Towards a unified agent architecture that combines rationality with reactivity

Robert Kowalski and Fariba Sadri
Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, UK
rak,fs@doc.ic.ac.uk

June 12, 1996

Abstract

In this paper we analyse the differences between rational and reactive agent architectures, and propose a uniform agent architecture that aims to capture both as special cases. For this purpose we employ a proof procedure, to control the agent's behaviour, which combines definitions with integrity constraints. The proof procedure is general, and has been shown elsewhere to unify abductive logic programming, constraint logic programming and semantic query optimisation. We also employ a resource-bounded formalisation of the proof procedure which allows the agent's reasoning to be interrupted and resumed, so that observations and actions can be performed.

1 Introduction

The traditional notion of a rational agent in Artificial Intelligence focuses on the agent's thinking process and downplays or ignores its interaction with the environment. This notion has been challenged in recent years by the contrary notion of a reactive agent that focuses on the agent's timely interaction with the environment and downplays or denies the role of thinking.

In this paper we propose an agent architecture that reconciles rationality with reactivity. Rationality is achieved by means of a proof procedure that employs definitions in a knowledge base to reduce goals to subgoals. Reactivity is achieved by employing appropriate integrity constraints in the proof procedure and by formulating the proof procedure in such a way that it can be interrupted to make observations and perform actions.

The rest of the paper is structured as follows. Sections 2 and 3 discuss the notions of rational and reactive agents, respectively, and section 4 analyses their differences. Sections 5-9 develop the uniform architecture and the proof procedure that captures both types of agents as special cases.

2 From knowledge based systems to rational agents

A traditional knowledge based system in Artificial Intelligence contains knowledge in symbolic (often logical) form, which can be both updated and queried. The only action such a system can perform is to return answers to queries. Moreover, it might expend an unlimited amount of resources to compute such answers.

Knowledge based systems differ from conventional database systems primarily in the richer forms of knowledge representation they employ. However, deductive databases, which represent knowledge in the form of both facts and rules, can be understood both as database systems and as knowledge based systems.

Traditional databases have many limitations. An important step towards removing these and towards enhancing their functionality is the extension to *active databases* e.g. [14, 15], which perform actions independently of users' queries. These actions are performed in response to events which occur externally in the environment or internally in the database. The actions serve a number of purposes, including view and integrity maintenance and communication with the environment.

Active databases typically achieve their added functionality through the use of a form of condition-action rules. We argue in this paper that such condition-action rules can be regarded as integrity constraints.

Databases which are both deductive and active can be understood as *agents*. The actions they perform can be understood as *goals*. These action goals arise either from higher level goals as the result of a goal-reduction process or from condition-action rules.

Example 2.1 The externally performed action *raise the alarm* might result from the higher-level internal goal *maintain security* by means of a rule

maintain security **if** *whenever there is an intruder raise the alarm*

which is "triggered" when an intruder is detected (by means of an update/observation/input to the knowledge base). $\triangleleft$

Rational agents have both beliefs (or knowledge) and goals. Goals include actions which are output to the environment as well as higher-level goals which guide the agent's behaviour.

The behaviour of a rational agent can be specified (and implemented) by means of an abstract procedure which defines the agent's observation-thought-action cycle:

to cycle at time T,

i) observe any input at time T,
ii) record any such input,
iii) (optionally) check inputs for satisfaction of integrity constraints,
iv) solve (or re-solve) goals by constructing a plan, employing resources R,
v) select a plan from among the alternatives,
vi) execute any requisite atomic action (in the selected plan), at time $T+R+1$,
vii) cycle at time $T+R+2$.

The amount of resources R a traditional agent might expend on checking inputs and solving goals is unbounded. Traditionally, such an agent constructs a *plan*, consisting of an appropriate structure of atomic actions which completely and provably solves its higher-level goals. Only after having generated a complete plan, does the agent begin to execute it. Any observations made on later iterations of *cycle* which violate the agent's expectations require the agent to replan its earlier solution to its goals.

