LETSEQ: An Implementation of the CFS-C Classifier System in a Task-Domain that Involves Learning to Predict Letter Sequences†

Rick L. Riolo

January 1986
Updated November 1988

Logic of Computers Group
Division of Computer Science and Engineering
University of Michigan

†This work was supported in part by National Science Foundation Grants DCR 83-05830 and IRI 86-10225, and by a University of Michigan Presidential Initiaive Fund Grant to the BACH research group.

Contents

1 Introduction

LETSEQ1 ("LETter-SEQuence" is a program that implements a learning classifier system in a specific, simple environment. In particular, the task for the LETSEQ1 classifier system is to learn to predict which letter (character) will come next in a sequence, based on the letters the system has seen so far.

This document briefly describes how to use the LETSEQ1 program, including how to start the program, specify the sequence of letters to be learned, generate random classifiers, modify the variables that control the system's learning mechanisms, produce output, and so on.

LETSEQ1 is implemented using CFS-C, a package of subroutines which implement a learning classifier system. For a complete description of the CFS-C classifier system, see [Riolo, 1988a]. In particular, before using the LETSEQ1 program, a user should be familiar with Section 2, "Classifier Systems," and Section 3, "The CFS-C Classifier System," of that document. Those sections give a detailed description of all the important characteristics and features of the CFS-C major-cycle, including:

1. How messages satisfy classifier conditions.
2. How bids are calculated.
3. How classifiers compete to post messages.
4. How effectors are matched and activated.
5. How classifier strength is re-distributed by the bucket-brigade, taxes, and so on.
6. How new classifiers are created by the system's rule discovery mechanisms.

That document also includes a complete description of all CFS-C commands, constants, and system variables, including the commands and variables briefly described in this document.

Note that LETSEQ1 is primarily intended to be an example of how to use the CFS-C package to implement a learning classifier system in a particular environment or task-domain. LETSEQ1 also provides a simple yet interesting domain for exploring the characteristics of classifier systems that use the bucket brigade to allocate credit to classifiers, both with and without the creation of new classifiers. Note well that the LETSEQ1 problem is *much harder* than it seems at first glance: recall that the classifier system does not know anything about letters, sequential relationships, repetitive patterns, or *anything* if it is started with random rules. For comparison, consider how hard it would be for a human to see just a bank of 16 lights[1], some on and some off at each step. Given a series of on-off patterns, one at each step, the human subject must then guess which lights will be on next!

However, I **do not** think the LETSEQ1 program uses a a good *representation* (i.e., mapping of letters to detector messages and messages to effector actions) of the task domain, and the reward scheme is probably inadequate as well. For example:

a. Each classifier represents some high level "concept" (e.g., "if the current letter is an *a* predict a *b*"), rather than having a number of classifiers represent each concept and each rule participate in a number of concepts.

[1] Or harder still, see what the LETSEQ1 classifiers see, i.e., four banks of lights.

b. Their are too few features characterizing letters, and they are not organized in any "natural" way. For instance, it might be better to have a number of *redundant* features (e.g., vowels, high-medium-low vowels, consonants, fricatives, voicing, etc.) describing a series of phonemes.

Basically, I think the current representation used in LETSEQ1 is too *coarse grained.* I think such a representation leads to inefficient use of the genetic algorithm as a discovery mechanism, and it makes it difficult for the system to learn *gracefully*, a major goal for classifier systems.

Thus I *strongly* recommend that you **do not** use LETSEQ1 an example of how to *design* a representation for the CFS-C (or any) classifier system. Instead, LETSEQ1 should be treated as a simple design that shows how to *implement* a design once one has been chosen.

Note well: While I do attempt to keep this document up-to-date, that is note always possible. So if you have any questions or problems, please look at the FSW1 source files for the definitive answers about the version you have. Also please feel free to contact me with questions, comments, suggestions and, of course, bug reports. See the README.TXT file on the distribution disk for information on how to contact me.

2 The Task-Domain for the LETSEQ1 Classifier System

In general, classifier systems are meant to operate in a specific environment or task-domain. An environment for the CFS-C classifier system is defined by specifying:

1. One or more *Detectors,* which act as an input interface that provides the system with information about the environment.

