Default Databases: Extending the Approach of Deductive Databases Using Default Logic

Anthony Hunter
Dept. of Computer Science
University College London
Gower Street
London WC1E 6BT
Tel: +44 171 380 7295
Fax: +44 171 387 1397
Email: a.hunter@cs.ucl.ac.uk

Peter McBrien
Dept. of Computer Science
King's College London
Strand
London WC2R 2LS
Tel: +44 171 873 2469
Fax: +44 171 873 2851
Email: pjm@dcs.kcl.ac.uk

Tuesday 31st March 1998

Abstract

Extending the relational data model using classical logic to give deductive databases has some significant benefits. In particular, classical logic rules offer an efficient representation: a universally quantified rule can represent many facts. However, classical logic does not support the representation of general rules, or synonymously defaults. General rules are rules that are usually valid, but occasionally have exceptions. They are useful in a database since they can allow for the derivation of relations on the basis of incomplete information. The need for incorporating general rules into a database is reinforced when considering that participants in the development process may naturally describe rules for a deductive database in the form of general rules. In order to meet this need for using general rules in databases, we extend the notion of deductive databases. In particular, we use default logic, an extension of classical logic that has been developed for representing and reasoning with default knowledge, to formalize the use of general rules in deductive databases, to give what we call *default databases*. In this paper, we provide an overview of default logic, motivate its applicability to capturing general rules in databases, and then develop a framework for default databases. In particular, we propose a methodology for developing default databases that is based on entity-relationship modelling.

1 Introduction

Handling uncertain information is becoming an increasingly significant issue in information technology. However, uncertainty in information is not a homogeneous concept. Rather there are clearly diverse types, such as fuzzy, probabilistic, and default information, that require distinct means for handling (see for example [**?**]).

Interest in default knowledge started with attempts to handle general rules, or defaults, of the form "if α holds, then β normally holds", where α and β are propositions. It is noteworthy that human practical reasoning relies much more on exploiting general rules (not to be understood as universal laws) than on a myriad of individual facts. General rules tend to be less than 100%

accurate, and so have exceptions. Nevertheless it is intuitive and advantageous to resort to such defaults and therefore allow the inference of useful conclusions, even if it does entail making some mistakes as not all exceptions to these defaults are necessarily known. Furthermore, it is often necessary to use general rules when we do not have sufficient information to allow us to specify or use universal laws.

The notion of default knowledge covers a diverse variety of information, including heuristics, rules of conjecture, null values in databases, closed world assumptions for databases, and some qualitative abstractions of probabilistic information. Defaults are a natural and very common form of information. There are also obvious advantages to applying the same default a number of times: There is an economy in stating (and dealing with) only a general rule instead of stating (and dealing with) maybe thousands of instances of such a general rule.

General rules are potentially useful in databases since they can allow for the derivation of relations on the basis of incomplete information. So for example, different types of null value can be assigned, or default facts can be inferred, according to the information available. The need for incorporating general rules into a database is reinforced when considering that participants in the development process may naturally describe rules for a deductive database in the form of general rules.

Extending the relational data model using classical logic to give deductive databases has some significant benefits. In particular, classical logic rules offer an efficient representation: A universally quantified rule can represent many facts. However, classical logic does not support the representation of general rules, or synonymously defaults. General rules are rules that are usually valid, but occasionally have exceptions.

In order to meet this need for using general rules in databases, we extend the notion of deductive databases. In particular, we use default logic, an extension of classical logic that has been developed for representing and reasoning with default knowledge [**?**], to formalize the use of general rules in deductive databases, to give what we call *default databases*. Previously, Cadoli *et al.* have proposed default logic as a query language for finite relational databases [**?**]. We will adopt the same language, which they call default query language (DQL), for our default databases.

In this paper, we provide an overview of default logic, motivate its applicability to capturing general rules in databases, for generating default facts, and then develop a framework for default databases. In particular, we propose a methodology for developing default databases that is based on entity-relationship modelling. Finally, we explore some properties of the framework, and in particular, discuss how it builds upon DQL.

2 Overview of Default Logic

As a basis of representing default knowledge we employ **default logic** originally introduced by Reiter [**?**]. Default logic is one of the best known and most widely studied formalisations of default reasoning [**?**, **?**, **?**, **?**]. Furthermore, it offers a very expressive and lucid language. In default logic, knowledge is represented as a *default theory*, which consists of a set of first-order formulae and a set of *default rules* for representing default information. Default rules are of the following form, where α, β and γ are first-order (classical) formulae,

$$\frac{\alpha : \beta}{\gamma}$$

The inference rules are those of classical logic plus a special mechanism to deal with default rules: Basically, if α is inferred, and $\neg\beta$ cannot be inferred, then infer γ. For this, α is called the pre-condition, β is called the justification, and γ is called the consequent.

Default logic is an extension of classical logic. Hence, all classical inferences from the classical information in a default theory are derivable (if there is an extension). The default theory then augments these classical inferences by default inferences derivable using the default rules.

More formally, we introduce the operator Γ that indicates what conclusions are to be associated with a given set E of formulae, where E is some set of classical formulae. Let (D, W) be a default theory, where D is a set of default rules and W is a set of classical formulae. Let Th be the function that for a set of formulae returns the set of classical consequences of those formulae. For this, $\Gamma(E)$ is the smallest set of classical formulae such that the following three conditions are satisfied.

1. $W \subseteq \Gamma(E)$
2. $\Gamma(E) = Th(\Gamma(E))$
3. For each default in D, where α is the pre-condition, β is the justification, and γ is the consequent, the following holds:

$$\text{if } \alpha \in \Gamma(E) \text{ and } \neg\beta \notin E \text{ then } \gamma \in \Gamma(E)$$

Once $\Gamma(E)$ has been identified, E is an extension of (D, W) iff $E = \Gamma(E)$. If E is an extension, then the first condition ensures that the set of classical formulae W is also in the extension, the second condition ensures the extension is closed under classical consequence, and the third condition ensures that for each default rule, if the pre-condition is in the extension, and the justification is consistent with the extension, then the consequent is in the extension.

We can view E as the set of formulae for which we are ensuring consistency with the justification of each default rule that we are attempting to apply. We can view $\Gamma(E)$ as the set of conclusions of a default theory: It contains W, it is closed under classical consequence, and for each default that is applicable (i.e. the precondition is in $\Gamma(E)$ and the justification is satisfiable with E), then the consequent is in $\Gamma(E)$. We ask for the smallest $\Gamma(E)$ to ensure that each default rule that is applied is grounded. This means that it is not the case that one or more default rules are self-supporting. For example, a single default rule is self-supporting if the pre-condition is satisfied using the consequent. We explain the notion of grounded further in the next section. The test $E = \Gamma(E)$ ensures that the set of formulae for which the justifications are checked for consistency coincides with the set of conclusions of the default theory.

We have chosen default logic as the formalism for default databases because it is one of the most well-explored formalisms for default knowledge. There is a range of useful variants, and inferencing technology is being developed (see Section 2.3). Last, but not least, default logic offers a natural and straightforward route for developing default databases.

In Section 2.1, we give some simple examples of default theories, and of the kinds of reasoning that follow from them. These examples use propositional formulae for the precondition, justification, and consequent. But, we can use first-order formulae. In Section 2.2, we give some examples where we use first-order formulae.

2.1 Some examples of default reasoning

Here we consider some examples of default logic.

1. Consider the default theory (D, W) where $W = \{\alpha, \neg\delta\}$, and D comprises the following two defaults.

$$\frac{\alpha\ :\ \beta}{\gamma} \qquad \frac{\gamma\ :\ \delta}{\beta}$$

¿From (D, W), we obtain an extension E that is the classical closure of the set $\{\alpha, \neg\delta, \gamma\}$. Note, β is not a member because the application of the second default is blocked by the presence of $\neg\delta$ in W.

2. Not all default theories have a unique extension. For example, consider the default theory (D, W), where $W = \emptyset$ and D contains just the following two default rules. Note, $\top$ is the propositional constant "true". It holds in any set of classical formulae.

$$\frac{\top : \alpha}{\alpha} \quad \frac{\top : \neg\alpha}{\neg\alpha}$$

Here there are two extensions. The first is the closure of $\{\alpha\}$ and the second is the closure of $\{\neg\alpha\}$.

3. Also, not all default theories have an extension. Consider the default theory (D, W), where $W = \emptyset$ and D contains just the following default rule.

$$\frac{\top : \alpha}{\neg\alpha}$$

4. All extensions must be grounded. Consider the default theory (D, W), where $W = \{\alpha\}$ and D contains just the following two default rules.

$$\frac{\beta : \gamma}{\gamma} \quad \frac{\gamma : \beta}{\beta}$$

Here the first default provides the precondition for the second, and vice versa. They are self-supporting, or ungrounded. These defaults are not applicable in this default theory, and so the extension contains just the classical closure of W.