Critics (e.g. [1, 2]) of the traditional knowledge based approach argue that the lack of a bound on the resources R renders the rational agent architecture unfeasible. They argue that rationality interferes with an agent's ability to react appropriately and in real time to changes that occur unpredictably in its environment.

3 (Generalised) Reactive agents

A reactive agent need have neither an explicit set of beliefs, stored in a knowledge base, nor an explicit representation of any goals. Reactivity can be achieved simply by means of an appropriate collection of stimulus-response, input-output or condition-action rules.

Example 3.1 The rule

if *there is an intruder* **then** *raise the alarm*

is triggered by an input observation of an intruder and immediately generates the action of raising an alarm as output to the environment. Such a rule needs no representation of its purpose, the goal towards which it is directed. In this example, there is no need for a representation of the world or even the internal state of the agent. ◁

Example 3.2 The rules

if *clear ahead* **then** *move forward*
if *obstacle ahead* **then** *turn right*

similarly require no knowledge base, nor goal representation. Employed together as a pair, they achieve the implicit goals of movement and obstacle avoidance.

Such rules, possibly sensitive to the contents of an updatable local state, are typical of the stimulus-response rules of Brook's subsumption architecture [1, 2]. $\lhd$

Example 3.3 The rule

if *Agent requests* $do(self, Act, T2)$ *at time* $T1$
  and *Agent is friendly*
  and *can* $do(self, Act, T2)$
  and $T1 < T2$
then $do(self, Act, T2)$

requires more powerful resources. It is triggered by an input request occurring at (*transaction*) time $T1$, but its remaining conditions are verified by consulting a knowledge base (or simply by looking up the value of variables in an internal state). However, the action itself cannot be performed immediately, but needs to become a *commitment*, which is executed later when the transaction time becomes equal to the *domain* time $T2$ of the action which is requested. Such condition-action rules are typical of the rules which govern the behaviour of agents in Shoham's AgentO architecture [12]. $\lhd$

Condition-action rules can also be formulated in a modal language such as METATEM [6], which exploits Gabbay's separation theorem [5], stating that any temporal logic formula can be rewritten in a logically equivalent form:

if *formula about* **past then** *formula about* **future**.

Example 3.4 The execution of a METATEM rule, such as

if yesterday *Agent requests Act* **then sometime in the future** $do(Act)$

is similarly a process of matching conditions against information in an updatable knowledge base and of creating commitments from the conclusions of the rule. $\lhd$

At an abstract level, all of these examples can be accommodated within a single condition-action rule production system interpreter:

to cycle at time T,
- *i) observe any input at time T,*
- *ii) (optionally) record any such input,*
- *iii) match conditions of condition-action rules against inputs,*
- *iv) (optionally) verify any remaining conditions of the rules using facts in the knowledge base,*
- *v) select an action to execute from the conclusions of competing rules (all of whose conditions are satisfied),*
- *vi) execute any such action at time $T + n + 1$,*

vii) cycle at time $T + n + 2$.

Reactivity is achieved by limiting the amount of resources employed in steps (iii) and (iv) to some fixed small amount of time n. This is easy in cases, like examples 3.1 and 3.2, where there is no knowledge base and conditions are verified simply by matching them with the input. It is more difficult in cases, like example 3.3, where reasoning is needed to verify some of the conditions. One practical solution, employed in systems such as AgentO, is to restrict the form of the sentences in the knowledge base, so that conditions can be verified efficiently, within the restricted available time, n.

4 How to reconcile rationality with reactivity

In our characterisations of them, there are two main differences between rational and reactive agents:

- the form in which "knowledge" and "goals" are represented
- the amount of resources that may be consumed within a single cycle.

In a rational agent, goals are represented explicitly and knowledge is represented as goal reduction rules. Such goal reduction rules typically have the form *conclusion if conditions*, associated both with logic programs and with deduction rules in deductive databases. Backward reasoning is used to match goals to conclusions of rules, reducing them to subgoals which are the conditions of the rules. Actions are atomic goals to which no reduction rules apply.

In the traditional agent architecture, top-level goals are fully reduced to complete plans, consisting of atomic subgoals, before they are executed. This can take an unbounded amount of time.

