Rational Engines for BDI Architectures

Deepak Kumar
Department of Computer Science
State University of New York at Buffalo, Buffalo, NY 14260
kumard@cs.buffalo.edu

1 Motivations

A survey of AI systems would reveal that it is somewhat awkward to do acting in reasoning (or logic-based) systems (but it is convenient to talk about representational and reasoning issues using them), and it is awkward to study reasoning and representational issues in systems designed for acting/planning. Thus, most "good" planning/acting systems are "bad" KRR systems and vice versa. For example, in a recent symposium on "Implemented KRR Systems" [Rich 1991b] out of a total of 22 KRR systems presented only 4 systems had capabilities for representation and reasoning about actions/plans (RHET [Allen 1991], CYC [Lenat and Guha 1991], CAKE [Rich 1991a] and SNePS [Shapiro 1991]). The work presented in this paper presents an approach that bridges this "representational/ behavioral gap." We extend the ontology of an intensional KRR system to facilitate representation and reasoning about acts and plans. I.e. a computational cognitive agent modeled using the extended ontology has representations for beliefs, acts, and plans, and is able to reason about them. I.e., the agent is able to represent its beliefs and desires (the 'B' and the 'D' of 'BDI').

In most current AI architectures reasoning is performed by an inference engine and acting is done under the control of some acting executive (or a plan/act interpreter). Our approach is based on the viewpoint that logical reasoning rules implicitly specify the act of believing. Thus the inference engine can be viewed as a *mental actor*. This enables us to establish a closer relationship between rules of inference and rules of acting (or planning). Believing is a state of knowledge; acting is the process of changing one state into another. A reasoning rule can be viewed as a rule specifying an act—that of believing some previously non-believed proposition—but the believe action is already included in the semantics of the propositional connective. McCarthy (1986) also suggested that inference can be treated as a mental action. This suggests that we can integrate our models of inference and acting by eliminating the acting executive (plan/act interpreter). These ideas are used in developing a computational model— called a *Rational Engine*, that is a unified model of acting and inference and can be used for modeling rational cognitive agents and their behavior. Acting and reasoning about beliefs, actions, and plans is performed by a single component— the Rational Engine. The rational engine implements the underlying logic as well as notions of intentionality (the 'I' of 'BDI').

The work presented here has evolved from research involved in extending a semantic network-based KRR system, SNePS (whose rational engine called SNeRE is described in [Kumar 1993a]), into a BDI architecture. In this paper we use an object-oriented approach to describe the BDI architecture. The resulting architecture is independent of, yet isomorphic to, the SNePS KRR formalism. The resulting BDI architecture enjoys all the advantages of object-oriented design— the ontology is easily extendible, as is the underlying logic, and amenable to a concurrent implementation [Kumar 1993b].

2 A Object-Oriented KR Hierarchy

The representational formalism is described as a conceptual object-oriented hierarchy. This is depicted in Figure 1. In an intensional representational framework, anything a cognitive agent can think about is termed a "mental concept" or a conceptual entity. More specifically these can be individuals, beliefs (propositions), or acts. In addition to standard beliefs that an agent is able to represent, we also define a special class of beliefs called *transformers*. A *transformer* is a propositional representation that accounts for various notions of inference and acting. A transformer is a representation that specifies a belief/act transformation. It has two parts—($\langle a\rangle, \langle b\rangle$), where both $\langle a\rangle$ and $\langle b\rangle$ can specify either beliefs or some act. Transformations can be applied in forward and/or backward chaining fashion. Using a transformer in forward chaining is equivalent to the interpretation "after the agent believes (or intends to perform) $\langle a\rangle$, it believes (or intends to perform) $\langle b\rangle$." The backward chaining interpretation of a transformer is, "if the agent wants to believe (or perform) $\langle b\rangle$, it must first believe (or perform) $\langle a\rangle$." Since both $\langle a\rangle$ and $\langle b\rangle$ can be sets of beliefs or an act, we have four types of transformers— *belief-belief, belief-act, act-belief,* and *act-act.*

Belief-Belief Transformers: These are standard reasoning rules (where $\langle a\rangle$ is a set of antecedent belief(s) and $\langle b\rangle$ is a set of consequent belief(s)). Such rules can be used in forward, backward, as well as bidirectional inference to derive new beliefs. We will call these AntCq transformers. Hereafter we will use the notation $\langle a\rangle \rightarrow \langle b\rangle$ to write them. For example "All blocks are supports" is represented as M1!: $\forall x[\mathtt{Isa}(x,\mathtt{BLOCK}) \rightarrow \mathtt{Isa}(x, \mathtt{SUPPORT})]$. In addition to the connective above (which is also called an or-entailment), our current vocabulary of connectives includes and-entailment, numerical-entailment, and-or, thresh, and non-derivable. Other quantifiers include the existential, and the numerical quantifiers (see [Shapiro and Group 1989]). Given the object-oriented design of the architecture one can define any additional classes of connectives depending on their own logical commitments.

