The Manual of CLARION software system

by

Xi Zhang

March 2005

Contents

Chapter 1

Guide For Programmers

Author: Yizchak Naveh-Benjamin

(Last modified: 8/2/2003)

Newcomers to the CLARION package may find it difficult to write their own task code from scratch. The purpose of this document is to clarify potentially confusing aspects of the programmer's interface. For a more concrete demonstration, see the XOR Task (provided along with the CLARION package), which is a simple, heavily-documented task meant to serve as a springboard for writing your own code.

Note that this tutorial only covers the basics of implementing a task. Only the action-centered subsystem (ACS) is considered; other, more advanced components are not discussed.

1.1 BASICS

Implementing a task in the CLARION package involves two basic steps:

1. Writing the task code.

   This involves extending the three classes Task.java, TaskAgent.java, and TaskClarion.java, and then adding your own custom code.

2. Creating a configuration folder for each type of agent in your task.

   This allows agent settings to be saved and loaded automatically for each run.

Begin by creating a copy of the XOR task directory (sourceCode \ tasksimulators \ newsimu \ XOR). Place it inside the "newsimu" folder and change its name to that of your task. I suggest that you begin with the XOR task, rather than with the three base files (Task.Java, TaskAgent.java, and TaskClarion.java), since the XOR task is an already-working piece of code with the minimum set of necessary methods implemented. However, as you continue to develop your task, you will undoubtedly need to refer to the three base classes as well.

Next, create a directory for your task configuration files. Duplicate the default setting directory (taskSetting \ DEFAULT), rename it to reflect your task's name, and place it inside the taskSetting directory. Inside your new directory (taskSetting \ yourDirName), you will find a separate folder for each agent type. Initially, there will be only one agent group, subjectGroup0. If your task requires more than one type of agent, you will later have to add more groups to your configuration folder.

1.2 WRITING THE TASK CODE

1.2.1 Task.java

The purpose of the Task class, roughly, is to act as the experimenter in an empirical study: it defines the input the agents receive at each step, collects their output at each step, and controls any interaction between the agents. To implement this class, follow these steps:

- Extend the Task java

  (or, if using the XOR task as a base, rename the TaskXOR appropriately.

You may also need to change the package declaration at the top of your file to tasksimulators.newsimu.¡YourTaskDirectoryName¿. This also applies to TaskAgent.java and TaskClarion.java).

- Implement the SimuTask interface, including its two methods, start() and getTaskName().

  This is necessary if you want to be able to run your task through the GUI.

- Create a default (parameter-less) constructor for your task.

  Here you should set up the data structures, etc. for your task. At a minimum, you should set the correct number of running agents (by calling setRunningAgentNum). You should NOT configure the agents themselves (their learning rate, etc.) at this point, since that will be done either via the GUI, or by the init() method if the GUI isn't used to configure the agents.

- Override the init() method to configure your individual agents.

  This method will automatically be called at the start of the task, unless you use the GUI to interactively configure the agents. At a minimum, you should initialize the "settings" array (defined in Task.java) here; "settings" should be initialized to a String array containing as many Strings as your task has agents, where each String represents the configuration file used to initialize that agent. For example, if your experiment has two agents of type "subordinate," and one agent of type "supervisor," then you should initialize "settings" as follows:

  String newSettings[] = "subordinate", "subordinate", "supervisor"; settings = newSettings;

- Then, you should call super.init() to load the settings from the files (into the

"globals" array, defined in Task.java).

- Finally, implement the start() method. This method defines how the task itself works. It is called automatically at the start of a task. At a minimum, you should instantiate "agents" (defined in Task.java) as an array of TaskAgents, and instantiate each agent using the correct setting in the "globals" array. Continuing the example above, you would initialize the subordinates and supervisors as follows:

  agents[0] = new MyAgent(globals[0]); // sub

  agents[1] = new MyAgent(globals[0]); // sub

  agents[2] = new MyAgent(globals[1]); // super

  See the XOR task for an example of how to implement start().

1.2.2 TaskAgent.java

TaskAgent, roughly, describes the behavior of a subject in an experiment. To implement it, follow these steps:

- Extend the TaskAgent class (or rename the TaskAgentXOR class if you are using the XOR task as a base).

- Write a default (parameter-less) constructor for the class. This constructor can be empty, and you will likely not be calling it. However, it is necessary for your program to work.

- Write a constructor that takes a "Global" parameter; for example, MyAgent(Global g) ...

The constructor should begin by calling super(g). Then, any other initializations should be performed. At a minimum, you should initialize the curSensoryInput array (which defines what the agent will see at each step; it is an array of Features) to the correct number of input dimensions for your task.

- You should implement a method that allows Task.java to set the input the agent will see (since curSensoryInput is declared private, and hence inaccessible outside the class).

  In this method, you will have to instantiate individual Features in the curSensoryInput array. This will probably require consulting the constructor definitions in the Feature class.

- Next, you should implement the getOutput method, which accepts an array (as an Object), and writes the current output to it.

  Be sure to typecast the parameter to an array of the correct type before writing to it.

- You should now implement the performAction method, which is called automatically after each invocation of run().

  This method should, at a minimum, set the output to be returned by the agent.

- You will need to implement the various abstract methods from TaskAgent.java: determineAmnNet(), isControlNacsAction(), and setRetrievalResults().

  Depending on your task, it may be possible to leave their implementation empty. See the TaskAgent class for more details.

- Finally, you should implement the run() method.

This method defines what the agent does at each step of the task. Somewhere in this method, you should call either clarion.training() (if this is a learning run), or clarion.testing() (if this is a testing run).

1.3 TaskClarion.java

This class defines the task-specific aspects of the Clarion model's behavior.

- First, you should implement the getEligibility() method, which determines which IDN networks are eligible at each step of the task. This method is described in detail in TaskClarion.java and TaskClarionXOR.java, so we do not repeat it here.

- You should also implement the getReinforcement() method. At a minimum, this method should read the most recent input seen by the agent (supplied a parameter to this function), compare it to the action taken by the agent (likewise supplied as a parameter), and compute a corresponding reward.

1.4 CREATING CONFIGURATION FILES

Once you have created one configuration folder for each agent type (by copying the default configuration; see Section I), it is time to configure your task. Since this is relatively easy to do using the GUI, we will only point out a few potentially troublesome spots:

- In general, the interface requires that you confirm ALL textual entries by pressing Return.

  This means that whenever you enter text into a text field, you MUST press Return afterwards!

- If your task involves more than one agent, you must enter the number of agents in the main task screen.

