N88-17233

A Formal Approach to Validation and Verification for Knowledge-Based Control Systems

Glen Castore
Honeywell Inc.
Systems and Research Center
Minneapolis, MN.

Abstract

As control systems become more complex, in response to desires for greater system flexibility, performance, and reliability the promise is held out that artificial intelligence might provide the means for building such systems. An obstacle to the use of symbolic processing constructs in this domain is the need for verification and validation of the system. Techniques currently in use do not seem appropriate for knowledge-based software. An outline of a formal approach to V&V for knowledge-based control systems is presented in this paper.

1 Introduction

Knowledge-based systems have been applied in areas as diverse as medical diagnosis, machine tool programming, and VLSI design. Such applications have the common characteristic that the recommendations of the expert system can be dealt with in a fairly relaxed manner. A doctor reviews the diagnosis made by an expert system to see if it is sensible. If there is some question about it, the diagnosis can be ignored or the system can be queried as to the basis for the analysis. Time pressure is not severe and control of the situation, in particular control of the use of the output of the expert system, remains in human hands. With many of these systems problems with the implementation or design can be detected while the software is in use, be fixed, and the expert system is still be considered sufficiently reliable to be useful.

This casual mode of operation is unacceptable when the knowledge-based system is operating as part of autonomous or semi-autonomous units such as machine tool controllers, robots, the space station life support module, or an aircraft flight control system. In such applications it becomes essential to have a precise language for specifying what the knowledge-based system should do, and to have an effective procedure for insuring that a particular implementation does meet these requirements. This is the goal of validation and verification (V&V) procedures.

While there are no standard definitions for verification or validation there is a general understanding that *verification* addresses the issue of whether the program specification accurately reflects the functions to be performed while *validation* addresses the question of whether the specifications are correctly implemented. The ideas are summarized in the phrases:

"Is the correct program being built?" - (verification)
"Is the program built correctly?" - (validation)

Verification is often largely a manual process. Specifications are read, cross-referenced, and checked for consistency and completeness. The quality of this work is heavily dependent on the environment available for developing and tracking specifications and requirements. Validation, on the other hand, has traditionally involved a large amount of testing, simulation and, to a much lesser extent, methods based in formal logic for establishing properties of a program.

For most knowledge-based systems, however, validation through testing and simulation is inappropriate. There are two dominant reasons for this. First, knowledge-based systems are usually most appropriately modeled as nondeterministic automata. A characteristic of nondeterministic machines is that identical inputs to the machine (in identical states) do not necessarily yield identical outputs. The basis for verification by simulation crumbles. Secondly, expert systems, a subclass of knowledge-based systems, are not always expected to give the right answer, just as experts do not always give the right answer. Thus the notion of program correctness cannot always be formulated in terms of input-output behavior, which is the assumption behind testing and simulation as well as some formal methods. While neither of these properties is unique to knowledge-based software, they are much more prominant than in, for example, operating system software. Moreover, these are not characteristics often found in software which must meet rigorous V&V criteria. Control logic for aircraft control systems, for example, is often designed explicitly in terms of finite state machines. Thus it is much more amenable to validation through testing and simulation.

In this paper a formal model is proposed for V&V for knowledge-based control systems. *Formal* means based in mathematical logic. *Knowledge-based control systems* (KBCS) are control systems in which symbolic processing methods are tightly coupled to standard control algorithms. The approach taken is to formulate a structural model for the KBCS. This model can be viewed as a representation of the nondeterministic automaton mentioned above. A logic is then developed for asserting and reasoning about properties of such structures. Specifications are interpreted as assertions about properties of the model. The role of the validation software is to prove these assertions.

2 V&V for KBCS

Within the domain of real time control the anticipated problems of V&V for knowledge-based systems are compounded by the real time aspects of the domain. These difficulties are ameliorated somewhat by restricting the domain of application to systems built in conformity with a prescribed model for KBCSs. This model applies to a class of control systems for which it appears that the use of knowledge-based methods can contribute to system performance and fault tolerance. It seems that such domain models may be necessary to reduce the computational complexity of the formal V&V methods to tractable proportions.

2.1 V&V Issues and AI

There are a number of aspects of artificial intelligence programs which appear to complicate the task of V&V.

1. There is often a strong nondeterministic flavor to A.I. programs.
2. Time of execution for inference algorithms can be extremely data dependent.

3. Interrupt handling is difficult and unreliable. There are no standard interfaces to other components of the system and no well defined methods for resuming an interrupted inference process.

4. Languages typically used for A.I. are usually weakly typed.

These are not unique characteristics of symbolic processing programs. It is the conjunction of these properties within A.I. programs, together with the conceptual complexity of the programs, which creates difficulties when attempting to base V&V procedures upon formal logic.

