ACT-R 4.0

A Users Manual

June 1998

Christian Lebiere
cl+@cmu.edu

Table Of Contents

Section 0 Introduction

The following conventions will be used throughout this document: Items appearing in **bold** are to be entered verbatim; items appearing in *italics* take user-supplied values such as chunk, slot or production names; items enclosed in **{**curly braces**}** are optional; * indicates that any number of items may be supplied; **|** indicates a choice between several options. **::=** indicates that the item on the left of that symbol is of the form given by the expression on the right. Note that arguments to commands *must not* be quoted (i.e., preceded by a "'" mark) as is standard for most LISP function calls. None of the commands are case-sensitive. Commands and sections of this document that are frequently used or are likely to be especially useful to beginners are indicated by the symbol "⇐".

ACT-R 4.0 (hereafter referred to as ACT-R, unless otherwise specified) was written in Macintosh Common LISP. However, it strictly adheres to the standard put forth in the second edition of "*Common Lisp: The Language*" by Guy Steele Jr. and should run without any modifications in any complete Common Lisp implementation. If a compiler is not available in your implementation, you should set the variable ***compile-eval-calls*** to NIL when it is defined at the top of the code file[1]. This manual is a reference for the ACT-R implementation. It is not meant to be a treatise, a tutorial or a textbook on ACT-R. A good understanding of the first four chapters of Anderson & Lebiere's upcoming 'The Atomic Components of Thougth' (ACT in short) are required before experimenting with ACT-R.

The **ACT-R folder** should contain:

- **ACT-R 4.0**: the source code, which can be loaded directly or first compiled.
- **Load ACT-R 4.0** (optional): a compiled version of the code for fast loading.
- **Examples**: a directory containing a number of ACT-R models.
- **Manual**: this manual, written in Microsoft Word 6.0.
- **Patches**: a directory holding the successive releases of ACT-R 4.0 and their release notes.

Feedback on this material is welcome. Please send your suggestions, remarks, bug reports and other communications to cl+@cmu.edu.

[1] The value of NIL for ***compile-eval-calls*** has been made the default in ACT-R 4.0.2. It makes running ACT-R in Allegro CL 5.0 much more efficient and makes little difference in MCL. Your mileage may vary however depending upon your model and Common Lisp implementation. If ACT-R seems unduly slow, you might want to change the setting, recompile and reload ACT-R, then benchmark your model using the Lisp command **time**.

Section 1 Commands

Command List

The following commands are available to define an ACT-R model and control the ACT-R production system. The command **Help** without any argument (i.e. **(Help)**) will print the list of commands, and if given one or more command(s) as argument will display a short description of the command(s) (e.g. **(Help Help)** will print something like "HELP: Outputs a short description of one or more ACT-R command(s), or the full list of commands if none is supplied.").

(Activation-Sources)
Displays the current activation sources with their level. Returns the list of sources.

(Actr-Time {*delay*}**)**
Returns the current ACT-R time and, if *delay* is specified, adds it to the current time.

(Add-IA {**(***chunkj chunki sji***)**}***)**
Adds inter-associativity (IA) values between chunks. *Sji* is defined as the new IA value from *chunkj* to *chunki*. Returns the list of added IAs.

(Add-DM {**(***chunkname* **isa** *chunktype* {*slot value*}***)**}***)**
Adds chunks to declarative memory. The new chunk *chunkname* is of chunk type *chunktype*, with each *slot* being initialized to *value*. Returns the list of added chunks.

(Chunk-Slot-Value *chunk slot* **)**
Returns the value of *slot* of *chunk*.

(Chunk-Type {*type* | **(***type* **(:include** *supertype***))**} {*slot* | **(***slot value***)**}* **)**
Defines a new chunk type, which may include a *supertype* from which *type* inherits its slots, and a number of *slot*, each with an optional initial *value*. If no argument is passed to **Chunk-Type**, then a list of the existing types is printed. Return the chunk type defined or the list of all chunk types if no argument is given.

(Clear-All {*save-model*}**)**
Clears everything from the ACT-R state, including chunk types, chunks, productions, global variables, etc. A call to **Clear-All** ***MUST*** be put at the top of each model file. If the optional argument *save-model* is t or is not specified, both the name of the file and its contents after the call to **Clear-All** are saved for use by the commands **reload** and **reset** respectively. If the argument is nil, neither is saved and neither command can be used. If the argument is a value other than t or nil (e.g. **compiled**), then only the file name is saved, not its contents, and **reset** becomes equivalent to **reload**. This setting can be useful if the model file must be compiled for efficiency purposes, making its contents unsavable.

(Clear-DM)
Clears all chunks from declarative memory. It is especially strongly recommended to use **Clear-All** or **Reset** instead for resetting purposes.

(Clear-Goal-Stack)
Clears the goal stack by popping all goals and restoring the top goal as the focus.

(Clear-Productions)
Clears all productions from production memory. It is strongly recommended to use **Clear-All** or **Reset** instead for resetting purposes.

(Close-Output)
Closes the current output stream file established by a call to **Output-Stream**.

(Close-Trace)
Closes the current trace stream file established by a call to **Trace-Stream**.

(Copy-Chunk {*chunk*}***)**
Makes a new copy of each *chunk* and returns the list of copies.

(Delete-Chunk {*chunk*}***)**
Deletes chunks from declarative memory. Do not use if *chunk* is still used in other chunks or productions. Returns the list of deleted chunks.

(DM {*chunk* }***)**
Displays the specified chunks, or the entire contents of declarative memory if none is supplied. For each *chunk*, its name is printed followed by its current activation, then its type and slot-value pairs. The current focus is displayed preceded by two asterisks ("**"). Returns the list of chunks printed.

(Focus-on-Goal *chunk***)**
Replaces the focus with *chunk* , similarly to the **!focus-on!** production command. Returns the new focus.

(Get-Base-Level {*chunk*}***)**
Returns base level activation of chunks. Returns the list of base levels.

(Get-Name {*chunk-or-production*}***)**
Returns the list of names of chunks and/or productions. This command is now outdated. See Important Update Note at the end of next subsection.

(Goal-Focus {*chunk*}***)** ⇐
Sets the focus to *chunk* , or prints the current focus if none is supplied. If more than one *chunk* is specified, then the focus is set to the first one and the other ones are kept in a list for use by **Run-Many**. Returns the new focus.

(Goal-Stack)
Displays the focus, then the current contents of the goal stack from top to bottom. Each goal is followed by its goal value **G**. Returns the goal stack, i.e. a list of the focus followed by each goal on the stack in descending order.

(Help { *command* }***)**
Displays a short description of *command*, or prints the whole list of commands if none is given.

(IA *chunkj chunki***)**
Displays the IA value between *chunkj* and *chunki* and return the value.

(Load-Model *file* {*directory*}**)**
Loads the model *file* in *directory* . If *directory* is not supplied, then *file* is considered relative to the directory of the file in which the command appears, if applicable.

(Mod-Chunk *chunk* {*slot value*}***)**
Modifies chunk by setting each *slot* to *value*. Returns the modified chunk.

(Mod-Focus {*slot value*}***)**
Same as **Mod-Chunk**, with the chunk set to the focus. Returns the modified focus.

(New-Name {*name*}**)**
Generates and returns a new unique chunk name based on the template *name* ("chunk" if no name is specified). Useful when using the functional form of the **Add-DM** command in scripts to make sure that the new chunk created does not have the same name as an existing one.

(No-Output {*command*}***)**
Evaluates a number of ACT-R *command* while turning all printing off. Useful in reducing output when passing results between commands. Returns the output of the last *command*.

(Output-Stream *file* {**:echo** *echo*}**)**
Causes all production output (i.e., anything produced by **!output!** commands) to be sent to the specified file. If the **:echo** keyword is supplied with a non-nil value, then the output is directed to the specified file, and a copy is also sent to the Listener window. The output stream created should always be explicitly closed using **Close-Output** before clearing, resetting or quitting ACT-R. Returns the stream.

(P *production lhs* **==>** *rhs***)**
Defines *production* as composed of a matching side *lhs* and an action side *rhs*. Returns the new production. See Section 6 for details.

(Parameters *production* {*parameter value*}***)**
Sets a number of *parameter* to *value* for *production*. Each possible parameter and its current value can be obtained using the **production-parameter** command . Parameters can be omitted and appear in any order. A warning is issued if any *value* is of the wrong type or range. Returns the list of values, or **:error** for failed settings. See Section 6 for details.

(PBreak {*production*}***)**
Sets breakpoints for productions. If *production* is about to fire, then the instantiation is displayed as if by **Production Trace**, but the production does not actually fire. Instead, a Lisp break occurs, allowing one to examine and/or change declarative memory, establish other breakpoints, etc. Then continue or abort as indicated by the Lisp environment. Use **PUnbreak** to remove breakpoints. Returns the list of productions with break points.

(PDisable {*production*}***)**
Disables productions. *production* is not allowed to match or fire, but it can be printed by **PP** and re-enabled by **PEnable**. Returns the list of disabled productions, including failed compiled productions.

(PEnable {*production*}***)**
Enables productions. Undoes the effect of **PDisable**. Returns the list of disabled productions, including failed compiled productions.

(PMatches)
Prints out all production instantiations that match against the current state of declarative memory. Returns the conflict set, i.e. the list of matching productions.

(Pop-Goal)
Pops the top goal off the stack and installs it as the focus, similarly to the **!pop!** production command. Returns the new focus.

(PP {*production*}***)**
Prints the specified productions or all enabled ones if none is supplied. Returns the list of productions printed.

(Production-Parameter *production* {*parameter*}***)**
Displays the value of *parameter* for *production*. If no parameter is specified, then all active (as defined by global parameters) parameters are displayed and the production is returned. If parameters are specified, then they are printed with their values and the list of values is returned. See Section 6 for details.

(PSet {*set* | *set* {*command* }* | {**(***set* {*command* }***)**}***}**)**
Parameters Set. Prints, defines and activates sets of parameters. If no argument is given, then the current parameter sets are printed. If the name of a set is given, then the commands composing that set are executed. Otherwise, one or more sets are defined by specifying the name of a new set (or an old one to be redefined) and a number of parameter-setting commands (e.g. **sgp**, **Sdp**, **spp**, **parameters**, or any other commands) composing that set.

(Pstep {*length*}**)**
Runs the production system as with the **Run** command. Displays the list of instantiations at each cycle and asks the user to choose between stepping once more, stopping the run, running without stepping, or selecting one of the instantiations to fire.

(PUnbreak {*production*}***)**
Removes breakpoints for productions. Undoes the effect of **PBreak**. If no production is given, all breakpoints are removed. Returns the list of productions with break points.

(Push-Goal *chunk***)**
Pushes the focus on the goal stack and installs *chunk* as the new focus, similarly to the **!push!** production command. Returns the new focus.

(Rehearse-Chunk {*chunk*}***)**
Rehearses *chunk* by increasing its base-level (if base-level learning is on) and associative strengths (if associative learning is on) as if the chunk had been retrieved. If *chunk* is a list, the the first element is the chunk to rehearse and the rest are the context elements for purposes of strengthening the associative links. Useful to simulate an accurate prior history when both base-level and associative learning are on.

(Reload {*seed*}**)**
Reloads the current model, including any updates made to the model file. A random *seed* can be provided for an initial randomization.

(Reset {*seed*}**)**
Resets ACT-R to its state after loading the last model, i.e. it undoes all interactive commands, production firings, etc. Faster than reload, but does not include updates made to the model file since the previous loading. . A random *seed* can be provided for an initial randomization.

(Reset-IA)
Resets default IA values using present connectivity.

(Run {*length*}**)**
Runs the production system for at most *length* cycles if *length* is an integer, *length* units of (ACT-R) time if *length* is a real value, or until no production can be instantiated, the top goal is **!pop!**ped or a **!stop!** command is issued by a production. Returns the run latency and number of cycles.

(Run-Many {*iteration*}**)**
Runs the production system by focusing in sequence on each chunk defined by the last **Goal-Focus** then calling **Run**, *iterations* times.

(SDM {**ISA** *type*} {*slot value*}***)**
Searches declarative memory. All chunks of type *type* (if specified, otherwise all chunks are considered) with *slot* value equal to *value* are displayed. Returns the list of chunks matching the condition(s).

(Sdp {*specification* | {**(***specification* **)**}*}**)**
specification ::= {*chunk* | **(**{*chunk*}***)**} {{**:***parameter value*}* | {**:***parameter*}*}
Show/Set Declarative Parameters. Any number of chunk parameter specifications can be given, each enclosed in parentheses. If only one is given, then the parentheses are optional. Each specification is composed of a chunk specification, followed by a parameter specification. The chunk specification specifies the chunk, or list of chunks to which it applies. If it is omitted then it applies to all chunks. The parameter specification is similar to the argument of **sgp**. If it is omitted, then all active (as determined by the current setting of global parameters) parameters for the chunk are printed and the chunk itself is returned. If a number of chunk parameters are specified, then those parameters are printed with their current values for the chunk, and a list of the values is returned. If a number of parameter-value pairs are specified, then each chunk parameter is set to its new value, and the list of values is returned, or **:error** for failed settings. See section 4 for details.