Belief-Act Transformers: These are transformers where $\langle a\rangle$ is a set of belief(s) and $\langle b\rangle$ is a set acts. Used during backward chaining, these can be propositions specifying preconditions of actions, i.e. $\langle a\rangle$ is a precondition of some act $\langle b\rangle$. We will call it a PreconditionAct transformer and write it as a predicate PreconditionAct($\langle a\rangle$,$\langle b\rangle$). For example, the sentence "Before picking up A it must be clear" may be represented as M26!: PreconditionAct(Clear(A),PICKUP(A)).

Used during forward chaining, these transformers can be propositions specifying the agent's desires to react to certain situations, i.e. the agent, upon coming to believe $\langle a\rangle$ will form an intention to perform $\langle b\rangle$. We will call these WhenDo transformers and denote them as WhenDo($\langle a\rangle$,$\langle b\rangle$) predicates.

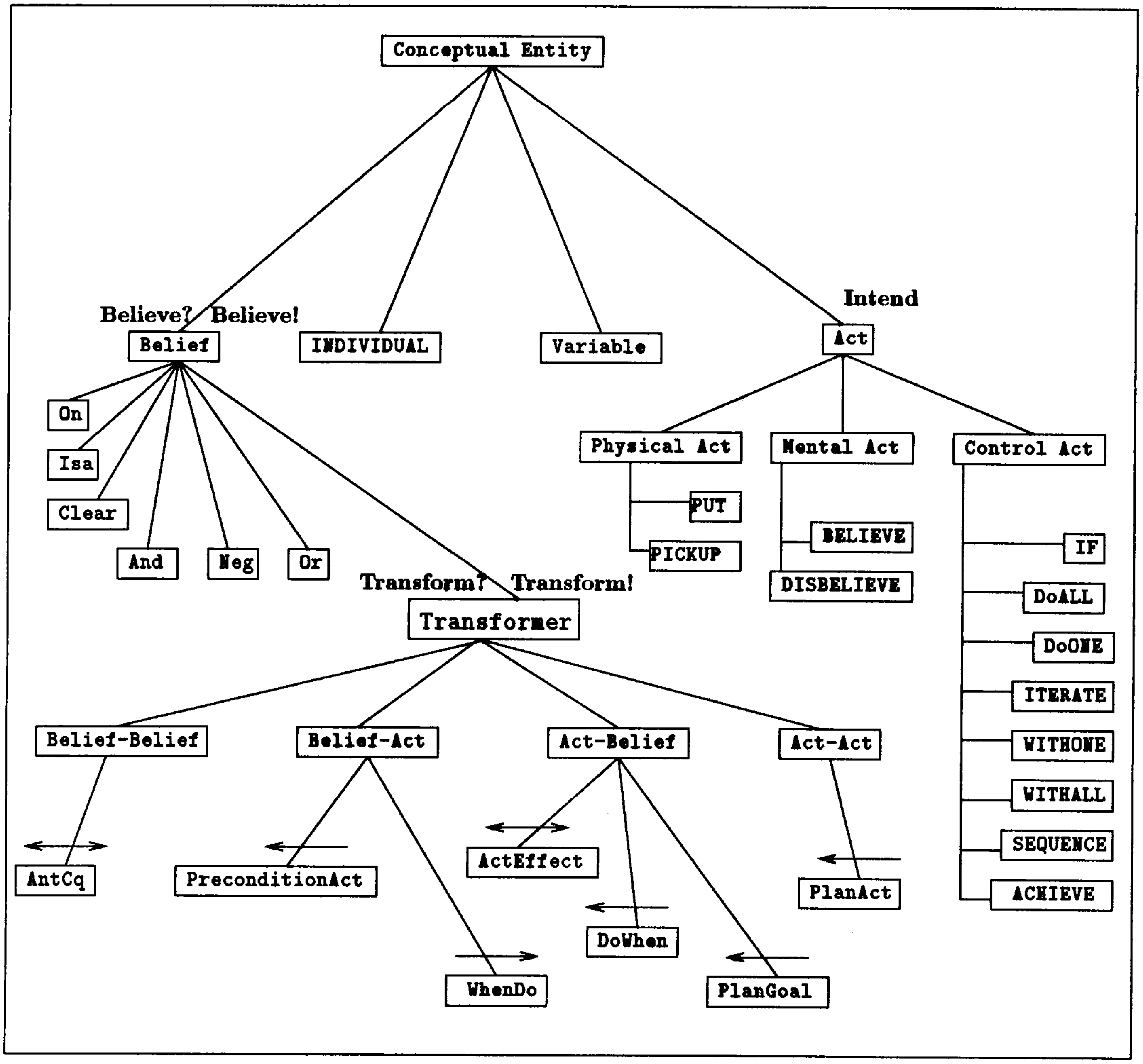

Figure 1: An Object-Oriented KR Formalism

Act-Belief Transformers: These are the propositions specifying effects of actions as well as those specifying plans for achieving goals. They will be denoted ActEffect and PlanGoal transformers respectively. The ActEffect transformer will be used in forward chaining to accomplish believing the effects of act ⟨a⟩. For example, the sentence, "After picking up A it is no longer clear" is represented as M30!: ActEffect(PICKUP(A),¬Clear(A)). It can also be used in backward chaining during the plan generation process (classical planning). The PlanGoal transformer is used during backward chaining to decompose the achieving of a goal ⟨b⟩ into a plan ⟨a⟩. For example, "A plan to achieve that A is held is to pick it up" is represented as M56!: PlanGoal(PICKUP(A),Held(A)).

Another backward chaining interpretation that can be derived from this transformer is, "if the agent wants to believe ⟨b⟩, it must perform ⟨a⟩," which is represented as a DoWhen transformer. For example, "Look at A to find out its color" can be represented as DoWhen(LOOKAT(A),Color(A)). For example, using this transformer, one can represent general desires like, "if something is broken, fix it," as the belief M43!: ∀ [WhenDo(Broken(x),Fix(x)].