2.2 Issues Raised by Real Time Applications

The phrase *real time*, when applied to computer programs, is generally used to invoke images of dire consequences of failure and dismally restrictive time constraints on program execution. This view is not altogether untrue, but is is perhaps too imprecise. In the context of developing knowledge based control systems four aspects of real time performance seem to dominate design and implementation decisions.

1. *Time constraints on system performance, and thus implicitly on software execution.* The software must be viewed in the context of the entire system. Constraints on software performance result from percolating system requirements through an architecture. Inadequate software performance can be indicative of an inappropriate architecture, as well as an inadequate implementation of the software itself.

2. *Actions have consequences and the penalty for not meeting requirements can be severe.* These consequences may be economic, such as ruining a batch of toilet paper, or they may lead to injury or loss of life.

3. *The timing of events is determined by the system environment, not by the programmer.* As with performance requirements, these constraints can result from choices concerning the hardware and communication's architecture as well as the original system requirements.

4. *Demands on the system may occur in parallel rather than sequentially.* Contention for resources will occur in patterns that the programmer has not anticipated.

The real issue here is not some mythical intrinsic sluggishness of knowledge-based systems. In fact performance, in the sense of speed of execution, is often adequate for embedded control applications. The issue is adding constructs to the base language which enable the system developer to incorporate timing and sequencing constraints, for example, within the KBCS without sacrificing clarity and abstraction.

3 Knowledge-Based Control Systems

Verification and validation of the implementation software is a standard requirement for many control systems. Consequently the successful incorporation of constructs from artificial intelligence within the framework of control theory requires that there be a method for V&V of knowledge-based control systems. If methods based in formal logic are to be used as the foundation for V&V in this domain, it is necessary to be able to describe, in a precise way, what constitutes a well-formed knowledge-based control system.

Traditional control theory deals with systems which can be described in terms of a state vector, $(x_1(t), \ldots, x_n(t))$ where the $x_i(t)$ are usually reasonable real-valued functions. The time evolution of the state is governed by differential, difference, or integral equations. Within this framework methods have been developed which enable designers to address questions of stability, coverage, reliability, and performance among other things. Control systems for a wide range of devices, from toasters to airplanes, have been built using these theories. There are problems, however, which fall naturally into the category of control but for which these methods appear to be inadequate [5]. Systems in which the state space description involves discrete, and perhaps nonnumeric, variables fall into this category. We such systems *hybrid*. Hybrid systems arise when there is mode selection, when switches or limiters are used, or when extensive fault management techniques are required. In such cases the "mode switching logic", or the "fault management logic", which constitutes the discrete aspect of the control system, is usually constructed in a fairly ad hoc manner.

The theory of knowledge-based control systems is meant to be an extension of traditional control theory which will enable integration of symbolic processing methods with standard approaches to control, while retaining the ability to rigorously address questions of stability, performance, and reliability. The model which has been developed is based largely upon work by Wonham and Ramadge [6] and will be described in detail in a forthcoming paper. *The value of such a formal domain model, from the perspective of V&V is that it enables a formal specification language to be built.* The language is complete in the sense that it completely describes this family of control systems, and statements in the specification language can be readily translated to assertions in a modal logic about the structure of the implemented KBCS.

If the modes of a control system are thought of as discrete entities defining the domain of applicability of some control law for a system, then the core of the KBCS is the *mode switching logic* which is generated by the *mode switching supervisor*. The mode switching logic (MSL) is a state-transition graph decsribing which mode transitions are enables. The MSL is generated by the mode switching supervisor (MSS), in accordance with the constraints of the control system design. The primary symbolic processing capability of the system of a KBCS is resident within the MSS. A control system may have several MSS, for example at each level of a hierarchy.

Thus a typical requirement for a control system is that the MSS always generates a finite state machine MSL. This is a statement in the specification language which becomes an assertion to be proved about the implementation of the KBCS. Another requirement might be that the MSL contain no infinite loops. That is, a control decision is always reached in every situation.

4 The Approach

4.1 Overview

The approach taken was to develop a model based upon modal logic which encompassed the control structure and the semantic content of the KBCS. Statements in the specification language could then be interpreted as assertions to be proved about the formal model. The intent is that the specification be developed in parallel with the KBCS and is refined while the system is being built. The environment in which the KBCS is built is based upon an expert system shell. RTBA (for Real Time Blackboard Architecture), developed at Honeywell S&RC. The developer never has to deal directly with the representation used for V&V purposes.

4.2 Representation of the KBCS

The KBCS is viewed as a *family* of graphs. A given graph in the family, corresponds to a "run" of the KBCS. It is built by tying together a number of graphs representing the knowledge and inferences used, much as a truth maintenance system builds a dependency graph. The graphs are a form of predicate transition net [2].

