Inductive-Deductive Databases for Knowledge Management

Marcelo A. T. Aragão, 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. The growing awareness by organizations that knowledge is a key asset has led to an intense interest in tools that support knowledge management tasks. Ideally, such tools will support not only the discovery of knowledge by inductive exploration of data but also the fluent exploitation of that knowledge to produce actionable information. This paper expresses a position that a logic-based approach to the integration of knowledge discovery and deductive query answering offers significant advantages in terms of effectiveness and usability. Adopting a knowledge management perspective throughout, the paper describes an engine that exhibits integrated inductive and deductive inference capabilities and briefly considers the issues that arise in such an integration endeavour. A proof-of-concept implementation of the engine has been built and the paper uses it to suggest the potential benefits accruing from the position adopted. This is done by describing the deployment of the prototype in a classical knowledge management workflow. The paper aims to contribute an approach to logic-based knowledge management tools that have the potential for high levels of effectiveness and usability as a direct consequence of the uniformity in both the representations used and in the algorithmic treatment by means of which data, knowledge and information are made used of or derived from those representations. In this respect, the goal is to support in as fluent as possible a manner a more comprehensive set of knowledge management workflows than has hitherto been possible.

1 Introduction

The last few decades have delivered efficient, reliable and relatively inexpensive technologies for the management and exploitation of data stocks. This has helped consolidate the view of data stocks as primary assets of modern organizations. More recently, there has been a surge of interest in treating knowledge[1] too as a primary asset [6]. By definition, the main purpose of data and knowledge is to feed the processes comprising information production.

From this viewpoint, actionable information is the ultimate goal in the value-adding chain that has roots in data and knowledge stocks. Nonetheless, it is increasingly clear that, in most organizations, an ever larger proportion of the available data stocks lies largely unexploited. Correspondingly, most organizations now realize that the amount of knowledge they effectively exploit is far smaller than it needs to be if they are to remain competitive. Therefore, organizations are now more conscious than ever that valuable, actionable information might be waiting to be uncovered in the data stocks they already hold or can easily acquire.

Increases in knowledge stocks, on the other hand, are much harder to achieve, as technological support for that is not yet in place. This paper is motivated by the desire to contribute to the foundations of a class of knowledge management tools that are likely to prove particularly useful in this context.

Although database technology has delivered the means to manage and exploit data stocks and recent progress in knowledge discovery from databases justifies a certain degree of optimism, there is still no detailed proposal for an integrated platform for the combined management and exploitation of data and knowledge stocks. For instance, the link between knowledge discovery and knowledge management is very much under-explored [19]. This paper presents one such proposal, which we refer to as *inductive-deductive databases (IDDBs)*.

A proof-of-concept implementation of the algorithms and policies contributed by Section 4 has been carried out[2] and the motivating example discussed in Section 2 runs exactly as described. Note, however, that no claim is made that this prototype implementation is a contribution on its own. Rather, it has been built solely for the purpose of allowing the potential effectiveness and (in some respects) usability of IDDBs, as opposed to their potential efficiency, to be experimented with. This is precisely how the prototype is used in Section 2, viz., to provide more concrete motivation for the benefits that might accrue from the contributions of the paper.

The remainder of the paper is structured as follows. Section 2 uses a prototype implementation of the combined inference engine to present an extended example and show how the databases characterized in Section 4 constitute a step in the direction of providing suitable platforms for organizations that aim at faster rates of growth in their knowledge stocks.

[1] In this paper, by *knowledge* is meant *explicit knowledge*, i.e., knowledge that can be formally represented for use in information production processes.

[2] Available for download at http://www.cs.man.ac.uk/~aragaom/iddb-1.tar.gz

This section assumes the reader possesses basic knowledge of logic programming and deductive databases [3], including the syntactic conventions in those areas. Technical background is introduced in Section 3. Section 4 characterizes IDDBs by presenting the algorithms that model the operation of their combined inference engines. Section 5 discusses how the IDDBs defined in Section 4 can comprise more than a single inductive engine. Related work is discussed in Section 6. Finally, Section 7 points at both work that is underway and at future work that the contributions of the paper make possible while drawing a few conclusions stemming from the latter.

