

Declarative Memory Related Issues

In this text we are going to investigate some of the most common problems which can arise when working with models that rely on the use of declarative retrievals when the subsymbolic calculations are enabled and thus activation affects the results. Unlike the models in previous debugging texts, the model which accompanies this text will not be performing any particular task. Instead, it is just a set of productions and declarative knowledge which allows us to see some of the issues which can occur when working with declarative memory and discuss how to determine what has happened and ways to address that in the model.

The Model Design

The model for this exercise, `declarative-issues.lisp`, is designed to perform a sequence of declarative retrievals and its declarative memory has been initialized with chunks that allow for the demonstration of specific retrieval situations. The goal and imaginal buffers will be used to control the sequencing of the productions and also to facilitate some of the retrieval situations. The specific details of the chunk-types and chunks involved will be described in the sections where it is meaningful below. The subsymbolic calculations for the model have been enabled using all of the activation equation's components. Most of the other parameters are left at their default values, or, in the case of those which are off by default like noise, have been set to values which reflect reasonable starting points based on the models from the tutorial and/or past ACT-R research.

First Retrieval Request

There are no warnings when this model is loaded. So, there are no syntax errors or other problems which we must fix before trying to run it. Because there is no task associated with this model we will just use the ACT-R run command to run it. The first retrieval which this model makes is for a chunk of type `simple-value` which has some value in the result slot (it is not `nil`) made by this production:

```
(p p1
  ?goal>
  buffer empty
==>
  +goal>
    isa task-state
  +retrieval>
    isa simple-value
  - result nil
)
```

This is the definition of the simple-value chunk-type:

```
(chunk-type simple-value result)
```

and here are the initial chunks of that type which are placed into the model's declarative memory with add-dm:

```
(v1 isa simple-value result "true")
(v2 isa simple-value result "false")
(v3 isa simple-value result nil)
```

In addition to that, the initial activations of those chunks have been set as follows:

```
(set-base-levels (v1 1 -1500)
                  (v2 1 -1500)
                  (v3 1 -1500))
```

Those settings reflect one past occurrence for each chunk 1500 seconds ago. We will consider that as an unchangeable property of these chunks for purposes of addressing issues with their retrieval (assume that they were learned as the result of some previous actions which we consider to be working correctly). Based on the initial declarative memory and the request that p1 makes we expect either chunk v1 or v2 to be retrieved. When we run the model for up to 10 seconds we get the following result:

```
> (run 10)
0.000  PROCEDURAL          CONFLICT-RESOLUTION
0.050  PROCEDURAL          PRODUCTION-FIRED P1
0.050  PROCEDURAL          CLEAR-BUFFER GOAL
0.050  PROCEDURAL          CLEAR-BUFFER RETRIEVAL
0.050  GOAL                SET-BUFFER-CHUNK GOAL TASK-STATE0
0.050  DECLARATIVE         START-RETRIEVAL
0.050  PROCEDURAL          CONFLICT-RESOLUTION
1.050  DECLARATIVE         RETRIEVAL-FAILURE
1.050  PROCEDURAL          CONFLICT-RESOLUTION
1.050  -----            Stopped because no events left to process
```

The model stopped because it failed to retrieve a chunk which prevents any further productions from firing. We now need to figure out why it didn't retrieve either of those chunks. There are several ways one could go about doing that. We will start with the most straightforward and then discuss some of the other things that could be done. The basic mechanism for investigating issues with retrievals is to turn on the activation trace in the model and run it again. That is done by setting the :act parameter to a non-nil value. In the unit texts it was set to t, but like the trace-detail parameter, it can also be set to values of high, medium, or low to control how much detail is shown. We will run the model with each value below to show the differences in information provided.

We could make that change in the model file, save the model, and then load it, but since the model itself hasn't changed we don't really need to perform those steps. Instead, we can reset the model (using either the reset command or the Reset button in the Environment) and then just call sgp from the prompt to change the parameter value before running it again. Changing the parameters interactively like that can be a convenient way to debug a model, but may not always be possible; particularly if there is additional Lisp code involved which is responsible for resetting and running the model. Here is the trace of the actions taken to get the activation trace with a value of t (which is the same as the high detail trace):

```
> (reset)
DEFAULT
> (sgp :act t)
(T)
> (run 10)
0.000    PROCEDURAL          CONFLICT-RESOLUTION
0.050    PROCEDURAL          PRODUCTION-FIRED P1
0.050    PROCEDURAL          CLEAR-BUFFER GOAL
0.050    PROCEDURAL          CLEAR-BUFFER RETRIEVAL
0.050    GOAL                SET-BUFFER-CHUNK GOAL TASK-STATE0
0.050    DECLARATIVE         START-RETRIEVAL
Chunk V3 does not match
Computing activation for chunk V1
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references NIL
  creation time: -1500.0 decay: 0.5  Optimized-learning: T
base-level value: -2.9634798
Total base-level: -2.9634798
Computing activation spreading from buffers
  Spreading 1.0 from buffer GOAL chunk TASK-STATE0-0
  sources of activation are: NIL
Total spreading activation: 0.0
Computing partial matching component
  comparing slot RESULT
  Requested: - NIL  Chunk's slot value: "true"
  similarity: -1.0
  negation test with similarity not ms has no effect
  effective similarity value is 0.0
Total similarity score 0.0
Adding transient noise -0.17835411
Adding permanent noise 0.0
Chunk V1 has an activation of: -3.1418338
Chunk V1 has the current best activation -3.1418338
Computing activation for chunk V2
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references NIL
  creation time: -1500.0 decay: 0.5  Optimized-learning: T
base-level value: -2.9634798
Total base-level: -2.9634798
Computing activation spreading from buffers
  Spreading 1.0 from buffer GOAL chunk TASK-STATE0-0
  sources of activation are: NIL
Total spreading activation: 0.0
Computing partial matching component
  comparing slot RESULT
  Requested: - NIL  Chunk's slot value: "false"
  similarity: -1.0
```

```

negation test with similarity not ms has no effect
effective similarity value is 0.0
Total similarity score 0.0
Adding transient noise 0.23814109
Adding permanent noise 0.0
Chunk V2 has an activation of: -2.7253387
Chunk V2 is now the current best with activation -2.7253387
No chunk above the retrieval threshold: 0.0
    0.050    PROCEDURAL          CONFLICT-RESOLUTION
    1.050    DECLARATIVE        RETRIEVAL-FAILURE
    1.050    PROCEDURAL          CONFLICT-RESOLUTION
    1.050    -----            Stopped because no events left to process

```

Here is what we get when it is set to medium:

```

> (reset)
DEFAULT
> (sgp :act medium)
(MEDIUM)
> (run 10)
    0.000    PROCEDURAL          CONFLICT-RESOLUTION
    0.050    PROCEDURAL          PRODUCTION-FIRED P1
    0.050    PROCEDURAL          CLEAR-BUFFER GOAL
    0.050    PROCEDURAL          CLEAR-BUFFER RETRIEVAL
    0.050    GOAL                SET-BUFFER-CHUNK GOAL TASK-STATE0
    0.050    DECLARATIVE        START-RETRIEVAL
Computing activation for chunk V1
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references NIL
  creation time: -1500.0 decay: 0.5  Optimized-learning: T
base-level value: -2.9634798
Total base-level: -2.9634798
Computing activation spreading from buffers
  Spreading 1.0 from buffer GOAL chunk TASK-STATE0-0
    sources of activation are: NIL
Total spreading activation: 0.0
Computing partial matching component
  comparing slot RESULT
  Requested: - NIL  Chunk's slot value: "true"
  similarity: -1.0
  negation test with similarity not ms has no effect
  effective similarity value is 0.0
Total similarity score 0.0
Adding transient noise -0.17835411
Adding permanent noise 0.0
Chunk V1 has an activation of: -3.1418338
Chunk V1 has the current best activation -3.1418338
Computing activation for chunk V2
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references NIL
  creation time: -1500.0 decay: 0.5  Optimized-learning: T
base-level value: -2.9634798
Total base-level: -2.9634798
Computing activation spreading from buffers
  Spreading 1.0 from buffer GOAL chunk TASK-STATE0-0
    sources of activation are: NIL
Total spreading activation: 0.0
Computing partial matching component
  comparing slot RESULT

```

```

Requested: - NIL   Chunk's slot value: "false"
similarity: -1.0
negation test with similarity not ms has no effect
effective similarity value is 0.0
Total similarity score 0.0
Adding transient noise 0.23814109
Adding permanent noise 0.0
Chunk V2 has an activation of: -2.7253387
Chunk V2 is now the current best with activation -2.7253387
No chunk above the retrieval threshold: 0.0
    0.050   PROCEDURAL          CONFLICT-RESOLUTION
    1.050   DECLARATIVE        RETRIEVAL-FAILURE
    1.050   PROCEDURAL          CONFLICT-RESOLUTION
    1.050   -----            Stopped because no events left to process

```

Both of those traces display all of the details of the activation calculations for the chunks which matched the request. The difference between them is that the high detail trace also indicates the chunks which were of the requested chunk-type but which did not match the other constraints of the request.

Here is what we get when it is set to low:

```

> (reset)
DEFAULT
> (sgp :act low)
(Low)
> (run 10)
    0.000   PROCEDURAL          CONFLICT-RESOLUTION
    0.050   PROCEDURAL          PRODUCTION-FIRED P1
    0.050   PROCEDURAL          CLEAR-BUFFER GOAL
    0.050   PROCEDURAL          CLEAR-BUFFER RETRIEVAL
    0.050   GOAL                SET-BUFFER-CHUNK GOAL TASK-STATE0
    0.050   DECLARATIVE        START-RETRIEVAL
Chunk V1 has an activation of: -3.1418338
Chunk V2 has an activation of: -2.7253387
No chunk above the retrieval threshold: 0.0
    0.050   PROCEDURAL          CONFLICT-RESOLUTION
    1.050   DECLARATIVE        RETRIEVAL-FAILURE
    1.050   PROCEDURAL          CONFLICT-RESOLUTION
    1.050   -----            Stopped because no events left to process

```

When set to low only the final activation values are shown instead of all the details.

Before looking at the details of that retrieval, we will introduce another command which can also be useful for interactively debugging a model: with-parameters. Instead of changing the value of a parameter with sgp one can use the with-parameters command to temporarily set parameter values and evaluate some other commands. Thus, instead of setting the activation trace to low and then running the model we could have done the following to set both the activation trace and the standard trace detail to low and then run with those settings:

```

> (reset)
DEFAULT

```

```

> (with-parameters (:trace-detail low :act low)
  (run 10))
0.050 PROCEDURAL PRODUCTION-FIRED P1
0.050 GOAL SET-BUFFER-CHUNK GOAL TASK-STATE0
Chunk V1 has an activation of: -3.1418338
Chunk V2 has an activation of: -2.7253387
No chunk above the retrieval threshold: 0.0
1.050 DECLARATIVE RETRIEVAL-FAILURE
1.050 ----- Stopped because no events left to process

```

After the with-parameters call is done the parameter values will be automatically returned to the values that they had previously.

As for the retrieval, regardless of which trace we look at, the critical line is the last one:

```
No chunk above the retrieval threshold: 0.0
```

No chunk is retrieved because they all have activations below the current retrieval threshold.