In reactive agents, goals are achieved implicitly by condition-action rules, and efficiency is obtained by avoiding the overheads associated with goal reduction. By appropriate restrictions on the form of the knowledge base, if there is one, conditions can be verified efficiently, and a single iteration of the agent's cycle can be completed within a fixed, small amount of time, n.

To reconcile the differences between these two kinds of agent, we need therefore

1) to understand better the relationships between goals, goal-reduction rules and condition-action rules, and

2) to organise goal-reduction so that it can be interrupted after a fixed amount of time, n, to execute actions and to perform observations, and so that it can be resumed correctly on the next cycle.

For the first problem we will argue that condition-action rules can be interpreted as integrity constraints, and that there is a simple relationship between integrity constraints and goal-reduction rules, which facilitates employing them together in a single proof procedure.

For the second problem we will argue that the simple definition of the provability predicate *demo*, used in logic programming, can be extended to define an enhanced *demo* predicate which performs goal-reduction in the required manner.

5 Goals, integrity constraints and generalised logic programs

Database systems distinguish between two kinds of sentences: sentences which *define* the data and, accordingly belong to the database, and sentences (integrity constraints) which *constrain* the data and are not actually part of it. Typically, sentences that define the data have a simple syntax – variable-free atomic sentences in the case of relational databases; positive Horn clauses, usually without function symbols, in the case of deductive databases. However, sentences that constrain the data generally have a much richer syntax. In both relational and deductive databases, integrity constraints can be unrestricted sentences of first-order logic. In this respect, integrity constraints are like database queries, which can also be sentences of first-order logic.

The similarity between integrity constraints and queries is more than syntactic. It can be argued that they have the same semantics. However, many different semantics have been proposed for integrity constraints. These include proposals that they be consistent with the database (or the completion of the database), that they be theorems of the database (or its completion), or that they be epistemic (or metalevel) statements that "hold" of the database.

Recently, we have developed a proof procedure [3, 4, 9, 10, 13] for abductive logic programs with integrity constraints, in which we have found it useful to treat integrity constraints and queries identically. This requires that they have the same semantics. To a first approximation, the semantics we employ can be thought of as requiring that queries and integrity constraints be theorems of the completion of the knowledge base (i.e. database or program)[1].

The proof procedure, in standard logic programming fashion, uses definitions to reduce atomic goals to subgoals. These subgoals can themselves be formulae of full first-order logic. Subgoals, therefore, have the same syntax as queries and integrity constraints. This syntax is the key to the relationship between goals, goal-reduction rules and condition-action rules that we are looking for.

Example 5.1 Consider the (generalised) logic program

$maintain\text{-}security \leftarrow \forall T\ [intruder\ at\ time\ T \rightarrow do(self, raise\text{-}alarm, T)].$

Given the top-level goal (or integrity constraint)

maintain-security

[1]More precisely, we require that they be true in all *intended* models of the knowledge base [10].

the program reduces this to the (generalised) subgoal (or integrity constraint)

$\forall T$ *[intruder at time* $T \rightarrow do(self, raise\text{-}alarm, T)$ $\triangleleft$

This example illustrates a general phenomenon: Given a set of goals G and goal-reduction rules Kb, we can, in many cases, replace them by an equivalent set of condition-action rules R. The latter are equivalent to the former in the sense that the rules R implicitly accomplish the goals G in the manner prescribed by Kb. The condition-action rules R can be viewed as a compiled form of G and Kb. The process of generating R from G and Kb, called partial evaluation [11], is a powerful, but standard logic programming technique.

Not every knowledge base and initial set of goals can be reduced to a set of condition-action rules. The goals must be defined non-recursively, in particular.

We do not claim that every set of condition-action rules is the result of partially evaluating an explicit goal-reduction representation. It is quite possible that an agent might learn its condition-action rules directly, as the result of its interactions with the environment, without the mediation of any goal-reduction representation.

But if we accept that condition-action rules are just a special kind of integrity constraint (or generalised goal) and if an agent can reason with *both* goal-reduction rules and integrity constraints, then there is no reason to require that condition-action rules be derived from "higher-level" explicit goal-reduction representations. Condition-action rules can join other (generalised) goals in the traditional rational agent architecture that recognises beliefs and goals as the two main components of an agent's state.

