Logic-Based Integration of Query Answering and Knowledge Discovery

Marcelo A.T. Aragão and Alvaro A.A. Fernandes

Department of Computer Science
University of Manchester
Oxford Road, Manchester M13 9PL, UK
{m.aragao,a.fernandes}@cs.man.ac.uk

Abstract. There is currently great interest in integrating knowledge discovery research into mainstream database systems. Such an enterprise is nontrivial because knowledge discovery and database systems are rooted in different paradigms, therefore foundational work needs to be carried out and a candidate unified syntax and semantics needs to be proposed. Elsewhere we have indeed carried out such foundational work and used it to propose a unified syntax and semantics for integrating query processing and knowledge discovery. We refer to the resulting class of database systems as *combined inference database systems* (CIDS), since they are a class of logic-based databases and the integration is anchored by a view of query answering as deductive inference and of knowledge discovery as inductive inference. The most important novel capability of CIDS is that of evaluating expressions which seamlessly compose query answering and knowledge discovery steps. This gives rise to increased flexibility, usability and expressiveness in user interactions with database systems, insofar as many relevant and challenging kinds of information needs can be catered for by CIDS that would be cumbersome to cater for by gluing together existing, state-of-the-art (but, syntactically and semantically, heterogeneous) components. In this paper, we provide an overview of CIDS, then we introduce two motivating applications, we show how CIDS elegantly support such challenging application needs, and we contrast our work with other attempts at integrating knowledge discovery and databases technology.

1 Introduction

In both science and industry, data stocks have grown so large and varied as to give rise to one of the defining challenges of modern computing, viz., how to extract from such stocks the most valuable information that they support.

Areas in science in which this challenge is very well characterized include post-genomic biology, climatology, and astrophysics and particle physics, among many others. In industry, businesses are everywhere concerned with deriving from their data stocks information that is likely to give them competitive advantage, with customer relationship management being perhaps the most prominent

H. Christiansen et al. (Eds.): FQAS 2004, LNAI 3055, pp. 68–83, 2004.

example of that. In spite of the outstanding success of database technology, extracting the most valuable kinds of information to be obtained from data stocks is beyond their traditional capabilities. The realization of this fact has given impetus to research on knowledge discovery and data mining.

For the purposes of this paper, we observe that query answering (the paradigmatic database task) retrieves *information in extensional form*, i.e., as (potentially large) collections of individual, specific items of information. For example, the answer to the question "Who has participated in which project?" would enumerate (i.e., extensionally present) individual, specific people who have participated in individual, specific projects. In contrast, the goal of knowledge discovery is to retrieve *information in intensional form*, i.e., as a few, general, characteristic statements that are supported by the data. For example, the answer to the question "Who can participate in some project?" would summarize (i.e., intensionally present) in one statement (or, possibly, a few) the constraints and conditions for people (in general) to participate in projects (also in general).

The above remarks make it clear that query answering and knowledge discovery are complementary information management technologies and that it would be beneficial to integrate them. In spite of this, reconciling query answering and knowledge discovery is nontrivial, since their roots lie somewhat apart (query answering in the intersection of logic and computer science, knowledge discovery in machine learning and the problem-solving-as-search paradigm in artificial intelligence). Thus, any attempt at integration must address this foundational issue before a unified syntax and semantics for the combined task can be proposed.

In [5] such foundational work was indeed carried out, giving rise to a proposed unified syntax and semantics for integrating query processing and knowledge discovery. The resulting class of database systems is referred to as *combined inference database systems* (CIDS). Our motivation for this is twofold: CIDS are a class of logic-based databases and the integrated approach to query answering and knowledge discovery that they support is founded on a view of query answering as deductive inference and of knowledge discovery as inductive inference. CIDS evaluate expressions which seamlessly compose query answering and knowledge discovery steps by compiling them to a database calculus that is then mapped to a database algebra and, in this form, optimized, before being passed on to a concrete database engine for evaluation. Thus, CIDS hold the promise of providing users with flexible, effective and efficient means for extending the range of information management tasks that they can seamlessly carry out. We believe that the benefits of that flexibility are pervasive and we indicate this with a discussion, and examples, of CID deployability.

The remainder of the paper is structured as follows. Section 2 contributes an introduction to CIDS. It introduces technical background used to define CIDS as formal constructs and shows how the syntax and semantics of tasks that combine query answering and knowledge discovery arises from that definition. Section 3 then describes two application scenarios in which information needs that require integrated query answering and knowledge discovery steps are currently largely uncatered for. Section 4 contrasts our work with other attempts at integrating

knowledge discovery and databases technology and indicates where CIDS are novel and uniquely useful. Finally, Section 5 lists the main contributions of the results reported and indicates the directions for our future work.

2 An Overview of Combined Inference Database Systems

The main technical goal in the proposal of CIDS was to establish a unified formal foundation for both query answering and knowledge discovery. The approach we have taken to achieve that is logic-based [19]. The main motivations for this choice were the established views of query answering as a deductive task [7] and of knowledge discovery as an inductive one [10]. The nontrivial challenge that CIDS are unique in responding to is, roughly, that of combining deductive and inductive processes in a single formalism to which a database-centric operational semantics can be assigned. In this section, we review, within the prevailing space constraints, the technical development that, for reasons of space, we have only reported in detail elsewhere [5].

