production system), while the retrieval buffer is the interface of the declarative memory module. Buffers can store only one chunk at any given time.

5.1 A very simple example

Declarative memories are the data store for the agent. In Python ACT-R, the agent can remember information by retrieving chunks from declarative memory, or learn new information by creating new chunks and adding them to the declarative memory.

Our Example 2 shows how an agent can store and retrieve information from declarative memory.

(6) File **example_2.py**:

```
import ccm
from ccm.lib.actr import *


class MyAgent(ACTR):

    goal = Buffer()
    # create a buffer for the declarative memory
    DMBuffer = Buffer()
    # create a declarative memory object called DM
    DM = Memory(DMBuffer)

    def init():
        # put a chunk into DM; has a slot/feature/attribute "condiment"
        DM.add("condiment:mustard")
        # set goal buffer to trigger recalling rule
        goal.set("get_condiment")
```

[2]Apparently, it's unclear when the process is restarted in Lisp ACT-R – immediately upon a buffer change or after the 50 ms have elapsed.

[3]Note that we do not distinguish between short-term memory and long-term memory, or between 'semantic' / 'general' memory (e.g., the information that *Robins are birds*) and episodic / 'specific' (e.g., indexed by a particular temporal and spatial location) memory. The only distinction that ACT-R explicitly makes is the distinction between declarative memory (which stores chunks / AVMs) and procedural memory (which stores rules / productions).

```
        def recalling(goal="get_condiment"):
            print "Retrieving the condiment chunk from DM into the DM buffer"
            # the value "?" means that the slot can match any content
            DM.request("condiment:?")
            # set goal buffer to trigger condiment rule
            goal.set("sandwich condiment")

        # Note how DMBuffer is used as a condition for the rule below
        def condiment(goal="sandwich condiment", DMBuffer="condiment:?condiment"):
            print "My recall of the condiment from memory is...."
            # the condiment variable we print is associated with the ?condiment variable
            print condiment
            # set goal buffer to trigger stop_production rule
            goal.set("stop")

        def stop_production(goal="stop"):
            self.stop()


    class MyEnvironment(ccm.Model):
        pass
```

Example 2 is an extension of Example 1. It includes a declarative memory module `DM` and the associated buffer `DMBuffer`. We use the `Buffer()` class to create `DMBuffer`, a new buffer for the memory object. We then use the `Memory()` class and `DMBuffer` to create a declarative memory object called `DM` for the agent, which will serve as the agent's declarative memory module.

We then add a new chunk to the declarative memory by using the `add` method of the memory object `DM`. Note that a single quoted string is used as a parameter for the method. The quoted string represents a chunk, i.e., a list of slot-value pairs. Just as before, the ":" inside the string is a separator which is used to separate the slot name from its value – `"condiment:mustard"` denotes a chunk with a single slot called `condiment`, the value of which is `mustard`.

Retrieving a value from declarative memory and doing something with it takes two steps, usually occurring in two distinct rules:

- the `DM.request` command in the `recalling` production rule makes a request to the memory object `DM` using a cue (slot name), `condiment` in our case; the "?" notation in the memory request indicates that the value of the slot can be anything as long as the chunk has a `condiment` slot;

- the rule `condiment(goal="sandwich condiment", DMBuffer="condiment:?condiment")` processes the resulting chunk, which has been retrieved from memory and is now stored and available in the memory buffer; the meaning of "?" changes here: when "?" precedes a name, it defines a variable for the value of the slot, `?condiment` in our case

Variables can then be used by the agent for further processing. In our example, the variable `?condiment` is assigned the value `mustard` by the chunk that has been retrieved from the declarative memory module and placed in the `DMBuffer`. We use the `?condiment` variable later on – but without the initial "?" – to print this value.

Once a Python ACT-R variable is assigned a value, that value remains fixed for the life of the variable. (Note that this is not true of Python in general, just of Python ACT-R.)

Any variable from the left-hand / condition side of a production rule can be used on the right-hand / action side, but the variable-value pair is discarded after the production is fired. This means that the expressive power of ACT-R variables is limited in (cognitive) time, and also in (cognitive) space – variables created within the production system, i.e., procedural momery / module, cannot be used outside of it.

(7) **Convention**: the initial "?" that indicates something is an ACT-R variable is dropped before these variables are used with regular Python commands like `print`.
For example, the value `mustard` will be printed to the console as a result of the `print condiment` statement in the `condiment` rule.

(8) **Convention**: we name variables over values by using the corresponding slot name, i.e., `"<slot name>:?<slot name>"`.
For example, this why the slot `condiment` in the `DMBuffer` chunk has as its value the variable `?condiment`.

The rest of the Example 2 model is very similar to the one in Example 1, so we won't discuss it in detail.

Let's run the simulation. We instantiate the environment and put the agent `tim` in it:

[**py6**]
```
>>> from example_2 import *
>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
```

Then we run the simulation:

[**py7**]
```
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold None
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_time 0.05
   0.000 agent.DM.record_all_chunks False
   0.000 agent.DM.threshold 0
   0.000 agent.DM.latency 0.05
   0.000 agent.DM.busy False
   0.000 agent.DM.maximum_time 10.0
   0.000 agent.DM.error False
   0.000 agent.goal.chunk None
   0.000 agent.DMBuffer.chunk None
>>> empty_environment.run()
   0.000 agent.goal.chunk get_condiment
   0.000 agent.production recalling
   0.050 agent.production None
Retrieving the condiment chunk from DM into the DM buffer
   0.050 agent.DM.busy True
   0.050 agent.goal.chunk sandwich condiment
   0.100 agent.DMBuffer.chunk condiment:mustard
   0.100 agent.DM.busy False
   0.100 agent.production condiment
   0.150 agent.production None
```

```
My recall of the condiment from memory is....
mustard
   0.150 agent.goal.chunk stop
   0.150 agent.production stop_production
   0.200 agent.production None
>>> ccm.finished()
```

The DM module has various parameters, listed in the initialization part of the model (the output of the `ccm.log_everything(empty_environment)` call). We will discuss them in a later section.

For now, note only that during the simulation run (the output of the `empty_environment.run()` call), we also keep track of whether the DM module is busy with a retrieval request or not (`0.050 agent.DM.busy True` and `0.100 agent.DM.busy False`).

This is important, because production rules can have such module-state specifications as part of their firing-condition patterns. That is, production-rule conditions in ACT-R can include 2 kinds of pattern matches:

i matches to the cognitive context, i.e., to the state of any of the buffers, and

ii matches to the state of the modules.

To accomplish matches to module states, a module could include a separate buffer that holds information about its state, with slots and values specific to each module. This buffer would include information about whether the module is currently performing an action or not. In Python ACT-R, we don't need to explicitly add such buffers – we can pattern match on module states directly. But in Lisp ACT-R, requests are sent to modules by placing the requests in a *separate* buffer associated with the module. For DM, for example, this buffer is different from the retrieval buffer. In Python ACT-R, the requests are sent directly to the module. Using the separate buffer in Lisp ACT-R does not impose any time delay, so the only constraint it imposes is that the request should be a chunk (hence, of limited size). This constraint is the same for Python ACT-R.

5.2 A slightly more complex example

We now turn to a slightly more complicated model – a model with several more productions. The most important thing to note is how the rules are fired according to the sequence set by the goal buffer.

In particular, the `condiment(goal="get_condiment")` production first requests the declarative memory module to retrieve the condiment that the customer ordered. This condiment has been stored in the declarative memory. The `order(goal="sandwich condiment", DMBuffer="condiment:?condiment")` production fires only when this retrieval has happened.

(9) File **example_3.py**:

```
import ccm
from ccm.lib.actr import *


class MyAgent(ACTR):

    goal = Buffer()
    DMBuffer = Buffer()
    DM = Memory(DMBuffer)

    def init():
        DM.add("condiment:mustard") # put a chunk into DM
        goal.set("sandwich bread")

    def bread_bottom(goal="sandwich bread"):
        print "I have a piece of bread"
        goal.set("sandwich cheese")

    def cheese(goal="sandwich cheese"):
        print "I just put cheese on the bread"
        goal.set("sandwich ham")

    def ham(goal="sandwich ham"):
        print "I just put ham on the cheese"
        goal.set("get_condiment")

    def condiment(goal="get_condiment"):
        print "Recalling the order"
        # retrieve a chunk from DM into the DM buffer; ? means that the slot
        # can match any content
        DM.request("condiment:?")
        goal.set("sandwich condiment")

    # this production rule matches to DMBuffer in addition to goal and assigns
    # the retrieved value of the condiment slot to the variable ?condiment
    def order(goal="sandwich condiment", DMBuffer="condiment:?condiment"):
        print "I recall they wanted .......", condiment
        print "I just put", condiment, "on the ham"
        goal.set("sandwich bread_top")

    def bread_top(goal="sandwich bread_top"):
        print "I just put bread on the ham"
        print "\nI made a ham and cheese sandwich!\n"
        goal.set("stop")

    def stop_production(goal="stop"):
        self.stop()


class MyEnvironment(ccm.Model):
    pass
```

The way we should think of the goal chunks in this model is as listing the main goal and the current subgoal. For example, when we initialize the goal buffer in the `init()` rule, we set it to the chunk `"sandwich bread"`, which simply lists two values and implicitly associates them with two distinct slots / features / attributes `"_0"` and `"_1"`. That is, `goal.set("sandwich bread")` is just a convenient abbreviation for `goal.set("_0:sandwich _1:bread")`. This convenient abbreviation is specific to Python ACT-R. In Lisp ACT-R, slot names need to be listed explicitly.

But the way we should intuitively think about this goal chunk is as having more explanatory slot names along these lines: `goal.set("main-goal:sandwich current-subgoal:bread")`. That is, we want to make a sandwich, and we are currently looking for the first piece of the sandwich, namely the bottom slice of bread. Lisp ACT-R uses 'explanatory' slot names like these, e.g., the `"isa"` slot that intuitively lists the type of the chunk. There is no actual semantics associated with slot names, so in an effort to be as transparent as possible about the inner workings of the system, Python ACT-R allows us to completely do away with explicitly listing slot names.

The resulting code might be less user-friendly, but it is very explicit about what the model actually does and doesn't do; that's the reason we use it here. Feel free to use whatever conventions you want in your own modeling. For example, if you were to show the code to non-specialists, adding intuitive slot names would probably be a very good idea. Just be fully aware of the capabilities and limitations of your models.

The production rule `bread_bottom` is waiting exactly for this kind of `"sandwich bread"` chunk in the goal buffer. Once the `bread_bottom` rule fires, it resets the goal buffer to a new chunk, namely `"sandwich cheese"`. Again, the way we should interpret this is as introducing a more verbose chunk along the lines of `goal.set("main-goal:sandwich current-subgoal:cheese")`. This goal chunk in turn triggers the production rule `cheese`, which will again modify the goal buffer. And so on, until the sandwich is done and we trigger the `stop` rule.

Note that in this model, we encode multiple goals or goal-subgoal assemblies by means of multiple slots in the goal buffer chunk. This is one way to do it. Another way is to make use of a goal stack and push / pop (sub)goals from that stack. The goal buffer will then store the currently active (sub)goal. Both techniques are used in ACT-R, sometimes simultaneously – see Anderson and Lebiere (1998) for more details.

Let's run the simulation:

[**py8**]
```
>>> from example_3 import *
>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold None
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_time 0.05
   0.000 agent.DM.record_all_chunks False
   0.000 agent.DM.threshold 0
   0.000 agent.DM.latency 0.05
   0.000 agent.DM.busy False
   0.000 agent.DM.maximum_time 10.0
   0.000 agent.DM.error False
   0.000 agent.goal.chunk None
   0.000 agent.DMBuffer.chunk None
>>> empty_environment.run()
   0.000 agent.goal.chunk sandwich bread
   0.000 agent.production bread_bottom
   0.050 agent.production None
I have a piece of bread
   0.050 agent.goal.chunk sandwich cheese
   0.050 agent.production cheese
   0.100 agent.production None
I just put cheese on the bread
   0.100 agent.goal.chunk sandwich ham
   0.100 agent.production ham
   0.150 agent.production None
I just put ham on the cheese
   0.150 agent.goal.chunk get_condiment
   0.150 agent.production condiment
   0.200 agent.production None
Recalling the order
   0.200 agent.DM.busy True
   0.200 agent.goal.chunk sandwich condiment
   0.250 agent.DMBuffer.chunk condiment:mustard
   0.250 agent.DM.busy False
   0.250 agent.production order
   0.300 agent.production None
I recall they wanted ....... mustard
I just put mustard on the ham
   0.300 agent.goal.chunk sandwich bread_top
   0.300 agent.production bread_top
   0.350 agent.production None
I just put bread on the ham

I made a ham and cheese sandwich!

   0.350 agent.goal.chunk stop
   0.350 agent.production stop_production
   0.400 agent.production None
>>> ccm.finished()
```

6 ACT-R unit 1 tutorials

These are the official Lisp ACT-R unit 1 tutorials rewritten in Python ACT-R and available in basic form as part of the `ccmsuite`. The original `ccmsuite` version of these tutorials has been modified here in mostly minor respects.

6.1 Counting

(10) File **example_4_counting.py**:

```
import ccm
from ccm.lib.actr import *


class Count(ACTR):
    goal = Buffer()
    DMBuffer = Buffer()
    DM = Memory(DMBuffer)

    def init():
        DM.add('count 0 1')
        DM.add('count 1 2')
        DM.add('count 2 3')
        DM.add('count 3 4')
        DM.add('count 4 5')
        DM.add('count 5 6')
        DM.add('count 6 7')
        DM.add('count 7 8')
        DM.add('count 8 9')
        DM.add('count 9 10')

    def start(goal='countFrom ?start ?end starting'):
        DM.request('count ?start ?next')
        goal.set('countFrom ?start ?end counting')

    def increment(goal='countFrom ?x !?x counting',
                  DMBuffer='count ?x ?next'):
        print "\n>> The current number is:", x, "\n"
        DM.request('count ?next ?nextNext')
        goal.modify(_1=next)

    def stop(goal='countFrom ?x ?x counting'):
        print "\n>> The final number is:", x, "\n"
        goal.set('countFrom ?x ?x stop')


class MyEnvironment(ccm.Model):
    pass
```

[**py9**]
```
>>> from example_4_counting import *
>>> counter = Count()
>>> counter.goal.set('countFrom 2 5 starting') # count from 2 to 5
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = counter
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold None
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_time 0.05
   0.000 agent.DM.record_all_chunks False
   0.000 agent.DM.threshold 0
   0.000 agent.DM.latency 0.05
   0.000 agent.DM.busy False
   0.000 agent.DM.maximum_time 10.0
   0.000 agent.DM.error False
   0.000 agent.DMBuffer.chunk None
>>> empty_environment.run()
   0.000 agent.production start
   0.050 agent.production None
   0.050 agent.DM.busy True
   0.050 agent.goal.chunk countFrom 2 5 counting
   0.100 agent.DMBuffer.chunk count 2 3
   0.100 agent.DM.busy False
   0.100 agent.production increment
   0.150 agent.production None

>> The current number is: 2

   0.150 agent.DM.busy True
   0.150 agent.goal.chunk countFrom 3 5 counting
   0.200 agent.DMBuffer.chunk count 3 4
   0.200 agent.DM.busy False
   0.200 agent.production increment
   0.250 agent.production None

>> The current number is: 3

   0.250 agent.DM.busy True
   0.250 agent.goal.chunk countFrom 4 5 counting
   0.300 agent.DMBuffer.chunk count 4 5
   0.300 agent.DM.busy False
   0.300 agent.production increment
   0.350 agent.production None

>> The current number is: 4

   0.350 agent.DM.busy True
   0.350 agent.goal.chunk countFrom 5 5 counting
   0.350 agent.production stop
   0.400 agent.DMBuffer.chunk count 5 6
   0.400 agent.DM.busy False
   0.400 agent.production None

>> The final number is: 5

   0.400 agent.goal.chunk countFrom 5 5 stop
>>> ccm.finished()
```

6.2 Addition

(11) File **example_5_addition.py**:

```
import ccm
from ccm.lib.actr import *


class Addition(ACTR):

    goal = Buffer()
    DMBuffer = Buffer()
    DM = Memory(DMBuffer)

    def init():
        DM.add('count 0 1')
        DM.add('count 1 2')
        DM.add('count 2 3')
        DM.add('count 3 4')
        DM.add('count 4 5')
        DM.add('count 5 6')
        DM.add('count 6 7')
        DM.add('count 7 8')

    def initializeAddition(goal='add ?num1 ?num2 count:None?count sum:None?sum'):
        goal.modify(count=0,sum=num1)
        DM.request('count ?num1 ?next')

    def terminateAddition(goal='add ?num1 ?num2 count:?num2 sum:?sum'):
        goal.set('result ?sum')
        print "\n>> The final sum is:", sum, "\n"

    def incrementSum(goal='add ?num1 ?num2 count:?count!?num2 sum:?sum',
                     DMBuffer='count ?sum ?next'):
        goal.modify(sum=next)
        DM.request('count ?count ?n2')

    def incrementCount(goal='add ?num1 ?num2 count:?count sum:?sum',
                       DMBuffer='count ?count ?next'):
        goal.modify(count=next)
        DM.request('count ?sum ?n2')


class MyEnvironment(ccm.Model):
    pass
```