(Set-All-Base-Levels {*baselevel* | *references* {*creation-time* } }**)**
Sets the base levels of all chunks. See **Set-Base-Levels** for parameters. Returns the new base level.

(Set-Compilation-Parameters {*parameter value*}***)**
Sets various parameters for initializing compiled productions. The arguments are the same as for the command **Parameters** (minus *production*). Returns the argument.

(Set-Base-Levels {**(***chunk baselevel***)** | **(***chunk references* {*creation-time* }**)**}***)**
Sets the base level activations of chunks. If **Base Level Learning** is off, then the *base level* of *chunk* should be supplied directly. Otherwise, the number of past *references* and perhaps the *creation-time* should be provided. Returns the list of base levels.

(Set-DM {**(***chunkname* **isa** *chunktype* {*slot value*}***)**}***)**
Same as **Add-DM**.

(Set-G *G* {**:threshold** *threshold*}**)**
Sets the value of the goal value *G* and perhaps the goal-threshold, as a percentage of *G*.. Provided for compatibility only. Use **sgp**.

(Set-General-Base-Levels {**(***chunk baselevel***) | (***chunk references creation-time***)**}***)**
Same as **Set-Base-Levels**.

(Set-IA {**(***chunkj chunki sji***)**}***)**
Same as **Add-IA**.

(Set-Similarities {**(***chunkj chunki similarity***)**}***)**
Sets the *similarity* value between *chunkj* and *chunki*.. Returns the list of similarity values.

(Sgp {{**:***parameter value*}* | {**:***parameter*}*}**)**
Show/Set Global Parameters. If no argument is specified, the list of all global parameters is printed, together with their shorthand keywords for use in **sgp** and their current value (nil is returned). If a list of keywords is specified, then each *parameter* is printed with its current value, and the list of values is returned. If a list of keyword-value pairs specified, then each *parameter* is set to its *value* and the list of values is returned. A warning is printed if a *value* is of the wrong type or range for *parameter*, in which case the keyword **:error** is returned instead of the value. See Section 3 for details.

(Similarity *chunkj chunki***)**
Displays the similarity value between *chunkj* and *chunki*.. Returns the similarity value.

(Spp {*specification* | {**(***specification* **)**}*}**)**
specification ::= {*production* | **(**{*production*}***)**} {{**:***parameter value*}* | {**:***parameter*}*} Show/Set Production Parameters. Any number of production parameter specifications can be given, each enclosed in parentheses. If only one is given, then the parentheses are optional. Each specification is composed of a production specification, followed by a parameter specification. The production specification specifies the production, or list of productions to which it applies. If it is omitted then it applies to all productions. The parameter specification is similar to the argument of **sgp**. If it is omitted, then all active (as determined by the current setting of global parameters) parameters for the production are printed and the production itself is returned. If a number of production parameters are specified, then those parameters are printed with their current values for the production, and a list of the values is returned. If a number of parameter-value pairs are specified, then each production parameter is set to its new value, and the list of values is returned, or **:error** for failed settings. See section 6 for details.

(Update-activation)
Updates the activations of all chunks. This involves recomputing all the learned quantities, and helps to update all the cached values.

(Trace-Stream *file* {**:echo** *echo*}**)**
Causes all trace output to be sent to the specified file. The arguments work as for **Output-Stream**, and the resulting stream should always be closed using **Close-Trace**. A finer degree of control over trace dispatching can be achieved through the use of **sgp**. Returns the stream.

(WhyNot {*production*}***)**
Attempts to match *production*(s) against the current state of declarative memory. A detailed matching trace is displayed as by **Exact Matching Trace** and **Partial Matching Trace**, and if any instantiations result they are also displayed. Returns the conflict set, i.e. the list of matching productions.

(WhyNot-Dependency {*dependency*}***)**
Attempts to compile *dependency* into production, with **Production Compilation Trace**, **Exact Matching Trace** and **Partial Matching Trace** on. Returns the list of productions compiled.

Combining Commands

More complex tasks can be accomplished by combining several commands together by sending the output from a command to another command and so on. A distinction must first be drawn between a command and its underlying function, which has the same name as the command with the "**-fct**" suffix appended. A command typically does not evaluate its arguments and, when any number of arguments may be supplied, they are given one after the other without being included in a list. The functions underlying commands, on the other hand, do evaluate their arguments and usually require multiple arguments to be specified as a list. In practice, that means that when results from a command or function are passed to another, then the function form must be used for the latter rather than the command form. E.g., to delete chunks **three**, **five** and **seven**, the command form would be:

(Delete-Chunk three five seven)

whereas the function form would require quoting the arguments and including them in a list:

(Delete-Chunk-fct '(three five seven))

As an example of command combination, to delete all chunks of type **addition-fact**, you could write:

(Delete-Chunk-fct (SDM isa addition-fact))

The internal **SDM** command will print and return the list of chunks of type addition-fact, which is then passed to the **Delete-Chunk-fct** function to be removed from memory. If you do not want the **SDM** command to print the list of chunks which it returns, you can use the **no-output** command to turn off printing during execution of its argument, while still returning the same value:

(Delete-Chunk-fct (no-output (SDM isa addition-fact)))

Some functions take their arguments in a slightly more complex form than a list. To find out the functional form equivalent to a command, use the Lisp function **macroexpand**:

(macroexpand '(Mod-Chunk dime on left next nickel))

will return **(Mod-Chunk-fct 'dime '(on left next nickel))**, indicating that **Mod-Chunk-fct** takes two arguments, first the chunk to be modified, then the list of slot-value pairs.

IMPORTANT UPDATE NOTE: *Unlike previous versions fo the system, the values returned by the ACT-R commands do not directly correspond to chunks and productions but instead to their names. The above list of commands should be sufficient to write code interacting with the ACT-R system. ACT-R users should try to only*

use the above list of commands, and refrain from accessing the internal representation or use internal ACT-R functions.

Outdated Commands

A number of command names have been changed in ACT-R 4.0 from earlier versions to systematize the use of the hyphen "-" as a word separator in commands and to remove references to the phrase "working memory" and its derivatives ("wm", "wme") with the more accurate "declarative memory" and related concepts ("dm", "chunk" and "goal"). ***The outdated commands can still be used, but usage of their replacements is recommended.*** The outdated commands are listed below with their replacements. That information can also be obtained by using the **Help** function with those commands.

(ActivationSources)
See **Activation-Sources**.

(ActrTime {*delay*}**)**
See **Actr-Time**.

(AddIA {**(***wmej wmei sji***)**}***)**
See **Add-IA**.

(AddWM {**(***wmename* **isa** *wmetype* {*slot value*}***)**}***)**
See **Add-DM**.

(ClearAll)
See **Clear-All**.

(ClearGoalStack)
See **Clear-Goal-Stack**.

(ClearProductions)
See **Clear-Productions**.

(ClearWM)
See **Clear-DM**.

(CloseOutput)
See **Close-Output**.

(CloseTrace)
See **Close-Trace**.

(CopyWme {*wme*}***)**
See **Copy-Chunk**.

(DeleteWM {*wme*}***)**
See **Delete-Chunk**.

(Focus-on-Wme *wme***)**
See **Focus-on-Goal**.

(GetBaseLevel {*wme*}***)**
See **Get-Base-Level**.

(GoalStack)
See **Goal-Stack**.

(ModFocus {*slot value*}***)**
See **Mod-Focus**.

(ModWME *wme* {*slot value*}***)**
See **Mod-Chunk**.

(OutputStream *file* {**:echo** *echo*}**)**
See **Output-Stream**.

(Pop-Wme)
See **Pop-Goal**.

(Push-Wme *wme***)**
See **Push-Goal**.

(ResetIA)
See **Reset-IA**.

(SetAllBaseLevels {*baselevel* | *references* {*creation-time* } }**)**
See **Set-All-Base-Levels**.

(SetAnalogizedParameters {*parameter value*}***)**
See **Set-Compiled-Parameters**.

(SetBaseLevels {**(***wme baselevel***)** | **(***wme references* {*creation-time* }**)**}***)**
See **Set-Base-Levels**.

(SetG *G* {**:threshold** *threshold*}**)**
See **Set-G**.

(SetGeneralBaseLevels {**(***wme baselevel***)** | **(***wme creation-time references***)**}***)**
See **Set-General-Base-Levels**.

(SetIA {**(***wmej wmei sji***)**}***)**
See **Set-IA**.

(SetSimilarities {**(***wmej wmei similarity***)**}***)**
See **Set-Similarities**.

(SetWM {**(***wmename* **isa** *wmetype* {*slot value*}***)**}***)**
See **Set-DM**.

(SWM {**ISA** *type*} {*slot value*}***)**
See **SDM**.

(Swp {*specification* | {**(***specification* **)**}*}**)**
See **Sdp**.

(TraceStream *file* {**:echo** *echo*}**)**
See **Trace-Stream**.

(WM {*wme* }***)**
See **DM**.

(WMESlotValue *wme slot* **)**
See **Chunk-Slot-Value**.

(WMEType {*type* | **(***type* **(:include** *supertype***))**} {*slot* | **(***slot value***)**}* **)**
See **Chunk-Type**.

(WMFocus {*wme*}***)**
See **Goal-Focus**.

Section 2 Models

Clear-All command

Although a fully interactive mode is possible, in general ACT-R models should be developed in a file or files instead of the listener to make saving and resetting easier. A call to **(Clear-All)** should always be at the top of the main model file, to clear previous models and define from where to reset the current model:

(Clear-All {*save-model*}**)**
Clears everything from the ACT-R state, including chunk types, chunks, productions, global variables, etc. A call to **Clear-All** ***MUST*** be put at the top of each model file. If the optional argument *save-model* is t or is not specified, both the name of the file and its contents after the call to clear-all are saved for use by the commands **reload** and **reset** respectively. If the argument is nil, neither is saved and neither command can be used. If the argument is a value other than t or nil (e.g. **compiled**), then only the file name is saved, not its contents, and **reset** becomes equivalent to **reload**. This setting can be useful if the model file must be compiled for efficiency purposes, making its contents unsavable.

It should be followed by a call to the **sgp** command (see Section 3) if any global parameters and traces need to be set.

Load and Reset

If preliminary model files need to be included (e.g. the example file **integers** as the standard representation of integers and arithmetic facts), they should be loaded immediately after the clearing and setting of global parameters, using either the Lisp **load** command or the ACT-R **load-model** command (see the Equation example):

(Load-Model *file* {*directory*}**)**
Loads the model *file* in *directory* . If *directory* is not supplied, then *file* is considered relative to the directory of the file in which the command appears, if applicable.

When the model has been run and needs to be reset to its starting point, use the **(reset)** command:

(Reset {*seed*}**)**
Resets ACT-R to its state after loading the last model, i.e. it undoes all interactive commands, production firings, etc. Faster than reload, but does not include updates made to the model file since the previous loading. . A random *seed* can be provided for an initial randomization.

Reset works by saving, when it encounters the initial **(Clear-All),** the contents of the model file *after the Clear-All command* and any other files it loads after it. Resetting then consists in reevaluating the saved contents without re-reading the file itself. This implies that if the model file has been modified since it was loaded, then it must be reloaded rather than reset, using the **reload** command:

(Reload {*seed*}**)**
Reloads the current model, including any updates made to the model file. A random *seed* can be provided for an initial randomization.

For **reset**, **reload** and **load-model** to work properly, the main model file should be loaded using **load-model**, rather than other commands such as **Eval Buffer** which may be available in the Lisp environment. In some Lisp implementations such as Macintosh Common Lisp and Allegro Common Lisp, using the Lisp command **load** will also work. Also, if the model includes some Lisp definitions, those should be either included before the **Clear-All** command or in a separate file to be loaded beforehand, either from the model file or independently.

General Model Format

The general model format should therefore be:

<load or define Lisp functions (if necessary)>

(Clear-All) ;; this command should only appear in the main model file

<set global parameters>

<load related models (if necessary)>

<define chunk types>

<define chunks>

<define chunk parameters (if necessary)>

<define productions>

<define production parameters (if necessary)>

<general commands, commented trace of model run, and other miscellaneous>

Model and Running

The core of the model should define declarative memory (using commands such as **Chunk-Type**, **Add-DM** and others described in Section 4), select the initial goal stack focus (using the **Goal-Focus** command described in Section 5) then procedural memory (using commands such as **p** and **parameters** described in Section 6). Keeping to that order helps avoid having items such as chunks being used before they are defined. The main command to run the model is the **run** command:

(Run {*length*}**)**
Runs the production system for at most *length* cycles if *length* is an integer, *length* units of (ACT-R) time if *length* is a real value, or until no production can be instantiated, the top goal is **!pop!**ped or a **!stop!** command is issued by a production. Returns the run latency and number of cycles.

The **Pstep** command lets the user step through a run and perhaps guide ACT-R by selecting the instantiation to fire:

(Pstep {*length*}**)**

Runs the production system as with the **Run** command. Displays the list of instantiations at each cycle and asks the user to choose between stepping once more, stopping the run, running without stepping, or selecting one of the instantiations to fire.