Another way we could have investigated the chunks' activations is by using the sdp command to see the current declarative parameters for each chunk. That will include the current activation as well as the activation it had the last time it was attempted to be retrieved. When the model stopped we could call sdp to print out all of those chunks and their parameters like this:

```

> (sdp v1 v2 v3)
Declarative parameters for chunk V1:
:Activation -2.875
:Permanent-Noise 0.000
:Base-Level -2.964
:Reference-Count 1.000
:Creation-Time -1500.000
:Source-Spread 0.000
:Sjis ((V1 . 3.0))
:Similarities ((V1 . 0.0))
:Last-Retrieval-Activation -3.142
:Last-Retrieval-Time 0.050
Declarative parameters for chunk V2:
:Activation -2.888
:Permanent-Noise 0.000
:Base-Level -2.964
:Reference-Count 1.000
:Creation-Time -1500.000
:Source-Spread 0.000
:Sjis ((V2 . 3.0))
:Similarities ((V2 . 0.0))
:Last-Retrieval-Activation -2.725
:Last-Retrieval-Time 0.050
Declarative parameters for chunk V3:
:Activation -2.990
:Permanent-Noise 0.000
:Base-Level -2.964
:Reference-Count 1.000
:Creation-Time -1500.000
:Source-Spread 0.000
:Sjis ((V3 . 3.0))
:Similarities ((V3 . 0.0))

```

(V1 V2 V3)

That doesn't completely explain why things failed, but if we knew the retrieval threshold in advance and suspected that to be the problem then `sdp` may have been useful without having to run the model again. That same information is shown in the declarative memory viewer of the Environment, so that could also be used to investigate the chunks' activations.

Now that we know the problem is that the chunk activations are below the retrieval threshold there are basically two ways to address that: either lower the retrieval threshold or increase the activation of those chunks in some way. Decreasing the threshold is an easy thing to do since all it takes is adding the parameter setting to the model. Changing the activation of the chunks can be accomplished in several ways, but given our constraint of not adjusting their histories limits us to essentially two options. One way to increase their activations would be to use the `:blc` (base-level constant) parameter to add a fixed value to all chunk activations. Another way would be to add additional information to those chunks which could provide a way for spreading activation to increase their activations.

As is usually the case, there is no one "right" answer as to how to fix this and one has to consider their theory for how people are performing the task, any data which is available, and the possible implications of making the change to other components of the model. For this model we do not have a theory or data since we are not modeling a real task. The effects on other parts of the model are also not relevant at this point for the same reason. So, since we don't have any reason to pick one option over the others, for the purpose of the exercise we will set the retrieval threshold lower and then note the consequences this has later on in the model. Before doing so we will look at some of the potential issues from the other changes which one might want to consider.

If the `:blc` parameter is adjusted that will affect the activation of all of the chunks which are retrieved by the model. Since the time to retrieve a chunk depends on its activation, not only will setting `:blc` affect whether a chunk is retrieved but also how long it will take. Thus, that may then necessitate the adjustments of other parameters as well to keep the response times in line with the data if that is important. However, if response time is not important to the data being modeled, then adjusting `:blc` might be a simple way to help ensure that chunks exceed the retrieval threshold and are retrieved quickly (since a higher activation will mean faster retrieval).

Using spreading activation to increase the activation of the chunks might be a plausible mechanism for the task. If the knowledge is prespecified for the model, like we are doing here, then it may be easy to add some additional context to that information to facilitate spreading activation. For example, although the grouped recall example from unit 5 didn't use spreading activation it did have a chunk which represented the current list itself as a member of the group

chunks which could have been used for that purpose. If the model is learning the chunks on its own however then one needs to have a way for the model to generate its own context. One way that is often done is to include the model's current goal or imaginal chunk as a slot value in the memories it creates. That way, each new goal or problem representation provides a particular context. The biggest downside to using this approach is primarily the additional complexity it requires in the model. One now has to have that context information available to spread the activation in a slot of a buffer at the time of retrieval, and it may require additional retrievals to remember past contexts as well as the needed information.

Now we will set the threshold lower and see how that affects the model. The first question is how low do we want to set it? To really answer that we need to decide how likely we want the model to fail to retrieve a chunk when there is one which can be retrieved, and to determine that we need to know what the activation of the chunks are and how much noise there is in the activations. Knowing the activation and noise value we can compute the recall probability for a chunk using the equation presented in unit 4. However, right now for this model, we just want it to succeed essentially every time. Thus we want to pick a value significantly lower than the activation of the chunks involved. From the activation trace we see that the chunks involved have activations of about -3.14 and -2.73. Therefore if we set the threshold to -10 that should be sufficient since with the model's activation noise set to .25 the recall probability for a chunk with an activation of -3.14 will be extremely close to 1.0 with that threshold. So we will add that additional setting to the sgp call in the model:

```
(sgp :esc t :v t :bll .5 :ans .25 :mas 3 :mp 10 :rt -10)
```

We need to save that change, load the model, and run it again. [Of course, like the :act setting above, we could have tried that parameter change at the prompt instead of making that change in the file, but since we will likely require it as a part of this model it makes sense to add it now.] Here is the trace that we get now:

```
> (run 10)
0.000 PROCEDURAL CONFLICT-RESOLUTION
0.050 PROCEDURAL PRODUCTION-FIRED P1
0.050 PROCEDURAL CLEAR-BUFFER GOAL
0.050 PROCEDURAL CLEAR-BUFFER RETRIEVAL
0.050 GOAL SET-BUFFER-CHUNK GOAL TASK-STATE0
0.050 DECLARATIVE START-RETRIEVAL
0.050 PROCEDURAL CONFLICT-RESOLUTION
10.000 ----- Stopped because time limit reached
```

The model stopped because it reached the end of the 10 seconds we asked it to run and it did not retrieve the chunk in that time. There are a couple of things to consider now. First, did the model successfully retrieve the chunk? If so, why did it take at least 10 seconds to complete?

Then we have to decide if that amount of time is reasonable for the model in performing this task.

As to whether or not the model successfully retrieved the chunk we have several options for testing that. First, we could just run the model some more until we find either a successful retrieval of a retrieval failure. In this case that would work just fine because there are no other productions that could fire to interfere with that. Alternatively, we could reset it and enable the activation trace so that we have the details of what happened. Again, for this model that is not a difficult task since this happens early in the run and because there is no task which is running the model it's easy to stop it where we want without having to use the stepper. Instead of using those however we are going to introduce a new command that can also be helpful in situations like this. That command is called mp-show-queue and it allows us to look ahead in time to see what the model is expecting to do in the future without actually running it. Here is what we get when we call that now:

```
> (mp-show-queue)
Events in the queue:
  15.312    DECLARATIVE          RETRIEVED-CHUNK V2
  15.312    PROCEDURAL           CONFLICT-RESOLUTION
```

That shows us that the model will complete the retrieval at time 15.312 and then perform another conflict-resolution action. Just because we see an event scheduled to occur at some future time with mp-show-queue however does not mean that we will necessarily see that same action in the trace if we continue to run the model. That's because things can happen to change the situation before that time arrives. Thus, looking ahead at the model's actions like that can be very useful in situations where a delayed action, like a retrieval completion, could be superseded by a new retrieval. For example, if another production were to fire and make a retrieval request at time 11.0 that would interrupt the ongoing retrieval and we would not actually see that retrieved-chunk action at time 15.312 if we were to run the model. That doesn't happen in this model, and we would have seen the same results if we were to just run it that long:

```
> (run 15.32)
  0.000    PROCEDURAL          CONFLICT-RESOLUTION
  0.050    PROCEDURAL          PRODUCTION-FIRED P1
  0.050    PROCEDURAL          CLEAR-BUFFER GOAL
  0.050    PROCEDURAL          CLEAR-BUFFER RETRIEVAL
  0.050    GOAL                SET-BUFFER-CHUNK GOAL TASK-STATE0
  0.050    DECLARATIVE         START-RETRIEVAL
  0.050    PROCEDURAL          CONFLICT-RESOLUTION
  15.312    DECLARATIVE         RETRIEVED-CHUNK V2
  15.312    DECLARATIVE         SET-BUFFER-CHUNK RETRIEVAL V2
  15.312    PROCEDURAL          CONFLICT-RESOLUTION
  15.320    -----           Stopped because time limit reached
```

Although it's not really necessary here, mp-show-queue is a useful tool to know about and can be very helpful in some situations.

Now that we know the retrieval did succeed we should consider the time that it took to do so. As shown in unit 4 the time it takes to retrieve a chunk depends on its activation by the equation:

$$Time = Fe^{-A}$$

The activation of our chunk is about -2.73 and the value of F is the latency factor parameter (:lf) in the model which defaults to 1. Since we don't set that parameter in our model the time to retrieve the chunk should be about $e^{2.73} \approx 15.3$ seconds, which is what we see in the trace.

If the model were performing a real task, particularly if we had data for comparison, we might now want to consider if a retrieval of that length is acceptable for the model, and if not, what we should do about it. Since this model is not performing any particular task we don't really have any basis for judging the length of that retrieval time, but we can still consider how we would change it if we wanted to. Based on the equation for the retrieval time there are two things we can do to affect the time. The first would be to change the activation of the chunk, and that could be done in the same ways as were discussed previously. The other option would be to change the :lf parameter. The thing to keep in mind when changing :lf is that it will affect all of the retrievals which the model performs. As an example we will change :lf for this model to decrease the time it takes to complete retrievals by setting it to .8:

```
(sgp :esc t :v t :bll .5 :ans .25 :mas 2 :mp 10 :rt -10 :lf .8)
```

Saving that change and then reloading the model here is what the trace looks like now with the activation trace set to low to show that while the time of the retrieval has changed the activations of the chunks are the same as they were previously:

```
> (run 12.26)
0.000 PROCEDURAL CONFLICT-RESOLUTION
0.050 PROCEDURAL PRODUCTION-FIRED P1
0.050 PROCEDURAL CLEAR-BUFFER GOAL
0.050 PROCEDURAL CLEAR-BUFFER RETRIEVAL
0.050 GOAL SET-BUFFER-CHUNK GOAL TASK-STATE0
0.050 DECLARATIVE START-RETRIEVAL
Chunk V1 has an activation of: -3.1418338
Chunk V2 has an activation of: -2.7253387
Chunk V2 with activation -2.7253387 is the best
0.050 PROCEDURAL CONFLICT-RESOLUTION
12.259 DECLARATIVE RETRIEVED-CHUNK V2
12.259 DECLARATIVE SET-BUFFER-CHUNK RETRIEVAL V2
12.259 PROCEDURAL CONFLICT-RESOLUTION
12.260 ----- Stopped because time limit reached
```

Second Retrieval Request

The second retrieval request the model makes is very similar to the first one. The only difference is that in this request there is no constraint placed on the retrieved chunk other than that it be of type simple-fact:

```
(p p2
  =goal>
    isa task-state
    state r2
  =retrieval>
    isa simple-value
==>
  =goal>
    state r3
  +retrieval>
    isa simple-value)
```

Here is a portion of the trace from running the model for this retrieval with the activation trace enabled:

```
> (run 19.52)
...
12.309    PROCEDURAL          PRODUCTION-FIRED P2
12.309    PROCEDURAL          CLEAR-BUFFER RETRIEVAL
12.309    DECLARATIVE         START-RETRIEVAL
Computing activation for chunk V1
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references NIL
  creation time: -1500.0 decay: 0.5  Optimized-learning: T
base-level value: -2.9675493
Total base-level: -2.9675493
Computing activation spreading from buffers
  Spreading 1.0 from buffer GOAL chunk TASK-STATE0-0
    sources of activation are: (R3)
  Spreading activation 0.0 from source R3 level 1.0 times Sji 0.0
Total spreading activation: 0.0
Computing partial matching component
Total similarity score 0.0
Adding transient noise 0.08887799
Adding permanent noise 0.0
Chunk V1 has an activation of: -2.8786714
Chunk V1 has the current best activation -2.8786714
Computing activation for chunk V2
Computing base-level
Starting with blc: 0.0
Computing base-level from 2 references NIL
  creation time: -1500.0 decay: 0.5  Optimized-learning: T
base-level value: -2.2744021
Total base-level: -2.2744021
Computing activation spreading from buffers
  Spreading 1.0 from buffer GOAL chunk TASK-STATE0-0
    sources of activation are: (R3)
  Spreading activation 0.0 from source R3 level 1.0 times Sji 0.0
Total spreading activation: 0.0
```

```

Computing partial matching component
Total similarity score 0.0
Adding transient noise 0.07606379
Adding permanent noise 0.0
Chunk V2 has an activation of: -2.1983383
Chunk V2 is now the current best with activation -2.1983383
Computing activation for chunk V3
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references NIL
  creation time: -1500.0 decay: 0.5  Optimized-learning: T
base-level value: -2.9675493
Total base-level: -2.9675493
Computing activation spreading from buffers
  Spreading 1.0 from buffer GOAL chunk TASK-STATE0-0
    sources of activation are: (R3)
  Spreading activation 0.0 from source R3 level 1.0 times Sji 0.0
Total spreading activation: 0.0
Computing partial matching component
Total similarity score 0.0
Adding transient noise -0.026406968
Adding permanent noise 0.0
Chunk V3 has an activation of: -2.9939563
Chunk V2 with activation -2.1983383 is the best
12.309  PROCEDURAL  CONFLICT-RESOLUTION
19.517  DECLARATIVE  RETRIEVED-CHUNK V2
19.517  DECLARATIVE  SET-BUFFER-CHUNK RETRIEVAL V2
19.517  PROCEDURAL  CONFLICT-RESOLUTION
19.520  -----  Stopped because time limit reached

```

We see that it retrieves chunk v2 again this time, and that this retrieval is faster than the last time requiring only a little more than 7 seconds instead of around 12. That is because the chunk has an extra reference now since it was retrieved previously and thus it has a higher base-level activation than the other two chunks of type simple-value.