2. One or more *Effectors,* which act as an output interface that provides a way for the classifier system to change the environment.

3. A *Payoff* function, which basically defines the goals the classifier system must try to attain.

Typically the environment itself is simulated in the computer that is running the classifier system. In that case, the state-space and transition functions that define the environment and its dynamic behaviors also must be specified.

This section describes the task-domain/environment implemented in the LETSEQ1 program.

2.1 The Letter-Sequence Environment State-Space

The environment for the LETSEQ1 system is just a sequence of lower-case letters, i.e., vowels (a, e, i, o, u, and y) and consonants (b, c, d, etc.). For a given run, the user must specify a particular sequence, e.g., "becidabecid". These letters are stored as a circular list, so that as far as the classifier system is concerned, the user-specified sequence is repeated indefinitely. At any given step in model-time, one letter in the sequence, the "current letter," is the letter the classifier system has seen most recently.

2.2 The Detectors

The LETSEQ1 program has four detectors. Each detector provides the system with information about one of the last four letters the classifier system has seen, i.e., about the current letter and its three predecessors.

In particular, each detector adds one message to the message list. All messages are strings from the $\{0,1\}^l, l = 16$. Each detector message starts with "0" in its left-most loci, which labels or *tags* the message as one from a detector.

The values in loci 4 and 5 (counting from the left symbol as 1) of detector messages indicates which detector produced the message. The following table shows the relation between the values in loci 4 and 5 and the detector that produced the message:

Loci 4,5	The Detector is for:
00	The current letter
01	The previous letter
10	The letter before the previous letter
11	The letter 2 before the previous letter

That is, the values at loci 4 and 5 can be interpreted as binary numbers 0, 1, 2, and 3, corrsponding to the recency of the letter described by the message (0 for current, 1 for previous, and so on).

The values of loci 10 to 16 (still counting from the left) of a detector message indicate what letter the detector found. In particular, loci 10 and 11 indicate what "type" of letter was seen:

Loci 10,11	Type
01	Vowel (blank, a, e, i, o, u, or y)
10	Consonant (blank, b, c, d, ..., w, x, z)
11	Punctuation (blank, comma, period, semicolon)

Loci 12 through 16 indicate which letter of the specified type was detected. That is, each letter is associated with an integer that indicates its place in the list of letters of its type (e.g., "a" is vowel 1, "e" is vowel 2, and so on, while "b" is consonant 1, "c" is consonant 2, etc.), and loci 12 to 16 contain the binary form of the number associated with the letter represented. (Note that the blank character is the 0^{th} character of each type.) The following list shows some example detector messages. Note that the messages have blank spaces inserted in them to aid the reader–the blanks are not part of the messages:

Message	Meaning
00 000 0000 01 00001	The current letter is "a"
00 001 0000 10 00001	The previous letter is "b"
00 010 0000 10 00010	The letter before the previous one is "c"
00 011 0000 01 00001	The letter 2 before the previous one is "a"

Normally the detectors add messages to the message list at the beginning of every major-cycle step. However, the LETSEQ1 user can change the detector "sampling rate" by changing the value of the "DetectRt" variable.

2.3 The Effectors

The LETSEQ1 classifier system must have at least one effector, 'GuessLetter." When the GuessLetter effector is activated, it does two things:

1. It makes a prediction, i.e., it specifies what letter the system predicts will come next in the letter sequence.

2. It advances the current-letter pointer one letter in the letter sequence that makes up the system's environment, so that what was the next letter becomes the current letter.

Any string of 16 characters from the $\{0, 1, \#\}$ alphabet can be used as the condition-part of the GuessLetter effector. The GuessLetter condition-string can also be prefixed by a "m", to indicate it is a match condition, or by a "$\sim$", to indicate it is a not-match condition. The following condition is the standard one used for GuessLetter:

$$m10\#\#\#\#\#\#\#\#\#\#\#\#\#\#$$

Note that any message that has "10" in its left-most loci will satisfy the above condition and "activate" the GuessLetter effector.

If only one message satisfies the effector-condition, that message is used to determine what prediction the system will make. Basically, the effector maps messages (strings of 0's and 1's) into letters using the inverse of the rules used by the detectors to map letters into messages.

