3 DEONTIC ACTION PROGRAMS

Thomas Eiter[1] and V.S. Subrahmanian[2]

[1] Institut und Ludwig Wittgenstein Labor für Informationssysteme
Technische Universität Wien, Treitlstraße 3, 1040 Wien
Austria
eiter@kr.tuwien.ac.at

[2] Institute for Advanced Computer Studies and Department of Computer Science
University of Maryland, College Park, Maryland 20742
U.S.A.
vs@cs.umd.edu

Abstract: We introduce the concept of an *Action Policy* in the context of databases. Intuitively, such a policy tells us which from a collection of actions *must* be executed (i.e. are obligatory), which are *forbidden* from being executed, and which are *permitted* (but not obligatory) given a state of the database. We approach this by a rule-based action language which uses concepts of deontic logic. We define a semantics for programs in this language based on the concept of a status set, which intuitively describes a consistent global view of the actions. Moreover, we consider refinements of status sets, and we investigate the complexity and computation of all these semantics.

3.1 INTRODUCTION

During the last several years, active databases have been studied extensively by researchers around the world. Following the lead provided by many successful implementations (Ceri and Widom, 1991; Ghandeharizadeh et al., 1993; Hanson, 1992; Dayal et al., 1995), formal semantics have recently been proposed for event-condition-action (ECA) rules, cf. (Fraternali and Tanca, 1995; Gottlob et al., 1996; Picouet and Vianu, 1995; Zaniolo, 1993).

In this paper, we introduce the concept of an *Action Policy*. Intuitively, given a feasible space $\mathcal{A}$ of actions, and a database state DB, an action policy $\wp$ tells us which of the actions in $\mathcal{A}$ *must* be executed (i.e. are obligatory), which are *forbidden* from being executed, and which are *permitted* (but not obligatory). Given a feasible space $\mathcal{A}$ of actions, a database state DB, an action policy further specifies a way of *determining* which actions to actually perform, based on the implied obligations and permissions. We will build our notion of an action policy on top of deontic logic (Åquist, 1984; Meyer and Wieringa, 1993). It will turn out that the well known ECA-rules are a special case of our notion of an action policy, thus making our semantics applicable not only to existing ECA-rules, but also to the extended set of action policies proposed in this paper.

Before proceeding any further, let us consider a simple example to see why action policies are needed. Consider a tax agency (such as the Internal Revenue Service in the USA) that monitors tax returns. The database monitors discrepancies between the amounts reported by individual taxpayers, and amounts reported by *all* employers who have made payments to that person. An action policy might make the following statement:

1. If the amount reported by the individual is less than 70% of the employer-reported income, then triggering an audit program is *obligatory*.
2. if the amount reported by the individual is less than 80% of the employer-reported income, then triggering an audit program is *permitted*.
3. If the amount reported by the individual is over 80% then it is *forbidden* to run the audit program.
4. If a person makes over 200K (according to employer filed returns) and it is permitted to audit him/her, then *do* the audit program run.
5. If an audit run is *done* on a person, then an action that notifies him/her to this effect must also be *done*.

Rules such as these (with different percentages) are often used to determine whom to audit, based on objective criteria that are legally defendable.[1]

If one examines the first three rules, we note that the individuals who reported as income, only 70-80% of what employers said they earned, fall in a "gray" area – audits are permitted on them, but it is not clear what mechanism is used to select exactly whom to audit from this group of people. The fourth rule presents one such way of determining which of these people should be audited. We note (for deontic logic aficionados) that the last two rules are not easily expressed in standard deontic logic.

In this paper, we will formally introduce the syntax of action policies (Section 3.2) by building upon a restricted version classical deontic logic, i.e. we will start with a fragment of deontic logic and extend it with non-deontic operators. We will then provide three alternative semantics for sets of action policies (Section 3.3). We will present results on the computational complexity of these different semantics (Section 3.4), as well as some results on the inter-relationships between them. We have developed algorithms to compute these semantics as well (Section 3.5). We will conclude with a brief discussion of related work in the area of active databases (Section 3.6).

3.2 SYNTAX

Throughout this section, we will assume the existence of a relational database $\mathcal{DB}$ of relations $p_1, \ldots, p_n$. We also assume that $\mathcal{DB}$ has an accompanying query language

QL which has a subclass of queries called *boolean queries* (of QL), together with a set $\mathcal{IC}$ of integrity constraints. Without loss of generality, we will assume that the initial state DB_0 of the database is consistent with the integrity constraints (and of course, we must ensure that our semantics preserves this consistency when active rules fire). Figure 3.1 shows an example of instances of some of the relations associated with the tax example.

Figure 3.1 Relations from the tax database

`return`

SSN	*Name*	*RawInc*	*AdjInc*
91-762-8976	Smith	75000	58000
34-289-0127	Jones	16000	14000
...	...	...	...
...	...	...	...

`employer_report`

EmpID	*SSN*	*Inc*
company1	91-762-8976	57000
company2	91-762-8976	17650
company1	34-289-0127	15800
...	...	...

`audit`

SSN	*Result*	*Date*
98-283-2829	prosecute	Nov. 5, 1997
83-211-9232	okay	Nov. 3, 1997
...	...	...

`notified`

SSN	*Date*
98-283-2829	Nov. 9, 1997
83-211-9232	Nov. 9, 1997
...	...

Definition 3.2.1 (action) *An* action α *has the following components:*

- *A name, α, and a list $(X_1, \ldots, X_n)$ of parameters X_i (treated as variables);*
- *A schema, usually written as $(\tau_1, \ldots, \tau_n)$, of types. Intuitively, this says that the variable X_i must be of type τ_i for all $1 \leq i \leq n$.*
- *a* boolean query $q(\vec{t})$ *on DB called the* precondition *of the action, denoted by $Pre(\alpha)$, where $\vec{t}$ is a vector of terms;*
- *a set $Add(\alpha) = \{A_1(\vec{t_1}), \ldots, A_n(\vec{t_n})\}$ of atoms to be inserted into DB; and,*
- *a set $Del(\alpha) = \{D_1(\vec{t'_1}), \ldots, D(\vec{t'_m})\}$ of atoms to be deleted from DB;*

Here, $A_i(\vec{t_i})$ (resp., $D_i(\vec{t'_i})$) are possibly non-ground atoms, each of which is associated with a boolean query $q_{A_i}(\vec{t_i})$ (resp., $q_{D_i}(\vec{t'_i})$); informally, they represent the ground atoms $A_i(\vec{t_i}\theta)$ (resp., $D_i(\vec{t'_i}\theta)$) such that $q_{A_i}(\vec{t_i}\theta)$ (resp., $q_{D_i}(\vec{t'_i}\theta)$) is true on a database state DB.