That is the expected result of base-level learning, activation of the chunk increases with practice which makes it more likely to be retrieved and faster when it is. However, there is a potential issue that can arise with respect to base-level learning. The issue to be wary of is that when using requests to declarative memory with few constraints on the contents of that chunk a single chunk may come to dominate and always be retrieved. In some situations that may be desirable, but in other situations one may not want one chunk to dominate like that.

If one does not want a single chunk to dominate, but can't provide additional constraints in the request or change the context to affect the other components of the activation equation then one may need to take advantage of the declarative finsts which were described in unit 3 of the tutorial. They can be used to suppress the retrieval of a chunk which has been recently retrieved so that a single chunk is not retrieved repeatedly. To do so one needs to add the :recently-retrieved request parameter to the request which is made to the retrieval buffer with a value of nil. That will then remove chunks which are currently marked with a declarative finst from those considered for that request. For demonstration purposes we will make that change to the request made in p2 and see the difference. Here is the new version of p2:

```

(p p2
  =goal>
    isa task-state
    state r2
  =retrieval>
    isa simple-value
==>
  =goal>
    state r3
  +retrieval>
    isa simple-value
    :recently-retrieved nil)

```

and here is the trace showing that retrieval now:

```

> (run 26.55)
...
12.309   PROCEDURAL           PRODUCTION-FIRED P2
12.309   PROCEDURAL           CLEAR-BUFFER RETRIEVAL
12.309   DECLARATIVE          START-RETRIEVAL
Removing recently retrieved chunks:
V2
Computing activation for chunk V1
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references NIL
  creation time: -1500.0 decay: 0.5  Optimized-learning: T
base-level value: -2.9675493
Total base-level: -2.9675493
Computing activation spreading from buffers
  Spreading 1.0 from buffer GOAL chunk TASK-STATE0-0
    sources of activation are: (R3)
  Spreading activation 0.0 from source R3 level 1.0 times Sji 0.0
Total spreading activation: 0.0
Computing partial matching component
Total similarity score 0.0
Adding transient noise 0.08887799
Adding permanent noise 0.0
Chunk V1 has an activation of: -2.8786714
Chunk V1 has the current best activation -2.8786714
Computing activation for chunk V3
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references NIL
  creation time: -1500.0 decay: 0.5  Optimized-learning: T
base-level value: -2.9675493
Total base-level: -2.9675493
Computing activation spreading from buffers
  Spreading 1.0 from buffer GOAL chunk TASK-STATE0-0
    sources of activation are: (R3)
  Spreading activation 0.0 from source R3 level 1.0 times Sji 0.0
Total spreading activation: 0.0
Computing partial matching component
Total similarity score 0.0
Adding transient noise 0.07606379
Adding permanent noise 0.0
Chunk V3 has an activation of: -2.8914855
Chunk V1 with activation -2.8786714 is the best
12.309   PROCEDURAL           CONFLICT-RESOLUTION
26.541   DECLARATIVE          RETRIEVED-CHUNK V1
26.541   DECLARATIVE          SET-BUFFER-CHUNK RETRIEVAL V1

```

```

26.541    PROCEDURAL          CONFLICT-RESOLUTION
26.550    -----          Stopped because time limit reached

```

This time chunk v1 was retrieved because chunk v2 was removed from consideration since it has a declarative first on it from the previous retrieval. If one wants to see which chunks are currently marked with a declarative first the print-dm-finsts command can be used. Here is the result of that after the run shown above:

```

> (print-dm-finsts)

Chunk name      Time Stamp
-----
V1              26.541

```

Note that only v1 currently has a first on it at this time because the default duration of declarative finsts is 3 seconds and more than that amount of time has passed since v2 was retrieved. If we instead check at time 12.3, after the first retrieval has completed and just before the second request is made, we will see that v2 does have a declarative first on it:

```

> (reset)
DEFAULT
> (run 12.3)
  0.000    PROCEDURAL          CONFLICT-RESOLUTION
  0.050    PROCEDURAL          PRODUCTION-FIRED P1
  0.050    PROCEDURAL          CLEAR-BUFFER GOAL
  0.050    PROCEDURAL          CLEAR-BUFFER RETRIEVAL
  0.050    GOAL                SET-BUFFER-CHUNK GOAL TASK-STATE0
  0.050    DECLARATIVE         START-RETRIEVAL
  0.050    PROCEDURAL          CONFLICT-RESOLUTION
 12.259    DECLARATIVE         RETRIEVED-CHUNK V2
 12.259    DECLARATIVE         SET-BUFFER-CHUNK RETRIEVAL V2
 12.259    PROCEDURAL          CONFLICT-RESOLUTION
 12.300    -----          Stopped because time limit reached
12.3
22
NIL
> (print-dm-finsts)

Chunk name      Time Stamp
-----
V2              12.259

```

If one cannot use finsts or some mechanism to affect the activation of the chunks, but still needs to avoid issues with a single chunk becoming dominant there are some other options available if one adds optional components to the ACT-R system. Distributed with the ACT-R source code are several extensions which have been developed for ACT-R. Those optional components are found in the extras directory of the distribution and each one is found in a separate subdirectory. Two of the extensions available affect the base-level learning equation and may help avoid the dominant chunk problem. Those two particular extras are in the spacing-effect and base-level-inhibition directories. Because they are not part of the standard ACT-R system we will not be

describing them in the tutorial materials, but one can find details and instructions on their use in the extra files provided.

Third Retrieval Request

The next retrieval request this model makes is for a chunk of type simple-fact which is defined like this:

```
(chunk-type simple-fact item attribute)
```

The production which makes the request is this one:

```
(p p3
  =goal>
    isa task-state
    state r3
  =retrieval>
    isa simple-value
  ==>
  =goal>
    state r4
  +retrieval>
    isa simple-fact
    attribute pink)
```

It matches the goal state and retrieval which result from the previous production's actions and makes a request for a simple-fact with an attribute slot value of pink.

Here are the simple-fact chunks which the model starts with in its declarative memory from the add-dm command in the model definition:

```
(f1 isa simple-fact item sky attribute blue)
(f2 isa simple-fact item rose attribute red)
(f3 isa simple-fact item grass attribute green)
```

There are no base-level activation values set for those chunks, thus for now they will each have one reference which occurs at time 0 because that's when they are added to the model's declarative memory.

Notice that none of the chunks have an attribute which matches the request which is being made, but this model does have partial matching enabled so perhaps one of them will still be retrieved. We will run it to find out what happens, and here is the result of running the model for 30 seconds:

```
> (run 30)
0.000   PROCEDURAL          CONFLICT-RESOLUTION
0.050   PROCEDURAL          PRODUCTION-FIRED P1
0.050   PROCEDURAL          CLEAR-BUFFER GOAL
0.050   PROCEDURAL          CLEAR-BUFFER RETRIEVAL
```

0.050	GOAL	SET-BUFFER-CHUNK GOAL TASK-STATE0
0.050	DECLARATIVE	START-RETRIEVAL
0.050	PROCEDURAL	CONFLICT-RESOLUTION
12.259	DECLARATIVE	RETRIEVED-CHUNK V2
12.259	DECLARATIVE	SET-BUFFER-CHUNK RETRIEVAL V2
12.259	PROCEDURAL	CONFLICT-RESOLUTION
12.309	PROCEDURAL	PRODUCTION-FIRED P2
12.309	PROCEDURAL	CLEAR-BUFFER RETRIEVAL
12.309	DECLARATIVE	START-RETRIEVAL
12.309	PROCEDURAL	CONFLICT-RESOLUTION
26.541	DECLARATIVE	RETRIEVED-CHUNK V1
26.541	DECLARATIVE	SET-BUFFER-CHUNK RETRIEVAL V1
26.541	PROCEDURAL	CONFLICT-RESOLUTION
26.591	PROCEDURAL	PRODUCTION-FIRED P3
26.591	PROCEDURAL	CLEAR-BUFFER RETRIEVAL
26.591	DECLARATIVE	START-RETRIEVAL
26.591	PROCEDURAL	CONFLICT-RESOLUTION
30.000	-----	Stopped because time limit reached

It has not retrieved a chunk at that point. Instead of continuing to run, we will again look ahead with mp-show-queue:

```
> (mp-show-queue)
Events in the queue:
17647.763  DECLARATIVE      RETRIEVAL-FAILURE
17647.763  PROCEDURAL        CONFLICT-RESOLUTION
```

That shows that the model has failed to retrieve a chunk as a result of that request and that it is going to take over 17000 seconds while trying. First we will look at why the model failed to retrieve a chunk and then we will consider why it takes so long when it fails.