Section 3 Global Parameters and Traces

Sgp command

The **sgp** command (for Set Global Parameters) is used to query and set the value of global parameters and toggle traces, both interactively and in model files. Since the internal variable names of some global parameters have been changed from ACT-R 2.0, it is ***strongly recommended*** to update the **setf** Lisp commands used to set them in previous models to the corresponding **sgp** command. The syntax of **sgp** is:

(Sgp {{**:***parameter value*}* | {**:***parameter*}*}**)**

Show/Set Global Parameters. If no argument is specified, the list of all global parameters is printed, together with their shorthand keywords for use in **sgp** and their current value (nil is returned). If a list of keywords is specified, then each *parameter* is printed with its current value, and the list of values is returned. If a list of keyword-value pairs specified, then each *parameter* is set to its *value* and the list of values is returned. A warning is printed if a *value* is of the wrong type or range for *parameter*, in which case the keyword **:error** is returned instead of the value.

When **sgp** is called without arguments (i.e. **(sgp)**), the list of global parameters is printed. Each line includes the full name of the parameter, a two- or three-letter keyword for use when setting its value with **sgp**, and its current value. The parameters are listed in groups separated by dotted lines, each containing a number of related parameters. Each group and its parameters is detailed below. A value of NIL means that the function corresponding to the parameter is disabled. A value of T means that it is enabled. Other types of values are specified when acceptable. *Note that values are set as specified and are **NOT** evaluated.*

Rational Analysis

Enable Rational Analysis

keyword: ERA **default value: NIL**

Enables the rational activation and expected gain computations.

G

keyword: G **default value: 20.0**

The value of the goal in the *PG-C* evaluation (ACT Equation 3.1). Must be a positive number.

Expected Gain S

keyword: EGS **default value: NIL**

If enabled, the *S*-value of the Gaussian noise added to the *PG-C* evaluation for each instantiation (ACT equation 3.4). Must be a positive number. The variance can be accessed as **EGN**.

Enable Randomness

keyword: ER **default value: NIL**

If enabled, ties among instantiations are broken randomly, rather than according to some unspecified but deterministic order.

Utility Threshold

keyword: UT **default value: 0.0**
The threshold for the utility (*PG-C*) of a production below which it will not be considered during conflict resolution. Setting it to NIL is equivalent to disabling the threshold, i.e. considering all productions.

Activation parameters

Goal Activation
keyword: GA **default value: 1.0**
The total level of source activation, i.e. the *Wj* in ACT equation 3.5. That level is divided evenly among chunks in the slot values of the current focus (see ARL paper). Must be a positive number.

Base Level Constant
keyword: BLC **default value: 0.0**
A constant added to all chunk base levels, i.e. (ß in ACT equation 3.6). Can be any number.

Activation Noise S
keyword: ANS **default value: NIL**
If enabled, the *S*-value of the Gaussian noise added to each chunk activation at every cycle. Must be a positive number. The variance (σ_2^2 in ACT equation 3.6) can be accessed as **AN**.

Permanent Activation Noise S
keyword: PAN **default value: NIL**
If enabled, the *S*-value of the Gaussian noise added to each chunk activation at creation. Must be a positive number. The variance (σ_1^2 in ACT equation 3.6) can be accessed as **PAN**.

Latency parameters

Latency Factor
keyword: LF **default value: 1.0**
The multiplicative factor *F* in ACT equation 3.10. Must be a positive number.

Latency Exponent
keyword: LE **default value: 1.0**
The exponent factor *f* in ACT equation 3.10. Must be a positive number.

Default Action Time
keyword: DAT **default value: 0.05**
The default action time of a production, i.e. the right-hand side part of the *a* parameter in ACT equation 3.3. Must be a positive number, usually interpreted in milliseconds. *This default has been changed from 1.0 in previous ACT-R versions.*

Partial Matching parameters

For further details on partial matching, see Section 7 in this manual and the papers cited in the bibliography.

Partial Matching
keyword: PM **default value: NIL**
Enables partial matching.

Mismatch Penalty
keyword: MP **default value: 1.5**
Penalty subtracted from the activation of a chunk for a complete mismatch in a slot pattern. Multiplied by *1.0 - similarity* for partial mismatches. Must be a positive number.

Retrieval Threshold
keyword: RT **default value: NIL**
When enabled, chunks with activation levels below this threshold cannot be retrieved. Works with both partial and exact (standard) matching. Can be any number.

Learning parameters

Optimized Learning
keyword: OL **default value: T**
If enabled, an efficient approximation to the formulas for production strength learning and base level learning is used (see the discussion of ACT equation 4.1).

Base Level Learning
keyword: BLL **default value: NIL**
If enabled, used as *d* in ACT equation 4.1 for learning the base level activations of chunks. Must be a positive number, with 0.5 the standard value.

Associative Learning
keyword: AL **default value: NIL**
If enabled, used as *assoc* in ACT equation 4.3 for learning the associative strength between pairs of chunks. Must be a positive number.

Strength Learning
keyword: SL **default value: NIL**
If enabled, used as *d* in ACT equation 4.4 for learning the strength of productions. Must be a positive number, preferably less than 1.0.

Parameters Learning
keyword: PL **default value: NIL**
If enabled, the parameters *a*, *b*, *q* and *r* will be learned for each production according to ACT equations 4.5 and 4.6. Can also be set to the decay rate in the ACT equations 4.7 and 4.8.

Traces

These traces are enumerated here roughly in the order they would appear during each production cycle. They can be enabled by being set to T, which sends the trace output to the standard output (Lisp listener), or to any file name, pathname or stream if output is to be directed to another destination. Each trace message starts on a new line and is indented by a number of spaces proportional to the current depth of the goal stack.

Exact Matching Trace
keyword: EMT **default value: NIL**
If enabled, prints a message during production matching every time a condition is evaluated.

Partial Matching Trace
keyword: PMT **default value: NIL**
If enabled, prints a message during production matching every time a condition is evaluated when **Partial Matching** is enabled.

Production Compilation Trace
keyword: PCT **default value: T**
If enabled, prints the main steps of the production compilation process.

Activation Trace
keyword: ACT **default value: NIL**
If enabled, prints a trace of the main steps during computation of chunk activation.

Conflict Resolution Trace
keyword: CRT **default value: NIL**
If enabled, prints the name of each production as it is matched curing conflict resolution.

Conflict Set Trace
keyword: CST **default value: NIL**
If this option is enabled, prints the number of instantiations in the conflict set, and, when rational analysis is enabled, the number of instantiations that were considered and the expected gain of the instantiation that will fire.

Matches Trace
keyword: MT **default value: NIL**
If enabled, prints all the instantiations generated. A setting of SHORT will simply print the list of variables and their values for each instantiation, and a setting of T will print the whole production(s) with the variables replaced by their values.

Production Trace
keyword: PT **default value: NIL**
If enabled, prints the selected production instantiation before firing it. A setting of SHORT will simply print the list of variables and their values, and a setting of T will print the whole production with the variables replaced by their values.

Cycle Trace
keyword: CT **default value: T**
If enabled, prints the cycle number, current time and name of selected production.

Latency Trace

keyword: LT **default value: T**

If enabled, prints the latency of the production matching (when rational analysis is enabled - see ACT equations 3.10) and the latency of the action side of the production.

Output Trace

keyword: OT **default value: T**

If enabled, prints the output of **!output!** commands on the right-hand side of productions.

Declarative Memory Trace

keyword: DMT **default value: NIL**

If enabled, prints a message when a chunk is created, deleted or modified.

Goal Trace

keyword: GT **default value: NIL**

If enabled, prints a message when the focus is changed.

Verbose

keyword: V **default value: T**

If enabled, the traces are printed as enabled. If disabled, turns off all trace output regardless of which is enabled. ***Disabling this trace is useful to easily turn off all trace output when running in batch mode where performance is key.***

Obtaining Parameter Value

To print and return the value of some but not all parameters, **sgp** can be called with just the list of those parameters. For example, to print the value of **G**, **GThreshold** and **Conflict Resolution Trace**:

(sgp :g :gth :crt)

which will also return the list of their (default here) values:

(20.0 0.05 NIL)

Setting Parameters and Traces

To set global parameters and traces, simply call the **sgp** command with as arguments the series of one- to three-letters keywords preceded by the keyword specifier **:** (no space), followed by the desired value. For example, to set the Goal value **G** to 10.0 and to turn **Activation Trace** on:

(sgp :g 10.0 :act t)

In general, **sgp** will check that the values supplied fall within the required type and range for the parameter, and reject those that don't while printing a warning message. Note again that turning **Verbose** off is a quick way to shut off all the traces for maximum running efficiency.

Section 4 Declarative Memory

Defining Chunk Types

Each chunk is of a particular type, which must be defined before an element of that type can be created. Types are defined using the **Chunk-Type** command:

(Chunk-Type {*type* | **(***type* **(:include** *supertype***))**} {*slot* | **(***slot value***)**}* **)**
Defines a new chunk type, which may include a *supertype* from which *type* inherits its slots, and a number of *slot*, each with an optional initial *value*. If no argument is passed to **Chunk-Type**, then a list of the existing types is printed. Return the chunk type defined or the list of all chunk types if no argument is given.

The ACT-R type hierarchy only allows single inheritance, i.e. only one (previously defined) type can be defined as the parent of a new type, which inherits all the slots of the parent. Chunks of the new type will match any production template specifying the type of the parent or further ancestors.

Defining Chunks

Chunks can be added using the **Add-DM** command:

(Add-DM {**(***chunkname* **isa** *chunktype* {*slot value*}***)**}***)**
Adds chunks to declarative memory. The new chunk *chunkname* is of chunk type *chunktype*, with each *slot* being initialized to *value*. Returns the list of added chunks.

Chunktype must have been previously defined as including each *slot*. Note that not all slots of the chunk need to be initialized (in which case they will contain the standard value NIL), and that they can appear in any order. If a symbol which hasn't yet been defined as a chunk appears as a slot *value* in a chunk (or production) definition, it will be defined by default as a chunk of the built-in type **chunk**, which has no slots. Chunks can be redefined of being of a different type, in which case the new type and slot/values definition will replace the old, although the rest of the chunk history will remain unchanged and a warning message will be printed.

A chunk can be modified using the **Mod-Chunk** command:

(Mod-Chunk *chunk* {*slot value*}***)**
Modifies *chunk* by setting each *slot* to *value*. Returns the modified chunk.

Chunks can be deleted using the **Delete-Chunk** command:

(Delete-Chunk {*chunk*}***)**
Deletes chunks from declarative memory. Do not use if *chunk* is still used in other chunks or productions. Returns the list of deleted chunks.

A warning message will be printed if a deleted chunk is still used as a value in another chunk or production, and the resulting effect is undefined (in other words: don't do it).

Search and Display

The contents of declarative memory, or just some selected chunks, can be displayed using the **DM** command:

(DM {*chunk* }***)**
Displays the specified chunks, or the entire contents of declarative memory if none is supplied. For each *chunk*, its name is printed followed by its current activation, then its type and slot-value pairs. The current focus is displayed preceded by two asterisks ("**"). Returns the list of chunks printed.

Declarative memory can also be searched for chunks fitting a particular pattern using the **SDM** command:

(SDM {**ISA** *type*} {*slot value*}***)**
Searches declarative memory. All chunks of type *type* (if specified, otherwise all chunks are considered) with *slot* value equal to *value* are displayed. Returns the list of chunks matching the condition(s).

For example, to display all chunks of type **addition-fact** with a value of slot **sum** of **five**:

```
(SDM isa addition-fact sum 5)
```

Individual slot values can be obtained using the **Chunk-Slot-Value** command:

(Chunk-Slot-Value *chunk slot* **)**
Returns the value of *slot* of *chunk*.

Chunk Activation

To each chunk is associated a quantity called activation, described in ACT equation 3.5, which is the sum of its base level and the activation spread from the current activation sources. The base level of chunks (see ACT equation 4.1) can be queried using the **Get-Base-Level** command and set using the **Set-Base-Levels** and **Set-All-Base-Levels** command:

(Get-Base-Level {*chunk*}***)**
Returns base level activation of chunks. Returns the list of base levels.

(Set-Base-Levels {**(***chunk baselevel***)** | **(***chunk references* {*creation-time* }**)**}***)**
Sets the base level activations of chunks. If **Base Level Learning** is off, then the *base level* of *chunk* should be supplied directly. Otherwise, the number of past *references* and perhaps the *creation-time* should be provided. Returns the list of base levels.

(Set-All-Base-Levels {*baselevel* | *references* {*creation-time* } }**)**
Sets the base levels of all chunks. See **Set-Base-Levels** for parameters. Returns the new base level.

For example, with **Base Level Learning** on, to specify that chunk **3+4=7** was created at time **-1000.0** and has been accessed **10** times since:

```
(Set-Base-Levels (3+4=7 -1000.0 10))
```

The rest of a chunk's activation is computed by summing the product of each activation source level times the Interactive Activation (IA) value from the source to the chunk. The activation sources are the set of chunks in the slots of the focus, at a level of **Goal Activation** divided by the number of sources. They can be displayed using the **Activation-Sources** command:

(Activation-Sources)
Displays the current activation sources with their level. Returns the list of sources.