2.2 Handling multiple extensions

Possible sets of conclusions from a default theory are given in terms of **extensions** of that theory. A default theory can possess multiple extensions because different ways of resolving conflicts among default rules lead to different alternative extensions. Two options for handling multiple extensions are the **credulous** approach, where we accept a query if it belongs to one of the extensions of a considered default theory, and the **skeptical** approach, where we accept a query if it belongs to all extensions of the default theory.

In a skeptical view, the logic is cautious and does not allow conflicting inferences, whereas in a credulous view, the logic is less cautious, and does allow conflicting inferences. The rationale behind a credulous view is that the user makes a selection from the conflicting inferences. For example, take the following defaults,

$$\frac{\mathsf{aircraft(X) : require\text{-}runway(X)}}{\mathsf{require\text{-}runway(X)}}$$

$$\frac{\mathsf{helicopter(X)} : \neg\mathsf{require\text{-}runway(X)}}{\neg\mathsf{require\text{-}runway(X)}}$$

and the following facts $W = \{\mathsf{aircraft(Sikorsky)}, \mathsf{helicopter(Sikorsky)}\}$. From this, there are two extensions, the first including require-runway(Sikorsky) and the second including ¬require-runway(Sikorsky). A credulous view would allow both require-runway(Sikorsky) and ¬require-runway(Sikorsky) as possible inferences, whereas a skeptical view would allow neither.

An alternative approach to eliminating the possibility of multiple extensions in this example is to amend the first rule slightly as follows.

$$\frac{\mathsf{aircraft(X)} : \neg\mathsf{helicopter(X)}}{\mathsf{require\text{-}runway(X)}}$$

¿From this, we now only derive one extension, containing ¬require-runway(Sikorsky), since the application of the amended rule is blocked by the fact that helicopter(Sikorsky) holds. If we now add the fact aircraft(Fokker), the amended default rule allows for the derivation of require-runway(Fokker), since we can't derive helicopter(Fokker). In this way, we have incorporated an implicit preference for the with the precondition helicopter(X) in the context of helicopter holding for a particular instance.

2.3 Inference Engines for Default Logic

There are experimental implementations of automated reasoning systems for default logic and related logics [?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?]. These include systems that compute extensions and systems that provide credulous and skeptical query-answering.

An inference engine can be viewed as being composed of classical reasoning and conflict resolution. These two activities are orthogonal sources of complexity in default reasoning [?]. For classical reasoning, automated reasoning techniques exist. Both classical deduction and consistency checking can be used in default reasoning. Conflict resolution is currently an important research area in default reasoning. The initial search space for the conflict resolution task is exponential with respect to the number of default rules and brute force search methods are able to handle only very modest sized sets of default rules.

For certain classes of default databases, where the interaction of default rules is limited, there exist techniques to solve the conflict resolution task in polynomial time [?]. In addition, there are some positive results on pruning the search space [?, ?], and on seeking fast approximate answers to queries [?].

3 Default Databases

The distinction between intensional and extensional data in database systems is well understood [?], and the usefulness of being able to define data intensionally widely appreciated, as witnessed by the use of views in commercial DBMS. Since default logic is an extension of the classical logic normally used to define data intensionally, we differentiate three types of object, the first two of which come from the same notions in datalog [?]:

- **extensional database** (EDB) objects are stored in the database, and manifest themselves as unit Horn clauses in the data model.
- **intentional database** (IDB) objects are specified solely by Horn clauses with a model-theorectic interpretation. Function symbols may not be used as arguments. All rules must be safe, in the sense that if all the predicates they contain only return finite extensions, then the result of the rule is also finite.
- **default rules database** (DDB) objects are specified as clauses in default logic, which we will detail below.

In this way, we propose that default databases extend the notion of deductive databases: an extensional database is a set of ground classical relations; an intensional database is set of classical formulae formed without functions; and a default rules database is a set of default rules of the following form,

$$\frac{\alpha : \beta}{\gamma}$$

where α, β, and γ are first-order formulae that contain no quantifiers or functions, though they may contain unquantified variables. Each default rule is a scheme where variables can be grounded only with constants. The set of constants is the set of attribute values used in the EDB. This language is the query language proposed in [**?**].

Now we have defined the components, we can define the notion of a default database. A default database is a default theory (D, W) where D is a default rules database, and W is the union of an extensional database and an intentional database.

Consider the following example where the extensional database is the relation $\mathsf{airline-customer}$ containing the tuples $\{\mathsf{John}, \mathsf{Mary}, \mathsf{Ann}\}$, the monadic relation $\mathsf{cargo\text{-}customer}$, containing $\{\mathsf{John}\}$, the intensional database is $\{\mathsf{cargo-customer(X)} \rightarrow \mathsf{exception(X)}\}$, and the default rule database contains just the following rule:

$$\frac{\mathsf{airline\text{-}customer(X)}\ :\neg\ \mathsf{exception(X)}}{\mathsf{passenger(X)}}$$

¿From the extensional and intensional databases, we obtain $\mathsf{exception(John)}$, and so from the default rule database, we obtain $\mathsf{passenger(Mary)}$, and $\mathsf{passenger(Ann)}$, but not $\mathsf{passenger(John)}$. If at a future point in time, say Ann is added to the $\mathsf{cargo\text{-}customer}$ relation, then $\mathsf{passenger(Ann)}$ can no longer be derived. In this way the approach is non-monotonic.

Theoretically, the relations generated using the default rule database can appear like any other relation. There can be access transparency. However, default relations can be flagged to indicate to the user the tentative nature of the tuples returned. If credulous reasoning is used, then something of this nature would become necessary in order to present the options made by multiple extensions.

In the next two sections, we consider data modelling, and rule engineering, respectively, for default databases. The data modelling is an extension of standard data modelling techniques used for databases. It covers the EDB, IDB, and DDB. The rule engineering is an approach to constructing the DDB.

4 Data Modelling for Default Databases

For a new computer technology such as default logic to gain acceptance in a commercial environment, it is essential that not only does the technology enhance the basic information processing capabilities of the organisation, but also that software engineering techniques are provided in the form of (1) tools to facilitate the modelling of the domain and capture of default rules applicable to the domain, and (2) methodological support for how the tools should be used.

4.1 Default Logic & Conceptual Modelling

Conceptual modelling of information systems, as exemplified by **structural analysis** [**?**] and **SSADM** [**?**], make a distinction between the modelling of the static and dynamic aspects of information systems, where:

- The **static aspects** cover those things in the **universe of discourse** (UoD) which have the implied notion of duration, such as people, departments, goods, *etc.* In practice usually some (mainly syntactic) variant of **entity-relationship** (ER) modelling [**?**] is used.
- The **dynamic aspects** cover those things which relate to the processing and alteration of the static aspects in the UoD, such as people changing departments, of departments buying goods. Some variant of **data-flow diagrams** (DFD) [**?**, **?**] is often used for modelling dynamic aspects, but there is less consensus as to the type of model to be used.

More recently, object-orientation has achieved considerable attention as a mechanism for modelling both static and dynamic aspects of an information system in an integrated model. However, we will concentrate on the more traditional approaches, since to gain acceptance, it is necessary that the new technology of default reasoning can be utilised using the same techniques used to manage other existing software in an organisation. This requires that the modelling languages we provide are as far as possible simple modifications of existing modelling languages.

We propose that the **semantic modelling** of a default reasoning system be divided into three connected models:

- the **data modelling** of objects in the UoD, their attributes, and the relationships between objects.
- the **process modelling** of the dynamics of the UoD, describing how the system interacts with external agents, and the sequence of actions that take place in the processing of information.
- the **rule modelling** of the logical constraints on the data and process model, formulated in either default logic or classical logic. Part of the rule modelling activity will necessitate the ability to navigate between rules according to their interactions with each other as defaults.

This division in the modelling allows us to connect the default reasoning supported in the rule modelling with the classical first order view of information in the data and process models, and hence with existing applications. This approach was used in the TEMPORA [**?**, **?**] CASE tool for connecting deductive and active reasoning in classical logic to standard information system data modelling, and we will following this approach in our handling of default reasoning.

As far as the specific issue of supporting the use of default reasoning in information processing is concerned, there are two activities that we need to address:

- How do we deduce what is subject to default reasoning in the domain, and capture this in conceptual modelling languages?
- How do we use those conceptual modelling languages to build a working information system that supports the use of default reasoning?

Before studying the two questions as they impact upon data and process modelling, we will define the ER model we shall use in the remainder of this paper, and introduce a small example to use in the discussion.

