Research report 194

THE ROLE OF FORMAL SPECIFICATION IN RULE-BASED REAL-TIME AI (EXTENDED ABSTRACT)

IAIN D CRAIG

RR194

This paper describes the role of formal specification in the development of AI systems: it is seen as an important step in the development of reliable real-time AI systems. It discusses some of the issues involved in formal specification and their relationship to current practise and theory. Three completed specifications are described. The paper describes the use of formal methods in the development of AI interpreters and relates formal methods at this level to work in knowledge representation. The role of formal specification in the modelling of experimental systems is also discussed.

Department of Computer Science
University of Warwick
Coventry CV4 7AL
United Kingdom

September 1991

The Role of Formal Specification in Rule-Based Real-Time AI (Extended Abstract)

Iain D. Craig
Department of Computer Science
University of Warwick
Coventry CV4 7AL
UK EC

September 6, 1991

Abstract

This paper describes the role of formal specification in the development of AI systems: it is seen as an important step in the development of reliable real-time AI systems. It discusses some of the issues involved in formal specification and their relationship to current practise and theory. Three completed specifications are described. The paper describes the use of formal methods in the development of AI interpreters and relates formal methods at this level to work in knowledge representation. The role of formal specification in the modelling of experimental systems is also discussed.

1 Introduction

Formal methods, and logic in particular, have been employed in AI from its inception. In [15], McCarthy describes a system that takes problem-solving advice from its human partner: the program represents its knowledge as formulae in logic. In the 1960s, logic was an active research area within AI, with much effort being devoted to the construction of theorem-proving programs and to the representation of knowledge using logic. The landmarks of that period are Robinson's Resolution Method [16] and Green's work on the use of situation calculus in program synthesis [9]. More recently, Hayes

[10,11,12] has argued that logic is the *only* candidate for knowledge representation. Work on so-called "naive" physics has centred on the development of formal theories of the commonsense world (see [13] for a book of that name which is entirely devoted to formal descriptions of physical processes and their logics).

2 Aspects of Specification in AI

In [11], Hayes states that formal (i.e., logical) theories should be developed and then implemented using any representation system that one likes: in other words, Hayes (and others) advocate the formal description of a *domain*. Under Hayes' interpretation, part of the role of a formal statement of the domain is to act as a vehicle for exploring the ontology by means of formal proof: he sees proof as a way of delineating the boundaries of the domain in question. Hayes argues that we should determine the ontology and the limits of the domain *before* implementing the representation. However, the idea that the implementation can be as informal as one would like tends to subvert the notion of stating the domain in logical form.

Almost universally, representation systems are not stated formally, nor are their implementations verified in any way other than trying out a few examples and checking that the "right" (i.e., intuitively correct) answer is forthcoming. With theorem-proving programs, matters are a little different because one already has a good idea of what should be produced, and one can always give an independent verification (via a formal proof). Given the state of knowledge representation, independent verification is not generally possible, although a formal domain theory can be used to guide one's intuitions (in the case of a disparity between intuition, formal theory and implementation, which is right?). What is needed is a better guarantee that the finished system will conform to the standards imposed by the domain theory.

Arguments of the above kind suggest that the AI systems that we implement should be subjected to the kinds of rigour that we are sometimes prepared to accept when modelling a domain. Furthermore, with the application of AI systems to real-time and safety-critical problems, the correctness of a system becomes an important issue. Clearly, there are a number of different aspects to the correctness problem for AI systems:

- The correctness (and appropriateness) of the knowledge encoded in the system.

- The correctness of the implementation (the representation and the inference engine).
- The correctness of the mapping between the domain and its representations and between the inferences licensed by the domain theory and those actually generated by the implementation.

In the Software Engineering literature, the second aspect is often cited, but the first and third are rarely, if ever, articulated. The first aspect appears to pose the greatest problems, particularly for real-time systems: these systems usually depend upon more than one person's expertise or knowledge; typically, one places trust in the judgements of experts. However, there will inevitably be times when experts differ in their interpretations: the problem posed by the first aspect is that of reconciling different accounts of the same or similar phenomena (the problem is posed even by domains in which we are all expert). The third aspect is less problematic, and deals with the fidelity of the implementation: if one has a formal statement of a particular domain and a formal specification of a representation system (which is taken to include an inference-making component), can it be shown that the implementation respects the domain? If such properties can be demonstrated, one will have greater confidence in the resulting system.