[**py10**]
```
>>> from example_5_addition import *
>>> adder = Addition()
>>> adder.goal.set('add 5 2 count:None sum:None')
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = adder
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold None
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_time 0.05
   0.000 agent.DM.record_all_chunks False
   0.000 agent.DM.threshold 0
   0.000 agent.DM.latency 0.05
   0.000 agent.DM.busy False
   0.000 agent.DM.maximum_time 10.0
   0.000 agent.DM.error False
   0.000 agent.DMBuffer.chunk None
>>> empty_environment.run()
```

```
  0.000 agent.production initializeAddition
  0.050 agent.production None
  0.050 agent.goal.chunk add 5 2 count:0 sum:None
  0.050 agent.goal.chunk add 5 2 count:0 sum:5
  0.050 agent.DM.busy True
  0.100 agent.DMBuffer.chunk count 5 6
  0.100 agent.DM.busy False
  0.100 agent.production incrementSum
  0.150 agent.production None
  0.150 agent.goal.chunk add 5 2 count:0 sum:6
  0.150 agent.DM.busy True
  0.200 agent.DMBuffer.chunk count 0 1
  0.200 agent.DM.busy False
  0.200 agent.production incrementCount
  0.250 agent.production None
  0.250 agent.goal.chunk add 5 2 count:1 sum:6
  0.250 agent.DM.busy True
  0.300 agent.DMBuffer.chunk count 6 7
  0.300 agent.DM.busy False
  0.300 agent.production incrementSum
  0.350 agent.production None
  0.350 agent.goal.chunk add 5 2 count:1 sum:7
  0.350 agent.DM.busy True
  0.400 agent.DMBuffer.chunk count 1 2
  0.400 agent.DM.busy False
  0.400 agent.production incrementCount
  0.450 agent.production None
  0.450 agent.goal.chunk add 5 2 count:2 sum:7
  0.450 agent.DM.busy True
  0.450 agent.production terminateAddition
  0.500 agent.DMBuffer.chunk count 7 8
  0.500 agent.DM.busy False
  0.500 agent.production None
  0.500 agent.goal.chunk result 7

>> The final sum is: 7

>>> ccm.finished()
```

6.3 Multicolumn addition

(12) File **example_6_multicolumn_addition.py**:

```
import ccm
from ccm.lib.actr import *


class Addition(ACTR):
    goal = Buffer()
    DMBuffer = Buffer()
    DM = Memory(DMBuffer )
    def init():
        DM.add('addfact 3 4 7')
        DM.add('addfact 6 7 13')
        DM.add('addfact 10 3 13')
        DM.add('addfact 1 7 8')

    def startPair(goal='add ? ?one1 ? ?one2 ? None?ans ?'):
        goal.modify(_6='busy')
        DM.request('addfact ?one1 ?one2 ?')

    def addOnes(goal='add ? ? ? ? ? busy?ans ?carry', DMBuffer='addfact ? ? ?sum'):
        goal.modify(_6=sum,_7='busy')
        DM.request('addfact 10 ? ?sum')

    def processCarry(goal='add ?ten1 ? ?ten2 ? None?tenAns ?oneAns busy?carry',DMBuffer='addf
        goal.modify(_6=rem,_7=1,_5='busy')
        DM.request('addfact ?ten1 ?ten2 ?')

    def noCarry(goal='add ?ten1 ? ?ten2 None?tenAns ?oneAns busy?carry',DM='error:True'):
        goal.modify(_6=0,_4='busy')
        DM.request('addfact ?ten1 ?ten2 ?')

    def addTensDone(goal='add ? ? ? ? busy?tenAns ?oneAns 0',DMBuffer='addfact ? ? ?sum'):
        print "\n>> The final sum is: tens --", sum, ", ones -- ", oneAns, "\n"
        goal.modify(_5=sum)

    def addTensCarry(goal='add ? ? ? ? busy?tenAns ? 1?carry',DMBuffer='addfact ? ? ?sum'):
        goal.modify(_7=0)
        DM.request('addfact 1 ?sum ?')


class MyEnvironment(ccm.Model):
    pass
```

[**py11**]
```
>>> from example_6_multicolumn_addition import *
>>> adder = Addition()
>>> adder.goal.set('add 3 6 4 7 None None None') # add 36 and 47
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = adder
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold None
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_time 0.05
   0.000 agent.DM.record_all_chunks False
   0.000 agent.DM.threshold 0
   0.000 agent.DM.latency 0.05
   0.000 agent.DM.busy False
   0.000 agent.DM.maximum_time 10.0
   0.000 agent.DM.error False
   0.000 agent.DMBuffer.chunk None
>>> empty_environment.run()
   0.000 agent.production startPair
   0.050 agent.production None
   0.050 agent.goal.chunk add 3 6 4 7 None busy None
   0.050 agent.DM.busy True
   0.100 agent.DMBuffer.chunk addfact 6 7 13
   0.100 agent.DM.busy False
   0.100 agent.production addOnes
   0.150 agent.production None
   0.150 agent.goal.chunk add 3 6 4 7 None busy busy
   0.150 agent.goal.chunk add 3 6 4 7 None 13 busy
   0.150 agent.DM.busy True
   0.200 agent.DMBuffer.chunk addfact 10 3 13
   0.200 agent.DM.busy False
   0.200 agent.production processCarry
   0.250 agent.production None
   0.250 agent.goal.chunk add 3 6 4 7 None 13 1
   0.250 agent.goal.chunk add 3 6 4 7 None 3 1
   0.250 agent.goal.chunk add 3 6 4 7 busy 3 1
   0.250 agent.DM.busy True
   0.250 agent.production addTensCarry
   0.300 agent.DMBuffer.chunk addfact 3 4 7
   0.300 agent.DM.busy False
   0.300 agent.production None
   0.300 agent.goal.chunk add 3 6 4 7 busy 3 0
   0.300 agent.DM.busy True
   0.300 agent.production addTensDone
   0.350 agent.DMBuffer.chunk addfact 1 7 8
   0.350 agent.DM.busy False
   0.350 agent.production None

>> The final sum is: tens -- 8 , ones --  3

   0.350 agent.goal.chunk add 3 6 4 7 8 3 0
>>> ccm.finished()
```

6.4 The ‘semantic’ model (navigating super/sub-category networks)

This model contains chunks encoding a network of categories and properties. Its productions are capable of searching this network to make decisions about whether one category is a member of another category.

(13) File **example_7_semantic.py**:

```
import ccm
from ccm.lib.actr import *


class Semantic(ACTR):
    goal = Buffer()
    DMBuffer = Buffer()
    DM = Memory(DMBuffer)

    #text=TextOutput()

    def init():
        DM.add('property shark dangerous true')
        DM.add('property shark locomotion swimming')
        DM.add('property shark category fish')
        DM.add('property fish category animal')
        DM.add('property bird category animal')
        DM.add('property canary category bird')

    def initialRetrieve(goal='isMember ?obj ?cat result:None'):
        goal.modify(result='pending')
        DM.request('property ?obj category ?')

    def directVerify(goal='isMember ?obj ?cat result:pending',
                    DMBuffer='property ?obj category ?cat'):
        goal.modify(result='yes')
        #text.write('\n>> Final answer: Yes\n')
        print '\n>> Final answer: Yes\n'

    def chainCategory(goal='isMember ?obj1 ?cat result:pending',
                        DMBuffer='property ?obj1 category ?obj2!?cat'):
        goal.modify(_1=obj2)
        DM.request('property ?obj2 category ?')

    def fail(goal='isMember ?obj1 ?cat result:pending',DM='error:True'):
        goal.modify(result='no')
        #text.write('\n>> Final answer: No\n')
        print '\n>> Final answer: No\n'


class MyEnvironment(ccm.Model):
    pass
```

[**py12**]
```
>>> from example_7_semantic import *

>>> print '\n>>>>>>>> Starting simulation 1 <<<<<<<<\n'

>>>>>>>> Starting simulation 1 <<<<<<<<

>>> decider = Semantic()
>>> decider.goal.set('isMember shark fish result:None')
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = decider
>>> ccm.log_everything(empty_environment)
```

```
   0.000 agent.production_threshold None
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_time 0.05
   0.000 agent.DM.record_all_chunks False
   0.000 agent.DM.threshold 0
   0.000 agent.DM.latency 0.05
   0.000 agent.DM.busy False
   0.000 agent.DM.maximum_time 10.0
   0.000 agent.DM.error False
   0.000 agent.DMBuffer.chunk None
>>> empty_environment.run()
   0.000 agent.production initialRetrieve
   0.050 agent.production None
   0.050 agent.goal.chunk isMember shark fish result:pending
   0.050 agent.DM.busy True
   0.100 agent.DMBuffer.chunk property shark category fish
   0.100 agent.DM.busy False
   0.100 agent.production directVerify
   0.150 agent.production None
   0.150 agent.goal.chunk isMember shark fish result:yes

>> Final answer: Yes

>>> ccm.finished()

>>> print '\n>>>>>>>> Starting simulation 2 <<<<<<<<\n'

>>>>>>>> Starting simulation 2 <<<<<<<<

>>> decider = Semantic()
>>> decider.goal.set('isMember shark animal result:None')
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = decider
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold None
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_time 0.05
   0.000 agent.DM.record_all_chunks False
   0.000 agent.DM.threshold 0
   0.000 agent.DM.latency 0.05
   0.000 agent.DM.busy False
   0.000 agent.DM.maximum_time 10.0
   0.000 agent.DM.error False
   0.000 agent.DMBuffer.chunk None
>>> empty_environment.run()
   0.000 agent.production initialRetrieve
   0.050 agent.production None
   0.050 agent.goal.chunk isMember shark animal result:pending
   0.050 agent.DM.busy True
   0.100 agent.DMBuffer.chunk property shark category fish
   0.100 agent.DM.busy False
   0.100 agent.production chainCategory
   0.150 agent.production None
   0.150 agent.goal.chunk isMember fish animal result:pending
   0.150 agent.DM.busy True
   0.200 agent.DMBuffer.chunk property fish category animal
   0.200 agent.DM.busy False
   0.200 agent.production directVerify
   0.250 agent.production None
   0.250 agent.goal.chunk isMember fish animal result:yes

>> Final answer: Yes

>>> ccm.finished()

>>> print '\n>>>>>>>> Starting simulation 3 <<<<<<<<\n'

>>>>>>>> Starting simulation 3 <<<<<<<<

>>> decider = Semantic()
>>> decider.goal.set('isMember canary fish result:None')
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = decider
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold None
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_time 0.05
   0.000 agent.DM.record_all_chunks False
   0.000 agent.DM.threshold 0
   0.000 agent.DM.latency 0.05
   0.000 agent.DM.busy False
   0.000 agent.DM.maximum_time 10.0
   0.000 agent.DM.error False
   0.000 agent.DMBuffer.chunk None
>>> empty_environment.run()
   0.000 agent.production initialRetrieve
   0.050 agent.production None
   0.050 agent.goal.chunk isMember canary fish result:pending
   0.050 agent.DM.busy True
   0.100 agent.DMBuffer.chunk property canary category bird
   0.100 agent.DM.busy False
   0.100 agent.production chainCategory
   0.150 agent.production None
   0.150 agent.goal.chunk isMember bird fish result:pending
   0.150 agent.DM.busy True
   0.200 agent.DMBuffer.chunk property bird category animal
   0.200 agent.DM.busy False
```

```
    0.200 agent.production chainCategory
    0.250 agent.production None
    0.250 agent.goal.chunk isMember animal fish result:pending
    0.250 agent.DM.busy True
    0.300 agent.DM.error True
    0.300 agent.DMBuffer.chunk None
    0.300 agent.DM.busy False
    0.300 agent.production fail
    0.350 agent.production None
    0.350 agent.goal.chunk isMember animal fish result:no

>> Final answer: No

>>> ccm.finished()
```

7 The subsymbolic level for the procedural module

7.1 Simple utility learning

In an ACT-R model more than one production can match the buffer conditions. In this case the production with the highest utility level is chosen. Utility levels are learned through experience and through reward. Different forms of reinforcement learning have been used to model this in ACT-R. These are available through Python ACT-R, which also includes Q-Learning, which is used in the Clarion architecture.

Although these learning algorithms are different, they all seek to adjust the utility of productions to reflect how often they lead to a reward state, discounted by how long it took to get from the production to the reward state.

Noise on the utility functions is also very important as it controls how much exploration (i.e., not choosing the highest utility production) occurs.

We exemplify below with the (now standard / old) $P \cdot G - C$ utility learning rule in chapter 3 of Anderson and Lebiere (1998): the utility of a rule is the gain G for achieving the goal, set to 20 seconds by default,[4] times the probability P of achieving that goal, set to 1 by default, minus the cost of performing the rule, set to 0.05 seconds (50 ms) by default.

(14) File **example_8_simple_utility_learning.py**:

```
import ccm
from ccm.lib.actr import *


class MyAgent(ACTR):

    goal = Buffer()
```

[4]Think of it along the following lines: the goal is worth 20 seconds of the agent's time (about the average time needed to complete one experimental item, let's say). This measures utility in time units, not money, but you can implicitly convert between them in some fashion if you want. There are goals that are measured in time by default: getting a PhD degree is worth about 5 years of your time (well, much less than that if you simply count the work hours), or getting a journal article published is worth let's say about 1 year of your time (again, much less than that if you count only the work hours). But this can be generalized to everything: getting an ice cream is worth several minutes of your time, depending on however you convert your (work) time into money, and getting a speeding ticket is worth a *negative* amount of your time since you lose the time needed to generate the money for the fine.

```
# production parameter settings
production_time = 0.05
production_sd = 0.01
production_threshold = -20

# standard PG-C
pm_pgc = PMPGC()

def init():
    goal.set('sandwich bread')

def bread_bottom(goal='sandwich bread'):
    print "\n\n>>>>>>>> Starting a new sandwich <<<<<<<<\n"
    self.print_production_utilities()
    print ">> I have a piece of bread"
    goal.set('sandwich cheese')

def cheese(goal='sandwich cheese'):
    print ">> I just put cheese on the bread"
    goal.set('sandwich ham-or-prosciutto')

def ham(goal='sandwich ham-or-prosciutto'):
    print ">> I just put ham on the cheese"
    goal.set('sandwich bread_top ham')

# the prosciutto production competes with the ham production
def prosciutto(goal='sandwich ham-or-prosciutto'):
    print ">> I just put prosciutto on the cheese"
    goal.set('sandwich bread_top prosciutto')

def bread_top_ham(goal='sandwich bread_top ham'):
    print ">> I just put bread on the ham"
    print "\n>> I made a    HAM    and cheese sandwich!\n"
    # make another sandwich
    goal.set('sandwich bread')

def bread_top_prosciutto(goal='sandwich bread_top prosciutto'):
    print ">> I just put bread on the prosciutto"
    print "\n>> I made a    PROSCIUTTO    and cheese sandwich!\n"
    # make another sandwich
    goal.set('sandwich bread')

def print_production_utilities(self):
    utilities_dict = self.get_activation()
    print "Current production utilities:"
    print "----------------------------"
    for key, value in utilities_dict.items():
        print key, "-->", round(value, 3)
    print "\n"
```

```
class MyEnvironment(ccm.Model):
    pass
```