2 Deploying IDDBs in Knowledge Management

Data and knowledge stocks feed the processes of information production and must be refreshed, adjusted, adapted and, ultimately, increased if information stocks are to underpin future competitiveness in organizations. The simplified dataflow diagram in Figure 1 illustrates how diverse and interdependent data, knowledge and information management tasks are. One key challenge arising from the complexity illustrated in Figure 1 lies in devising platforms that can fluently support most of those tasks and foster purposeful growth and exploitation of all the stocks involved. This section exemplifies how IDDBs might be used in a range of tasks depicted in Figure 1 with levels of effectiveness and fluency that are a direct consequence of the uniformity in both the representations used and in the algorithmic treatment by means of which data, knowledge and information are made used of or derived from those representations.

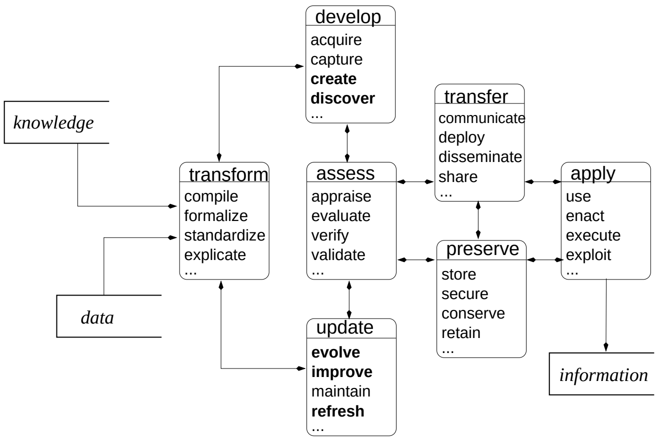

Figure 1. Information Management Activities (inspired by [11])

The authors have built a prototype IDDB to begin exploring the validity of this claim. The prototype implements the engine and policies formalized in Section 4. The remainder of this section consists of a summarized, step by step, description of how a workflow involving the exploitation and growth of data and knowledge stocks for the production of actionable information can be carried out using the prototype IDDB. Each step is annotated with reference to the tasks in Figure 1 that the step contributes to.

Step 1: [**preserve**] Suppose an organization records data about its employees (the convention is followed in the paper that metadata, e.g., schema expressions are signalled by an initial '`$` ').

```
$employee(<name>,<position>,<degree>,<graduate_from>).
$involves(<position>,<skill>).
 employee(mary, programmer, maths, oxford).
 ...
 involves(programmer, databases).
 ...
```

The fragment above shows that, for each employee, their name, position, degree, and the university from which they graduated is recorded. The skills (e.g., `databases`) that each position involves are also recorded.

Step 2: [**develop**] Assume now that the organization decides to implement a personnel policy of allocating employees to positions that minimize the need for training.

```
$suitable(<name>,<position>).
suitable(Employee,Position):-
  competent(Employee,Skill),
  involves(Position,Skill).
```

The policy can be roughly described by the rule above. It defines an employee as suitable for a position if s/he is competent in the skills involved. The rule is asserted in the intensional part of the database.

Step 3: [**develop, apply**] However, for this policy to be effective, one must be able to characterize the competences that employees possess. In particular, because the definition of how competences arise is taken to be complex and to suffer from volatility, the organization decides to characterize `competent/2` as an inducible concept (signalled by an initial '`&` '). There is a need, therefore, to induce a definition for it in terms of skills. To do so, examples are needed. So, a sample of employees is invited to reply to a survey about what skills they consider themselves to have. The results of the survey are used to augment with an additional attribute the data stocks about those employees that replied to the survey.

```
&competent(<name>,<skill>).
 competent(fiona, databases).
 ...
```

This can take the form of asserted positive examples of `competent/2` such as indicated above. Note that, if the induction algorithm cannot learn from positive data only (as is the case with mFOIL [7] used in the prototype IDDB built by the authors), a set of negatives examples can be methodically generated (as is the case in the prototype) by appealing to the closed-world assumption inherited by IDDBs from databases in general.

Step 4: [**develop, preserve, transfer**] Given the current background knowledge, the positive examples obtained from the survey and, possibly, negative examples valid under the closed-world assumption, the prototype learns the definition below for `competent/2` . The request for the definition to be learned is signalled by an initial '`?-` ' (as is a request for a query to be answered). The prototype responds with the definition of `competent/2` below.

```
?- competent(Employee, Skill).
```

```
competent(Employee,Skill):-
  employee(Employee,Position,_Degree,_Graduate),
  involves(Position,Skill).
```

Note that, of course, the prototype takes quite a simplistic view of how language bias is specified (as can be verified in Section 4). A more expressive implementation would allow more sophisticated mechanisms for specifying language biases, ideally with single-task granularity.