6 The proof procedure

We now outline a proof procedure based on [3, 4, 9, 10, 13], which reasons with two kinds of sentences:

i) *Definitions* in if-and-only-if form:

$$G \leftrightarrow D_1 \vee \ldots \vee D_n \qquad n \geq 0.$$

These are used for goal reduction. If $n = 0$ the disjunction is equivalent $to false$.

ii) *Integrity constraints* in clausal form:

$$A_1 \vee \ldots \vee A_n \leftarrow B_1 \wedge \ldots \wedge B_m \qquad m, n \geq 0.$$

If $m = 0$ the conjunction is equivalent to $true$. If $n = 0$ the disjunction is equivalent to $false$.

The disjuncts D_i of a definition are conjunctions

$$C_1 \wedge \ldots \wedge C_m \qquad m \geq 1$$

where each conjunct is either an atom (possibly $true$) or an implication with syntactic form (ii) identical to that of an integrity constraint. The A_i and B_i, in the conclusion and conditions of implications (ii) are atomic formulae. As in N-Prolog [7], negative literals $\neg A$ are written as implications of the form

$$false \leftarrow A.$$

For simplicity, in this part of the paper, we ignore variables and quantifiers, restricting ourselves to the propositional case. A more complete account of similar proof procedures can be found in [3, 4, 9, 10, 13].

The inference rules of the proof procedure transform a *goal statement* which has the syntax

$$D_1 \vee ... \vee D_n$$

of the body of a definition into another goal statement, which has the same form. All integrity constraints are conjoined to every goal statement. By distributing disjunction over conjunction they become a conjunct of every disjunct of every goal statement. Thus, integrity constraints are treated as goals which are always present.

Goal statements have a procedural interpretation as an *or-and tree*. They can also be written in a logically equivalent *and-or tree* form (i.e. as a conjunction of disjunctions). In particular they can be put into a form where integrity constraints are written only once, conjoined to every goal statement.

Atomic subgoals of the form

$$do(self, Act, T)$$

are "solved" by executing them in step (vi)[2] of the agent cycle when the domain time T becomes the current transaction time.

The proof procedure has four main inference rules:

i) **Goal reduction** uses a definition

$$G \leftrightarrow D_1 \vee ... \vee D_n$$

case 1: to replace a goal statement of the form[3]

$$\begin{array}{l}(G \wedge G') \vee D \quad \text{by} \\ ((D_1 \vee ... \vee D_n) \wedge G') \vee D\end{array}$$

case 2: to replace a goal statement of the form

$$\begin{array}{l}((D' \leftarrow G \wedge G') \wedge G'') \vee D \qquad \text{by} \\ ((D' \leftarrow D_1 \wedge G') \wedge ... \wedge (D' \leftarrow D_n \wedge G') \wedge G'') \vee D.\end{array}$$

[2]Note that the rational and reactive agents cycles differ primarily in steps (iii), (iv), (v).

[3]Here and elsewhere, we assume the commutativity and associativity of conjunction and disjunction. In particular, when we write a goal statement in the form $(G \wedge G') \vee D$, we intend that $G \wedge G'$ may be any disjunct in the goal statement and that G may be any conjunct in the disjunct.

The second form of goal-reduction implicitly uses the equivalence

$$A \leftarrow (B \vee C) \quad \leftrightarrow \quad (A \leftarrow B) \wedge (A \leftarrow C).$$

ii) **Splitting** explicitly distributes $\vee$ over $\wedge$, replacing a formula of the form

$$\begin{array}{ll} (D \vee D') \wedge G & \text{by} \\ (D \wedge G) \vee (D' \wedge G). & \end{array}$$

iii) **Propagation** replaces a goal statement of the form

$$\begin{array}{ll} (G \wedge (D' \leftarrow G \wedge G') \wedge G'') \vee D & \text{by} \\ (G \wedge (D' \leftarrow G') \wedge G'') \vee D. & \end{array}$$