[py13]
```
>>> from example_8_simple_utility_learning import *

>>> print '\n>>>>>>>> Starting simulation 1 <<<<<<<<\n'

>>>>>>>> Starting simulation 1 <<<<<<<<

>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold -20
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_sd 0.01
   0.000 agent.production_time 0.05
   0.000 agent.pm_pgc.goal 20
   0.000 agent.goal.chunk None
>>> #log = ccm.log(html=True)
>>> # run for 1.05 seconds
>>> empty_environment.run(1.05)
   0.000 agent.goal.chunk sandwich bread
   0.000 agent.production bread_bottom
   0.050 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.95
ham --> 19.95
prosciutto --> 19.95
bread_bottom --> 19.95
bread_top_ham --> 19.95
bread_top_prosciutto --> 19.95


>> I have a piece of bread
   0.050 agent.goal.chunk sandwich cheese
   0.050 agent.production cheese
   0.100 agent.production None
>> I just put cheese on the bread
   0.100 agent.goal.chunk sandwich ham-or-prosciutto
   0.100 agent.production ham
   0.150 agent.production None
```

```
>> I just put ham on the cheese
   0.150 agent.goal.chunk sandwich bread_top ham
   0.150 agent.production bread_top_ham
   0.200 agent.production None
>> I just put bread on the ham

>> I made a    HAM    and cheese sandwich!

   0.200 agent.goal.chunk sandwich bread
   0.200 agent.production bread_bottom
   0.250 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.95
ham --> 19.95
prosciutto --> 19.95
bread_bottom --> 19.95
bread_top_ham --> 19.95
bread_top_prosciutto --> 19.95


>> I have a piece of bread
   0.250 agent.goal.chunk sandwich cheese
   0.250 agent.production cheese
   0.300 agent.production None
>> I just put cheese on the bread
   0.300 agent.goal.chunk sandwich ham-or-prosciutto
   0.300 agent.production ham
   0.350 agent.production None
>> I just put ham on the cheese
   0.350 agent.goal.chunk sandwich bread_top ham
   0.350 agent.production bread_top_ham
   0.400 agent.production None
>> I just put bread on the ham

>> I made a    HAM    and cheese sandwich!

   0.400 agent.goal.chunk sandwich bread
   0.400 agent.production bread_bottom
   0.450 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
```

```
cheese --> 19.95
ham --> 19.95
prosciutto --> 19.95
bread_bottom --> 19.95
bread_top_ham --> 19.95
bread_top_prosciutto --> 19.95


>> I have a piece of bread
   0.450 agent.goal.chunk sandwich cheese
   0.450 agent.production cheese
   0.500 agent.production None
>> I just put cheese on the bread
   0.500 agent.goal.chunk sandwich ham-or-prosciutto
   0.500 agent.production prosciutto
   0.550 agent.production None
>> I just put prosciutto on the cheese
   0.550 agent.goal.chunk sandwich bread_top prosciutto
   0.550 agent.production bread_top_prosciutto
   0.600 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.600 agent.goal.chunk sandwich bread
   0.600 agent.production bread_bottom
   0.650 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.95
ham --> 19.95
prosciutto --> 19.95
bread_bottom --> 19.95
bread_top_ham --> 19.95
bread_top_prosciutto --> 19.95


>> I have a piece of bread
   0.650 agent.goal.chunk sandwich cheese
   0.650 agent.production cheese
   0.700 agent.production None
>> I just put cheese on the bread
   0.700 agent.goal.chunk sandwich ham-or-prosciutto
   0.700 agent.production ham
   0.750 agent.production None
>> I just put ham on the cheese
```

```
   0.750 agent.goal.chunk sandwich bread_top ham
   0.750 agent.production bread_top_ham
   0.800 agent.production None
>> I just put bread on the ham

>> I made a    HAM    and cheese sandwich!

   0.800 agent.goal.chunk sandwich bread
   0.800 agent.production bread_bottom
   0.850 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.95
ham --> 19.95
prosciutto --> 19.95
bread_bottom --> 19.95
bread_top_ham --> 19.95
bread_top_prosciutto --> 19.95


>> I have a piece of bread
   0.850 agent.goal.chunk sandwich cheese
   0.850 agent.production cheese
   0.900 agent.production None
>> I just put cheese on the bread
   0.900 agent.goal.chunk sandwich ham-or-prosciutto
   0.900 agent.production prosciutto
   0.950 agent.production None
>> I just put prosciutto on the cheese
   0.950 agent.goal.chunk sandwich bread_top prosciutto
   0.950 agent.production bread_top_prosciutto
   1.000 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   1.000 agent.goal.chunk sandwich bread
   1.000 agent.production bread_bottom
>>> ccm.finished()

>>> print '\n>>>>>>>> Starting simulation 2 <<<<<<<<\n'

>>>>>>>> Starting simulation 2 <<<<<<<<

>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold -20
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_sd 0.01
   0.000 agent.production_time 0.05
   0.000 agent.pm_pgc.goal 20
   0.000 agent.goal.chunk None
>>> #log = ccm.log(html=True)
>>> # run for 1.05 seconds
>>> empty_environment.run(1.05)
   0.000 agent.goal.chunk sandwich bread
   0.000 agent.production bread_bottom
   0.050 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.95
ham --> 19.95
prosciutto --> 19.95
bread_bottom --> 19.95
bread_top_ham --> 19.95
bread_top_prosciutto --> 19.95


>> I have a piece of bread
   0.050 agent.goal.chunk sandwich cheese
   0.050 agent.production cheese
   0.100 agent.production None
>> I just put cheese on the bread
   0.100 agent.goal.chunk sandwich ham-or-prosciutto
   0.100 agent.production prosciutto
   0.150 agent.production None
>> I just put prosciutto on the cheese
   0.150 agent.goal.chunk sandwich bread_top prosciutto
   0.150 agent.production bread_top_prosciutto
   0.200 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.200 agent.goal.chunk sandwich bread
   0.200 agent.production bread_bottom
   0.250 agent.production None
```

```
>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.95
ham --> 19.95
prosciutto --> 19.95
bread_bottom --> 19.95
bread_top_ham --> 19.95
bread_top_prosciutto --> 19.95


>> I have a piece of bread
   0.250 agent.goal.chunk sandwich cheese
   0.250 agent.production cheese
   0.300 agent.production None
>> I just put cheese on the bread
   0.300 agent.goal.chunk sandwich ham-or-prosciutto
   0.300 agent.production prosciutto
   0.350 agent.production None
>> I just put prosciutto on the cheese
   0.350 agent.goal.chunk sandwich bread_top prosciutto
   0.350 agent.production bread_top_prosciutto
   0.400 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.400 agent.goal.chunk sandwich bread
   0.400 agent.production bread_bottom
   0.450 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.95
ham --> 19.95
prosciutto --> 19.95
bread_bottom --> 19.95
bread_top_ham --> 19.95
bread_top_prosciutto --> 19.95


>> I have a piece of bread
   0.450 agent.goal.chunk sandwich cheese
   0.450 agent.production cheese
   0.500 agent.production None
>> I just put cheese on the bread
```

```
   0.500 agent.goal.chunk sandwich ham-or-prosciutto
   0.500 agent.production prosciutto
   0.550 agent.production None
>> I just put prosciutto on the cheese
   0.550 agent.goal.chunk sandwich bread_top prosciutto
   0.550 agent.production bread_top_prosciutto
   0.600 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.600 agent.goal.chunk sandwich bread
   0.600 agent.production bread_bottom
   0.650 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.95
ham --> 19.95
prosciutto --> 19.95
bread_bottom --> 19.95
bread_top_ham --> 19.95
bread_top_prosciutto --> 19.95


>> I have a piece of bread
   0.650 agent.goal.chunk sandwich cheese
   0.650 agent.production cheese
   0.700 agent.production None
>> I just put cheese on the bread
   0.700 agent.goal.chunk sandwich ham-or-prosciutto
   0.700 agent.production ham
   0.750 agent.production None
>> I just put ham on the cheese
   0.750 agent.goal.chunk sandwich bread_top ham
   0.750 agent.production bread_top_ham
   0.800 agent.production None
>> I just put bread on the ham

>> I made a    HAM    and cheese sandwich!

   0.800 agent.goal.chunk sandwich bread
   0.800 agent.production bread_bottom
   0.850 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<
```

```
Current production utilities:
-----------------------------
cheese --> 19.95
ham --> 19.95
prosciutto --> 19.95
bread_bottom --> 19.95
bread_top_ham --> 19.95
bread_top_prosciutto --> 19.95


>> I have a piece of bread
   0.850 agent.goal.chunk sandwich cheese
   0.850 agent.production cheese
   0.900 agent.production None
>> I just put cheese on the bread
   0.900 agent.goal.chunk sandwich ham-or-prosciutto
   0.900 agent.production prosciutto
   0.950 agent.production None
>> I just put prosciutto on the cheese
   0.950 agent.goal.chunk sandwich bread_top prosciutto
   0.950 agent.production bread_top_prosciutto
   1.000 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   1.000 agent.goal.chunk sandwich bread
   1.000 agent.production bread_bottom
>>> ccm.finished()
```

7.2 Parametrizing the production system

There are a variety of modules that can affect how the production system works. The primary effect of these modules is to adjust the utility values within the procedural memory system. All of the parameters shown below are optional (with default values given), and can be adjusted while the model is running. They are all optional and can even be combined together.

(15)
```
class SimpleModel(ACTR):
    # production noise
    pm_noise = PMNoise(noise=0,baseNoise=0)

    # standard PG-C
    pm_pgc = PMPGC(goal=20)

    # success-weighted PG-C (see Gray, Schoelles, & Sims, 2005)
    pm_pgcs = PMPGCSuccessWeighted(goal=20)

    # mixed-weighted PG-C (see Gray, Schoelles, & Sims, 2005)
    pm_pgcm = PMPGCMixedWeighted(goal=20)
```

```
# Temporal Difference Learning (see Fu & Anderson, 2004)
pm_td = PMTD(alpha=0.1,discount=1,cost=0.05)

# The new TD-inspired utility learning system
pm_new = PMNew(alpha=0.2)
```

Most of these systems require some sort of indication of a 'success' or 'failure' for a given production. This is done within a production using one of the following commands in its (action) body:[5]

(16)
```
self.success()
self.fail()
self.reward(0.8) # or any other numerical reward value,
                 #  positive being good and negative being bad)
```

The `success` command is actually defined as `reward(1)`, and `failure` is defined as `reward(-1)`.

7.3 Setting the utility of a production

By default, all productions start with a utility of 0. If you want to set this utility manually, rather than using a learning rule, you can do it like this:

(17)
```
def attendProbe(goal='state:start', vision='busy:False', location='?x ?y', utility=0.3):
```

7.4 Utility Learning Rules

The following production utility learning rules are available in Python ACT-R:

(18) PMPGC: the old PG-C learning rule:

```
pm=PMPGC(goal=20)
```

(19) PMPGCSuccessWeighted: the success-weighted PG-C rule:

```
pm=PMPGCSuccessWeighted(goal=20)
```

(20) PMPGCMixedWeighted: the mixed-weighted PG-C rule:

```
pm=PMPGCMixedWeighted(goal=20)
```

(21) PMQLearn: standard Q-learning:

```
pm=PMQLearn(alpha=0.2,gamma=0.9,initial=0)
```

(22) PMTD: TD-Learning (Fu & Anderson, 2004):

```
pm=PMQLearn(alpha=0.1,discount=1,cost=0.05)
```

(23) PMNew: the new ACT-R 6 standard learning rule:

```
pm=PMNew(alpha=0.2)
```

[5] There is also a mechanism supporting production compilation. This requires the precise specification as to what commands to compile over.

Furthermore, one unique feature of Python ACT-R is the ability to have multiple production systems. This allows us to quickly create new brain modules using the familiar syntax used to create the core production system.

7.5 Simple utility learning in more detail

Consider the code for the `PMNoise()` and `PMPGC()` classes in more detail:

(24) Module **ccm.lib.actr.pm**:

```
class PMNoise(ProceduralSubModule):
    def __init__(self,noise=0,baseNoise=0.0):
        self.noise=noise
        self.baseNoise=baseNoise
    def create(self,prod,parents=None):
        prod.baseNoise=self.logisticNoise(self.baseNoise)
    def utility(self,prod):
        return prod.baseNoise+self.logisticNoise(self.noise)
    def logisticNoise(self,s):
        x=self.random.random()
        return s*math.log(1.0/x -1.0)
```

(25) Module **ccm.lib.actr.pm** (ctd.):

```
class PMPGC(ProceduralSubModule):
    def __init__(self,goal=20):
        self.history=[]
        self.goal=goal
        self._clearFlag=False
    def create(self,prod,parents=None):
        prod.successes=1
        prod.failures=0
        prod.time=self.parent.production_time
        if callable(prod.time): prod.time=prod.time()
        prod.lock_pgc=False
    def selecting(self,prod):
        if self._clearFlag:
        del self.history[:]
        self._clearFlag=False
        self.history.append((prod,self.now()))
    def reward(self,value):
        now=self.now()
        for p,t in self.history:
            if not p.lock_pgc:
            dt=now-t
            p.time+=dt
            if value>=0: p.successes+=value
            else: p.failures-=value
        self._clearFlag=True
    def utility(self,prod):
        p=prod.successes/(prod.successes+prod.failures)
        c=prod.time/(prod.successes+prod.failures)
        g=self.goal
        return p*g-c
    def set(self,prod,successes=None,failures=None,time=None,lock=None):
        if not isinstance(prod,Production):
            prod=self.parent._productions[prod]
        if successes is not None: prod.successes=successes
        if failures is not None: prod.failures=failures
        if time is not None: prod.time=time
        if lock is not None: prod.lock_pgc=lock
```

Let us now explore various values for the subsymbolic parameters.

7.5.1 Base noise

(26) File **example_8_simple_utility_learning_base_noise.py**:

```
import ccm
from ccm.lib.actr import *


class MyAgent(ACTR):

    goal = Buffer()

    # production parameter settings
    production_time = 0.05
    production_sd = 0.01
    production_threshold = -20

    # production noise
    pm_noise = PMNoise(noise=0, baseNoise=0.5)

    # standard PG-C
    pm_pgc = PMPGC()

    def init():
        goal.set('sandwich bread')

    def bread_bottom(goal='sandwich bread'):
        print "\n\n>>>>>>>> Starting a new sandwich <<<<<<<<\n"
        self.print_production_utilities()
        print ">> I have a piece of bread"
        goal.set('sandwich cheese')

    def cheese(goal='sandwich cheese'):
        print ">> I just put cheese on the bread"
        goal.set('sandwich ham-or-prosciutto')

    def ham(goal='sandwich ham-or-prosciutto'):
        print ">> I just put ham on the cheese"
        goal.set('sandwich bread_top ham')

    # this production competes with the ham production
    def prosciutto(goal='sandwich ham-or-prosciutto'):
        print ">> I just put prosciutto on the cheese"
        goal.set('sandwich bread_top prosciutto')
    def bread_top_ham(goal='sandwich bread_top ham'):
        print ">> I just put bread on the ham"
        print "\n>> I made a    HAM    and cheese sandwich!\n"
        # make another sandwich
        goal.set('sandwich bread')

    def bread_top_prosciutto(goal='sandwich bread_top prosciutto'):
        print ">> I just put bread on the prosciutto"
        print "\n>> I made a    PROSCIUTTO    and cheese sandwich!\n"
        # make another sandwich
        goal.set('sandwich bread')

    def print_production_utilities(self):
        utilities_dict = self.get_activation()
        print "Current production utilities:"
        print "------------------------------"
        for key, value in utilities_dict.items():
            print key, "-->", round(value, 3)
        print "\n"


class MyEnvironment(ccm.Model):
    pass
```