In detail, the rightmost 7 loci of an effector message specify a *character* to be guessed. The leftmost two bits of those seven loci[2] specifiy the type of the letter:

left loci	Type
01	Vowel (blank, a, e, i, o, u, or y)
10	Consonant (blank, b, c, d, ..., w, x, z)
11	Punctuation (blank, comma, period, semicolon)

The rightmost five are used to determine, for a given type, what "letter" of that type is to be the prediction. For instance, if the left loci are "01" (i.e., the type is vowel), the right-most loci are first interpreted as a binary number and then that number serves as an index into a list of vowels (i.e., vowel[1] is "a", vowel[2] is "b", and so on). Similarly, if the left loci are "10" (i.e., the type is consonant), and the rightmost five loci are binary 2, the message means "guess b". Note again that the blank character is the 0^{th} letter of each of the three types of letters (just to make *a* be the first vowel, etc.).

Clearly there are 128 possible characters represented by the the rightmost 7 loci of a message, but there are only 32 "letters" in the LETSEQ1 alphabet. Any messages that map into characters that are not valid letters (where letters includes alphabet and the punctuation listed above) are mapped into the "@" character. For example, the message

[2]That is, loci 10 and 11 of the whole message, counting from the left.

1000 0000 0000 0000 does not represent a valid letter and so it is interpreted as "guess @" (which will always be wrong).

The following list shows some example messages that match the effector condition shown above (the blank spaces are inserted in them to aid the reader):

Message	Prediction
10 0001100 01 00001	Predict an "a"
10 0010110 10 00010	Predict a "c"
10 0101011 01 00011	Predict an "i"

Note that loci 3 through 9 do not affect the prediction–those loci can have any sequence of 1's and 0's.

If more than one message matches the GuessLetter effector on a given step, there is a potential for a conflict which must be resolved, as when one message means "guess a" and another means "guess c". In this case each effector message is treated as "support" for guessing the character its rightmost 7 bits represent (including characters that are not in the LETSEQ1 alphabet), and the LETSEQ1 effector conflict resolution mechanism selects (probabilistically) the one with the most support to be the system's guess.

In particular, the support for a particular character c is just:

$$Sup(c) = \sum I_{m(c)}$$

where the sum is over the intensity, $I_{m(c)}$, of each effector message m that maps into "guess c" under the rules specified above[3]. The probability that character c is actually chosen is just:

$$Sup(c) / \sum Sup(c')$$

where c' ranges over all characters supported by some effector message during the current step. Once c is chosen, it is then mapped into a letter (or an @) as described above. Further, all effector messages that support $c' \neq c$ are *deleted* from the message list, so that only messages that agree with the guess made (or that don't make a guess) remain on the list.

If no message matches the effector-condition during a major-cycle step, and the only effector is the GuessLetter effector described above, the LETSEQ1 system makes no prediction, and it doesn't advance the current-letter pointer during this current major-cycle step. This can easily lead to the LETSEQ1 system getting "stuck," i.e., never making predictions and so never getting any more information about the letter sequence. For example, if no classifier produces a message, the message list will be empty, so no prediction will be made, and the current-letter pointer will not be advanced.

To avoid this possibility, there is a second effector available in the LETSEQ1 system, called the "DefaultGuess" effector. Any string of 16 characters from the $\{0, 1, \#\}$ alphabet can be used as the condition-part of the DefaultGuess effector, and it can be either a match or a not-match condition. The following condition is the standard one used for DefaultGuess:

[3]The intensity is just the bid made to post the message. See the CFS-C documentation for a discussion of message intensity.

$\sim 10\#\#\#\#\#\#\#\#\#\#\#\#\#\#$

This condition is satisfied just when there are no messages that begin with the "10". That is, DefaultGuess is satisfied just when the GuessLetter effector is not satisfied. When the DefaultGuess effector is activated, it acts just like the GuessLetter effector, i.e., it makes a prediction about the next letter and it advances the current-letter pointer. However, unlike the GuessLetter effector, DefaultGuess will simply make a guess at random. Note that the random guess is selected from all 128 (2^7) possible effector values, *not* from the letters already seen or the letters in the whole sequence.

Just as LETSEQ1 detector "sampling rate" can be altered by changing the DetectRt variable, the rate at which effectors are activated also can be altered by changing the value of the "EffectRt" variable.