Logical Framework. The logical formalism underlying CIDS is *parametric Datalog* (p-Datalog, for short) [15], for increased clarity in the context of this paper, we adapt. This represents a choice of interpretation for the parameters of the logical system in [15]. p-Datalog extends Datalog [7] to contexts where absolute validity is no longer a requirement of the logical system. In [15], the presentation context is that of uncertain reasoning, but other contexts are admissible. In any case, in this paper, what our interpretation of p-Datalog provides is a principled treatment of the validity requirement when absolute validity is too stringent. Syntactically, the extensions manifest themselves in two annotations to each clause: one assigns a *validity assessment* to the clause, the other assigns to the clause a triple of functions, referred to as *validity combinators* that, taken together, determine precisely how the validity assessment of the clause is propagated in p-Datalog derivations in which the clause is used as a premiss. Semantically, the extensions are shown to lead to a logic that enjoys the same desirable metalogical properties as Datalog (e.g., a proof-theoretic, a model-theoretic and a fixpoint semantics that coincide) and of which Datalog (and hence relational databases) are a special case.

A *p-Datalog clause* is a Datalog clause annotated with a validity assessment C and with a triple of validity combinators $\langle f^{\wedge}, f^{\leftarrow}, f^{\vee} \rangle$. The validity assessment can be interpreted (e.g., as it is in [15]) as the degree of certainty associated with the clause, but in CIDS there is no (and there is no need to have any) such specific commitment. The validity combinators determine, respectively, how the validity of the literals in the body are to be combined, how the validity of the body is assigned to the clause as a whole, and how the different validities associated with the different clauses that share the same predicate name are to be combined into a single validity assessment. Thus, if $\mathsf{p(X,Y)} \leftarrow \mathsf{q(X,Z)}, \mathsf{p(Z,Y)}$ is a Datalog clause, then one could annotate it with a validity assessment and validity combinators to yield, e.g., the following p-Datalog clause: $\mathsf{p(X,Y)} \leftarrow\!\!\!\shortmid \top \vdash \mathsf{q(X,Z)}, \mathsf{p(Z,Y)}\ \langle min, min, max \rangle$. If we let $\top$ denote

the largest possible validity assessment and *min* and *max* denote the functions that return, respectively, the smallest and the largest value in a set, then, if we so annotate every clause, p-Datalog reduces to Datalog.

In CIDS, use is made of validity assessments and validity combinators to reconcile in a single logical formalism the derivation of deductive and inductive consequences in a principled manner. In particular, since p-Datalog terms, literals, clauses and clause sets are the only primitive types, closure is ensured, but for compositionality to be also ensured, there is a need to reconcile the fact that query answering, being deductive, is truth-preserving while knowledge discovery, being inductive, is not. Compositionality goes beyond closure in requiring not only that the outputs of one task are syntactically acceptable as inputs to subsequent ones but, also, that they are semantically so. CIDS rely on the clause annotations of p-Datalog to underpin its principled treatment of these semantic concerns. In particular, many specific instantiations of CIDS (including those referred to in this paper) are proved (in [5]) to preserve the classically desirable metalogical properties of p-Datalog (and hence, of Datalog).

CIDS as Deductive Database Systems. A **combined inference database** (CID) $\mathcal{D}$ is a pair $\langle IDB, EDB\rangle$ where *IDB* is a set of p-Datalog rules, referred to as the *intensional database*, and *EDB* is a set of p-Datalog ground facts, referred to as the *extensional database*. The clauses in *IDB* and in *EDB* are constrained to belong to a given language $\mathcal{L}_\mathcal{D}$, i.e., $(IDB \cup EDB) \subseteq \mathcal{L}_\mathcal{D}$. A *CID schema* defines the lexicon for $\mathcal{L}_\mathcal{D}$ and may include relation names whose extent is not defined either extensionally in *EDB* or intensionally in *IDB*. These undefined relation names are, precisely, those whose definition may be computed by inductive inference, thus characterizing the discovery of knowledge. Constraints [7] imposed on a Datalog-based deductive database are also imposed on a CID, viz., clauses must be range restricted, and predicate symbols occurring as relation names in *IDB* must not occur in *EDB*, and vice-versa. The constraints [15] imposed on p-Datalog programs are also imposed on CIDs but these are not particular relevant here (see [5] for details).

Unlike a Datalog-based deductive database, in which the inferential apparatus is fixed (and, classically, based on resolution [7]), in the technical construction of CIDS we consider their inferential apparatus separately. Thus, a CID system is a pair $\langle \mathcal{D}, \mathcal{I}_\mathcal{D}\rangle$ where $\mathcal{D}$ is a CID and $\mathcal{I}_\mathcal{D}$ is a non-empty set of inference rules. A CID inference rule takes as premisses, and infers as consequences, p-Datalog clauses in the language $\mathcal{L}_\mathcal{D}$. CID inference rules are either deductive (in which case they have facts, i.e., information in extensional form, as consequences), or inductive (in which case they have rules, i.e., information in intensional form, as consequences). In either case, it is part of the definition of a CID inference rule that it specifies precisely which validity assessment it annotates its consequences with, given the validity assessments and combinators that annotate the premisses involved in the inference.

A single underlying logic and a unified inferential apparatus, taken together, reconcile deductive inference and inductive inference in a principled manner. Thus, compositionality arises, as expected, as the characteristic property of pro-

cesses that combine CID inference rules in arbitrarily complex formal derivations over the p-Datalog clauses in a CID. More specifically, combinations of inference rules in $\mathcal{I}_{\mathcal{D}}$ can capture particular combinations of query answering and knowledge discovery, as described below.