The above definition states what an action is, but allows the possibility that an action allows an atom to be simultaneously added and deleted. Classical AI systems like STRIPS (Nilsson, 1980) also allow this to happen in the syntax (e.g. in a STRIPS rule, an atom could occur in both the Add and $Delete$ lists) and handle potential problems of this sort by first doing deletions and then insertions. We mimic this as well. Table 3.1 shows some of the actions associated with the tax example:

Table 3.1 Some actions in the tax example

Action	*run_audit*	*notify*
Name	run_audit(Name,SSN)	notify(SSN)
Precondition	return(SSN,Name,Raw,Adj) & S=sum(Inc,employer(E,SSN,Inc)) & Adj $< 0.7 \times S$	audited(SSN,Res,Date) –
Add	audit(SSN,Result,Date): Result = eval_audit(SSN) & Date = today()	notified(SSN,Date): Date = today()
Del	–	–

(Here, predefined function queries eval_audit(SSN) and today() are available).

As usual, a (ground) action α is *executable* in a database state DB iff $pre(\alpha)$ evaluates to true in DB. We define the *result* of executing such an α in state DB as the state $DB' = (DB \setminus Del(\alpha, DB)) \cup Add(\alpha, DB)$, where $Add(\alpha, DB) = \{A(\vec{t}\gamma) \mid A(\vec{t}) \in Add(\alpha), DB \models q_A(\vec{t}\gamma)\}$ and $Del(\alpha, DB) = \{D(\vec{t}\gamma) \mid D(\vec{t}) \in Del(\alpha), DB \models q_D(\vec{t}\gamma)\}$ are the sets of ground instances of atoms in $Add(\alpha)$ (resp., $Del(\alpha)$) over DB such that the associated query q_A (resp., q_D) is true on DB.

We next describe a natural definition for the execution of a collection of ground actions on DB.

Definition 3.2.2 (concurrent execution) *Let $AS = \{\alpha_1, \ldots, \alpha_n\}$ be a set of ground actions, and DB a state such that α_i is executable in DB, $i = 1, \ldots, n$. Then AS is* concurrently executable*, iff (i) every sequential execution $AS[\pi] = \alpha_{\pi(1)}, \ldots, \alpha_{\pi(n)}$ on DB is feasible, i.e., $\alpha_{\pi(i)}$ is executable on DB^{π}_{i-1}, where $DB^{\pi}_0 = DB$ and DB^{π}_i results from executing $\alpha_{\pi(i)}$ on DB^{π}_{i-1}, $i = 1, \ldots, n$, and (ii) $DB^{\pi}_n = DB^{\pi'}_n$ for every permutations π and π' on $\{1, \ldots, n\}$.*

Unfortunately, this notion is intractable, even in a very simple setting.

Theorem 1 *Let AS be a set of ground actions and DB a database state, and suppose that QL contains conjunctive queries on DB. Then, deciding whether AS is concurrently executable is* co-NP*-hard.* ∎

A dual result (NP-hardness) holds for choosing any feasible $AS[\pi]$ for execution, as deciding the existence of such an $AS[\pi]$ is NP-hard. In (Eiter et al., 1998) we use an alternative polynomial-time concurrent execution model: The execution of a nonground action $\alpha(\vec{X})$ in DB is defined as the simultaneous execution of all ground instances α' of α which are executable on DB.

Definition 3.2.3 (action base & status atom) *An* action base*, $\mathcal{AB}$, is any finite collection of actions. Suppose $\alpha(\vec{t})$ is an action atom, i.e., an action name with a vector $\vec{t}$ of terms (variables or domain elements) matching the type schema of α. Then, the formulas $\mathbf{P}(\alpha(\vec{t}))$, $\mathbf{F}(\alpha(\vec{t}))$, $\mathbf{O}(\alpha(\vec{t}))$, and $\mathbf{Do}(\alpha(\vec{t}))$ are* action status atoms.

For convenience, we sometimes omit parentheses and write $\mathbf{P}\alpha$, $\mathbf{F}\alpha$ instead of $\mathbf{P}(\alpha)$, $\mathbf{P}(\alpha)$ etc. An action status atom has the following intuitive meaning:

- $\mathbf{P}\alpha$ means that action α is permitted;
- $\mathbf{F}\alpha$ means that action α is forbidden;
- $\mathbf{O}\alpha$ means that action α is obligatory; and,
- $\mathbf{Do}\alpha$ means that action α is in fact executed.

Definition 3.2.4 (action rules & programs) *A* rule *is a clause r of the form*

$$A \leftarrow L_1, \ldots, L_n \tag{3.1}$$

where A is an action status atom, and every L_i is either action status atom $Op(\vec{t})$, possibly preceded by a negation sign ($\neg$), or a – possibly negated – Boolean query $q(\vec{t_i})$ from QL, where $\vec{t_i}$ is a vector of terms. An action program *$\mathcal{P}$ is a finite collection of rules. An action program $\mathcal{P}$ is* positive*, if all its rules are positive.*

The rules specified in the tax example may now be easily encoded through the following rules. We use the predicate `tot_rep_inc(SSN,Inc)` as a view on top of the base relation, `employer_report`. `tot_rep_inc(SSN,Inc)` specifies the total income that a particular individual has received based on all tuples pertaining to him in the `employer_report` relation.

$$\begin{aligned}
\mathbf{O}run_audit(Name, SSN) \leftarrow\ & return(SSN, Name, RInc, AInc),\\
& tot_rep_inc(SSN, Inc), AInc \leq 0.7 \times Inc.\\
\mathbf{P}run_audit(Name, SSN) \leftarrow\ & return(SSN, Name, RInc, AInc),\\
& tot_rep_inc(SSN, Inc), AInc \leq 0.8 \times Inc.\\
\mathbf{F}run_audit(Name, SSN) \leftarrow\ & return(SSN, Name, RInc, AInc),\\
& tot_rep_inc(SSN, Inc), AInc > 0.8 \times Inc.\\
\mathbf{Do}run_audit(Name, SSN) \leftarrow\ & \mathbf{P}run_audit(Name, SSN),\\
& tot_rep_inc(SSN, Inc), Inc > 200K.\\
\mathbf{Do}notify(SSN) \leftarrow\ & \mathbf{Do}run_audit(Name, SSN).
\end{aligned}$$

3.3 SEMANTICS

In what follows, we assume the existence of an action base $\mathcal{AB}$ and a set of integrity constraints $\mathcal{IC}$, all associated with an underlying database $\mathcal{DB}$.

For defining the semantics of action programs, we first introduce the concept of a *feasible status set*. Feasible status sets provide the basic structure in terms of which we define our semantics.

Definition 3.3.1 *A* status set *is any set S of ground action status atoms over DB. For any action status operator Op, we denote $Op(S) = \{\alpha \mid Op(\alpha) \in S\}$.*

Informally, if some atom $Op(\alpha)$ occurs in S, then this means that the status Op is true for α.