Finally, note that the condition-part for the LETSEQ1 effectors are loaded from a file when the LETSEQ1 program is run, so other conditions can easily be used, and the DefaultGuess effector can be ignored completely. (See Section 3, "How to Use the LETSEQ1 Program," for more information about loading conditions for effectors.)

2.4 The Payoff Function

All classifier systems implemented with the CFS-C package must include a payoff function of some kind. Basically, the payoff function maps states of the environment into numerical (real-valued) rewards. The rewards can be thought of as information from the environment to the classifier system about how well the system is doing on the task defined for it. (In fact, as far as the classifier system is concerned, the payoff functon does define the goals the system will try to achieve.)

Typically, most environment states are neutral with respect to a particular task, i.e., the reward associated with most states is zero. However some states are intended to be goal states, e.g., correct solutions to the task; these states should have positive rewards associated with them. Other states can be designated failure states by associating negative rewards with them.

In the letter-sequence environment, the system receives a reward only when the system's letter-predicting effector is activated. The value of the reward is determined by comparing the system's prediction (i.e., the letter it claims will be next) to the letter that actually does come next.

In particular, the LETSEQ1 effectors store their predictions in the variable "GuessLet" and they advance the current-letter pointer one letter in the letter sequence (i.e., making what was the next letter in the sequence the current letter). The reward is determined simply by comparing the predicted letter to the current letter.

The following table shows the three possible outcomes for this comparison. Note that the rewards are given in terms of program variables. The default values for these variables are shown in parentheses, but the other values can be assigned to these variables.

Reward is:	When:
SameLRew	The prediction is correct, i.e., the predicted letter is the same as the actual letter.

SameTRew The prediction is not correct, but the predicted letter is of the same type as the actual letter (e.g,. the prediction is "a", a vowel, and the actual letter is "e", another vowel).

WrongRew The prediction is just wrong; it's not the correct letter and it's not even the same type.

The default values for the reward variables are loaded from a file when the LETSEQ1 program is run, so other values can easily be used. (See Section 3, "How to Use the LETSEQ1 Program," for more information about loading the reward values.)

By default, when a non-zero reward is received at step T, the *full* payoff is added to *all* active classifiers. The CFS-C system provides a number of run-time settable parameters that can be used to control the allocation of payoff (reward) from the enviroment, including:

- *ShareRew.* If ShareRew is 0 (the default), the full reward is allocated to all eligible classifiers. If ShareRew is 1, the payoff is divided equally amoung the eligible classifiers.

- *FrPayNEC.* If FrPayNEC is 1 (the default), all active classifiers (those that *posted* messages not eliminated by the effector resolution mechanism) have the payoff (or payoff share) added to their strengths. If $0 \leq FrPayNEC < 1$, classifiers that produced effector messages still have the full reward (or share) added to their strengths, but other non-effector message produces have $FrPayNEC * Reward$ added to their strengths.

See the CFS-C documentation for more information about these and other CFS-C parameters.

2.5 How to Use the LETSEQ1 Program

This section describes how to run the LETSEQ1 program in general terms. The instructions for running LETSEQ1 may differ slightly on various computers with different operating systems.

Note that the rule discovery algorithms are *off* by default; to use these algorithms (the genetic algorithm, triggered chaining mechanism, etc.), use the SET command to turn them "on." See the CFS-C documentation for a description of all the variables that control the CFS-C system. Also see the README.TXT file (and files it refers to) on the distribution disk for example command files to run the LETSEQ1 system.

The simplest way to start the LETSEQ1 system is to enter:

letseq1 path

where **path** is the path specification to the path that contains the file INIT.CFS[4]. When the system is started in this way, the environment, message list, and classifier list are initialized using information from three files, INIT.ENV, INIT.MSG, and INIT.CF, respectively. If

[4] INIT.CFS contains a list of all run-time settable variables for CFS-C; see the file INTRO.TXT on the distribution disk for more details.

those files are empty or do not exist, warning messages will be displayed (but don't despair—you can load an environment and classifier using CFS-C commands). The system will print out some start-up notices and then display the system's prompt for a command:

```
->
```

The user may then enter a CFS-C command to start the major-cycle running, display the classifiers, set a system variable to some other value, and so on. (See Section 4, "Basic LETSEQ1 Commands," for a description of the commands a user may enter.) After the system has executed a command, it again prints the "->" prompt and waits for another command.