Considered in isolation, the second of the three aspects is similar to the usual requirement in Software Engineering. For this reason, formal specification seems applicable to the development of AI systems. In real-time and safety-critical applications, one wants the best possible guarantee that the resulting software will behave as one wants. The second aspect concentrates on what might be called the *interpreter* of the system. Unless the interpreter is experimental (as is the ELEKTRA system—see below), it tends not to change over the course of a project. Given the static nature of the basic software, formal specification is applicable and gives the kinds of assurance that are normally associated with it. It is worth making this point because it is often claimed in AI that Software Engineering techniques are inapplicable because of the exploratory nature of AI systems: a formal specification is useless when one wants to make some change, and the task of producing a formal specification becomes too much of a burden if the system is rapidly changing. This point is very often made in conjunction with the claim that the program *is* the (standard representation of the) theory being presented (e.g., [17]). I will not present a counter-argument to this position here (such an argument deals with the distinction between form and content), and will only re-emphasise the point that the initial view of formal specification applies to those parts of an AI system which interpret (in the Computer Science sense) domain knowledge: in other words, I consider that there is a clear separation between what is being represented and what is doing the representation and inference.

3 Interpreter Specification

It is worth being clear about what one is doing when one gives a formal specification of an AI system. What the initial proposal for specification amounts to is the specification of the interpretational mechanisms that drive the system. The sense in which "interpretation" should be taken is that of Computer Science, not of logic. The mechanisms that are being specified are entirely *syntactic*, and not semantic. The central concern of the specification is the definition of a set of structures and a means of transforming them. The transformations are conceived of as being of a syntactic nature: they merely determine what the transformations of the structures used by the interpreter may be. The semantics is not part of the formal specification: it must be supplied as a separate component, although it informs the specification process. In other words, the interpretation (in the logical sense) must be provided *in addition to* the interpreter (in the Computer Science sense).

The specification involves the determination of the structures that the interpreter uses and it also determines the mechanisms which transform those structures. For a production system, the specification defines working memory elements and rules without reference to the symbols which will actually be used in the implementation of any particular application. The production rule specification then describes or models the process of matching rule conditions, executing actions, the various operations on working memory, and so on. This can be viewed as a way of constraining an implementation or it can be viewed as providing an abstract model of the system: both of these views have their part to play. Once the specification has been produced, it can be refined to an implementation (which has all the guarantees that formal specification provides), or it can be used to determine properties of the system (e.g., commutativity and equivalence of operations, termination); the specification can play both these roles simultaneously.

It is important to be clear about what a formal specification provides and what it does not. A formal specification can be considered as a *model* of the system. As such, it forms the basis for reasoning about the system and for exploring some of its properties. If the specification is refined to an implementation, each stage of the refinement can be viewed as a progressively more detailed description of the system as it will be when complete: the relationships between the representations developed during refinement can be illuminating. Formal specification also has the benefit of making one think carefully about the system and what it is intended to do.

A formal specification does not, directly at least, give information about the time-space complexity of the resulting system: time and space complexity are properties of individual algorithms, not of abstract models. After refinement, it may be possible to engage in such an analysis, but this requires that algorithms have been inferred. For real-time systems, complexity measures may be of considerable importance, but they are not deducible from abstract specifications—this is a limit of the method (as is the fact that an excellent specification may lead to an intollerably slow or large implementation).

4 Work to Date

The formal specification of AI systems, particularly their "basic" software, has not attracted much attention. In theorem-proving, it is common for a program to have a formally stated specification (for example, [14]), although the implementation is almost invariably conducted using informal techniques and then verified by appeal to results in logic. The KRYPTON knowledge representation system [1] has been given a model-theoretic semantics [2], but was implemented using informal methods.

Recently, however, there has been some work on the specification of AI software: this work has been done at Warwick University (by the author) and at RMCS, Shrivenham. The work in these institutions differs somewhat in its aims and methods (and by the fact that the author used Z, while RMCS used VDM). The RMCS work is concerned with the formal specification of *particular* systems: this requires an analysis of the problem domain and its ontology. The author's work has been of a more problem-independent nature, and the specifications that have been developed are more abstract than those at RMCS.

The author's work began with the question as to whether a formal specification of a blackboard system was possible. The Appendix to [5] contains the specification of a blackboard shell: the specification is written in a mixture of English and LISP. The specification attempts completeness, but the questions of ambiguity, perspicuity and completeness were clear. Blackboard systems are, furthermore, plagued by the informality of the architecture and by the often different terminology used by workers (for example, 'Knowledge Source Instance' is a concept equivalent to 'Knowledge Source Activation Record'): what appeared necessary was some means of presenting an interpretation of the architecture in neutral terms. Also, in his doctoral thesis (published as [3]), the author suggested that a formal definition of the CASSANDRA architecture be produced as an unambiguous standard.

The two specifications presented in [4] deal with these issues. The blackboard specification answers the question as to whether a formal specification is possible: the specification represents an existence proof—formal specification *is* possible. This specification is intended as a statement of one possible interpretation of the architecture[1], and contains proofs of a number of properties of the system (for example, the property that an entry can reside on exactly one abstraction level): the proofs were considered an important aspect of the specification because they make explicit some properties which are often ignored in the literature (the uniqueness property mentioned above is one example). The blackboard system is specified at a high level of abstraction (parts of it are refined in Appendix A of [4]), partly because of the tutorial nature of the book, and partly because it allowed properties to be proved in the clearest possible manner.