[**py14**]
```
>>> from example_8_simple_utility_learning_base_noise import *

>>> print '\n>>>>>>>> Starting simulation 1 <<<<<<<<\n'

>>>>>>>> Starting simulation 1 <<<<<<<<

>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold -20
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_sd 0.01
   0.000 agent.production_time 0.05
   0.000 agent.pm_noise.baseNoise 0.5
   0.000 agent.pm_noise.noise 0
   0.000 agent.pm_pgc.goal 20
   0.000 agent.goal.chunk None
>>> #log = ccm.log(html=True)
>>> # run for 1.05 seconds
>>> empty_environment.run(1.05)
   0.000 agent.goal.chunk sandwich bread
   0.000 agent.production bread_bottom
   0.050 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 20.301
ham --> 19.621
prosciutto --> 20.133
bread_bottom --> 19.464
bread_top_ham --> 19.721
bread_top_prosciutto --> 20.339


>> I have a piece of bread
   0.050 agent.goal.chunk sandwich cheese
   0.050 agent.production cheese
   0.100 agent.production None
>> I just put cheese on the bread
   0.100 agent.goal.chunk sandwich ham-or-prosciutto
   0.100 agent.production prosciutto
   0.150 agent.production None
>> I just put prosciutto on the cheese
   0.150 agent.goal.chunk sandwich bread_top prosciutto
   0.150 agent.production bread_top_prosciutto
   0.200 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.200 agent.goal.chunk sandwich bread
   0.200 agent.production bread_bottom
   0.250 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 20.301
ham --> 19.621
prosciutto --> 20.133
bread_bottom --> 19.464
bread_top_ham --> 19.721
bread_top_prosciutto --> 20.339


>> I have a piece of bread
   0.250 agent.goal.chunk sandwich cheese
   0.250 agent.production cheese
   0.300 agent.production None
>> I just put cheese on the bread
   0.300 agent.goal.chunk sandwich ham-or-prosciutto
   0.300 agent.production prosciutto
   0.350 agent.production None
>> I just put prosciutto on the cheese
   0.350 agent.goal.chunk sandwich bread_top prosciutto
   0.350 agent.production bread_top_prosciutto
   0.400 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.400 agent.goal.chunk sandwich bread
   0.400 agent.production bread_bottom
   0.450 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 20.301
ham --> 19.621
prosciutto --> 20.133
bread_bottom --> 19.464
bread_top_ham --> 19.721
bread_top_prosciutto --> 20.339


>> I have a piece of bread
   0.450 agent.goal.chunk sandwich cheese
   0.450 agent.production cheese
   0.500 agent.production None
>> I just put cheese on the bread
   0.500 agent.goal.chunk sandwich ham-or-prosciutto
   0.500 agent.production prosciutto
   0.550 agent.production None
>> I just put prosciutto on the cheese
   0.550 agent.goal.chunk sandwich bread_top prosciutto
   0.550 agent.production bread_top_prosciutto
   0.600 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.600 agent.goal.chunk sandwich bread
   0.600 agent.production bread_bottom
   0.650 agent.production None
```

```
>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 20.301
ham --> 19.621
prosciutto --> 20.133
bread_bottom --> 19.464
bread_top_ham --> 19.721
bread_top_prosciutto --> 20.339


>> I have a piece of bread
   0.650 agent.goal.chunk sandwich cheese
   0.650 agent.production cheese
   0.700 agent.production None
>> I just put cheese on the bread
   0.700 agent.goal.chunk sandwich ham-or-prosciutto
   0.700 agent.production prosciutto
   0.750 agent.production None
>> I just put prosciutto on the cheese
   0.750 agent.goal.chunk sandwich bread_top prosciutto
   0.750 agent.production bread_top_prosciutto
   0.800 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.800 agent.goal.chunk sandwich bread
   0.800 agent.production bread_bottom
   0.850 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 20.301
ham --> 19.621
prosciutto --> 20.133
bread_bottom --> 19.464
bread_top_ham --> 19.721
bread_top_prosciutto --> 20.339


>> I have a piece of bread
   0.850 agent.goal.chunk sandwich cheese
   0.850 agent.production cheese
   0.900 agent.production None
>> I just put cheese on the bread
   0.900 agent.goal.chunk sandwich ham-or-prosciutto
   0.900 agent.production prosciutto
   0.950 agent.production None
>> I just put prosciutto on the cheese
   0.950 agent.goal.chunk sandwich bread_top prosciutto
   0.950 agent.production bread_top_prosciutto
   1.000 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   1.000 agent.goal.chunk sandwich bread
   1.000 agent.production bread_bottom
>>> ccm.finished()

>>> print '\n>>>>>>>> Starting simulation 2 <<<<<<<<\n'

>>>>>>>> Starting simulation 2 <<<<<<<<

>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold -20
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_sd 0.01
   0.000 agent.production_time 0.05
   0.000 agent.pm_noise.baseNoise 0.5
   0.000 agent.pm_noise.noise 0
   0.000 agent.pm_pgc.goal 20
   0.000 agent.goal.chunk None
>>> #log = ccm.log(html=True)
>>> # run for 1.05 seconds
>>> empty_environment.run(1.05)
   0.000 agent.goal.chunk sandwich bread
   0.000 agent.production bread_bottom
   0.050 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.176
ham --> 19.626
prosciutto --> 21.335
bread_bottom --> 21.867
```

```
bread_top_ham --> 20.379
bread_top_prosciutto --> 19.743


>> I have a piece of bread
   0.050 agent.goal.chunk sandwich cheese
   0.050 agent.production cheese
   0.100 agent.production None
>> I just put cheese on the bread
   0.100 agent.goal.chunk sandwich ham-or-prosciutto
   0.100 agent.production prosciutto
   0.150 agent.production None
>> I just put prosciutto on the cheese
   0.150 agent.goal.chunk sandwich bread_top prosciutto
   0.150 agent.production bread_top_prosciutto
   0.200 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.200 agent.goal.chunk sandwich bread
   0.200 agent.production bread_bottom
   0.250 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.176
ham --> 19.626
prosciutto --> 21.335
bread_bottom --> 21.867
bread_top_ham --> 20.379
bread_top_prosciutto --> 19.743


>> I have a piece of bread
   0.250 agent.goal.chunk sandwich cheese
   0.250 agent.production cheese
   0.300 agent.production None
>> I just put cheese on the bread
   0.300 agent.goal.chunk sandwich ham-or-prosciutto
   0.300 agent.production prosciutto
   0.350 agent.production None
>> I just put prosciutto on the cheese
   0.350 agent.goal.chunk sandwich bread_top prosciutto
   0.350 agent.production bread_top_prosciutto
   0.400 agent.production None
>> I just put bread on the prosciutto
>> I made a    PROSCIUTTO    and cheese sandwich!

   0.400 agent.goal.chunk sandwich bread
   0.400 agent.production bread_bottom
   0.450 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.176
ham --> 19.626
prosciutto --> 21.335
bread_bottom --> 21.867
bread_top_ham --> 20.379
bread_top_prosciutto --> 19.743


>> I have a piece of bread
   0.450 agent.goal.chunk sandwich cheese
   0.450 agent.production cheese
   0.500 agent.production None
>> I just put cheese on the bread
   0.500 agent.goal.chunk sandwich ham-or-prosciutto
   0.500 agent.production prosciutto
   0.550 agent.production None
>> I just put prosciutto on the cheese
   0.550 agent.goal.chunk sandwich bread_top prosciutto
   0.550 agent.production bread_top_prosciutto
   0.600 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.600 agent.goal.chunk sandwich bread
   0.600 agent.production bread_bottom
   0.650 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.176
ham --> 19.626
prosciutto --> 21.335
bread_bottom --> 21.867
bread_top_ham --> 20.379
```

```
bread_top_prosciutto --> 19.743


>> I have a piece of bread
   0.650 agent.goal.chunk sandwich cheese
   0.650 agent.production cheese
   0.700 agent.production None
>> I just put cheese on the bread
   0.700 agent.goal.chunk sandwich ham-or-prosciutto
   0.700 agent.production prosciutto
   0.750 agent.production None
>> I just put prosciutto on the cheese
   0.750 agent.goal.chunk sandwich bread_top prosciutto
   0.750 agent.production bread_top_prosciutto
   0.800 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.800 agent.goal.chunk sandwich bread
   0.800 agent.production bread_bottom
   0.850 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.176
ham --> 19.626
prosciutto --> 21.335
bread_bottom --> 21.867
bread_top_ham --> 20.379
bread_top_prosciutto --> 19.743


>> I have a piece of bread
   0.850 agent.goal.chunk sandwich cheese
   0.850 agent.production cheese
   0.900 agent.production None
>> I just put cheese on the bread
   0.900 agent.goal.chunk sandwich ham-or-prosciutto
   0.900 agent.production prosciutto
   0.950 agent.production None
>> I just put prosciutto on the cheese
   0.950 agent.goal.chunk sandwich bread_top prosciutto
   0.950 agent.production bread_top_prosciutto
   1.000 agent.production None
>> I just put bread on the prosciutto
>> I made a    PROSCIUTTO    and cheese sandwich!

   1.000 agent.goal.chunk sandwich bread
   1.000 agent.production bread_bottom
>>> ccm.finished()
```

7.5.2 Noise

(27) File **example_8_simple_utility_learning_noise.py**:

```
import ccm
from ccm.lib.actr import *


class MyAgent(ACTR):

    goal = Buffer()

    # production parameter settings
    production_time = 0.05
    production_sd = 0.01
    production_threshold = -20

    # production noise
    pm_noise = PMNoise(noise=0.5, baseNoise=0)

    # standard PG-C
    pm_pgc = PMPGC()

    def init():
        goal.set('sandwich bread')

    def bread_bottom(goal='sandwich bread'):
        print "\n\n>>>>>>>> Starting a new sandwich <<<<<<<<\n"
        self.print_production_utilities()
        print ">> I have a piece of bread"
        goal.set('sandwich cheese')

    def cheese(goal='sandwich cheese'):
        print ">> I just put cheese on the bread"
        goal.set('sandwich ham-or-prosciutto')

    def ham(goal='sandwich ham-or-prosciutto'):
        print ">> I just put ham on the cheese"
        goal.set('sandwich bread_top ham')

    # this production competes with the ham production
    def prosciutto(goal='sandwich ham-or-prosciutto'):
        print ">> I just put prosciutto on the cheese"
        goal.set('sandwich bread_top prosciutto')
    def bread_top_ham(goal='sandwich bread_top ham'):
        print ">> I just put bread on the ham"
        print "\n>> I made a    HAM    and cheese sandwich!\n"
        # make another sandwich
        goal.set('sandwich bread')

    def bread_top_prosciutto(goal='sandwich bread_top prosciutto'):
        print ">> I just put bread on the prosciutto"
        print "\n>> I made a    PROSCIUTTO    and cheese sandwich!\n"
        # make another sandwich
        goal.set('sandwich bread')

    def print_production_utilities(self):
        utilities_dict = self.get_activation()
        print "Current production utilities:"
        print "------------------------------"
        for key, value in utilities_dict.items():
            print key, "-->", round(value, 3)
        print "\n"


class MyEnvironment(ccm.Model):
    pass
```

[py15]
```
>>> from example_8_simple_utility_learning_noise import *

>>> print '\n>>>>>>>> Starting simulation 1 <<<<<<<<\n'

>>>>>>>> Starting simulation 1 <<<<<<<<

>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold -20
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_sd 0.01
   0.000 agent.production_time 0.05
   0.000 agent.pm_noise.baseNoise 0
   0.000 agent.pm_noise.noise 0.5
   0.000 agent.pm_pgc.goal 20
   0.000 agent.goal.chunk None
>>> #log = ccm.log(html=True)
>>> # run for 1.05 seconds
>>> empty_environment.run(1.05)
   0.000 agent.goal.chunk sandwich bread
   0.000 agent.production bread_bottom
   0.050 agent.production None
```

```
>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 17.771
ham --> 21.072
prosciutto --> 19.442
bread_bottom --> 19.606
bread_top_ham --> 20.372
bread_top_prosciutto --> 21.177


>> I have a piece of bread
   0.050 agent.goal.chunk sandwich cheese
   0.050 agent.production cheese
   0.100 agent.production None
>> I just put cheese on the bread
   0.100 agent.goal.chunk sandwich ham-or-prosciutto
   0.100 agent.production prosciutto
   0.150 agent.production None
>> I just put prosciutto on the cheese
   0.150 agent.goal.chunk sandwich bread_top prosciutto
   0.150 agent.production bread_top_prosciutto
   0.200 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.200 agent.goal.chunk sandwich bread
   0.200 agent.production bread_bottom
   0.250 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.417
ham --> 20.492
prosciutto --> 21.276
bread_bottom --> 19.311
bread_top_ham --> 20.677
bread_top_prosciutto --> 20.971


>> I have a piece of bread
   0.250 agent.goal.chunk sandwich cheese
   0.250 agent.production cheese
   0.300 agent.production None
>> I just put cheese on the bread
   0.300 agent.goal.chunk sandwich ham-or-prosciutto
   0.300 agent.production prosciutto
   0.350 agent.production None
>> I just put prosciutto on the cheese
   0.350 agent.goal.chunk sandwich bread_top prosciutto
   0.350 agent.production bread_top_prosciutto
   0.400 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.400 agent.goal.chunk sandwich bread
   0.400 agent.production bread_bottom
   0.450 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 20.005
ham --> 19.259
prosciutto --> 20.259
bread_bottom --> 19.741
bread_top_ham --> 19.479
bread_top_prosciutto --> 20.21


>> I have a piece of bread
   0.450 agent.goal.chunk sandwich cheese
   0.450 agent.production cheese
   0.500 agent.production None
>> I just put cheese on the bread
   0.500 agent.goal.chunk sandwich ham-or-prosciutto
   0.500 agent.production ham
   0.550 agent.production None
>> I just put ham on the cheese
   0.550 agent.goal.chunk sandwich bread_top ham
   0.550 agent.production bread_top_ham
   0.600 agent.production None
>> I just put bread on the ham

>> I made a    HAM    and cheese sandwich!

   0.600 agent.goal.chunk sandwich bread
   0.600 agent.production bread_bottom
   0.650 agent.production None
```

```
>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 20.001
ham --> 19.605
prosciutto --> 19.741
bread_bottom --> 20.457
bread_top_ham --> 19.339
bread_top_prosciutto --> 19.493


>> I have a piece of bread
   0.650 agent.goal.chunk sandwich cheese
   0.650 agent.production cheese
   0.700 agent.production None
>> I just put cheese on the bread
   0.700 agent.goal.chunk sandwich ham-or-prosciutto
   0.700 agent.production prosciutto
   0.750 agent.production None
>> I just put prosciutto on the cheese
   0.750 agent.goal.chunk sandwich bread_top prosciutto
   0.750 agent.production bread_top_prosciutto
   0.800 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.800 agent.goal.chunk sandwich bread
   0.800 agent.production bread_bottom
   0.850 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 18.645
ham --> 20.161
prosciutto --> 20.547
bread_bottom --> 19.686
bread_top_ham --> 18.542
bread_top_prosciutto --> 20.05


>> I have a piece of bread
   0.850 agent.goal.chunk sandwich cheese
   0.850 agent.production cheese
   0.900 agent.production None
```

```
>> I just put cheese on the bread
   0.900 agent.goal.chunk sandwich ham-or-prosciutto
   0.900 agent.production ham
   0.950 agent.production None
>> I just put ham on the cheese
   0.950 agent.goal.chunk sandwich bread_top ham
   0.950 agent.production bread_top_ham
   1.000 agent.production None
>> I just put bread on the ham

>> I made a    HAM    and cheese sandwich!

   1.000 agent.goal.chunk sandwich bread
   1.000 agent.production bread_bottom
>>> ccm.finished()

>>> print '\n>>>>>>>> Starting simulation 2 <<<<<<<<\n'

>>>>>>>> Starting simulation 2 <<<<<<<<

>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold -20
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_sd 0.01
   0.000 agent.production_time 0.05
   0.000 agent.pm_noise.baseNoise 0
   0.000 agent.pm_noise.noise 0.5
   0.000 agent.pm_pgc.goal 20
   0.000 agent.goal.chunk None
>>> #log = ccm.log(html=True)
>>> # run for 1.05 seconds
>>> empty_environment.run(1.05)
   0.000 agent.goal.chunk sandwich bread
   0.000 agent.production bread_bottom
   0.050 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 21.658
ham --> 19.552
prosciutto --> 22.706
bread_bottom --> 19.365
bread_top_ham --> 20.117
```

```
bread_top_prosciutto --> 19.404


>> I have a piece of bread
   0.050 agent.goal.chunk sandwich cheese
   0.050 agent.production cheese
   0.100 agent.production None
>> I just put cheese on the bread
   0.100 agent.goal.chunk sandwich ham-or-prosciutto
   0.100 agent.production prosciutto
   0.150 agent.production None
>> I just put prosciutto on the cheese
   0.150 agent.goal.chunk sandwich bread_top prosciutto
   0.150 agent.production bread_top_prosciutto
   0.200 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.200 agent.goal.chunk sandwich bread
   0.200 agent.production bread_bottom
   0.250 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 20.579
ham --> 20.042
prosciutto --> 20.014
bread_bottom --> 21.385
bread_top_ham --> 19.604
bread_top_prosciutto --> 19.793


>> I have a piece of bread
   0.250 agent.goal.chunk sandwich cheese
   0.250 agent.production cheese
   0.300 agent.production None
>> I just put cheese on the bread
   0.300 agent.goal.chunk sandwich ham-or-prosciutto
   0.300 agent.production ham
   0.350 agent.production None
>> I just put ham on the cheese
   0.350 agent.goal.chunk sandwich bread_top ham
   0.350 agent.production bread_top_ham
   0.400 agent.production None
>> I just put bread on the ham
>> I made a    HAM    and cheese sandwich!

   0.400 agent.goal.chunk sandwich bread
   0.400 agent.production bread_bottom
   0.450 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 21.399
ham --> 20.944
prosciutto --> 19.61
bread_bottom --> 19.122
bread_top_ham --> 19.033
bread_top_prosciutto --> 19.774


>> I have a piece of bread
   0.450 agent.goal.chunk sandwich cheese
   0.450 agent.production cheese
   0.500 agent.production None
>> I just put cheese on the bread
   0.500 agent.goal.chunk sandwich ham-or-prosciutto
   0.500 agent.production ham
   0.550 agent.production None
>> I just put ham on the cheese
   0.550 agent.goal.chunk sandwich bread_top ham
   0.550 agent.production bread_top_ham
   0.600 agent.production None
>> I just put bread on the ham

>> I made a    HAM    and cheese sandwich!

   0.600 agent.goal.chunk sandwich bread
   0.600 agent.production bread_bottom
   0.650 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 20.967
ham --> 21.235
prosciutto --> 20.918
bread_bottom --> 19.746
bread_top_ham --> 20.43
bread_top_prosciutto --> 16.957
```

```
>> I have a piece of bread
   0.650 agent.goal.chunk sandwich cheese
   0.650 agent.production cheese
   0.700 agent.production None
>> I just put cheese on the bread
   0.700 agent.goal.chunk sandwich ham-or-prosciutto
   0.700 agent.production prosciutto
   0.750 agent.production None
>> I just put prosciutto on the cheese
   0.750 agent.goal.chunk sandwich bread_top prosciutto
   0.750 agent.production bread_top_prosciutto
   0.800 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.800 agent.goal.chunk sandwich bread
   0.800 agent.production bread_bottom
   0.850 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 20.952
ham --> 19.989
prosciutto --> 19.421
bread_bottom --> 20.817
bread_top_ham --> 20.867
bread_top_prosciutto --> 18.784


>> I have a piece of bread
   0.850 agent.goal.chunk sandwich cheese
   0.850 agent.production cheese
   0.900 agent.production None
>> I just put cheese on the bread
   0.900 agent.goal.chunk sandwich ham-or-prosciutto
   0.900 agent.production prosciutto
   0.950 agent.production None
>> I just put prosciutto on the cheese
   0.950 agent.goal.chunk sandwich bread_top prosciutto
   0.950 agent.production bread_top_prosciutto
   1.000 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!
```

```
  1.000 agent.goal.chunk sandwich bread
  1.000 agent.production bread_bottom
>>> ccm.finished()
```