4.2 Entity Relationship Modelling

Table 1 defines the graphical constructs of the ER modelling language we shall use in this paper. The modelling language is basically the enriched ER of [?], an extension of the original notation of Chen [?]. The most important of the extensions is the introduction of concepts of **subset** (allowing is-a associations between entity sets) and **generalisations** (which produce the union entity set of entity sets). For the purposes of simplifying the presentation of the relationship to default logic, we will also assume that all relationships are binary, though this assumption is not essential to our work. The additions we make to the notation of [?] are shown in Table 2, and which are used describe when objects are implemented as EDB, IDB or DDB. As in [?], we use dashed lines to indicate any object which is implemented as IDB rather than EDB. We introduce the use of shading to indicate any object which is implemented as DDB. It should be noted that the declarative reading of the ER model given in Table 1 is not altered by the use of the notations in Table 2 indicating the implementation of objects as EDB, IDB or DDB.

Figure 1 represents an ER model, which we use in an example of designing an information system to handle claims arriving at an insurance company. Each claim must be associated with a policy, where policies are identified by a number, and have recorded the name and age of the holder, the number of years that the policy has run, and the total number of claims made in the policy. Claims are identified by a serial number SNo, and may be divided into three types fire, accident or theft. Part of the system involves deciding if a particular claim should be paid without human intervention, or should be sent to the investigation department for further consideration. This is subject to a number of rules, which appear to some extent be contradictory. Constructing such an ER model may largely be achieved by following textbook design techniques for ER models [?, ?]. However we have introduced the notions of DDB, IDB and DDB objects, which can be initially differentiated during design as follows:

- EDB objects are things in the UoD which can be regarded as non-derivative facts. In our example things like policies, the names of the policy holders, and the claims they make are clearly examples of objects of this type.
- IDB objects are things in the UoD which are directly derivatives of other facts. In our example, the number of claims made on a policy is a derivative of the number of claims we have on a policy.
- DDB objects are things in the UoD which are essentially non-factual, but rather the subject some sort of decision making process. In our example, the method by which we handle claims is an example of an object of this type.

These vague notions will be formalised when we come to build rule sets for the system. The initial denotation of objects in the ER model as EDB, IDB and DDB will focus the capture of rules to those objects that require them. Before we can write rules, we must represent the ER

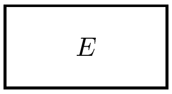
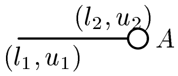
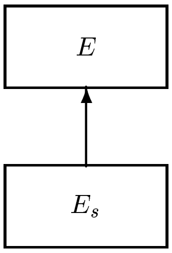
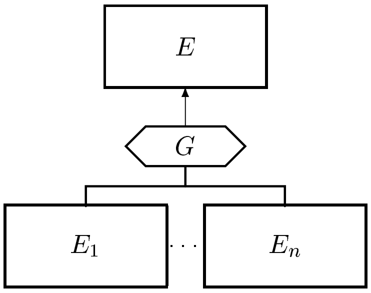
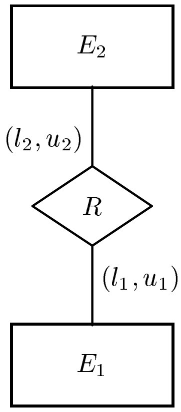
Name	*Symbol*	*Description*
entity set		Describes a set of entities E in the UoD which share some common properties. The instances of E are identified by a key attribute e, and the predicate $E(e)$ holds for each e in the DB.
attribute		Describes a set of values A which are of a certain type, and are associated with members of an entity set. The cardinality constraint (l_1, u_1) gives the minimum and maximum number of values that may be associated with any one entity instance, and (l_2, u_2) gives the minimum and maximum number of entity instances that may be associated to a particular value. The predicate $E\_A(e, a)$ holds for each instance a of an attribute A of E, where e is instance of E.
key attribute		An attribute for which $l_1 = u_1 = l_2 = u_2 = 1$, and thus is a key attribute.
subset		Denotes that any instance of the **subset entity** E_s is also a member of the **superset entity** E.
generalisation		Denotes a series of subset entities $E_1, \ldots, E_n$ of superset E, which have the additional property that the instances of E must be also instances of exactly one of $E_1, \ldots, E_n$.
relationship		Denotes a set R of binary associations between entities, where the associations are all of a certain type. The predicate $E_1\_R\_E_2(e_1, e_2)$ holds for each pair of entities from entity sets E_1, E_2 which are involved in a relationship R.

Table 1: Constructs and declarative semantics of an entity-relationship model

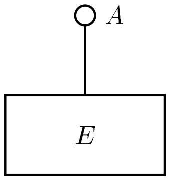
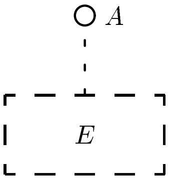
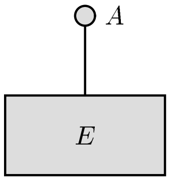
Name	*Symbol*	*Description*
EDB		A solid lined and white filled version of any symbol indicates that it models an EDB object.
IDB		A dashed version of any symbol indicates that it models an IDB object, but the declarative semantics remain the same.
DDB		A shaded version of any symbol indicates that it models an DDB object, but the declarative semantics remain the same.

Table 2: Identification of implementation method used for an ER model

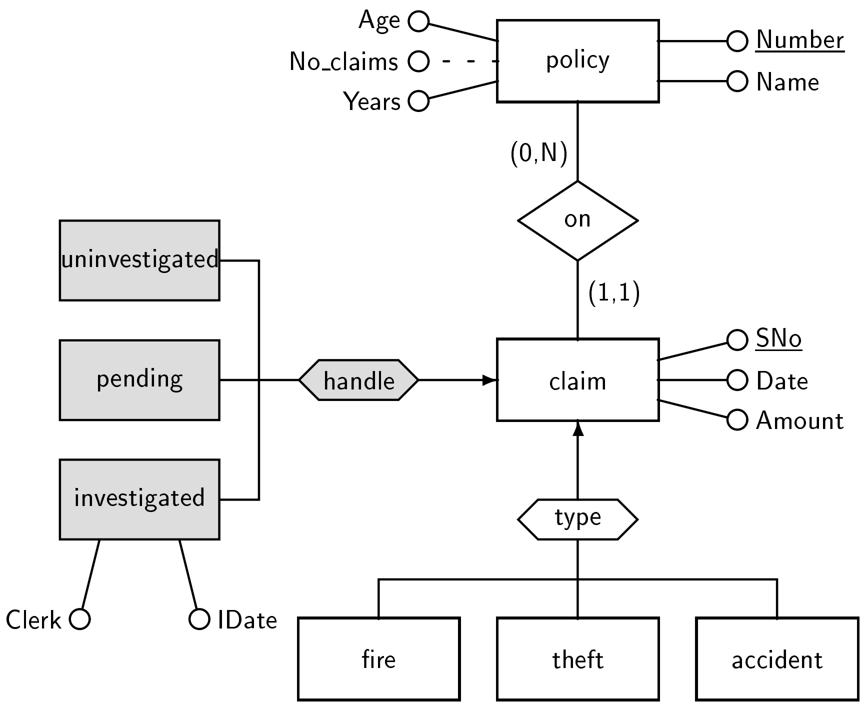


Figure 1: ER model for insurance claim records

model as predicates. From the basic semantics of the ER model in Table 1, we can regard any model as having a **canonical representation** made up of the predicates $E(e)$, $E\_A(e,a)$, and $E_1\_R\_E_2(e_1,e_2)$ mentioned in Table 1, where we can use the key attribute of entities to act as the object identifier e. Thus the model in Figure 3 has the canonical model of Example 4.1. (If we use the key attribute as the object identifier, no predicate is required for the key attribute, since for the key attribute K the dependency $E(e) \equiv E\_K(e,e)$ holds.)

Example 4.1 Canonical representation of claims ER model

policy(Number)	policy_name(Number,Name)	policy_age(Number,Age)
	policy_no_claims(Number,No_claims)	policy_years(Number,Years)
policy_on_claim(Number,SNo)		
claim(SNo)	claim_date(SNo,Date)	claim_amount(SNo,Amount)
fire(SNo)		
theft(SNo)		
accident(SNo)		
uninvestigated(SNo)		
pending(SNo)		
investigated(SNo)	investigated_clerk(SNo,Clerk)	investigated_idate(SNo,IDate)

□

4.3 Dependency Rules for ER Models

¿From the description of an ER model in the table, it may be realised that there are (using the usual database terminology) **inclusion dependencies** [?] or **exclusion dependencies** [?] between objects in ER models — instances of one type of object may only exist if there are corresponding instances in another type of object. In particular:

Dep 1 trivially, members of any entity, attribute or relationship set X are dependent on themselves, $X(x) \rightarrow X(x)$ and $X(x_1,x_2) \rightarrow X(x_1,x_2)$. We only include this dependency to make the later presentation in Section 5.2 uniform.