  Currently, this setting is not saved, so you will have to re-enter it each time you run the task.

- In the screen labeled "Clarion Parameter Settings," pressing the "Setting Group" button will bring up a list of all agent types for the current task.

- To save a setting, enter a new setting name in the text field to the right of the "Save to Files" button, and press Return. This activates the button.

- Do not overwrite the current setting (i.e., save under the name of the current setting); due to a bug, this does not currently work. Instead, save under a different name, then manually open your taskSettings folder, and rename the new setting under the original name.

Chapter 2

Task Simulations Using the System

2.1 The Overall Process of Task Simulation

The Overall process of task simulation has two stages. The first stage is to load a task you want to simulate and the second stage is to online configure the loaded task (only for with Graphical User Interface (GUI) mode) and then execute it. The configuration of a task without GUI mode is done offline before CLARION system is started.

2.2 Task Loading

After CLARION system is initially started, the first screen titled 'CLARION MODEL' appears. Now, the system is ready to load a task.

1. On the upper part of this screen allows you to select a task to load for simulating by click the "LOAD A TASK" button. Once the name of the loaded task is shown on the right to the button, that means the task loading is finished.

2. Once a task is loaded, the "NEW TASK SIMULATOR" is enabled. Click it to enter into task simulator screen.

2.3 Configuration and Execution of A Task

Generally, there are seven steps in a logic order to allow you to configure and run a task simulation using this newly designed interface. You have one focus at one time and should confirm your selection at current step before you can move to the next step. Every time, there is only one step is highlighted and enabled which allows you to make a selection and the other steps are disabled which prevents you from doing selection.

This new interface is more guidable than old version and makes the flow control easy to implement and maintain. It removes user's randomness of selecting across steps with potential inter- dependency which makes the situation more complicated.

The task configuration has two parts, one for CLARION itself configuration if you want CLARION to simulate your task and the other is task-specific configuration. The CLARION configuration is optional and the task-specific is mandatory.

To make the process of configuration much more easier. We strongly recommend you to write a configuration template for your task which is composed by a set of normal files in .txt format which contents conform to format CLARION defines. Once you finish it and put them in the proper directory and in the proper hierarchy CLARION defines. See manual on the details. Also there are many samples of such configuration files, see them to get a better and deeper knowledge.

The reason why the above template is recommended because it is a fast way of configuration once you know well the pre-defined format which actually is very easy to grasp. Base on this template, you can derive various variations as many as you want. The configuration of CLARION itself is very complicated if start from scratch without a template at hand. Once you have a template for your task at your hand. the configuration become much easier: you can load it from files to GUI

frames when you are in 'configure with GUI' mode, then tailor it for configuration of each particular agent. Another reason is that although the system can provide you a default configuration automatically for your task in the case that you don't have a particular template for your task and sometimes it can reduce your work significantly if your task is not very complicated, it is far from being accurate for you own task and it has still a lot to do.

In the following, we will introduce the seven steps in details step by step.

1. Step 1 : You need to input an eligible number of agents running simultaneously from 1 to 100 in the text field and confirm it by clicking the corresponding 'confirm' button on the right side. This is important configuration parameter because current new version allows multiple agents run simultaneously. Make sure your task should run in a multiple-agents mode. In the further steps, the system will create a package of GUI frames for you to configure every agent's individual characteristics according to this number you provide.

2. Step 2. Control mode selection. You need to make a decision on simulation control between 'BY HUMAN', 'BY CLARION' AND 'BY ACT-R'. 'BY CLARION' is focus of this step and also the overall software system. 'BY CLARION' means the task will be simulated based on CLARION cognitive model. 'BY ACT-R' means the task will be simulated based on ACT-R model which is not implemented yet in current version. 'BY HUMAN' means the task will be run by simulating human's behaviors in the experiment which is task-specific and only Process Control Task is implemented for function. Once your decision is made, then user the 'confirm' button on the right side to confirm. The following Step 5 of CLARION configuration by GUI be depended on the selection on this step.

3. Step 3. Configuration mode selection. In this step, you need to choose 'configure with GUI' or 'configure without GUI' to implement your task configuration. When you select the former, a package of GUI frames will be provided in the further step to allow you to configure your task more visually. And when you select the latter, you need to do more program work in advance. See Manual on the details of know how to do it using this mode. Confirm your decision by the 'confirm' button on the right side.

4. Step 4. task-specific configuration with GUI. To do it, click the button 'Go To Task Specific GUI'. Then you will be provided a GUI frame which is the main interface for your task specific configuration. Currently, this part of configuration is much simpler than the CLARION configuration. Confirm your configuration by clicking the 'confirm' button on the right side.

5. Step 5. CLARION configuration with GUI (On-line configuration). It is the most important part of configuration process. Similar to the previous step, to do it, click button 'Go To CLARION GUI'. Then you will be provided a GUI frame which is the main interface for the CLARION configuration. Currently, this part is very complicated and time- consuming. This part of interface is composed of 20 to 30 pages in a hierarchical way according to the architecture of CLARION model. The exact number of pages is depended on your task. Although the system has the function for dependency check across different options, you still should be very careful about your choice of each option. Again, confirm your configuration by clicking the 'confirm' button on the right side.

6. One particular note should be paid enough attention to the reentrant attribute

of both parts of configurations. This attribute allow you do reconfiguration by clicking the corresponding button again. Once you reenter into the GUI, your previous work is automatically invalidated.

7. Step 6. Selection of output data file. Your final simulation data will be written into this file unless you provide some other files in somewhere in your program code to override it. To do it, there are two ways. one is by clicking on the 'Select Output File' button, this is a more visual way. The other is to input an eligible file name in the text field below the the button. In the first way, once you are finished with an file selected, the 'confirm' button is enabled for you to confirm it. At the same time, the file name is shown in the text field and then the text field becomes not editable and disabled. In the second way, you can also enable the 'confirm' button by input a ENTER key after an eligible file name. Again, click 'confirm' button on the right side to confirm your file name. The data will be stored in a text format and can be read by word or wordpad in windows and vi in unix and linux and other text viewers can also read it. Notepad has some problem viewing it.

8. Step 7. Run your task, exit the system or go to step 1 by clicking the corresponding buttons. Once your task is finished with running, you can go to step 1 to make another round of configuration and run process. This is particularly useful for the task with different groups of agents indicating different experimental conditions.

2.4 Configuration of CLARION

CLARION has many options needs to be set. Although these options are complicated and it is time-consuming to set them, they are so powerful and flexible that make simulating a task easy. So, this step is very important. There are two ways to make the CLARION option setting properly.