Turning on the activation trace and running again we see these activation computations for this request:

```
...
26.591  PROCEDURAL      PRODUCTION-FIRED P3
26.591  PROCEDURAL      CLEAR-BUFFER RETRIEVAL
26.591  DECLARATIVE     START-RETRIEVAL
26.591  DECLARATIVE     START-RETRIEVAL
Computing activation for chunk F1
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references (0.0)
  creation time: 0.0 decay: 0.5 Optimized-learning: T
base-level value: -0.9471392
Total base-level: -0.9471392
Computing activation spreading from buffers
  Spreading 1.0 from buffer GOAL chunk TASK-STATE0-0
    sources of activation are: (R3)
  Spreading activation 0.0 from source R3 level 1.0 times Sji 0.0
Total spreading activation: 0.0
Computing partial matching component
  comparing slot ATTRIBUTE
  Requested: = PINK Chunk's slot value: BLUE
  similarity: -1.0
  effective similarity value is -10.0
```



```

Total similarity score -10.0
Adding transient noise -0.026406968
Adding permanent noise 0.0
Chunk F1 has an activation of: -10.973546
Chunk F1 has the current best activation -10.973546
Computing activation for chunk F2
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references (0.0)
  creation time: 0.0 decay: 0.5  Optimized-learning: T
base-level value: -0.9471392
Total base-level: -0.9471392
Computing activation spreading from buffers
  Spreading 1.0 from buffer GOAL chunk TASK-STATE0-0
    sources of activation are: (R3)
  Spreading activation 0.0 from source R3 level 1.0 times Sji 0.0
Total spreading activation: 0.0
Computing partial matching component
  comparing slot ATTRIBUTE
  Requested: = PINK  Chunk's slot value: RED
  similarity: -1.0
  effective similarity value is -10.0
Total similarity score -10.0
Adding transient noise -0.42355087
Adding permanent noise 0.0
Chunk F2 has an activation of: -11.370689
Computing activation for chunk F3
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references (0.0)
  creation time: 0.0 decay: 0.5  Optimized-learning: T
base-level value: -0.9471392
Total base-level: -0.9471392
Computing activation spreading from buffers
  Spreading 1.0 from buffer GOAL chunk TASK-STATE0-0
    sources of activation are: (R3)
  Spreading activation 0.0 from source R3 level 1.0 times Sji 0.0
Total spreading activation: 0.0
Computing partial matching component
  comparing slot ATTRIBUTE
  Requested: = PINK  Chunk's slot value: GREEN
  similarity: -1.0
  effective similarity value is -10.0
Total similarity score -10.0
Adding transient noise 0.43639642
Adding permanent noise 0.0
Chunk F3 has an activation of: -10.510742
Chunk F3 is now the current best with activation -10.510742
No chunk above the retrieval threshold: -10.0

```

Each chunk has a base-level activation of around -0.947 but then loses more activation because of the mismatch to the requested slot value. Because we have not set any similarities in this model they all default to the worst mismatch value of -1, and because the :mp parameter is set to 10 in the sgp settings for the model, each chunk then has $10 * -1$ added to its activation for a total of around -10.947 before noise is added. None of the noise values are large enough to bring a chunk above the retrieval threshold which we set previously at -10. Thus, the model fails to retrieve any of them.

When there is a retrieval failure the time that it will take uses the same equation as for a successful retrieval, except that the retrieval threshold is used instead of a chunk's activation. Thus, with our current parameter settings a retrieval failure will take $.8 * e^{10}$ seconds, which is almost 5 hours of simulated time in which the model sits trying to retrieve a chunk before it fails. A delay of that long is likely to be unacceptable in any reasonable model, so we need to decide what to do about it. Since we don't have a task to guide us, we're free to explore many possible alternatives for how to address that. First we will consider options for making it more likely that one of the existing chunks will be retrieved, and then we will consider how we can handle things if the model does still have retrieval failures.

If we want one of these chunks to be retrieved then we need to either raise their activations or lower the retrieval threshold again. If we were to lower the threshold then the retrieval of these chunks would take as long as the retrieval failure did now, and the lower we make the threshold the worse things are with respect to the time it takes when there is a failure. So we will not consider that a good choice at this point. Instead, we will look at how we can raise the activation of these chunks.

There are three components to the activation equation: base-level, spreading activation, and partial matching and each of those provides opportunities to increase the chunk's activation. We discussed the spreading activation change with respect to the first retrieval, and again will not choose to modify the model in that way for this retrieval. Instead we will look at the options for affecting the other two components of activation.

For the chunks' base-levels we again have the option of setting the :blc parameter to increase every chunk's base-level. Having seen the problem of very long retrieval times for low threshold and activations, we may want to consider doing that so we can shift things to a level that produces more reasonable times, but we will come back to that in a later section. Another option which we have available for changing the base-level of these chunks is to explicitly set their base-level activations using the set-base-levels command. For the first retrieval we had considered the base-level settings of the chunks involved as fixed values, but for these chunks we will not. Thus, we could set their creation time and number of references to values that provide sufficiently large base-level activations. When choosing to modify the base-levels of chunks one should take into account what those chunks represent and what the values for creation time and references mean for the model. For example, if the chunks represent information which the model would not have learned prior to doing the task then their creation times probably shouldn't be any earlier than when the model would have started the task, and similarly they shouldn't have more references than would be reasonable in that time. However, if the chunks are representing background knowledge that is meant to represent information the model had long before doing the task then a much earlier creation time and larger reference count are warranted. For background knowledge of that nature it's often difficult to determine what would be an

appropriate creation time and number of references so more arbitrary values are used to achieve an appropriate activation for the task.

Setting the base-levels of the chunks for this retrieval seems like it would be a reasonable thing to do since they are representing facts one would assume a person would have learned long before the task and which have been encountered frequently. Thus, as a first step we will give those chunks a strong history with this setting:

```
(set-base-levels (f1 1000 -10000)
                 (f2 1000 -10000)
                 (f3 1000 -10000))
```

That gives the chunks a history which, while in absolute terms is probably not likely (having only learned the chunk 10000 seconds ago and having used it 1000 times since then), but may be sufficient to provide a fairly stable and relatively strong base-level activation over the course of the current task.

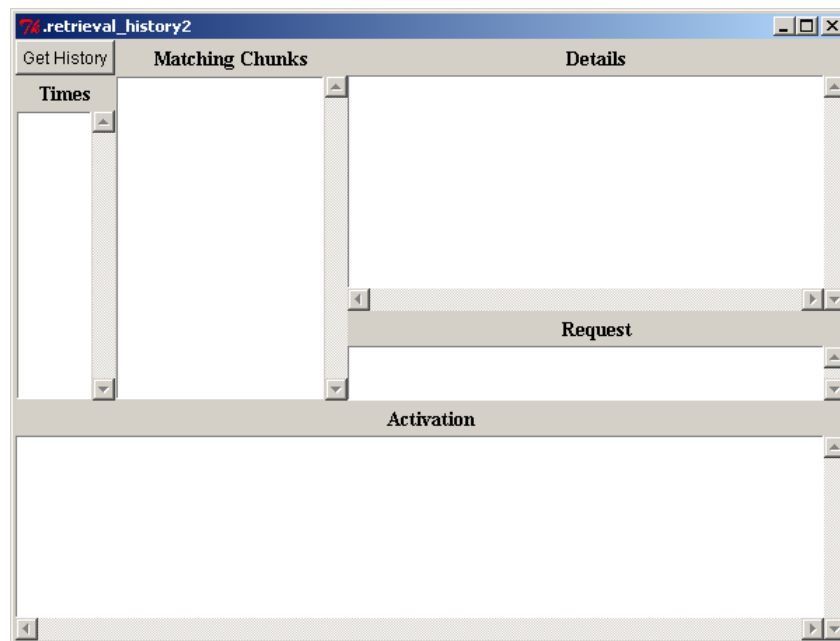
Running the model now we see that it does succeed in retrieving one of those chunks, but it still takes a significant amount of time:

```
> (run 30)
0.000 PROCEDURAL CONFLICT-RESOLUTION
0.050 PROCEDURAL PRODUCTION-FIRED P1
0.050 PROCEDURAL CLEAR-BUFFER GOAL
0.050 PROCEDURAL CLEAR-BUFFER RETRIEVAL
0.050 GOAL SET-BUFFER-CHUNK GOAL TASK-STATE0
0.050 DECLARATIVE START-RETRIEVAL
0.050 PROCEDURAL CONFLICT-RESOLUTION
12.259 DECLARATIVE RETRIEVED-CHUNK V2
12.259 DECLARATIVE SET-BUFFER-CHUNK RETRIEVAL V2
12.259 PROCEDURAL CONFLICT-RESOLUTION
12.309 PROCEDURAL PRODUCTION-FIRED P2
12.309 PROCEDURAL CLEAR-BUFFER RETRIEVAL
12.309 DECLARATIVE START-RETRIEVAL
12.309 PROCEDURAL CONFLICT-RESOLUTION
26.541 DECLARATIVE RETRIEVED-CHUNK V1
26.541 DECLARATIVE SET-BUFFER-CHUNK RETRIEVAL V1
26.541 PROCEDURAL CONFLICT-RESOLUTION
26.591 PROCEDURAL PRODUCTION-FIRED P3
26.591 PROCEDURAL CLEAR-BUFFER RETRIEVAL
26.591 DECLARATIVE START-RETRIEVAL
26.591 PROCEDURAL CONFLICT-RESOLUTION
30.000 ----- Stopped because time limit reached

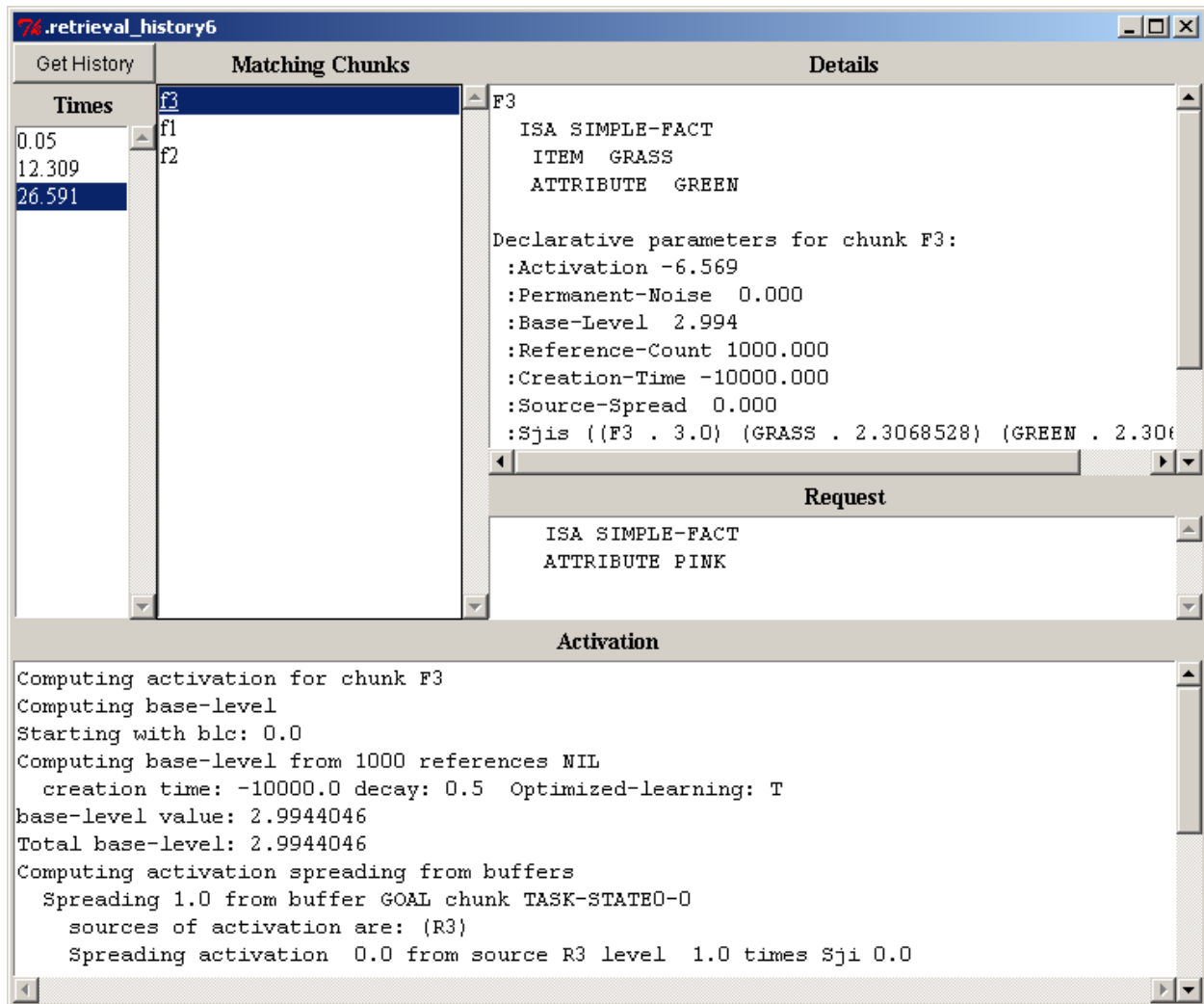
> (mp-show-queue)
Events in the queue:
596.830 DECLARATIVE RETRIEVED-CHUNK F3
596.830 PROCEDURAL CONFLICT-RESOLUTION
```

We could turn the activation trace on to see why that happens, but instead we will introduce another tool available in the Environment. The “Retrieval History” tool can be used to record all of the activation trace information which occurs as the model runs without printing it in the trace.