Dep 2 members of relationship sets R are dependent on the existence of members of the entity sets E_1, E_2 they connect, such that $\forall e_1, e_2 \,.\, E_1\_R\_E_2(e_1,e_2) \rightarrow E_1(e_1) \wedge E_2(e_2)$

Dep 3 members of attribute sets are dependent on the existence of instances of the relationship or entity to which they are attached, such that $\forall e, a \,.\, E\_A(e,a) \rightarrow E(e)$

Dep 4 members of a *child* entity set E_i of a generalisation are dependent on the non-existence of members of the other child entity sets E_j, such that $\forall e, i \,.\, E_i(e) \rightarrow \neg\exists j \,.\, j \neq i \wedge E_j(e)$

Dep 5 members of a child entity set E_s in a subset or generalisation are dependent on members of the *parent* entity set E, such that $\forall e \,.\, E_s(e) \rightarrow E(e)$

In any implementation of the ER model, we should aim to ensure that these dependencies are maintained by updates to the model.

4.4 Maintaining Dependency Rules of ER Models

We introduce a classification of the method used for maintaining dependencies in the ER model. The choices made in this classification is of importance in that process outlined in the first choice of amagamating predicates together causing the dependency rules above to have wider scope. Hence later when the depedency rules are associated to the deductive and default reasoning, the choice effects that reasoning process.

- *Maintenance by structure* — the implementation of the ER model is so constructed that it is impossible to violate the inclusion dependency. This involves restructuring the canonical representation in the following ways:
  - Attribute sets which are associated to entity sets by a $(x,1)$ cardinality constraint can be stored in the relation which holds the entity, providing they are of the same type (EDB, IDB or DDB). Thus we can implement the EDB policy entity set in Figure 1 as the relation below, with all the EDB attributes stored in one relation:

    policy(<u>Number</u>,Name,Years,Age)

    Note that we can note include the No_claims attribute, since it is an IDB object.
  - Relationship sets which are associated to an entity set by a $(x,1)$ cardinality constraint may be stored with the relation implementing the entity set, providing they are the same type of object. Thus we may implement the on relationship set as an attribute of the claim relation, together with the EDB attributes, giving:

    claim(<u>SNo</u>,Number,Date,Amount)
  - Child entities of a generalisation may be stored with the parent entity. Thus we may implement the fire, theft and accident entity sets as a Type attribute classifying claim into subsets:

    claim(<u>SNo</u>,Number,Date,Amount,Type)
- *Maintenance by constraint rule* — the implementation of the ER model includes a constraint rule to prevent violation of the inclusion/exclusion dependency. Many commercial DBMS implementations, such as Sybase 10 [?], now directly support the specification of inclusion dependencies. This allows us to implement each ER object as a separate relation, and specify the inclusion/exclusion dependency implied by the model directly as a rule. Although this method has a certain elegance, it is also highly inefficient. Thus if used at all, it is normally only used to cover the situations which can not be handled by the maintenance-by-structure approach. The most common such areas are (1) many-many relationships, (2) attributes/relationships/child entities which are of a different object type from the entities they are related to. In the second case, the dependent objects will be defined by rules, and the constraint can form part of the rule. Refering to Figure 3 and Example 4.2, when we write rules for defining the IDB object policy_no_claims the rule body includes policy, and rules defining the EDB object investigation_of_investigated the rule body includes investigation and investigated.

Adopting the above approaches to maintaining the dependency rules (*i.e.* avoiding the use of constraint rules as far as possible) results in a restructuring of the canonical model of Example 4.1 to an optimised model shown in Example 4.2, which will be the basis of all further examples. However, our approach does not assume any particular optimisation of the canonical model is used, and any combination of the above techniques can be used. Equally, the original canonical model could have been used.

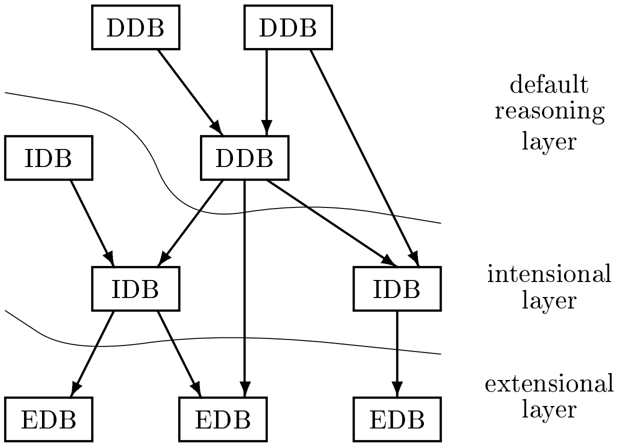


Figure 2: Stratification of rules

Example 4.2 Optimised representation of claims ER model

policy(Number,Name,Age,Years) policy_no_claims(Number,No_claims)
claim(SNo,Number,Date,Amount,Type)
uninvestigated(SNo)
pending(SNo)
investigated(SNo)
investigation_of_investigated(SNo,ISNo)
investigation(ISno,Clerk,IDate)

□

4.5 Using ER Modelling for Default Logic Rules

We have already stated that since we are building upon Datalog [?] in the way we use default logic in database systems. In the context of Datalog, this implies that any ER object is only implemented as either an EDB relation in the database, or a set of IDB deductive rules. Since the rules may use relations in their definitions as well as other rules, there is a type of stratification where IDB objects are defined 'on top' of EDB objects. Extending Datalog to default logic extends this with an additional layer for DDB rules, where DDB objects are defined in terms of other DDB objects, as well as IDB and EDB objects. This stratification is illustrated in a conceptual manner in Figure 2, where the arrows point from rules which use certain objects in there bodies to the rules/relations which define those objects.

Thus any IDB (DDB) object must have at least one classical (default) logic rule associated to it, and EDB objects must have no rules associated with them. Since default logic is an extension of classical logic, it is possible to specify both classical and default rules for an DDB object. If the body of a rule refers to a DDB object then the rule is DDB, if it only refers to IDB or EDB objects then it must be at least IDB if not DDB.

The dependency rules inherent in an ER model lead us to identify two further restrictions. If an inclusion dependency $E_1(e) \rightarrow E_2(e)$ exists, then E_2 being IDB or DDB requires E_1 to be at least IDB. The consequence of this not being the case would be that the existence of a subset of an IDB or DDB entity E_2 would be defined as EDB, breaking the Datalog model we are extending. Similarly, an exclusion dependency $E_1(e) \rightarrow \neg E_2(e)$ where E_2 is IDB or DDB forces E_1 to be at least IDB.

Should we be stricter, and say the if $A \rightarrow B$ then if B is DDB then A must also be, so that all of the extent of B is subject to default reasoning?

The following list gives a methodology to follow in the initial determination of the type of each ER object; the purpose is to direct attention to the area of the model that might require classical or default reasoning to be employed, and hence rules to be captured.

1. Identify as EDB those non-child which are defined in the domain by keeping explicit records which can be regarded as facts about the domain.
2. For each non-child derived as a result of the existence of other objects in the domain, identify it as DDB if there is some ambiguity about the method by which it is derived, and identify it as IDB otherwise.
3. For attributes, relationships and non-child entities, repeat a similar analysis to steps 1 and 2, but also use the following guidelines:
   - attributes must be at least IDB if the entity to which they are attached is IDB (DDB).
   - relationships must be at least IDB if either of the entities related is IDB (DDB).
   - child entities must be at least IDB if the parent entity is IDB (DDB), and the child entities of a particular generalisation will all be of the same type.

For our example ER model in Figure 1, we may consider that handle and its associated entity classes investigated, pending and uninvestigated are in the domain based on default reasoning when analysis has resulted in a number of rules with no clear hierarchy. We present these rules in the next subsection. Since investigated is a DDB object, so should all its attributes be DDB or IDB. However our initial modelling has them as EDB. Either there are rules to be found to define these attributes, or there model must be changed so that they may be asserted in the database as relations. Figure 3 shows one possible remodelling which allows us to maintain clerk and date as EDB objects, attached to a new entity class investigation. We will take this to be the model used for the remainder of this paper.

5 Rule Engineering for Default Databases

The purpose of rule engineering is to develop the default rules for the default database. The first part of this process is to capture general rules as *"if ... then"* rules. The second part is identify to identify pairwise conflicts between the general rules. The third part is to identify a preference ordering over these rules, so that if two rules have conflicting consequences, we know which rule takes priority. The fourth part is to translate these general rules, with associated preference ordering, into default rules (i.e. rules of default logic). The four parts are covered in the next four subsections respectively. We have chosen to use general rules as a first step in rule engineering because they are a common and natural way for participants in a systems engineering exercise to specify default information.