7.5.3 Both base noise and noise

(28) File **example_8_simple_utility_learning_base_noise_and_noise.py**:

```
import ccm
from ccm.lib.actr import *


class MyAgent(ACTR):

    goal = Buffer()

    # production parameter settings
    production_time = 0.05
    production_sd = 0.01
    production_threshold = -20

    # production noise
    pm_noise = PMNoise(noise=0.5, baseNoise=0.5)

    # standard PG-C
    pm_pgc = PMPGC()

    def init():
        goal.set('sandwich bread')

    def bread_bottom(goal='sandwich bread'):
        print "\n\n>>>>>>>> Starting a new sandwich <<<<<<<<\n"
        self.print_production_utilities()
        print ">> I have a piece of bread"
        goal.set('sandwich cheese')

    def cheese(goal='sandwich cheese'):
        print ">> I just put cheese on the bread"
        goal.set('sandwich ham-or-prosciutto')

    def ham(goal='sandwich ham-or-prosciutto'):
        print ">> I just put ham on the cheese"
        goal.set('sandwich bread_top ham')

    # this production competes with the ham production
    def prosciutto(goal='sandwich ham-or-prosciutto'):
        print ">> I just put prosciutto on the cheese"
        goal.set('sandwich bread_top prosciutto')

    def bread_top_ham(goal='sandwich bread_top ham'):
        print ">> I just put bread on the ham"
        print "\n>> I made a    HAM    and cheese sandwich!\n"
        # make another sandwich
        goal.set('sandwich bread')

    def bread_top_prosciutto(goal='sandwich bread_top prosciutto'):
        print ">> I just put bread on the prosciutto"
        print "\n>> I made a    PROSCIUTTO    and cheese sandwich!\n"
        # make another sandwich
        goal.set('sandwich bread')

    def print_production_utilities(self):
        utilities_dict = self.get_activation()
        print "Current production utilities:"
        print "-----------------------------"
        for key, value in utilities_dict.items():
            print key, "-->", round(value, 3)
        print "\n"


class MyEnvironment(ccm.Model):
    pass
```

[**py16**]
```
>>> from example_8_simple_utility_learning_base_noise_and_noise import *

>>> print '\n>>>>>>>> Starting simulation 1 <<<<<<<<\n'

>>>>>>>> Starting simulation 1 <<<<<<<<

>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold -20
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_sd 0.01
   0.000 agent.production_time 0.05
   0.000 agent.pm_noise.baseNoise 0.5
   0.000 agent.pm_noise.noise 0.5
   0.000 agent.pm_pgc.goal 20
   0.000 agent.goal.chunk None
>>> #log = ccm.log(html=True)
>>> # run for 1.05 seconds
>>> empty_environment.run(1.05)
   0.000 agent.goal.chunk sandwich bread
   0.000 agent.production bread_bottom
   0.050 agent.production None
```

```
>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.166
ham --> 19.693
prosciutto --> 18.699
bread_bottom --> 19.32
bread_top_ham --> 19.231
bread_top_prosciutto --> 21.933


>> I have a piece of bread
   0.050 agent.goal.chunk sandwich cheese
   0.050 agent.production cheese
   0.100 agent.production None
>> I just put cheese on the bread
   0.100 agent.goal.chunk sandwich ham-or-prosciutto
   0.100 agent.production ham
   0.150 agent.production None
>> I just put ham on the cheese
   0.150 agent.goal.chunk sandwich bread_top ham
   0.150 agent.production bread_top_ham
   0.200 agent.production None
>> I just put bread on the ham

>> I made a    HAM    and cheese sandwich!

   0.200 agent.goal.chunk sandwich bread
   0.200 agent.production bread_bottom
   0.250 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 20.834
ham --> 19.037
prosciutto --> 20.121
bread_bottom --> 20.68
bread_top_ham --> 18.993
bread_top_prosciutto --> 21.369


>> I have a piece of bread
   0.250 agent.goal.chunk sandwich cheese
   0.250 agent.production cheese
   0.300 agent.production None
```

```
>> I just put cheese on the bread
   0.300 agent.goal.chunk sandwich ham-or-prosciutto
   0.300 agent.production ham
   0.350 agent.production None
>> I just put ham on the cheese
   0.350 agent.goal.chunk sandwich bread_top ham
   0.350 agent.production bread_top_ham
   0.400 agent.production None
>> I just put bread on the ham

>> I made a    HAM    and cheese sandwich!

   0.400 agent.goal.chunk sandwich bread
   0.400 agent.production bread_bottom
   0.450 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 18.733
ham --> 19.957
prosciutto --> 19.31
bread_bottom --> 19.951
bread_top_ham --> 18.388
bread_top_prosciutto --> 21.063


>> I have a piece of bread
   0.450 agent.goal.chunk sandwich cheese
   0.450 agent.production cheese
   0.500 agent.production None
>> I just put cheese on the bread
   0.500 agent.goal.chunk sandwich ham-or-prosciutto
   0.500 agent.production ham
   0.550 agent.production None
>> I just put ham on the cheese
   0.550 agent.goal.chunk sandwich bread_top ham
   0.550 agent.production bread_top_ham
   0.600 agent.production None
>> I just put bread on the ham

>> I made a    HAM    and cheese sandwich!

   0.600 agent.goal.chunk sandwich bread
   0.600 agent.production bread_bottom
   0.650 agent.production None
```

```
>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.316
ham --> 19.06
prosciutto --> 19.493
bread_bottom --> 19.356
bread_top_ham --> 19.477
bread_top_prosciutto --> 21.331


>> I have a piece of bread
   0.650 agent.goal.chunk sandwich cheese
   0.650 agent.production cheese
   0.700 agent.production None
>> I just put cheese on the bread
   0.700 agent.goal.chunk sandwich ham-or-prosciutto
   0.700 agent.production ham
   0.750 agent.production None
>> I just put ham on the cheese
   0.750 agent.goal.chunk sandwich bread_top ham
   0.750 agent.production bread_top_ham
   0.800 agent.production None
>> I just put bread on the ham

>> I made a    HAM    and cheese sandwich!

   0.800 agent.goal.chunk sandwich bread
   0.800 agent.production bread_bottom
   0.850 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 16.441
ham --> 20.69
prosciutto --> 18.919
bread_bottom --> 20.548
bread_top_ham --> 19.977
bread_top_prosciutto --> 21.477


>> I have a piece of bread
   0.850 agent.goal.chunk sandwich cheese
   0.850 agent.production cheese
   0.900 agent.production None
>> I just put cheese on the bread
```

```
   0.900 agent.goal.chunk sandwich ham-or-prosciutto
   0.900 agent.production ham
   0.950 agent.production None
>> I just put ham on the cheese
   0.950 agent.goal.chunk sandwich bread_top ham
   0.950 agent.production bread_top_ham
   1.000 agent.production None
>> I just put bread on the ham

>> I made a    HAM    and cheese sandwich!

   1.000 agent.goal.chunk sandwich bread
   1.000 agent.production bread_bottom
>>> ccm.finished()

>>> print '\n>>>>>>>> Starting simulation 2 <<<<<<<<\n'

>>>>>>>> Starting simulation 2 <<<<<<<<

>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold -20
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_sd 0.01
   0.000 agent.production_time 0.05
   0.000 agent.pm_noise.baseNoise 0.5
   0.000 agent.pm_noise.noise 0.5
   0.000 agent.pm_pgc.goal 20
   0.000 agent.goal.chunk None
>>> #log = ccm.log(html=True)
>>> # run for 1.05 seconds
>>> empty_environment.run(1.05)
   0.000 agent.goal.chunk sandwich bread
   0.000 agent.production bread_bottom
   0.050 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.716
ham --> 20.215
prosciutto --> 19.236
bread_bottom --> 18.644
bread_top_ham --> 22.048
bread_top_prosciutto --> 18.875
```

```
>> I have a piece of bread
   0.050 agent.goal.chunk sandwich cheese
   0.050 agent.production cheese
   0.100 agent.production None
>> I just put cheese on the bread
   0.100 agent.goal.chunk sandwich ham-or-prosciutto
   0.100 agent.production prosciutto
   0.150 agent.production None
>> I just put prosciutto on the cheese
   0.150 agent.goal.chunk sandwich bread_top prosciutto
   0.150 agent.production bread_top_prosciutto
   0.200 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.200 agent.goal.chunk sandwich bread
   0.200 agent.production bread_bottom
   0.250 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 18.247
ham --> 17.137
prosciutto --> 18.524
bread_bottom --> 19.076
bread_top_ham --> 20.718
bread_top_prosciutto --> 18.867


>> I have a piece of bread
   0.250 agent.goal.chunk sandwich cheese
   0.250 agent.production cheese
   0.300 agent.production None
>> I just put cheese on the bread
   0.300 agent.goal.chunk sandwich ham-or-prosciutto
   0.300 agent.production prosciutto
   0.350 agent.production None
>> I just put prosciutto on the cheese
   0.350 agent.goal.chunk sandwich bread_top prosciutto
   0.350 agent.production bread_top_prosciutto
   0.400 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!
```

```
   0.400 agent.goal.chunk sandwich bread
   0.400 agent.production bread_bottom
   0.450 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 20.036
ham --> 18.572
prosciutto --> 18.362
bread_bottom --> 18.161
bread_top_ham --> 23.098
bread_top_prosciutto --> 19.214


>> I have a piece of bread
   0.450 agent.goal.chunk sandwich cheese
   0.450 agent.production cheese
   0.500 agent.production None
>> I just put cheese on the bread
   0.500 agent.goal.chunk sandwich ham-or-prosciutto
   0.500 agent.production prosciutto
   0.550 agent.production None
>> I just put prosciutto on the cheese
   0.550 agent.goal.chunk sandwich bread_top prosciutto
   0.550 agent.production bread_top_prosciutto
   0.600 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.600 agent.goal.chunk sandwich bread
   0.600 agent.production bread_bottom
   0.650 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.899
ham --> 17.263
prosciutto --> 20.46
bread_bottom --> 19.185
bread_top_ham --> 20.74
bread_top_prosciutto --> 17.606
```

```
>> I have a piece of bread
   0.650 agent.goal.chunk sandwich cheese
   0.650 agent.production cheese
   0.700 agent.production None
>> I just put cheese on the bread
   0.700 agent.goal.chunk sandwich ham-or-prosciutto
   0.700 agent.production prosciutto
   0.750 agent.production None
>> I just put prosciutto on the cheese
   0.750 agent.goal.chunk sandwich bread_top prosciutto
   0.750 agent.production bread_top_prosciutto
   0.800 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.800 agent.goal.chunk sandwich bread
   0.800 agent.production bread_bottom
   0.850 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.189
ham --> 19.536
prosciutto --> 20.32
bread_bottom --> 20.867
bread_top_ham --> 20.94
bread_top_prosciutto --> 17.789


>> I have a piece of bread
   0.850 agent.goal.chunk sandwich cheese
   0.850 agent.production cheese
   0.900 agent.production None
>> I just put cheese on the bread
   0.900 agent.goal.chunk sandwich ham-or-prosciutto
   0.900 agent.production ham
   0.950 agent.production None
>> I just put ham on the cheese
   0.950 agent.goal.chunk sandwich bread_top ham
   0.950 agent.production bread_top_ham
   1.000 agent.production None
>> I just put bread on the ham

>> I made a    HAM    and cheese sandwich!
```

```
   1.000 agent.goal.chunk sandwich bread
   1.000 agent.production bread_bottom
>>> ccm.finished()
```

7.5.4 Rewards

We will now zero out the noise again and add rewards: negative for the production `ham()`, and positive for the production `prosciutto()`. We see that even if `ham()` is chosen the first or second time, the agent quickly learns to prefer prosciutto.

Note how the positive or negative rewards get 'percolated' back to the rules that were applied before the actual reward was given. The `cheese()` rule, for example, immediately precedes the `ham()` or `prosciutto()` productions, and its utility decreases after a `ham()` production and increases after a `prosciutto()` production.

(29) File **example_8_simple_utility_learning_reward.py**:

```
import ccm
from ccm.lib.actr import *


class MyAgent(ACTR):

    goal = Buffer()

    # production parameter settings
    production_time = 0.05
    production_sd = 0.01
    production_threshold = -20

    # production noise
    pm_noise = PMNoise(noise=0, baseNoise=0)

    # standard PG-C
    pm_pgc = PMPGC(goal=20)

    def init():
        goal.set('sandwich bread')

    def bread_bottom(goal='sandwich bread'):
        print "\n\n>>>>>>>> Starting a new sandwich <<<<<<<<\n"
        self.print_production_utilities()
        print ">> I have a piece of bread"
        goal.set('sandwich cheese')

    def cheese(goal='sandwich cheese'):
        print ">> I just put cheese on the bread"
        goal.set('sandwich ham-or-prosciutto')

    def ham(goal='sandwich ham-or-prosciutto'):
        print ">> I just put ham on the cheese"
        goal.set('sandwich bread_top ham')
```

```
            # using ham rather than prosciutto is intrinsically less rewarding
            self.reward(-0.1)

        # this production competes with the ham production
        def prosciutto(goal='sandwich ham-or-prosciutto'):
            print ">> I just put prosciutto on the cheese"
            goal.set('sandwich bread_top prosciutto')
            # using prosciutto rather than ham is intrinsically more rewarding
            self.reward(0.1)

        def bread_top_ham(goal='sandwich bread_top ham'):
            print ">> I just put bread on the ham"
            print "\n>> I made a    HAM    and cheese sandwich!\n"
            # make another sandwich
            goal.set('sandwich bread')

        def bread_top_prosciutto(goal='sandwich bread_top prosciutto'):
            print ">> I just put bread on the prosciutto"
            print "\n>> I made a    PROSCIUTTO    and cheese sandwich!\n"
            # make another sandwich
            goal.set('sandwich bread')

        def print_production_utilities(self):
            utilities_dict = self.get_activation()
            print "Current production utilities:"
            print "----------------------------"
            for key, value in utilities_dict.items():
                print key, "-->", round(value, 3)
            print "\n"


    class MyEnvironment(ccm.Model):
        pass
```

