Towards Goal-Driven Reflective Learning

Pierre E. Bonzon
HEC-Inforge,
University of Lausanne, 1015 Switzerland

pbonzon@hec.unil.ch

1. Introduction

More than 35 years ago (which is quite a long time in AI history), John McCarthy wrote:

"... in order for a program to be capable of learning something, it must first be capable of being told it... Included in the set of imperatives which may be obeyed is the routine which deduces and obeys." [6]

Except for various ad hoc (and sometimes quite successful) systems, this de facto manifesto calling for the study of introspective systems did not give rise to what may be called "a general architecture for declarative and/or reflective machine learning". Some recent research taking place under the label of "goal-driven learning" signals however a renewed interest in these very basic issues: " *learning in such systems is an active process... to generate learning goals, goal-driven learning systems must be introspective*" [5].

Interestingly enough, researchers from outside of the AI community have developped high hopes from such systems: "*..decision support systems can be provided by facilitating and stimulating reflective learning*" [1]. For most people however reflective learning remains an unexplained concept. A tentative intuitive description goes as follows:

after solving a problem, *reflect* on the solution, i.e. try and express *implicit* solving processes and/or ideas as *explicit* procedures and/or knowledge.

A more constructive approach leads to the following 3-steps process:

- after focusing on a problem, select a particular goal
- look for ways to achieve this goal and then build a plan leading to a solution
- finally, "reflect" on this plan and come up with a reusable framework allowing one to easily retrieve and/or deduce pairs of goal/plan instances.

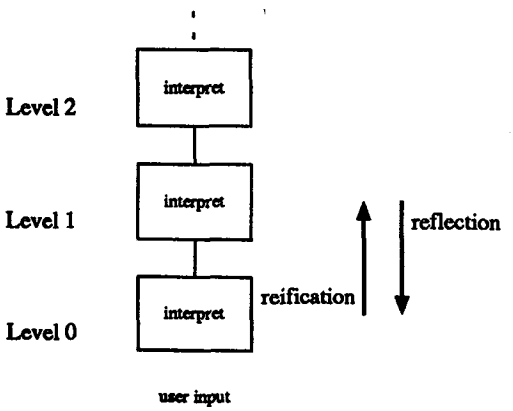
This last description bears numerous analogies with previously advocated learning paradigms, such as explanation-based learning (EBL) . In fact many EBL problem solvers have been designed "to learn from successful operator sequences" with the definite goal of improving the system's overall problem solving performance [7]. Our own approach basically follows the same path. To try and distinguish it from previous work, let us mention its use of

- a *reflection* scheme allowing the deliberative integration of planning, execution and learning steps [4]
- predefined *generic operators* that permit one to improve problem solving performance by switching from "search guided only by weak methods to domain-dependent focused behavior" [2].

The rest of this paper is organized as follows: section 2 reviews some fundamental concepts about *reflection*; section 3 introduces the one particular framework we have been experimenting with so far; section 4 illustrates the preceeding discussions with a simple working example.

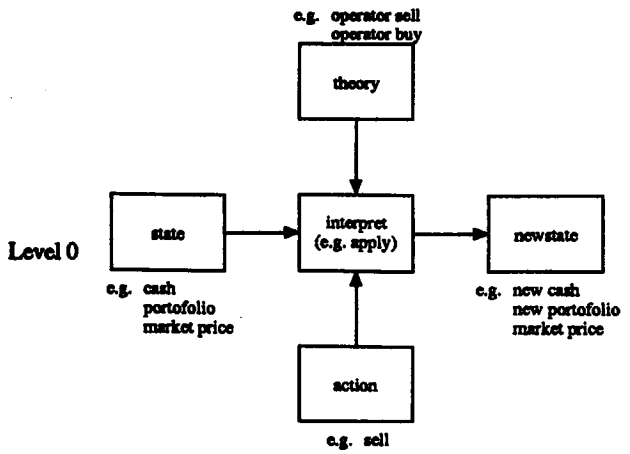
2.-Towards a reflective learning tower architecture

Following the pioneering work of B. Smith [9], it has become customary to introduce reflective systems by considering an infinite *tower* of metacircular language processors (or interpreters) in which each processor interprets the one below it, with the interpreter at the bottom executing user input and the whole tower being run by an "ultimate machine" at the top. As recalled by Jefferson and Friedman [3] (which also provide a very readable and enjoyable reconstruction of an equivalent *finite* tower), the various levels in this tower are connected by a mechanism, called *reification*, that permits a program running at one level to provide code to the next higher level. Thus when a user level *reifier* is applied, "its body is run as if it were code belonging to the interpreter running the application". In another view of this same process, the contents of an interpreter registers are passed above suitably package or reifed, such that the receiving interpreter can manipulate them (this process is then described as converting program into data). Conversely, *reflection* (giving rise to so-called *deifiers*) activates a level down in the tower, and can be seen as the process by which some data values from an interpreter are loaded into the lower interpreter registers. Consequently, "this process may be thought of as turning data into program" [10].