5.1 Capturing General Rules

As we described in the introduction, a **general rule** is a rule that is usually valid, but since it has exceptions, sometimes it is invalid. We assume the following language for general rules:

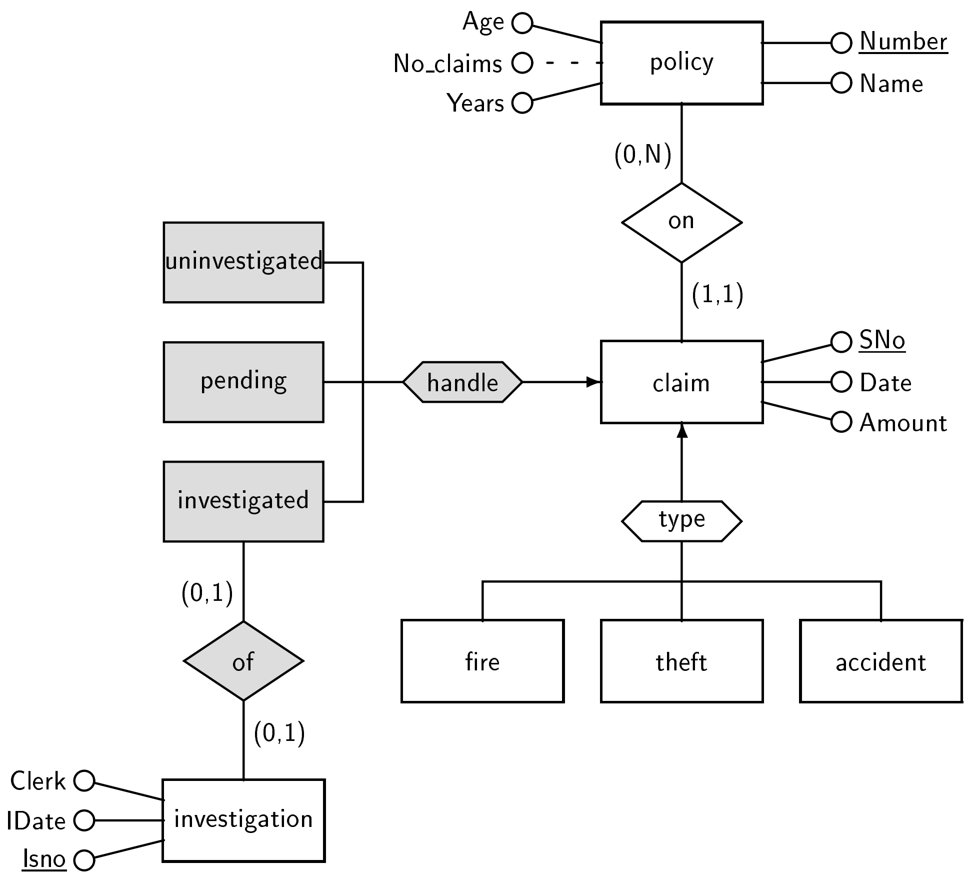


Figure 3: Restructuring of investigated entity to use default rules

$$\alpha \text{ :- } \beta_1, \ldots, \beta_n$$

where α is a literal, and $\beta_1, \ldots, \beta_n$ is a conjunction of literals. In addition, each general rule is implicitly universally quantified. We denote a set of general rules generated for an application by the symbol G. General rules are a subset of classical logic. The proof theory and semantics of classical logic therefore apply to each general rule.

We will assume that in developing a database, statements about the domain are captured in natural language, and that these are then reformulated as general rules. These are familiar procedures from respectively requirements analysis and Datalog/Prolog programming. The special feature about general rules is the criterion for positing a general rule. Each general rule must be a useful rule in some context, so if the antecedent holds, then the consequent is usually correct. However using this criterion, a set of general rules may conflict in some contexts. We define conflict formally in the next section. But, when is arises, it means that the general rules involved have antecedents that are under-specified. That brings us to the resolution techniques, based on preference ordering, that we discuss in Section 5.3.

To illustrate, Example 5.1 gives a set of general rules for our case study, obtained by a normal analysis and design process. We will use these twelve rules as a running example in the rest of this section, denoting each as R_1, R_2, *etc.*

Example 5.1 General rules for claims example

1. *We do not regard claims for fire damage suspicious unless there has been another claim for fire damage from the same policy holder.*

```
uninvestigated(SNo) :-
    claim(SNo,P,Date,_,fire),
    \+ ( claim(SNo2,P,_,_,fire),SNo2 ⟨⟩ SNo ).
```

2. *Small claims are rarely investigated.*

```
uninvestigated(SNo) :-
    claim(SNo,_,_,Amount,_),
    Amount = ⟨ 100.
```

3. *If there have been three claims previously, we usually investigate all further claims.*

```
investigated(SNo) :-
    claim(SNo,P,Date,_,_),
    policy_no_claims(P,C),
    C ⟩ 3.
```

4. *A policy holder who has been with the company for ten or more years is regarded as more reliable, and we do not investigate provided that the number of claims does not exceed six.*

```
uninvestigated(SNo) :-
    claim(SNo,P,Date,_,_),
    policy(P,_,_,Years),
    Years ≥ 10,
    policy_no_claims(P,C),
    C = ⟨ 6.
```

5. *An accident by an old person will be investigated to determine if the policy holder is still safe to be driving.*

```
investigated(SNo) :-
    claim(SNo,P,_,_,accident),
    policy(P,_,Age,_),
    Age ⟩= 70.
```

6. *Young people tend to be untruthful about claims for theft from their cars.*

```
investigated(SNo) :-
    claim(SNo,P,_,_,theft),
    policy(P,_,Age,_),
    Age < 30.
```

7. *A policy holder for three or more years is trusted for their first claim, provided it is less that £500.*

```
uninvestigated(SNo) :-
    claim(SNo,P,Date,Amount,_),
    Amount ⟨ 500,
    policy(P,_,_,Years),
    Years ⟩= 3,
    policy_no_claims(P,1),
```

8. *Always investigate large claims.*

```
investigated(SNo) :-
    claim(SNo,_,_,Amount,_),
    Amount ⟩= 10000.
```

9. *A claim is obviously categorised as investigated once an investigation is ordered by the department manager.*

```
investigated(SNo) :-
    investigation_of_investigated(SNo,_).
```

10. *We denote as* pending *any claim which can not be decided upon.*

```
pending(SNo).
```

11. *The* no_claims *attribute of* policy *is defined to be the number of claims the policy has had over its lifetime.*

```
policy_no_claims(Number,No_claims) :-
    policy(Number,_,_,_),
    countof( (claims(_,Number,_,_,_) ,No_claims).
```

12. *We assume a relationship between any* investigation *and an* investigated *claim with the same number.*

```
investigation_of_investigated(SNo,SNo) :-
    investigated(SNo),
    investigation(SNo,_).
```

□

5.2 Identifying Conflicts Between General Rules

As we discussed in the previous section, individual general rules are posited if they are useful in some contexts. However, in some contexts there may be **conflicts** between general rules — that is rules may have contradictory consequences for some extensions of the data. Most obviously these conflicts may be between rules with the same predicate in their heads, where one of the rules has the positive consequence for the predicate, and the other a negative consequence. More generally, conflicts can exist between rules $r_1, r_2 \in G$ with predicates p_1, p_2 in their heads, where there is some inclusion or exclusion dependency between p_1 and p_2. Formalising a conflict as the irreflexive symmetric relation conflict(r_1, r_2), and denoting $\rightarrow^+$ to be transitive closure of the dependency rules $\rightarrow$ listed in Section 4.3, we can define conflict by:

$$\begin{array}{l}
\text{conflict}(r_1, r_2) \text{ if } r_1 = (p_1 \text{ :- } \ldots) \wedge r_2 = (\backslash + \ p_2 \text{ :- } \ldots) \wedge p_1 \rightarrow^+ p_2 \\
\text{conflict}(r_1, r_2) \text{ if } r_1 = (p_1 \text{ :- } \ldots) \wedge r_2 = (p_2 \text{ :- } \ldots) \wedge p_1 \rightarrow^+ \neg p_2 \\
\text{conflict}(r_1, r_2) \text{ if conflict}(r_2, r_1)
\end{array}$$

Note that the conflict relation need only be binary since the dependency rules are binary, and we are in effect generating a type of closure on the dependency rules where conflict(r_1, r_2) holds if the syntax of r_1, r_2 makes it possible that for some extension of the database there exists a conflict. The following list illustrates how conflict arises by considering the five rules Dep 1–5 in turn; by the nature of the ER model we use, the only situation where a non-trivial closure of dependency rules arises (*i.e.* one involving more than one rule) is for generalisation hierarchies, which involve the use of Dep 4 and 5.