Definition 3.3.2 (deontic consistency) *A status set S is called* deontically consistent, *if it satisfies the following rules for any ground action α: (1) $\mathbf{P}\alpha \in S \Rightarrow \mathbf{F}\alpha \notin S$, (2) $\mathbf{P}\alpha \in S \Rightarrow \alpha$ is executable in state DB, and (3) $\mathbf{F}\alpha \in S \Rightarrow \mathbf{P}\alpha \notin S$.*

Definition 3.3.3 (deontic and action closure) *Let S be a status set. The deontic (resp., action) closure of S, denoted $DCl(S)$ (resp., $ACl(S)$), is the closure of S under the rule $\mathbf{O}\alpha \Rightarrow \mathbf{P}\alpha$ (resp., the rules $\mathbf{O}\alpha \Rightarrow \mathbf{Do}\alpha$, $\mathbf{Do}\alpha \Rightarrow \mathbf{P}\alpha$), where α is any ground action.*

Definition 3.3.4 (feasible status set) *A status set S is a* feasible status set *for $\mathcal{P}$ on database state DB, if the following conditions $S1$–$S4$ hold:*

$S1$ *Let $r : A \leftarrow L_1, \ldots, L_n$ be any ground instance of some rule in $\mathcal{P}$. Suppose the following conditions 1.–4. hold:*

1. *If L_i is a positive status atom $Op(\alpha)$, then $Op(\alpha) \in S$;*
2. *if L_i, A is a positive status atom $\mathbf{P}(\alpha)$, $\mathbf{O}(\alpha)$, or $\mathbf{Do}(\alpha)$, then α is executable in DB;*
3. *If L_i is a negated status atom $\neg Op(\alpha)$, then $Op(\alpha) \notin S$;*
4. *For every non-status literal L_i, $DB \models L_i$, i.e., the (possibly negated) query represented by L_i is true on DB.*

Then, the atom A is in S. For later reference, denote by $App_{\mathcal{P},DB}(S)$ the set of all such A, for all ground rules r.

$S2$ *S is deontically consistent;*

$S3$ *S is deontically and action closed, i.e., $S = DCl(S)$ and $S = ACl(S)$;*

$S4$ *$DB' \models \mathcal{IC}$, where $DB' = apply(\mathbf{Do}(S), DB)$ is the state which results by executing the actions in $\mathbf{Do}(S)$ on DB (cf. above).*

In general, there are action programs that have zero, one or several feasible status sets. This is shown by the following example.

Example 3.3.1 Consider the relation `emp` below, together with the sets of rules P_1, P_2, P_3:

P_1:
$$\mathbf{O}send\_notice(P) \leftarrow emp(P, Salary), Salary \geq 80K.$$
$$\mathbf{P}send\_notice(P) \leftarrow emp(P, Salary), 70K \leq Salary < 80K.$$
$$\mathbf{F}send\_notice(P) \leftarrow emp(P, Salary), Salary < 70K.$$

P_2:
$$\mathbf{O}send\_notice(P) \leftarrow emp(P, Salary), Salary \geq 80K.$$
$$\mathbf{F}send\_notice(P) \leftarrow emp(P, Salary), Salary < 80K.$$

P_3:
$$\mathbf{Do}send\_notice(P) \leftarrow \neg\mathbf{P}send\_notice(P), emp(P, Salary).$$
$$\mathbf{F}(send\_notice(Dina)) \leftarrow$$

`emp`

Name	*Salary*
Harry	76K
Mary	90K
Ted	34K
Ken	44K
Dina	65K

It is easy to see that P_1 has (among others) the following two feasible status sets:

$$\begin{aligned} FSS_1 = \{\ & \mathbf{O}send\_notice(Mary), \mathbf{Do}send\_notice(Mary), \\ & \mathbf{P}send\_notice(Mary), \mathbf{F}send\_notice(Ken), \\ & \mathbf{F}send\_notice(Ted), \mathbf{F}send\_notice(Dina), \mathbf{P}send\_notice(Harry)\ \} \end{aligned}$$

$$\begin{aligned} FSS_2 = \{\ & \mathbf{O}send\_notice(Mary), \mathbf{Do}send\_notice(Mary), \\ & \mathbf{P}send\_notice(Mary), \mathbf{F}send\_notice(Ken), \\ & \mathbf{F}send\_notice(Ted), \mathbf{F}send\_notice(Dina), \\ & \mathbf{P}send\_notice(Harry), \mathbf{Do}send\_notice(Harry)\}. \end{aligned}$$

On the other hand, P_2 has the following feasible status set:

$$\begin{aligned} FSS_3 = \{\ & \mathbf{O}send\_notice(Mary), \mathbf{Do}send\_notice(Mary), \\ & \mathbf{P}send\_notice(Mary), \mathbf{F}send\_notice(Ken), \\ & \mathbf{F}send\_notice(Ted), \mathbf{F}send\_notice(Dina), \mathbf{F}send\_notice(Harry)\}. \end{aligned}$$

P_3 can be easily seen to have no feasible status set. ∎

3.3.1 Rational status sets

Intuitively, a feasible status set describes a set of status atoms that are compatible with the state of the database, the obligations and restrictions imposed by its associated action program, and the deontic consistency requirements. Nevertheless, we note from the previous example that feasible status sets may include **Do**ing actions that are not strictly necessary (FSS_2). The notion of a rational status set is based on the principle that each action which is executed should be sufficiently "grounded" or "justified" by the action program. That is, there should be evidence from the rules of the action program that a certain action must be executed. For example, it seems unacceptable that an action α is executed, if α does not occur in any rule of the action program. This way, we also have to make sure that execution of an action must not be driven by the need to preserve the consistency of the database state. Rather, the integrity constraints should serve to prevent executions which appear to be rational if no integrity constraints were present. This motivates the following formal notion of groundedness.

Definition 3.3.5 (rational status set) *Let S be a feasible status set S for $\mathcal{P}$ on DB. S is grounded, if there exists no status set $S' \neq S$ such that $S' \subseteq S$ and S' satisfies* *conditions $S1$–$S3$ of a feasible status set for $\mathcal{P}$ on DB. S is a* rational status set *of $\mathcal{P}$ on DB, if (i) S is a feasible status set of P on DB, and (ii) S is grounded.*

Returning to the previous example, we note that the feasible status set FSS_2 is not rational.