Act-Act Transformers: These are propositions specifying plan decompositions for complex actions (called PlanAct transformers), where ⟨b⟩ is a complex act and ⟨a⟩ is a plan that decomposes it into simpler acts. For example, in the sentence, "To pile A on B first put B on the table and then put A on B" (where piling involves creating a pile of two blocks on a table), piling is a complex act and the plan that decomposes it is expressed in the proposition M71!: PlanAct(SNSEQUENCE(PUT(B,TABLE),PUT(A,B)),PILE(A,B)).

Our present model of acting is based upon a state-change model (see [Kumar and Shapiro 1991]). We identify three types of states —external world states, mental states (belief space), and intentional states (agent's current intentions). Accordingly, we identify three classes of actions —physical actions, mental actions, and control actions that bring about changes in their respective states. Thus PICKUP is a physical action, we have BELIEVE and DISBELIEVE as mental actions whose objects are beliefs, and control actions are described below. Acts can be *primitive* or *complex* (not shown in the figure). A primitive act has an effectory procedural component which is executed when the act is performed. Complex acts have to be decomposed into plans.

Plans, in our ontology, are also conceptual entities. However, like acts, we do not define a separate class for them as they are also acts— albeit control acts. Control acts, when performed, change the agent's intentions about carrying out acts. Our repertoire of control actions includes *sequencing* (for representing linear plans), *conditional, iterative*, disjunctive (equivalent to the OR-splits of the Procedural Net formalism [Sacerdoti 1977, Wilkins 1988]), *conjunctive* (AND-splits), *selective*, and *achieve* acts (for goal-based plan invocation). These are summarized in Table 1. These control acts are capable of representing most of the existing plan structures found in traditional planning systems (and more). We should emphasize, once again, that since plans are also conceptual entities (and represented in the same formalism) they can be represented, reasoned about, discussed, as well as followed by an agent modeled in this architecture.