IA values can be queried and set using the **IA** and **Add-IA** command respectively:

(IA *chunkj chunki***)**
Displays the IA value between *chunkj* and *chunki* and return the value.

(Add-IA {**(***chunkj chunki sji***)**}***)**
Adds inter-associativity (IA) values between chunks. *Sji* is defined as the new IA value from *chunkj* to *chunki*. Returns the list of added IAs.

IA values which are not explicitly specified are initialized using ACT equation 4.2: the value of the IA value *Sji* between *chunkj* and *chunki* is *log(m/n)* if *chunkj* appears as a slot value of *chunki* and 0.0 otherwise. *m* is the number of chunks in declarative memory and *n* is the fan of *chunkj*, i.e. the number of chunks in which *chunkj* appears as a slot value.

Declarative Memory Parameters

Sdp Command

Base-levels, IA values and other declarative memory parameters can also be displayed, returned and set using the more general **Sdp** command:

(Sdp {*specification* | {**(***specification* **)**}*}**)**
specification ::= {*chunk* | **(**{*chunk*}***)**} {{**:***parameter value*}* | {**:***parameter*}*}
Show/Set Declarative Parameters. Any number of chunk parameter specifications can be given, each enclosed in parentheses. If only one is given, then the parentheses are optional. Each specification is composed of a chunk specification, followed by a parameter specification. The chunk specification specifies the chunk, or list of chunks to which it applies. If it is omitted then it applies to all chunks. The parameter specification is similar to the argument of **sgp**. If it is omitted, then all active (as determined by the current setting of global parameters) parameters for the chunk are printed and the chunk itself is returned. If a number of chunk parameters are specified, then those parameters are printed with their current values for the chunk, and a list of the values is returned. If a number of parameter-value pairs are specified, then each chunk parameter is set to its new value, and the list of values is returned, or **:error** for failed settings.

This command uses a syntax similar to **sgp**, where any number of *parameter* keywords can be specified in any order, perhaps followed by an assigned *value*. *Again, values are **NOT** evaluated.* The list of chunk parameters follows, with their keywords and default values, and a description:

Name

keyword: :name
The name of the chunk. Cannot be set.

Activation
keyword: :activation **default value: 0.0**
The chunk activation, *Ai* in ACT equation 3.5. Cannot be set directly.

Source
keyword: :source **default value: NIL**
The chunk source level, *Wj* in ACT equation 3.5. Can be set to NIL (0.0) or any number.

Base Level
keyword: :base-level **default value: Base Level Constant**
The chunk base level, *Bi* in ACT equation 3.5. Can be set to any number, only when **Base Level Learning** is disabled.

Creation Time
keyword: :creation-time **default value: 0.0**
The time at which the chunk was created. Used in **Base Level Learning** approximation equation to compute *L* (see discussion of ACT equation 4.1). Can be set to any number.

References
keyword: :references **default value: (1.0)**
The number and, when **Optimized Learning** is off, list of production matching times of chunk. Used in ACT equation 4.1. Can be set to the number or the list of references.

Source Spread
keyword: :source-spread **default value: 0.0**
The spreading activation to chunk, the second term in ACT equation 3.5. Cannot be set directly.

IAs
keyword: :ias **default value: see above section**
The Interactive Activation values to chunk *i*, *Sji* in ACT equation 3.5. Can be set to a list of chunk-number pairs used to update the IAs to *i*.

Creation Cycle
keyword: :creation-cycle **default value: 0.0**
The production cycle at which the chunk was created, used to compute *F* in ACT equation 4.3. Can be set to any number.

Needed
keyword: :needed **default value: 0.0**
The number of times the chunk has been retrieved, *F(Ni)* in ACT equation 4.3. Can be set to any number.

Contexts
keyword: :contexts **default value: 0.0**
The number of production cycles the chunk was in context, scaled by its source level at the time, *F(Cj)* in ACT equation 4.3. Can be set to any number.

Permanent Noise

keyword: :permanent-noise **default value: 0.0**
The permanent chunk noise, in ACT equation 3.6. Can be set to any number.

Similarities

keyword: :similarities **default value: ((chunk . 1.0))**
The similarity values used in partial matching (see ACT equation 3.8). Can be set to a list of chunk-number pairs used to update the similarities to chunk.

Use of Sdp Command

For example, in the Tower of Hanoi model, to print all active parameters for all chunks:

```
(Sdp)
```

To print the activation, base level and source spread for chunks penny, nickel, dime and quarter:

```
(Sdp (penny nickel dime quarter) :activation :base-level :source-spread)
```

To set the base level of those chunks to their monetary values, then print the same information as above:

```
(Sdp (penny :base-level 0.01) (nickel :base-level 0.05)
     (dime :base-level 0.1) (quarter :base-level 0.25)
     ((penny nickel dime quarter)
     :activation :base-level :source-spread))
```

Section 5 Goal Stack

ACT-R provides for a simple goal-stacking mechanism. At any point in time, (at most) one chunk is the **focus**, aka the (top) goal. This element is the root from which all productions must match. A chunk can be designated as the focus, replacing the current one if one had previously been selected, using the **Goal-Focus** command:

(Goal-Focus {*chunk*}***)**
Sets the focus to *chunk* , or prints the current focus if none is supplied. If more than one *chunk* is specified, then the focus is set to the first one and the other ones are kept in a list for use by **Run-Many**. Returns the new focus.

The slot values of the focus element can also be modified using the **Mod-Focus** command:

(Mod-Focus {*slot value*}***)**
Same as **Mod-Chunk**, with the chunk set to the focus. Returns the modified focus.

The focus can also be manipulated by productions, which can push the focus on the **goal stack**, replacing it with a new one, pop the top goal off the goal stack to replace the current focus, or simply switch the focus without manipulating the stack (see Section 6). The contents of the goal stack can be inspected and cleared using the **Goal-Stack** and **Clear-Goal-Stack** commands respectively:

(Goal-Stack)
Displays the focus, then the current contents of the goal stack from top to bottom. Each goal is followed by its goal value **G**. Returns the goal stack, i.e. a list of the focus followed by each goal on the stack in descending order.

(Clear-Goal-Stack)
Clears the goal stack by popping all goals and restoring the top goal as the focus.

Section 6 Procedural Memory General Production Syntax

An ACT-R production consists of two halves: a matching part aka the **left-hand side** (lhs) and an action side aka the **right-hand side** (rhs). A production is declared using the **p** command:

```
(p  production-name
  { =chunk>
       isa  chank-type
     { {-} slot  {value  | =variable }}*
  | !eval!  expression
  | !bind!  =variable  expression }*
==>
  { =chunk>
     { isa  chunk-type  }
     { slot  {value  | =variable }}*
  | !push!  =variable
  | !pop!
  | !focus-on!  =variable
  | !output!  ("output-string"  {expression  }*)
  | !eval!  expression
  | !bind!  =variable  expression
  | !delete!  =variable
  | !copy!  =variable  {chunk
  | !stop!
  | !restart! }*
)
```

The opening "**(p**", delimiting "**==>**" and final "**)**" are all required. *production-name* is the name of the production, and lhs and rhs are the two sides of the production separated by the delimiter. Each side is composed of a number of chunk patterns used to match (for the lhs), create or modify (for the rhs) chunks in declarative memory, and some special commands appearing between exclamation marks "**!**".

Productions Left-hand side

Variables can be bound to any element in declarative memory except the special element NIL used to initialize slots. They are represented by an equal sign "**=**" followed by the variable name. A left-hand side chunk pattern is composed of the variable bound to the *chunk* followed by the chunk selector "**>**", a type identification consisting of **isa** followed by the *chunk type*, then any number of conditions on its slot values. A condition consists of the *slot* name, followed by a *value* or *variable*. If a *value* is specified, then the content of the slot must equal the value for the condition to succeed. If a *variable* is specified and that variable has not appeared earlier in the lhs, then the condition succeeds and the variable is **bound**, i.e. its value later in the production is set to the value of the slot. If that value then appears later in a condition then its previously set value is sued to test the condition. If the negation marker "**-**" appears before the slot name, then the test is negated, i.e. it succeeds if it would otherwise fail and vice versa.

A chunk will match a chunk pattern it is of the required chunk type (or one of its descendants - see the chunk type hierarchy in Section 4) and if all the slot conditions are satisfied. **The first chunk pattern**

must match to the focus. If a variable which has been previously bound is matched at the head of a chunk pattern, it is matched against the pattern in what is called a **direct retrieval**. If it hasn't yet been bound, all chunks in declarative memory are matched against the pattern independently in what is called an **indirect retrieval**. A set of bindings for all the variables appearing on the lhs of a production is called an **instantiation** of that production.

A number of special commands can appear on the lhs of a production:

!eval! *expression*
expression is evaluated by Lisp. If the result is NIL, then matching fails, otherwise it succeeds. *expression* can be any legal Lisp expression and can include ACT-R variables previously bound in the lhs. Failures are traced by **Exact Matching Trace**. This command, inserted in parentheses, can also be used to directly specify a slot value.

!bind! *variable expression*
expression is evaluated by Lisp and *variable* is bound to the result. If the result is NIL then matching fails, and succeeds otherwise. Failures are traced by **Exact Matching Trace**.

Productions Right-hand side

The syntax of the right-hand side is similar to the left-hand side's, but the semantics is different. When a chunk pattern appears in the right-hand side, if the corresponding variable has been bound in the left-hand side the chunk will be modified, otherwise it will be created. The chunk type only needs to be specified is it hasn't previously been. Slot/value pairs now indicate how to modify or initialize the chunk slots. All variables appearing as slot values must have been previously bound, except for **return variables**. If a previously unbound variable appears as a slot value of a chunk which will be pushed on the stack (see below), it will be bound to the value of that slot *when the chunk is popped off the stack.* It will then be copied to the chunk slots in which it appears later in the right-hand side (see examples factorial and fibonacci for a simple and complex usage of return variables, respectively).

A number of special commands can also appear on the rhs of a production:

!push! *variable*
Pushes the current focus on top of the stack and replaces it with the subgoal *variable*, which must have previously bound. The previous focus can be restored by a later production using the **!pop!** command. This command is traced by **Goal Trace**.

!pop!
Pops the current focus and restores as the focus the top goal of the stack. If the stack is empty, then the focus becomes NIL and the production system stops. This command is traced by **Goal Trace**.

!focus-on! *variable*
Replaces the current focus with the goal *variable*, which must have been previously bound. The current focus is *not* saved on the stack. This command is traced by **Goal Trace**.

!restart!
Pops all the goals on the stack and restores the top-most goal. This command is traced by **Goal Trace**.

!output! ("*output-string*" {*expression* }* | {*expression* }***)**
Prints a message to **Output Trace** , which defaults to T, i.e. the Listener window, but can be redirected to any file or turned off by being set to NIL. The message consists of *output-string*, with the special dispatchers in the string replaced by the evaluation of the expressions, in order. In most cases, the only dispatcher needed is the two-character sequence ~**S**, which tries to print the best representation of the result of *expression*. If the default format is used, then the output string can be omitted and the values to be printed can be simply listed. For other dispatchers, see the Lisp command **format**. This command is controlled by **Output Trace**.

!eval! *expression*
expression is evaluated by LISP. This command is used strictly for the side-effects of *expression*, and unlike its left-hand side version does not test the resulting value.

!bind! *variable expression*
expression is evaluated by Lisp and *variable* is bound to the result.

!delete! *variable*
Deletes the chunk bound to *variable*. This command is traced by **Declarative Memory Trace**.

!copy! *variable* {*chunk* }*
Copies each *chunk* (specified by a variable or expression) then binds *variable* to the list of chunk copies.

!stop!
Stops the production system after this cycle.

Production Parameters

Spp and related commands

Conflict resolution is the process by which the system chooses among several competing instantiations. The value *PG - C* of a production instantiation is determined by a number of quantities associated with a production (see ACT equations equations 3.2, 3.3, 4.5 and 4.6) and which can be displayed and set using the **production-parameter** and **parameters** commands, respectively, or the more general **spp** command:

(Production-Parameter *production* {*parameter*}***)**
Displays the value of *parameter* for *production*. If no parameter is specified, then all active (as defined by global parameters) parameters are displayed and the production is returned. If parameters are specified, then they are printed with their values and the list of values is returned.

(Parameters *production* {*parameter value*}***)**
Sets a number of *parameter* to *value* for *production*. Each possible parameter and its current value can be obtained using the **production-parameter** command . Parameters can be omitted and appear in any order. A warning is issued if any *value* is of the wrong type or range. Returns the list of values, or **:error** for failed settings.

