LETF

A Lisp-Based Exploratory Testing Framework for Computational Cognitive Models

Clayton T. Stanley (clayton.stanley@wpafb.af.mil)
Air Force Research Laboratory
WPAFB, OH 45431 USA

Keywords: exploratory testing, ACT-R, high performance computing

Introduction

Developing a computational cognitive model is an iterative process. Due to this, any model validation/testing technique cannot be static, and must be agile enough to evolve alongside the model's development. Over the past few decades, a software validation paradigm called exploratory testing has emerged within the software engineering community. Exploratory testing is not static, favors a high level of interactivity between the tester and the program, and advocates that programmers should "build time and enthusiasm for parallel research, test development, and test execution" (Kaner, 2004). As cognitive modelers, trying to develop a model that performs within a certain range of objective performance metrics while maintaining a level of cognitive plausibility, exploratory testing is not new to us. In fact, testing frameworks are already available to the cognitive modeling community. In ACT-R, for example, there is the visual-location crosshair for the vision module, the buffer activity trace, and the fMRI BOLD visualization.

However, many of these exploratory testing formalisms are specific to the cognitive architecture that the modeler is using, and there are a fair amount of exploratory tests that are architecturally agnostic. For example, exploring the architectural/strategy/parameter space of a cognitive model, computing objective performance measures, and capturing the central tendency of stochastic models are all common modeling issues. In order to enable modelers to easily explore these sorts of generic performance metrics, I have developed LETF, a lisp-based exploratory testing framework for computational cognitive modelers.

LETF

LETF is a lightweight configurable lisp program that layers on top of a cognitive model. After LETF is configured and launched, it spawns the cognitive model as a separate process, sending inputs to the model as command line arguments, and grabbing outputs from the model by capturing the standard output stream. Model inputs are specified in a work file, where each row in the file is a particular parameter combination, and columns correspond to the values for a parameter in the model. A flexible configuration file allows for extended processing of model outputs, and configurable display format by means of a modular print method. The flexibility of the configuration file is an important consideration, because once the model is set up to interface with LETF, modifying and adding model tests does not require altering the model's code. Instead, you express the tests by modifying the configuration file.

In order to express a large range of different model tests in the configuration file, we are not providing a large number of specific APIs to support each test. Instead, we have removed the API layer altogether, and grounded the syntax of the configuration file to the underlying language of the generic testing framework. That is, the syntax of the configuration file *is* lisp. And it is this critical feature that makes adding and changing model tests in LETF so expressive and agile.

The best way to make this point is with a simple example. Suppose we have a model written in lisp (e.g., an ACT-R model), with independent variables (IVs) 'noise', 'speed', dependent variables (DVs) 'rt1', 'rt2', 'rt3', and we want to compute the correlation between the observed and model data. The configuration file to express this model test (i.e., compute the correlation) could look like Figure 1.

```
IV= noise
IV= speed
file2load= extras.lisp
modelRTs= (list [rt1] [rt2] [rt3])
observedRTs= (getObservedRTs)
correlRTs= (correl [modelRTs] [observedRTs])
DV= correlRTs
```

Figure 1: Example configuration file.

The model code that would communicate with LETF for this example could look like Figure 2.

```
(defun run-model (&key (noise) (speed))
   …run the model using the values for IVs 'noise' and
   'speed' that were passed to 'run-model'
   (format t "rt1=1.1~%")
   (format t "rt2=2.2~%")
   (format t "rt3=3.3~%"))
```

Figure 2: Example model.

Note that LETF offers a more direct interface when the model is in lisp (shown in Figure 2). In this case, LETF can communicate with the model by calling the entry function 'run-model' instead of spawning a separate process.