To use the tool it needs to be opened prior to running the model, like the other tracing and history tools. It will show a window which looks something like this:



With the window open the model should then be reset and run again. When it is done running, clicking the “Get History” button in the upper left of the Retrieval History window will cause the “Times” section to display all the times in the model run at which a retrieval request was made. Picking one of those times will then cause all of the chunks which matched the request to be displayed in the “Matching Chunks” section and the request which was made to be shown in the “Request” section. The top chunk in the list will be the one which was selected for retrieval, or it will be the symbol :retrieval-failure if there was no matching chunk with an activation above the retrieval threshold. Picking one of those chunks will then cause the “Details” section to display the chunk and its parameters at that time and the “Activation” section to display the complete activation trace for how that chunk’s activation was computed for that retrieval request at that time. Here is what we see for the chunk F3 which is the one that will be retrieved for the request made at time 26.591:



and here is all of the information from the activation section:

```

Computing activation for chunk F3
Computing base-level
Starting with blc: 0.0
Computing base-level from 1000 references NIL
  creation time: -10000.0 decay: 0.5 Optimized-learning: T
base-level value: 2.9944046
Total base-level: 2.9944046
Computing activation spreading from buffers
  Spreading 1.0 from buffer GOAL chunk TASK-STATE0-0
  sources of activation are: (R3)
  Spreading activation 0.0 from source R3 level 1.0 times Sji 0.0
Total spreading activation: 0.0
Computing partial matching component
  comparing slot ATTRIBUTE
  Requested: = PINK Chunk's slot value: GREEN
  similarity: -1.0
  effective similarity value is -10.0
Total similarity score -10.0
Adding transient noise 0.43639642
Adding permanent noise 0.0

```

Chunk F3 has an activation of: -6.5691986

Before looking at those results we will first mention that one can also get that information at the command line after the model runs instead of through the ACT-R Environment. First, the `:sact` parameter must be set to `t` before the model runs so that the declarative module saves the information. Then, the `saved-activation-history` command can be used to see when the retrievals occurred and which chunks were attempted, and the `print-activation-trace` and `print-chunk-activation-trace` commands can be used to get the activation information for a specific time and/or chunk. More details on those commands can be found in the reference manual.

Looking at the information in the trace we see that the base-level of that chunk looks strong (it would take less than 50ms to retrieve a chunk with an activation of ~ 3 with the current parameter settings) but it takes a big negative hit from the partial matching component of the equation. The other two chunks show a similar pattern, and it's only the noise value which differentiates them making `f3` the one to be retrieved. Since the base-level looks strong we will now consider the adjusting the partial matching component of the activations.

For partial matching there are three things to consider: what values are requested in the slots of the retrieval request, what the similarities are between those requested values and the values in the slots of the chunks in declarative memory, and the mismatch penalty parameter (`:mp`) setting. We will consider some of the issues related to each of those, and then decide what changes, if any, to make to the model.

Choosing how specific to make the retrieval request can be important in determining how likely that request is to succeed when using partial matching because each additional constraint in the request is an added opportunity to decrease the activation of the target chunks. That is because partial matching provides a penalty to chunks which do not match the request – it does not increase the activation of chunks which do (at least not under default and recommended parameter settings). If there is a chunk which matches the request then the specificity of the request doesn't really matter since there will be no penalty to that matching chunk. However, when the model is making requests in situations where it may not have a perfectly matching chunk (for instance in the one-hit blackjack game from the unit 5 assignment) one will need to carefully determine what is important to put in the request. If there are too many constraints the model may fail to find any chunk which is close enough to all of the constraints to be above the threshold, but conversely if there are too few constraints put on the request it may retrieve a chunk which is not really relevant to the current situation.

In the current model the chunks of interest for this request do not have a lot of slots to test and the request for a chunk based on a single constraint seems reasonable for the chunks involved. So, we will not adjust that aspect of the model.

The settings of the similarities between items and the :mp parameter are related, so we will discuss them together. By default a chunk is maximally similar to itself and it is maximally dissimilar to all other values. To have chunks related by some intermediate similarity the modeler must set those values. The question becomes how to decide what to make similar and how to set those similarities so that they provide the desired effects. In setting the similarities there are two things to consider: the magnitude of the effect a single mismatch will have on the activation and the relative similarity values among the items involved.

The total effect of similarity depends on how many mismatches there are as discussed with respect to the specificity of the request above, but the effect that each individual mismatch has is based on the similarity setting between the items involved and the setting of :mp. The default range for similarities is from -1 to 0, and the similarity value between the items is multiplied by the setting of the :mp parameter to determine the penalty to the activation. Setting similarities in the default range and then using :mp to scale them often works out well. However, because the :mp parameter is a constant used in all retrieval requests when one needs there to be different similarity effects for different types of items it may be necessary to change the range of similarities instead of just scaling them all with :mp. To do that one can change the similarity range by setting the :md and :ms parameters (maximum difference and maximum similarity respectively). The recommendation is to always leave :ms at 0, but :md can be set to any value needed to provide an appropriate range. When changing the range, it's often best to then just set :mp to 1 so the similarity values directly reflect the effect on activation, but that's not required and one can still scale them with :mp as well if desired.

How to set the relative similarities between items depends on what sorts of effects one would like the model to show. While it is possible to set each possible similarity value explicitly in the model to produce specific results, it's usually more plausible to set them systematically. In some situations one can rely on other experimental results for guidance in how to set them, for instance research on numbers, language, or perceptual effects may provide a general equation or metric to use in those situations, but other times one may need to determine values appropriate for the current task by parameter exploration or analysis of the data. If parameter estimation is required, one thing that may be useful in determining similarity values is to look at the activations of the competing chunks at the time of the request along with the activation noise value. Assuming all the chunks are sufficiently above the retrieval threshold this equation describes the probability of chunk *i* being retrieved among those which are being considered (the set *j*):

$$\text{Probability}(i) = \frac{e^{A_i / \sqrt{2}s}}{\sum_j e^{A_j / \sqrt{2}s}}$$

Using that equation for probability of retrieval one can then determine the expected changes in retrieval probability based on the differences in similarity values. Of course, that level of analysis will not always be necessary, but sometimes it's useful to be able to investigate things in that way.

For this model we do not have any similarity values set and thus since there is no simple-fact chunk which has an attribute of pink they all get the maximum penalty of -10 (:mp of 10 times the default mismatch value of -1). If we want the model to have a preference for particular items then we will need to set some similarities and we may want to adjust the :mp value as well. Based on this retrieval request and the initial declarative memory chunks, all we need are the similarities between pink and each of blue, red, and green since those are the only items involved that could be partial-matched to the request. However, if this were a task which required a richer set of information and which may involve requests for any color then we would likely want to set all of the possible similarities, and using a sim-hook function like the 1-hit blackjack model does for similarities between numbers might be a good way to do so. For the purposes of the demonstration model however we will just set one similarity value so that red is considered more similar to pink than pink is to blue or green. That way the model will be most likely to retrieve the chunk f2 when requesting a simple-fact with an attribute of pink, and since we are not concerned with exactly how similar the items are or exactly how much more likely it should be for this model we will just set that to a similarity of -.4 by adding this setting to the model definition:

```
(set-similarities (pink red -.4))
```

and then investigate the effect that has before considering further changes.

Here is the trace of the model run with that addition made:

```
> (run 30)
0.000 PROCEDURAL CONFLICT-RESOLUTION
0.050 PROCEDURAL PRODUCTION-FIRED P1
0.050 PROCEDURAL CLEAR-BUFFER GOAL
0.050 PROCEDURAL CLEAR-BUFFER RETRIEVAL
0.050 GOAL SET-BUFFER-CHUNK GOAL TASK-STATE0
0.050 DECLARATIVE START-RETRIEVAL
0.050 PROCEDURAL CONFLICT-RESOLUTION
12.259 DECLARATIVE RETRIEVED-CHUNK V2
12.259 DECLARATIVE SET-BUFFER-CHUNK RETRIEVAL V2
12.259 PROCEDURAL CONFLICT-RESOLUTION
12.309 PROCEDURAL PRODUCTION-FIRED P2
12.309 PROCEDURAL CLEAR-BUFFER RETRIEVAL
12.309 DECLARATIVE START-RETRIEVAL
12.309 PROCEDURAL CONFLICT-RESOLUTION
26.541 DECLARATIVE RETRIEVED-CHUNK V1
26.541 DECLARATIVE SET-BUFFER-CHUNK RETRIEVAL V1
26.541 PROCEDURAL CONFLICT-RESOLUTION
26.591 PROCEDURAL PRODUCTION-FIRED P3
26.591 PROCEDURAL CLEAR-BUFFER RETRIEVAL
```


26.591	DECLARATIVE	START-RETRIEVAL
26.591	PROCEDURAL	CONFLICT-RESOLUTION
29.931	DECLARATIVE	RETRIEVED-CHUNK F2
29.931	DECLARATIVE	SET-BUFFER-CHUNK RETRIEVAL F2
29.931	PROCEDURAL	CONFLICT-RESOLUTION
29.981	PROCEDURAL	PRODUCTION-FIRED P4
29.981	PROCEDURAL	CLEAR-BUFFER RETRIEVAL
29.981	DECLARATIVE	START-RETRIEVAL
29.981	PROCEDURAL	CONFLICT-RESOLUTION
30.000	-----	Stopped because time limit reached

Now we see that the chunk f2 is retrieved in about 3 seconds for this request. That is sufficient for this demonstration so we will not change anything else to affect that retrieval.

Fourth Retrieval Request

The next retrieval request this model makes is very similar to the previous one, except this time the request is for a simple-fact with an attribute of black:

```
(p p4
  =goal>
    isa task-state
    state r4
  ?retrieval>
    state free
==>
  =goal>
    state r5
  +retrieval>
    isa simple-fact
    attribute black)
```

Like the previous request there is no matching chunk and thus this request will either fail or be satisfied by a partially matched result if there is one above the retrieval threshold. The previous run shows that production firing and here is what we see when we look ahead to determine what the result will be:

```
> (mp-show-queue)
Events in the queue:
261.522  DECLARATIVE      RETRIEVED-CHUNK F1
261.522  PROCEDURAL      CONFLICT-RESOLUTION
```

It will retrieve the chunk f1 after more than a 200 second delay. That delay seems unreasonable thus we need to do something to change that, and there are multiple options available depending on what exactly we want to have happen in the model.

If we want it to retrieve some chunk in a reasonable amount of time, then we will need to do something to either change the retrieval time or activation values. The options for doing that have been discussed in previous sections, and we will just summarize them here. We can change the latency factor parameter to make the retrieval faster. We can increase the activation of the

potential chunks by changing their base-level activation either by giving them a stronger history or setting the base-level constant. We can decrease the penalty for partial matching by setting similarity values or changing the `:mp` parameter. Finally, we can add additional context to the buffers so that spreading activation will increase the chunk's activation.