2.4.1 The Process of configuration With GUI mode

It is also called on-line configuration mode because only after the CLARION system is started, you can use this mode to configure a task. With this mode, you should manually set all of the CLARION options.

1. By clicking the 'Go To CLARION GUI' button in above task simulator screen, you will enter into the main interface of GUI. This main interface is only for one running agent at a time. In this main interface, there are several buttons on the bottom part.

2. First, you need load a setting group for your own task. The system assumes that there are some setting groups already put under your task directory which is a sub-directory of 'taskSetting'.

   To do it, you can either by clicking "Setting Group" button or inputting a setting group name in the text field just right to the 'Setting Group' button. If you click the 'Setting Group' button, all of the existing setting groups for your task will be shown in a pop-up window. After you choose one group from the pop-up window, the name of the chosen setting group will be shown in the text field right to the button. Also, you can input the name of an eligible setting group into the text field and click 'ENTER' key to confirm your input.

You can change your mind repeatedly in either way. After a setting group is loaded, all of the buttons representing relevant CLARION options in center of the interface are enabled. Now, the interface is ready for you to configure your task manually.

3. Manually configure your task by clicking the corresponding button to enter into a CLARION subsystem. The structure of overall GUI for CLARION are constructed according the description of CLARION model. The options is a hierarchical structure. Different options is in different screens. It is not necessary to change the options and when you do some changes, make sure every option value is correctly set.

4. Confirm what you have done when you finish setting the options by returning to the main GUI interface and clicking the "CONFIRM" button.

   That is, make sure your selections are made and what you have done will be saved into Global.java for system use. "CONFIRM" button can be enabled either by loading an existing setting group which needs be confirmed. or by any change on the options you have made. (this confirmation step is required by the software system).

5. Once you have confirmed your configuration, Save your configuration as a new setting group for later use by clicking the "SAVE TO FILES" button just right to the 'CONFIRM' button.(this step is optional),

   "SAVE TO FILES" button can be enabled by inputting a valid setting name (a normal directory name) followed by an ENTER key to confirm it.

6. Now, the configuration for current agent is finished. You have to do the same configuration process for the next agent if exists by clicking the 'NEXT'

button. The 'NEXT' button may or may not appear depending on whether current agent is last agent or not.

7. Once you have finished configurations for all of the running agents, the 'RETURN' button appears and is enabled. Now, you can return to the task simulator screen to continue selection of your output data file.

2.4.2 The Process of Configuration Without GUI mode

It is also called off-line configuration mode because all of the configuration work is done without starting the CLARION system. In this mode, you don't need set all of the CLARION options by GUI manually, but you need do a lot of preparation work, for example, all of the CLARION options should be initialized by running your own codes.

1. Prepare the configuration files.

   Copy an existing setting group of the simulated task or copy the system default setting group if the task is new. then, look at each option file and change option values if necessary.

2. Designate a setting group for each running agent.

   To achieve it, initializes the variable "setttingNames" in Task.java. this variable is used for storing all of the names of the setting option groups each corresponding to a running agent, you can specify all of the names of the setting option groups for the running agents.

3. Initialize the variables in Global.java assigned to each running agent.

   Override the method named "init()" in Task.java in your code file named like "TaskPC.java". The method can be used to initialize all of the variables in

Global.java. Actually, finally, all of the CLARION option settings will be stored into the corresponding variables in Global.java.

4. Once all the necessary variables in Global.java are populated, that means, the configuration process is done. This statement is applied to configuration either with GUI or without GUI

5. Now, the offline preparation work is done. The only thing left is to start the CLARION system and follow the instructions in the above 'Configuration and Execution of A Task' section to execute your task.

2.5 Implementation of the Actual Process in Your Own Task

Once you finish task configuration, you can run a task. When click the button "RUN TASK" in the task simulator screen, the actual process of your task will be executed by calling your own implementation codes. Because your actual task process is task-specific, so you have to implement this process programmatically.

1. Implement three classes.

   - a task template named "Task.java".

     Task.java focuses on the procedure of a specific task experiment and saving experiment data.

   - a task agent template "TaskAgent.java".

     TaskAgent.java, like a subject in an experiment, focuses on task simulator's communicating with CLARION model.

   - a task clarion template "TaskClarion.java".

TaskClarion.java implements the task-specific CLARION options.

The above three classes are required since there are many methods especially in TaskAgent.java and TaskClarion.java need be implemented by user's codes in order for CLARION system to run properly. Their combination function as a task simulator and they are located in a subdirectory – "sourceCode \ clarion \ system".

2. After you implement the three templates and add your implementation codes to the subdirectory - "sourceCode \ simulators \ newsimu" where all of the tasks so far implemented are located.

   the above step 1 and 2, should be completed off-line before CLARION system is started. That is, you should finish your implementation codes and put them in the correct location before running CLARION system. Your implementation codes will be loaded during task loading stage.

2.5.1 Important methods in the classes for task simulator

- init() in Task.java

  for Global.java variables initialization.

- start() in Task.java

  the point where a simulated task starts to run.

- run() in TaskAgent.java

  the point where a simulated subject starts to run.

- The codes in TaskClarion.java are all samples for user's task-specific clarion options.

2.6 The simulated tasks

There are several tasks so far simulated and the corresponding user's codes lie in the directory:

"sourceCode \ simulators \ newsimu"

For all users, please run the tasks: XOR or PC to get a sense of how CLARION works.

For the programmers, please take as reference the user's codes for those simulated tasks and also see the file named "ClarionForProgrammer.txt" to get some sense of how to encode a task simulator.

2.7 The Reason implementation of the templates

It is a good way to run a simulated task and makes easier using CLARION to control since the functions of the two templates are task-specific and it is convenient for user to write the task-specific codes freely.

2.8 IMPORTANT notes for this version.

Currently, when you want to change some option value by typing something, your typing have to be ended by a ENTER key to confirm your typing. This inconvenience may disappear in later version.

Chapter 3

The Structure and Format of Configuration Files

3.1 The Structure

Under the root directory, CLARION option setting (configuration) has its own directory – "taskSetting" where there are several sub-directory each corresponding to a simulated task. Under each task, there exist a set of setting groups for this particular task. The structure of each setting group is hierarchical corresponding to the interface structure of configuration by GUI. This structure conforms strictly to the CLARION model description. For better understanding, please reference the CLARION tutorial under the documentation directory.

3.2 The Format

The format of configuration files are defined by CLARION system and basically it is a normal text file with some tags embedded.