3 The Rational Engine

Next, we will outline details of the integrated reasoning and acting module— called a *Rational Engine* (as opposed to an inference engine that only performs inference). A Rational Engine is the 'operational' component of the architecture (the interpreter) that is responsible for producing the modeled agent's reasoning and acting (and reacting) behavior. It is specified by three types of methods (or messages)—

Believe– A method that can be applied to beliefs for assertional or querying purposes. Consequently there are two versions—

- *Believe*(p)!– where p is a belief, the method denotes the process of asserting the belief, p, in the agent's belief space. It returns all the beliefs that can be derived via forward chaining inference/acting.
- *Believe*(p)?– where p is a belief, it denotes the process of querying the assertional status of p. It returns all the beliefs that unify with p and are believed by the modeled agent either explicitly or via backward chaining inference/acting.

Intend– that takes an act as its argument (*Intend*(a)) and denotes the modeled agent's intention to perform the act, a.

Transform– These methods enable various transformations when applied to transformers. Corresponding to backward and forward chaining interpretations there are two versions— *Transform*? and *Transform*!, respectively.

Notice that the first two also correspond to the propositional attitudes of belief and intention. The methods *Believe* and *Intend* can be invoked by a user interacting with the agent. New beliefs about the external world can be added to the agent's belief space by using *Believe*! and queries regarding agent's beliefs are generated using *Believe*?. These methods, when used in conjunction with transformers lead to chaining via the semantics of the transformers defined above. The architecture also inherently provides capabilities for consistency maintenance. Each specific object that is a belief can have slots for its underlying support. The support is updated and maintained by the *Believe* methods as well as the mental actions BELIEVE and DISBELIEVE (together they form the TMS). The effectory procedures for BELIEVE and DISBELIEVE are implemented as belief revision procedures. We have found that such an integrated TMS facility simplifies several action and plan representations (see [Kumar and Shapiro 1993] for details). The *Intend* method is used to specify the fulfillment of agent's intentions by performing acts. All these methods can be specified (and specialized) for the hierarchy as well as inherited. Thus, domain specific acts (physical acts) will inherit the standard method for the agent to accomplish its intentions (i.e. the specific theory of intentionality employed), where as specializations of the *Intend* method can be defined for mental and control acts (to implement the semantics of respective acts).

Thus, an object-oriented design not only provides a uniform representational formalism, it also facilitates an extendible ontology. The semantics of representations is described by reasoning and acting methods that can be either individually specified or inherited and further specialized, as the case may be. Further, we would also like to claim that the representational formalism is 'canonical' in that its user interface (which is mainly defined via 'print methods') is also extendible. For example, the same object (say, a belief proposition) can be displayed as a frame, a predicate, a semantic network, or some other communicational

Control Action	Description
SEQUENCE(a_1,a_2)	The acts a_1 and a_2 are performed in sequence. Example: SEQUENCE(PICKUP(A),PUT(A,TABLE)) is the act of first picking up A and then putting it on the table.
DoONE(a_1,...,a_n)	One of the acts $a_1,\ldots,a_n$ is performed. Example: DoONE(PICKUP(A),PICKUP(B)) is the act of picking up A or picking up B.
DoALL(a_1,...,a_n)	All of the acts $a_1,\ldots,a_n$ are performed in some order. Example: DoALL(PICKUP(A),PICKUP(B)) is the act of picking up A and picking up B.
IF((p_1,a_1),...,(p_n,a_n))	Some act a_i whose p_i is believed is performed. Example: IF((Clear(A),PICKUP(A)),(Clear(B),PICKUP(B))) is the act of either picking up A (if A is clear) or picking up B (if B is clear).
ITERATE((p_1,a_1),...,(p_n,a_n))	Some act in a_i whose corresponding p_i is believed is performed and the act is repeated. Example: ITERATE((Clear(A),PICKUP(A)),(Clear(B),PICKUP(B))) the act of picking up A (if A is clear) and picking up B (if B is clear).
ACHIEVE(p)	The act of achieving the proposition p. Example: ACHIEVE(Clear(A)) is the act of achieving that A is clear.
WITHONE(x,y,...)(p(x,y,...),a(x,y,...))	Find some x, y, etc that satisfy p and perform the act a on them. Example: WITHONE(x)(Held(x), PUT(x, TABLE)) is the act of putting on the table something that is being held.
WITHALL(x,y,...)(p(x,y,...),a(x,y,...))	Find all x, y, etc that satisfy p and perform the act a on them. Example: WITHALL(x)(Held(x), PUT(x, TABLE)) is the act of putting on the table everything that is being held

Table 1: Summary of control actions

entity (*ala* KIF). At present, these ideas are implemented using SNePS (for *S*emantic *N*etwork *P*rocessing *S*ystem) [Shapiro and Rapaport 1987, Shapiro and Group 1989] —an intensional, propositional, semantic network system used for modeling cognitive agents. SNePS-based cognitive agents have network representations for individuals, propositions, deduction rules, actions, acts, and plans. Acting and reasoning about beliefs, actions, and plans, is performed by a single component, SNeRE— the *SNe*PS *R*ational *E*ngine.