(Spp {*specification* | {**(***specification* **)**}*}**)**

specification ::= {*production* | **(**{*production*}***)**} {{**:***parameter value*}* | {**:***parameter*}*} Show/Set Production Parameters. Any number of production parameter specifications can be given, each enclosed in parentheses. If only one is given, then the parentheses are optional. Each specification is composed of a production specification, followed by a parameter specification. The production specification specifies the production, or list of productions to which it applies. If it is omitted then it applies to all productions. The parameter specification is similar to the argument of **sgp**. If it is omitted, then all active (as determined by the current setting of global parameters) parameters for the production are printed and the production itself is returned. If a number of production parameters are specified, then those parameters are printed with their current values for the production, and a list of the values is returned. If a number of parameter-value pairs are specified, then each production parameter is set to its new value, and the list of values is returned, or **:error** for failed settings. See section 6 for details.

These commands use a syntax similar to **sgp**, where any number of *parameter* keywords can be specified in any order, perhaps followed by an assigned *value*. The list of production parameters follows, with their keywords and default values, and a description:

Name

keyword: :name

The name of the production. Cannot be set.

Strength

keyword: :strength **default value: 0.0**

The production strength, *Sp* in ACT equation 3.10. Can be set to any number, only when **Production Strength Learning** is off.

Creation Time

keyword: :creation-time **default value: 0.0**

The time at which the production was created. Used in **Production Strength Learning**. Can be any number.

References

keyword: :references **default value: (1.0)**

The number and, when **Optimized Learning** is off, list of firing times for production. Used in ACT equation 4.4. Can be set to the number or the list of references.

Q

keyword: :q **default value: 1.0**

The estimated probability that the production will achieve its intended effect. Used to randomly determine the success of production firing if **Chance** isn't specified. Should be a probability, i.e. a number between 0.0 and 1.0.

A

keyword: :a **default value: Default Action Time**

The estimated effort spent executing the production, usually interpreted as the amount of time. Used to increment the Time counter if **Effort** isn't specified. Should be a positive number.

R

keyword: :r **default value: 1.0**
The estimated probability that the production will lead to eventual success. Should be a probability, i.e. a number between 0.0 and 1.0.

B

keyword: :b **default value: 1.0**
The estimated effort spent after executing the production, usually interpreted as the amount of time. Should be a positive number.

PG-C

keyword: :pg-c **default value: (q*r) * G - (a + b)**
The current **PG-C** value of the production when rational analysis is enabled. Cannot be set directly.

Value

keyword: :value **default value: 0.0**
The value of the production when rational analysis is disabled. Can be any Lisp expression evaluating to a number.

Successes

keyword: :successes **default value: 0.0**
The *m* in ACT equation 4.5 and *n* in 4.6, i.e. the actual number of successful firings of production. Is learned by **Parameters Learning**. A prior can be set to any positive number or list of times.

Failures

keyword: :failures **default value: 0.0**
The *n* in ACT equation 4.6, i.e. the actual number of failures of production. Is learned by **Parameters Learning**. A prior can be set to any positive number or list of times.

Efforts

keyword: :efforts **default value: 0.0**
The *sum of Ei* in ACT equation 4.6, i.e. the actual sum of efforts spent successfully executing production. Is learned by **Parameters Learning**. A prior can be set to any positive number or list of efforts.

Eventual Successes

keyword: :eventual-successes **default value: 0.0**
The *m* in the **r** version of ACT equation 4.5 and *n* in the **b** version of 4.6, i.e. the actual number of eventual successes. Is learned by **Parameters Learning**. A prior can be set to any positive number or list of times.

Eventual Failures

keyword: :eventual-failures **default value: 0.0**
The *n* in the **r** version of ACT equation 4.5, i.e. the actual number of eventual failures. Is learned by **Parameters Learning** . A prior can be set to any positive number or list of times.

Eventual Efforts

keyword: :eventual-efforts **default value: 0.0**
The *sum of Ei* in the **b** version of ACT equation 4.5, i.e. the actual sum of efforts spent after executing the production until reaching eventual success. Is learned by **Parameters Learning** . A prior can be set to any positive number or list of efforts.

Q-alpha

keyword: :q-alpha **default value: 1.0**

The *alpha* in the **Parameters Learning** ACT equation 4.5, i.e. the prior number of successful firings of production. Should be a positive number. SET SUCCESSES INSTEAD.

Q-beta

keyword: :q-beta **default value: 0.0**

The *beta* in the same equation, i.e. the prior number of failures of production. Should be a positive number. SET FAILURES INSTEAD.

A-z

keyword: :a-z **default value: Default Action Time**

The *z* in ACT equation 4.6, i.e. the prior amount of effort spent on the production. Should be a positive number. Also **a-b** for compatibility purposes. SET EFFORTS INSTEAD

A-v

keyword: :a-v **default value: 1 .0**

The *v* in the same equation, i.e. the prior number of firings of production. Should be a positive number. SET SUCCESSES INSTEAD.

R-alpha

keyword: :r-alpha **default value: 1.0**

The *alpha* in the **r** version of ACT equation 4.5, i.e. the prior number of eventual successes. Should be a positive number. SET EVENTUAL-SUCCESSES INSTEAD.

R-beta

keyword: :r-beta **default value: 0.0**

The *beta* in the same equation, i.e. the prior number of eventual failures. Should be a positive number. SET EVENTUAL-FAILURES INSTEAD.

B-z

keyword: :b-z **default value: 1.0**

The *z* in the **b** version of ACT equation 4.6, i.e. the prior amount of effort spent after firing the production. Should be a positive number. Also **b-b** for compatibility purposes. SET EVENTUAL-EFFORTS INSTEAD.

B-v

keyword: :b-v **default value: 1.0**

The *v* in the same equation, i.e. the prior number of firings of production. Should be a positive number. SET EVENTUAL-SUCCESSES INSTEAD.

Chance

keyword: :chance **default value: 1.0**

The "real-world" probability that the production action will achieve its intended effect. Used to randomly determine the success of production firing. **Q** will be learned from this quantity if learning is enabled. Can be any Lisp expression evaluating to a probability.

Effort

keyword: :effort **default value: NIL**

The "real-world" effort spent executing the production, usually interpreted as the amount of time. Used to increment the Time counter. **A** and **B** will be learned from this quantity if learning is enabled. Can be any Lisp expression evaluating to a positive number.

Success

keyword: :success **default value: NIL**

When set to T, defines the production to be the end of a successful run, leading to the learning of **R** and **B** if **Parameters Learning** is enabled. Can be any Lisp expression.

Failure

keyword: :failure **default value: NIL**

When set to T, defines the production to be the end of a failed run. If both **Success** and **Failure** are set in the same production, then that production ends the run, i.e. clears the run history, without either success and failure. Can be any Lisp expression.

Use of Spp Command

A number of parameters (**value**, **chance**, **effort**, **success** and **failure**) can take as value an arbitrary Lisp expression, possibly including variables from the production. Those expressions are evaluated for every instantiation. *All other parameters are set as specified and are **NOT** evaluated*.

For example, in the Tower of Hanoi model, to print all active parameters for all productions:

```
(spp)
```

To print only the strength and pg-c for productions move-disk and clear-disk:

```
(spp (move-disk clear-disk) :strength :pg-c)
```

To set the strength of move-disk to 1.0 and its a to 1.0 and the r of clear-disk to 0.5 and its b to 2.0, then print all active parameters for both productions:

```
(spp (move-disk :strength 1.0 :a 1.0)
     (clear-disk :r 0.5 :b 2.0)
     ((move-disk clear-disk)))
```

To define move-disk as a success production and clear-disk as a failure (after switching parameters learning on):

```
(spp (move-disk :success t) (clear-disk :failure t))
```

Display and Selection

The contents of procedural memory or some selected productions can be displayed and returned using the **pp** command:

(PP {*production*}***)**

Prints the specified productions or all enabled ones if none is supplied. Returns the list of productions printed.

Note that disabled productions are not printed by the **(pp)** command, but will be if given explicitly as arguments. Productions can be disabled and reactivated using the **PDisable** and **PEnable** commands respectively:

(PDisable {*production*}***)**
Disables productions. *production* is not allowed to match or fire, but it can be printed by **PP** and re-enabled by **PEnable**. Returns the list of disabled productions, including failed compiled productions.

(PEnable {*production*}***)**
Enables productions. Undoes the effect of **PDisable**. Returns the list of disabled productions, including failed compiled productions.

Breakpoints can be set and removed for productions by using the **PBreak** and **PUnbreak** productions respectively:

(PBreak {*production*}***)**
Sets breakpoints for productions. If *production* is about to fire, then the instantiation is displayed as if by **Production Trace**, but the production does not actually fire. Instead, a Lisp break occurs, allowing one to examine and/or change declarative memory, establish other breakpoints, etc. Then continue or abort as indicated by the Lisp environment. Use **PUnbreak** to remove breakpoints. Returns the list of productions with break points.

(PUnbreak {*production*}***)**
Removes breakpoints for productions. Undoes the effect of **PBreak**. If no production is given, all breakpoints are removed. Returns the list of productions with break points.

To determine which production instantiations match the current state of memory, and why a particular production does not, use the **PMatches** and **whynot** productions respectively:

(PMatches)
Prints out all production instantiations that match against the current state of declarative memory. Returns the conflict set, i.e. the list of matching productions.

(WhyNot {*production*}***)**
Attempts to match *production*(s) against the current state of declarative memory. A detailed matching trace is displayed as by **Exact Matching Trace** and **Partial Matching Trace**, and if any instantiations result they are also displayed. Returns the conflict set, i.e. the list of matching productions.

Section 7 Partial Matching

Enabling Partial Matching

Partial matching enables ACT-R to display errors of omission and commission, as described in the papers cited in the bibliography. It consists in modifying the pattern matching algorithm for production lhs from an exact backtracking search to a sequential, activation-penalty matching. It can be activated and traced by the global parameter **partial matching** and trace **partial matching trace**, respectively:

Partial Matching
keyword: PM **default value: NIL**
Enables partial matching.

Partial Matching Trace
keyword: PMT **default value: NIL**
If enabled, prints a message during production matching every time a condition is evaluated when **Partial Matching** is enabled.

Penalty and Threshold

Partial matching does not apply to the matching of the focus (first pattern in each production lhs) or to the type-checking of chunks. For all chunks after the focus, a mismatch between an actual slot value and the desired pattern results not in outright failure, but in the subtraction from the activation of the chunk of an amount equal to the **mismatch penalty** constant times the degree of mismatch between desired and actual value. If, after matching, the modified activation of the chunk remains above a **retrieval threshold**, then the match is successful, otherwise it fails. For **indirect matches**, only the chunk with the highest resulting activation matches (thus no multiple instantiations of a production), and no backtracking occurs to consider other less active elements if matching fails later in the production. Note that **retrieval threshold** defaults to NIL and therefore *MUST* be set when **partial matching** is activated (it can also be set to work with standard matching).

The **mismatch penalty** and **retrieval threshold** global parameters are:

Mismatch Penalty
keyword: MP **default value: 1.5**
Penalty subtracted from the activation of a chunk for a complete mismatch in a slot pattern. Multiplied by *1.0 - similarity* for partial mismatches. Must be a positive number.

Retrieval Threshold
keyword: RT **default value: NIL**
When enabled, chunks with activation levels below this threshold cannot be retrieved. Works with both partial and exact (standard) matching. Can be any number.

Similarities

The degree of match, aka the **similarity** (which is equal to 1.0 - the degree of mismatch), between two values defaults to 1.0 if the two values are identical chunks, and 0.0 if they are different chunks and not chunks at all. Similarities between chunks can be queried and set using the **similarity** and **Set-Similarities** commands respectively (note that similarities are symmetrical, i.e. the similarity between *chunkj* and *chunki* is equal to the similarity between *chunki* and *chunkj*, and **Set-Similarities** sets both at the same time):

(Similarity *chunkj chunki***)**
Displays the similarity value between *chunkj* and *chunki*.. Returns the similarity value.

(Set-Similarities {**(***chunkj chunki similarity***)**}***)**
Sets the *similarity* value between *chunkj* and *chunki*.. Returns the list of similarity values.

Section 8 Production Compilation

Overview

Declarative examples to be compiled into productions are stored in chunks of the predefined type **dependency**. When a goal of type dependency is successfully popped, an attempt is made to compile a production based on that dependency. The resulting production is strictly based on the contents of the slots of the dependency, as detailed below. Note that if the dependency is popped by a production defined as a failure (by the :failure parameter) or is popped because no production can apply, then no production compilation is attempted.

Dependencies

The predefined chunk type **dependency** contains the following slots, which as usual default to NIL and can be left unspecified if not needed:

Goal
The goal to which the compiled production will apply.

Modified
An appropriately modified copy of the goal if it needs to be modified.

Stack
A subgoal or list of subgoals used to solve the goal. They are generated in listing order and **!push!**ed on the stack in reverse listing order. The slot can also contain the values **success** or **failure**, in which case the production will be marked as a success or a failure and the goal will be popped.

Constraints
A constraint or list of constraints used to specify the left-hand side mapping between the goal and the right-hand side. The most simple constraint consists of the name of a chunk, to be expanded into its full description. A description constraint is a possibly partial chunk description consisting of a list of chunk name, type and slot-value pairs similar to those appearing in the **Add-DM** command. An eval constraint is a list containing an **!eval!** command. Those constraints will be inserted in the left-hand side of the compiled production in listing order.