If we don't mind having the request fail, then we can increase the retrieval threshold so that the model fails to retrieve in a shorter time than that. Since the time for a failure is determined by the setting of the retrieval threshold and the latency factor the settings of those parameters effectively specify the upper bound for how long the model can take to perform any retrieval. However, changing those parameters will affect all of the retrieval requests which the model makes. Another way to handle that would be to use the temporal module in the model so that it can monitor the time that has passed explicitly and then stop waiting once too much time has passed. That allows the model to have a flexible "failure time", but it will typically require the model having additional productions to set up and use the temporal information. Details on using the temporal module are not yet available in the tutorial, but you can find information on using it in the ACT-R reference manual. Another option is available if it's possible for the model to produce the response without completing the retrieval, for example if it can also find the information by searching for it visually. In situations like that, while the retrieval is happening the model can also be engaged in the alternate process of determining the information. If the retrieval completes before the other method then the model can stop and use the retrieved information. If the other method completes first, then the model will not have to wait until the retrieval succeeds or fails. That can work very well in a learning model as long as the result of the alternate process results in strengthening the same declarative information each time. Then, as the activation of that chunk increases the model will shift from always having to do the deliberate process to being able to rely entirely on the retrieved information. That is similar to how the zbrodoff model in unit 4 of the tutorial operated, except that it did not perform the retrieval of the information in parallel with the alternative mechanism since in that case, the other process, counting, also required the use of declarative retrievals and the model can only perform one retrieval at a time.

In this model we would like some chunk to be retrieved and there is no alternative method available for producing a result. Thus, we need to make some adjustment to change the retrieval time. We are again going to avoid using spreading activation. So that leaves us with partial matching, base-level activation, or the latency factor to be adjusted. For partial matching we could adjust the similarities as we did with the previous retrieval, but that doesn't seem as appropriate to do for black and the target colors. We could also adjust the mismatch penalty, which might work well in this situation though changing it will also affect all of the other partially matched retrievals as well. Since we have already set the chunks' base-level histories to something fairly strong, we don't want to adjust that any further. Changing the base-level constant would also affect the base-level activations, but we will again avoid doing that. If we

change the latency factor that is going to affect all of the retrievals which the model makes, and while they might be useful we are not going to do so here. So, among the options available, changing the mismatch penalty is the one that we will investigate further for this retrieval.

The current setting in the model is 10 and with the default maximum dissimilarity value of -1 causes the chunks which mismatch the requested value of black in the attribute slot to have -10 added to their activations. We want the activation of those chunks to increase so that they are retrieved faster, thus we need to decrease the penalty. If we had data to fit that would give us a guide as to how long the retrievals should be taking and suggest a more specific change to make, but since this is just for demonstration purposes we will just set it to something lower and look at the result. If we decrease the :mp value to 2 and run the model again here is what we get:

```
> (run 26.9)
0.000 PROCEDURAL CONFLICT-RESOLUTION
0.050 PROCEDURAL PRODUCTION-FIRED P1
0.050 PROCEDURAL CLEAR-BUFFER GOAL
0.050 PROCEDURAL CLEAR-BUFFER RETRIEVAL
0.050 GOAL SET-BUFFER-CHUNK GOAL TASK-STATE0
0.050 DECLARATIVE START-RETRIEVAL
0.050 PROCEDURAL CONFLICT-RESOLUTION
12.259 DECLARATIVE RETRIEVED-CHUNK V2
12.259 DECLARATIVE SET-BUFFER-CHUNK RETRIEVAL V2
12.259 PROCEDURAL CONFLICT-RESOLUTION
12.309 PROCEDURAL PRODUCTION-FIRED P2
12.309 PROCEDURAL CLEAR-BUFFER RETRIEVAL
12.309 DECLARATIVE START-RETRIEVAL
12.309 PROCEDURAL CONFLICT-RESOLUTION
26.541 DECLARATIVE RETRIEVED-CHUNK V1
26.541 DECLARATIVE SET-BUFFER-CHUNK RETRIEVAL V1
26.541 PROCEDURAL CONFLICT-RESOLUTION
26.591 PROCEDURAL PRODUCTION-FIRED P3
26.591 PROCEDURAL CLEAR-BUFFER RETRIEVAL
26.591 DECLARATIVE START-RETRIEVAL
26.591 PROCEDURAL CONFLICT-RESOLUTION
26.727 DECLARATIVE RETRIEVED-CHUNK F2
26.727 DECLARATIVE SET-BUFFER-CHUNK RETRIEVAL F2
26.727 PROCEDURAL CONFLICT-RESOLUTION
26.777 PROCEDURAL PRODUCTION-FIRED P4
26.777 PROCEDURAL CLEAR-BUFFER RETRIEVAL
26.777 DECLARATIVE START-RETRIEVAL
26.777 PROCEDURAL CONFLICT-RESOLUTION
26.855 DECLARATIVE RETRIEVED-CHUNK F1
26.855 DECLARATIVE SET-BUFFER-CHUNK RETRIEVAL F1
26.855 PROCEDURAL CONFLICT-RESOLUTION
26.900 ----- Stopped because time limit reached
```

The fourth retrieval now completes in less than 200 ms as does the third retrieval which previously took more than 3 seconds. If we look closely at the activation traces we will see that without any similarities set f1 is the chunk chosen here because of noise since the other activation quantities of the potential chunks are very similar to each other, with the only difference being that f2 has a slightly higher base-level activation than f1 and f3 since it has an

extra reference. We are going to consider that sufficient for this request and move on to the next one the model performs.

Fifth Retrieval Request

Up to this point we have avoided the effects of spreading activation on retrievals, but for this request we will investigate issues related to using it. To do that we are going to use chunks with the following types:

```
(chunk-type number representation)
(chunk-type math-fact arg1 arg2 result operator)
(chunk-type context val1 val2 val3 goal)
```

and these initial chunks in declarative memory:

```
(zero isa number representation "0")
(one isa number representation "1")
(two isa number representation "2")
(three isa number representation "3")

(1+1 isa math-fact arg1 one arg2 one result two operator add)
(1+2 isa math-fact arg1 one arg2 two result three operator add)
(1-1 isa math-fact arg1 one arg2 one result zero operator subtract)
(2-1 isa math-fact arg1 two arg2 one result one operator subtract)
(3-2 isa math-fact arg1 three arg2 two result one operator subtract)
```

The first thing to consider when using spreading activation is which buffers the model is going to use as sources for spreading activation. By default, only the goal buffer is considered a source of activation, but typically one also will want to use the imaginal buffer. Doing that will require setting the `:imaginal-activation` parameter to enable the imaginal buffer as a source. For this model we will only be using the imaginal buffer as a source since our goal buffer is only being used to hold state information which is not related to items in declarative memory and turning off the goal buffer as a source will simplify the activation trace information. To do that we will add these parameter settings to the model to turn off the goal buffer as a source of activation and enable the imaginal buffer as a source of activation:

```
(sgp :ga 0 :imaginal-activation 1)
```

The value used in setting the source spread from a buffer is often set at 1, as we have done here, but other values may be used and some researchers have found that adjustments to the source spread parameters can account very well for differences between individuals. For this model we will not adjust that parameter, but you may want to investigate that on your own after working through the demonstrations to see how it affects things.

The only other parameter required for using spreading activation is `:mas` which when set to a number both enables spreading activation and specifies the value of S in the equation for S_{ji}

values. In this model we have set that parameter to a value of 2 initially, but we may need to modify that as we go along.

The next request which the model makes is with this production:

```
(p p5
  =goal>
    isa task-state
    state r5
  =retrieval>
    isa simple-fact
==>
  =goal>
    state r6
  +imaginal>
    isa context
    val1 one
    val2 two
  +retrieval>
    isa math-fact)
```

That production makes both a request to the imaginal buffer to create a chunk of type context and a retrieval request for a math-fact. Running the model with the activation trace enabled produces this output for that retrieval request:

```
> (run 27)
...
26.905    PROCEDURAL          PRODUCTION-FIRED P5
26.905    PROCEDURAL          CLEAR-BUFFER IMAGINAL
26.905    PROCEDURAL          CLEAR-BUFFER RETRIEVAL
26.905    DECLARATIVE         START-RETRIEVAL
Computing activation for chunk 1-1
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references (0.0)
  creation time: 0.0 decay: 0.5  Optimized-learning: T
base-level value: -0.9530089
Total base-level: -0.9530089
Computing activation spreading from buffers
Total spreading activation: 0.0
Computing partial matching component
Total similarity score 0.0
Adding transient noise 0.36828277
Adding permanent noise 0.0
Chunk 1-1 has an activation of: -0.5847261
Chunk 1-1 has the current best activation -0.5847261
Computing activation for chunk 2-1
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references (0.0)
  creation time: 0.0 decay: 0.5  Optimized-learning: T
base-level value: -0.9530089
Total base-level: -0.9530089
Computing activation spreading from buffers
Total spreading activation: 0.0
Computing partial matching component
Total similarity score 0.0
```

```

Adding transient noise -0.17789373
Adding permanent noise 0.0
Chunk 2-1 has an activation of: -1.1309026
Computing activation for chunk 3-2
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references (0.0)
  creation time: 0.0 decay: 0.5 Optimized-learning: T
base-level value: -0.9530089
Total base-level: -0.9530089
Computing activation spreading from buffers
Total spreading activation: 0.0
Computing partial matching component
Total similarity score 0.0
Adding transient noise 0.33106902
Adding permanent noise 0.0
Chunk 3-2 has an activation of: -0.6219399
Computing activation for chunk 1+1
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references (0.0)
  creation time: 0.0 decay: 0.5 Optimized-learning: T
base-level value: -0.9530089
Total base-level: -0.9530089
Computing activation spreading from buffers
Total spreading activation: 0.0
Computing partial matching component
Total similarity score 0.0
Adding transient noise -0.38912883
Adding permanent noise 0.0
Chunk 1+1 has an activation of: -1.3421377
Computing activation for chunk 1+2
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references (0.0)
  creation time: 0.0 decay: 0.5 Optimized-learning: T
base-level value: -0.9530089
Total base-level: -0.9530089
Computing activation spreading from buffers
Total spreading activation: 0.0
Computing partial matching component
Total similarity score 0.0
Adding transient noise 0.0474016
Adding permanent noise 0.0
Chunk 1+2 has an activation of: -0.9056073
Chunk 1-1 with activation -0.5847261 is the best

```

The thing to note there is that there is no spreading activation occurring even though that production made a request to create a chunk in the imaginal buffer. The reason for that is because the sources of activation are determined at the time the request is made, but it takes the imaginal module time to create the chunk. Thus, there is no chunk in the imaginal buffer at the time the retrieval request occurs from which to spread activation. Since we want to see the effects of spreading activation from that chunk we will change the model so that production p5 does not make a retrieval request and then look at the next production, p6, which makes that same request:

```

=goal>
  isa task-state
  state r5
=retrieval>
  isa simple-fact
==>
=goal>
  state r6
+imaginal>
  isa context
  val1 one
  val2 two)

(p p6
  =goal>
    isa task-state
    state r6
  =imaginal>
    isa context
==>
  +retrieval>
    isa math-fact
  =goal>
    state r7
  =imaginal>
    val3 add)

```

Production p6 will not fire until there is a chunk in the imaginal buffer. It then modifies the chunk in the imaginal buffer along with making a retrieval request for a math-fact. Thus, there should now be some activation spreading and here is the activation trace generated from this request:

```

> (run 28)
...
27.155   PROCEDURAL           PRODUCTION-FIRED P6
27.155   PROCEDURAL           CLEAR-BUFFER RETRIEVAL
27.155   DECLARATIVE          START-RETRIEVAL
Computing activation for chunk 1-1
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references (0.0)
  creation time: 0.0 decay: 0.5 Optimized-learning: T
base-level value: -0.95763344
Total base-level: -0.95763344
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CONTEXT0-0
    sources of activation are: (ONE TWO ADD)
    Spreading activation  0.13018736 from source ONE level  0.333 times Sji 0.39056206
    Spreading activation  0.0 from source TWO level  0.333 times Sji 0.0
    Spreading activation  0.0 from source ADD level  0.333 times Sji 0.0
Total spreading activation: 0.13018736
Computing partial matching component
Total similarity score 0.0
Adding transient noise 0.36828277
Adding permanent noise 0.0
Chunk 1-1 has an activation of: -0.45916334
Chunk 1-1 has the current best activation -0.45916334
Computing activation for chunk 2-1
Computing base-level
Starting with blc: 0.0