The CASSANDRA specification was intended to be *definitional* in nature. The aim was to present a clear statement of what constitutes a CASSANDRA system. The resulting specification borrows heavily from the blackboard specification: this is not surprising because the CASSANDRA architecture is based on the older one. Formal properties of the system are stated but no proofs given: this is because of space limitations, not because of an *a priori* difficulties. Nevertheless, a relatively complete specification was produced (the reasons for the qualification are explained in the next paragraph).

The results of these two large specifications were encouraging. The scope of both exercises was the specification of the central components of the two architectures, including control structure. There were, as is to be expected, problem areas. The specification of knowledge sources posed an interesting problem.

The knowledge source model adopted includes a precondition which is composed of arbitrary implementation-language code. Without detailed knowledge of the application, it is not possible to say what the precondition will be: thus, it becomes impossible to give preconditions a formal

[1] Although not quite the author's.

specification. The specification was made partial by this difficulty. What *was* possible was to define an invariant which all preconditions must satisfy (essentially that the blackboard state is unchanged by executing a precondition). The problem with preconditions carried over to the CASSANDRA specification.

A second problem that troubles both specifications is that the scheduler cannot be specified without knowledge of the problem under attack. This is because control in both systems is based, at least in part, upon values that are stored in the database. Simply put, without knowledge of the attributes and values that are important in scheduling, a scheduler cannot be defined. The solution that was adopted was to assume that there is some function which rates knowledge source instances and re-orders the control agenda.

The third large-scale exercise was the specification of ELEKTRA, a reflective production rule interpreter [6,7,8]. This specification involved the complete specification of a relatively conventional rule interpreter (minus file and user interfaces; unification was also omitted—the author had a specification available, as well as an implementation refined from it): this specification served as the basis for the specification of the reflective version. Facilities for meta-level processing were included and the standard control structure heavily revised. The specification of the reflective version was subsequently refined into an implementation in Scheme; a new version in CommonLISP, also based on the refinement, is now under construction.

In the case of ELEKTRA, the aims of the specification exercise were different. One aim was the conventional one of generating an implementation. However, ELEKTRA is an experimental system and one that reasons about itself. To reason about itself, ELEKTRA needs a representation of its knowledge and control behaviour; in addition, it has access to the components of its own interpreter (so that it can be directly called from rules). In order to make the system work, a theory of ELEKTRA was needed. This theory needed to be formal and also needed to cover the entire interpreter. The specification served a number of purposes:

- It acted as a statement of the ELEKTRA theory. The theory is formal and is written in Z.

- It enabled the critical parts of the interpreter to be identified. The critical parts are those components which rules need to access to engage in reflection.

- It served as a vehicle for trying out ideas. At the start of the project, there were a number of different ways of forming the theory and the implementation. The specification that is presented in [6] is the result of considerable experimentation, all of which took place at the level of the specification.

As the last item suggests, formal specification took the place of implementation: ideas were tried out by writing them down in Z and then drawing out their consequences—all of this was done mathematically. The relative success of this part of the project indicates that formal specification can be used as a method for experimenting without the need for a running program. In other words, the mathematics replaced the programming language (this is the point alluded to above in connection with the impossibility of formal specification in AI).

The result of the ELEKTRA specification looks much more like a kit of parts than a complete system. The user is able to modify the interpreter in various ways (some at runtime). The system is able to engage in reflection, in particular reflection on its own control behaviour. In [8], various rule interpreters are presented in the form of production rules: ELEKTRA is able to interpret these rules, thus changing the default nature of the ELEKTRA system itself. When run, the rule interpreters interpret other rules in the system (in some cases, they can also interpret themselves). Because the interpreters are written as ordinary rules, they can be the subject of reasoning and manipulation by still more rules.

5 Future Work

The work reported above has been sequential in nature: Z was used as the specification language because of its schema calculus, the range of mathematical constructs defined as standard, and because of its pleasing semantics (essentially, a first-order typed logic). Z is, of course, for use in the specification of *sequential* systems. In the case of the blackboard and CASSANDRA architectures, a sequential interpretation is possible, but both are amenable to parallelisation; the CASSANDRA architecture was designed for distributed AI applications. In both cases, it seems desirable to develop a specification which brings out the concurrency or which allows the specification of distributed systems. Concurrency is becoming an active research topic in AI and its application to real-time problems is clear. For these reasons, the next step is to investigate concurrent and distributed AI systems from the viewpoint of formal specification.