4 Related Work

Our use of the term 'BDI Architectures' comes from [Georgeff 1987] that mentions the challenges of designing rational agents capable of goal-directed as well as reactive behavior based on the attitudes of beliefs, desires, and intentions. Georgeff specifically mentions that, 'the problem that then arises is specifying properties we expect of these attitudes, the way they interrelate, and the ways they determine rational behavior in a situated agent.' As explained in Section 1, we have taken the task of designing BDI architectures by defining a unified intensional representational formalism; identifying the semantic interrelationships between beliefs, desires, and intentions; capturing these into the idea of transformers; and finally designing a rational engine that brings about rational behavior based on these entities.

There has been work describing formal BDI models [Cohen and Levesque 1990, Rao and Georgeff 1991]. There are also architectures that have been proposed that address various issues relating to rational agency. For instance [Bratman *et al.* 1988, Pollack 1991] describe a high-level BDI architecture that specifically focuses on issues relating to resource boundedness of rational agent behavior. Their work explores the hypothesis that plans, once committed, in addition to guiding the agent's actions, also constrain the agent's reasoning behavior. [Rao and Georgeff 1991, Rao and Georgeff 1992] have also studied formally the nature of intention and commitment in the context of rational agent behavior. The architecture reported in [Rao and Georgeff 1992] provides a very simplistic representation of beliefs (thus suffering from some of the concerns mentioned in Section 1) together with a transition network-like formalism for plans. It is a (limited, though successful) attempt towards bridging the their earlier work on PRS [Georgeff *et al.* 1985, Georgeff and Lansky 1987] and their later work on formal foundations of rational agents [Rao and Georgeff 1991]. The work presented here complements these models. It provides a general representational framework which these models lack. At the same time, it can facilitate easy incorporation of their ideas by virtue of the extendibility of the design.

We have taken a unified approach to representations. [Drummond 1987] expresses the need for a single unified formalism for representing beliefs, acts, and plans. This facilitates a single reasoning module to be able to reason about beliefs, acts, and plans. We have taken this approach a step further by explicitly identifying the semantic relationship between inference and acting so that a single module, a rational engine, in addition to reasoning, is also responsible for carrying out physical acts and plans. In our formalism, act representations are different from standard operator-based representations of classical planning/acting systems. Elsewhere [Kumar and Shapiro 1993], we have also shown how even simple act representations can benefit from an integrated TMS. In the presence of a TMS the even the simplest acting model (that of adding and deleting the act's effects) implements the extended STRIPS assumption. As a result, ours is a deductive approach to acting. While this leads to tractability concerns, we feel that it provides consistency in the modeled agent's belief space and forms basis for rational behavior. This also facilitates a deductive approach to hierarchical plan decomposition (specific `PlanAct` and `PlanGoal` propositions can be deduced in order to find plan decompositions). Search during reasoning/acting/plan decomposition is focused by means of some KR principles, the Uniqueness Principle being one (there is a one-to-one correspondence between instances and intensional entities) [Kumar 1993a]. The Uniqueness Principle helps focus the chaining (method/message propagation) through a restricted set of entities.

The object-oriented approach provides a promising approach to building BDI architectures. It can be used to implement a unified representational formalism that bridges the gap between classical approaches to representation/acting/planning and the emerging paradigms for designing and implementing integrated intelligent architectures.

References

[Allen 1991] James Allen. The RHET System. In Charles Rich (Guest Editor), editor, *SIGART BULLETIN Special Issue on Implemented KRR Systems*, pages 1–7, June 1991.

[Bratman *et al.* 1988] Michael E. Bratman, David J. Israel, and Martha E. Pollack. Plans and Resource-Bounded Practical Reasoning. *Computational Intelligence*, 4(4), 1988.

[Cohen and Levesque 1990] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3), 1990.