Proposition 3.3.1 *Let $\mathcal{P}$ be an action program.*

1. *If $\mathcal{P}$ is positive, then $\mathcal{P}$ has for any DB a unique rational status set (if one exists on DB).*
2. *If $\mathcal{IC} = \emptyset$, then $\mathcal{P}$ (positive or not) has a rational status set on DB iff $\mathcal{P}$ has a feasible status set on DB.*

In the previous example, FSS_1 and FSS_3 are the unique rational status sets of $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. Notice that an apparently "contradictory" program may have feasible and rational status sets; e.g., the two rules

$$\begin{array}{rcl} \mathbf{O}send\_notice(P) & \leftarrow & emp(P, Salary), Salary \geq 80K. \\ \mathbf{F}send\_notice(P) & \leftarrow & emp(P, Salary), Salary < 90K. \end{array}$$

are contradictory; however, w.r.t. the previous relation `emp`, they have a feasible and a rational status set (even if we adopt the functional dependency $Name \rightarrow Salary$ on `emp` in $\mathcal{IC}$).

3.3.2 Reasonable status sets

A more serious attack against rational status sets, is that for action programs with negation, the semantics of rational status sets allows logical contraposition of the program rules. For example, consider the following program:

$$\mathbf{Do}(\alpha) \quad \leftarrow \quad \neg\mathbf{Do}(\beta)$$

This program has two rational status sets: $S_1 = \{\mathbf{Do}(\alpha), \mathbf{P}\alpha\}$, and $S_2 = \{\mathbf{Do}(\beta), \mathbf{P}\beta\}$. The second rational status set is obtained by applying the contrapositive of the rule:

$$\mathbf{Do}(\beta) \quad \leftarrow \quad \neg\mathbf{Do}(\alpha)$$

However, the second rational set seems less intuitive than the first as there is no explicit rule in the above program that justifies the derivation of this $\mathbf{Do}(\beta)$.

This observation leads us to the following remarks. First, in logic programming, deductive databases, and knowledge representation, the meaning of negation in rules has been extensively discussed and there is broad consensus that contraposition should not be applied as a proof principle; rather, derivation should be constructive from rules. These observations led to the well known stable model semantics for logic programs (Gelfond and Lifschitz, 1988).

Second, the presence of derivation by contraposition may have a detrimental effect on the complexity of programs, since it inherently simulates disjunction. Therefore, it is advisable to have a mechanism which cuts down rational status sets in an acceptable way, so that negation can be largely used without introducing high computational cost.

For these reasons, we introduce the concept of a *reasonable status set*.

Definition 3.3.6 (reasonable status set) *Let $\mathcal{P}$ be an action program, let DB be a database state, and let S be status set on DB. If $\mathcal{P}$ is a positive action program, then S is a* reasonable status set *for $\mathcal{P}$ on DB, iff S is a rational status set for $\mathcal{P}$ on DB.*

For an arbitrary program $\mathcal{P}$, let the reduct of $\mathcal{P}$ w.r.t. S and DB, denoted by $red^S(\mathcal{P}, DB)$, be the program obtained from the ground instance of $\mathcal{P}$ over DB as follows.

1. *Remove every rule $A \leftarrow \ldots, \neg Op(\alpha), \ldots$ where $Op(\alpha) \in S$, for any status atom $Op(\alpha)$;*
2. *Remove all negated status action atoms $\neg Op(\alpha)$ from the remaining rules.*

Then, S is a reasonable status set *for $\mathcal{P}$ on DB, iff it is a reasonable status set of $red^S(\mathcal{P}, DB)$ on DB.*

Example 3.3.2 For the program $\mathcal{P}$: $\mathbf{Do}(\beta) \leftarrow \neg\mathbf{Do}(\alpha)$, the reduct of $\mathcal{P}$ w.r.t. $S = \{\mathbf{Do}(\beta), \mathbf{P}(\beta)\}$ is P: $\mathbf{Do}(\beta) \leftarrow$. Clearly, S is a reasonable status set of P, and hence S is a reasonable status set of $\mathcal{P}$. ∎

The following result tells us that reasonable status sets are always rational (but the converse may not be true). Thus, reasonable status sets prune the rational status sets.

Proposition 3.3.2 *Let $\mathcal{P}$ be an action program and DB a database state. Then, every reasonable status set of $\mathcal{P}$ on DB is a rational status set of $\mathcal{P}$ on DB.* ∎

3.4 COMPLEXITY RESULTS

In this section, we address the computational cost of action programs. We assume that the reader is familiar with the basic concepts of complexity theory, cf. (Johnson, 1990; Papadimitriou, 1994).

We consider here *data complexity* of action programs in the sense of (Vardi, 1982), i.e., evaluation of a *fixed* action program $\mathcal{P}$ over varying instances DB of a relational database $\mathcal{DB}$; we assume that $\mathcal{AB}$ and $\mathcal{IC}$ are fixed, and that the associated Boolean query language QL has polynomial data complexity and $\mathcal{IC}$ contains first-order constraints. We adopt an *active domain* assumption: all objects considered for grounding the program rules, evaluation of the action preconditions, the conditions of the actions constraints and the integrity constraints must be from DB. Under this assumption, the size of the ground instance of $\mathcal{P}$ over DB is polynomial in the size of DB, and the evaluation of integrity constraints is polynomial (as is the evaluation of preconditions and action constraints). Moreover, a set of ground actions AS is executable on DB in polynomial time.

Our results on the computational complexity of action programs are summarized in Tables 3.2 and 3.3 below. The hardness results for arbitrary $\mathcal{IC}$ hold for the case where $\mathcal{IC}$ contains only functional dependencies. The problems considered are:

(consistency) deciding the existence of an acceptable status set on a state DB;

(recognition) the recognition of an acceptable status set on DB;

(computation) the computation of an arbitrary acceptable status set on DB; and

(action reasoning) reasoning about whether the system executes an action α on DB, for

- the possibility variant (α is executed in some acceptable status set), and
- the certainty variant (α is executed in every acceptable status set).

Table 3.2 Complexity of fixed positive action programs

$\frac{\mathcal{IC} = \emptyset}{\mathcal{IC} \text{ arbitrary}}$	consistency	computation	recognition	action reasoning possible	action reasoning certain
feasible	$\frac{\text{P}}{\text{NP}}$	$\frac{\text{FP}}{\text{FNP}}$	P	$\frac{\text{P}}{\text{NP}}$	$\frac{\text{P}}{\text{co-NP}}$
rational	P	FP	P	P	P
reasonable	P	FP	P	P	P

Table 3.3 Complexity of fixed action programs with negation

$\frac{\mathcal{IC} = \emptyset}{\mathcal{IC} \text{ arbitrary}}$	consistency	computation	recognition	action reasoning possible	action reasoning certain
feasible	NP	FNP	P	NP	co-NP
rational	$\frac{\text{NP}}{\Sigma_2^P}$	$\frac{{}^{*}\ \text{FNP}//\log}{\text{F}\Sigma_2^P}$	co-NP	Σ_2^P	$\frac{\text{co-NP}}{\Pi_2^P}$
reasonable	NP	FNP	P	NP	co-NP
F-preferred rational	$\frac{\text{NP}}{\Sigma_2^P}$	$\frac{\text{FNP}//\log}{{}^{\dagger}\ \text{RP} \cdot \text{FP}_{\parallel}^{\Sigma_2^P}}$	$\frac{\text{co-NP}}{\Pi_2^P}$	$\frac{\Sigma_2^P}{\Sigma_3^P}$	$\frac{\Pi_2^P}{\Pi_3^P}$

F-preferred semantics selects those rational status sets S such that $\mathbf{F}(S)$ is minimal w.r.t. inclusion.