- Each configuration file is composed in the following order: the number of options in this files and followed by a set of *options*.
- The information of each *Option* is composed in the following order: option

label, the number of choices in this option, the default choice for this option, the user choice for this option and then followed by a set of *Option Choice*.

- The information of each *Option Choice* is composed by the following order: item label, the number of parameters in this choice and then followed by a set of *Parameters*.

- The information of each *Parameter* is composed in the following order: the parameter major type, the parameter minor type, the parameter label and parameter value.

  The parameter major type indicate whether the type of the parameter is a primitive type (encoded by 0) or composite (array)(encoded by 1). The Parameter minor type indicate a parameter is classified as one of the following: boolean type (encoded by 0), integer type (encoded by 1), float(double) type (encoded by 2) or string type (encoded by 3). The parameter value is an eligible value in the corresponding to the major and minor type. If the parameter is composite (array) type, the value is a value list separated by ',' .

3.3 The encoding of Dimensions

Currently in CLARION system, there are four and only four types of dimensions : system defined dimensions (encoded as 0), input dimensions (encoded as 1), output dimensions (encoded as 2) and goal dimensions (encoded as 3). In CLARION, each dimension is identified by its type and its order in that type of dimension (must start from 0). So, users can specify each of their own dimensions by giving the dimension type and dimension order. Users must conform strictly to the CLARION encoding format in order to make system running properly. Disobey of this encoding format

may result in serious system problem such as system crash.

3.4 One Sample

The following is a sample part from configuration file for IDN.

The number of options : 10

Option label : INPUT DIMENSION INFO

Option choice number : 1

Default choice : 0

User choice : 0

Item label : info

Parameter number : 3

Parameter major type : 0

Parameter minor type : 0

Parameter label : allow system specify

Parameter value : false

Parameter major type : 1

Parameter minor type : 1

Parameter label : dimension types

Parameter value : 1

Parameter major type : 1

Parameter minor type : 1

Parameter label : dimension indices

Parameter value : 0

Chapter 4

ACS action formats

It includes all action dimensions as output of all action types and allows user selection through GUI for each IDN/rule group.

4.1 Current ACS action types in CLARION

1. external actions
2. goal structure actions
3. working memory actions
4. NACS control actions (part of external action)

4.2 General format of each action type

Each action may have 3 parts:

1. the TYPE dimension

   indicates the action is an external action, GS action, WM action or NACS control action.

   element 0 : external action

element 1 : GS action

element 2 : WM action

element 3 : NACS control action

2. dimension(s) for action specification

   defines what the action will do.

3. dimension(s) for parameters

   some actions may use parameters when they are performed.

4.3 General format of each parameter dimension

a "signal value" is added into the end of each of the parameter dimensions to indicate if the dimension allows multiple active values or not at current step. (in the following action formats, such "signal value"s are not shown.)

4.4 Unified action format

Currently, we use a 3-dimensional array to represent the unified action format.

dimension 1 : indicates action type: External, GS, WM or NACS control.

dimension 2 : indicates a dimension index on a specific action type.

dimension 3 : stores currently activated values in the specific dimension.

4.5 Format of each action type

4.5.1 External actions

Since this kind of action is task-specific, the format of external actions should be specified by user. User should specify all of the dimensions in the action format by

giving each dimension the name, the number of values.

4.5.2 Goal structure actions

This kind of action is for internal use, that is, for CLARION system use. the format is the following:

- dimension 0 : action type

  set the element indicating GS action.

- dimension 1 : action specification

  element 0 : DO-NOTHING

  element 1 : SET

  element 2 : RESET

- dimension 2 : goal dimension

  element 0 : goal 0 is currently active . . . .

  element n-1 : goal n-1 is currently active

  (n is the number of goals user defines)

- other parameter dimensions

  these dimensions are for the goal parameters along with the currently active goal. Since the goal parameters are task-specific, so, these dimensions should be defined by user. They are maybe empty sometimes.

4.5.3 Working memory actions

Also, this kind of action is for internal use. The format is the following:

- dimension 0 : action type

  set the element indicating WM action.

- dimension 1 : action specification

  element 0 : DO-NOTHING

  element 1 : SET

  element 2 : RESET (one slot)

  element 3 : RESET-ALL

- dimension 2 : the set of WM slots for storing data.

  element 0 : WM slot 0 is involved

  element 1 : WM slot 1 is involved . . . .

  element i : WM slot i is involved . . . .

  element n-1 : WM slot n-1 is involved

  (n is the size of WM)

  Actually, the actions: DO-NOTHING and RESET-ALL needn't parameters. Only the actions SET and RESET use the dimension 4 as parameters.

4.5.4 NACS control actions

the format is the following:

- dimension for action type

  dim0: action type

  set the element indicating NACS control action.

- subaction dimensions.

  In this part, each dimension is a subaction. So, the control action actually is composed of a series of subactions.

  dim1: activate NACS or report

  element 0 - not activate NACS and not report

  element 1 - activate NACS

  element 2 - report to external destination from the retrieval buffer of NACS

  dim2: choose level

  element 0 - choose GKS

  element 1 - choose AMNs

  element 2 - choose both

  dim3: encode externally given knowledge

  element 0 - do nothing

  element 1 - encodes as associative rules

  dim4: assimilate explicit knowledge

  element 0 - do nothing

  element 1 - assimilate

  dim5: do inference & retrieval

  element 0 - do nothing

  element 1 - retrieve one chunk from NACS above a threshold by Boltzmann distribution

  element 2 - retrieve all chunks from NACS above a threshold

dim6: set chunk strength

element 0 - do nothing

element 1 - set strength

- parameter dimensions.

  This part is to help the control action work properly.

  dim7: format of input to NACS (using input to ACS)

  the length of this dimension = number of dimensions of input to ACS. if dim7(i) > 0.5, then the ith dimension of input to ACS is input to NACS

  dim8: format of input to NACS (using output from ACS)

  the length of this dimension = number of dimensions of output from ACS. if dim8(i) > 0.5, then the ith dimension of output from ACS is input to NACS

  dim9: format of output from NACS (using input to ACS)

  the length of this dimension = number of dimensions of input to ACS. if dim9(i) > 0.5, then the ith dimension of input to ACS is output from NACS.

  dim10: format of output from NACS (using output from ACS)

  the length of this dimension = number of dimensions of output from ACS. if dim10(i) > 0.5, then the ith dimension of output from ACS is output from NACS.

  dim11: GKS reasoning method

  element 0 - forward reasoning

  element 1 - forward chaining with similarity-based reasoning

  dim12: number of iterations of GKS reasoning

(let maximum number of iterations be 10)

element 0 - number of iterations is 1.

element 1 - number of iterations is 2.