[Drummond 1987] Mark E. Drummond. A representation of action and belief for automatic planning systems. In Michael P. Georgeff and Amy L. Lansky, editors, *Reasoning about Actions and Plans - Proceedings of the 1986 Workshop*, pages 189–212, Los Altos, CA, 1987. AAAI and CSLI, Morgan Kauffmann.

[Georgeff and Lansky 1987] Michael. P. Georgeff and Amy. Lansky. Procedural knowledge. Technical Note 411, AI Center, SRI International, 1987.

[Georgeff *et al.* 1985] M. P. Georgeff, A. Lansky, and P. Bessiere. A procedural logic. In *Proceedings of the 9th IJCAI*, 1985.

[Georgeff 1987] Michael P. Georgeff. Planning. In *Annual Reviews of Computer Science Volume 2*, pages 359–400. Annual Reviews Inc., Palo Alto, CA, 1987.

[Kumar and Shapiro 1991] Deepak Kumar and Stuart C. Shapiro. Architecture of an intelligent agent in SNePS. *SIGART Bulletin*, 2(4):89–92, August 1991.

[Kumar and Shapiro 1993] Deepak Kumar and Stuart C. Shapiro. Deductive Efficiency + Belief Revision: How they affect an ontology of actions and acting. *Journal of Experimental and Theoretical AI (JETAI)*, 5(1), 1993. Forthcoming.

[Kumar 1993a] Deepak Kumar. A Unified Model of Acting and Inference. In *Proceedings of the Twenty-Sixth Hawaii International Conference on System Sciences.* IEEE Computer Society Press, Los Alamitos, CA, 1993.

[Kumar 1993b] Deepak Kumar. An AI Architecture Based on Message Passing. In James Geller, editor, *Proceedings of The 1993 AAAI Spring Symposium on Innovative Applications of Massively Parallel Architectures.* AAAI Press, March 1993.

[Lenat and Guha 1991] Douglas B. Lenat and Ramanathan V. Guha. The Evolution of CYCL, The CYC Representation Language. In Charles Rich (Guest Editor), editor, *SIGART BULLETIN Special Issue on Implemented KRR Systems*, pages 84–87, June 1991.

[McCarthy 1986] John McCarthy. Mental situation calculus. In Joseph Y. Halpern, editor, *Theoretical Aspects of Reasoning about Knowledge—Proceedings of the 1986 Conference*, page 307, 1986.

[Pollack 1991] Martha E. Pollack. Overloading Intentions for Efficient Practical Reasoning. *Noûs*, XXV(4):513–536, September 1991.

[Rao and Georgeff 1991] Anand S. Rao and Michael P. Georgeff. Modeling Rational Agents within a BDI-Architecture. In *Principles of Knowledge Representation and Reasoning— Proceedings of the Second International Conference(KR91*, pages 473–485. AAAI, IJCAI, CSCSI, April 1991.

[Rao and Georgeff 1992] Anand S. Rao and Michael P. Georgeff. An Abstract Architecture for Rational Agents. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *Proceedings of the 2nd Conference on Principles of Knowledge Representation and Reasoning*, pages 439–449, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

[Rich 1991a] Charles Rich. CAKE: An Implemented Hybrid KR and Limited Reasoning System. In Charles Rich (Guest Editor), editor, *SIGART BULLETIN Special Issue on Implemented KRR Systems*, pages 120–127, June 1991.

[Rich 1991b] Charles Rich. Special Issue on Implemented Knowledge Representation and Reasoning Systems—Letter from the Guest Editor. *SIGART Bulletin*, 2(3), June 1991.

[Sacerdoti 1977] Earl D. Sacerdoti. *A Structure for Plans and Behavior.* Elsevier North Holland, New York, NY, 1977.

[Shapiro and Group 1989] S. C. Shapiro and The SNePS Implementation Group. *SNePS-2 User's Manual.* Department of Computer Science, SUNY at Buffalo, 1989.

[Shapiro and Rapaport 1987] S. C. Shapiro and W. J. Rapaport. SNePS considered as a fully intensional propositional semantic network. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier*, pages 263–315. Springer–Verlag, New York, 1987.

[Shapiro 1991] Stuart C. Shapiro. Case studies of SNePS. *SIGART Bulletin*, 2(3):128–134, June 1991.

[Wilkins 1988] David E. Wilkins. *Practical Planning-Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, Palo Alto, CA, 1988.