Many cognitive models are stochastic, and so it is common to run them multiple times to reveal the central tendency. This can be accomplished with the simple addition to the configuration file

```
collapseQuota= 100                                    (1)
```

And, if we wanted to use a collapsing function other than the default 'mean', we could specify our own directly in the configuration file (note the lisp syntax)

```
collapseFn= (lambda (a) (+ (mean a) 1000))            (2)
```

LETF calculates the value of a variable X on a 'DV=X' line by expanding the string expression that variable X represents, and then invoking the lisp reader to interpret that expression and return a value. Using the example in Figure 1, the 'DV=correlRTs' line tells the program to find the expanded string expression for the variable 'correlRTs'. LETF finds the 'correlRTs=' line in the configuration file, and sets the value of 'correlRTs' to

```
(correl [modelRTs] [observedRTs])                     (3)
```

Then, all variable names within brackets '[]' are recursively expanded to their values, and the program evaluates the expression by invoking the lisp reader

```
(eval (read-from-string
 "(correl  (list 1.1 2.2 3.3) (getObservedRTs))")     (4)
```

Note that once the brackets are recursively expanded, the API between the configuration file and LETF *is* lisp, matching the underlying language of the generic testing framework. Going back to Figure 1, the modeler is writing a lisp expression *around* the variables sent by the model (rt1-3) in order to calculate a DV of interest ('correlRTs'). The function 'getObservedRTs' (which returns a list of observed response times for trials 1 through 3) would be defined in the lisp file 'extras.lisp', which is loaded (by specifying the 'file2load=extras.lisp' line) and therefore visible to LETF when it evaluates the string expression in [4]. The 'correl' function has already been defined in LETF, which computes the correlation between two lists, so the expression in [4] will bind the correlation of RTs between the model and observed data to the variable 'correlRTs'. Having this on-the-fly configurability available directly in the configuration file – both syntactically and semantically anchored in the experimental testing framework's own language – can be a very powerful paradigm.

Discussion

LETF supports the exploratory testing that a modeler might perform during the early stages of model development. For example, with modest computational resources (e.g., a laptop workstation) and a few lines of code, a modeler can build an exploratory test framework for their cognitive model. Iterative changes in the model might drive evolution of the exploratory tests, while results of the tests might drive iterative changes in the model. Test results can be displayed in whatever format the modeler thinks is informative (e.g., printing to the terminal, communicating via a socket to a data visualization program, writing to a text file) and can evolve over time as well. In fact, this sort of exploratory testing technique is currently being used to test incremental changes to LETF's own code.

LETF also supports large-scale exploration of cognitive models by taking care of data aggregation and restructuring (e.g., calculating DVs, collapsing to determine central tendency) before outputting the results in a configurable format that can be coupled with specific parameter search algorithms and High Performance Computing (HPC) systems. For example, it was recently coupled with Moore's regression tree search algorithm 'Cell' (2010) to run a computational cognitive model of the Change Signal Task (Moore, Gunzelmann, & Brown, 2010) on the Maui High Performance Computing Center, Mana. In just a few minutes, LETF was configured to interface the model and the search algorithm (Cell) for an exploratory analysis that used over 6000 processor hours (running in less than 12 hours wall clock time) on Mana.

We recognize that there are a fair amount of exploratory tests that apply generally across architectures, and have provided a generic exploratory testing framework to easily specify those tests. Further, LETF has no API layer between the configuration file and the language that the framework was built in (i.e., lisp), allowing for an unbound number of tests that can be expressed. It is light enough to run a model on a standalone desktop computer through a small range of hand-coded configurations, and generic enough to couple with large scale exploratory tests on HPC clusters. Very much in the spirit of exploratory testing, LETF helps accelerate the pace that cognitive modelers can develop, test, and improve their computational models[1].

References

Kaner, C. (2004). The ongoing revolution in software testing. *Proceedings of Software Test and Performance Conference.* Baltimore, MD.

Moore, L. R., Jr., Gunzelmann, G., & Brown, J. W. (in press - 2010). Modeling Statistical Learning and Response Inhibition with the Change Signal Task. In *Proceedings of the 10th International Conference on Cognitive Modeling, Manchester, United Kingdom*.

Moore, L. R. (2010). Cognitive model exploration and optimization: a new challenge for computational science. In T. Jastrzembski (Ed.), *Proceedings of the 2010 Behavior Representation in Modeling and Simulation (BRIMS) Conference*. Orlando, FL: Simulation Interoperability Standards Organization.

[1] Please contact the author if you are interested in obtaining the source code