. . .

element 9 - number of iterations is 10.

element 10 - number of iterations is unlimited.

dim13: number of AMN passes for each iteration

(let maximum number of passes be 10)

element 0 - number of passes is 1.

element 1 - number of passes is 2.

. . .

element 9 - number of passes is 10.

dim14: involved AMNs

element 0 - AMN 0 is involved

element 1 - AMN 1 is involved

. . .

element n-1 - AMN n-1 is involved (n is the number of AMN)

dim15: type of chunk to store into retrieval buffer

element 0 - all chunk types

element 1 - state related chunks

element 2 - state related chunks not involved in the input to NACS.

element 3 - action related chunks

element 4 - action related chunks not involved in the input to NACS

dim16: the chunk to set strength of

element 0 - chunk 0 in GKS

element 1 - chunk 1 in GKS

. . .

element n-1 - chunk n-1 in GKS

(n is the chunk number in GKS)

dim17: strength level to set at

element 0 - 0.0

element 1 - 0.1

. . .

element 10 - 1.0

Chapter 5

List of Option Dependency

(note: ENABLED means the option can be set)

1. WM IDN ENABLED

IF WM = on

2. GS IDN ENABLED

IF GS-TYPE ≠ empty

3. WM IDN options ENABLED

IF WM IDN = on

4. GS IDN options ENABLED

IF GS IDN = on

5. Reinforcement-functions-for-each-network ENABLED

IF reinforcement function = one function for each network

6. Reinforcement-functions-for-all-networks ENABLED

IF reinforcement function = one function for all

7-1. Reinforcement-function-for-WM-actions ENABLED

IF WM = on

7-2. Reinforcement-function-for-GS-actions ENABLED

IF GS-TYPE != empty

8. WM ARS options ENABLED

IF WM = on

9. GS ARS options ENABLED

IF GS-TYPE != empty

10. WM rule learning options ENABLED

IF WM = on

11. GS rule learning options ENABLED

IF GS-TYPE != empty

12. WM rule representation options ENABLED

IF WM = on

13. GS rule representation options ENABLED

IF GS-TYPE != empty

14. RER options ENABLED

IF RER = on

15. IRL options ENABLED

IF IRL = on

16. FR options ENABLED

IF FR = on

17. Weights of action rule conditions ENABLED

IF rule support = on

18. action rule bla ENABLED

IF rule bla = on

19. action chunk bla ENABLED

IF action chunk bla = on

20. rule utility ENABLED

IF rule utility = on

21. goal structure options ENABLED

IF GS-TYPE != empty

22. goal threshold ENABLED

IF GS-TYPE = goal list

23. goal bla ENABLED

IF GS-TYPE = goal list

24. WM options ENABLED

IF WM = on

25. flag number ENABLED

IF flags in WM = on

26. Integration of outcomes of the 2 levels ENABLED

IF cross level combination determination = not by the MCS

27-1. Stochastic selection parameters ENABLED

IF Integration of outcomes of the 2 levels = stochastic selection

27-2. Bottom-up verification ENABLED

IF Integration of outcomes of the 2 levels = bottom-up verification 27-3. Top-down guidance ENABLED IF Integration of outcomes of the 2 levels = top-down guidance

28. GKS options ENABLED

IF GKS = on

29. EM options ENABLED

IF EM = on

30. AEM options ENABLED

IF AEM = on

31. weights of associative rule conditions ENABLED

IF rule support = on

32. associative rule bla ENABLED

IF associative rule bla = on

33. NACS chunk bla ENABLED

IF NACS chunk bla = on

34. action-oriented EM bla ENABLED

IF EM = on

35. non-action-oriented EM bla ENABLED

IF EM = on

36. EM thresholds ENABLED

IF EM = on

37. EM use parameters ENABLED

IF EM = on

38. similarity-based reasoning options ENABLED

IF SBR = on

39. AMN types ENABLED

IF number of AMN ¿ 0

40. number of iterations in using AMN ENABLED

IF using AMN = multiple pass

41. AAM assimilation options ENABLED

IF AMN type = AAM

42. HAM assimilation options ENABLED

IF AMN type = HAM

43. AEM options ENABLED

IF AEM = on

44. action-directed-reasoning options ENABLED

IF coordinating ACS/NACS = action-directed-reasoning

45. MS options ENABLED

IF MS = on

46. MCS options ENABLED

IF MCS = on

47. RT options ENABLED

IF RT = on

48. Boltzmann temperatures for overall action selection ENABLED

IF weight-sum method = on

49-1. Temperature for top-level action selection ENABLED

IF integration of outcomes of the 2 levels = stochastic selection

49-2. Temperature for bottom-level action selection ENABLED

IF integration of outcomes of the 2 levels = stochastic selection

50. Goal actions by top level only ENABLED

IF FR corresponding to GS IDN = on

51. WM actions by top level only ENABLED

IF FR corresponding to WM IDN = on

52. Goal actions by both levels ENABLED

IF GS IDN = on and FR corresponding to GS IDN = on

53. WM actions by both levels ENABLED

IF WM IDN = on and FR corresponding to WM IDN = on

54. associative rule extraction parameters ENABLED

IF associative rule extraction = on

55. NACS chunk extraction parameters ENABLED

IF NACS chunk extraction from AMN = on

56. associative rule encoding parameters ENABLED

IF encoding associative rules = on

57. NACS chunk encoding parameters ENABLED

IF encoding NACS chunks = on

58. partial matching ENABLED

IF rule support = on

59. number of ACS networks = number of External networks + number of GS networks + number of WM networks

60. $P_{BL} + P_{RER} + P_{IRL} + P_{FR} = 1.0$,

each should be no less than 0.0.

61. $P_{EXT} + P_{GS} = 1.0$,

each should be no less than 0.0

62. the sum of weights of rule condition = 1.0,

each weight should be no less than 0.0

63. $threshold_1 \geq threshold_2$, $threshold_1 \geq 0.0$

64. $threshold_6 \geq threshold_7$, $threshold_6 \geq 0.0$

Chapter 6

The missing default values

6.1 Default values Never used so far

- the goal threshold

$threshold_g = 0.2$

- the reasoning threshold

$threshold_r$ is not defined.