CIDS as Classical Database Systems. In addition to the model-theoretic and fixpoint semantics that we extend and adapt from [15], an operational semantics has been defined for CIDS that is based on a database calculus and algebra such as underpin mainstream query processing engines. The benefits of this technical move is to ensure that CIDS admit of an implementation strategy in which they emerge as mainstream database systems, unlike, e.g., the many approaches to deductive databases surveyed in [19]. Recall that the combined inference tasks that a CIDS evaluates are combinations of inference rules in $\mathcal{I}_{\mathcal{D}}$. Now, every CID inference rule in $\mathcal{I}_{\mathcal{D}}$ is a comprehension expression, in the Fegaras-Maier monoid calculus [11], over p-Datalog clauses in $\mathcal{L}_{\mathcal{D}}$. It follows that every combined inference task that is expressible in a given CIDS can be compiled into a monoid calculus expression, and this, in turn, can be translated into an equivalent, optimizable, concretely implementable monoid algebra (using the results in [11] and, e.g., a physical algebra such as in [23]).

Figure 1 depicts six monoid comprehensions that are CID inference rules. For the full details of how to interpret the expressions in Figure 1, we refer the reader to [5] and, on the monoid calculus and algebra, to [11]. For reasons of space, we can only briefly describe their meaning here. Firstly, we need to introduce some technical background, as follows.

The Monoid Approach to Query Processing. In abstract algebra, a **monoid** is a triple $\langle \mathbb{T}_{\oplus}, \oplus, \mathcal{Z}_{\oplus} \rangle$ consisting of a set $\mathbb{T}_{\oplus}$ together with a binary associative operation $\oplus : \mathbb{T}_{\oplus} \times \mathbb{T}_{\oplus} \rightarrow \mathbb{T}_{\oplus}$, called the *merge* function for the monoid, and an identity element $\mathcal{Z}_{\oplus}$ for $\oplus$, called the *zero* element of the monoid. Monoids may also be commutative or idempotent. Some monoids are on scalar types and are referred to as **primitive monoids**. Examples of primitive monoids are $\langle int, +, 0 \rangle$, $\langle int, max, 0 \rangle$, $\langle boolean, \wedge, true \rangle$, and $\langle boolean, \vee, false \rangle$. Monoids on collection types are referred to as **collection monoids**. Collection monoids require an additional function $\mathcal{U}_{\oplus}$, referred to as the *unit* function, so that it is possible to construct all instances of the collection type. An example of a collection monoid is $\langle set, \cup, \{\}, \lambda x.\{x\} \rangle$, where $\mathcal{U}_{\oplus} = \lambda x.\{x\}$ is the function that given any x returns the singleton $\{x\}$. Thus, to construct the set of integers $\{2, 7, 5\}$ one could write $(\mathcal{U}_{\oplus}(2) \oplus (\mathcal{U}_{\oplus}(7) \oplus \mathcal{U}_{\oplus}(5)))$ which gives, for $\oplus = \cup$ and $\mathcal{U}_{\oplus} = \lambda x.\{x\}$, the set $\{2\} \cup (\{7\} \cup \{5\}) = \{2, 7, 5\}$. Other examples of collection monoids are bags $\langle bag, \uplus, \{\!\{\}\!\}, \lambda x.\{\!\{x\}\!\} \rangle$ and lists $\langle list, +\!\!+, [\,], \lambda x.[x] \rangle$, where a $\uplus$ denotes union without duplicate removal and $+\!\!+$ denotes list append, and $\{\!\{\}\!\}$ and $[\,]$ denote the the empty bag and the empty list, respectively. Crucially for this paper, collections (e.g., set, bags and lists) of p-Datalog clauses are also monoids. This fact underpins the closure property of CIDS.

A *monoid comprehension* over a monoid $\oplus$ is an expression of the form $\oplus\{e \parallel w_1, w_2, \ldots, w_n\}$, $n > 0$. The merge function $\oplus$ is called the *accumulator* of the comprehension and the expression e is called the *head* of the comprehension.

$$eep(IDB,EDB) \equiv$$
$$\uplus\{\ substitute((h \rightarrowtail C' \vdash), \theta)\ \|$$
$$(h \rightarrowtail C_0 \vdash l_1, \ldots, l_n) \leftarrow IDB,$$
$$(r_1 \rightarrowtail C_1 \vdash) \leftarrow EDB,$$
$$\ldots,$$
$$(r_n \rightarrowtail C_n \vdash) \leftarrow EDB,$$
$$\theta \equiv mgu([l_1, \ldots, l_n], [r_1, \ldots, r_n]),$$
$$\theta \neq \nabla,$$
$$C' \equiv f^{\leftarrow}(C_0, f^{\wedge}(C_1, \ldots, C_n))$$
$$\}$$

(a) Elementary Production [7]

$$query\_answer(Q, \langle IDB, EDB \rangle) \equiv$$
$$\Phi(eep(\{Q\}, (EDB \cup eep(IDB,EDB))))$$

(b) Bottom-Up Naive Evaluation [7]

$$decision\_rules(L, \langle O^+, O^- \rangle, S, M^+, M^-) \equiv$$
$$\cup\{\ h \rightarrowtail C' \vdash l_1, \ldots, l_n\ \|$$
$$(h \leftarrow l_1, \ldots, l_n) \leftarrow L,$$
$$X \equiv query\_answers((h \leftarrow l_1, \ldots, l_n), S),$$
$$|O^+ \cap X| \geq M^+,$$
$$|O^- \setminus X| \geq M^-,$$
$$C' \equiv \frac{|O^+ \cap X| + |O^- \setminus X|}{|O^+ \cup O^-|}$$
$$\}$$

(c) Concept Learning [22]

$$classification\_rules(L, O^*, S, M^+, M^-) \equiv$$
$$decision\_rules(L, O_1^+, (O^* \setminus O_1^+), S, M^+, M^-) \cup$$
$$decision\_rules(L, O_2^+, (O^* \setminus O_2^+), S, M^+, M^-) \cup \ \ldots \ \cup$$
$$decision\_rules(L, O_k^+, (O^* \setminus O_k^+), S, M^+, M^-)$$