[py17]
```
>>> from example_8_simple_utility_learning_reward import *

>>> print '\n>>>>>>>> Starting simulation 1 <<<<<<<<\n'

>>>>>>>> Starting simulation 1 <<<<<<<<

>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold -20
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_sd 0.01
   0.000 agent.production_time 0.05
   0.000 agent.pm_noise.baseNoise 0
   0.000 agent.pm_noise.noise 0
   0.000 agent.pm_pgc.goal 20
   0.000 agent.goal.chunk None
>>> #log = ccm.log(html=True)
>>> # run for 1.05 seconds
>>> empty_environment.run(1.05)
   0.000 agent.goal.chunk sandwich bread
   0.000 agent.production bread_bottom
   0.050 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.95
ham --> 19.95
prosciutto --> 19.95
bread_bottom --> 19.95
bread_top_ham --> 19.95
bread_top_prosciutto --> 19.95


>> I have a piece of bread
   0.050 agent.goal.chunk sandwich cheese
   0.050 agent.production cheese
   0.100 agent.production None
>> I just put cheese on the bread
   0.100 agent.goal.chunk sandwich ham-or-prosciutto
   0.100 agent.production prosciutto
   0.150 agent.production None
>> I just put prosciutto on the cheese
   0.150 agent.goal.chunk sandwich bread_top prosciutto
   0.150 agent.production bread_top_prosciutto
   0.200 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.200 agent.goal.chunk sandwich bread
   0.200 agent.production bread_bottom
   0.250 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.864
```

```
ham --> 19.95
prosciutto --> 19.909
bread_bottom --> 19.818
bread_top_ham --> 19.95
bread_top_prosciutto --> 19.95


>> I have a piece of bread
   0.250 agent.goal.chunk sandwich cheese
   0.250 agent.production cheese
   0.300 agent.production None
>> I just put cheese on the bread
   0.300 agent.goal.chunk sandwich ham-or-prosciutto
   0.300 agent.production ham
   0.350 agent.production None
>> I just put ham on the cheese
   0.350 agent.goal.chunk sandwich bread_top ham
   0.350 agent.production bread_top_ham
   0.400 agent.production None
>> I just put bread on the ham

>> I made a    HAM    and cheese sandwich!

   0.400 agent.goal.chunk sandwich bread
   0.400 agent.production bread_bottom
   0.450 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 18.125
ham --> 18.091
prosciutto --> 19.909
bread_bottom --> 18.042
bread_top_ham --> 19.95
bread_top_prosciutto --> 17.955


>> I have a piece of bread
   0.450 agent.goal.chunk sandwich cheese
   0.450 agent.production cheese
   0.500 agent.production None
>> I just put cheese on the bread
   0.500 agent.goal.chunk sandwich ham-or-prosciutto
   0.500 agent.production prosciutto
   0.550 agent.production None
>> I just put prosciutto on the cheese
   0.550 agent.goal.chunk sandwich bread_top prosciutto
   0.550 agent.production bread_top_prosciutto
   0.600 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.600 agent.goal.chunk sandwich bread
   0.600 agent.production bread_bottom
   0.650 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 18.192
ham --> 18.091
prosciutto --> 19.875
bread_bottom --> 18.077
bread_top_ham --> 19.773
bread_top_prosciutto --> 17.955


>> I have a piece of bread
   0.650 agent.goal.chunk sandwich cheese
   0.650 agent.production cheese
   0.700 agent.production None
>> I just put cheese on the bread
   0.700 agent.goal.chunk sandwich ham-or-prosciutto
   0.700 agent.production prosciutto
   0.750 agent.production None
>> I just put prosciutto on the cheese
   0.750 agent.goal.chunk sandwich bread_top prosciutto
   0.750 agent.production bread_top_prosciutto
   0.800 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.800 agent.goal.chunk sandwich bread
   0.800 agent.production bread_bottom
   0.850 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 18.25
ham --> 18.091
```

```
prosciutto --> 19.846
bread_bottom --> 18.107
bread_top_ham --> 19.773
bread_top_prosciutto --> 17.958


>> I have a piece of bread
   0.850 agent.goal.chunk sandwich cheese
   0.850 agent.production cheese
   0.900 agent.production None
>> I just put cheese on the bread
   0.900 agent.goal.chunk sandwich ham-or-prosciutto
   0.900 agent.production prosciutto
   0.950 agent.production None
>> I just put prosciutto on the cheese
   0.950 agent.goal.chunk sandwich bread_top prosciutto
   0.950 agent.production bread_top_prosciutto
   1.000 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   1.000 agent.goal.chunk sandwich bread
   1.000 agent.production bread_bottom
>>> ccm.finished()

>>> print '\n>>>>>>>> Starting simulation 2 <<<<<<<<\n'

>>>>>>>> Starting simulation 2 <<<<<<<<

>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold -20
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_sd 0.01
   0.000 agent.production_time 0.05
   0.000 agent.pm_noise.baseNoise 0
   0.000 agent.pm_noise.noise 0
   0.000 agent.pm_pgc.goal 20
   0.000 agent.goal.chunk None
>>> #log = ccm.log(html=True)
>>> # run for 1.05 seconds
>>> empty_environment.run(1.05)
   0.000 agent.goal.chunk sandwich bread
   0.000 agent.production bread_bottom
   0.050 agent.production None
```

```
>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.95
ham --> 19.95
prosciutto --> 19.95
bread_bottom --> 19.95
bread_top_ham --> 19.95
bread_top_prosciutto --> 19.95


>> I have a piece of bread
   0.050 agent.goal.chunk sandwich cheese
   0.050 agent.production cheese
   0.100 agent.production None
>> I just put cheese on the bread
   0.100 agent.goal.chunk sandwich ham-or-prosciutto
   0.100 agent.production prosciutto
   0.150 agent.production None
>> I just put prosciutto on the cheese
   0.150 agent.goal.chunk sandwich bread_top prosciutto
   0.150 agent.production bread_top_prosciutto
   0.200 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.200 agent.goal.chunk sandwich bread
   0.200 agent.production bread_bottom
   0.250 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 19.864
ham --> 19.95
prosciutto --> 19.909
bread_bottom --> 19.818
bread_top_ham --> 19.95
bread_top_prosciutto --> 19.95


>> I have a piece of bread
   0.250 agent.goal.chunk sandwich cheese
   0.250 agent.production cheese
   0.300 agent.production None
```

```
>> I just put cheese on the bread
   0.300 agent.goal.chunk sandwich ham-or-prosciutto
   0.300 agent.production ham
   0.350 agent.production None
>> I just put ham on the cheese
   0.350 agent.goal.chunk sandwich bread_top ham
   0.350 agent.production bread_top_ham
   0.400 agent.production None
>> I just put bread on the ham

>> I made a    HAM    and cheese sandwich!

   0.400 agent.goal.chunk sandwich bread
   0.400 agent.production bread_bottom
   0.450 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 18.125
ham --> 18.091
prosciutto --> 19.909
bread_bottom --> 18.042
bread_top_ham --> 19.95
bread_top_prosciutto --> 17.955


>> I have a piece of bread
   0.450 agent.goal.chunk sandwich cheese
   0.450 agent.production cheese
   0.500 agent.production None
>> I just put cheese on the bread
   0.500 agent.goal.chunk sandwich ham-or-prosciutto
   0.500 agent.production prosciutto
   0.550 agent.production None
>> I just put prosciutto on the cheese
   0.550 agent.goal.chunk sandwich bread_top prosciutto
   0.550 agent.production bread_top_prosciutto
   0.600 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.600 agent.goal.chunk sandwich bread
   0.600 agent.production bread_bottom
   0.650 agent.production None
```

```
>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 18.192
ham --> 18.091
prosciutto --> 19.875
bread_bottom --> 18.077
bread_top_ham --> 19.773
bread_top_prosciutto --> 17.955


>> I have a piece of bread
   0.650 agent.goal.chunk sandwich cheese
   0.650 agent.production cheese
   0.700 agent.production None
>> I just put cheese on the bread
   0.700 agent.goal.chunk sandwich ham-or-prosciutto
   0.700 agent.production prosciutto
   0.750 agent.production None
>> I just put prosciutto on the cheese
   0.750 agent.goal.chunk sandwich bread_top prosciutto
   0.750 agent.production bread_top_prosciutto
   0.800 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   0.800 agent.goal.chunk sandwich bread
   0.800 agent.production bread_bottom
   0.850 agent.production None


>>>>>>>> Starting a new sandwich <<<<<<<<

Current production utilities:
-----------------------------
cheese --> 18.25
ham --> 18.091
prosciutto --> 19.846
bread_bottom --> 18.107
bread_top_ham --> 19.773
bread_top_prosciutto --> 17.958


>> I have a piece of bread
   0.850 agent.goal.chunk sandwich cheese
   0.850 agent.production cheese
   0.900 agent.production None
>> I just put cheese on the bread
```

```
   0.900 agent.goal.chunk sandwich ham-or-prosciutto
   0.900 agent.production prosciutto
   0.950 agent.production None
>> I just put prosciutto on the cheese
   0.950 agent.goal.chunk sandwich bread_top prosciutto
   0.950 agent.production bread_top_prosciutto
   1.000 agent.production None
>> I just put bread on the prosciutto

>> I made a    PROSCIUTTO    and cheese sandwich!

   1.000 agent.goal.chunk sandwich bread
   1.000 agent.production bread_bottom
>>> ccm.finished()
```

8 The subsymbolic level for the declarative memory module

Memories fade and some become inaccessible. The retrieval of a particular declarative memory chunk is dependent on the *activation level* of the chunk. When there are several chunks that match the requested pattern, the chunk with the highest activation is retrieved.

For more complex models, there are a variety of sub-modules which can adjust the activation of the chunks in memory, giving them different probabilities of being retrieved. If no sub-modules are used, then the activation of all chunks is always 0, and the default behavior of the declarative memory system is to randomly choose one of the chunks that match the request.

The main components of the subsymbolic memory activation system are the following:

- **the base level learning equation**: the activation of a chunk is increased when it is used, and this activation decays over time.
- **spreading activation**: a formula for increasing the activation of chunks that are similar to chunks already in buffers
- **partial matching**: a formula for adjusting the activation of chunks that almost match the memory request pattern
- **noise**: a random amount of noise that leads to variability in behavior

When creating our own ACT-R models that include a declarative memory module, the first step is always to create a retrieval buffer and a basic declarative memory system. This is what we did in Example 2 – and the relevant code is repeated below for convenience.

(30) Initializing the declarative memory system (repeated from file **example_2.py**:

```
# create a buffer for the declarative memory
DMBuffer = Buffer()
# create a declarative memory object called DM
DM = Memory(DMBuffer)
```

The memory system in Python ACT-R has a variety of parameters:

- `latency`: controls the relationship between activation and how long it takes to recall the chunk

- `threshold`: chunks must have activation greater than or equal to this to be recalled (can be set to `None`)
- `maximum_time`: no retrieval will take longer than this amount of time
- `finst_size`: the FINST system allows you to recall items that have not been recently recalled; this controls how many recent items are remembered
- `finst_time`: controls how recent recalls must be for them to be included in the FINST system.

What are finsts? To avoid the problem of repeatedly retrieving the most active chunk in situations where this is not appropriate, ACT-R has a system for maintaining *fingers of instantiation* (FINSTs). This allows the memory request to indicate that it should not recall a recently retrieved item. See Anderson and Lebiere (1998) for details.

(31) From the ACT-R 6.0 manual (by Dan Bothell, http://act-r.psy.cmu.edu/actr6/reference-manual.pdf):

> The declarative module maintains a record of the chunks which have been retrieved and provides a mechanism which allows one to explicitly retrieve or not retrieve one which is so marked. This is done through the use of a set of finsts (fingers of instantiation) which mark those chunks. The finsts are limited in the number that are available and how long they persist. The number and duration are both controlled by parameters. If more finsts are required than are available, then the oldest one (the one marking the chunk retrieved at the earliest time) is removed and used to mark the most recently retrieved chunk. (p. 226)

Here's an example of initializing the declarative memory system (module + buffer) in which we explicitly use these parameters:

(32) Initializing the declarative memory system and explicitly setting its parameters:

```
DMBuffer = Buffer()
DM = Memory(DMBuffer,\
            latency=0.05,\
            threshold=0,\
            maximum_time=10.0,\
            finst_size=0,\
            finst_time=3.0)
```

The following systems adjust the activation in various ways. They are shown along with the default values for their parameters.

(33) Adjusting activation:

```
# standard random noise
dm_n = DMNoise(DM, noise=0.3, baseNoise=0.0)

# the standard base level learning system
# if limit is set to a number, then the hybrid optimized equation
# from Petrov (2006) is used.
dm_bl = DMBaseLevel(DM, decay=0.5, limit=None)

# the spacing effect system from (Pavlik & Anderson 2005)
```

```
dm_space = DMSpacing(DM, decayScale=0.0, decayIntercept=0.5)

# the standard fan-effect spreading activation system
dm_spread = DMSpreading(DM, goal) # specify the buffer(s) to spread from

# other parameters are configured like this:
dm_spread.strength = 1
dm_spread.weight[goal] = 0.3
```

8.1 Example: Forgetting

We will now turn on the subsymbolic processing for DM, which causes forgetting.

In particular, we will turn on the base-level activation functions for DM. This means the contents of DM decay over time. If the threshold parameter is raised, it will cause the model to forget. So we need an extra production to handle an unsuccessful memory request.

(34) File **example_9_simple_forget.py**:

```
import ccm
from ccm.lib.actr import *


class MyEnvironment(ccm.Model):
    pass


class MyAgent(ACTR):

    goal = Buffer()

    DMBuffer = Buffer()
    # latency controls the relationship between activation and recall
    # activation must be above threshold - can be set to none
    DM = Memory(DMBuffer, latency=0.05, threshold=0)
    # turn on for DM subsymbolic processing
    dm_n = DMNoise(DM, noise=0.0, baseNoise=0.0)
    # turn on for DM subsymbolic processing
    dm_bl = DMBaseLevel(DM, decay=0.5, limit=None)

    def init():
        DM.add("cue:customer condiment:mustard")
        goal.set("sandwich bread")

    def bread_bottom(goal="sandwich bread"):
        print "I have a piece of bread"
        goal.set("sandwich cheese")

    def cheese(goal="sandwich cheese"):
        print "I put cheese on the bread"
        goal.set("sandwich ham")
    def ham(goal="sandwich ham"):
        print "I put ham on the cheese"
        goal.set("customer condiment")

    def condiment(goal="customer condiment"):
        print "Recalling the order"
        DM.request("cue:customer condiment:?condiment")
        goal.set("sandwich condiment")

    def order(goal="sandwich condiment", DMBuffer="cue:customer condiment:?condiment"):
        print "I recall they wanted......."
        print condiment
        print "I put the condiment on the sandwich"
        goal.set("sandwich bread_top")

    # DMBuffer = none means the buffer is empty
    # DM = "error:True" means the search was unsucessful
    def forgot(goal="sandwich condiment", DMBuffer=None, DM="error:True"):
        print "I recall they wanted......."
        print "I forgot"
        goal.set("stop")

    def bread_top(goal="sandwich bread_top"):
        print "I put bread on the ham"
        print "I made a ham and cheese sandwich"
        goal.set("stop")

    def stop_production(goal="stop"):
        self.stop()
```

[py18]
```
>>> from example_9_simple_forget import *
>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold None
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_time 0.05
   0.000 agent.DM.record_all_chunks False
   0.000 agent.DM.threshold 0
   0.000 agent.DM.latency 0.05
   0.000 agent.DM.busy False
   0.000 agent.DM.maximum_time 10.0
   0.000 agent.DM.error False
   0.000 agent.goal.chunk None
   0.000 agent.DMBuffer.chunk None
>>> empty_environment.run()
   0.000 agent.goal.chunk sandwich bread
```

```
      0.000 agent.production bread_bottom
      0.050 agent.production None
   I have a piece of bread
      0.050 agent.goal.chunk sandwich cheese
      0.050 agent.production cheese
      0.100 agent.production None
   I put cheese on the bread
      0.100 agent.goal.chunk sandwich ham
      0.100 agent.production ham
      0.150 agent.production None
   I put ham on the cheese
      0.150 agent.goal.chunk customer condiment
      0.150 agent.production condiment
      0.200 agent.production None
   Recalling the order
      0.200 agent.DM.busy True
      0.200 agent.goal.chunk sandwich condiment
      0.222 agent.DMBuffer.chunk condiment:mustard cue:customer
      0.222 agent.DM.busy False
      0.222 agent.production order
      0.272 agent.production None
   I recall they wanted.......
   mustard
   I put the condiment on the sandwich
      0.272 agent.goal.chunk sandwich bread_top
      0.272 agent.production bread_top
      0.322 agent.production None
   I put bread on the ham
   I made a ham and cheese sandwich
      0.322 agent.goal.chunk stop
      0.322 agent.production stop_production
      0.372 agent.production None
   >>> ccm.finished()
```

8.2 Example: Activation

We continue our exploration of subsymbolic processing for DM. In this model, a chunk is retrieved over and over. Each time a chunk is successfully retrieved in ACT-R, it is referred to as a 'harvest'. In ACT-R theory, when a chunk is harvested it receives a boost in activation. In Python ACT-R, this is not automatic (as it is in LISP ACT-R), so you need to add a `.add` function (method, in Python terms).

With this in place, the activation of the chunk gets stronger and stronger and the time to retrieve it decreases, along the lines of what we saw in the plots above.

(35) File **example_10_simple_activation.py**:

```
import ccm
from ccm.lib.actr import *


class MyEnvironment(ccm.Model):
    pass
```

```
class MyAgent(ACTR):

    goal = Buffer()

    DMBuffer = Buffer()
    # latency controls the relationship between activation and recall
    # activation must be above threshold; can be set to None
    DM = Memory(DMBuffer, latency=1.0, threshold=1)
    dm_n = DMNoise(DM, noise=0.0, baseNoise=0.0)
    dm_bl = DMBaseLevel(DM, decay=0.5, limit=None)

    def init():
        DM.add("customer:customer1 condiment:mustard")
        goal.set("rehearse")
        self.print_activations()

    def request_chunk(goal="rehearse"):
        print "recalling the order"
        DM.request("customer:customer1 condiment:?condiment")
        goal.set("recall")
        self.print_activations()

    def recall_chunk(goal="recall", DMBuffer="customer:customer1 condiment:?condiment"):
        print "Customer 1 wants......."
        print condiment
        # each time we put something in memory (DM.add), it increases activation
        DM.add("customer:customer1 ?condiment")
        DMBuffer.clear()
        goal.set("rehearse")
        self.print_activations()

    def forgot(goal="recall", DMBuffer=None, DM="error:True"):
        print "I recall they wanted......."
        print "I forgot"
        goal.set("stop")
        self.print_activations()

    def print_activations(self):
        list_of_chunks = self.DM.find_matching_chunks('')
        print "\nCurrent chunk activation values:"
        print "--------------------------------"
        for chunk in list_of_chunks:
            if chunk.has_key("condiment"):
                print chunk["customer"], chunk["condiment"], "-->", \
                      round(self.DM.get_activation(chunk), 3)
            else:
                print chunk["customer"], "-->", \
                      round(self.DM.get_activation(chunk), 3)
            print "\n"
```

We run the model just as before, except this time we run it for 2 seconds only.