When the implication $D' \leftarrow G \wedge G'$ is an integrity constraint or an implication derived from an integrity constraint, propagation contributes to integrity verification[4].

iv) **Logical equivalence** replaces a formula by another formula which is both logically equivalent and more suitable for manipulation by the other inference rules. These include the following equivalences used as rewrite rules:

$$\begin{array}{lcl} G \wedge true & \leftrightarrow & G \\ G \wedge false & \leftrightarrow & false \\ D \vee true & \leftrightarrow & true \\ D \vee false & \leftrightarrow & D. \end{array}$$

The proof procedure was developed initially in an attempt to unify abductive logic programming, constraint logic programming and semantic query optimisation. For this purpose, definitions are used only for completely defined predicates. Incompletely defined predicates, such as abducibles, have to be treated differently. Technically the simplest, and most elegant way to deal with them is to approximate their definitions by means of integrity constraints, which are included conceptually, therefore, in every disjunct of every goal statement. These integrity constraints, in the simplest case, consist only of atoms or the negation of atoms.

Incompletely defined predicates include all input predicates, which describe observations, and all output predicates, which record the result of actions attempted by the agent. By recording observations in goal statements, the propagation inference rule implements the triggering of condition-action rules by the input as a special case. The verification of the remaining conditions of the condition-action rules is performed by other propagation steps or by case 2 of the goal-reduction rule.

We will see in section 8 that by recording the results of attempted actions in goal statements, propagation also eliminates any disjuncts of a goal statement which are incompatible with the results of the action.

[4] In the propositional case, the implication $D' \leftarrow G \wedge G'$ is not needed in the derived goal statement, because it is "subsumed" by the derived implication $D' \leftarrow G'$. In the more general case, it needs to be retained when it is not subsumed.

7 Resource-bounded reasoning

The proof procedure just described combines the goal-reduction rules of a rational agent with the condition-action rules of a reactive agent. It combines them, moreover, as a special case of a general proof procedure which combines goal-directed, backward reasoning using definitions, with data-driven, forward reasoning using integrity constraints. This proof procedure provides us with a solution to the first of the two subproblems which we identified in section 4, of the problem of reconciling rationality with reactivity.

We still need a solution to the second subproblem, which is to control the reasoning process so that it functions correctly with bounded resources. For this purpose we employ a simple enhancement of the standard *demo* predicate. We intend the enhanced *demo* predicate to be invoked by the agent in place of steps (iii) and (iv) of the agent cycle.

The standard *demo* predicate for a propositional Horn clause language has a simple definition by means of a metalogic program:

$$
\begin{aligned}
&demo(KB, G) \leftarrow demo(KB, G \leftarrow G') \wedge demo(KB, G')\\
&demo(KB, G \wedge G') \leftarrow demo(KB, G) \wedge demo(KB, G')\\
&demo(KB, true)
\end{aligned}
$$

Here the same symbols, $\leftarrow$ and $\wedge$, are used for object level and metalevel implication and conjunction, respectively. KB names a "knowledge base" of propositional Horn clauses, G' names a conjunction of atoms and G names an atom. The first rule reduces G to the subgoals G', the second reduces the conjunction of goals $G \wedge G'$ to separate subgoals G, G', and the third asserts that an empty set of goals is trivially solvable.

The simple *demo* predicate above assumes that an unbounded amount of resources is available for goal-reduction. Moreover, it returns no output. It simply holds (at the metalevel) whenever the knowledge base solves the goal (at the object level). Furthermore, it is restricted to goals which are conjunctions of atoms. We need to remove all of these restrictions to obtain an enhanced *demo* predicate that can be used to control the reasoning component of a resource-bounded agent's cycle.