(d) Classification [22]

$$taxon(L,E,S,T_0,d_{intra},d_{inter}) \equiv$$
$$max\{\ t(T_0,T_1,T_2) \rightarrowtail C' \vdash\ \|$$
$$(t(T_1,\_,\_) \rightarrowtail C_1 \vdash) \leftarrow E,$$
$$(t(T_2,\_,\_) \rightarrowtail C_2 \vdash) \leftarrow E,$$
$$T_1 \neq T_2,$$
$$\wedge\{t(\_,T_1,\_) \neq I \wedge t(\_,\_,T_1) \neq I\ \|\ I \leftarrow E\},$$
$$\wedge\{t(\_,T_2,\_) \neq I \wedge t(\_,\_,T_2) \neq I\ \|\ I \leftarrow E\},$$
$$C' \equiv \frac{1}{1+d_{inter}(T_1,T_2,S)}$$
$$\}$$

(e) Cluster Formation [13]

$$taxonomy(L,E,S,N,d_{intra},d_{inter}) \equiv$$
$$\circ\{\lambda R.taxon(L,R,S,T,d_{intra},d_{inter}) \cup R\ \|$$
$$T \leftarrow N\}(E)$$

(**f**) **Conceptual Clustering** [13]

Fig. 1. Examples of CID Inference Rules

Each term w_i is called a *qualifier*, and is either a *generator* of the form $v \leftarrow D$, where v is a *range variable* and D is a *generator domain* (i.e., an expression that denotes a collection or a nested comprehension expression); a *filter* t, where t is a predicate over a range variable in scope; or a *binding* of the form $v \equiv \alpha$, where v is a new range variable and α is an expression that defines possible values for v (i.e., binding is a notational convenience with textual replacement semantics). Monoid comprehensions have a simple procedural interpretation, in which generators give rise to nested loops and multiple filters give rise to a conjunction of predicates. For example, a monoid comprehension of the form $\oplus\{e\ \|\ v \leftarrow D, p(v)\}$ denotes a result `R` computed as follows:

```
R := Z_⊕                          //where, e.g., Z_⊕ is ∅ if ⊕ is ∪
for each v in D :
  if p(v)  then
     R := R ⊕ U_⊕(e)  //where, e.g., U_⊕(e) is {e} if ⊕ is ∪
return R
```

CID Inference Rules as Monoid Comprehensions. With these definitions in place, we can now give a brief explanation of the semantics of the CID inference rules in Figure 1. Note that each one is a monoid comprehension that specifies a particular query answering or knowledge discovery task. Firstly, as far query answering is concerned, (a) is a monoid comprehension that characterizes the inference rule known as *elementary production*, upon which the query evaluation procedure known as *bottom-up naive evaluation*, in (b), can be defined. Secondly, as far as knowledge discovery is concerned, (d) builds upon (c) (and, respectively, (f) upon (e)) to characterize *classification* (respectively, *hierarchical conceptual clustering*) as an inferential task.

The inference rule denoted by *eep*, in (a), exploits the close structural similarity between the procedural definition of elementary production [7] and the procedural semantics of a monoid comprehension. So, the generators denote scans of *IDB* and *EDB* that yield candidate premisses. These are filtered to ensure that they have a most general unifier (with $\triangledown$ denoting that the *mgu* function failed to find one). Those premisses that satisfy the filter justify the ejection of the corresponding consequence, which is constructed by applying the substitution θ to the head of the first premiss. The only significant difference lies in the use of validity combinators to compute the validity assessment that annotates the consequence from those asserted to the premisses (in the last line in *eep*). The use of the bag monoid ($\uplus$) is necessary for the sound application of disjunctive combinators [15] because they are defined over bags of validity assessments. Then, the inference rule denoted by *query_answer*, in (b), simply applies *eep* taking premisses from a singleton whose element is the query (represented as a clause) and from the set of all *eep*-inferrable consequences of $IDB \cup EDB$. The monoid comprehension Φ in (b) (see [5]) enforces the constraint embodied in the disjunctive combinators of [15].

The inference rule denoted by *decision_rules*, in (c), begins by choosing a candidate rule from the language L. A language L is an enumerable set of Datalog clauses, which is assumed given and which can be specified by alternative, compact formalisms, e.g., one based on definite clause grammars (DCGs), such as [9]. Then, the candidate rule, interpreted as query, is posed, using the *query_answers* comprehension defined in (b), against the background data and knowledge S. The result is the set X of *eep*-inferrable consequences generated by augmenting S with the candidate rule. A candidate rule r is considered to have matched the requirements for ejection if it results in a number $|O^+ \cap X|$ of true positives and a number $|O^- \setminus X|$ of true negatives that are above the stipulated thresholds M^+ and M^-. The validity assessment for the ejected consequence r is computed, in the last line of *decision_rules*, as the *success ratio* of r [13]. Now, rule-based clas-

sifiers typically discover models for one target class at a time, and this is also the approach used in CIDS. Thus, the inference rule denoted by *classification_rules*, in (d), operates by associating to each target class its corresponding partition in the given set of positive examples. The negative examples for a target class are (in this formulation) the complement of the positive ones.

The inference rules denoted by *taxon* and *taxonomy*, in (e) and (f), respectively, cooperate to specify the iterative, agglomerative induction of (a binary tree denoting) the subset relation over a set of individuals. So, the induced tree is interpreted as a taxonomy, where nonleaf nodes are taxa, and a directed edge (x, y) can be read as x *is_a_kind_of* y expressed as a fact in the extent of a database relation with schema *t(taxon_id, left_subtree, right_subtree)*. Thus, the induced model constitutes a conceptual model that can be used to assign a degree of similarity based on the attribute values of the relation from which elements were drawn to seed the agglomerative strategy. A taxon is obtained in (e) as the merge of the two closest child taxa. The chosen candidate taxon maximizes the validity assessment that is computed for it in the last line of *taxon* (i.e., it minimizes the intercluster distance between its left and right subtaxa). The two nested $\wedge$-monoid expressions ensure that a candidate parent always has children that have not been merged before. The taxonomy itself is characterized, in (f), by an outer function-composition (denoted by $\circ$) monoid comprehension that expresses the bounded $n - 1$ iteration to merge n leaf nodes.