Default values used in MCM, LOK task

- the WM threshold

$threshold_{WM} = 0.2$

Default values used in PC task

- IRL deletion threshold

$threshold_4 = 0.22$ (in control group)

$threshold_4 = 0.4$ (in original group, memory group, sample rule group)

6.2 Default values used in AGL task

- DT_{TL}

DT_{TL} = operation-time + t0 * 1 / rule-bla + t1 * 1 / chunk-bla

operation-time = 500

t0 = 50

t1 = 50

- associative rule applicaiton time in GKS

t_a = t4 + t5 * 1 / rule-bla

t4 = 50

t5 = 50

- chunk retrieval time

t_c = t2 + t3 * 1 / chunk-bla

t2 = 50

t3 = 50

6.3 Derivation

$threshold_{sim} < threshold_r < threshold_{ce}$

when SBR is applied,

one conclusion is drawn → the strength level of the conclusion > $threshold_{sim}$

chunks are sent back to ACS → the strength level of the chunks > $threshold_r$

chunks are sent back to ACS → some conclusions are already drawn → the strength level of the chunks > $threshold_{sim}$

1. We assume $threshold_r < threshold_{sim}$, there may exist some chunks, assumed c1 is one of them, such that

$$threshold_r < \text{strength of c1} < threshold_{sim}$$

strength of c1 < $threshold_{sim}$ → chunk c1 cannot be drawn → chunk c1 cannot be in retrieval buffer. (1)

$threshold_r <$ strength of c1 $\rightarrow$ chunk c1 can compete with other activated chunks and can be sent back $\rightarrow$ chunk c1 is in retrieval buffer. (2)
the above (1) and (2) are in contradiction, so, the assumption we made is wrong, so, $threshold_{sim} < threshold_r$

2. $threshold_{ce}$ means the threshold for creating a chunk. Every time we create a new chunk with the assumption that it is useful in the future with a high probability. "useful" here means it can compete with other activated chunks to become a winner given a specific input to NACS, that is, its strength should be greater than $threshold_r$.
since both every new created chunk and its strength actually depends on $threshold_{ce}$, so when a new chunk is created, $threshold_{ce} > threshold_r$

Chapter 7

List of task-specific CLARION options

The following task-specific CLARION options needs user's definitions in user's codes.

1. eligibility conditions

(default: all are eligible)

2. reinforcement functions

(default: empty)

3. IRL rule sets

(default: empty)

4. Fixed rules

(default: empty)

5. RER positivity criterion

(default: in the tutorial)

6. IRL positivity criterion

(default: in the tutorial)

7. RER IG measure

(default: in the tutorial)

8. IRL IG measure

(default: in the tutorial)

9. weights of action rule conditions

(default: all weights are equal)

10. rule utility

(default: in the tutorial)

11. the set of all the possible goals

(default: empty)

12. the parameter dimensions of a goal

(default: empty)

13. correction in bottom-up verification and top-down guidance combination

(default: empty)

14. weights of associative rule conditions

(default: all weights are equal)

15. similarity measure

(default: in the tutorial)

16. type of data pattern for training AAM

(default: experienced states)

17. overall RT function

(default: RT = RT_{BL} or RT_{TL}, depends on which level is used)

18. bottom level RT function

(default: $RT_{BL} = PT_{BL} + DT_{BL} + AT_{BL}$)

19. top level RT function

(default: $RT_{TL} = PT_{TL} + DT_{TL} + AT_{TL}$)

20. PT_{TL}

(default: PT_{BL} + 100ms)

21. DT_{TL}

(default: action-time + t0/rule-bla + t1/chunk-bla)

22. AT_{TL}

(no default)

23. associative rule application time in GKS

(default: t4 + t5/rule-bla)

24. chunk retrieval time in GKS

(default: t2 + t3/chunk-bla)

25. AMN one-pass time

(default: 350ms)

Chapter 8

List of routines requires user definitions

1. goals.

abstract public short[] getGoals();

2. each goal parameter dimension

abstract public short[][] getGoalDims();

3. bottom-up correction.

abstract public void getBUCorrection();

4. top-down guidance correction.

abstract public void getTGCorrection();

5. eligibility of ACS networks, return the eligible networks.

abstract public int getEligibility();

6. reinforcement of ACS networks.

abstract public double getReinforcement();

7. each dimension weight in RER condition.

abstract public double[] getRerWeights();

8. utility measure of RER

abstract public double getRerUtility();

9. RER positivity criterion, return if it is positive

abstract public boolean getRerPositivity();

10. RER Information Gain criterion

abstract public double getRerIG();

11. each dimension weight in IRL condition.

abstract public double[] getIrlWeights();

12. utility measure of IRL

abstract public double getIrlUtility();

13. IRL positivity criterion, return if it is positive

abstract public boolean getIrlPositivity();

14. IRL Information Gain criterion

abstract public double getIrlIG();

15. IRL parameters to decide an action

abstract public double[][][] getIrlParameters();

16. each dimension weight in FR condition.

abstract public double[] getFrWeights();

17. utility measure of Fixed Rule

abstract public double getFrUtility();

18. Fixed Rule conditions

abstract public short[][][] getFrConditions();

19. Fixed Rule actions

abstract public short[] getFrActions();

20. each dimension weight in Associative rule condition.

abstract public double[] getAssocWeights();

21. utility measure of Associative rule.

abstract public double getAssocUtility();

22. Associative Rule similarity measure

abstract public double getSimilarityMeasure();

23. overall RT formula

abstract public int getOverallRT();

24. overall Bottom-Level RT formula

abstract public int getBLOverallRT();

25. overall Top-Level RT formula

abstract public int getTLOverallRT();

26. PT of TL

abstract public int getPT_TL();

27. DT of TL

abstract public int getDT_TL();

28. AT of TL

abstract public int getAT_TL();

29. Associative rule application time.

abstract public int getAssocAppTime();

30. NACS chunk retrieval time.

abstract public int getChunkRetrtime();

31. AMN one-pass time.

abstract public int getAmnPassTime();

Chapter 9

The List of Source Files

NOTE:

1. The root directory of the code files : sourceCode

2. only ACS is an essential component of CLARION.

NACS and MCS/MS are optional.

9.1 Overall essential files

1) main class of the software

- main \ ClarionMain.java

2) classes for common use

- clarion \ common \ BPNet.java

- clarion \ common \ QBPNet.java

- clarion \ common \ Chunk.java

- clarion \ common \ Feature.java

- clarion \ common \ RuleAttributes.java

3) classes for coordinating.

- clarion \ coordinate \ ExGScoordinator.java

- clarion \ coordinate \ StochasticComb.java

- clarion \ coordinate \ WeightSumComb.java

- clarion \ coordinate \ CorrectionComb.java

4) classes for GUI.