```

Computing base-level from 1 references (0.0)
creation time: 0.0 decay: 0.5 Optimized-learning: T
base-level value: -0.95763344
Total base-level: -0.95763344
Computing activation spreading from buffers
Spreading 1.0 from buffer IMAGINAL chunk CONTEXT0-0
sources of activation are: (ONE TWO ADD)
Spreading activation 0.13018736 from source ONE level 0.333 times Sji 0.39056206
Spreading activation 0.0694135 from source TWO level 0.333 times Sji 0.20824051
Spreading activation 0.0 from source ADD level 0.333 times Sji 0.0
Total spreading activation: 0.19960088
Computing partial matching component
Total similarity score 0.0
Adding transient noise -0.17789373
Adding permanent noise 0.0
Chunk 2-1 has an activation of: -0.9359263
Computing activation for chunk 3-2
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references (0.0)
creation time: 0.0 decay: 0.5 Optimized-learning: T
base-level value: -0.95763344
Total base-level: -0.95763344
Computing activation spreading from buffers
Spreading 1.0 from buffer IMAGINAL chunk CONTEXT0-0
sources of activation are: (ONE TWO ADD)
Spreading activation -0.10086171 from source ONE level 0.333 times Sji -0.30258512
Spreading activation 0.0694135 from source TWO level 0.333 times Sji 0.20824051
Spreading activation 0.0 from source ADD level 0.333 times Sji 0.0
Total spreading activation: -0.031448208
Computing partial matching component
Total similarity score 0.0
Adding transient noise 0.33106902
Adding permanent noise 0.0
Chunk 3-2 has an activation of: -0.6580126
Computing activation for chunk 1+1
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references (0.0)
creation time: 0.0 decay: 0.5 Optimized-learning: T
base-level value: -0.95763344
Total base-level: -0.95763344
Computing activation spreading from buffers
Spreading 1.0 from buffer IMAGINAL chunk CONTEXT0-0
sources of activation are: (ONE TWO ADD)
Spreading activation 0.13018736 from source ONE level 0.333 times Sji 0.39056206
Spreading activation 0.0694135 from source TWO level 0.333 times Sji 0.20824051
Spreading activation 0.20456855 from source ADD level 0.333 times Sji 0.61370564
Total spreading activation: 0.40416944
Computing partial matching component
Total similarity score 0.0
Adding transient noise -0.38912883
Adding permanent noise 0.0
Chunk 1+1 has an activation of: -0.94259286
Computing activation for chunk 1+2
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references (0.0)
creation time: 0.0 decay: 0.5 Optimized-learning: T
base-level value: -0.95763344
Total base-level: -0.95763344
Computing activation spreading from buffers
Spreading 1.0 from buffer IMAGINAL chunk CONTEXT0-0


```

sources of activation are: (ONE TWO ADD)
Spreading activation -0.10086171 from source ONE level 0.333 times Sji -0.30258512
Spreading activation 0.0694135 from source TWO level 0.333 times Sji 0.20824051
Spreading activation 0.20456855 from source ADD level 0.333 times Sji 0.61370564
Total spreading activation: 0.17312035
Computing partial matching component
Total similarity score 0.0
Adding transient noise 0.0474016
Adding permanent noise 0.0
Chunk 1+2 has an activation of: -0.7371115
Chunk 1-1 with activation -0.45916334 is the best

```

Before looking at the final result of the request we will first look at what the sources of activation are. From the activation trace we see that it lists these three chunks as sources: one, two, and add. The thing to note is that the chunk add being in a slot of the imaginal buffer chunk was the result of a modification to the chunk made as an action in that production. Modifications made directly by the production will always take effect before the retrieval request starts. If we enable the high detail trace and run it again that can be seen in this sequence of events following the production firing:

28.391	PROCEDURAL	PRODUCTION-FIRED P6
28.391	PROCEDURAL	MOD-BUFFER-CHUNK GOAL
28.391	PROCEDURAL	MOD-BUFFER-CHUNK IMAGINAL
28.391	PROCEDURAL	MODULE-REQUEST RETRIEVAL
28.391	PROCEDURAL	CLEAR-BUFFER RETRIEVAL
28.391	DECLARATIVE	START-RETRIEVAL

The mod-buffer-chunk actions for the goal and imaginal buffer occur before the declarative module starts the retrieval. Also worth noting is that the clearing of buffers by the production will also precede the start of the declarative retrieval. Using the high detail trace can be helpful to determine why items are or are not sources when looking at other situations because to be a source the change must occur prior to the start-retrieval action of the declarative module.

Looking at the result of that retrieval we see that it retrieved this chunk:

```

1-1
ISA MATH-FACT
ARG1 ONE
ARG2 ONE
RESULT ZERO
OPERATOR SUBTRACT

```

which seems unusual given that we have sources of one, two, and add and there's another chunk which looks like this that seems like it should be getting more spreading activation:

```

1+2
ISA MATH-FACT
ARG1 ONE
ARG2 TWO
RESULT THREE
OPERATOR ADD

```

We will look at the activation trace for those two items to see what causes the difference, and here are the relevant traces:

```
Computing activation for chunk 1-1
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references (0.0)
  creation time: 0.0 decay: 0.5 Optimized-learning: T
base-level value: -0.95763344
Total base-level: -0.95763344
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CONTEXT0-0
    sources of activation are: (ONE TWO ADD)
    Spreading activation 0.16530752 from source ONE level 0.333 times Sji 0.49592257
    Spreading activation 0.0 from source TWO level 0.333 times Sji 0.0
    Spreading activation 0.0 from source ADD level 0.333 times Sji 0.0
Total spreading activation: 0.16530752
Computing partial matching component
Total similarity score 0.0
Adding transient noise 0.36828277
Adding permanent noise 0.0
Chunk 1-1 has an activation of: -0.42404315
Chunk 1-1 has the current best activation -0.42404315

Computing activation for chunk 1+2
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references (0.0)
  creation time: 0.0 decay: 0.5 Optimized-learning: T
base-level value: -0.95763344
Total base-level: -0.95763344
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CONTEXT0-0
    sources of activation are: (ONE TWO ADD)
    Spreading activation -0.06574154 from source ONE level 0.333 times Sji -0.1972246
    Spreading activation 0.13018736 from source TWO level 0.333 times Sji 0.39056206
    Spreading activation 0.30046257 from source ADD level 0.333 times Sji 0.9013877
Total spreading activation: 0.3649084
Computing partial matching component
Total similarity score 0.0
Adding transient noise 0.0474016
Adding permanent noise 0.0
Chunk 1+2 has an activation of: -0.54532343
```

Looking at those traces we see that those two chunks have the same base-level activation and chunk 1+2 does have a higher total spreading activation value. Chunk 1-1 gets a greater boost from noise than 1+2, so the first though might be that it's just an issue with noise. However, a closer look at the spreading activation calculations raises some interesting questions:

```
Computing activation for chunk 1-1
...
  Spreading 1.0 from buffer IMAGINAL chunk CONTEXT0-0
    sources of activation are: (ONE TWO ADD)
    Spreading activation 0.16530752 from source ONE level 0.333 times Sji 0.49592257
    Spreading activation 0.0 from source TWO level 0.333 times Sji 0.0
    Spreading activation 0.0 from source ADD level 0.333 times Sji 0.0
```

Total spreading activation: 0.16530752

Computing activation for chunk 1+2

...

```
Spreading 1.0 from buffer IMAGINAL chunk CONTEXT0-0
sources of activation are: (ONE TWO ADD)
Spreading activation -0.06574154 from source ONE level 0.333 times Sji -0.1972246
Spreading activation 0.13018736 from source TWO level 0.333 times Sji 0.39056206
Spreading activation 0.30046257 from source ADD level 0.333 times Sji 0.9013877
Total spreading activation: 0.3649084
```

The first is why does the S_{ji} from one to 1-1 differ from the S_{ji} from one to 1+2, and another is why is the S_{ji} from one to 1+2 negative?

The answer to the first issue has to do with how S_{ji} s are computed when there are multiple references within a chunk. The equation for S_{ji} from the main unit 5 text:

$$S_{ji} = S - \ln(fan_j)$$

is a simplification of the full calculation which is true when there's a single link between chunks j and i , but in this case the chunk one occurs in two different slots of the chunk 1-1. The more general form of the equation for S_{ji} uses the value fan_{ji} instead of fan_j where fan_{ji} is defined as:

$$fan_{ji} = \frac{1 + slots_j}{slotsof_{ji}}$$

slots_j: the number of slots in which j is the value across all chunks in declarative memory

slotsof_{ji}: the number of slots in chunk i which have j as the value (plus 1 if chunk i is chunk j)

In this case, j is the chunk one and i is the chunk 1-1. Chunk one is a value in 8 slots of the chunks in declarative memory, so that is $slots_j$, and it occurs in two slots of chunk 1-1, so that is the value for $slotsof_{ji}$. Combining that with the value of 2 for S as was set in the model we get:

$$S_{ji} = 2 - \ln\left(\frac{1+8}{2}\right) = 2 - 1.5040774 = 0.49592257$$

which is what we see in the trace. For the S_{ji} between the chunk one and the chunk 1+2 the equation is:

$$S_{ji} = 2 - \ln\left(\frac{1+8}{1}\right) = 2 - 2.1972246 = -0.1972246$$

which explains the negative association.

In general, negative associations are not recommended because that is then inhibiting the retrieval of related information instead of supporting it. The easy way to fix that is to make sure that the S value is set high enough to avoid the negative value. Occasionally situations occur where one may want that inhibitory behavior, and in those situations it's still advised to set S high enough that items don't automatically get negative S_{ji} values. Instead, the recommendation is to set those desired negative associations explicitly with the add-sji command.

To fix the issue with negative associations in the model we will set our S value to 4 which should be sufficient to keep all S_{ji} values positive (as long as fan_{ji} is less than 54 it will be positive). When we run the model after making that change we see that the model does retrieve the chunk we expected it to:

27.155	PROCEDURAL	PRODUCTION-FIRED P6
27.155	PROCEDURAL	MOD-BUFFER-CHUNK GOAL
27.155	PROCEDURAL	MOD-BUFFER-CHUNK IMAGINAL
27.155	PROCEDURAL	MODULE-REQUEST RETRIEVAL
27.155	PROCEDURAL	CLEAR-BUFFER RETRIEVAL
27.155	DECLARATIVE	START-RETRIEVAL
27.155	PROCEDURAL	CONFLICT-RESOLUTION
27.381	DECLARATIVE	RETRIEVED-CHUNK 1+2
27.381	DECLARATIVE	SET-BUFFER-CHUNK RETRIEVAL 1+2