The entries for decision problems are well-known classes from the polynomial hierarchy (PH). For NP, co-NP, Σ_k^P, and Π_k^P they denote completeness. For P, we have completeness (under log space reductions) in general, depending on QL; since datalog has P-complete data complexity (Vardi, 1982), it can be seen that P-completeness holds for arbitrary (reasonable) QL except for recognition of feasible status sets.

The entries for the computation problem are the classes $\mathrm{F}\mathcal{C}$ and $\mathrm{FNP}/\!/\log$ (= $\mathrm{FNP}/\!/\mathrm{OptP}[O(\log n)]$) according to (Papadimitriou, 1994; Chen and Toda, 1993) (i.e., compute any solution for an instance of a search problem). The entries FNP, $\mathrm{F}\Sigma_2^P$, and $\mathrm{FNP}/\!/\log$ stand for completeness under appropriate polynomial-time reductions as used in (Papadimitriou, 1994; Chen and Toda, 1993), while $\mathrm{FNP}/\!/\log$ marked with '*' stands for membership in $\mathrm{FNP}/\!/\log$ and lower bounds of $\mathrm{FP}_{\|}^{\mathrm{NP}}$ and FNP. ($\mathrm{FP}_{\|}^{\mathrm{NP}}$ contains the problems solvable in polynomial time with parallel access to an NP oracle).

The class FNP (resp., $\mathrm{F}\Sigma_2^P$) contains the problems for which each solution can be generated nondeterministically in polynomial time, such that a given candidate for a solution can be checked in polynomial time (resp., the check is in co-NP). The class is contained in the class NPMV (resp., $\mathrm{NPMV}^{\mathrm{NP}}$), which contains all multi-valued functions computable in nondeterministic polynomial time (resp., in nondeterministic polynomial time with an NP oracle) (Fenner et al., 1997).

The class $\mathrm{FNP}/\!/\log$ (Chen and Toda, 1993) contains informally those search problems for which a solution can be nondeterministically found in polynomial time for instance I, if the optimal value $opt(I)$ of an NP optimization problem on I is known, where $opt(I)$ is a number which must have $O(\log|I|)$ bits; for example, computing a maximum clique in a graph is a typical problem in $\mathrm{FNP}/\!/\log$. It holds that with respect to problem solving power, $\mathrm{FP}^{\mathrm{NP}}$ is (presumably) more powerful than $\mathrm{FNP}/\!/\log$, which is (presumably) more powerful than $\mathrm{FP}_{\|}^{\mathrm{NP}}$ cf. (Chen and Toda, 1993). (Observe that $\mathrm{FNP}/\!/\log$, like FNP, expresses problem solving in the presence of *multiple* solutions (i.e., multivalued functions), while $\mathrm{FP}_{\|}^{\mathrm{NP}}$ and $\mathrm{FP}^{\mathrm{NP}}$ express problems with a single solution (i.e., single-valued functions). Thus, formally $\mathrm{FP}_{\|}^{\mathrm{NP}} \subset \mathrm{FNP}/\!/\log$ and $\mathrm{FNP}/\!/\log \not\subseteq \mathrm{FP}^{\mathrm{NP}}$ holds. $\mathrm{RP}\cdot\mathrm{FP}_{\|}^{\Sigma_2^P}$ is a randomized version of $\mathrm{FP}_{\|}^{\Sigma_2^P}$; the symbol '†' denotes hardness for $\mathrm{FP}_{\|}^{\Sigma_2^P}$ and $\mathrm{F}\Sigma_2^P$.

For space reasons, a detailed treatment and derivation of all these results is beyond the scope of this paper. Some of the results, in particular for the polynomial cases in Table 3.2, follow from the results of the next section. These and the other results are also obtained from the detailed analysis of the respective problems in the slightly more general framework described in (Eiter et al., 1998). Here, we confine to describe the intuition behind the results.

Briefly, the complexity of feasible status sets intuitively explained by analogy to models of a propositional formula. A feasible status set can be verified in polynomial time, but an exponential number of candidates for a feasible status set S exists. A positive program without integrity constraints has a least feasible status set, if it is consistent. For a rational status set, on a positive program only a single candidate S needs to be checked, which is polynomial time computable (see next section). Moreover, rational and reasonable status sets coincide on positive programs.

On general programs, the complexity of rational status sets can be intuitively explained by a rough analogy to minimal models of a propositional formula, where the groundedness condition corresponds to minimality. For $\mathcal{IC} = \emptyset$, this analogy is proper, while for arbitrary $\mathcal{IC}$, it no longer applies: here, a feasible status set does not necessarily include a rational status set. Thus, while existence of a feasible status set implies the existence of a rational status set S if $\mathcal{IC} = \emptyset$ (which yields NP), for arbitrary $\mathcal{IC}$, we can guess and verify S in polynomial time with an NP oracle, and there seems to be no more efficient algorithm in general.

Reasonable status sets are in analogy to stable models of a logic program (Gelfond and Lifschitz, 1988); observe that their complexity is immune to integrity constraints. For a suitable concept of stratification, their complexity is polynomial for all problems.

F-preferred rational status sets roughly correspond to a selection from the minimal models of a propositional formula, if minimization is restricted to part of the variables, and the remaining variables may float to support minimization; thus, they correspond to a particular instance of the so called $P; Z$-minimal models of a formula (Lifschitz, 1985).

The high complexity of action reasoning using F-preferred status sets can be intuitively explained as follows. There are three sources of complexity, which act in a way orthogonally on each other:

1. The (possibly exponential) number of candidates S for a feasible status sets;
2. the test whether S is grounded; and,
3. the (possibly exponential) number of candidates S' which are F-preferred over S.

For the reasonable status sets, only one of the above sources is present, namely (1) (we do not consider F-preferred reasonable status sets here). The intuitive reason is that the refutation of a reasonable status sets reduces to the computation of the rational status set of a positive action program, and there integrity constraints do not affect complexity.