CID Tasks as Compositions of Inference Steps. Having defined some concrete CID inference rules, we can show how to compose them and specify tasks for a CID system to perform. Note, first, that the monoid calculus allows the formation of composite comprehensions. An inner monoid comprehension can be referred to anywhere that a variable of the same monoid type might occur in the outer monoid comprehension, viz., in the head, in the domain of a generator or of a binding, and in a filter. For example, the inner *query_answers* is nested into *decision_rule* in Figure 1 (c). Comprehension nesting underpins the closure property of CID inference rules. To see that, observe the general signatures of the deductive and of the inductive CID inference rules in Figure 1. Notice that *query_answers* maps a *query* to *answers* with respect to an *intensional* and an *extensional* database. Likewise, *decision_rules*, *classification_rules* and *taxonomy* all map some *evidence* into a *model* constrained by a *language bias* and with respect to some *background data* and some *background knowledge*. These abstractions can be graphically represented, as in Figure 2, the deductive case to the left, the inductive to the right. Notice further that the type of E, A, D and O is 'set of ground CID clauses', the type of Q, I, B, L and M is 'set of CID rules'. These observations establish the typing rules for composing the left and the right subgraphs into arbitrarily complex graphs. The interpretation is that an overlap of two arrows denotes variable binding (equivalently, comprehension nesting). For example, observe that one would typically want to use answers to a query as evidence for the induction of a model. In terms of Figure 2, this would mean composing the left and right parts with the A-arrow from the left overlapping with the O-arrow from the right, denoting the binding $\mathsf{A} \equiv \mathsf{O}$. The complex

graphs that result from such compositions depict well-formed combinations of deductive and inductive inferential steps (captured as applications of CID inference rules in monoid form), which we refer to *CID tasks*. The textual form of CID tasks can be gauged by reference to Figure 1. The prototype evaluators we have built (as described in [5]) evaluate CID tasks in such textual form.

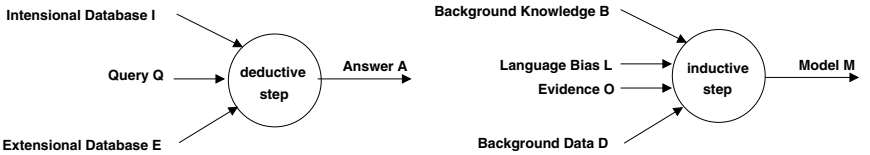

Fig. 2. Deductive and Inductive Steps as CID Task Graphs

We have stressed that CIDs display a *closure* property (as a result of the unified representation language) and a *compositionality* property (as a result of the compositionality of monoid comprehensions onto which CID tasks map). There are two further implications of the remarks above. Firstly, CIDS display a property that we refer to as *seamless evaluation*. Recall that CID tasks are arbitrary compositions of deductive and inductive tasks, nonetheless any CID task can be evaluated by a database engine based on the monoid calculus and algebra, and we are currently involved in implementing several of these, most recently one for the Grid [2]. Secondly, CIDS display a property that we refer to as *extensibility*, insofar as new deductive and inductive tasks can be performed provided that they can be expressed as described above. Once this is done, modulo the need for specific optimization strategies, the new task can be evaluated by the underlying engine without further ado.

3 The Need for Combined Inference Database Systems

The notion of a *virtual organization* (VO) [20] describes and explains the opportunistic, on-demand formation of, often short-lived, alliances between independent businesses to pounce on opportunities that would be beyond the reach of any member acting in isolation. This notion encompasses such concepts as that of a task force, a community of practice, or an advisory body, whose members are not so much independent businesses but independent experts, each in their specific domains. One diagnostic feature of VOs is that they are information businesses, both in the sense that information is often their primary raw material and primary product but, also, that asset without which they might not form at all, and, if they form, might not survive without. VOs depend on information management tools whose degree of sophistication is still only barely met by the most advanced research prototypes. Thus, it is our contention in this paper that, by their very nature, VOs are a prime source of usage scenarios in

which CIDS can be shown to make a significant difference in practice, as we describe and motivate below.

Assume that a governmental health agency responds to an epidemic by setting up a task force that brings together several experts, say, medical doctors and scientists from different hospitals, universities and industrial labs. This task force is set up as a VO. Experts become members on the strength of their track record, which, in turn, comprises data about affiliation to institutions, publication, professional recognition and past participation in similar endeavours.

The virtual nature of the task force and its need to react quickly to events pose particular challenges for information management. For example, the time and effort that can be thrown at solving a problem are at a premium because, like most VOs, the task force cannot rely on ground support and infrastructure, of the kind that institutions such as those the experts are affiliated to often have at their disposal. Thus, for one example, if a problem arises that requires the deployment of sophisticated information management techniques for which support exists only in the form of a disparate, heterogeneous collection of tools, each independently developed and supplied, the resources may not be available with which to glue them together in time. Although businesses learn to cope with this kind of problem in their day-to-day activities, the extreme conditions under which a VO operates often cause this kind of problem to become mission-critical.