The second area where more work is needed is in the area of semantics. As was mentioned above, both the blackboard and CASSANDRA specifications were rendered incomplete by the absence of a semantics for their representations. The provision of a semantics is part of the formal enterprise outlined in section one above: without a semantics, it is not possible to determine the mapping between formal domain models and the structures used in the implementation of the interpreter—it is not possible, in other words, to determine that the interpreter meets the semantic criteria imposed by the model of the problem domain. What is needed, in order to perform this task properly, is a demonstration that the inferences licensed by the domain model are correctly performed by the implementation.

The author has given a denotational semantics for the knowledge sources used in CASSANDRA (unpublished note). The semantics relates the components of knowledge sources to the state of the machine on which they execute. No semantics has been given for the attribute-value representation used by entries, however; a denotational semantics for ELEKTRA is being contemplated as an extension to the (operational) theory that was used in its development. A denotational style was considered for a number of reasons:

- Denotational semantics deals with the objects that are denoted by structures in the representation language; the denotations are related to the state of the processor. With other criteria lacking, the behaviour of the implementation becomes important in the assessment of correctness. In addition, the symbols of the representation language must somehow be related to the objects in the real world (Tarskian semantics also has a denotational aspect).

- Knowledge sources in blackboard and CASSANDRA systems are essentially transformational in nature: they transform the state of the database. Model-theoretic semantics, although it deals with inference, does not treat inference as a transformational process (the consequences of an axiom are, in a sense, "out there" waiting to be discovered). An account of the semantics of an AI system should deal with this property.

The work on the denotational semantics is highly inconclusive. Such a model can serve a variety of purposes: it can be used to categorise things, or it can serve as a model for a theory. What denotational semantics, like Tarskian semantics, cannot do is to provide a definite connection between the AI system's representations and the outside world (similarly, neither can assign unique meanings to expressions). The problem of semantics, particularly for the representations used in real-time AI systems, is an important issue that needs more investigation. Work on concurrent AI systems may lead to a clearer understanding of the problems involved: this is because, in concurrent systems, the concept of a dynamic environment external to the individual process is of paramount importance, as are the causal connections between processes.

References

[1] Brachman, R.J., Fikes, R.E. and Levesque, H.J., KRYPTON: A Functional Approach to Knowledge Representation, *IEEE Computer*, Vol. 16, pp. 67-73, 1983.

[2] Brachman, R.J. and Levesque, H.J., The Tractability of Subsumption in Frame-Based Description Languages, *Proc. AAAI-84*, pp. 34-37, 1984.

[3] Craig, I.D., *The CASSANDRA Architecture*, Ellis Horwood, Chichester, 1988.

[4] Craig, I.D., *The Formal Specification of Advanced AI Architectures*, Ellis Horwood, Chichester, 1991.

[5] Craig, I.D., *Blackboard Systems*, Ablex Publishing Corp., Norwood, NJ, *in prep.*.

[6] Craig, I.D., *The Formal Specification of ELEKTRA*, Research Report, Department of Computer Science, University of Warwick, *in prep.*, 1991.

[7] Craig, I.D., *ELEKTRA: A Reflective Production System*, Research Report No. 184, Department of Computer Science, University of Warwick, 1991.

[8] Craig, I.D., *Rule Interpreters in ELEKTRA*, Research Report No. 191, Department of Computer Science, University of Warwick, 1991.

[9] Green, C., Application of theorem proving to problem solving, in Meltzer, B. and Michie, D., (eds.), *Machine Intelligence 4*, Edinburgh University Press, 1969.

[10] Hayes, P.J., The Naive Physics Manifesto, in Michie, D. (ed.) *Expert Systems in the Microelectronic Age*, Edinburgh University Press, 1979.

[11] Hayes, P.J., The Second Naive Physics Manifesto, in. Hobbs, D. and Moore, R.C., (eds.), *Formal Theories of the Commonsense World*, Ablex Publishing Corp, Norwood, NJ, 1985.

[12] Hayes, P.J., Ontology for Liquids, in Hobbs, D. and Moore, R.C., (eds.), *Formal Theories of the Commonsense World*, Ablex Publishing Corp, Norwood, NJ, 1985.

[13] Hobbs, D. and Moore, R.C., (eds.), *Formal Theories of the Commonsense World*, Ablex Publishing Corp, Norwood, NJ, 1985.

[14] Loveland, D., *Automated Theorem Proving: A Logical Basis*, North-Holland, New York, 1978.

[15] McCarthy, J., Programs with Common Sense, *Proc. Symposium on Mechanisation of Thought*, Vol.1, pp. 77-84, HMSO, 1959.

[16] Robinson, J.A., A Machine-oriented Logic Based on the Resolution Principle, *JACM*, Vol. 12, pp. 23-41, 1965.

[17] Wilensky, R., *Planning and Understanding*, Addison-Wesley, Reading, MA, 1983.