Actions
A production rhs command or list of commands (each command in its own list), to be inserted at the end of the right-hand side of the compiled production.

Generals
A value or list of values which should be variabilized in the compiled production instead of appearing as constants.

Specifics
A value or list of values which should appear as constants in the compiled production instead of being variabilized.

Dont-Cares
A value or list of values which should be left out of the compiled production together with the slots in which they appear

Differents
A list (or list of lists) of values which, when the first one appears as a slot value in a production lhs chunk slot, leads to the insertion of a negation test for that slot against the rest of the values in the list.

Compiled Productions

The name of the new production is automatically generated based on the goal type. Chunk headers (appearing followed by the selector **>**) and chunks which appear more than once in the production are variabilized. Other chunks and non-chunk values are left as constant. The special dependency slots **generals**, **specifics** and **dont-cares** can be used to override this variabilization rule. Negation tests can be generated using the **differents** dependency slot.

```
(p     goaltype-production
       goal
       {constraints}*
==>
       {subgoals}*
       {modified}
       {!pop! | !focus-on! subgoal}
       {!push! subgoal}*
       {actions}*)
```

The left-hand side of the compiled production consists of the **goal** description followed by the list of **constraints**. The right-hand side consists of the list of **subgoals** and the **modified** goal if specified (only the modified part of which is actually repeated, but the whole goal counts for variabilization purposes), followed by a **!pop!** command if **success** or **failure** was specified, the respective **!push!** statements in reverse order (the first may be aggregated with the optional **!pop!** into a **!focus-on!**) and finally the **actions**.

The progress of the production compilation mechanism can be traced using **production compilation trace**, which is set by default:

Production Compilation Trace
keyword: PCT **default value: T**
If enabled, prints the main steps of the production compilation process.

The initial parameters (see Section 6) of the compiled productions can be set using the **Set-Compilation-Parameters** command:

(Set-Compilation-Parameters {*parameter value*}***)**
Sets various parameters for initializing compiled productions. The arguments are the same as for the command **Parameters** (minus *production*). Returns the argument.

Finally, the production compilation process can be applied to selected dependencies with the **whynot-dependency** command:

(WhyNot-Dependency {*dependency*}***)**

Attempts to compile *dependency* into production, with **Production Compilation Trace**, **Exact Matching Trace** and **Partial Matching Trace** on. Returns the list of productions compiled.

Section 9 Hooks

A number of hooks are provided to better monitor and/or alter ACT-R's runtime behavior. Users need to be at least moderately proficient in Lisp to use this advanced feature. The hooks are global variables which, if set to a proper function definition, can be used to transfer control of the production system to the user's code at a number of places in each run cycle. The list of hooks with their arguments and values (when applicable) is:

conflict-set-hook-fn

argument: Conflict Set **returned value:** Instantiations & Latency

The conflict set hook function is called after all the instantiations have been generated but before one is chosen by the conflict resolution mechanism. It takes one argument which is the conflict set, i.e. the list of instantiations. It should return two values: an instantiation or instantiations and a latency. If the first value returned is a list of instantiations, then that list becomes the conflict set, which is passed to the conflict resolution process as usual, and the second value is ignored. If the first value is an instantiation, then it is selected as the one to fire without calling the conflict resolution, and the second argument is interpreted as the cycle's matching latency.

firing-hook-fn

argument: Instantiation **returned value:** NIL

The firing hook function is called just before the selected instantiation is to fire, and is given the instantiation as argument.

cycle-hook-fn

argument: Instantiation **returned value:** NIL

The cycle hook function is called at the end of each cycle, and is given the instantiation just fired as argument.

end-run-hook-fn

argument: Latency **returned value:** NIL

The end run hook function is called when the run command is completed, and is given as argument the total latency of the command, i.e. the difference between the present time and the time it was called.

Ultimately I hope to document the set of internal functions used to manipulate those data structures but until then you should just poke around the code and figure it out. Hopefully, it should be simple enough, but if you have any problem feel free to contact me at cl+@cmu.edu.

Section 10 Bibliography

The main ACT-R reference is:

Anderson, J. R., & Lebiere, C. (1998). *The Atomic Components of Thought*. Hillsdale, NJ: Erlbaum.

The previous ACT-R reference was:

Anderson, J. R. (1993). *Rules of the Mind*. Hillsdale, NJ: Erlbaum.

Partial matching in ACT-R was first introduced in:

Lebiere, C., Anderson, J. R., & Reder, L. M. (1994). Error modeling in the ACT-R production system. In *Proceedings of the 16th Annual Conference of the Cognitive Science Society*, pp. 555-559. Erlbaum.

Further use of partial matching and the introduction of the constraint on total source activation can be found in:

Anderson, J. R., Reder, L. M., & Lebiere, C. (1996). Working memory: Activation limitations on retrieval. *Cognitive Psychology, 30*, 221-256.

Glossary

Activation-Sources

(Activation-Sources)
Displays the current activation sources with their level. Returns the list of sources.

Actr-Time

(Actr-Time {*delay*}**)**
Returns the current ACT-R time and, if *delay* is specified, adds it to the current time.

Add-IA

(Add-IA {**(***chunkj chunki sji***)**}***)**
Adds inter-associativity (IA) values between chunks. *Sji* is defined as the new IA value from *chunkj* to *chunki*. Returns the list of added IAs.

Add-DM

(Add-DM {**(***chunkname* **isa** *chunktype* {*slot value*}***)**}***)**
Adds chunks to declarative memory. The new chunk *chunkname* is of chunk type *chunktype*, with each *slot* being initialized to *value*. Returns the list of added chunks.

Chunk-Slot-Value

(Chunk-Slot-Value *chunk slot* **)**
Returns the value of *slot* of *chunk*.

Chunk-Type

(Chunk-Type {*type* | **(***type* **(:include** *supertype***)**} {*slot* | **(***slot value***)**}* **)**
Defines a new chunk type, which may include a *supertype* from which *type* inherits its slots, and a number of *slot*, each with an optional initial *value*. If no argument is passed to **Chunk-Type**, then a list of the existing types is printed. Return the chunk type defined or the list of all chunk types if no argument is given.

Clear-All

(Clear-All)
Clears everything from the ACT-R state, including chunk types, chunks, productions, global variables, etc. A call to **Clear-All** ***MUST*** be put at the top of each model file.

Clear-DM

(Clear-DM)
Clears all chunks from declarative memory. It is especially strongly recommended to use **Clear-All** or **Reset** instead for resetting purposes.

Clear-Goal-Stack

(Clear-Goal-Stack)
Clears the goal stack by popping all goals and restoring the top goal as the focus.

Clear-Productions

(Clear-Productions)
Clears all productions from production memory. It is strongly recommended to use **Clear-All** or **Reset** instead for resetting purposes.

Close-Output

(Close-Output)
Closes the current output stream file established by a call to **Output-Stream**.

Close-Trace

(Close-Trace)
Closes the current trace stream file established by a call to **Trace-Stream**.

Copy-Chunk

(Copy-Chunk {*chunk*}***)**
Makes a new copy of each *chunk* and returns the list of copies.

Delete-Chunk

(Delete-Chunk {*chunk*}***)**
Deletes chunks from declarative memory. Do not use if *chunk* is still used in other chunks or productions. Returns the list of deleted chunks.

DM

(DM {*chunk* }***)**
Displays the specified chunks, or the entire contents of declarative memory if none is supplied. For each *chunk*, its name is printed followed by its current activation, then its type and slot-value pairs. The current focus is displayed preceded by two asterisks ("**"). Returns the list of chunks printed.

Focus-on-Goal

(Focus-on-Goal *chunk***)**
Replaces the focus with *chunk* , similarly to the **!focus-on!** production command. Returns the new focus.

Get-Base-Level

(Get-Base-Level {*chunk*}***)**
Returns base level activation of chunks. Returns the list of base levels.

Get-Name

(Get-Name {*chunk-or-production*}***)**
Returns the list of names of chunks and/or productions.

Goal-Focus

(Goal-Focus {*chunk*}***)**
Sets the focus to *chunk* , or prints the current focus if none is supplied. If more than one *chunk* is specified, then the focus is set to the first one and the other ones are kept in a list for use by **Run-Many**. Returns the new focus.

Goal-Stack

(Goal-Stack)
Displays the focus, then the current contents of the goal stack from top to bottom. Each goal is followed by its goal value **G**. Returns the goal stack, i.e. a list of the focus followed by each goal on the stack in descending order.

Help

(Help { *command* }***)**
Displays a short description of *command*, or prints the whole list of command if none is given.

IA

(IA *chunkj chunki***)**
Displays the IA value between *chunkj* and *chunki* and return the value.

Load-Model

(Load-Model *file* {*directory*}**)**
Loads the model *file* in *directory* . If *directory* is not supplied, then *file* is considered relative to the directory of the file in which the command appears, if applicable.

Mod-Chunk

(Mod-Chunk *chunk* {*slot value*}***)**
Modifies chunk by setting each *slot* to *value*. Returns the modified chunk.

Mod-Focus

(Mod-Focus {*slot value*}***)**
Same as **Mod-Chunk**, with the chunk set to the focus. Returns the modified focus.

No-Output

(No-Output {*command*}***)**
Evaluates a number of ACT-R *command* while turning all printing off. Useful in reducing output when passing results between commands. Returns the output of the last *command*.

Output-Stream

(Output-Stream *file* {**:echo** *echo*}**)**
Causes all production output (i.e., anything produced by **!output!** commands) to be sent to the specified file. If the **:echo** keyword is supplied with a non-nil value, then the output is directed to the specified file, and a copy is also sent to the Listener window. The output stream created should always be explicitly closed using **Close-Output** before clearing, resetting or quitting ACT-R. Returns the stream.

P

(P *production lhs* **==>** *rhs***)**
Defines *production* as composed of a matching side *lhs* and an action side *rhs*. Returns the new production. See Section 6 for details.

Parameters

(Parameters *production* {*parameter value*}***)**
Sets a number of *parameter* to *value* for *production*. Each possible parameter and its current value can be obtained using the **production-parameter** command . Parameters can be omitted and appear in any order. A warning is issued if any *value* is of the wrong type or range. Returns the list of values, or **:error** for failed settings. See Section 6 for details.

PBreak

(PBreak {*production*}***)**
Sets breakpoints for productions. If *production* is about to fire, then the instantiation is displayed as if by **Production Trace**, but the production does not actually fire. Instead, a Lisp break occurs, allowing one to examine and/or change declarative memory, establish other breakpoints, etc. Then continue or abort as indicated by the Lisp environment. Use **PUnbreak** to remove breakpoints. Returns the list of productions with break points.

PDisable

(PDisable {*production*}***)**
Disables productions. *production* is not allowed to match or fire, but it can be printed by **PP** and re-enabled by **PEnable**. Returns the list of disabled productions, including failed compiled productions.

PEnable

(PEnable {*production*}***)**
Enables productions. Undoes the effect of **PDisable**. Returns the list of disabled productions, including failed compiled productions.

PMatches

(PMatches)
Prints out all production instantiations that match against the current state of declarative memory. Returns the conflict set, i.e. the list of matching productions.

Pop-Goal

(Pop-Goal)
Pops the top goal off the stack and installs it as the focus, similarly to the **!pop!** production command. Returns the new focus.

PP

(PP {*production*}***)**
Prints the specified productions or all enabled ones if none is supplied. Returns the list of productions printed.

Production-Parameter

(Production-Parameter *production* {*parameter*}***)**
Displays the value of *parameter* for *production*. If no parameter is specified, then all active (as defined by global parameters) parameters are displayed and the production is returned. If parameters are specified, then they are printed with their values and the list of values is returned. See Section 6 for details.

PSet

(PSet {*set* | *set* {*command* }* | {**(***set* {*command* }***)**}*}**)**
Parameters Set. Prints, defines and activates sets of parameters. If no argument is given, then the current parameter sets are printed. If the name of a set is given, then the commands composing that set are executed. Otherwise, one or more sets are defined by specifying the name of a new set (or an old one to be redefined) and a number of parameter-setting commands (e.g. **sgp**, **Sdp**, **spp**, **parameters**, or any other commands) composing that set.

Pstep

(Pstep {*length*}**)**
Runs the production system as with the **Run** command. Displays the list of instantiations at each cycle and asks the user to choose between stepping once more, stopping the run, running without stepping, or selecting one of the instantiations to fire.

PUnbreak

(PUnbreak {*production*}***)**
Removes breakpoints for productions. Undoes the effect of **PBreak**. If no production is given, all breakpoints are removed. Returns the list of productions with break points.

Push-Goal

(Push-Goal *chunk***)**
Pushes the focus on the goal stack and installs *chunk* as the new focus, similarly to the **!push!** production command. Returns the new focus.

Reload

(Reload {*seed*}**)**
Reloads the current model, including any updates made to the model file. A random *seed* can be provided for an initial randomization.