Case 1 A pair of rules conflict where one has a positive consequent for a predicate which the other rule has a negative consequent.

e.g. A pair of rules policy_no_claims(_,_) :-... and \+ policy_no_claims(_,_) :-... would be said to conflict.

Case 2 Any predicate defining an ER relationship will conflict with any predicate that conflicts with the predicate defining the entity of the relationship.

e.g. A pair of rules \+ policy(_,_,_,_) :-... and claim(_,_,_,_,_) :-... would be said to conflict, since the second attribute of claim depends on the first attribute of policy.

Case 3 Any predicate defining an ER attribute will conflict with any predicate that conflicts with the predicate defining the entity of the attribute.

e.g. Consider the attribute No_claims which can be represented in the relation policy_no_claims. Here a pair of rules policy_no_claims(_,_) :-... and ¬ policy(_,_,_,_) :-... would be said to conflict.

Case 4 A conflict occurs if one rule of the pair has a positive consequent for a predicate defining an entity, and the other rule of the pair has a positive consequent for a predicate defining a sibling entity in the generalisation hierarchy.

e.g. A pair of rules investigated(_) :-... and uninvestigated(_) :-... would be said to conflict

Case 5 A conflict occurs if one rule of the pair has a negative consequent for a predicate which defines an entity, and the other rule of the pair has a positive consequent for a predicate defines a second entity which is a subset of the first entity.

e.g. A pair of rules investigated(_) :-... and ¬ claim(_,_,_,_,_) :-... would be said to conflict

The notion of conflict is open-ended — if the modelling language were enhanced to add additional dependency rules, then they would be handled in the same manner. Note that our definition of conflict is purely a syntactic one, and makes no allowance for semantic analysis of rules. For example, if might be that in Case 4 above, the rules are defined as investigated(SNo) :-fire(SNo), ... and ¬ claim(SNo,_,_,_,_) :-accident(SNo), ..., in which case they clearly will never conflict for any extension of the database. Whilst the use of theorem provers may be able to detect some such cases, in general it is undecidable to determine if conflict(r_1, r_2) can hold for certain extensions of the database where syntactically it might. Thus for the purposes of this presentation we will ignore any restriction on the conflict relation that may result from theorem provers, whilst admitting that in practice this may be a useful addition to the methodology. Example **??** illustrates the conflict relation for the running example. Remember that symmetry holds for the conflict relation.

Example 5.2 Conflicts pairs for rules in Example 5.1

conflict(R_{10}, R_9)	conflict(R_{10}, R_8)	conflict(R_{10}, R_7)	conflict(R_{10}, R_6)
conflict(R_{10}, R_5)	conflict(R_{10}, R_4)	conflict(R_{10}, R_3)	conflict(R_{10}, R_1)
conflict(R_{10}, R_1)	conflict(R_9, R_7)	conflict(R_9, R_4)	conflict(R_9, R_2)
conflict(R_9, R_1)	conflict(R_8, R_7)	conflict(R_8, R_4)	conflict(R_8, R_2)
conflict(R_8, R_1)	conflict(R_7, R_6)	conflict(R_7, R_5)	conflict(R_7, R_3)
conflict(R_6, R_4)	conflict(R_5, R_4)	conflict(R_4, R_3)	conflict(R_3, R_2)

□

Our definition of a conflict relation now allows us to formally define the distinction between IDB and DDB rules (and hence the objects associated to them). Any object with no rules defining instances of that object is defined as EDB. For those objects with rules associated, the object and associated rules are IDB iff they are not involved in the conflict relation, and the bodies of the rules contain only IDB and EDB objects. Otherwise the object and rules are DDB. For our running example, the objects in Example 4.2 are classified in Example 5.3.

Example 5.3 Classification of objects in the claims example

EDB	IDB	DDB
policy(_,_,_,_)	policy_no_claims(_,_)	uninvestigated(_)
claim(_,_,_,_,_)		pending(_)
investigation(_,_,_)		investigated(_)
		investigation_of_investigated(_,_)

□

5.3 Identifying a Preference Ordering over General Rules

Consider two conflicting rules R_i and R_j defined as H_i:-B_i and H_i:-B_i. This is problematical since we may have a logical inconsistency (i.e. $H_i \wedge H_j$ is a contradiction) or we may have a violation of the ER model (due to the dependency rules making $H_i \wedge H_j$ a contradiction). Of course default reasoning is designed to handle such conflicts, but the blind use of default reasoning on a set of general rules would be incorrect, since general rules have been formulated without regard to the interplay between rules. A solution to this problem is to adopt a preference ordering over the general rules, so that when a pair of conflicting rules both have an antecedent satisfied, the more preferred rule is used, and the less preferred rule is ignored. In this way, we are using the preference ordering to restrict the contexts in which a general rule may be applied. We use this preference information in constructing the default rules from the general rules.

A preference ordering is represented by a directed acyclic graph (DAG) where each node has one or more associated rules, and each arc between pairs of nodes indicates the relative preferences for the

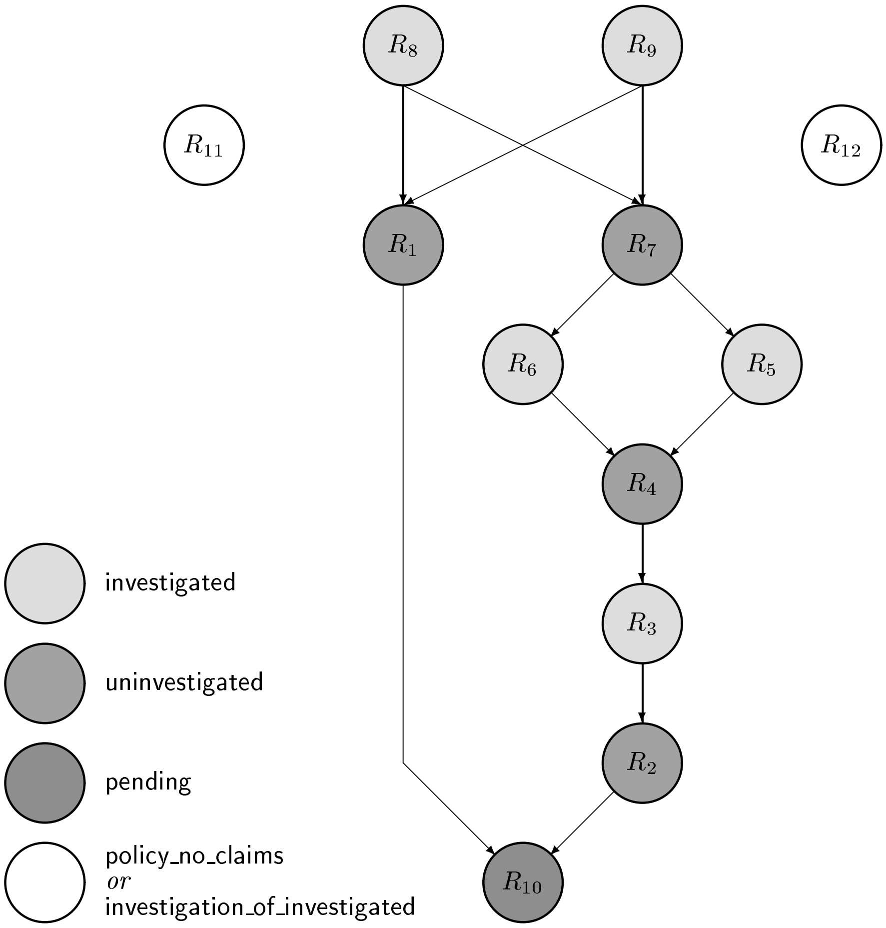


Figure 4: A Fully Resolved Rule Hierarchy DAG

rules. Building the DAG can be regarded as an important part of the knowledge elicitation process during system specification and development. Usually a general rule should only be selected to overrule another when it is properly verified with domain experts that this is the case.

To continue our running example, it might be determined that R_3 should overrule R_2, and so an arc is placed between them with the arrow pointing towards R_2 — as illustrated in Figure 4. We note that in Figure 4, all the rules that conflict are examples of Case 4 — that is rules between child entities in one generalisation hierarchy. However, from the point of view of the methodology this is immaterial, since we are simply interested in resolving instances of the conflict relation.

We summarize this in the following definition for $(\mu, X, <)$, where G is a set of general rules, $(X, <)$ is a partially ordered set, and μ is a mapping from G to X.