However, to be sure things are doing what we expect we should look at the activation trace to make sure, and here is the trace with the base-level and similarity sections removed since those are identical among these chunks:

```
Computing activation for chunk 1-1
...
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CONTEXT0-0
    sources of activation are: (ONE TWO ADD)
    Spreading activation 0.796854 from source ONE level 0.333 times Sji 2.390562
    Spreading activation 0.0 from source TWO level 0.333 times Sji 0.0
    Spreading activation 0.0 from source ADD level 0.333 times Sji 0.0
  Total spreading activation: 0.796854
...
Adding transient noise 0.36828277
Adding permanent noise 0.0
Chunk 1-1 has an activation of: 0.20750335
Chunk 1-1 has the current best activation 0.20750335
```

```

Computing activation for chunk 2-1
...
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CONTEXT0-0
    sources of activation are: (ONE TWO ADD)
    Spreading activation  0.796854 from source ONE level  0.333 times Sji 2.390562
    Spreading activation  0.73608017 from source TWO level  0.333 times Sji 2.2082405
    Spreading activation  0.0 from source ADD level  0.333 times Sji 0.0
Total spreading activation: 1.5329342
...
Adding transient noise -0.17789373
Adding permanent noise 0.0
Chunk 2-1 has an activation of: 0.39740703
Chunk 2-1 is now the current best with activation 0.39740703

Computing activation for chunk 3-2
...
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CONTEXT0-0
    sources of activation are: (ONE TWO ADD)
    Spreading activation  0.56580496 from source ONE level  0.333 times Sji 1.6974149
    Spreading activation  0.73608017 from source TWO level  0.333 times Sji 2.2082405
    Spreading activation  0.0 from source ADD level  0.333 times Sji 0.0
Total spreading activation: 1.3018851
...
Adding transient noise 0.33106902
Adding permanent noise 0.0
Chunk 3-2 has an activation of: 0.67532074
Chunk 3-2 is now the current best with activation 0.67532074

Computing activation for chunk 1+1
...
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CONTEXT0-0
    sources of activation are: (ONE TWO ADD)
    Spreading activation  0.796854 from source ONE level  0.333 times Sji 2.390562
    Spreading activation  0.73608017 from source TWO level  0.333 times Sji 2.2082405
    Spreading activation  0.87123525 from source ADD level  0.333 times Sji 2.6137056
Total spreading activation: 2.4041696
...
Adding transient noise -0.38912883
Adding permanent noise 0.0
Chunk 1+1 has an activation of: 1.0574073
Chunk 1+1 is now the current best with activation 1.0574073

Computing activation for chunk 1+2
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CONTEXT0-0
    sources of activation are: (ONE TWO ADD)
    Spreading activation  0.56580496 from source ONE level  0.333 times Sji 1.6974149
    Spreading activation  0.73608017 from source TWO level  0.333 times Sji 2.2082405
    Spreading activation  0.87123525 from source ADD level  0.333 times Sji 2.6137056
Total spreading activation: 2.1731205
...
Adding transient noise 0.0474016
Adding permanent noise 0.0
Chunk 1+2 has an activation of: 1.2628886
Chunk 1+2 is now the current best with activation 1.2628886

```

Looking over the spreading activation values shows that in fact the chunk 1+1 gets more spreading activation than chunk 1+2 and it is only because of noise that we retrieved 1+2 this time. The reason for that is because the chunk 1+1 is also receiving activation spread from each of the sources since it also contains each of those chunks in some slot, and because it has two occurrences of the chunk one it has a greater S_{ji} from one than the chunk 1+2 does.

This highlights a big distinction between spreading activation and partial matching. Spreading activation is a bottom-up mechanism which increases the activation of chunks relative to how well they match the current context without regard for the specific structure of that information in the target chunks. Whereas partial matching is a top-down process which penalizes those chunks which do not match the specific pattern provided in the request. In models of simple tasks often only one effect or the other is desired and to keep things simple only that particular mechanism is enabled, as was the case for the tutorial unit models, but in more complex models both effects may be desirable in which case one has to be more careful about both maintaining an appropriate context and making appropriate requests to achieve the desired results.

If we want this model to be relatively certain of retrieving the fact associated with adding one and two we will need to add that pattern information into the request. We will not make that change to the model as part of the demonstration, but you should feel free to try that and see how the activations change. You may also want to try changing the similarities between the number chunks to see how that affects things as well because the current model does not set any similarities between the number chunks.

Last two productions

The final issue we will look at does not involve a retrieval. Instead we will look at a potential issue which can arise when creating chunks that will be merged into declarative memory. In most situations chunks will merge and strengthen existing declarative chunks as one would expect, but there is a situation which can sometimes occur which is worth discussing here because it can be confusing. The issue can arise when a model creates a chunk which has slot values that are chunks which are not in declarative memory (typically because they reference a chunk currently in a buffer) and then merges that chunk into declarative memory.

For the example we will investigate what happens when these two productions fire:

```
(p p7
  =goal>
    isa task-state
    state r7
  =imaginal>
    isa context
  =retrieval>
    isa math-fact
```

```

==>
=goal>
  state r8
=imaginal>
  goal =goal)

(p p8
=goal>
  isa task-state
  state r8
=imaginal>
  isa context
==>
+goal>
  isa task-state
-imaginal>)

```

in the context of these chunks existing in declarative memory:

```

G1
  ISA TASK-STATE
  STATE R8

OLD-CONTEXT
  ISA CONTEXT
  VAL1 ONE
  VAL2 TWO
  VAL3 ADD
  GOAL G1

```

Before running the model we will look at what we might expect to happen. Production p7 waits for the previous retrieval to complete and then modifies the chunks in the goal and imaginal buffers. The goal chunk is modified such that it now looks just like g1 and the imaginal chunk has that current goal buffer chunk placed into its goal slot. Production p8 fires next since the goal buffer state matches and then it performs two explicit actions. It makes a request to the goal module to create a new task-state chunk, which will implicitly clear the current chunk from the goal buffer, and it clears the chunk from the imaginal buffer. What we might expect to happen here is that the goal buffer's chunk will merge with chunk g1 and then the imaginal buffer's chunk will merge with the chunk old-context.

After running the model however here is what we see in declarative memory with respect to task-state and context chunks (found using the sdm command here but using the filter in the declarative viewer could also be used):

```

> (sdm isa task-state)
G1
  ISA TASK-STATE
  STATE R8

(G1)
> (sdm isa context)
OLD-CONTEXT

```

```

ISA CONTEXT
VAL1  ONE
VAL2  TWO
VAL3  ADD
GOAL  G1

CONTEXT0-0
ISA CONTEXT
VAL1  ONE
VAL2  TWO
VAL3  ADD
GOAL  G1

(OLD-CONTEXT CONTEXT0-0)

```

We see that there is only one task-state chunk as we expected, but there are two apparently identical context chunks. One way to see why that happens would be to step through the actions which occur as a result of that production firing and inspect things carefully after each event. If you would like to do that you can do so, but here we will just inspect the actions in the trace and explain the outcome.

Before doing that, there is something else which we can do that might make things clearer. If we turn off the `:ncnar` parameter the model will not automatically normalize the chunk names when chunks are merged and that might also help to see what has happened. Thus, before running the model again we should add that to the `sgp` call in the model:

```
(sgp ... :ncnar nil)
```

Now, after we run the model this is what is shown for the chunks of type context in declarative memory:

```

> (sdm isa context)
OLD-CONTEXT
ISA CONTEXT
VAL1  ONE
VAL2  TWO
VAL3  ADD
GOAL  G1

CONTEXT0-0
ISA CONTEXT
VAL1  ONE
VAL2  TWO
VAL3  ADD
GOAL  TASK-STATE0-0

```

An important thing to note is that while the `context0-0` chunk now looks different than it did after the previous run when `:ncnar` was enabled there is no difference between them in terms of the model's operation because changing the `:ncnar` parameter only changes how the information is displayed to the modeler.

Here are the events from the trace, which may suggest what has happened, but we will still go over the details:

27.481	PROCEDURAL	PRODUCTION-FIRED P8
27.481	PROCEDURAL	MODULE-REQUEST GOAL
27.481	PROCEDURAL	CLEAR-BUFFER IMAGINAL
27.481	PROCEDURAL	CLEAR-BUFFER GOAL

While all of those events are listed as occurring at the same time, as we've seen using the stepper, each is executed individually in the order that they are shown. Thus, first the production fires, then the request is made to the goal buffer, then the imaginal buffer gets cleared, and finally the goal buffer gets cleared.

The important question is then how does declarative memory handle merging chunks? The answer is that it only attempts to merge chunks immediately upon their being cleared from a buffer, and it will only merge chunks if all of their contents are perfect matches. When the slot values are chunk names a perfect match means that they must refer to the same chunk (note however that that doesn't mean that the slot values must have the same chunk name because merged chunks can still be referenced by either name).

Thus, when the imaginal buffer gets cleared the chunk in it looks like this:

```
CONTEXT0-0
ISA CONTEXT
VAL1 ONE
VAL2 TWO
VAL3 ADD
GOAL TASK-STATE0-0
```

At that time the chunk task-state0-0 is still in the goal buffer and has not been merged with chunk g1 in declarative memory. Because of that the chunk context0-0 is not a perfect match to the chunk old-context which is in declarative memory since the value of their goal slots, task-state0-0 and g1, are different chunks. That means context0-0 must be added to declarative memory as a new chunk. Then, the goal buffer gets cleared. Because chunk task-state0-0 is a perfect match to the chunk g1 those two chunks are then merged. The merging of those task-state chunks does not make the declarative module retroactively merge the chunks old-context and context0-0. Thus, declarative memory still has two context chunks; one with a value of g1 in the goal slot and one with a value of task-state0-0 in the goal slot, but both of those values now reference a single chunk (that is why when :ncnar is turned on we see the same value, g1, displayed for both).

That may seem like a problem with how merging works, but there are good reasons for having it work sequentially like that. One of those reasons is that it allows the modeler to control what

happens – sometimes one might want separate chunks instead of having them merged. If we do not want separate chunks, then we have to ensure that all the chunks in the slots of the chunk we want to merge with a chunk in declarative memory are merged into declarative memory first (in this case the chunk in the goal buffer must be merged into declarative memory before the chunk in the imaginal buffer since that goal buffer chunk is in a slot of the chunk in the imaginal buffer). If we want that to happen within a single production, then this becomes one of the very rare situations where controlling the order in which a production's actions occur matters.

Generally, the order in which a production performs its actions does not matter since they are all happening at the same time and there are usually no interactions among them. However, since the simulation has to perform the actions sequentially, in situations like this one the modeler may need to make sure some things happen in a particular order, but the modeler cannot arbitrarily order a production's actions. A production will always perform its actions in the following order: all user actions (!eval!, !bind!, and !output!), all buffer modifications, all requests, then all buffer clearing actions. Within a particular type of action it will perform the explicit specified actions in the order provided in the production followed by any implicit actions of that type (like clearing the buffer due to strict harvesting or as a result of a request) in no particular order. Thus, if we want the goal buffer to be cleared prior to the imaginal buffer we will have to explicitly perform that action in the production instead of letting it happen implicitly, and it will have to be placed before the imaginal buffer clearing.

Here is a modified version of p8 which adds an explicit clearing of the goal buffer before the clearing of the imaginal buffer:

```
(p p8
  =goal>
    isa task-state
    state r8
  =imaginal>
    isa context
==>
  +goal>
    isa task-state
  -goal>
  -imaginal>)
```

When we run the model after saving that change and reloading we get the following result for chunks of type context in declarative memory:

```
> (sdm isa context)
OLD-CONTEXT
ISA CONTEXT
VAL1  ONE
VAL2  TWO
VAL3  ADD
GOAL  G1
```

(OLD-CONTEXT)

which shows that there is only one chunk now thus the imaginal chunk has been merged with the old-context chunk. If we wanted to investigate further we could make sure that that chunk has two references by looking at the details in the declarative viewer or by using the sdp command to check its parameters, and if we do so we find that it does have a value of two for its reference-count:

```
> (sdp old-context)
Declarative parameters for chunk OLD-CONTEXT:
:Activation -0.746
:Permanent-Noise 0.000
:Base-Level -0.270
:Reference-Count 2.000
:Creation-Time 0.000
:Source-Spread 0.000
:Sjis ((OLD-CONTEXT . 4.0) (ONE . 1.6974149) (TWO . 2.2082405) (ADD . 2.6137056)
      (G1 . 3.3068528))
:Similarities ((OLD-CONTEXT . 0.0))
(OLD-CONTEXT)
```