If the user wants to start the LETSEQ1 system with particular messages in the message list, those messages must be in the INIT.MSG file in a particular format. Similarly, if the user wants to start the LETSEQ1 system with paricular classifiers in the classifier list, those classifiers must be in the INIT.CF file in a particular format. Alternatively the user can enter the CFS-C *LOADMSG* or *LOADCFS* commands to load classifiers from files—see the CFS-C documentation for more details.

The LETSEQ1 program also can be started without reading initial messages and/or classifiers from external files. For example, a user can start a LETSEQ1 run with an empty message list and with classifiers created by the GENRANDCFS ("generate random classifiers") command.

However, the user must specify an environment before the system can be run. This can be done by putting information in the INIT.ENV file and having it loaded automatically at start-up time, or by using the *LOADENV* command to load the information from some other file. In either case, the file must specify:

1. The reward-values that define the payoff function, i.e., values for the variables SameLRew, SameTRew, and WrongRew;
2. The sequence of letters that is to be predicted; and
3. The condition-part for the effectors.

This information must be layed out as shown by the following example INIT.ENV file:

```
; Lines that begin with a semicolon are ignored.
; The payoff function values and some settings are given first:
; SameLRew SameTRew WrongRew
50 10 -10 1 0 2
; That is, SameLRew is set to 50, SameTRew is 10, and
; WrongRew is -10.  The settings are included to be consistent
; with a saved file:  see the file LETSEQ1.C for details.
; The next line contains intial counts of the total number
; of each kind of Reward, followed by the count of the
; number of default guesses made.  This line is usually
; all zeros, but it necessary for consistency with the
; files created by the SAVESYS command.
0 0 0 0
```

```
; The next line contains the letter sequence itself:
ababab
ENDLETSEQ1
; Then comes the effector-conditions,
; each followed by an optional effector name.
m10##############, GuessLetter
~10##############, DefaultGuess
END-EFFS
```

The reward values must be on the first non-comment line, followed by initial value for some system counters, followed by one or more lines that contain the letter-sequence itself. There must be no blanks or other characters on the lines that contain the letter sequence, and the letter sequence is terminated by a line that contains the literal string "ENDLETSEQ" starting in column 1. The next data line after the ENDLETSEQ line must contain the condition for the GuessLetter effector. The effector name may be may be up to 16 characters long. If the user wants to use the DefaultGuess effector, the next line should contain the condition for that effector, as shown in the above example. If no condition is specified for the DefaultGuess effector, it just will not be used during the run. The last line of the INIT.ENV file should contain the literal string "END-EFFS" starting in column 1.

Note that when the LETSEQ1 system is started, the first four letters of the user-specified letter sequence are used to generate the detector messages for the first major-cycle step. For example, if the user-specified sequence of letters is "abcdefghi", the letter "d" is used as the "current letter," "c" is "previous letter," and so on.

2.6 Executing Commands Stored in a File

Besides entering commands from a keyboard, a user may run LETSEQ1 and have it read commands from a specified file if the operating system provides a re-direction facility similar to that provided by MS-DOS or UNIX-like operatinng systems. This facility provides an easy way to run experiments in which each experiment is run using the same basic commands, except that some system variable values are set to different values at the beginning of the run.

For example, if the file EXP-1.SOU contains the commands:

SET ECHO=1
SET LOG=EXP-1.LOG
SET bid_k=1.5 HeadTax=0.02 BidTax=0.0
GENRANDCFS 200
SET ADENVINT=50
CLASSIFY 2000
STOP

a user could run LETSEQ1 by entering:

letseq1 path <EXP-1.SOU

where **path** is again a path specification to the location of the INIT.CFS file. LETSEQ1 would load the environment from the file INIT.ENV, and attempt to load messages and classifiers from the files INIT.MSG and INIT.CF, respectively. Then LETSEQ1 would execute the commands in the file EXP-1.SOU, one by one, just as if the user had entered them from the keyboard. Note that the command "SET ECHO=1" is included at the beginning of the EXP-1.SOU file so that the commands read from that file will be echoed, i.e., written to the screen as they are read from the source file.