As an example of a problem of this kind in the hypothetical VO we are assuming, consider how candidate replacements for an expert might be procured efficiently and effectively. Efficiency is critical here because budgets are always limited and time is of the essence in responding to an epidemic: ideally one would want to produce a shortlist with the smallest possible effort. Effectiveness is also critical here because the success of the response depends on appropriate expertise deployed on the appropriate aspect of the epidemic: ideally one would want to produce a shortlist that homes in into the fewest best candidates.

Clearly, the solution to the problem requires more than simple querying of data stocks. There must exist a model of expertise that provides the background for any inference, by querying, as to who is competent in what. Likewise, there must exist a model of similarity of expertise that can rank and compare identified experts. In most cases, the two models alluded to would most likely only exist as tacit knowledge shared by a community of practice, which would be consulted about candidate replacements. Notice, however, the extreme conditions under which a VO operates may make that infeasible.

Now, the field of knowledge discovery in databases has as one of its main aims precisely that generation of explicit knowledge from data. The question then arises as to whether the tacit knowledge alluded to might be discoverable in the data held by the VO. Since the model of expertise can be represented as a classification model and the similarity model can be represented as a taxonomy, and since there are tools for the generation of both such models, it should be possible to elicit the required knowledge. So, a solution would be the more desirable the better it were to rely on integrating query answering and knowledge discovery steps. In particular, with respect to the computing infrastructure for

solving the problem, it would ideally offer a flexible way of integrating query answering and knowledge discovery steps without resorting to extra programming. From the end-user viewpoint, the transition from performing one kind of step to performing the other should be as seamless as possible.

Most issues that prevent the identified solution to be readily deployed can be related to lack of closure, compositionality, seamless evaluation and extensibility. As we have indicated in Section 2, CIDS respond to the challenges arising from these issues with specific capabilities that arise from their very design at both the foundational and the deployment level. As a result of this, a CID task can be specified that solves the given problem as shown by the CID task graph in Figure 3, which can be interpreted as follows: (1) apply *decision_rules* (in Figure 1 (c)) to derive an intensional definition of a new relation that captures intensionally the notion of competence; then (2) apply *taxonomy* (in Figure 1 (f)) to derive an intensional definition of a new relation that models research profile similarities, taking the relation induced in (1) as background knowledge; finally, (3) apply *query_answers* (in Figure 1 (b)) to retrieve, based on the original data and the knowledge acquired in (2), a set of answers for the query 'Which candidate Y can replace which expert X, given the currently-held models of expertise and of expertise similarity?' We refer the reader to [5] for a detailed description of how the specified CID task is expressed as a monoid comprehension, then compiled into its equivalent monoid algebra, optimised, mapped into a physical algebra which an iterator-based evaluation engine then execute. In addition, [5] also prints the full execution log for the above task.

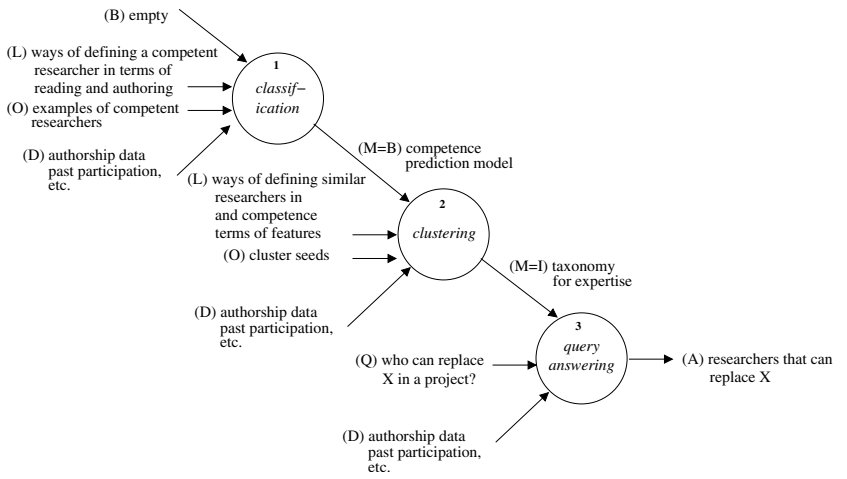

Fig. 3. Procuring Candidate Replacements for an Expert

It can therefore be seen that a CID system is a flexible platform to deploy combinations of query answering and knowledge discovery capabilities with ben-

efits that generalise over a a wide range of applications. We have also developed case studies in which CIDS were deployed to solve problems in web-service substitutivity, functional annotation of genes, and identification of malpractice in foreign currency markets. This range of applications show how CID tasks do exhibit flexibility and expressiveness.

For illustration purposes, consider one more concrete scenario. In most open economies, banks hold on to foreign currency stocks resulting from their customers' trade. This leads to an interbank market, regulated and monitored by a *bank supervising authority* (BSA), where foreign currency surpluses and deficits are traded. This market acts to promote a balance between currency supply and demand, created by imports and exports, respectively, and provides banks with profit opportunities due to fluctuations in currency prices. BSAs set limits and margins which banks must conform themselves to so that risks are managed as stipulated in international agreements. However, in their attempt to maximize profits, banks often resort to speculative practices that can lead to severe losses if the market shifts suddenly. Since, in the limit, such practices may undermine the economic stability of entire countries and linked economies, BSAs monitor the market and intervene to coerce, possibly by reviewing the regulations. However, these practices are hard to spot, because they arise by composition of several individually-legitimate transactions.