Note also that assuming, e.g., a policy for assimilation of inductive outcome that updates the knowledge stocks with the new definition and re-partitions appropriately the set of deducible and inducible predicates symbols, the stock of knowledge grows as a result of the system's having learned the definition above. Because of the uniform representation and algorithmic treatment, the new knowledge is fluently made available for information production.

Note, finally, that if, instead of submitting an explicit learning task as above, the user had requested the performance of the deductive task `?-suitable(Employee,Position)`, then, the definition above would be learnt on the fly, because the deductive evaluation of `suitable/2` depends on that of `competent/2`, so, if the latter lacked a definition one would be inductively derived there and then. In other words, the implemented prototype behaves lazily and suspends the deductive process to induce a definition for the required predicate and resumes the former once the latter has been obtained. In such cases, the user is warned that query answering required learning a new concept.

Step 5: [develop] Suppose now that a new-blood policy to hire recent graduates comes into force. The organization therefore announces several positions and the applications received are stored as indicated below.

```
$applicant(<name>,<degree>,<graduate_from>).
 applicant(zoe,cs,oxford).
 ...
```

Step 6: [develop] If the organization were to want to derive from the new data stocks information about the potential competences of the applicants so as to gauge to which positions they might be allocated, then the current definition of `competent/2` would not suffice. Since recent graduates are unlikely to have acquired the high-end skills that are targetted, changing the data capture is also not a solution. The organization therefore decides to refresh the current definition of `competent/2` and make it sensitive to the universities the applicants originate from.

The idea is to build upon traditional strengths of different institutions to infer that their graduates are likely to be competent in such areas as the institutions are strong in. For example, graduates in computer science from Stanford are likely to have high-end skills in database technology, and this data could be captured from a variety of sources (e.g., the Stanford CS web site) and stored as indicated below.

```
$strong_at(<graduate_from>,<degree>,<skill>).
 strong_at(stanford,cs,databases).
 ...
```

Step 7: [update, develop, apply] A new learning task can then be devised as follows. The example sets can be augmented by examples derived from the original applications of current employees. These employees and their respective skills are well-known in the organization, and an inductive task that takes those as positive instances will tend to reflect the organization's past decisions to hire.

The current definition for `competent/2` is dismissed by repartitioning the set of inducible and deducible predicates so as to include `competent/2` in the former and `suitable`, `employee`, `involves`, `strong_at`, and `applicant` in the latter. Since the goal at this point in the workflow is to refresh the characterization of `competent/2` in the light of the enhanced background knowledge about the strengths of the applicant's academic roots, it is sensible to drop the previously learned definition. Then, submitting the inductive task again causes the knowledge about competences to be refreshed as per the new definition shown below.

```
?- competent(Employee, Skill).

 competent(Employee,Skill):-
   employee(Employee,Position,_Degree,_Graduate),
   involves(Position,Skill).
 competent(Applicant,Skill):-
   strong_at(Graduate,Degree,Skill),
   applicant(Applicant,Degree,Graduate).
```

Step 8: [transfer] Finally, assuming this definition to be retained, the answer to a query such as: `?-applicant(Applicant,_,_),suitable(Applicant, Position)` can be used as actionable information with which to allocate applicants to positions while respecting the policies in place. Therefore, in the example above, every time the allocation policy has to be applied, a deductive task can provide the necessary information. Correspondingly, every time the notion of competence, upon which an allocation policy is based, has to be developed, an inductive task can exploit the current stocks of data and knowledge available in order to create or update it.

Note that, data stocks have increased in Steps 3, 5 and 6. Knowledge stocks have increased in Step 2, by explicit assertion, and in Steps 4 and 7, by automated induction. Finally, information stocks have grown throughout, as both data and knowledge stocks are handled in an integrated manner. This extended example runs exactly as described in the proof-of-concept prototype that the authors have built and made available.

This section has exemplified the broad functionality of IDDBs with reference to Figure 1 in order to provide some evidence for the claim that IDDBs hold the promise of fluent and effective data, knowledge and information management. In the next two sections, first the background to IDDBs is presented and then IDDBs themselves are characterized in some detail.