(all files under "clarion \ options \ " is for GUI)

- clarion \ options \ GuiClarion.java

- clarion \ options \ GuiTemplate.java

- clarion \ options \ SelectionTemplate.java

- clarion \ options \ SettingTransfer.java

- clarion \ options \ Option.java

- clarion \ options \ OptionItem.java

- clarion \ options \ Parameter.java

5) classes for CLARION system use.

- clarion \ system \ Clarion.java

entry class for CLARION system, in charge of overall CLARION running.

- clarion \ system \ ClarionClassLoader.java

class for class loading.

- clarion \ system \ Global.java

class for storing all of globally used variables.

- clarion \ system \ Task.java

class for user to override to simulate a task.

- clarion \ system \ TaskAgent.java

class for user to override to simulate an agent in a task.

- clarion \ system \ TaskClarion.java

class for user to override to specify some task-specific stuff in CLARION.

6) class as tools

- clarion \ tools \ CommonMethods.java

- clarion \ tools \ StochasticDecider.java

- clarion \ tools \ WriteFile.java

7) class for task simulator.

- tasksimulator \ Simulator.java

- tasksimulator \ NewSimulator.java

9.2 ACS only

1) mandatory files

- clarion \ acs \ ACS.java

entry class for ACS subsystem, in charge of ACS decision making.

- clarion \ acs \ net \ AcsNet.java

- clarion \ acs \ net \ EXNet.java

- clarion \ acs \ net \ AcsNetComp.java

- clarion \ acs \ net \ IDN.java

classes for implementing of ACS subsystem

- clarion \ acs \ net \ auxilary \ TLRuleSet.java

- clarion \ acs \ net \ auxilary \ TLRule.java

- clarion \ acs \ net \ auxilary \ EligibilityCheck.java

auxiliary classes for implementing of ACS subsystem

2) optional files

all the files except ACS mandatory files in the directory: clarion \ acs and its subdirectory.

9.3 ACS + NACS

1) mandatory files.

a) ACS mandatory files

b) NACS mandatory files

- clarion \ nacs \ NACS.java

entry class for NACS subsystem.

- clarion \ nacs \ GKS.java

- clarion \ nacs \ AMNet.java

- clarion \ nacs \ AssocRuleGrp.java

- clarion \ nacs \ AssocRule.java

- clarion \ nacs \ GKSChunk.java

classes for implementing of NACS.

2) optional files.

a) ACS optional files

b) NACS optional files

all the files except NACS mandatory files in the directory: clarion \ nacs

9.4 ACS + NACS + MS/MCS

1) mandatory files.

a) ACS mandatory files

b) NACS mandatory files

c) MS/MCS mandatory files

- clarion \ ms \ MS.java

entry class for MS subsystem.

- clarion \ mcs \ MCS.java

entry class for MCS subsystem.

- clarion \ mcs \ Drives.java

- clarion \ mcs \ MonitorBuf.java

classes for implementing of MCS.

2) optional files.

a) ACS optional files

b) NACS optional files

Chapter 10

Sample Action Sequences

In the current format of NACS control action, an action sequence for retrieving from NACS and storing into WM is the following:

10.1 do inference & retrieval

- set one of the following elements in the dim6.

  dim6: do inference & retrieval

  element 1 - retrieve one chunk from NACS above a threshold by Boltzmann distribution

  element 2 - retrieve all chunks from NACS above a threshold

- at the same time, set the following parameter dimensions.

  dim13: GKS reasoning method

  dim14: number of iterations of GKS reasoning

  dim15: number of AMN passes for each iteration

  dim16: involved AMNs

after doing this action, the retrieval result will be in the retrieval buffer of NACS.

10.2 do reporting

- set the following element in the dim7.

  element 1 - report

- at the same time, set one of the following elements in the parameter dim17.

  dim17: type of chunk to report back

  element 0 - all chunk types

  element 1 - state related chunks

  element 2 - action related chunks

  element 3 - chunks not involved in the input to NACS

  after doing this action, the selected results from retrieval buffer are ready to be stored into WM.

10.3 storing into WM

use WM actions to achieve this.
In the WM action format,

- dimension 0 : action type

  set the element indicating WM action.

- dimension 1 : action specification

  set element 1 : SET

- dimension 2 : data type to be stored.

  set one of the following elements.

element 0 : external input

element 1 : GS top item

element 2 : WM items

element 3 : action

- dimension 3 : the set of WM items (in NACS retrieval buffer) to be stored into current WM if the dimension "data type to be stored" is set to "WM items". (this step is optional.)

  select a set of the following elements

  element 0 : WM item 0

  element 1 : WM item 1

  . . . .

  element i : WM item i

  . . . .

  element n-1 : WM slot n-1

  (n is the size of WM)

- dimension 4 : the set of WM slots for storing data.

  select a set of the following elements

  element 0 : WM slot 0 is involved

  element 1 : WM slot 1 is involved

  . . . .

  element i : WM slot i is involved

. . . .

element n-1 : WM slot n-1 is involved

(n is the size of WM)

after doing this WM action, the selected retrieval results will be stored into current WM.

Chapter 11

The List of changes

Overall change

rule support and chunk strength are always on.

11.1 ACS

1. From Dr. Sun

1) $threshold_{PM}$: if Partial Match is on, it is used in action decision making.

2) rule selection : use utility or rule support.

3) revision of rule support and partial match.

4) rule support : always on, and it is not an option.

5) Partial Match : an option but no SBR in ACS.

6) if no Partial Match, the activation is only 0 or 1 for chunks and features.

7) goal item representation: same as chunk (features with D-V value pair)

8) WM item representation: same as chunk (features with D-V value pair)

9) BLA for WM, GS and EM: based only on SETTING, ENCODING and STORAGE, not based on USE or RETRIEVAL.

10) IRL encoding.

11) IRL gen/spec direction: removed.

12) IRL density is very small.

13) $threshold_{IRL}$ for positivity

threshold6 for generalization

threshold7 for specialization

threshold4 for rule deletion

14) ARS learning & IDN learning:

for each IDN/group, it is determined by MCS if MCS is on. MCS decides when each is used.

15) Response Time for integration methods such as stochastic, top-down guide and bottom-up.