1. For all $r \in G, \mu(r) \in X$.
2. For all $x \in X$, there is an $r \in G$ such that $\mu(r) = x$.
3. For all $r, r' \in G$ such that conflict(r, r') holds, and the antecedents of r and r' are not mutually exclusive (in other words, the conjunction of the antecedents for r and r' is classically consistent, thus the rules could hold at the same time), then $\mu(r) > \mu(r')$ iff r overrules r'.

The DAG is **fully resolved** with respect to a general rule set iff for all conflict(r, r') then either $\mu(r) < \mu(r')$ or $\mu(r') < \mu(r)$ — *i.e.* there is some path between any pair on rules that appears in the conflict relation. Note the syntax of the DAG does not allow the creation of cycles in the rules. Also, the conflict resolution means that all root nodes of a fully resolved DAG will not conflict. Figure 4 shows a possible DAG constructed for our example rules R_1—R_{12}. The nodes involved in conflict are denoted by shading. Note that all root nodes (*i.e.* nodes with no incoming arcs) are either rules for investigate (which can not conflict with each other), or the two rules R_{11} and R_{12} which never appear in conflict.

If we do have a conflict, and we cannot make a preference, we can amend the rule. For example, one or both of the rules may need to be decomposed into several rules, to allow for the formation of more complex hierarchies in the DAG. For example, we may break-up a rule for investigated accidents into one for accidents more than a year old, and one for accidents one year or younger. It may be then found that a conflicting uninvestigated rule is overridden by one but overrides the other.

However, for a conflict for which we cannot make a preference — that is we are only able to form at partially resolved DAG — we can leave the conflict in place, and use default reasoning. This use of defualt reasoning is entirely appropriate, since we have already discounted the situations where default reasoning would have been used 'by chance' had we not identified preferences between rules. We now must make the choice of skeptical/credulous default reasoning. A skeptical approach will lead to some situations were no decision can be reached — in our example this would leave some claims as none of investigated, uninvestigated or undecided. A credulous approach would instead lead to some situations were a claim is more than one of these at a time. In both cases there are two non-exclusive methods of handling the situation:

- Regard the violation of the ER model as a system integrity issue, causing further development work to occur to prevent the conflict reoccuring.
- Flag to the user of the system the fact that there is some doubt to the solution — if credulous reasoning is used, this can be done by returning alternative tables in response to a database query.

Thus default reasoning can be used in a natural manner to handle those aspects of the system where a problem can not be fully resolved into the deterministic choices that the fully resolved DAG implies.

5.4 Forming Default Rules from General Rules

We now consider how to form default rules from the general rules, and the associated preference ordering over them. We first consider the problem informally, via a simple example where we have the following two rules, such that the first rule overrules the second rule, and the conflict relation holds for the pair of rules H_1 :- B_1 and H_2 :- B_2. If the antecedent to the first rule holds, then we wish to derive the consequent H_1 irrespective of whether the antecedent to the second rule holds. Translating this intuition into a default rule, we use the antecedent to the first rule as the precondition, and effectively use no justification, and then have the consequent of the first rule as the consequent of the default rule. This is gives the following default rule.

$$\frac{B_1 : \top}{H_1}$$

If the antecedent to the first does not hold, and the antecedent to the second rule does hold, then we want to be able to derive the conclusion H_2. In other words, we are seeking a failure to prove B_1 — note, this is different to showing that $\neg B_1$ holds. Translating this intuition into a default rule, we use the antecedent to the second rule as the precondition, we use the negation of the antecedent to the first rule as justification, and the consequent to the second rule as the consequent to the default rule. This is gives the following default rule.

$$\frac{B_2 : \neg B_1}{H_2}$$

This states that we may derive the consequent H_2 from the pre-condition B_2 provided that we have no proof that B_1 holds.

However, if we return to our running example, we have ten general rules to consider (*i.e.* those involved in conflicts), and some preferences represented in the DAG in Figure 4. We therefore need a method more sophisticated than that suggested by the above two rule example.

Essentially, for any general rule R_i of the form H_i:-B_i, to construct its implementation as a default rule R'_i we need to consider all the rules that overrule it, as reflected by the conflict resolution and the preferences in the DAG. For a pair of rules R_i, R_j, we say R_j overrules R_i when $\text{conflict}(R_i, R_j)$ holds and R_j is more preferred to R_i in the DAG.

Let $\text{overrulers}(H_i \text{ :- } B_i)$ be the set of antecedents of general rules that overrule the general rule $H_i \text{ :- } B_i$. We form a default rule for $H_i \text{ :- } B_i$, by making B_i the pre-condition of the default rule, H_i the consequent of the default rule, and the conjunction of the elements in $\text{overrulers}(H_i \text{ :- } B_i)$ be the justification of the default rule. If $\text{overrulers}(H_i \text{ :- } B_i)$ is an empty set, then use $\top$ as the justification.

Using this principle, we now translate the general rules in our running example. We denote each rule R_i by the schema $H_i \text{ :- } B_i$. First, consider R_8 and R_9. These rules are not overridden, and hence $\text{overrulers}(H_9 \text{ :- } B_9) = \text{overrulers}(H_8 \text{ :- } B_8) = \emptyset$.

$$R'_9 = \frac{B_9 : \top}{H_9} \qquad R'_8 = \frac{B_8 : \top}{H_8}$$

Now consider R_1 and R_7. Here $\text{overrulers}(H_7 \text{ :- } B_7) = \text{overrulers}(H_1 \text{ :- } B_1) = \{B_8, B_9\}$.

$$R'_1 = \frac{B_1 : \neg B_8 \wedge \neg B_9}{H_1} \qquad R'_7 = \frac{B_7 : \neg B_8 \wedge \neg B_9}{H_7}$$

Hence, the rules overridden by these (*i.e.* R_7 and R_1) need only include a justification that R_8 and R_9 do not hold. This can be achieved by ensuring that the conditions for R_8 and R_9 fail. We can capture this by showing that the negation of these conditions is consistent with the extension we generate from the default rules.

Similarly, for R_5 and R_6 we obtain the following default rules. Note, here R_7 is the only rule that conflicts with R_5 and is more preferred than R_5, and similarly for R_6.

$$R'_6 = \frac{B_6 : \neg B_7}{H_6} \qquad R'_5 = \frac{B_5 : \neg B_7}{H_5}$$

As further examples, consider R_2 and R_{10}. For this we obtain the following default rules.

$$R'_2 = \frac{B_2 : \neg B_3 \wedge \neg B_5 \wedge \neg B_6 \wedge \neg B_8 \wedge \neg B_9}{H_2}$$

$$R'_{10} = \frac{B_{10} : \neg B_2 \wedge \neg B_3 \wedge \neg B_4 \wedge \neg B_5 \wedge \neg B_6 \wedge \neg B_7 \wedge \neg B_8 \wedge \neg B_9}{H_{10}}$$

We summarize this process of generating default rules by the following definition. Let G be a set of general rules and let $(\mu, X, \geq)$ be the associated ordering over G. We define overrulers(H_i :- B_i) to be the smallest subset of G such that for all $B_j \in$ overrulers(H_i :- B_i) the following relations hold.

$$\text{conflict}(H_i :\!\!- B_i, H_j :\!\!- B_j)$$

$$\mu(H_i :\!\!- B_i) < \mu(H_j :\!\!- B_j)$$

Now we define justification(H_i :- B_i) as follows.

if overrulers(H_i :- B_i) $= \emptyset$
then justification(H_i :- B_i) $= \top$
else overrulers(H_i :- B_i) $= \{B_1, \cdots, B_h\}$ and justification(H_i :- B_i) $= \neg(B_1 \wedge \cdots \wedge B_h)$

Finally, we define a default rule as following: for each general rule $R_i = H_i$:- $B_i \in G$, the corresponding default rule $R'_i \in D$ is given by,

$$R'_i = \frac{B_i : \text{justification}(H_i :\!\!- B_i)}{H_i}$$

where D is the smallest set of default rules that satisfies this condition.

Example 5.4 illustrates the default rules that we have generated by explicitly representing the pre-condition, justification and consequent of general rules R_5 and R_8 in Example 5.1. The rest can be instantiated similarly.