3 Background

The representation language used in this paper is **Datalog**, described in detail in [3]. Recall that Datalog is a function-free Horn-clausal language, with well-understood model- and proof-theories. Datalog can be extended with negated atoms in the body (e.g., by stratification). This paper assumes the notion of a **deductive database** [3], defined to be a set S of

Datalog clauses partitioned into an intensional part (consisting entirely of safe rules – i.e., rules in which every variable occurring in the head occurs in at least one literal in the body), called the **intensional database** and denoted by IDB_S, and an extensional part (consisting entirely of ground facts) called the **extensional database** and denoted by EDB_S. For technical reasons, in a deductive database S, the predicate symbols occurring in the head of clauses in IDB_S are not allowed to occur in the head of clauses in EDB_S. Furthermore, queries can only involve predicate symbols in IDB_S. Therefore, querying EDB_S must be done indirectly. It is customary to assume that each predicate symbol in EDB_S has a corresponding implicitly-defined select-all view in IDB_S and not to distinguish two predicate symbols that are related in this way. Given a finite set S of Datalog clauses, recall that its **Herbrand universe** HU_S is the set of individual constants in S and its **Herbrand base** HB_S is the set of atoms formed from the predicate symbols in S and the terms in HU_S, and since S is finite, so are HU_S and HB_S. If S is a deductive database, then $EDB_S \subseteq HB_S$. A **query** Q_S over a deductive database S is a rule body formed with the predicate symbols in S. Classically, an **answer** A_{Q_S} to a query over a database is a set of substitutions of variables in Q_S with elements from HU_S so as to characterize atoms in the unique **least Herbrand model** LHM_S associated with S and Q_S. In this paper, the assumption is made that the subset of LHM_S corresponding to Q_S is returned rather than the substitutions that characterize them. Henceforth, subscripts are often omitted if the context is clear. The query evaluation process over deductive databases is tractable and has been extensively studied in both a bottom-up and a top-down direction [3]. In this paper, a query evaluation algorithm over deductive databases is assumed and referred to as deduce. Any one of many such algorithms described in the literature [3, 5, 13] may serve as a denotation for deduce. Thus, the remainder of the paper assumes to be well defined an expression such as $A := \mathsf{deduce}(Q, (IDB, EDB))$ that assigns to A the answer to a query Q over a deductive database $IDB \cup EDB$.

Inductive logic programming (ILP) [17] can be seen as an approach to the provision of inductive functionality that shares the underlying foundations of deductive databases. This paper confines itself to Datalog as the representation language and adopts throughout the example setting of ILP [17]. Thus, given a set B of clauses, taken to be **background knowledge**, and sets E^+ and E^- of **positive** and **negative examples**, respectively, such that some elements in E^+, and all elements in E^-, are false in LHM_B, the goal is to find a set H of clauses, referred to as the **hypothesis**, such that all elements in E^+ are true in $LHM_{B \cup H}$ and all elements in E^- are false in $LHM_{B \cup H}$. This concept learning process has been extensively studied in both a bottom-up and a top-down direction [17, 20]. In this paper, a concept learning algorithm over deductive databases is assumed and referred to as induce. Any one of many such algorithms described in the literature [10, 16] may serve as a denotation for induce. Thus, the remainder of the paper assumes to be well defined an expression such as $H := \mathsf{induce}(L, (B, E^+, E^-))$ that assigns to H a hypothesis conforming to the language bias L that, given background knowledge B, covers all the positive examples in E^+ and none of the negative ones in E^-.

4 Inductive-Deductive Databases

This section builds upon the wealth of research into deductive databases and into ILP to characterize a class of databases (represented as sets of Datalog clauses) that exhibits deductive and inductive inference capabilities.

Given deduce and induce functions as described in Section 3, the new functionalities provided by the class of databases contributed by the paper are embodied in an information-extracting algorithm that outputs data (represented as Datalog facts) by the application of deduce and knowledge (represented as Datalog rules) by the application of induce. The main challenge involved lies in devising and implementing a strategy to partition, in the dynamic way required, the data and knowledge stocks that are input to each inferential step. The main benefit derived is the integration of deductive and inductive inference into a continuum that underpins knowledge management workflows approximating the schema in Figure 1.