Reset

(Reset {*seed*}**)**
Resets ACT-R to its state after loading the last model, i.e. it undoes all interactive commands, production firings, etc. Faster than reload, but does not include updates made to the model file since the previous loading. . A random *seed* can be provided for an initial randomization.

Reset-IA

(Reset-IA)
Resets default IA values using present connectivity.

Run

(Run {*length*}**)**
Runs the production system for at most *length* cycles if *length* is an integer, *length* units of (ACT-R) time if *length* is a real value, or until no production can be instantiated, the top goal is **!pop!**ped or a **!stop!** command is issued by a production. Returns the run latency and number of cycles.

Run-Many

(Run-Many {*iteration*}**)**
Runs the production system by focusing in sequence on each chunk defined by the last **Goal-Focus** then calling **Run**, *iterations* times.

SDM

(SDM {**ISA** *type*} {*slot value*}***)**
Searches declarative memory. All chunks of type *type* (if specified, otherwise all chunks are considered) with *slot* value equal to *value* are displayed. Returns the list of chunks matching the condition(s).

Sdp

(Sdp {*specification* | {**(***specification* **)**}*}**)**
specification ::= {*chunk* | **(**{*chunk*}***)**} {{**:***parameter value*}* | {**:***parameter*}*}
Show/Set Declarative Parameters. Any number of chunk parameter specifications can be given, each enclosed in parentheses. If only one is given, then the parentheses are optional. Each specification is composed of a chunk specification, followed by a parameter specification. The chunk specification specifies the chunk, or list of chunks to which it applies. If it is omitted then it applies to all chunks. The parameter specification is similar to the argument of **sgp**. If it is omitted, then all active (as determined by the current setting of global parameters) parameters for the chunk are printed and the chunk itself is returned. If a number of chunk parameters are specified, then those parameters are printed with their current values for the chunk, and a list of the values is returned. If a number of parameter-value pairs are specified, then each chunk parameter is set to its new value, and the list of values is returned, or **:error** for failed settings. See section 4 for details.

Set-All-Base-Levels

(Set-All-Base-Levels {*baselevel* | *references* {*creation-time* } }**)**
Sets the base levels of all chunks. See **Set-Base-Levels** for parameters. Returns the new base level.

Set-Compilation-Parameters

(Set-Compilation-Parameters {*parameter value*}***)**
Sets various parameters for initializing compiled productions. The arguments are the same as for the command **Parameters** (minus *production*). Returns the argument.

Set-Base-Levels

(Set-Base-Levels {**(***chunk baselevel***)** | **(***chunk references* {*creation-time* }**)**}***)**
Sets the base level activations of chunks. If **Base Level Learning** is off, then the *base level* of *chunk* should be supplied directly. Otherwise, the number of past *references* and perhaps the *creation-time* should be provided. Returns the list of base levels.

Set-DM

(Set-DM {**(***chunkname* **isa** *chunktype* {*slot value*}***)**}***)**
Same as **Add-DM**.

Set-G

(Set-G *G* {**:threshold** *threshold*}**)**
Sets the value of the goal value *G* and perhaps the goal-threshold, as a percentage of *G*.. Provided for compatibility only. Use **sgp**.

Set-General-Base-Levels

(Set-General-Base-Levels {**(***chunk baselevel***)** | **(***chunk references creation-time***)**}***)**
Same as **Set-Base-Levels**.

Set-IA

(Set-IA {**(***chunkj chunki sji***)**}***)**
Same as **Add-IA**.

Set-Similarities

(Set-Similarities {**(***chunkj chunki similarity***)**}***)**
Sets the *similarity* value between *chunkj* and *chunki*.. Returns the list of similarity values.

Sgp

(Sgp {{**:***parameter value*}* | {**:***parameter*}*}**)**
Show/Set Global Parameters. If no argument is specified, the list of all global parameters is printed, together with their shorthand keywords for use in **sgp** and their current value (nil is returned). If a list of keywords is specified, then each *parameter* is printed with its current value, and the list of values is returned. If a list of keyword-value pairs specified, then each *parameter* is set to its *value* and the list of values is returned. A warning is printed if a *value* is of the wrong type or range for *parameter*, in which case the keyword **:error** is returned instead of the value. See Section 3 for details.

Similarity

(Similarity *chunkj chunki***)**
Displays the similarity value between *chunkj* and *chunki*.. Returns the similarity value.

Spp

(Spp {*specification* | {**(***specification* **)**}*}**)**
specification ::= {*production* | **(**{*production*}***)**} {{**:***parameter value*}* | {**:***parameter*}*} Show/Set Production Parameters. Any number of production parameter specifications can be given, each enclosed in parentheses. If only one is given, then the parentheses are optional. Each specification is composed of a production specification, followed by a parameter specification. The production specification specifies the production, or list of productions to which it applies. If it is omitted then it applies to all productions. The parameter specification is similar to the argument of **sgp**. If it is omitted, then all active (as determined by the current setting of global parameters) parameters for the production are printed and the production itself is returned. If a number of production parameters are specified, then those parameters are printed with their current values for the production, and a list of the values is returned. If a number of parameter-value pairs are specified, then each production parameter is set to its new value, and the list of values is returned, or **:error** for failed settings. See section 6 for details.

Update-Activation

(Update-activation)
Updates the activations of all chunks. This involves recomputing all the learned quantities, and helps to update all the cached values.

Trace-Stream

(Trace-Stream *file* {**:echo** *echo*}**)**
Causes all trace output to be sent to the specified file. The arguments work as for **Output-Stream**, and the resulting stream should always be closed using **Close-Trace**. A finer degree of control over trace dispatching can be achieved through the use of **sgp**. Returns the stream.

WhyNot

(WhyNot {*production*}***)**
Attempts to match *production*(s) against the current state of declarative memory. A detailed matching trace is displayed as by **Exact Matching Trace** and **Partial Matching Trace**, and if any instantiations result they are also displayed. Returns the conflict set, i.e. the list of matching productions.

WhyNot-Dependency

(WhyNot-Dependency {*dependency*}***)**
Attempts to compile *dependency* into production, with **Production Compilation Trace**, **Exact Matching Trace** and **Partial Matching Trace** on. Returns the list of productions compiled.

ActivationSources

(ActivationSources)
See **Activation-Sources**.

ActrTime

(ActrTime {*delay*}**)**
See **Actr-Time**.

AddIA

(AddIA {**(***wmej wmei sji***)**}***)**
See **Add-IA**.

AddWM

(AddWM {**(***wmename* **isa** *wmetype* {*slot value*}***)**}***)**
See **Add-DM**.

ClearAll

(ClearAll)
See **Clear-All**.

ClearGoalStack

(ClearGoalStack)
See **Clear-Goal-Stack**.

ClearProductions

(ClearProductions)
See **Clear-Productions**.

ClearWM

(ClearWM)
See **Clear-DM**.

CloseOutput

(CloseOutput)
See **Close-Output**.

CloseTrace

(CloseTrace)
See **Close-Trace**.

CopyWme

(CopyWme {*wme*}***)**
See **Copy-Chunk**.

DeleteWM

(DeleteWM {*wme*}***)**
See **Delete-Chunk**.

Focus-on-Wme

(Focus-on-Wme *wme***)**
See **Focus-on-Goal**.

GetBaseLevel

(GetBaseLevel {*wme*}***)**
See **Get-Base-Level**.

GoalStack

(GoalStack)
See **Goal-Stack**.

ModFocus

(ModFocus {*slot value*}***)**
See **Mod-Focus**.

ModWME

(ModWME *wme* {*slot value*}***)**
See **Mod-Chunk**.

OutputStream

(OutputStream *file* {**:echo** *echo*}**)**
See **Output-Stream**.

Pop-Wme

(Pop-Wme)
See **Pop-Goal**.

Push-Wme

(Push-Wme *wme***)**
See **Push-Goal**.

ResetIA

(ResetIA)
See **Reset-IA**.

SetAllBaseLevels

(SetAllBaseLevels {*baselevel* | *references* {*creation-time* } }**)**
See **Set-All-Base-Levels**.

SetAnalogizedParameters

(SetAnalogizedParameters {*parameter value*}***)**
See **Set-Compiled-Parameters**.

SetBaseLevels

(SetBaseLevels {**(***wme baselevel***)** | **(***wme references* {*creation-time* }**)**}***)**
See **Set-Base-Levels**.

SetG

(SetG *G* {**:threshold** *threshold*}**)**
See **Set-G**.

SetGeneralBaseLevels

(SetGeneralBaseLevels {**(***wme baselevel***)** | **(***wme creation-time references***)**}***)**
See **Set-General-Base-Levels**.

SetIA

(SetIA {**(***wmej wmei sji***)**}***)**
See **Set-IA**.

SetSimilarities

(SetSimilarities {**(***wmej wmei similarity***)**}***)**
See **Set-Similarities**.

SetWM

(SetWM {**(***wmename* **isa** *wmetype* {*slot value*}***)**}***)**
See **Set-DM**.

SWM

(SWM {**ISA** *type*} {*slot value*}***)**
See **SDM**.

Swp

(Swp {*specification* | {**(***specification* **)**}*}**)**
See **Sdp**.

TraceStream

(TraceStream *file* {**:echo** *echo*}**)**
See **Trace-Stream**.

WM

(WM {*wme* }***)**
See **DM**.

WMESlotValue

(WMESlotValue *wme slot* **)**
See **Chunk-Slot-Value**.

WMEType

(WMEType {*type* | **(***type* **(:include** *supertype***)**} {*slot* | **(***slot value***)**}* **)**
See **Chunk-Type**.

WMFocus

(WMFocus {*wme*}***)**
See **Goal-Focus**.

Enable Rational Analysis
keyword: ERA **default value: NIL**
Enables the rational activation and expected gain computations.

G parameter

keyword: G **default value: 20.0**

The value of the goal in the *PG-C* evaluation (ACT Equation 3.1). Must be a positive number.

Expected Gain S

keyword: EGS **default value: NIL**

If enabled, the *S*-value of the Gaussian noise added to the *PG-C* evaluation for each instantiation (ACT equation 3.4). Must be a positive number. The variance can be accessed as **EGN**.

Enable Randomness

keyword: ER **default value: NIL**

If enabled, ties among instantiations are broken randomly, rather than according to some unspecified but deterministic order.

Utility Threshold

keyword: UT **default value: 0.0**

The threshold for the utility (*PG-C*) of a production below which it will not be considered during conflict resolution. Setting it to NIL is equivalent to disabling the threshold, i.e. considering all productions.

Goal Activation

keyword: GA **default value: 1.0**

The total level of source activation, i.e. the *Wj* in ACT equation 3.5. That level is divided evenly among chunks in the slot values of the current focus (see ARL paper). Must be a positive number.

Base Level Constant

keyword: BLC **default value: 0.0**

A constant added to all chunk base levels, i.e. (ß in ACT equation 3.6). Can be any number.

Activation Noise

keyword: ANS **default value: NIL**

If enabled, the *S*-value of the Gaussian noise added to each chunk activation at every cycle. Must be a positive number. The variance (σ_2^2 in ACT equation 3.6) can be accessed as **AN**.

Permanent Activation Noise S

keyword: PAN **default value: NIL**

If enabled, the *S*-value of the Gaussian noise added to each chunk activation at creation. Must be a positive number. The variance (σ_1^2 in ACT equation 3.6) can be accessed as **PAN**.

Latency Factor

keyword: LF **default value: 1.0**

The multiplicative factor *F* in ACT equation 3.10. Must be a positive number.

Latency Exponent

keyword: LE **default value: 1.0**

The exponent factor *f* in ACT equation 3.10. Must be a positive number.

Default Action Time
keyword: DAT **default value: 0.05**
The default action time of a production, i.e. the right-hand side part of the *a* parameter in ACT equation 3.3. Must be a positive number, usually interpreted in milliseconds. *This default has been changed from 1.0 in previous ACT-R versions.*

Partial Matching
keyword: PM **default value: NIL**
Enables partial matching.

Mismatch Penalty
keyword: MP **default value: 1.5**
Penalty subtracted from the activation of a chunk for a complete mismatch in a slot pattern. Multiplied by *1.0 - similarity* for partial mismatches. Must be a positive number.

Retrieval Threshold
keyword: RT **default value: NIL**
When enabled, chunks with activation levels below this threshold cannot be retrieved. Works with both partial and exact (standard) matching. Can be any number.

Optimized Learning
keyword: OL **default value: T**
If enabled, an efficient approximation to the formulas for production strength learning and base level learning is used (see the discussion of ACT equation 4.1).

Base Level Learning
keyword: BLL **default value: NIL**
If enabled, used as *d* in ACT equation 4.1 for learning the base level activations of chunks. Must be a positive number, with 0.5 the standard value.

Associative Learning
keyword: AL **default value: NIL**
If enabled, used as *assoc* in ACT equation 4.3 for learning the associative strength between pairs of chunks. Must be a positive number.

Strength Learning
keyword: SL **default value: NIL**
If enabled, used as *d* in ACT equation 4.4 for learning the strength of productions. Must be a positive number, preferably less than 1.0.

