This document describes the blending module for ACT-R 6.

The slides in the blending.pdf file describe the original ACT-R 4 version of the mechanism. The theory of this module is basically same, but the implementation is different in ACT-R 6. This document and the example blending-test models show the details of using it in ACT-R 6.

To use the module move the blending.lisp file into the modules directory of the ACT-R source file distribution before loading the ACT-R 6 loader or load that file explicitly after loading the ACT-R 6 loader.

Once the module is added to the system it will add another buffer called blending for a model to use. There is no switch needed to activate blending once the module is loaded.

There is also a history tool for use with the ACT-R environment if desired. To use that put the blending-history-tool.lisp file into the other-files directory and the 77b-blending-history.tcl file into the environment/GUI/dialogs directory. The blending history tool works basically the same as the retrieval history tool which is described in the Environment's manual.

The blending module assumes that the default declarative module is also loaded. If that module has been modified or removed from the system then the blending module may not work correctly.

The blending module has several parameters:

:blt - the blending trace.

If this parameter is set to t then a detailed trace of the blending process will be output in the model's trace.

:sblt - save blending trace

If this parameter is set to t then the details of the blending trace will be saved while the model runs and can be displayed after by calling print-blending-trace with the time of the blended request for which the trace is desired.

:tmp - the temperature.

If set to a number this value will be used as the temperature in the chunk probability equation. If it is set to nil then the temperature will be the model's :ans value times the square root of 2.

:min-bl - mimimum base-level

The minimum base-level activation a chunk in DM must have to be considered in the matching-set. Note that because it is only a test of base-level activation it does not take into account any context effects. It doesn't have any theoretical basis, but may be useful for performance purposes if a model generates a lot of chunks over time and only the recent and/or very strong ones are necessary for blending purposes.

:value->mag - a mapping function.

If :value->mag is set to a function, then that function will be used to convert all blending slot values into a magnitude to use for blending (see details below).

:mag->value - a mapping function.

If :mag->value is set to a function, then that function will be used to convert magnitudes back into values when the :value-mag function has returned magnitude values (see details below).

:blend-all-slots

If :blend-all-slots is nil, which is the default, then the fixed values specified in the request will occur explicitly in the corresponding slots of the resulting chunk. If :blend-all-slots is set to t then all of the slots of the resulting chunk will be determined through the blending process. For this parameter to be useful the declarative partial matching must also be turned on (:mp set to non-nil) otherwise only chunks which match the request explicitly will be considered and there will be nothing to blend for those slots.

:blending-request-hook

Can be specified as a function to call when there is a blending request. That function should take one parameter which will be the chunk-spec of the request. The return value of the function is ignored. Each setting adds a new function and the "current value" of the parameter is a list of all functions that are set. Setting a value of nil clears all the functions from the hook.

:blending-result-hook

Can be specified as a function to call when a blending request completes. That function should take one parameter which will be either the chunk saved in the buffer or nil if the request failed or was terminated by a new request. The return value of the function is ignored. Note that since the chunk may not be put into the buffer until some later time this function may not

be called at the same time as the request hook function. Each setting adds a new function and the "current value" of the parameter is a list of all functions that are set. Setting a value of nil clears all the functions from the hook.

:blending-set-hook

Can be specified as a function to call when there is a blending request. That function should take one parameter which will be the list of chunks matched to be blended. The return value of the function is ignored. Each setting adds a new function and the "current value" of the parameter is a list of all functions that are set. Setting a value of nil clears all the functions from the hook.

The blending module only responds to the default queries for state {free, busy, and error}. The module will be busy between the time a request is received and the time that a chunk is placed into the buffer in response to the request or until it signals an error due to failure. If an error is set it will remain set until the next request is received by the module.

The blending buffer takes requests the same way the retrieval buffer does and it will attempt to find a chunk that matches the request among the chunks in the model's declarative memory.

It differs in that the chunk returned will have "blended" values in its slots. How those blended values are computed is described below.

The module can only handle one request at a time. If a new request is received while the module is busy it will cancel the previous request, process the new request and print a warning in the trace.

There are two request parameters allowed in a request :ignore-slots and :do-not-generalize.

If :ignore-slots is provided it must specify a list of slot names. Those slots will not get a value in the result chunk i.e. they will have a value of nil. When using that mechanism one needs to be careful because there could be issues if that result chunk is allowed to enter declarative memory. One issue is that since it will likely become another potential target to be blended the empty slots could affect the results of later blended retrievals. Another issue is that with lots of empty slots it may be more likely to be merged with other blended results. That could lead to issues with the blended results because of higher activation of those merged results.