Let $\Pi = (\Delta, I)$ denote a partition of a set of predicate symbols (each with a fixed arity – denoted, for a predicate symbol p, by $\mathsf{arity\_of}(p)$) into two sets. The set Δ is the set of predicate symbols defined to be available for deduction as would be classically understood in a deductive database context. One can think of it as characterizing the relation names in a database schema, and hence, as the names of concepts in the belief set that underpins deductive processes. The set I is the set of predicate symbols targetted for induction as would be classically understood in language biases of ILP tasks. Let the set I of inducible predicate symbols in $\Pi = (\Delta, I)$ be denoted by $\mathsf{inducibles\_in}(\Pi)$. Finally, given a Datalog clause C of the form $A \leftarrow B_1, \ldots, B_n$, let $\mathsf{head\_of}(C)$ denote the atom A, and given an atom A, let $\mathsf{predicate\_of}(A)$ denote the predicate symbol of A.

An **inductive-deductive database** (IDDB) is a triple $S = (\Pi, K, D)$ where K denotes a stock of knowledge in the form of an intensional database, D denotes a stock of data in the form of an extensional database, and $\Pi = (\Delta, I)$ is a pair of predicate-symbol partitions as described above. Given a list $t_1, \ldots, t_n$ of atoms where each $t_i, 1 \leq i \leq n$, denotes either a deductive or an inductive task, an algorithm can be defined over an initial IDDB $S_0 = (\Pi_0, K_0, D_0)$ that, for each task in $t_1, \ldots, t_n$, effects the transition of S_{i-1} into a new IDDB $S_i = (\Pi_i, K_i, D_i)$, $1 \leq i \leq n$ and returns an outcome O, where O is the information derived from the task.

One such algorithm is given in Figure 2. It can perform a single task, as in lines 3 to 14, or it can perform task flows sequentially, as in lines 15 to 18. In this last case, the outcomes of each task are accumulated, as in line 18. The base case is trivial, as in lines 1 to 2.

The information derived by the algorithm is data (i.e., Datalog facts) if the task is deductive, and knowledge (i.e., Datalog rules) if the task is inductive. The decision as to whether it is a deductive task or an inductive one is made in line 4, according to the predicate partition in use. Hence, a deductive task is performed from lines 6 to 9, while an inductive task is performed from lines 11 to 14.

The algorithm in Figure 2 makes use of four auxiliary functions, which are better thought of as implementing policies regarding the user's view on the epistemological status of the set of concepts captured in Π. In particular, policies must

```
perform(Tasks,(Π,K,D)) ≡
    CASE (Tasks MATCHES [])
       THEN RETURN ((Π,K,D),∅)
    ELSE (Tasks MATCHES [Task])
       IF predicate_of(Task) ∉ inducibles_in(Π)
          THEN
             (IDB,EDB) := set_deductive_basis(Π,K,D)
             Outcome := deduce(Task,(IDB,EDB))
             (Π1,K1,D1) := assimilate_deductive_outcome(Task,(Π,K,D),Outcome)
             RETURN ((Π1,K1,D1),Outcome)
          ELSE /* predicate_of(Task) ∈ inducibles_in(Π) */
              (B,E+,E−) := set_inductive_basis(Task,(Π,K,D))
             Outcome := induce(Task,(B,E+,E−))
             (Π1,K1,D1) := assimilate_inductive_outcome(Task,(Π,K,D),Outcome)
             RETURN ((Π1,K1,D1),Outcome)
    ELSE (Tasks MATCHES [First | Rest])
       ((Π1,K1,D1),O1) := perform([First],(Π,K,D))
       ((Π2,K2,D2),O2) := perform(Rest,(Π1,K1,D1))
       RETURN ((Π2,K2,D2),O1 ∪ O2)
    ESAC
```

Figure 2. An Algorithm for Performing Tasks Over Inductive-Deductive Databases

be in place to determine at each inference step, what is to count as the basis for that inference step and what is to be done with its outcome. For example, a policy is needed to decide what, if any, induced knowledge can be used in subsequent deductive inferences. The spectrum varies from total permissiveness (i.e., all induced clauses are always unconditionally automatically assimilated into knowledge stocks) to total skepticism (i.e., no induced clause is ever automatically assimilated), with – as might be expected – intermediate, and more realistic and useful, points (e.g., automatically assimilate only those induced clauses that survive evaluation hurdles regarding one or more of accuracy, relevance, significance, justifiability, understandability, novelty, etc.).