3 Some Basic LETSEQ1 Commands

This section briefly describes some basic CFS-C/LETSEQ1 commands. For a complete description of these and other CFS-C commands, see "Appendix A, The CFS-C System Commands" in [Riolo, 1988a]. Note that all commands and command parameters can be entered in any case, and parameters can be separated from each other by one or more blanks.

3.1 The STOP Command

To end a run and return to the operating system, just enter:

    STOP

3.2 The DISPLAY Command

The DISPLAY command can be used to display various items, e.g., the message list, the classifier list, and so on. By default the items are displayed on the standard output device (usually the computer's screen), but the output can also be directed to other files and devices.

For example, to see the contents of the current message list, enter:

    DISPLAY ML

The messages will be displayed, one message per line. Each line will include the message-string itself, along with the message's intensity (the amount bid to produce it) and the Id number of the classifier that produced it.

Note that since the DISPLAY command (and all other commands) are processed when the system is stopped at the beginning of a major-cycle, the current message list will contain the messages posted by classifiers active at the previous major-cycle step. The displayed list never contains detector messages, since they are not added to the message list until a major-cycle has begun to execute.

Classifiers can be displayed in several formats. In each format the classifiers are displayed one per line, in order from highest strength classifiers to lowest. In particular, to display classifiers in the first format, enter:

    DISPLAY CL,1

or

    DISPLAY CL

Each display line will contain the Id number of a classifier and the classifier itself, i.e., its left and right condition strings, the type of each condition ("m" for match, "~" for not-match), its action-type (e.g., "PASS" for pass-through), and its action-string. Each line will also contain the classifier's strength and bidratio.

To display classifiers in the second format, enter:

DISPLAY CL,2

Each line will again contain information about one classifier, but the lines will contain different information. In particular, each line will begin with the classifier's Id number, its strength and its change-in-strength since the last step, and its bidratio, and a other information about the classifier's history of activity. See the CFS-C documentation for more details on the first two domain-independent displays of classifiers.

LETSEQ1 also provides two other domain *dependent* displays of classifiers, i.e., displays of classifiers that have been interpreted with respect to the domain representation specified by the detector and effector definitions. For example, if the classifiers are display using **DISPLAY CL,21** (or 22 or 23) the classifiers will be displayed in the following format:

id> stag: let [w] itag; stag: let [w] itag =actt=> stag: let [w] itag {s} use uic

where:

id Classifier id number.

stag "Source" tag, the leftmost 2 bits interpreted as follows: 00 → D (detector), 10 → G (effector), 01 → X and 11 → Y (classifier-to-classifier messages). Note that a wildcard in these loci, e.g., "1#", will result in an interpreted form like "GY".

let The letters specified by rightmost 7 bits. This has most meaning for conditions matching detectors (stag = D) and effectors (stag = G). There may be a list of letters, if condition has wildcards.

w The detector window positions matched, interpreted from loci 3 and 4 (counting from 1 on left). For example, a detector message from the current letter will give w = 0. This may contain more than one number from $\{0, 1, 2, 3\}$ if there are wildcards in the condition. This has most meaning for conditions matching detectors (stag = D).

itag Uninterpreted loci 3 to 8, counting leftmost locus as 1. These are loci not used by detectors or effectors, so they are available for use as classifier-to-classifier tags. This is displayed for formats 22 and 23.

actt The action type, e.g., PASS, AND, etc. (The PASS action is not written.)

s The classifier's current strength, bidratio and support, in braces.

use How rule was used in the most recent step: *b* means it bid, *w* means it won, and *p* means it posted messages (after effector conflict resolution.)

uic Uninterpreted classifier, with no punctuation. This is displayed in format 23.

Note that the "let" (letter) region of a string will be interpreted even when the string is not actually a detector or effector message. Similarly, the "w" (window) region of a string will be interpreted for non-detector parts of classifiers.

In any format, after all the classifiers are displayed, a line of total values for all current classifiers willl be displayed, (e.g., total strength for all current classifiers, total number of positive rewards, etc.), followed by some total values for all classifiers that are or were on the classifier list (i.e., total number of positive rewards for all current classifiers and for all classifiers that have been replaced by the system's rule discovery learning mechanisms).

