Chapter 4
Analyzing Learning Methods in a Functional Environment

Alberto de la Encina
Universidad Complutense de Madrid, Spain

Mercedes Hidalgo-Herrero
Universidad Complutense de Madrid, Spain

Pablo Rabanal
Universidad Complutense de Madrid, Spain

Ismael Rodríguez
Universidad Complutense de Madrid, Spain

Fernando Rubio
Universidad Complutense de Madrid, Spain

ABSTRACT

Developing cognitive programs is a complex task. Thus, special purpose languages can help developing such systems. This chapter presents a programming environment to help studying the behavior of cognitive models. This environment allows to easily define new cognitive processes, it simplifies the methods to interconnect them, and it also provides graphical information to analyze how a complex cognitive system is evolving. Moreover, it also includes observation facilities, so that the user can analyze the internal behavior of each of the cognitive entities appearing in the system. The authors illustrate the usefulness of their system by using several examples within the chapter.

DOI: 10.4018/978-1-60566-902-1.ch004
Analyzing Learning Methods in a Functional Environment

The relation between the Computer Science and other older well established sciences is known to be twofold. On the one hand, the Computer Science provides tools, automatic analysis techniques, and even new concepts to other sciences, which enables the construction of new methodologies and theories in these areas. On the other hand, many methods used in Computer Science are inspired in the nature. Examples of this relation are Ants-based Algorithms, Neural Networks, and river formation dynamics (Rabanal, Rubio, & Rodríguez, 2007, 2008). This twofold relation is particularly fruitful in the scope of Cognitive Science, which exploits the similarities between natural and artificial reasoning processes. In particular, the roots of several Artificial Intelligence algorithms are based on the cognitive processes of the brain, while simulating a cognitive process in a computational environment may help to validate/refute a theory in the field of Neuroscience.

There is a topic that has been typically concerned by informatics scientists that, actually, fits perfectly in the context of Cognitive Informatics (Zhang et al., 2007; Wang, 2002; Wang & Kinsner, 2006; Kinsner, 2008). This is the relation between artificial cognitive processes and human cognitive processes. This issue has been addressed for years by computer scientists, and it is one of the main topics of interest in Artificial Intelligence. One of the first approaches proposed to relate both concepts was made by Alan Turing and it consists in a nowadays classical test. In brief, the Turing test (Saygin, Cicekli, & Akman, 2000; Turing, 1950) says that a system should be considered intelligent if its behavior cannot be distinguished from that of a human being.

Following this idea, in the long term we are specially interested in being able to compare how both automatic systems and humans learn the rules governing their environment. Let us note that comparing the learning processes of both humans and machines is hard because it requires to compare the cognitive theories each of them builds and refines along time. This enables the comparison of the levels of accurateness of each of them along time.

In this article, we will concentrate on developing an environment to help analyzing the behavior of an automatic system. We will be able to observe its external behavior by recording the relations between its input stimuli and its output responses. However, the most interesting part is that we will also be able to observe its internal behavior. That is, we will observe how the entity constructs its responses to the input stimuli.

In order to obtain our objectives, we will consider an specific programming environment. The main part of our programming environment is its core language. Eden (Klusik, Loogen, Priebe, & Rubio, 2001; Loogen, Ortega-Mallen, Pena, Priebe, & Rubio, 2002; Rubio & Rodriguez, 2003; Hidalgo-Herrero, Ortega-Mallen, & Rubio, 2006) is a parallel extension of the functional language Haskell (Jones & Hughes, 1999) which provides ‘controlled’ parallelism, as opposed both to more implicit approaches such as GpH (Glasgow Parallel Haskell) (Trinder et al., 1996, 1998) and to more explicit ones such as Concurrent ML (Reppy, 1991). In this sense, in Eden the programmer decides which expressions are processes and how processes are connected, having explicit control over the communication topology. The remaining aspects such as sending and receiving messages, process placement, data distribution, etc. are implicitly controlled by the runtime system. The parallel constructions of Eden allow to easily specify any topology. This is specially important when simulating the complex topologies that can appear in cognitive models. Moreover, Eden programming has the advantages of functional languages. That is, its high-level constructions simplifies the task of developing programs, and it facilitates the creation of executable templates that reduce the time needed to develop applications.