[**py19**]
```
>>> from example_10_simple_activation import *
>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold None
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_time 0.05
   0.000 agent.DM.record_all_chunks False
   0.000 agent.DM.threshold 1
   0.000 agent.DM.latency 1.0
   0.000 agent.DM.busy False
   0.000 agent.DM.maximum_time 10.0
   0.000 agent.DM.error False
   0.000 agent.goal.chunk None
   0.000 agent.DMBuffer.chunk None
>>> #log = ccm.log(html=True) # we turn on html logging
>>> empty_environment.run(2)
   0.000 agent.goal.chunk rehearse

Current chunk activation values:
--------------------------------
customer1 mustard --> 2.649



   0.000 agent.production request_chunk
   0.050 agent.production None
recalling the order
   0.050 agent.DM.busy True
   0.050 agent.goal.chunk recall

Current chunk activation values:
--------------------------------
customer1 mustard --> 1.498



   0.274 agent.DMBuffer.chunk condiment:mustard customer:customer1
   0.274 agent.DM.busy False
   0.274 agent.production recall_chunk
   0.324 agent.production None
Customer 1 wants.......
mustard
   0.324 agent.DMBuffer.chunk None
   0.324 agent.goal.chunk rehearse
```

```
Current chunk activation values:
--------------------------------
customer1 mustard --> 0.564
customer1 --> 2.649


   0.324 agent.production request_chunk
   0.374 agent.production None
recalling the order
   0.374 agent.DM.busy True
   0.374 agent.goal.chunk recall

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.492
customer1 --> 1.498


   0.741 agent.DM.error True
   0.741 agent.DMBuffer.chunk None
   0.741 agent.DM.busy False
   0.741 agent.production forgot
   0.791 agent.production None
I recall they wanted.......
I forgot
   0.791 agent.goal.chunk stop

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.117
customer1 --> 0.38


>>> ccm.finished()
```

8.3 Example: Spreading Activation

This model turns on spreading activation. This means that DM chunks that have slot contents that match what's in the buffers will receive a boost in activation. Note that the threshold is turned up to 1, which would cause forgetting in the previous example.

(36) File **example_11_simple_forget.py**:

```
import ccm
from ccm.lib.actr import *


class MyEnvironment(ccm.Model):
    pass
```

```
class MyAgent(ACTR):

    goal = Buffer()

    DMBuffer = Buffer()
    DM = Memory(DMBuffer, latency=0.05, threshold=1)
    dm_n = DMNoise(DM, noise=0.0, baseNoise=0.0)
    dm_bl = DMBaseLevel(DM, decay=0.5, limit=None)
    # turn on spreading activation for DM from goal
    dm_spread = DMSpreading(DM, goal)
    # set strength of activation for buffers
    dm_spread.strength = 2
    # set weight to adjust for how many slots in the buffer
    # usually this is strength divided by number of slots
    dm_spread.weight[goal] = .5

    def init():
        DM.add("customer:customer condiment:mustard")
        goal.set("sandwich bread")
        self.print_activations()

    def bread_bottom(goal="sandwich bread"):
        print "I have a piece of bread"
        goal.set("sandwich cheese")
        self.print_activations()

    def cheese(goal="sandwich cheese"):
        print "I put cheese on the bread"
        goal.set("sandwich ham")
        self.print_activations()

    def ham(goal="sandwich ham"):
        print "I put ham on the cheese"
        goal.set("customer condiment")
        self.print_activations()

    # customer will spread activation to "customer mustard"
    def condiment(goal="customer condiment"):
        print "Recalling the order"
        # request gets boost from spreading activation
        DM.request("customer:customer condiment:?condiment")
        goal.set("sandwich condiment")
        self.print_activations()

    def order(goal="sandwich condiment", DMBuffer="customer:customer condiment:?condiment"):
        print "I recall they wanted......."
        print condiment
        print "I put the condiment on the sandwich"
        goal.set("sandwich bread_top")
        self.print_activations()
    def forgot(goal="sandwich condiment", DMBuffer=None, DM="error:True"):
        print "I recall they wanted......."
        print "I forgot"
        goal.set("stop")
        self.print_activations()

    def bread_top(goal="sandwich bread_top"):
        print "I put bread on the ham"
        print "I made a ham and cheese sandwich"
        goal.set("stop")
        self.print_activations()

    def stop_production(goal="stop"):
        self.print_activations()
        self.stop()

    def print_activations(self):
        list_of_chunks = self.DM.find_matching_chunks('')
        print "\nCurrent chunk activation values:"
        print "--------------------------------"
        for chunk in list_of_chunks:
            if chunk.has_key("condiment"):
                print chunk["customer"], chunk["condiment"], "-->", \
                      round(self.DM.get_activation(chunk), 3)
            else:
                print chunk["customer"], "-->", \
                      round(self.DM.get_activation(chunk), 3)
        print "\n"
```

[**py20**]
```
>>> from example_11_simple_spread import *
>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold None
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_time 0.05
   0.000 agent.DM.record_all_chunks False
   0.000 agent.DM.threshold 1
   0.000 agent.DM.latency 0.05
   0.000 agent.DM.busy False
   0.000 agent.DM.maximum_time 10.0
   0.000 agent.DM.error False
   0.000 agent.goal.chunk None
   0.000 agent.DMBuffer.chunk None
>>> # log = ccm.log(html=True)
>>> empty_environment.run()
```

```
   0.000 agent.goal.chunk sandwich bread

Current chunk activation values:
--------------------------------
customer mustard --> 2.649


   0.000 agent.production bread_bottom
   0.050 agent.production None
I have a piece of bread
   0.050 agent.goal.chunk sandwich cheese

Current chunk activation values:
--------------------------------
customer mustard --> 1.498


   0.050 agent.production cheese
   0.100 agent.production None
I put cheese on the bread
   0.100 agent.goal.chunk sandwich ham

Current chunk activation values:
--------------------------------
customer mustard --> 1.151


   0.100 agent.production ham
   0.150 agent.production None
I put ham on the cheese
   0.150 agent.goal.chunk customer condiment

Current chunk activation values:
--------------------------------
customer mustard --> 1.602


   0.150 agent.production condiment
   0.200 agent.production None
Recalling the order
   0.200 agent.DM.busy True
   0.200 agent.goal.chunk sandwich condiment

Current chunk activation values:
--------------------------------
customer mustard --> 0.805


   0.212 agent.DMBuffer.chunk condiment:mustard customer:customer
   0.212 agent.DM.busy False
   0.212 agent.production order
   0.262 agent.production None
I recall they wanted.......
mustard
I put the condiment on the sandwich
   0.262 agent.goal.chunk sandwich bread_top

Current chunk activation values:
--------------------------------
customer mustard --> 0.67


   0.262 agent.production bread_top
   0.312 agent.production None
I put bread on the ham
I made a ham and cheese sandwich
   0.312 agent.goal.chunk stop

Current chunk activation values:
--------------------------------
customer mustard --> 0.583


   0.312 agent.production stop_production
   0.362 agent.production None

Current chunk activation values:
--------------------------------
customer mustard --> 0.509


>>> ccm.finished()
```

8.4 Example: Partial Matching

In this model partial matching is turned on and the similarity between chunks is set. Therefore, when we add some noise it is possible for the model to recall the wrong chunk, especially if it has a high similarity level. The noise also makes it possible that the model will fail to make a retrieval at all. Run the model several times to see all of these effects.

(37) File **example_12_simple_partial.py**:

```
import ccm
from ccm.lib.actr import *


class MyEnvironment(ccm.Model):
    pass


class MyAgent(ACTR):
```

```python
goal = Buffer()

DMBuffer = Buffer()
DM = Memory(DMBuffer, latency=0.05, threshold=1)
# turn on some noise to allow errors
dm_n = DMNoise(DM, noise=0.6, baseNoise=0.0)
dm_bl = DMBaseLevel(DM, decay=0.5, limit=None)
dm_spread = DMSpreading(DM, goal)
dm_spread.strength = 2
dm_spread.weight[goal] = .5
# turn on partial matching
partial = Partial(DM, strength=1.0, limit=-1.0)
# set the similarity between customer1 and customer2 - they are very similar
partial.similarity("customer1", "customer2", -0.1)
# set the similarity between customer1 and customer3 - not so similar
partial.similarity("customer1", "customer3", -0.9)

def init():
    DM.add("customer:customer1 condiment:mustard")     # customer1's order
    DM.add("customer:customer2 condiment:ketchup")     # customer2's order
    DM.add("customer:customer3 condiment:mayonnaise")  # customer3's order
    goal.set("sandwich bread")
    self.print_activations()

def bread_bottom(goal="sandwich bread"):
    print "I have a piece of bread"
    goal.set("sandwich cheese")
    self.print_activations()

def cheese(goal="sandwich cheese"):
    print "I put cheese on the bread"
    goal.set("sandwich ham")
    self.print_activations()

def ham(goal="sandwich ham"):
    print "I put ham on the cheese"
    goal.set("customer1 condiment")
    self.print_activations()

# customer1 will spread activation to "customer1 mustard"
# but also some to "customer2 ketchup" and less to "customer3 mayonaise"
def condiment(goal="customer1 condiment"):
    print "Recalling the order"
    DM.request("customer:customer1 condiment:?condiment")
    goal.set("sandwich condiment")
    self.print_activations()

def order(goal="sandwich condiment", DMBuffer="customer:? condiment:?condiment"):
    print "I recall they wanted......."
    print condiment
```

```
            print "I put the condiment on the sandwich"
            goal.set("sandwich bread_top")
            self.print_activations()

        def forgot(goal="sandwich condiment", DMBuffer=None, DM="error:True"):
            print "I recall they wanted......."
            print "I forgot"
            goal.set("stop")
            self.print_activations()

        def bread_top(goal="sandwich bread_top"):
            print "I put bread on the ham"
            print "I made a ham and cheese sandwich"
            goal.set("stop")
            self.print_activations()

        def stop_production(goal="stop"):
            self.print_activations()
            self.stop()

        def print_activations(self):
            list_of_chunks = self.DM.find_matching_chunks('')
            print "\nCurrent chunk activation values:"
            print "---------------------------------"
            for chunk in list_of_chunks:
                if chunk.has_key("condiment"):
                    print chunk["customer"], chunk["condiment"], "-->", \
                          round(self.DM.get_activation(chunk), 3)
                else:
                    print chunk["customer"], "-->", \
                          round(self.DM.get_activation(chunk), 3)
            print "\n"
```

[**py21**]
```
>>> from example_12_simple_partial import *

>>> # run 1

>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold None
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_time 0.05
   0.000 agent.DM.record_all_chunks False
   0.000 agent.DM.threshold 1
   0.000 agent.DM.latency 0.05
   0.000 agent.DM.busy False
   0.000 agent.DM.maximum_time 10.0
   0.000 agent.DM.error False
   0.000 agent.goal.chunk None
   0.000 agent.DMBuffer.chunk None
>>> # log = ccm.log(html=True)
>>> empty_environment.run()
   0.000 agent.goal.chunk sandwich bread

Current chunk activation values:
--------------------------------
customer1 mustard --> 2.015
customer2 ketchup --> 3.702
customer3 mayonnaise --> 2.561


   0.000 agent.production bread_bottom
   0.050 agent.production None
I have a piece of bread
   0.050 agent.goal.chunk sandwich cheese

Current chunk activation values:
--------------------------------
customer1 mustard --> 1.873
customer2 ketchup --> -0.329
customer3 mayonnaise --> 3.091


   0.050 agent.production cheese
   0.100 agent.production None
I put cheese on the bread
   0.100 agent.goal.chunk sandwich ham

Current chunk activation values:
--------------------------------
customer1 mustard --> -1.561
customer2 ketchup --> 0.467
customer3 mayonnaise --> 1.915


   0.100 agent.production ham
   0.150 agent.production None
I put ham on the cheese
   0.150 agent.goal.chunk customer1 condiment

Current chunk activation values:
--------------------------------
customer1 mustard --> 2.021
customer2 ketchup --> 1.682
customer3 mayonnaise --> 0.534
```

```
   0.150 agent.production condiment
   0.200 agent.production None
Recalling the order
   0.200 agent.DM.busy True
   0.200 agent.goal.chunk sandwich condiment

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.575
customer2 ketchup --> 0.233
customer3 mayonnaise --> 0.996


   0.218 agent.DMBuffer.chunk condiment:mustard customer:customer1
   0.218 agent.DM.busy False
   0.218 agent.production order
   0.268 agent.production None
I recall they wanted.......
mustard
I put the condiment on the sandwich
   0.268 agent.goal.chunk sandwich bread_top

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.502
customer2 ketchup --> 1.383
customer3 mayonnaise --> -0.007


   0.268 agent.production bread_top
   0.318 agent.production None
I put bread on the ham
I made a ham and cheese sandwich
   0.318 agent.goal.chunk stop

Current chunk activation values:
--------------------------------
customer1 mustard --> 1.55
customer2 ketchup --> -0.113
customer3 mayonnaise --> 1.361


   0.318 agent.production stop_production
   0.368 agent.production None

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.325
customer2 ketchup --> 0.725
```

```
customer3 mayonnaise --> 0.699


>>> ccm.finished()

>>> # run 2

>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold None
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_time 0.05
   0.000 agent.DM.record_all_chunks False
   0.000 agent.DM.threshold 1
   0.000 agent.DM.latency 0.05
   0.000 agent.DM.busy False
   0.000 agent.DM.maximum_time 10.0
   0.000 agent.DM.error False
   0.000 agent.goal.chunk None
   0.000 agent.DMBuffer.chunk None
>>> # log = ccm.log(html=True)
>>> empty_environment.run()
   0.000 agent.goal.chunk sandwich bread

Current chunk activation values:
--------------------------------
customer1 mustard --> 2.415
customer2 ketchup --> 2.858
customer3 mayonnaise --> 3.014


   0.000 agent.production bread_bottom
   0.050 agent.production None
I have a piece of bread
   0.050 agent.goal.chunk sandwich cheese

Current chunk activation values:
--------------------------------
customer1 mustard --> -2.825
customer2 ketchup --> 1.186
customer3 mayonnaise --> 1.648


   0.050 agent.production cheese
   0.100 agent.production None
I put cheese on the bread
   0.100 agent.goal.chunk sandwich ham
```

```
Current chunk activation values:
--------------------------------
customer1 mustard --> 2.352
customer2 ketchup --> 1.047
customer3 mayonnaise --> 0.76


   0.100 agent.production ham
   0.150 agent.production None
I put ham on the cheese
   0.150 agent.goal.chunk customer1 condiment

Current chunk activation values:
--------------------------------
customer1 mustard --> -0.372
customer2 ketchup --> -2.112
customer3 mayonnaise --> 2.965


   0.150 agent.production condiment
   0.200 agent.production None
Recalling the order
   0.200 agent.DM.busy True
   0.200 agent.goal.chunk sandwich condiment

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.372
customer2 ketchup --> 0.453
customer3 mayonnaise --> 1.344


   0.216 agent.DMBuffer.chunk condiment:mustard customer:customer1
   0.216 agent.DM.busy False
   0.216 agent.production order
   0.266 agent.production None
I recall they wanted.......
mustard
I put the condiment on the sandwich
   0.266 agent.goal.chunk sandwich bread_top

Current chunk activation values:
--------------------------------
customer1 mustard --> 2.919
customer2 ketchup --> 1.858
customer3 mayonnaise --> -1.471


   0.266 agent.production bread_top
   0.316 agent.production None
I put bread on the ham
I made a ham and cheese sandwich
   0.316 agent.goal.chunk stop

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.05
customer2 ketchup --> 0.57
customer3 mayonnaise --> -0.623


   0.316 agent.production stop_production
   0.366 agent.production None

Current chunk activation values:
--------------------------------
customer1 mustard --> 1.214
customer2 ketchup --> 1.733
customer3 mayonnaise --> -0.115


>>> ccm.finished()

>>> # run 3

>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold None
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_time 0.05
   0.000 agent.DM.record_all_chunks False
   0.000 agent.DM.threshold 1
   0.000 agent.DM.latency 0.05
   0.000 agent.DM.busy False
   0.000 agent.DM.maximum_time 10.0
   0.000 agent.DM.error False
   0.000 agent.goal.chunk None
   0.000 agent.DMBuffer.chunk None
>>> # log = ccm.log(html=True)
>>> empty_environment.run()
   0.000 agent.goal.chunk sandwich bread

Current chunk activation values:
--------------------------------
customer1 mustard --> 1.379
customer2 ketchup --> 3.411
```

```
customer3 mayonnaise --> 2.887


   0.000 agent.production bread_bottom
   0.050 agent.production None
I have a piece of bread
   0.050 agent.goal.chunk sandwich cheese

Current chunk activation values:
--------------------------------
customer1 mustard --> 1.237
customer2 ketchup --> 4.474
customer3 mayonnaise --> 1.848


   0.050 agent.production cheese
   0.100 agent.production None
I put cheese on the bread
   0.100 agent.goal.chunk sandwich ham

Current chunk activation values:
--------------------------------
customer1 mustard --> 1.062
customer2 ketchup --> 0.953
customer3 mayonnaise --> -1.23


   0.100 agent.production ham
   0.150 agent.production None
I put ham on the cheese
   0.150 agent.goal.chunk customer1 condiment

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.77
customer2 ketchup --> 2.443
customer3 mayonnaise --> 0.947


   0.150 agent.production condiment
   0.200 agent.production None
Recalling the order
   0.200 agent.DM.busy True
   0.200 agent.goal.chunk sandwich condiment

Current chunk activation values:
--------------------------------
customer1 mustard --> 1.36
customer2 ketchup --> -0.252
customer3 mayonnaise --> -0.304
```

```
   0.205 agent.DMBuffer.chunk condiment:mustard customer:customer1
   0.205 agent.DM.busy False
   0.205 agent.production order
   0.255 agent.production None
I recall they wanted.......
mustard
I put the condiment on the sandwich
   0.255 agent.goal.chunk sandwich bread_top

Current chunk activation values:
--------------------------------
customer1 mustard --> 2.826
customer2 ketchup --> -1.809
customer3 mayonnaise --> -0.122


   0.255 agent.production bread_top
   0.305 agent.production None
I put bread on the ham
I made a ham and cheese sandwich
   0.305 agent.goal.chunk stop

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.591
customer2 ketchup --> -0.904
customer3 mayonnaise --> -0.201


   0.305 agent.production stop_production
   0.355 agent.production None

Current chunk activation values:
--------------------------------
customer1 mustard --> 2.086
customer2 ketchup --> 2.957
customer3 mayonnaise --> -1.912


>>> ccm.finished()
```

8.5 Example: Refraction

In ACT-R, a retrieval makes a chunk stronger so it is more likely to be recalled the next time. However, in some cases this is problematic. For example, if you are recalling different types of animals and the tiger has the highest activation it will be recalled first, and also second, and third, and so on. So there is a way to tell a production to retrieve a chunk that has not recently been recalled. In this model a sandwich maker has several different orders and makes them in the order he recalls them.