3.5 ALGORITHMS

It is possible to characterize the unique rational status set of a positive program in terms of a fixpoint operator. Given any *positive* action program $\mathcal{P}$ and a database state DB, define the operator $T_{\mathcal{P},DB}$ which maps status sets to status sets as follows:

$$T_{\mathcal{P},DB}(S) \quad = \quad App_{\mathcal{P},DB}(S) \cup DCl(S) \cup ACl(S). \qquad \text{(cf. Def. 3.3.4,3.3.3)}$$

It is easy to see that $T_{\mathcal{P},DB}$ is a monotonic and continuous operator (Lloyd, 1987) on a complete lattice, and thus it has a least fixpoint $lfp(T_{\mathcal{P},DB})$ given by $lfp(T_{\mathcal{P},DB}) = \bigcup_{i\geq 0} T^i_{\mathcal{P},DB}$ where $T^0_{\mathcal{P},DB} = \emptyset$ and $T^{i+1}_{\mathcal{P},DB} = T_{\mathcal{P},DB}(T^i_{\mathcal{P},DB})$, for all $i \geq 0$. We then have the following result.

Theorem 2 *Let $\mathcal{P}$ be a positive action program. Then, for any DB, $lfp(T_{\mathcal{P},DB})$ is the unique rational status set of $T_{\mathcal{P},DB}$ on DB iff $lfp(T_{\mathcal{P},DB})$ is a feasible status set.* ∎

The following algorithm computes a rational status set for positive action programs.

Algorithm COMPUTE-P-RSS

Input: database state DB (fixed positive action program $\mathcal{P}$);

Output: the (unique) rational status set of $\mathcal{P}$, if a rational status set exists; "No", otherwise.

Step 1. Compute $S = \mathit{lfp}(T_{\mathcal{P},DB})$;

Step 2. Check whether S satisfies conditions $S2$ and $S4$ of a feasible status set;

Step 3. If S satisfies $S2$ and $S4$, then output S; otherwise, output "No". Halt ∎

By Theorem 2 and the fact that S satisfies $S1$ and $S3$, algorithm COMPUTE-P-RSS correctly computes the unique rational status set of $\mathcal{P}$ on DB. Step 1 is polynomial, since the ground instance of fixed $\mathcal{P}$ is polynomial in DB, and we can compute $S = \mathit{lfp}(T_{\mathcal{P},DB})$ bottom up by fixpoint iteration. Step 2, is, by our assumptions, possible in polynomial time. Therefore, for fixed $\mathcal{P}$, algorithm COMPUTE-POS-RATIONAL-SS runs in polynomial time.

Observe that if $\mathcal{IC} = \emptyset$, then $\mathcal{P}$ has a feasible status set on DB if and only if $\mathit{lfp}(T_{\mathcal{P},DB})$ is a feasible status set. Therefore, deciding the existence of a feasible status set (and computing one) is polynomial if $\mathcal{IC} = \emptyset$.

In the absence of integrity constraints, the rational status sets coincide with the $\subseteq$-minimal feasible status sets. Using an NP oracle, we can thus compute a rational status set as follows.

Algorithm COMPUTE-RATIONAL-SS

Input: database state DB (fixed action program $\mathcal{P}$, $\mathcal{IC} = \emptyset$);

Output: a rational status set of $\mathcal{P}$, if one exists; "No", otherwise.

Step 1. Set $S := \emptyset$ and At := set of all ground action status atoms.

Step 2. Check if S is a feasible status set; if true, then output S and halt.

Step 3. If $At = \emptyset$, then output "No" and halt.

Step 4. Choose some atom $A \in At$ and query the oracle whether a feasible status set S' exists such that $S \subseteq S' \subseteq S \cup (At \setminus \{A\})$; If the answer is "no", then $S := S \cup \{A\}$;

Step 5. set $At := At \setminus \{A\}$ and continue at Step 2. ∎

Clearly, COMPUTE-RATIONAL-SS runs in polynomial time modulo calls to the oracle. Hence, the problem is in $\mathrm{FP}^{\mathrm{NP}}$. This upper bound can be improved to $\mathrm{FNP}//\log$; indeed, if the size s of a smallest feasible status set S is known, then such an S (which is a rational status set) can be nondeterministically computed in polynomial time. Since computation of s is an NP-optimization problem with $O(\log|I|)$ output bits, it follows that computing a rational status set is in $\mathrm{FNP}//\log$ if $\mathcal{IC} = \emptyset$. In fact, it is polynomially equivalent to computing a (global) $\subseteq$-maximal model of a CNF formula, which is in $\mathrm{FNP}//\log$ and known to be $\mathrm{FP}^{\mathrm{NP}}_{\|}$-hard (Chen and Toda, 1993).

3.6 CONCLUSIONS

During the last few years, there has been an explosion in the amount of data accessible to database users. The sheer quantity of such data makes it very difficult for users to constantly monitor the content of the database. Active databases monitor conditions on databases and take actions as specified by the active rules defined by the application developer. Early work on active databases focused on building scalable implementations, e.g. the Starburst system of IBM (Ceri and Widom, 1991) and the Ariel system (Hanson, 1992).

More recently, there has been a trend towards developing a formal semantics for active databases. (Zaniolo, 1993) suggested the use of Datalog-1S for this purpose. (Fraternali and Tanca, 1995) also proposed a semantics for ECA rules. The expressive power of implemented active database systems has been analyzed in (Picouet and Vianu, 1995), where computations over time are considered. If we consider evaluation of action programs over time, then over a fixed active domain, all computations in PSPACE are feasible, while if new values can be suitably created, all computations are possible. (Baral and Lobo, 1996) use situation semantics (a more general version of Datalog-1S) to give a logic program semantics for active rules. (Gottlob et al., 1996) proposed a semantics for active rules which allowed the user to specify different conflict resolution policies. (Ludäscher et al., 1997) develop methods to handle actions so as to preserve referential integrity constraints.

However, in many applications (e.g. the tax application, and various legal applications), the database administrators have certain legal obligations (that is, they are required to take certain actions), as well as certain restrictions (that is, they are forbidden to take certain actions) if certain conditions are true. However, not all actions are either forbidden or obligatory. The vast majority of actions fall within a "gray" area – they are permitted, but neither obligatory or forbidden. To date, no active database system has provided a formal semantics for obligatory, permitted, and forbidden actions. In this paper, we have done so, building on top of classical deontic logic syntax (Åquist, 1984; Meyer and Wieringa, 1993).

Elements of deontic logic have been repeatedly used in the context of computer security to build a formal system for expressing security and authorization policies. Examples are the approach by (Glasgow et al., 1989) and the security logic proposed in (Bieber and Cuppens, 1993), where further work on this subject is reviewed. The quoted approaches basically propose logics of knowledge in a modal language, which are extended by an operator $P\phi$ which expresses permission to know ϕ; in (Bieber and Cuppens, 1993), the operator is indexed by time, and also the deontic operators O and F are available. The systems allow arbitrarily nested formulas, and suitable axioms for a formal system (including the security axiom $K\phi \rightarrow P\phi$) as well as Kripke style semantics are investigated. Thus, these approaches are in the tradition of standard modal logic, and in they aim at providing abstract frameworks for expressing different security policies on information access (viewed as knowledge) and reasoning about statements in these logics. Our system is concerned with singling out actions for execution, a rather different task.