The following metalogic program defines the top-level of the enhanced *demo* predicate we require. The new predicate $demo(KB, S, S', R)$ holds when the goal statement S can be reduced to the goal statement S' in R inference steps of the proof procedure outlined in section 6.

$$
\begin{aligned}
&demo(KB, S, S', R) \leftarrow step(KB, S, S'') \wedge demo(KB, S'', S', R-1)\\
&demo(KB, S, S, 0)
\end{aligned}
$$

Here $step(KB, S, S'')$ holds when S can be reduced to S'' in one inference step. For example, the following two rules deal with case 1 of goal reduction and with splitting, respectively:

$$step(KB, (G \wedge G') \vee D, (D' \wedge G') \vee D) \quad \leftarrow \quad (G \leftrightarrow D') \in KB$$
$$step(KB, ((D \vee D') \wedge G) \vee D'', (D \wedge G) \vee (D' \wedge G) \vee D'')$$

8 The cycle of an agent that combines rationality with reactivity

We can now reformulate the agent cycle in a uniform manner that includes both the rational agent and the reactive agent as special cases:

to cycle at time T,

i) observe any input at time T,

ii) record any such inputs,

iii) resume the proof procedure by first propagating the inputs,

iv) continue applying the proof procedure for a total of n inference steps,

v) select, from among the alternatives, an atomic action whose domain time is compatible with the transaction time $T + n + 1$,

vi) execute any such action and record the results (success or failure),

vii) cycle at time $T + n + 2$.

As mentioned in section 6, recording inputs and results of attempted actions is performed by adding to the current goal statement, atoms (in the case of observations and successful actions), or negations of atoms in the form

$$false \leftarrow do(self, act, t + n + 1)$$

(in the case of an action that fails at time $t + n + 1$).

The proof procedure is resumed at time t in step (iii) by calling the enhanced *demo* predicate with arguments

$$demo(kb, s, s', n)$$

where kb names the knowledge base, s names the goal statement at time t, and s' names the resulting goal statement at time $t + n$. For simplicity, we ignore the time taken to observe and record the input.

The goal statement s' may contain one or more alternative actions that can be performed at time $t + n + 1$. This is the case whenever the goal statement has the form

$$(do(self, act, T) \wedge G) \vee D$$

where $T = t + n + 1$ is compatible with any constraints on T in G. In step (v) the cycle commits to one of these actions (or more, if simultaneous execution of atomic actions is possible). Recording the result of any such attempted action by adding it to the goal statement s' makes the result available for propagation on the next cycle. Such propagation will have the effect, if the action failed, of adding the implication $false \leftarrow T = t+n+1$, i.e. $T \neq t+n+1$ to the selected disjunct

$$do(self, act, T) \wedge G.$$

If the action succeeded it will have the effect of adding the implication $false \leftarrow T' = t+n+1$, i.e. $T' \neq t+n+1$ to any other disjunct which contains an action

$$do(self, act', T')$$

which is incompatible with *act* because of an integrity constraint such as

$$\forall T[false \leftarrow do(self, act, T) \wedge do(self, act', T)].$$

9 Partial plans and scheduling

The feasibility of our proposed agent architecture depends in large measure on the form of the agent's goal-reduction rules and integrity constraints. The goal-reduction rules, in particular, need to construct partial plans in an incremental manner, so that execution can commence before all the atomic actions have been generated.

For this purpose, the goal-reduction rules have to provide greater detail about the beginning of a plan than they do about the end. This is illustrated by the following example of a rule which reduces the goal of moving from one place, A, to another place, B, to the subgoals of first taking a single step to a next location, C, and then moving from C to B:

Example 9.1

$$\begin{array}{l} do(Agent, move(A, B), [T1, T2]) \leftarrow \\ \quad next(A, C) \wedge clear(C, T) \wedge T1 \leq T \leq T2 \wedge \\ \quad do(Agent, step(A, C), T) \wedge \\ \quad do(Agent, move(C, B), [T, T2]) \end{array}$$

Here $T1$ and $T2$ are the earliest start time and the latest finish time for the action, $move(A, B)$, respectively.

Execution of actions generated by the rule above can commence at some time $T \geq T1$, after the first two conditions have been eliminated by goal reduction. The time T for the execution of $step(A, C)$ must be early enough for the remainder of the partial plan, $move(C, B)$, to be completed before time $T2$.