Besides displaying the message list and classifier list, several other items can also be displayed:

DISPLAY VARS Some selected CFS-C system variables are displayed, e.g., the system treasury value, the number of classifiers in classifier list, how many classifiers bid on the last step, the highest and lowest strength classifiers, and so on.

DISPLAY CYCLESTP The major-cycle step counter is displayed, which is equal to the number of cycles completed plus 1.

DISPLAY ENV The state of the environment can be displayed. The system will respond by displaying (a) the current letter (i.e., the one just read in), the three letters that preceded it, and the rest of the letter sequence, and (b) the values that define the payoff function, namely, SameLRew, SameTRew, and WrongRew.

Several items may be included with one DISPLAY command, and the DISPLAY command can be abbreviated "DI". For example, the command:

DI ML CL,2 ENV

will cause the system to display the message list, followed by the classifier list (in the second format), followed by the state of the environment.

All of the CFS-C system variables can also be displayed. For example, in response to the command:

DISPLAY Bid_k

the system will display the value of the "Bid_k" variable (i.e., the proportionality constant used when calculating bids). Similarly, the command:

DI HeadTax BidTax

will cause LETSEQ1 to display the values of the CFS-C taxes.

The output produced for each item displayed can be routed to any available file or device. For example, in response to:

DI ML,,MSGFILE CL,1,CFFILE ENV,,ENVFILE

the system will write the message list to the file MSGFILE, the classifiers (in format 1) to the file CFFILE, and the environment display to the file ENVFILE.

3.3 The GENRANDCFS command

The GENRANDCFS command can be used to fill the classifier list with classifiers generated at random. This command provides an easy way for a user to start an experiment with a list of random classifiers.

For example, if the user enters:

GENRANDCFS 100

the system will generate 100 classifiers and *append* them to any classifiers on the list. By default each classifier will have a bidratio equal to its specificity. Note that the LETSEQ1 user can control several properties of the classifiers generated by the GENRANDCFS command, including:

1. The mean strength of the classifiers,
2. The mean bidratio of the classifiers,
3. The proportion of classifiers with not-match second conditions, and
4. The proportion of classifiers with pass-through and other action-types.

The user can also have the random classifiers *replace* any classifiers already in the list. See the CFS-C documentation for details.

3.4 The CLASSIFY and STEP Commands

The CLASSIFY and STEP commands can be used to actually execute the major-cycle of the classifier system. For example, in response to the command:

CLASSIFY 100

the major-cycle will be executed 100 steps before the user is again prompted for a command. The command:

STEP

can be used to execute one major-cycle (i.e., it is equivalent to CLASSIFY 1).

The CLASSIFY command can be abbreviated CL, and the step command can be abbreviated ST.

3.5 The SET Command

The SET command can be used to change the value of almost any of the CFS-C system variables. These system variables control many aspects of the bidding process, the bucket brigade, the discovery learning algorithms, and so on. These variables also control some of the system's I/O facilities, e.g., the auto-save and auto-display features described later in this document.

For example, the command:

SET Bid_k=0.15

assigns the value 0.15 to the *Bid_k* (the proportionality constant used when calculating bids). Several variables can be included in one SET command, e.g.:

SET HeadTax=0.01 BidTax=0.02 PrdTaxMx=0.04

set the variables that control CFS-C taxes to the indicated values.

For a complete list of CFS-C variables that can be used with the SET command, see the CFS-C documentation.

3.6 The ECMD Command

The ECMD ("environment command") command can be used to change the values of variables that are domain-specific, i.e., specific to the letter sequence domain. The ECMD can also be used to see what the next detector messages are going to be.

For example, to change the value of the SameLRew variable (the amount of reward given to classifiers when the system predicts the correct letter) a user may enter:

ECMD SET LR=20.0

In this case SameLRew will be assigned the value 20.0.

The ECMD can also be used to control the way the CFS-C/LETSEQ1 effector conflict resolution picks one letter to guess when co-active classifiers have supported conflicting guesses (see section 2.3, above). In particular:

- *ECMD SET E=0* tells the system to pick the message with the highest intensity and use the character it supports as the system's guess.
- *ECMD SET E=1* tells the system to pick the character with the most support, deterministically.
- *ECMD SET E=2* tells the system to pick the character probabilistically, with probability proportional to the sum of the support for each character.