We have added to deontic logic as well in two ways: first, we have introduced the **Do** operator which standard deontic logic does not contain. Second, our semantics is completely new – to our knowledge, the concepts of feasible, rational and reasonable status sets are novel and new. Last, but not least, our framework also allows nonmonotonic inference through the negation operator in rule bodies.

Acknowledgments

We gratefully acknowledge the support of the Army Research Office under Grants DAAH-04-95-10174, DAAH-04-96-10297, and DAAH04-96-1-0398, of the Army Research Laboratory under contract number DAAL01-97-K0135, of an NSF Young Investigator award IRI-93-57756, and of a DAAD grant.

The main part of this work has been carried out while the first author was with the Institute of Informatics, University of Gießen, Germany.

Appendix:

This appendix provides sketches for the proofs of some results. Proofs of all result are contained in (Eiter et al., 1998).

Theorem 1 *Let AS be a set of ground actions and DB a database state, and suppose that QL contains conjunctive queries on DB. Then, deciding whether AS is concurrently executable is* co-NP-*hard.*

Proof. (Sketch) To show that the problem is co-NP-hard, we reduce the problem M3UNSAT, the complement of M3SAT (Garey and Johnson, 1979), to concurrent executability checking. Let I be an instance of M3UNSAT, consisting of positive clauses $P_1, \ldots, P_{n_p}$ and negative clauses $N_1, \ldots, N_{n_n}$, $n_p, n_n \geq 1$ over atoms $a_1, \ldots, a_n$. Here, each clause is supposed to have three (not necessarily distinct) literals.

Let I' be the instance of I which results if every positive clause is made negative and vice versa, and if every atom a_i is replaced by a_{n+i}. Clearly, I' is a Yes-instance of M3UNSAT if and only if I is a Yes-instance, if and only if $I \cup I'$ is not satisfied by some truth assignment to $a_1, \ldots, a_{2n}$ in which a_i has opposite value to the value of a_{n+i}, for every $i = 1, \ldots, n$.

The relational database $\mathcal{DB}$ we construct has four relations, namely

- POS(V_1, V_2, V_3), NEG(V_1, V_2, V_3), which serve for storing the positive and negative clauses of $I \cup I'$, respectively;
- VAL(Var, BV), which stores a truth value assignment to the variables, s.t. variable a_i is true if $(a_i, 1) \in$ VAL, and a_i is false if $(a_i, 0) \in$ VAL, for every $i = 1, \ldots, 2n$;
- a 0-ary relation (propositional letter) UNSAT.

The initial database DB contains the relations POS and NEG storing the clauses of $I \cup I'$, the relation VAL which holds the tuples $(a_i, 0)$ and $(a_{n+i}, 1)$, for every $i = 1, \ldots, n$, and the relation UNSAT is empty.

The action base contains the actions $\texttt{switch}(X, Y)$ and $\texttt{eval}$, where

- $Pre(\texttt{switch}(X, Y)) =$ true,

  $Add(\texttt{switch}(X, Y)) = \{\, \text{VAL}(X, 1), \text{VAL}(Y, 0)\}$, and

  $Del(\texttt{switch}(X, Y)) = \{\text{VAL}(X, 0), \text{VAL}(Y, 1)\}$;

- $$\begin{aligned} Pre(\texttt{eval}) = \{\, & \text{POS}(V_1, V_2, V_3) \wedge \text{VAL}(V_1, 0) \wedge \text{VAL}(V_2, 0) \wedge \text{VAL}(V_3, 0) \wedge \\ & \text{NEG}(V_1', V_2', V_3') \wedge \text{VAL}(V_1', 1) \wedge \text{VAL}(V_2', 1) \wedge \text{VAL}(V_3', 1) \,\}, \end{aligned}$$

  $Add(\texttt{eval}) = \{\text{ UNSAT }\}$, $Del(\texttt{eval}) = \emptyset$.

(The free variables in $Pre(\texttt{eval})$ are existentially quantified.) Observe that we assume tacitly that the action base is fixed; otherwise, already the evaluation of varying conjunctive queries is intractable.

The action set is $AS = \{\texttt{switch}(a_i, a_{n+i}) \mid 1 \leq i \leq n\} \cup \{\texttt{eval}\}$.

Intuitively, an action $\mathtt{switch}(a_i, a_{n+i})$ flips the value of a_i from 0 to 1 and the value of a_{n+i} from 1 to 0. The `eval` action checks whether for the truth assignment to $a_1, \ldots, a_{2n}$ given in VAL, there is some positive clause and some negative clause which are both violated.

Clearly, all actions in AS are executable on DB, and it is easy to see that every feasible $AS[\pi]$ yields the same result DB'. It can be shown that AS is concurrently executable, if and only if I is a Yes-Instance of M3UNSAT. ∎

Proposition 3.3.2 *Let $\mathcal{P}$ be an action program and DB a database state. Then, every reasonable status set of $\mathcal{P}$ on DB is a rational status set of $\mathcal{P}$ on DB.*

Proof. (Sketch) To show that a reasonable status set S of $\mathcal{P}$ is a rational status of $\mathcal{P}$, we have to verify that S is a feasible status set and that S is grounded.

Since S is a reasonable status set of $\mathcal{P}$, it is a rational status set of $\mathcal{P}' = red^S(\mathcal{P}, DB)$, i.e., a feasible and grounded status set of $\mathcal{P}'$. As the conditions $(S2)$–$(S4)$ of the definition of feasible status set depend only on S and DB but not on the program, for showing that S is feasible it remains to check $(S1)$ (closure under the program rules).

Let thus r be a ground instance of a rule from $\mathcal{P}$. Suppose the body of r satisfies the conditions 1.–4. of $(S1)$. Then, by the definition of $red^S(\mathcal{P}, DB)$, the reduct of the rule r obtained by removing all negative status literals is in $\mathcal{P}'$. Since S is closed under the rules of $\mathcal{P}'$, the head of r is in S. Thus, S is closed under the rules of $\mathcal{P}$, which means $(S1)$ is satisfied. It follows that S is feasible status set.