BSAs have inspectors sifting through daily-updated transactional data, so as to shortlist those banks whose pratice merit closer scrutiny. Due to the sheer volume of transactions, automated support for the inspectors is crucial. Moreover, banks roughly know (or attemp to guess) where inspection might focus, and if the intention to do so is present, so is the ability to cover signals that could be identified extensionally, i.e., by query answering alone. Thus, from the viewpoint of the inspectors, the target is a moving one: they must learn and then learn again and again what practices are being deployed that may merit intervention. In other words, in trying to keep abreast with constantly changing pratices, BSA inspectors would benefit from CIDS, since CID tasks compose knowledge discovery with query answering steps. Figure 4 is a CID task graph combining clustering, association analysis [1] and querying. The problem it is meant to solve is that of identifying violations of the assumptions that transactions between medium-sized banks with similar business profile are often similar. Violation of this assumption raises the suspicion of collusion between the transacting banks, referred to, in this case, as (often volatile) *close partners*. BSA inspectors are after sophisticated speculative practice in which volatile close partnerships are formed between several banks. Identifying such episodes provides a good target for closer scrutiny, while knowledge of legal regulation helps exclude legitimate (often stable) partnerships. In short, the ad-hoc, intuition-guided, knowledge-intensive, but time-constrained, nature of the BSA inspection brief precludes the engineering of long-lived software solutions. For this reason, the capabilities designed into CIDS seem particularly appropriate. Preliminary experiments we have conducted were successful in identifying one speculative pratice known to have been actually employed in the past.

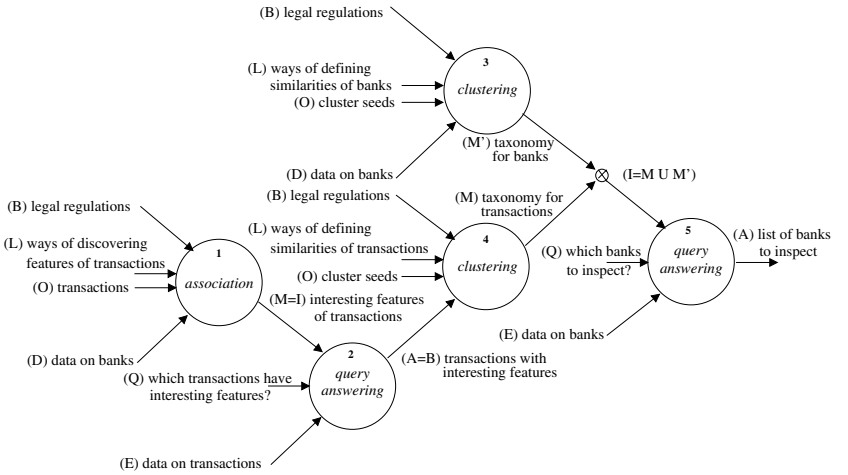

Fig. 4. Identifying Speculative Practices in Foreign Currency Markets

4 Related Work

Some extensions of SQL or some new SQL-like languages with constructs that can refer to mining algorithms have been proposed [14, 12, 18, 21]. Since these approaches involve hard-wired knowledge discovery functionalities, they deliver modest results in terms of closure, compositionality and seamlessness. Other researchers [6, 17, 16] have proposed integrating knowledge discovery capabilities into database systems. These proposals have in common the lack of a foundation which reconciles query answering and knowledge discovery of the kind that CIDS provide. Thus, the use of inductive inference in deductive databases to derive information in intensional form (first suggested in [6]) provides scant indication of how the combined functionality might be realized, and to the best of our knowledge that seminal work was not pursued further. In another approach, [17] hints at the idea of combining data and knowledge stocks into a class of information management tools called *inductive databases*, but again the few published details warrant no speculation as to how this class of systems might be formalized or realized in practice. Finally, the proposal in [16] uses version spaces to induce, from databases, strings of boolean propositions that capture simple forms of knowledge. It is, once more, difficult to be specific as to the relevant contrasts with CIDS as the published detail is not plentiful. However, it is possible to argue that, in general, the lack of a reconciliation of the inductive and deductive methods into a unified foundation has led, at the user level, all of [6, 17, 16] to somewhat ad-hoc emphases, lack of flexibility and of a clear route for incorporation into mainstream database technology.

In contrast to the above, [8] is a well-founded, Prolog-embedded, constraint-based meta-interpreter that can perform deductive and inductive inference. Unlike CIDS, for which a clear route to incorporation into mainstream database

technology already stems from their formal operational semantics, the proposal in [8] relies on top-down SLDNF resolution which, as argued, e.g., in [25], is not likely to scale sufficiently for the class of applications we have described.

Perhaps more closely related to this aspect of our contributions is our own previous work on applying combinations of distinct query answering and knowledge discovery techniques. For example, a prototype implementation of some ideas based on the inductive relations of [6] was applied as a knowledge management platform [4] and later used to characterize web-services substitutivity [3]. When compared to CIDS, the implementation that underpins the contributions reported in [4] and [3] falls short in terms of closure, and seamless evaluation. The failure to deliver on closure (and hence, on compositionality) is caused by the inability to represent syntactically the distinction between outcomes with different validity assessments (also a feature, e.g., of [8]). Instead, in [4] and [3] we relied on explicit, user-defined assimilation policies to circumvent the need for propagating outcomes with non-uniform validity assessments. In the work reported here, we have chosen to treat such questions in the underlying logic instead of delegating the task to users via the assimilation policies of [4] and [3]. In terms of seamless evaluation, the implementation used in [4] and [3] does not have a database-centric operational semantics, and in this respect resembles [8].

