Inductive Acquisition of Algebraic Specifications

Extended Abstract

Lutz Hamel and Chi Shen
Department of Computer Science and Statistics
University of Rhode Island

Inductive machine learning [1, 2] suggests an alternative approach to the algebraic specification of software systems [3]: rather than using test cases to validate an existing specification we use the test cases to *induce* a specification. In the algebraic setting test cases are ground equations that represent specific aspects of the desired system behavior or, in the case of negative test cases, represent specific behavior that is to be excluded from the system. Acceptable specifications must satisfy the positive test cases and must not satisfy the negative test cases.

It is interesting to observe that in this alternative approach the burden of constructing a specification is placed on the machine. This leaves the system designer free to concentrate on the quality of the test cases for the desired system behavior. In addition, the induction process can be viewed as a coherence test for the test cases. For example, a failure of the system to induce a specification that satisfies all the (positive) test cases can be due to the fact that some of the test cases are contradictory. Inductive logic programming [4] and in particular inductive equational logic programming [5, 6] seem well suited for this task. In this framework the learning system is asked to search for an equational theory (hypothesis) H such that,

$$H \cup B \models E^+, \text{ (1)} \quad \text{and} \quad H \cup B \not\models E^-, \text{ (2)}$$

where B is a possibly empty equational theory representing background knowledge, E^+ is the set of ground equations representing the positive test cases, and E^- is the set of ground equations representing the negative test cases. Posterior sufficiency (1) states that the theory $H \cup B$ has to satisfy all the positive test cases and posterior satisfiability (2) states that none of the negative test cases can be a logical consequence of the induced theory together with the background. In addition, some other technical requirements such as prior satisfiability and prior necessity need to hold in order for this induction framework to make sense [4].

The following is a simple example that shows the induction of a stack specification from a set of positive test cases for the stack operations 'top', 'push', and 'pop' [6]. They are given in the syntax of the OBJ3 specification language [3].

```
obj STACK-FACTS is sorts Stack Element .
  ops a b :-> Element . op v: -> Stack .
  op top : Stack -> Element .
  op pop : Stack -> Stack .
  op push : Stack Element -> Stack .
  eq top(push(v,a)) = a .
  eq top(push(push(v,a),b)) = b .
  eq top(push(push(v,b),a)) = a .
  eq pop(push(v,a))= v .
  eq pop(push(push(v,a),b)) = push(v,a) .
  eq pop(push(push(v,b),a)) = push(v,b) .
endo
```

The set of negative examples and the background knowledge is empty. A hypothesis specification that satisfies the positive facts is,

```
obj STACK is sorts Stack Element .
  op top : Stack -> Element .
  op pop : Stack -> Stack .
  op push : Stack Element -> Stack .
  var S : Stack . var E : Element .
  eq top(push(S,E)) = E .
  eq pop(push(S,E)) = S .
endo
```

It is noteworthy that our implementation of an inductive equational logic system within the Maude specification system [5, 7] induces this specification unassisted. In addition, we have used our system to induce the algebraic specifications of a number of challenging problems [5, 6]. Most interestingly, the system was able to induce a successful description of the trains in Michalski's train problem [8].

The system, as implemented at this point, only supports many sorted equational logic. We have plans to investigate the implementation of order sorted as well as hidden sorted equational logic. Of these extensions the hidden sorted equational logic is particularly interesting because induced theories only need to satisfy the test cases over the visible sorts or behaviorally. Therefore, the induction algorithm has much more flexibility in the construction of suitable hypotheses.

A number of systems have been designed that aim to induce equational theories from a set of (ground) equations [9-11]. However, our approach seems to be unique in that we firmly base it on inductive learning principles.

References

[1] S. Muggleton, *Inductive acquisition of expert knowledge,* Glasgow, Scotland; Reading, Mass.: Turing Institute Press ;Wokingham, England; Addison-Wesley, 1990.

[2] T.M. Mitchell, *Machine Learning,* New York: McGraw-Hill, 1997.

[3] J. Goguen and G. Malcolm, *Software engineering with OBJ : algebraic specification in action,* , vol. 2, Boston: Kluwer Academic, 2000.

[4] S. Muggleton and L.D. Raedt, "Inductive Logic Programming: Theory and Methods," Journal of Logic Programming, 19,20:629-679, 1994.

[5] C. Shen, "Evolutionary Concept Learning in Equational Logic," Masters Thesis, University of Rhode Island, 2006.

[6] L. Hamel, "Evolutionary Search in Inductive Equational Logic Programming," in *Proceedings of the Congress on Evolutionary Computation (CEC2003),* 2003, pp. 2426-2434.

[7] M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer and J.F. Quesada, "Maude: Specification and Programming in Rewriting Logic," Theoretical Computer Science, 2002.

[8] J. Larson and R.S. Michalski, "Inductive inference of VL decision rules," *SIGART Bull.,* pp. 38-44, 1977.

[9] J. Darlington and R. Burstall, "A system which automatically improves programs," *Acta Informatica,* vol. 6, pp. 41-60, 1976.

[10] J. Hernandez-Orallo and M.J. Ramirez-Quintana, "A Strong Complete Schema for Inductive Functional Logic Programming," In Proc. of the Ninth International Workshop on Inductive Logic Programming, ILP'99, volume 1634 of LNAI, pages 116-127, 1999.

[11] N. Dershowitz and U.S. Reddy, "Deductive and Inductive Synthesis of Equational Programs," J. Symb. Comput. 15(5/6): 467-494 (1993).