If :do-not-generalize is provided it must specify a list of slot names. Those slots in the resulting chunk will still be computed via the blending mechanism, but it will not use method c if the values to be blended are all chunks (as described later) for computing the value. Instead it will use method d for that slot. This may be necessary when one wants to use chunks for the slot contents and doesn't want blending to potentially consider all other chunks as values.

Here is the description of how the resulting chunk to a blending request is created.

The request provided is used to find the set of chunks which match the request in the same way that a retrieval request does given the declarative module's current parameter values. That set of chunks does not take into account the activation of those chunks i.e. chunks in that set may have activations below the retrieval threshold. We will call that set of chunks the matching set and use MS to refer to it in the description below.

For each chunk i in MS its activation is computed based on the current settings of the declarative module (including noise), which will be called A(i). Those activations and the temperature value for the blending module are used to compute the probability of recall for the chunks in MS using the Boltzmann equation (shown in blending.pdf). The probability of retrieving chunk i from MS will be called p(i).

The chunk that results from the blending request will be of the chunk-type specified in the request. The contents of the slots of that chunk will be set as follows:

If the :blend-all-slots parameter is nil and the slot has an explicit value in the request then that explicit value will occur in that slot of the resulting chunk. An explicit value is a value requested with an equality test. If a slot is requested with both equality and non-equal tests the explicit value from the equality test will be used.

If the :ignore-slots request parameter is provided all of the slots in that list will have a value of nil in the resulting chunk. For all other slots (including those with an explicit value if the :blend-all-slots parameter is t) the following process is performed to determine the value for that slot (referenced by the index k below) in the resulting chunk:

A list of potential values is created by taking the value from slot k for each of the chunks in MS. Those values will be referred to as: v(k,i) - the value of the slot k in chunk i from from MS.

For each v(k,i) it calls the value->mag function of the module with

v(k,i) as the only parameter. The return value of that function
is the magnitude to use for blending and will be referred to as: m(k,i).
The default value->mag function is identity and thus v(k,i) = m(k,i)
by default.

Based on the set of m(k,i) values one of the following methods
is used to create the blended result for slot k, called R(k):

a) all the m(k,i) values are numbers:

Then R(k) is the sum over all chunks i in MS of p(i)*m(k,i).

b) all the m(k,i) values are nil:

Then R(k) is nil.

c) all the m(k,i) values are chunks or nil and the slot k is not
in the :do-not-generalize list specified with the request

It determines a set of potential value chunks based on
the chunk-types of the non-nil m(k,i) values. That set will be
called PV(k) and created as follows:

i) all of the non-nil m(k,i) values are of the same chunk-type or
have a common parent chunk-type

PV(k) is the set of all of the chunks of that common chunk-type
in the model's DM.

ii) there is no common parent chunk-type

PV(k) is the set of all the chunks in the model's DM.

For each chunk j in PV(k) a value B(j) is computed as the
sum over the chunks i in MS of p(i) times the square of the
similarity between chunk j and m(k,i).

R(k) is then the chunk j for which the B(j) value is minimum.

d) the m(k,i) values are neither all numbers nor all chunks, or
the slot k is in the :do-not-generalize list specified in the request

A set of potential values, PV(k), is created which contains all
of the m(k,i) values.

For each item j in PV(k) a value B(j) is computed as the
sum over the chunks i in MS of p(i) times the square of the
similarity between j and m(k,i).

Note: Since the items in PV(k) may not be chunks this assumes that
there is a similarity function set for the declarative module
otherwise all non-equal items will have maximum dissimilarity
values.

R(k) is then the value j for which the B(j) value is minimum.

If there is no mag->value function specified for the module or the value->mag function returned values such that m(k,i) = v(k,i) for all i then the value for the slot k will be the value R(k).

If there is a mag->value function and there exists an i such that m(k,i) != v(k,i) then the value for the slot k will be the result of calling the mag->value function with two parameters. The first will be the value R(k). The second will be the common chunk-type from case c above if there is one, otherwise it will be nil.

After all slots of the resulting chunk have been computed the module determines whether or not the chunk can be successfully created by blending. To determine that, a match score, M, is computed as the log of the sum over the chunks i in MS of e to the power A(i). If M is greater than or equal to the retrieval threshold of the declarative module then the created chunk is placed into the blending buffer with a latency computed in the same way the declarative module computes retrieval latency using M as the activation of the chunk. If M is below the retrieval threshold then the module will fail to produce the chunk and an error will be signaled after a latency based on the retrieval threshold passes.