The capabilities that CIDS make available are not matched by commercial database and data-mining systems. Modulo user-interfacing facilities, the state-of-the-art for commercial database systems is to equate data mining algorithms to user-defined functions invokable in SQL queries. In particular, IBM Intelligent Miner allows classification and clustering models coded in XML to be deployed and applied to row sets. Microsoft SQL Server with OLE DB DM allows classification models to be specified as special kind of SQL table. The model is built by inserting tuples (interprested as training data) into it. Once built, such a model can be used in a query by a joining the corresponding table to a (normal) table through a specialized operator, referred to as a *prediction join*. In both, beyond the most trivial case, closure is restricted, and, as a result, so is compositionality. In the data-mining market, SPSS Clementine provides *application templates* that can compose data preparation, mining, and visualization steps, but *not* more than mining step. Moreover, Clementine is not a database engine, therefore evaluation is plagued by impedance mismatches, rather than seamless as in CIDS. This is because different kinds of tasks are supported by different components and the corresponding switched in execution context give rise to impedance. The prevailing levels of coupling, be it between internal components or with independent tools, are often so loose that significant engineering effort is required for interoperation to be possible.

5 Conclusions and Future Work

The main contributions of this paper are: (1) an overview of CIDS, i.e., a new class of flexible, expressive database systems, based on combining inference strategies, that can seamlessly evaluate tasks expressed involving query answer-

ing and knowledge discovery steps; and (2) the description and discussion of typical applications where combined query answering and knowledge discovery capabilities are needed and currently uncatered for, for which CIDS offer an elegant, flexible solution.

Although we have built a prototype CID system (described in some detail in [5]), much work remains to be done to render CIDS truly efficient. In the short term, we plan to investigate user-interfacing issues (e.g., a strutcured query syntax, graphical interfaces) and to identify and study in detail optimization opportunities (beyond those already described in [5]) so as to devise concrete data structures and algorithms that capitalize on that. In the medium term, we plan to deploy CIDS as a conservative extension to OGSA-DQP [2], a distributed query processor for the Grid which we have developed jointly with colleagues, and which uses the same monoid-based query processing approach that give CIDS their operational semantics. In the long term, we aim to deploy this implementation of CIDS in ongoing work in e-Science, specifically in post-genomic bioinformatics [24], in which we are involved and of which the current OGSA-DQP is already a component.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. VLDB'94*, pages 487–499. Morgan Kaufmann, 1994.
2. M. Alpdemir, A. Mukherjee, A. Gounaris, N. Paton, P. Watson, A. Fernandes, and D. Fitzgerald. OGSA-DQP: Service-based distributed querying on the grid. In *Proc. EDBT 2004*, volume 2992 of *LNCS*, pages 858–861. Springer, 2004.
3. M. A. T. Aragão and A. A. A. Fernandes. Characterizing web service substitutivity with combined deductive and inductive engines. In *Proc. ADVIS'02*, volume 2457 of *LNCS*, pages 244–254. Springer, 2002.
4. M. A. T. Aragão and A. A. A. Fernandes. Inductive-deductive databases for knowledge management. In *Proc. ECAI KM&OM'02*, pages 11–19, 2002.
5. M. A. T. Aragão and A. A. A. Fernandes. Combining query answering and knowledge discovery. Technical report, University of Machester, 2003.
6. F. Bergadano. Inductive database relations. *IEEE TKDE*, 5(6):969–971, 1993.
7. S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. 1990.
8. H. Christiansen. Automated reasoning with a constraint-based metainterpreter. *Journal of Logic Programming*, 37(1-3):213–254, 1998.
9. L. Dehaspe and L. D. Raedt. Dlab: A declarative language bias formalism. In *Proc. ISMIS'96*, volume 1079 of *LNCS*, pages 613–622, 1996.
10. S. Džeroski and N. Lavrač, editors. *Relational Data Mining*. Springer, 2001.
11. L. Fegaras and D. Maier. Optimizing object queries using an effective calculus. *ACM TODS*, 25(4):457–516, 2000.
12. J. Han, Y. Fu, W. Wang, J. Chiang, O. R. Zaïane, and K. Koperski. DBMiner: interactive mining of multiple-level knowledge in relational databases. In *Proc. SIGMOD'96*, pages 550–550, 1996. ACM Press.
13. D. J. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT, 2001.
14. T. Imielinski and A. Virmani. MSQL: A query language for database mining. *DMKD*, 3(4):373–408, 1999.

15. L. V. S. Lakshmanan and N. Shiri-Varnaamkhaasti. A parametric approach to deductive databases with uncertainty. *IEEE TKDE*, 13(4):554–570, 2001.
16. S. D. Lee and L. de Raedt. An algebra for inductive query evaluation. In *Proc. ICDM'03*, 2003.
17. H. Mannila. Inductive databases and condensed representations for data mining. In *Proc. ILP'97*, volume 13, pages 21–30. MIT Press, 1997.
18. R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In *Proc. VLDB'96*, pages 122–133, 3–6 1996. Morgan Kaufmann.
19. J. Minker. Logic and databases: A 20 year retrospective. In *Proc. LID'96*, LNCS 1154, pages 3–58, 1996.
20. A. Mowshowitz. Virtual organizations. *CACM*, 40(9):30–37, 1997.
21. A. Netz, S. Chaudhuri, U. M. Fayyad, and J. Bernhardt. Integrating data mining with sql databases: OLE DB for data mining. In *Proc. ICDE'01*, pages 379–387, 2001. IEEE Computer.
22. J. R. Quinlan. Learning decision tree classifiers. *ACM Computing Surveys*, 28(1):71–72, 1996.
23. S. Sampaio, N. W. Paton, J. Smith, and P. Watson. Validated cost models for parallel OQL query processing. In *Proc. OOIS'02*, LNCS, pages 60–75, 2002.
24. R. D. Stevens, A. J. Robinson, and C. A. Goble. myGrid: personalised bioinformatics on the information grid. *Bioinformatics*, 19(1), 2003.
25. J. D. Ullman. Bottom-up beats top-down for datalog. In *Proc. 8th ACM SIGACT-SIGMOD-SIGART PODS'89*, pages 140–149, 1989. ACM Press.