E=2 is the default mechanism, as described in section 2.3.

To see what other ECMD parameters are available, enter:

ECMD HELP

Brief descriptions of the allowed parameters will be displayed.

3.7 The SAVESYS Command

At any time during a LETSEQ1 run, the SAVEVSYS command can be used to save (i.e., write) the entire state of the system, including the message list, the classifier list, the state of the environment, and so on, to a user-specified file. This file can then be used, after the LETSEQ1 program has been stopped, to re-start the LETSEQ1 program in exactly the same state it was in when the system was saved to that file.

For example, after running the LETSEQ1 program for 800 steps a user could enter the command:

SAVESYS EXP1-800

The state of the system at that step would be written to the file named EXP1-800. The user could then continue the run, record the results, and stop the program. The user could then re-run the program by entering:

LETSEQ1 *path* EXP1-800

(where *path* is a path specification to the location of the INIT.CFS file). The user could then change some system variables, continue the run, record the new results, and compare them to the final results of the first run to see what effect the changes had.

3.8 The Automatic-Save Facility

Besides saving the state of the system by entering the SAVESYS command, a user can have the state of the system periodically saved to a series of files, each file containing the state of the system at a particular major-cycle step.

For example, if a user enters:

SET SAVEINT=500

the entire state of the system will be saved every 500 major-cycle steps, until the run stops or until the user enters:

SET SAVEINT=0

The system-states will be saved in files that have names of the form:

Smddhhmm.nnn

where "S" is a constant, and:

m is the month of the run (in hex),

dd is the day of the run,

hh is the hour the run started,

mm is the minute the run started, and

nnn is 001, 002, and so on, one for each file saved during a run.

The files created by the automatic save facility are located on the active drive or directory.

As with the SAVESYS command, each "save" file contains values for almost all the system constants and variables, the current and old message lists, the classifier list, and the state of the environment at the end of the major-cycle during which the file was written. Besides serving as initialization files for subsequent runs, the contents of these files can be analyzed and/or summarized to determine and describe what happened during the course of a particular experiment.

3.9 The Automatic-Display Facility

Besides displaying various items by entering DISPLAY commands, LETSEQ1 users can have items periodically displayed as the major-cycle is being executed (i.e., while a CLASSIFY command is executing).

For example, to have the classifier list displayed every 250 steps, a user should enter:

SET ADCFINT=25

The classifier list will be displayed (in format 2) every 250 major-cycle steps, until either the run stops or until the user enters:

SET ADCFINT=0

By default the classifiers are displayed in format 2 when they are displayed automatically. The command:

SET ADCFFMT=1

can be entered to have the classifiers automatically displayed in format 1.

Other items can also be automatically displayed. For example, the command

SET ADDMINT=50 ADENVINT=50 ADVARINT=50

could be entered to have (a) the messages added by the detecters, (b) the environment state, and (c) selected system variables all displayed every 50 steps.

Note that all items except for detector messages are automatically displayed at the end of major-cycle steps. Detector messages are automatically displayed when they are added to the message list, i.e., at the beginning of a major-cycle step.

3.10 The Logging Facility

The LETSEQ1 program provides a logging facility, i.e., a way to have all user input and all output that is normally written to the computer's screen also written to a user-specified file or device. This is useful for some operating systems (like DOS). For example, if a user enters:

SET LOG=G:EXP-32.LG1

would cause the log to be written to a file named EXP-32.LG2 on the E: drive of a DOS system. The user can stop the logging at any time by entering:

SET LOG=OFF

References

[Riolo, 1988a] Riolo, Rick L. "CFS-C: A Package if Domain Independent Subroutines for Implementing Classifier Systems in Arbitrary, User-Defined Environments." Logic of Computers Group, Division of Computer Science and Engineering, University of Michigan, Ann Arbor, 1988.

[Riolo, 1988c] Riolo, Rick L. "CFS-C/FSW1: An Implementation of the CFS-C Classifier System in a Domain that Involves Learning to Control a Markov Process." Logic of Computers Group, Division of Computer Science and Engineering, University of Michigan, Ann Arbor, 1988.