(38) File **example_13_simple_refraction.py**:

```
import ccm
from ccm.lib.actr import *


class MyEnvironment(ccm.Model):
    pass


class MyAgent(ACTR):

    goal = Buffer()

    DMBuffer = Buffer()
    # turn down threshold
    # maximum time - how long it will wait for a memory retrieval
    # finst_size - how many chunks can be kept track of
    # finst_time - how long a chunk can be kept track of
    DM = Memory(DMBuffer, latency=0.05, threshold=-25, maximum_time=20, \
                finst_size=10, finst_time=30.0)
    dm_n = DMNoise(DM, noise=0.0, baseNoise=0.0)
    dm_bl = DMBaseLevel(DM, decay=0.5, limit=None)
    dm_spread = DMSpreading(DM, goal)
    dm_spread.strength = 2
    dm_spread.weight[goal] = .5
    partial = Partial(DM, strength=1.0, limit=-1.0)
    partial.similarity("customer1", "customer2", -0.1)
    partial.similarity("customer1", "customer3", -0.9)

    # note that this model uses slot names - slotname:slotcontent
    def init():
        # customer1's order
        DM.add("isa:order customer:customer1 type:ham_cheese condiment:mustard")
        # customer2's order
        DM.add("isa:order customer:customer2 type:ham_cheese condiment:ketchup")
        # customer3's order
        DM.add("isa:order customer:customer3 type:ham_cheese condiment:mayonnaise")
        # customer4's order
        DM.add("isa:order customer:customer4 type:ham_cheese condiment:hot_sauce")
        goal.set("isa:ingredient type:bread")
        self.print_activations()

    def bread_bottom(goal="isa:ingredient type:bread"):
        print "I have a piece of bread"
        goal.set("isa:ingredient type:cheese")
        self.print_activations()

    def cheese(goal="isa:ingredient type:cheese"):
        print "I put cheese on the bread"
```

```
            goal.set("isa:ingredient type:ham")
            self.print_activations()

        def ham(goal="isa:ingredient type:ham"):
            print "I put ham on the cheese"
            goal.set("isa:order customer:customer1 type:ham_cheese condiment:unknown")
            self.print_activations()

        def condiment(goal="isa:order customer:customer1 type:ham_cheese condiment:unknown"):
            print "Recalling the order"
            # retrieve something that has not recently been retrieved
            DM.request("isa:order type:ham_cheese", require_new=True)
            goal.set("retrieve_condiment")
            self.print_activations()

        def order(goal="retrieve_condiment", DMBuffer="isa:order type:ham_cheese condiment:?condi
            print "I recall they wanted......."
            print condiment_order
            print "I put the condiment on the sandwich"
            goal.set("isa:ingredient type:bread_top")
            self.print_activations()

        def bread_top(goal="isa:ingredient type:bread_top"):
            print "I put bread on the ham"
            print "I made a ham and cheese sandwich"
            goal.set("isa:ingredient type:bread")
            # clear the buffer for the next cycle
            DMBuffer.clear()
            self.print_activations()

        def print_activations(self):
            list_of_chunks = self.DM.find_matching_chunks('')
            print "\nCurrent chunk activation values:"
            print "--------------------------------"
            for chunk in list_of_chunks:
                if chunk.has_key("condiment"):
                    print chunk["customer"], chunk["condiment"], "-->", \
                          round(self.DM.get_activation(chunk), 3)
                else:
                    print chunk["customer"], "-->", \
                          round(self.DM.get_activation(chunk), 3)
            print "\n"
```

[**py22**]
```
>>> from example_13_simple_refraction import *
>>> tim = MyAgent()
>>> empty_environment = MyEnvironment()
>>> empty_environment.agent = tim
>>> ccm.log_everything(empty_environment)
   0.000 agent.production_threshold None
   0.000 agent.production_time_sd None
   0.000 agent.production_match_delay 0
   0.000 agent.production_time 0.05
   0.000 agent.DM.record_all_chunks False
   0.000 agent.DM.threshold -25
   0.000 agent.DM.latency 0.05
   0.000 agent.DM.busy False
   0.000 agent.DM.maximum_time 20
   0.000 agent.DM.error False
   0.000 agent.goal.chunk None
   0.000 agent.DMBuffer.chunk None
>>> # log = ccm.log(html=True)
>>> empty_environment.run(3)
   0.000 agent.goal.chunk isa:ingredient type:bread

Current chunk activation values:
--------------------------------
customer1 mustard --> 2.649
customer2 ketchup --> 2.649
customer3 mayonnaise --> 2.649
customer4 hot_sauce --> 2.649


   0.000 agent.production bread_bottom
   0.050 agent.production None
I have a piece of bread
   0.050 agent.goal.chunk isa:ingredient type:cheese

Current chunk activation values:
--------------------------------
customer1 mustard --> 1.498
customer2 ketchup --> 1.498
customer3 mayonnaise --> 1.498
customer4 hot_sauce --> 1.498


   0.050 agent.production cheese
   0.100 agent.production None
I put cheese on the bread
   0.100 agent.goal.chunk isa:ingredient type:ham

Current chunk activation values:
--------------------------------
customer1 mustard --> 1.151
customer2 ketchup --> 1.151
customer3 mayonnaise --> 1.151
customer4 hot_sauce --> 1.151


   0.100 agent.production ham
   0.150 agent.production None
I put ham on the cheese
   0.150 agent.goal.chunk condiment:unknown customer:customer1 isa:order type:ham_cheese

Current chunk activation values:
--------------------------------
customer1 mustard --> 1.993
customer2 ketchup --> 1.339
customer3 mayonnaise --> 1.339
customer4 hot_sauce --> 1.339


   0.150 agent.production condiment
   0.200 agent.production None
Recalling the order
   0.200 agent.DM.busy True
   0.200 agent.goal.chunk retrieve_condiment

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.805
customer2 ketchup --> 0.805
customer3 mayonnaise --> 0.805
customer4 hot_sauce --> 0.805


   0.208 agent.DMBuffer.chunk condiment:mustard customer:customer1 isa:order type:ham_cheese
   0.208 agent.DM.busy False
   0.208 agent.production order
   0.258 agent.production None
I recall they wanted.......
mustard
I put the condiment on the sandwich
   0.258 agent.goal.chunk isa:ingredient type:bread_top

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.678
customer2 ketchup --> 0.678
customer3 mayonnaise --> 0.678
customer4 hot_sauce --> 0.678


   0.258 agent.production bread_top
   0.308 agent.production None
I put bread on the ham
I made a ham and cheese sandwich
   0.308 agent.goal.chunk isa:ingredient type:bread
   0.308 agent.DMBuffer.chunk None
```

```
Current chunk activation values:
--------------------------------
customer1 mustard --> 0.589
customer2 ketchup --> 0.589
customer3 mayonnaise --> 0.589
customer4 hot_sauce --> 0.589


   0.308 agent.production bread_bottom
   0.358 agent.production None
I have a piece of bread
   0.358 agent.goal.chunk isa:ingredient type:cheese

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.514
customer2 ketchup --> 0.514
customer3 mayonnaise --> 0.514
customer4 hot_sauce --> 0.514


   0.358 agent.production cheese
   0.408 agent.production None
I put cheese on the bread
   0.408 agent.goal.chunk isa:ingredient type:ham

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.448
customer2 ketchup --> 0.448
customer3 mayonnaise --> 0.448
customer4 hot_sauce --> 0.448


   0.408 agent.production ham
   0.458 agent.production None
I put ham on the cheese
   0.458 agent.goal.chunk condiment:unknown customer:customer1 isa:order type:ham_cheese

Current chunk activation values:
--------------------------------
customer1 mustard --> 1.435
customer2 ketchup --> 0.781
customer3 mayonnaise --> 0.781
customer4 hot_sauce --> 0.781


   0.458 agent.production condiment
   0.508 agent.production None
Recalling the order
```

```
   0.508 agent.DM.busy True
   0.508 agent.goal.chunk retrieve_condiment

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.339
customer2 ketchup --> 0.339
customer3 mayonnaise --> 0.339
customer4 hot_sauce --> 0.339


   0.532 agent.DMBuffer.chunk condiment:hot_sauce customer:customer4 isa:order type:ham_cheese
   0.532 agent.DM.busy False
   0.532 agent.production order
   0.582 agent.production None
I recall they wanted.......
hot_sauce
I put the condiment on the sandwich
   0.582 agent.goal.chunk isa:ingredient type:bread_top

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.271
customer2 ketchup --> 0.271
customer3 mayonnaise --> 0.271
customer4 hot_sauce --> 0.271


   0.582 agent.production bread_top
   0.632 agent.production None
I put bread on the ham
I made a ham and cheese sandwich
   0.632 agent.goal.chunk isa:ingredient type:bread
   0.632 agent.DMBuffer.chunk None

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.229
customer2 ketchup --> 0.229
customer3 mayonnaise --> 0.229
customer4 hot_sauce --> 0.229


   0.632 agent.production bread_bottom
   0.682 agent.production None
I have a piece of bread
   0.682 agent.goal.chunk isa:ingredient type:cheese

Current chunk activation values:
--------------------------------
```

```
customer1 mustard --> 0.191
customer2 ketchup --> 0.191
customer3 mayonnaise --> 0.191
customer4 hot_sauce --> 0.191


   0.682 agent.production cheese
   0.732 agent.production None
I put cheese on the bread
   0.732 agent.goal.chunk isa:ingredient type:ham

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.156
customer2 ketchup --> 0.156
customer3 mayonnaise --> 0.156
customer4 hot_sauce --> 0.156


   0.732 agent.production ham
   0.782 agent.production None
I put ham on the cheese
   0.782 agent.goal.chunk condiment:unknown customer:customer1 isa:order type:ham_cheese

Current chunk activation values:
--------------------------------
customer1 mustard --> 1.167
customer2 ketchup --> 0.514
customer3 mayonnaise --> 0.514
customer4 hot_sauce --> 0.514


   0.782 agent.production condiment
   0.832 agent.production None
Recalling the order
   0.832 agent.DM.busy True
   0.832 agent.goal.chunk retrieve_condiment

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.092
customer2 ketchup --> 0.092
customer3 mayonnaise --> 0.092
customer4 hot_sauce --> 0.092


   0.863 agent.DMBuffer.chunk condiment:ketchup customer:customer2 isa:order type:ham_cheese
   0.863 agent.DM.busy False
   0.863 agent.production order
   0.913 agent.production None
```

```
I recall they wanted.......
ketchup
I put the condiment on the sandwich
   0.913 agent.goal.chunk isa:ingredient type:bread_top

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.046
customer2 ketchup --> 0.046
customer3 mayonnaise --> 0.046
customer4 hot_sauce --> 0.046


   0.913 agent.production bread_top
   0.963 agent.production None
I put bread on the ham
I made a ham and cheese sandwich
   0.963 agent.goal.chunk isa:ingredient type:bread
   0.963 agent.DMBuffer.chunk None

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.019
customer2 ketchup --> 0.019
customer3 mayonnaise --> 0.019
customer4 hot_sauce --> 0.019


   0.963 agent.production bread_bottom
   1.013 agent.production None
I have a piece of bread
   1.013 agent.goal.chunk isa:ingredient type:cheese

Current chunk activation values:
--------------------------------
customer1 mustard --> -0.006
customer2 ketchup --> -0.006
customer3 mayonnaise --> -0.006
customer4 hot_sauce --> -0.006


   1.013 agent.production cheese
   1.063 agent.production None
I put cheese on the bread
   1.063 agent.goal.chunk isa:ingredient type:ham

Current chunk activation values:
--------------------------------
customer1 mustard --> -0.03
customer2 ketchup --> -0.03
```

```
customer3 mayonnaise --> -0.03
customer4 hot_sauce --> -0.03


   1.063 agent.production ham
   1.113 agent.production None
I put ham on the cheese
   1.113 agent.goal.chunk condiment:unknown customer:customer1 isa:order type:ham_cheese

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.991
customer2 ketchup --> 0.337
customer3 mayonnaise --> 0.337
customer4 hot_sauce --> 0.337


   1.113 agent.production condiment
   1.163 agent.production None
Recalling the order
   1.163 agent.DM.busy True
   1.163 agent.goal.chunk retrieve_condiment

Current chunk activation values:
--------------------------------
customer1 mustard --> -0.075
customer2 ketchup --> -0.075
customer3 mayonnaise --> -0.075
customer4 hot_sauce --> -0.075


   1.199 agent.DMBuffer.chunk condiment:mayonnaise customer:customer3 isa:order type:ham_chees
   1.199 agent.DM.busy False
   1.199 agent.production order
   1.249 agent.production None
I recall they wanted.......
mayonnaise
I put the condiment on the sandwich
   1.249 agent.goal.chunk isa:ingredient type:bread_top

Current chunk activation values:
--------------------------------
customer1 mustard --> -0.111
customer2 ketchup --> -0.111
customer3 mayonnaise --> -0.111
customer4 hot_sauce --> -0.111


   1.249 agent.production bread_top
   1.299 agent.production None
```

```
I put bread on the ham
I made a ham and cheese sandwich
   1.299 agent.goal.chunk isa:ingredient type:bread
   1.299 agent.DMBuffer.chunk None

Current chunk activation values:
--------------------------------
customer1 mustard --> -0.131
customer2 ketchup --> -0.131
customer3 mayonnaise --> -0.131
customer4 hot_sauce --> -0.131


   1.299 agent.production bread_bottom
   1.349 agent.production None
I have a piece of bread
   1.349 agent.goal.chunk isa:ingredient type:cheese

Current chunk activation values:
--------------------------------
customer1 mustard --> -0.15
customer2 ketchup --> -0.15
customer3 mayonnaise --> -0.15
customer4 hot_sauce --> -0.15


   1.349 agent.production cheese
   1.399 agent.production None
I put cheese on the bread
   1.399 agent.goal.chunk isa:ingredient type:ham

Current chunk activation values:
--------------------------------
customer1 mustard --> -0.168
customer2 ketchup --> -0.168
customer3 mayonnaise --> -0.168
customer4 hot_sauce --> -0.168


   1.399 agent.production ham
   1.449 agent.production None
I put ham on the cheese
   1.449 agent.goal.chunk condiment:unknown customer:customer1 isa:order type:ham_cheese

Current chunk activation values:
--------------------------------
customer1 mustard --> 0.858
customer2 ketchup --> 0.205
customer3 mayonnaise --> 0.205
customer4 hot_sauce --> 0.205
```

```
   1.449 agent.production condiment
   1.499 agent.production None
Recalling the order
   1.499 agent.DM.busy True
   1.499 agent.goal.chunk retrieve_condiment

Current chunk activation values:
--------------------------------
customer1 mustard --> -0.203
customer2 ketchup --> -0.203
customer3 mayonnaise --> -0.203
customer4 hot_sauce --> -0.203
```

```
>>> ccm.finished()
```

References

Anderson, John R. and Christian Lebiere (1998). *The Atomic Components of Thought*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Lewandowsky, S. and S. Farrell (2010). *Computational Modeling in Cognition: Principles and Practice*. Thousand Oaks, CA, USA: SAGE Publications.

Poore, Geoffrey M. (2013). “Reproducible Documents with PythonTeX”. In: *Proceedings of the 12th Python in Science Conference*. Ed. by Stéfan van der Walt et al., pp. 78–84.

Stewart, Terrence C. (2007). “A Methodology for Computational Cognitive Modelling”. PhD thesis. Ottawa, Ontario: Carleton University.

Stewart, Terrence C. and Robert L. West (2007). “Deconstructing and reconstructing ACT-R: Exploring the architectural space”. In: *Cognitive Systems Research* 8, pp. 227–236.