The "tying together" is done through a control graph which models the decision points and the knowledge sources of the system. In RTBA, in its current form, control of invocation of knowledge sources and oracles is represented separately from the domain knowledge sources. *The approach to V&V presented here presumes that such a separation can be made*, although it need not be done as explicitly as in RTBA.

Starting from the control graph of the KBCS form the path space of this graph consisting of all paths based at START. There is an infinite number of these paths. Let C denote the control graph and P(C) the path space of C, where it is understood that "path" means "path based at START". Each path can be expanded into a form of data flow graph representing the types of information and rules which are actually active when the control procedure follows the given path. These data flow graphs, each of which is a member of P(C), are predicate transition nets in the sense Genrich and Lautenbach [2].

A form of modal logic was developed for making statements about, and establishing properties of P(C), this very large family of graphs. The semantic content of the system is represented by interpreting the family of graphs as a Kripke model of the modal logic.

4.3 A Logic for Reasoning about KBCS

We have adapted a form of modal logic, called *computational tree logic* [1] to support making statements about, and proving properties of, this family of graphs. The use of modal logics as a basis for formal verification methods has been proposed by Hoare, Pratt, and a number of other researchers. This work builds upon their efforts.

The operators in this logic are built to form statements about properties of graphs. Examples of modal operators are: *A* meaning *for all paths*; *E* meaning *for some path*; and *X* meaning *nexttime*.

In the world of formal logic this family of predicate transition nets can be treated as a Kripke model of this modal logic. A Kripke model, is a triple **(G, R, INF)** where *G* is a set, *R* is a relation on the set, and *INF* is a set of inference rules. Intuitively *G* can be viewed as a set of possible worlds. it R tells which worlds are accessible from a given world. *INF* tells how true statements in a world are related to true statements in worlds accessible to that world. In our case, an element of *G* is a path in the control graph. *R* is subset inclusion of paths. *INF* is the set of inference rules for the logical operators described above. This model-theoretic interpretation provides a way to deal explicitly with the semantic content of the expert system. The realization of the model in term of graphs means that many of the computations of interest become linear algebra calculations.

5 Other Issues

If validation and verification are concerns when building a system, it is prudent to consider them when building the environment within which the system will be built. For knowledge-based systems this means having well defined methods for knowledge acquisition, including tools for checking the consistency and completeness of information. There also needs to be a formal language for expressing specifications, which supports refinement and explanation, much in the spirit of

Swartout's work. Without this type of support formal methods have little chance of success in practical terms.

It is well to keep in mind that there are often different levels of software criticality in a system. For example, subsystems of a flight control systems might be classified as; life critical, system critical, or mission critical. The level of V&V appropriate for a subsystem is governed in large part by the criticality level of that subsystem. A weakness of the approach to V&V outlined in this paper is that is does not incorporate a mechanism for tailoring the degree of rigor of V&V procedures to the level of criticality of a the knowledge-based system.

Intertwined with method for V&V are questions about software safety and reliability. The goal of V&V has been to insure that software is reliable in that the implementation meets the specifications and is reasonably free of errors. However techiques for achieving reliability and safety in software are sometimes at odds with the requirements for testing. It can be difficult to test software which has been written to mask faults. It is possible that formal methods for V&V offer a solution to this impasse.

It is possible that V&V may actually become easier for knowledge-based systems than for traditional software. As more capability is moved into compilers through the use of program transformation methods, the specifications move closer to becoming the program. Much of the work of validation may then become a one-time effort of insuring the quality of the compiler.

References

1. Emerson, E.A., and A.P. Sistala, "Deciding Full Branching Time Logic", **Information and Control**, vol. 61, 1984, (175-201).

2. Genrich, H.J., and K. Lautenbach, "System Modelling with High-Level Petri Nets", **Theoretical Computer Science**, vol. 13, 1981, (109-136).

3. Rang, E.R. and N.C. Wood, **The Use of Logic Languages for the Formal Specification of Flight Control Systems**, Honeywell SRC, contract P.O. No. RC 52571 prepared for Lockheed-Georgia Company, April 1986.

4. Cook, Thomas M., "KBS Validation and Verification: Issues for the NASA's Space Station Program", talk given at a conference on validation and verification of KBSs at NASA/AMES, May 14-15, 1987.

5. Levis at al., "Challenges to Control", **IEEE Trans. on Automatic Control**, vol. AC-32, no. 4, April 1987.

6. Ramadge, P.J., and W.M. Wonham, "Supervisory Control of a Class of Discrete Event Processes", **SIAM Journal on Control and Optimization**, Jan. 1987. (206-230).