2. From Xi Zhang

1) some action output may involve complex structures, for example an action output may consist of multiple primitive actions. (P15)

2) For any particular domain, there are a number of possible external actions, each of which is represented by a set of dimension-value pairs. That is, each external action has a number of action dimensions. Do-nothing is included as special case. (P17)

3) the eligibility condition of an external action network may be specified based on goals: each subtask may have a different goal, and the current goal indicates the eligibility of different networks. It may also be specified based on more elaborate information (such as the current state along with the current goal, or certain patterns in the past state sequences). (P18)

4) In the case there are multiple action dimension, this process is carried out for each of these dimensions, separately. (P21)

5) the learning methods may be under the control of MCS (P23)

6) each dimension is either ordinal (discrete or continuous) or nominal. (P25)

7) That is, it "activates" all the rule condtion chunks that specify a subset of dimension-value pairs as indicates by state x. (P27)

8) The number of match-all rules. (P30)

9) we hypothesize that these different types of rules represent different "layers" of knowledge, from the most fluid to the most entrenched. (P36)

10) the strength of a chunk and the rule support. (P37)

11) To select an action recommendation out of all the rules whose conditions match the current input, if rule utility is not used, the Boltzmann distribution is constructed using the rule support. (P40)

12) bottom-up rectification and top-down guidance. (P45)

13) WM actions: set i. (P51)

14) coordinating multiple action types: Perform all the chosen actions simultaneously (P54)

11.2 NACS

1. From Dr. Sun

1) reasoning method: with or without SBR.

2) combination method: use MAX to integrate the results from the 2 levels of NACS.

3) In GKS, potentially, there is no difference between I/O feature.

4) EM may have multiple types of item such as associative rule, action rule and steps.

5) EM items are similar to experience-specific chunks except a time stamp.

6) remove counterfactual, backward chaining and proof by contradiction.

7) no balancing parameters c14 and c15, and may scale by similarity measure and

rule weights.

8) MACS rule condition : use WEIGHT-SUM across condition chunks.

9) $threshold_{EM}$: a threshold used to check if an item needs to be removed.

10) priority : ACS action can overwrite MCS and MCS can overwrite default value in Option & Parameter

2. From Xi Zhang

1) chunks encode co-occurrences of features. Links across chunks encode associations between chunks. (P62)

2) the figure 3.1 (P63)

3) bottom-up activations. (P64)

4) the features are activated to the same strength level as the chunk, iff one of dimensions of a chunk has mutiple allowable values, the strength of the chunk is evenly split among these allowable values of the same dimension. if a dimensional value receives activations from multiple chunks, the max of these activations is used. (P64)

5) the condition of an associative rule consists of a single chunk or multiple chunks. (P65)

6) we may divide up associative rules in the GKS into multiple groups with each corresponding to one of the networks in the bottom level. (P65)

7) these different operational modes may be decided by the ACS, which often controls the operations of NACS. (P68)

8) in the case of SBR, the chunk strength. (P68)

9) combination of chunk strength both from GKS and AMN results. (P68)

10) the input to NACS is normally determined by ACS actions. (P69)

11) All applicable associative rules in GKS fire simultaneously. (P70)

12) to integrate the outcomes of the GKS and AMNs, the process of bottom-up activation and top-down activation. (P70)

13) reasoning methods and a $threshold_r$ that determines whether a conclusion is acceptable or not. (P71)

14) the rule support. (P72)

15) the implementation of SBR. (P73)

16) the encoding of externally given knowledge is under the control of the ACS.

17) other chunks are formed when they are (1). extracted from the bottom level of the NACS OR (2). extracted as a result of RER or IRL rule learning in the ACS. (P79)

18) In setting the strength level of an extracted chunk, the bottom-up activation process is used. (P81)

19) when there is an overlapping dimension where the cue and the result specify different values, an OR operation may be used to combine them. (P81)

20) Assimilating explicit knowledge through training an AMN using association stored in EM. (P82)

21) common patterns and clusters may emerge, which enables some generalization. (P82)

22) AMN may be trained under the direct control of ACS. (P82-P83)

23) ACS dictates the type of reasoning to be performed by the NACS. (P83-P84)

24) ACS action commands may decide how outcomes from the NACS are to be used. (P84)

25) EM may have different types of items. (P86)

26) EM items are not subject to the chunk density parameter or the associative rule density parameter. However, they are subject to the chunk encoding probabil-

ity parameter or the associative rule encoding probability parameter, respectively. (P87)

27) AEM: maximum number of states and actions, the encoding of states and actions, reinforcement quantization. (P88)

28) the algorithm for Action-Directed Reasoning. (P90)

11.3 MS and MCS

1) track top conclusions of AMN in terms of chunks in GKS with bottom-up activation.

11.4 RT

1) RT for stochastic selection, bottom-up rectification and top-down guidance. (P117)

2) DT_{TL} = operation-time + t0/rule-bla + t1/chunk-bla. (P119)

3) if multiple rules matching current input, we select one rule out of these rules, then the selected rule is applied. The RT of the applied rule is calculated. (P119)

4) BLA of WM items are not included in this formula. (P119)

5) the retrieval time of an item has been calculated into the time of applying a retrieval rule. (P119 and P121)

6) the total associative rule applicaiton time is the maximum of individual associative rule application and result chunk retrieval times. t_{GKS}. (P121)

7) RT of NACS is part of AT_{TL} or AT_{BL}. (P121)

8) the RT of MS and MCS. (P122)

9) MS, MCS and the ACS operate in parallel. (P122)

11.5 OPTIONS & PARAMETERS

1) the default selection for all of the BLA options = 2.

2) major subsystems involved. (P124)

3) specifications of actions that operate within the system. (P124-P125) combine the external actions for reporting WM and GS contents and for controlling NACS.

4) a commonly used actions, especially for controlling NACS, are pre-set and available for use. (P125)

5) the Q value discount rate, default value = 0.99. (P126)

6) the weights of a chunk representing an action rule condition. (P127-P128)

7) the weight of an action rule. (P128)

8) use of action rules in action decision making. (P128)

9) threshold4, threshold5 and threshold6. (P130)

10) cross-level integration methods. (P132)

11) coordination of external, goal and WM actions. (P133)

12) dimensions and values of the GKS. (P134)

13) NACS: rule support and chunk strength are always on. (P134)

14) weights of associative rule condition chunks. (P134)

15) weights of dimensions in a chunk. (P135)

16) the probability of using EM for offline training of ACS at each step. (P136)

17) removal of c14 and c15 in SBR. (P136)

18) GKS reasoning methods. (P137)

19) the reasoning threshold. (P137)

20) the types of items from the EM to use for traning this AMN at each step. (P138)

21) the maximum of states and actions. (P138)

22) the range and increment for quantization of reinforcement. (P138)

23) removal of C20 - C31. (P141-P142)

24) AT_{BL}, default = 500ms. (P143)

25) DT_TL. (P143)

26) AT_{TL}, default = 500ms. (P143)

27) RT OF MS and MCS. (P144)