Example 5.4 Defualt rules formed from general rules

$$R'_8 = \frac{B_8 \ : \ \top}{H_8} = \frac{\begin{array}{l}\text{claim(SNo,\_,\_,Amount),} \\ \text{Amount >= 10000}\end{array} \ : \ \top}{\text{investigated(SNo)}}$$

$$R'_5 = \frac{B_5 \ : \ \neg B_7}{H_5} = \frac{\begin{array}{l}\text{Age >= 70,} \\ \text{policy(P,\_,Age),} \\ \text{accident(SNo),} \\ \text{claim(SNo,P,\_,\_)}\end{array} \ : \ \begin{array}{l}\text{\textbackslash+ claim(SNo,P,Date,Amount,\_);} \\ \text{\textbackslash+ Amount < 500;} \\ \text{\textbackslash+ policy(P,\_,\_,Years);} \\ \text{\textbackslash+ Years >= 3;} \\ \text{\textbackslash+ policy\_no\_claims(P,1)}\end{array}}{\text{investigated(SNo)}}$$

□

6 Discussion

The notion of defaults covers a diverse variety of information, including heuristics, rules of conjecture, null values in databases, closed world assumptions for databases, and some qualitative abstractions of probabilistic information. Defaults are a natural and very common form of information. There are also obvious advantages to applying the same default a number of times: There is an economy in stating (and dealing with) only a general proposition instead of stating (and dealing with) maybe thousands of instances of such a general proposition.

In this paper, we have shown how default logic can be used to extend the notion of a deductive database to give what we call a default database. We have used DQL by Cadoli *et al.* [**?**], a restriction of default rules to being quantifier-free and function-free, as the language for defaults. DQL is more expressive than DATALOG$^{\neg}$ (DATALOG with negated literals in the body of the rules). Furthermore, there are useful (as opposed to pathological) queries that can be constructed in DQL that cannot be constructed in DATALOG.

In order to exploit DQL, we have presented data modelling and rule engineering techniques for constructing default databases. The data modelling is a development of entity-relationship modelling that makes explicit the default objects in the diagram, and appropriate conditions on them. Data modelling is then used in rule engineering to indicate general rules required. We consider capturing general rules, identifying a preference ordering over general rules, and forming default rules from general rules. We believe that general rules constitute a useful intermediate step in the rule engineering process since they are a natural and common form of representing default information. So far we have only considered the impact of using default reasoning on the data model, but there will be a consequence of the use of default reasoning when modelling the dynamic (update) aspects of the domain. We leave the dynamic aspects to a future paper.

We believe in the value of formal systems, and that is why we have adopted the well-explored approach of default logic to extend deductive databases. Because default logic is an inherently sophisticated formalism, we believe using general rules as an intermediate step in the specification process is advantageous. Though we do have other options besides default logic. One is to directly use general rules together with associated ordering. This would necessitate adopting an appropriate proof theory such as that of a defeasible logic (for a review see [**?**]), and insisting that the DAGs created during are design process are fully resolved.

Finally, via the running example on car insurance claim processing, we have illustrated how default databases can be useful in information systems. We believe that there are many possible applications for using default databases within information systems. In these applications, waiting for full information on some topic is sub-optimal, and so filling-in gaps in a database using default rules can be useful.

Acknowledgements

We would like to thank to Torsten Schaub for making us aware of DQL. We are also grateful to the anonymous referees for some very helpful comments and corrections.

References

[AS93] C. Ashworth and L. Slater. *An Introduction to SSADM version 4*. International series in software engineering. McGraw-Hill, 1993.

[Bes89] P. Besnard. *An Introduction to Default Logic.* Springer, 1989.

[BL84] C. Batini and M. Lenzerini. A methodology for data schema integration in the entity relationship model. *IEEE Transactions on Software Engineering*, 10(6):650–664, November 1984.

[BQQ83] P Besnard, R Quiniou, and P Quinton. A theorem-prover for a decidable subset of default logic. In *Proceedings of the National Conference on Artificial Intelligence*, pages 27–30, 1983.

[Bre91] G Brewka. *Common-sense Reasoning.* Cambridge University Press, 1991.

[BS94] S Brüning and T Schaub. Using classical theorem-proving techniques for approximate reasoning. In B Bouchon-Meunier and R Yager, editors, *Proceedings of the Fifth International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems*, pages 493–498. IPMU, 1994.

[CEG94] M Cadoli, T Eiter, and G Gottlob. Default logic as a query language. In J Doyle, E Sandewall, and P Torasso, editors, *Proceedings of the Fourth International Conference Principles of Knowledge Representation and Reasoning*, pages 473–484. Morgan Kaufmann, 1994.

[CFP82] M.A. Casanova, R. Fagin, and C.H. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. In *1st ACM SIGACT–SIGMOD Symposium on Principles of Database Systems*, pages 171–176, 1982.

[Che76] P.P. Chen. The Entity-Relationship model — toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[CV83] M.A. Casanova and V.M.P. Vidal. Towards a sound view integration methodology. In *2nd ACM SIGACT–SIGMOD Symposium on Principles of Database Systems*, pages 36–47, 1983.

[deM78] T. deMarco. *Structured Analysis and System Specification.* Yourdon Press, 1978.

[EN94] R. Elmasri and S. Navathe. *Fundamentals of Database Systems.* The Benjamin/Cummings Publishing Company, Inc., 2nd edition, 1994.

[GHR94] D. Gabbay, C. Hogger, and J. Robinson, editors. *Handbook of Logic in Artificial Intelligence and Logic Programming: Volume 3 — Nonmonotonic Reasoning and Uncertainty Reasoning.* Oxford University Press, 1994.

[GLV96] P Geerts, E Laenens, and D Vermeir. Defeasible logics. In *Handbook of Defeasible Reasoning and Uncertainty Management.* Kluwer, 1996.

[Got92] G Gottlob. Complexity results for non-monotonic logics. *Journal of Logic and Computation*, pages 397–425, 1992.

[GS78] C. Gane and T. Sarson. *Structured Systems Analysis: Tools and Techniques.* Prentice-Hall, 1978.

[Hop93] M Hopkins. Default logic: Orderings and extensions. In *Symbolic and Qualitative Approaches to Reasoning and Uncertainty*, volume 747 of *Lecture Notes in Computer Science*, pages 174–179. Springer, 1993.

[Hun96] A Hunter. *Uncertainty in Information Systems.* McGraw-Hill, 1996.

[JK90] U. Junker and K. Konolige. Computing the extensions of autoepistemic and default logics with a truth maintenance system. In *Proceedings of the Eighth National Conference on Artifical Intelligence (AAAI'90)*, pages 278–283. MIT Press, 1990.

[LMS+91] P. Loucopoulos, P.J. McBrien, F. Schumacker, B. Theodoulidis, V. Kopanas, and B. Wangler. Integrating database technology, rule-based systems and temporal reasoning for effective software: the TEMPORA paradigm. *Journal of Information Systems*, 1(2), 1991.

[LS95] T Linke and T Schaub. Lemma handling in default logic theorem provers. In C Froidevaux and J Kohlas, editors, *Symbolic and Qualitative Approaches to Reasoning and Uncertainty*, volume 946 of *Lecture Notes in Computer Science*, pages 285–292. Springer, 1995.

[MD92] D. McGoveran and C.J. Date. *Sybase and SQL Server.* Addison Wesley, 1992.

[MT93] W. Marek and M. Truszczyński. *Nonmonotonic Logic: Context-Dependent Reasoning.* Springer-Verlag, 1993.

[Nie94] I. Niemelä. A decision method for nonmonotonic reasoning based on autoepistemic reasoning. In J Doyle, E Sandewall, and P Torasso, editors, *Proceedings of the Fourth International Conference Principles of Knowledge Representation and Reasoning*, pages 473–484. Morgan Kaufmann, 1994.

[Nie95a] I Niemelä. A decision method for nonmonotonic reasoning based on autoepistemic reasoning. *Journal of Automated Reasoning*, 14:3–42, 1995.

[Nie95b] I Niemelä. Towards efficient default reasoning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 312–318. Morgan Kaufmann Publishers, 1995.

[NR92] I. Niemelä and J. Rintanen. On the impact of stratification on the complexity of nonmonotonic reasoning. In B Nebel and W Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 627–638. Morgan Kaufmann, 1992.

[NS96] I Niemelä and P Simons. Efficient implementation of the well-founded and stable model semantics. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 289–303. The MIT Press, 1996.

[Rei80] R. Reiter. Default logic. *Artificial Intelligence*, 13:81–132, 1980.

[RS94] V. Risch and C. Schwind. Tableau-based characterization and theorem proving for default logic. *Journal of Automated Reasoning*, 13:223–242, 1994.

[SB96] T Schaub and S Brüning. Prolog technology for default reasoning. In W Wahlster, editor, *Proceedings of the European Conference on Artificial Intelligence*, pages 105–109. John Wiley, 1996.

[Sch95] T Schaub. A new methodology for query-answering in default logics via structure-oriented theorem proving. *Journal of Automated Reasoning*, 15:95–165, 1995.

[Sch97] T Schaub. *Automating reasoning from incomplete information: From semantic foundations to efficient computation.* Lecture Notes in Computer Science. Springer, 1997.

[TEM93] TEMPORA project members. The TEMPORA project manual. Technical report, BIM S.A., 1993.

[Ull88] J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988.