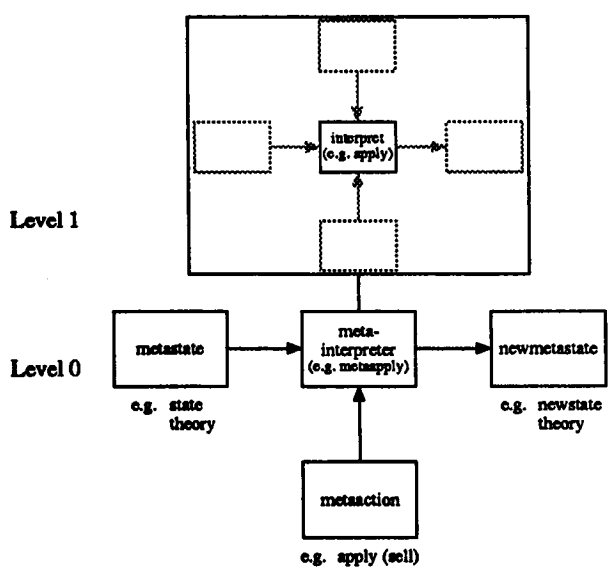


As a result the language itself becomes *extensible*. On the whole this process achieves *dynamic code declaration* and/or *interpretation* and thus opens the door to adaptive and/or learning behaviour. We have therefore been looking for a *reflective learning tower* architecture, whereby each level in this new tower would correspond to an interpreter operating on some theory representing knowledge about either a base domain or meta-level control.

We choose to model each level with explicit input (e.g an *action, state* and *theory)* as well as output (e.g a *new state)*, and thus get the following one level picture:



Note how some of the state components are updated while others are left unchanged. Taking into account the desired reflective structure of the architecture, we then get the following two level picture:



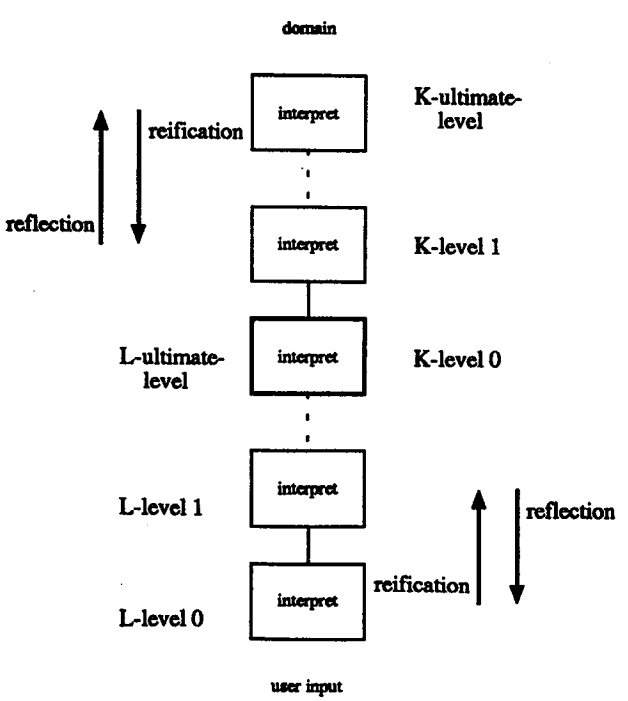
While this simple metainterpreter introduces the new base state into its own new metastate, it does retain throughout the original base theory, thus mirroring back the one-level model given above. More elaborate models could easily be designed to *update* the base theory itsef, thus mirroring a true *learning* behaviour.

With respect to the original tower implementing metacircular language processors, this new tower structure has the following characteristics:

- both user input (i.e. the metastate and the metaaction) *and* the ultimate processor (i.e. the metainterpreter, assumed here to be supported by a built-in interpreter) are now at the *bottom*
- each processor at a given level interprets the one *above* it (a consequence of the ultimate processor bottom location)
- reifiers (i.e processes running as if they were code belonging to the interpreter running the application) now permits to run code at the next *lower* level, while deifiers activates a level *up* in the tower.

For example, in order to activate the upper (i.e. less abstract) interpreter from within the metainterpreter, a deifer only needs to load level 0 data from the metastate and metaaction (i.e. a state, theory and action) into the corresponding dotted slots at level 1 . Conversely, a reifier may be used to run a lower (i.e. more abstract) metainterpreter from within a given interpreter: for such an example, see below section 3 where we sketch how a particular chain of reifier applications stands at the heart of our learning model architecture.