The outcome of deductive inferences can also be the subject of a policy regarding assimilation. This might model materialization of queries, for example, although one would expect the default case to be no assimilation at all. More interestingly, a policy is needed to establish the basis for a deductive step. In particular, if clauses in the intensional database are allowed to refer in their bodies to predicates that need to be learned (e.g., for refreshment, assuming that the knowledge they capture is volatile), then one needs to consider inducing these before they can be used in answering a query. This can be done lazily, on an on-demand basis, regardless of whether the deductive engine proceeds top-down or bottom-up. Alternatively, one might want to cache (or even materialize) the entire (or selected portions of the) latent knowledge stock as lemmas.

For illustration purposes, Figure 3 presents example policies. With respect to what is to count as the basis for each deductive inference step, set_deductive_basis implements a policy of total inclusiveness and maximum recency, i.e., it always automatically exploits all the latest knowledge. Thus, a new definition for each predicate currently in inducibles_in(Π) is induced and unioned to the intensional database, as in lines 3 to 8, in order to avoid that a deductive task fails or blocks because one predicate lacks definition or has an outdated one. Although this policy may be onerous, it does allow for sequence, iteration, alternation and interleaving of deductive and inductive steps, thereby providing great flexibility to users. The pseudo-code in Figure 3 is at a level of abstraction higher than that in which lazy, on-demand induction of concepts might be specified, but the prototype implementation used in Section 2 deploys exactly such an implementation strategy. With respect to what is to be done with the outcome of a deductive inference step, assimilate_deductive_outcome implements a policy of zero assimilation, i.e., irrespective of the task and its outcome, the results of deductive inferences are never automatically assimilated, as in line 1 of Figure 3. With respect to what is to count as the basis for each inductive inference step, set_inductive_basis implements a policy suitable for concept description, i.e., taking as background the stock of current data and knowledge, modulo what is targetted for induction, in line 1, as positive examples any facts that are known to be instances of the target concept, in line 2, and as negative examples, appealing to the closed-world assumption [3] that deductive databases abide by, (possibly a subset of) the facts in the Herbrand base of the concept that are not known to be instances of the concept, in line 3. With respect to what is to be done with the outcome of an inductive inference step, assimilate_inductive_outcome implements a policy of total permissiveness, i.e., irrespective of the task and its outcome, the results of inductive inferences are always automatically unconditionally assimilated, i.e., the target concept is removed if it exists, in lines 1 and 2, and the new definition is inserted, in lines 3 and 4. Note that no claim is made that any of these policies are in any way guaranteed to have desirable consequences only. Clearly, many issues are raised that demand further investigation. It is important to stress that policies are, by definition, user-specific and the granularity with which they are put in place needs to be carefully thought through in each case. Options vary from very fine (e.g., a task may be accompanied by a policy that is set or chosen to hold with scope bound to that task alone) to very coarse (e.g., policies are hardwired into the implementations). It may also be the case that policies are devised by, and put in place for, groups of, rather than individual, users. This flexibility can be explored, e.g., in incremental data mining and in deploying sophisticated knowledge assimilation techniques.

```
set_deductive_basis(Π, K, D) ≡
    EDB := { x ∈ D | predicate_of(head_of(x)) ∉ inducibles_in(Π) }
    IDB := { x ∈ K | predicate_of(head_of(x)) ∉ inducibles_in(Π) }
    FOR EACH Π IN inducibles_in(Π) /* assuming arity_of(Π) = n */
        BEGIN
            B := IDB ∪ EDB
            E+ := {x | x ∈ deduce(← π(V1,...,Vn), (K,D))}
            E− := {x ∈ HB(K ∪ D) | x ∈ HB({π(V1,...,Vn) ←}) ∧ x ∉ E+}
            IDB := IDB ∪ induce(π(V1,...,Vn),(B,E+,E−))
        END
    RETURN (IDB,EDB)

assimilate_deductive_outcome(Task,(Π,K,D),Outcome) ≡
    RETURN (Π,K,D) /* e.g., independently of Task and Outcome */

set_inductive_basis(π(V1,...,Vn),(Π,K,D)) ≡
    B := { x ∈ K ∪ D | predicate_of(head_of(x)) ∉ inducibles_in(Π) }
    E+ := {x | x ∈ deduce(← π(V1,...,Vn), (K,D))}
    E− := {x ∈ HB(K ∪ D) | x ∈ HB({π(V1,...,Vn) ←}) ∧ x ∉ E+}
    RETURN (B,E+,E−)

assimilate_inductive_outcome(π(V1,...,Vn),(Π,K,D),Outcome) ≡
    K1 := K ∪ { x ∈ Outcome | predicate_of(head_of(x)) = Π }
    D1 := D \ { x ∈ Outcome | predicate_of(head_of(x)) = Π }
    I1 := inducibles_in(Π) \ {π}
    Δ1 := (Π \ inducibles_in(Π)) ∪ {π}
    RETURN ((Δ1, I1), K1, D1) /* e.g., new knowledge persists unconditionally */
```