Parameters Learning
keyword: PL **default value: NIL**
If enabled, the parameters *a*, *b*, *q* and *r* will be learned for each production according to ACT equations 4.5 and 4.6. Can also be set to the decay rate in the ACT equations 4.7 and 4.8.

Exact Matching Trace
keyword: EMT **default value: NIL**
If enabled, prints a message during production matching every time a condition is evaluated.

Partial Matching Trace
keyword: PMT **default value: NIL**
If enabled, prints a message during production matching every time a condition is evaluated when **Partial Matching** is enabled.

Analogy Trace
keyword: AT **default value: T**
If enabled, prints the main steps of the analogy process.

Activation Trace
keyword: ACT **default value: NIL**
If enabled, prints a trace of the main steps during computation of chunk activation.

Conflict Resolution Trace
keyword: CRT **default value: NIL**
If enabled, prints the name of each production as it is matched curing conflict resolution.

Conflict Set Trace
keyword: CST **default value: NIL**
If this option is enabled, prints the number of instantiations in the conflict set, and, when rational analysis is enabled, the number of instantiations that were considered and the expected gain of the instantiation that will fire.

Matches Trace
keyword: MT **default value: NIL**
If enabled, prints all the instantiations generated. A setting of SHORT will simply print the list of variables and their values for each instantiation, and a setting of T will print the whole production(s) with the variables replaced by their values.

Production Trace
keyword: PT **default value: NIL**
If enabled, prints the selected production instantiation before firing it. A setting of SHORT will simply print the list of variables and their values, and a setting of T will print the whole production with the variables replaced by their values.

Cycle Trace
keyword: CT **default value: T**
If enabled, prints the cycle number, current time and name of selected production.

Latency Trace
keyword: LT **default value: T**
If enabled, prints the latency of the production matching (when rational analysis is enabled - see ACT equations 3.10) and the latency of the action side of the production.

Output Trace
keyword: OT **default value: T**
If enabled, prints the output of **!output!** commands on the right-hand side of productions.

Declarative Memory Trace
keyword: DMT **default value: NIL**
If enabled, prints a message when a chunk is created, deleted or modified.

Goal Trace

keyword: GT **default value: NIL**

If enabled, prints a message when the focus is changed.

Verbose

keyword: V **default value: T**

If enabled, the traces are printed as enabled. If disabled, turns off all trace output regardless of which is enabled. ***Disabling this trace is useful to easily turn off all trace output when running in batch mode where performance is key.***

:Name

keyword: :name

The name of the item. Cannot be set.

:Activation

keyword: :activation **default value: 0.0**

The chunk activation, *Ai* in ACT equation 3.5. Cannot be set directly.

:Source

keyword: :source **default value: NIL**

The chunk source level, *Wj* in ACT equation 3.5. Can be set to NIL (0.0) or any number.

:Base-Level

keyword: :base-level **default value: Base Level Constant**

The chunk base level, *Bi* in ACT equation 3.5. Can be set to any number, only when **Base Level Learning** is disabled.

:Creation-Time

keyword: :creation-time **default value: 0.0**

The time at which the chunk was created. Used in **Base Level Learning** approximation equation to compute *L* (see discussion of ACT equation 4.1). Can be set to any number.

:References

keyword: :references **default value: (1.0)**

The number and, when **Optimized Learning** is off, list of production matching times of chunk. Used in ACT equation 4.1. Can be set to the number or the list of references.

:Source-Spread

keyword: :source-spread **default value: 0.0**

The spreading activation to chunk, the second term in ACT equation 3.5. Cannot be set directly.

:IAs

keyword: :ias **default value: see above section**

The Interactive Activation values to chunk *i*, *Sji* in ACT equation 3.5. Can be set to a list of chunk-number pairs used to update the IAs to *i*.

:Creation-Cycle

keyword: :creation-cycle **default value: 0.0**

The production cycle at which the chunk was created, used to compute *F* in ACT equation 4.3. Can be set to any number.

:Needed

keyword: :needed **default value: 0.0**

The number of times the chunk has been retrieved, *F(Ni)* in ACT equation 4.3. Can be set to any number.

:Context

keyword: :contexts **default value: 0.0**

The number of production cycles the chunk was in context, scaled by its source level at the time, *F(Cj)* in ACT equation 4.3. Can be set to any number.

:Similarities

keyword: :similarities **default value: ((chunk . 1.0))**

The similarity values used in partial matching (see ACT equation 3.8). Can be set to a list of chunk-number pairs used to update the similarities to chunk.

!push!

!push! *variable*

Pushes the current focus on top of the stack and replaces it with the subgoal *variable*, which must have previously bound. The previous focus can be restored by a later production using the **!pop!** command. This command is traced by **Goal Trace**.

!pop!

!pop!

Pops the current focus and restores as the focus the top goal of the stack. If the stack is empty, then the focus becomes NIL and the production system stops. This command is traced by **Goal Trace**.

!focus-on!

!focus-on! *variable*

Replaces the current focus with the goal *variable*, which must have been previously bound. The current focus is *not* saved on the stack. This command is traced by **Goal Trace**.

!restart!

!restart!

Pops all the goals on the stack and restores the top-most goal. This command is traced by **Goal Trace**.

!output!

!output! ("*output-string*" {*expression* }* | {*expression* }***)**

Prints a message to **Output Trace** , which defaults to T, i.e. the Listener window, but can be redirected to any file or turned off by being set to NIL. The message consists of *output-string*, with the special dispatchers in the string replaced by the evaluation of the expressions, in order. In most cases, the only dispatcher needed is the two-character sequence ~**S**, which tries to print the best representation of the result of *expression*. If the default format is used, then the output string can be omitted and the values to be printed can be simply listed. For other dispatchers, see the Lisp command **format**. This command is controlled by **Output Trace**.

!eval!

!eval! *expression*
expression is evaluated by LISP. This command is used strictly for the side-effects of *expression*, and unlike its left-hand side version does not test the resulting value.

!bind!

!bind! *variable expression*
expression is evaluated by Lisp and *variable* is bound to the result.

!delete!

!delete! *variable*
Deletes the chunk bound to *variable*. This command is traced by **Declarative Memory Trace**.

!copy!

!copy! *variable* {*chunk* }*
Copies each *chunk* (specified by a variable or expression) then binds *variable* to the list of chunk copies.

!stop!

!stop!
Stops the production system after this cycle.

:Name
keyword: :name
The name of the item. Cannot be set.

:Strength
keyword: :strength **default value: 0.0**
The production strength, *Sp* in ACT equation 3.10. Can be set to any number, only when **Production Strength Learning** is off.

:Creation-Time
keyword: :creation-time **default value: 0.0**
The time at which the production was created. Used in **Production Strength Learning**. Can be any number.

:References
keyword: :references **default value: (1.0)**
The number and, when **Optimized Learning** is off, list of firing times for production. Used in ACT equation 4.4. Can be set to the number or the list of references.

:Q
keyword: :q **default value: 1.0**
The estimated probability that the production will achieve its intended effect. Used to randomly determine the success of production firing if **Chance** isn't specified. Should be a probability, i.e. a number between 0.0 and 1.0.

:A

keyword: :a **default value: Default Action Time**

The estimated effort spent executing the production, usually interpreted as the amount of time. Used to increment the Time counter if **Effort** isn't specified. Should be a positive number.

:R

keyword: :r **default value: 1.0**

The estimated probability that the production will lead to eventual success. Should be a probability, i.e. a number between 0.0 and 1.0.

:B

keyword: :b **default value: 1.0**

The estimated effort spent after executing the production, usually interpreted as the amount of time. Should be a positive number.

:PG-C

keyword: :pg-c **default value: (q*r) * G - (a + b)**

The current **PG-C** value of the production when rational analysis is enabled. Cannot be set directly.

:Value

keyword: :value **default value: 0.0**

The value of the production when rational analysis is disabled. Can be any Lisp expression evaluating to a number.

:Successes

keyword: :successes **default value: 0.0**

The *m* in ACT equation 4.5 and *n* in 4.6, i.e. the actual number of successful firings of production. Is learned by **Parameters Learning**. A prior can be set to any positive number or list of times.

:Failures

keyword: :failures **default value: 0.0**

The *n* in ACT equation 4.6, i.e. the actual number of failures of production. Is learned by **Parameters Learning**. A prior can be set to any positive number or list of times.

:Efforts

keyword: :efforts **default value: 0.0**

The *sum of Ei* in ACT equation 4.6, i.e. the actual sum of efforts spent successfully executing production. Is learned by **Parameters Learning**. A prior can be set to any positive number or list of efforts.

:Eventual-Successes

keyword: :eventual-successes **default value: 0.0**

The *m* in the **r** version of ACT equation 4.5 and *n* in the **b** version of 4.6, i.e. the actual number of eventual successes. Is learned by **Parameters Learning**. A prior can be set to any positive number or list of times.

:Eventual-Failures

keyword: :eventual-failures **default value: 0.0**

The *n* in the **r** version of ACT equation 4.5, i.e. the actual number of eventual failures. Is learned by **Parameters Learning** . A prior can be set to any positive number or list of times.

:Eventual-Efforts

keyword: :eventual-efforts **default value: 0.0**

The *sum of Ei* in the **b** version of ACT equation 4.5, i.e. the actual sum of efforts spent after executing the production until reaching eventual success. Is learned by **Parameters Learning** . A prior can be set to any positive number or list of efforts.

:Q-alpha

keyword: :q-alpha **default value: 1.0**

The *alpha* in the **Parameters Learning** ACT equation 4.5, i.e. the prior number of successful firings of production. Should be a positive number. SET SUCCESSES INSTEAD.

:Q-beta

keyword: :q-beta **default value: 0.0**

The *beta* in the same equation, i.e. the prior number of failures of production. Should be a positive number. SET FAILURES INSTEAD.

:A-z

keyword: :a-z **default value: Default Action Time**

The *z* in ACT equation 4.6, i.e. the prior amount of effort spent on the production. Should be a positive number. Also **a-b** for compatibility purposes. SET EFFORTS INSTEAD

:A-v

keyword: :a-v **default value: 1 .0**

The *v* in the same equation, i.e. the prior number of firings of production. Should be a positive number. SET SUCCESSES INSTEAD.

:R-alpha

keyword: :r-alpha **default value: 1.0**

The *alpha* in the **r** version of ACT equation 4.5, i.e. the prior number of eventual successes. Should be a positive number. SET EVENTUAL-SUCCESSES INSTEAD.

:R-beta

keyword: :r-beta **default value: 0.0**

The *beta* in the same equation, i.e. the prior number of eventual failures. Should be a positive number. SET EVENTUAL-FAILURES INSTEAD.

:B-z

keyword: :b-z **default value: 1.0**

The *z* in the **b** version of ACT equation 4.6, i.e. the prior amount of effort spent after firing the production. Should be a positive number. Also **b-b** for compatibility purposes. SET EVENTUAL-EFFORTS INSTEAD.

:B-v

keyword: :b-v **default value: 1.0**

The *v* in the same equation, i.e. the prior number of firings of production. Should be a positive number. SET EVENTUAL-SUCCESSES INSTEAD.

:Chance

keyword: :chance **default value: NIL**

The "real-world" probability that the production will achieve its intended effect. Used to randomly determine the success of production firing. **Q** will be learned from this quantity if learning is enabled. Can be any Lisp expression evaluating to a probability.

:Effort

keyword: :effort **default value: NIL**

The "real-world" effort spent executing the production, usually interpreted as the amount of time. Used to increment the Time counter. **A** and **B** will be learned from this quantity if learning is enabled. Can be any Lisp expression evaluating to a positive number.

:Success

keyword: :success **default value: NIL**

When set to T, defines the production to be the end of a successful run, leading to the learning of **R** and **B** if **Parameters Learning** is enabled. Can be any Lisp expression.

:Failure

keyword: :failure **default value: NIL**

When set to T, defines the production to be the end of a failed run. If both **Success** and **Failure** are set in the same production, then that production ends the run, i.e. clears the run history, without either success and failure. Can be any Lisp expression.

conflict-set-hook-fn

argument: Conflict Set **returned value:** Instantiations & Latency

The conflict set hook function is called after all the instantiations have been generated but before one is chosen by the conflict resolution mechanism. It takes one argument which is the conflict set, i.e. the list of instantiations. It should return two values: an instantiation or instantiations and a latency. If the first value returned is a list of instantiations, then that list becomes the conflict set, which is passed to the conflict resolution process as usual, and the second value is ignored. If the first value is an instantiation, then it is selected as the one to fire without calling the conflict resolution, and the second argument is interpreted as the cycle's matching latency.

firing-hook-fn

argument: Instantiation **returned value:** NIL

The firing hook function is called just before the selected instantiation is to fire, and is given the instantiation as argument.

cycle-hook-fn

argument: Instantiation **returned value:** NIL

The cycle hook function is called at the end of each cycle, and is given the instantiation just fired as argument.

end-run-hook-fn

argument: Latency **returned value:** NIL

The end run hook function is called when the run command is completed, and is given as argument the total latency of the command, i.e. the difference between the present time and the time it was called.