We assume that such scheduling of actions is performed by a separate control layer which decides how to implement the non-determinism of the proof procedure and the selection step (v) of the agent cycle. The control layer will need to decide, therefore, not only when to attempt the atomic action $step(A, C)$, but also which location, C, from among the alternative next locations, to select. The control layer can use the same evaluation function both to guide the search strategy of the proof procedure and to select among alternative disjuncts of the current goal statement when commitments to actions need to be made. $\triangleleft$

10 Conclusion

In this paper we have outlined an attempt to reconcile the traditional notion of a rational agent with the contrary notion of a reactive agent. This work extends an earlier paper [8] with the same objectives. It also builds on recent developments of a proof procedure [3, 4, 9, 10, 13] which combines reasoning with both definitions and integrity constraints. Definitions are used to reduce goals to subgoals, in the manner of a rational agent. Integrity constraints are used to generate actions in response to updates from the environment, in the manner of a reactive agent.

A further key feature of the agent architecture is the resource-bounded formalisation of the proof procedure, which allows the agent's reasoning to be interrupted and resumed between one cycle and the next.

Future work includes further development of the temporal component of the object language, as well as applications of the single agent architecture to multi-agent systems.

Acknowledgements
The authors are grateful to Tze Ho Fung, Jacinto Davila, Francesca Toni and Gerhard Wetzel for discussions and their contributions to this work.

References

[1] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[2] Rodney A. Brooks. Intelligence without reason. In J. Mylopolous and R. Reiter, editors, *Proceedings of IJCAI 91*, pages 569–595. Morgan Kaufmann Publishers, 1991.

[3] Tze Ho Fung. A modified abductive framework. In N. Fuchs and G. Gottlob, editors, *Proceedings of Logic Programming Workshop*, 1994.

[4] Tze Ho Fung. *Abduction by deduction.* PhD thesis, Imperial College, University of London, 1996.

[5] Dov Gabbay. The declarative past and imperative future. In Howard Barringer, editor, *Proccedings of the Colloquium on Temporal Logic and Specifications, LNCS, Vol. 398*, pages 409–448. Springer-Verlag, 1989.

[6] Dov Gabbay, Howard Barringer, Michael Fisher, Graham Gough, and Richard P. Owens. MetateM: A framework for programming in temporal logic. In *REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness. Mook, Netherlands. LNCS Vol. 430*, pages 94–129. Springer-Verlag, 1989.

[7] Dov Gabbay and Uwe Reyle. N-prolog: An extension of prolog with hypothetical implications I. *Journal of Logic Programming*, 1:319–355, 1984.

[8] Robert A. Kowalski. Using meta-logic to reconcile reactive with rational agents. In K. Apt and F. Turini, editors, *Meta-Logic and Logic Programming*, pages 227–242. MIT Press, 1995.

[9] Robert A. Kowalski, Francesca Toni, and Gerhard Wetzel. Towards a declarative and efficient glass-box clp language. In N. Fuchs and G. Gottlob, editors, *Proceedings of Logic Programming Workshop (WLP'94)*, 1994.

[10] Robert A. Kowalski, Gerhard Wetzel, and Francesca Toni. A unifying framework for alp, clp and sqo. Technical report, Department of Computing, Imperial College, London, April 1996.

[11] John W. Lloyd and John C. Shepherdson. Partial evaluation in logic programming. *Journal of Logic Programming*, 11:217–242, 1991.

[12] Yoav Shoham. Agent-oriented programming. *AI Journal*, 60(1), pages 51–92, 1993.

[13] Gerhard Wetzel, Robert A. Kowalski, and Francesca Toni. A theorem-proving approach to clp. In Geske U., Krall A., (eds), *Workshop Logische Programmierung*, volume 270 of *GMD-Studien*, pages 63–72. Bonn, Germany, 1995.

[14] Jennifer Widom. Deductive and active databases: two paradigms or ends of a spectrum. In N.W. Paton and H. Williams, editors, *Rules in Database Systems: Proceedings of the 1st International Workshop*, pages 306–315. Springer-Verlag, 1994.

[15] Carlo Zaniolo. A unified semantics for active and deductive databases. In N.W. Paton and H. Williams, editors, *Rules in Database Systems: Proceedings of the 1st International Workshop*, pages 271–287. Springer-Verlag, 1994.