As user input still comes in at the bottom, it is important to note that this presentation is *not* just an upside-down developped view of the traditional one. In fact this new tower, whose purpose is to model some activities related to the external (or real) world, could well be placed above the previous one, whose sole purpose is to model language interpretation (without any reference whatsoever to a model of the external world). Conversely, any language tower could be placed under any learning tower. The resulting structure could be depicted as follows:



This final structure clearly distinguishes two fundamental components, i.e. the L-levels and the K-levels (for language- and knowledge-level, respectively). While user inputs are received at the bottom, they are ultimately interpreted at the L-ultimate level, which is where "some concrete computation really proceeds, say, by using underlying hardware resources"[8]. This ultimate L-level also coincides with the base K-level, where the knowledge interpretation process is actually initiated. On-going computations up to the K-ultimate level eventually relate to the domain at the top, which models the external world.

3.-A particular learning scheme based on predefined generic operators

For the purpose of our discussion it suffices to note in a first approach that each processor in the learning tower (except for the one which coincides with the L-ultimate level) can be represented in a STRIPS-like formalism involving the usual *precondition, add* and *delete* lists, together with an additional *execute* list containing calls to reifiers and/or deifiers. Besides base level knowledge taking the form of domain operators, a meta-level may well embody a classical means-end analysis forming an MEA planner, i.e a typical *weak* search method. Our learning scheme is then based on the following assumptions:

> to any *plan* returned by the MEA planner (the result of a weak search method analogous to a backward chaining behavior possibly involving backtracking) corresponds a *learned* program analogous to an optimized forward chaining behavior involving the application of *derived* domain operators which can be used without preconditions.

Among the (meta)theories that are put to work via deifiers to implement this scheme, we single out:

- a *"generic"* theory containing partially uninstantiated operators whose execute list *chains together successive calls to reifiers* allowing to activate a lower level metatheory (such as *metaapply),* allowing one in turn to activate derived upper level domain operators
- a *learning* meta-theory which, given a particular MEA plan, is used to instantiate operators from the generic theory and to derive directly applicable operators from the original *domain* theory, thus giving rise to a directly applicable learned domain theory.

4.-Application: a dynamic student's model of skill acquisition

The learning scheme just introduced was inspired by and intended to serve as the basis for modelling student's acquisition of financial accounting skills from textbook knowledge. Such student routinely face the goal of finding the right journal entries corresponding to given accounting situations. Applicable textbook knowledge, as expressed in a formal base-level theory, first provides the kind of entries associated with various possible basic transactions. Further textbook knowledge indicates the kind of events leading to such transactions. Under the hypothesis that unskilled students will "go by the book", the implicit solving process allowing them to deduce the correct entries will be equivalent to a backward chaining search, and thus will match the behavior of the MEA planner.

When the number of possible events becomes high, forward chaining is a better choice. Skilled students, reflecting on their experience, will eventually switch to this mode. When encountering successive instances of the same kind of event they will further cut their search effort, a number of analogies gradually allowing them to "blindly" follow the same procedure. Directly applicable operators are thus implicitly put to work.

As hypothesized in section 3, MEA plans can be used as a starting point to model this learning process. The plans thus obtained can then be used by the learning theory to instantiate generic *entry* operators. Together with modified domain operators bearing the name of single preconditions, these entry operators will constitute a new domain theory that in essence allows one to forward chain the right set of journal entries.

5.- Acknowledgement

The author is greatly indebted to anonymous referees for valuable comments on a previous version of this paper.

6. References

[1] Angehrn, A., Stimulus Agents: An alternative Framework for Computer-aided Decision Making, DSS-92 Transactions, M. Silver (ed), The Institute of Management Science, 1992

[2] Carbonnel, J. , Introduction: Paradigms for Machine Learning, *Artificial Intelligence*, 40, 1989

[3] Jefferson, S., and Friedman, D., A Simple Reflective Interpreter, in Yonezawa, Y. and Smith, B., (eds), *Reflection and Meta-Level Architecture*, Proc . IMSA Workshop, 1992

[4] Kuokka, D. The Deliberative Integration of Planning, Execution and Learning, PhD Thesis, Carnegie Mellon Univ., 1990

[5] Leake, D, and Ram, A., Goal-Driven Learning: Fundamental Issues, AI Magazine, 14,4, 1993

[6] McCarthy, J., Programs with Common Sense, 1958, appeared in M. Minsky (ed), *Semantic Information Processing*, MIT Press, 1968

[7] Minton, S. et al., Explanation-Based Learning, A Problem Solving Perspective, *Artificial Intelligence*, 40 1989

[8] Nakajima, S., What Makes a Language Reflective and How (preliminary results), in Yonezawa, Y. and Smith, B., (eds), *Reflection and Meta-Level Architecture*, Proc . IMSA Workshop, 1992

[9] Smith, B., Reflection and and Semantics in Lisp, Proc. ACM POPL,1984

[10] Wand, M., and Friedman, D., The Mystery of the Tower Revealed: a Non-Reflective Description of the Reflective Tower, Lisp and Symbolic Computation, 1,1, 1988