Figure 3. Example Policies for Exploiting and Assimilating Data and Knowledge

4.1 The IDDB Prototype

A prototype of an inductive and deductive database was implemented in Prolog. The prototype implements the algorithm in Figure 2 and, with additional performance improvements, the policies in Figure 3 on top of a service layer that enables storage and retrieval of Datalog clauses. For the deductive function, it uses the deductive database described in [5], while, for the inductive function, it uses the implementation of mFOIL described in [10].

The architecture of the prototype separates, via well-defined call interfaces, inference functions and policies, so that their replacement can be carried out non-disruptively and all other parts of the algorithms remain unchanged. Section 5 shows how this can be done.

For instance, different inductive algorithms may be used interchangeably provided that they adhere to the interface defined by the function induce.

The prototype was tested with the example in Section 2, and shown to be more effective than the separate application of SLD-resolution and mFOIL, because it enables an increase in data and knowledge stocks which are not comparably achievable otherwise. For example, a deductive task may derive answers that are only obtainable if some concept is induced on-the-fly (in which case the epistemological status of any output is clearly signalled).

The prototype qualifies as a proof-of-concept that combining deductive and inductive inferences is effective. Moreover, its development demonstrates that combining inference modes need not be more complex than the inference functions and policies already are. This is the central insight borne out by the implementation of the IDDB prototype described in this paper.

Although the primary goal is not efficiency and scalability, the prototype brought to light interesting issues. Firstly, combining deductive and inductive inference does not necessarily increase the overheads, in terms of time complexity, inherent to the inference functions. Secondly, this combination would not scale up if inputs to and outputs from the inference functions were passed by copy, thus requiring that larger amounts of data (or knowledge) be kept and copied in memory. The algorithm in Figure 2 is defined in this way, but only for clarity of exposition. The prototype addresses scalability issues more efficiently. The inference functions were wrapped in such way that its parameters and temporary results can be accessed on demand, so as to make the prototype as scalable as the inference functions it instantiates. Finally, improvements in the inference functions and policies are likely to propagate to the combined inference database engine, as illustrated above by the use of lazy evaluation to implement a sophisticated preparation policy for deduction. Based on this, it is possible to claim that the efficiency and scalability obtained for the combination of inference modes is not necessarily worse than for the implementations of the inference functions individually. Therefore, the choice of good implementations for the inference functions and policies is likely to pay off in terms of efficiency and scalability.

Another feature of the prototype is that data and knowledge are assigned an explicit epistemological status so that its use during inference is kept coherent. The prototype maintains a partition of the set of predicate symbols to control this, and it is this partition that is allowed to vary from task to task.

Further work is needed, and indeed is planned, to explore in detail both the abstract and the concrete issues arising from the ideas presented in this paper.

5 Extending IDDB Engines

An extended version of the IDDB prototype was implemented which, in addition to concept learning (based on mFOIL), also supports conceptual clustering. Support for clustering was included without affecting the existing functionalities, due to the modular architecture of the prototype.

This was done by assuming the expression $T :=$ cluster$(F,(B,C))$ to be well defined which, given background knowledge B, conforming to the features defined in F and the set of instances in C, assigns to T the conceptual clusters derived with respect to the similarity function provided by the algorithm that instantiates it. Then, the algorithm in Figure 2 was extended to output conceptual clusters (represented as Datalog rules) by applying the inference function denoted by cluster, as follows:

1. another partition is introduced where elements are the predicate symbols targetted for clustering, i.e., containing the names of relations whose instances may be clustered;
2. preparation and assimilation policies are introduced as the functions set_clustering_basis and assimilate_clustering_outcome, respectively. For example, set_clustering_basis may generate an axiomatization for the relations is_a and a_kind_of, that can be used as background knowledge, while assimilate_clustering_outcome may transform and assimilate the clustering information as a set of Datalog clauses; and
3. a clustering algorithm is selected that implements the function cluster. The implementation of COBWEB [9] used is by Joerg-Uwe Kietz, available publically in [14].