Assuming that S is not grounded, i.e., some smaller $S' \subset S$ satisfies $(S1)$–$(S3)$ for $\mathcal{P}$, a contradiction can be derived. If S' satisfies $(S1)$ for $\mathcal{P}$, then S' satisfies $(S1)$ for $\mathcal{P}'$. For, if r is a rule from $\mathcal{P}'$ such that 1.–4. of $(S1)$ hold, then there is a ground rule r' of $\mathcal{P}$ such that r is obtained from r' in the construction of $red^S(\mathcal{P}, DB)$ and, as easily seen, 1.–4. of $(S1)$ hold. Since S' satisfies $(S1)$ for $\mathcal{P}$, the head of r is in S'. It follows that S' satisfies $(S1)$ for $\mathcal{P}'$. Since conditions $(S2)$ and $(S3)$ are independent of the program, they are satisfied for S' w.r.t. $\mathcal{P}'$. This means that S is not a rational status set of $\mathcal{P}'$, which is the desired contradiction. ∎

Theorem 2 *Let $\mathcal{P}$ be a positive action program. Then, for any DB, $lfp(T_{\mathcal{P},DB})$ is the unique rational status set of $T_{\mathcal{P},DB}$ on DB iff $lfp(T_{\mathcal{P},DB})$ is a feasible status set.*

Proof. (Sketch) From the definition of the operator $T_{\mathcal{P},DB}$, one can show by induction on the stages i of the operator that each feasible status S must include $T^i_{\mathcal{P},DB}$, $i \geq 0$, and hence S must include $lfp(T_{\mathcal{P},DB})$.

As $lfp(T_{\mathcal{P},DB})$ is a fixpoint of $T_{\mathcal{P},DB}$, it satisfies the conditions $(S1)$–$(S3)$ of a feasible status set. Thus, from the definitions of groundedness and rational status set it follows that the only candidate for a rational status set of $\mathcal{P}$ on DB is $lfp(T_{\mathcal{P},DB})$ itself. This set is rational iff it is a feasible status set, i.e., $lfp(T_{\mathcal{P},DB})$ also satisfies $(S4)$ besides $(S1)$–$(S3)$. ∎

Notes

1. Deontic concepts were used in McCarty's classic TAXMAN system (McCarty, 1977) for corporate tax law. The purpose of this system, whose foundations have been extended in (McCarty, 1983), was checking whether a corporate reorganization can be classified as a tax-free transaction, cf. (Meyer and Wieringa, 1993).

References

Åquist, L. (1984). Deontic Logic. In Gabbay, D. and Guenthner, F., editors, *Handbook of Philosophical Logic, Vol.II*, chapter II.11, pages 605–714. D. Reidel Publishing Company.

Baral, C. and Lobo, J. (1996). Formal Characterization of Active Databases. In Pedreschi, D. and Zaniolo, C., editors, *Workshop of Logic on Databases (LID '96)*, volume 1154 of *Lecture Notes in Computer Science*, pages 175–195, San Miniato, Italy.

Bieber, P. and Cuppens, F. (1993). Expression of Confidentiality Policies. In (Meyer and Wieringa, 1993), chapter 5, pages 103–123.

Ceri, S. and Widom, J. (1991). Deriving Production Rules for Incremental View Maintenance. In *Proceedings VLDB-91*, pages 577–589.

Chen, Z.-Z. and Toda, S. (1993). The Complexity of Selecting Maximal Solutions. In *Proceedings of the 8th IEEE Structure in Complexity Theory Conference*, pages 313–325.

Dayal, U., Hanson, E., and Widom, J. (1995). Active Database Systems. In Kim, W., editor, *Modern Database Systems*, pages 434–456. Addison Wesley.

Eiter, T., Subrahmanian, V., and Pick, G. (1998). Heterogenous Active Agents. Technical Report UMCP-CSD CS-TR-3880, Computer Science Department, University of Maryland, College Park, Maryland. iii + 114 pp. Also IFIG-RR 9802, University of Gießen.

Fenner, S., Homer, S., Ogihara, M., and Selman, A. (1997). Oracles That Compute Values. *SIAM Journal on Computing*, 26(4):1043–1065.

Fraternali, P. and Tanca, L. (1995). A Structured Approach for the Definition of the Semantics of Active Databases. *ACM Trans. on Database Syst.*, 20(4):414–471.

Garey, M. and Johnson, D. S. (1979). *Computers and Intractability – A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York.

Gelfond, M. and Lifschitz, V. (1988). The Stable Model Semantics for Logic Programming. In *Logic Programming: Proceedings Fifth Intl Conference and Symposium*, pages 1070–1080, Cambridge, Mass. MIT Press.

Ghandeharizadeh, S., Hull, R., and Jacobs, et al, D. (1993). On Implementing a Language for Specifying Active Database Execution Models. In *Proceedings VLDB-93*, pages 441–454.

Glasgow, J., MacEwen, G., and Panangaden, P. (1989). Security by Permission in Databases. In Landwehr, C., editor, *Database Security II: Status and Prospects*, pages 197–205, Results of the IFIP WG 1.3 Workshop on Database Security (1988), Ontario, Canada. North Holland.

Gottlob, G., Moerkotte, G., and Subrahmanian, V. (1996). The PARK Semantics for Active Rules. In *Proceedings Int'l Conf. on Extending Database Technology (EDBT '96)*, pages 35–55.

Hanson, E. (1992). Rule Condition Testing and Action Execution in Ariel. In *Proceedings ACM SIGMOD '92*, pages 49–58.

Johnson, D. S. (1990). A Catalog of Complexity Classes. In van Leeuwen, J., editor, *Handbook of Theoretical Computer Science*, volume A, chapter 2. Elsevier Science Publishers B.V. (North-Holland).

Lifschitz, V. (1985). Computing Circumscription. In *Proceedings International Joint Conference on Artificial Intelligence (JCAI-85)*, pages 121–127.

Lloyd, J. (1984, 1987). *Foundations of Logic Programming*. Springer, Berlin.

Ludäscher, B., May, W., and Lausen, G. (1997). Actions as Logical Rules. In *Proceedings PODS-97*, pages 217–277.

McCarty, T. (1977). Reflections on TAXMAN: An Experiment in Artificial Intelligence and Legal Reasoning. *Harvard Law Review*, 90:837–893.

McCarty, T. (1983). Permissions and Obligations. In *Proceedings IJCAI '83*, pages 287–294.

Meyer, J.-J. C. and Wieringa, R., editors (1993). *Deontic Logic in Computer Science*. Wiley & Sons, Chichester et al.

Nilsson, N. (1980). *Principles of Artificial Intelligence*. Morgan Kaufmann.

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley.

Picouet, P. and Vianu, V. (1995). Semantics and Expressiveness Issues in Active Databases. In *Proceedings PODS-95*, pages 126–138.

Vardi, M. (1982). Complexity of Relational Query Languages. In *Proceedings 14th STOC*, pages 137–146, San Francisco.

Zaniolo, C. (1993). On the Unification of Active Databases and Deductive Databases. In *Advances in Databases: Proceedings 11th British National Conference on Databases (BNCOD 93)*, number 696 in LNCS, pages 23–39. Springer.