This extension represents an improvement in terms of usability for the prototype, in the sense that it supports knowledge discovery via supervised or via unsupervised learning. Thus, a broader class of knowledge management applications can be supported.

In order to evaluate the extended prototype, it was applied to one of such application, i.e., the construction and use of taxonomies. A Lotus Discovery Server [4] tutorial also uses this application to illustrate its potential as a knowledge management platform.

A taxonomy can provide organizations with a common business language and can serve as a navigational aid to finding business information. This allows its workers to drill down through abstraction levels until they find a class that describes information they need. In this example, key words and expressions (e.g., obtained from internal documents, white papers and web pages via information retrieval techniques) are represented extensionally as terms in Datalog facts that stand for the source documents. The preparation policy selects this set of Datalog facts and generates an axiomatization for the relations is_a and a_kind_of. Both are passed as inputs to the conceptual clustering algorithm (as, respectively, the set of instances and the background knowledge), while the clustering features are passed as arguments at task submission time. The assimilation policy converts the output of the clustering algorithm into Datalog rules that represent, declaratively, the structure and behaviour of the induced taxonomy. Then, the extended prototype can support, via deductive inference, browsing taxonomy hierarchies and classifying new instances accordingly.

In this context, the extended prototype was able to emulate the Lotus Discovery Server in building a working taxonomy from key expressions extracted from documents. No claim regarding accuracy or quality of the taxonomy derived is made here, because the extended prototype is based on a simpler algorithm for clustering, viz., COBWEB. Nevertheless, the extended prototype seems to provide tangible benefits in terms of fluency and effectiveness that would otherwise be unlikely to accrue from non-logical approaches.

Benefits regarding fluency are noticeable when specifying both taxonomy construction and exploitation as inferential tasks, hence, allowing them to be submitted in flexible and varied ways, as required. The prototype can support these tasks seamlessly. This may be useful for building taxonomies on demand in specific circumstances, either incrementally, so as to reduce engineering effort, or periodically, for refreshing taxonomic knowledge.

Benefits regarding effectiveness are noticeable when combining taxonomic knowledge with existing knowledge stocks because both are uniformly represented as Datalog clauses. There are also benefits in exploring taxonomic knowledge through deductive inference, since recursive, ad-hoc queries can be used to relate concepts that are not explicitly asserted in the taxonomy. These features would be hard to support with less expressive mechanisms without significant additional programming effort.

6 Related Work

The contributions of this paper are motivated by ideas stemming from the intersection of concerns such as integrating more tightly databases and data mining, providing a scalable platform for knowledge discovery in the large, extending databases with the ability to perform inductive inferences, and others. Common to these research areas is the need to integrate the querying of models of data and the induction of such models from data. If one views querying as deduction, the way is open for a logic-based approach to integrating acquisition and exploitation. This paper follows more closely the application-driven view taken in [8]. In fact, the contributions of this paper constitute a detailed and concrete, albeit preliminary, exploration of the issues raised in [8].

Note that, despite the natural interest in doing so, for reasons of space, this section is silent about initiatives in which tools are brought together but not integrated at the representation level nor at that of the core engine (e.g., a data mining tool, such as Darwin, and a database management system, such as Oracle).

A few other proposals to extend database technology with inductive capabilities have shaped, or are related to, the contributions of this paper. The first such proposal introduced the idea of *inductive database-relations*. In [1], some of the relations over a deductive database may be left undefined and uninstantiated. This is similar to the partitioning in this paper of the predicate symbols into a set of inducible and a set of deducible ones. Little consideration is given in [1] to certain issues that this paper addresses in some detail, e.g., how to adjust data and knowledge stocks in the wake of, possibly interleaved and implicit, deductive and inductive steps. For this reason, while [1] has been inspirational, the similarities of that work with the one reported here do not run deep.

Another system that bears some resemblance to the work described here is Mobal [15]. Mobal can be seen as a knowledge acquisition environment that brings together several inductive logic programming schemes into an integrated whole and provides sophisticated services, such as theory restructuring, that the engine described in this paper is silent on.. While the overall functionality delivered by Mobal is impressive, it is also fixed and closed insofar as it is hard-coded behind a user interface. Mobal is not an instance of a database system (e.g., it lacks bottom-up query evaluation and integrity constraint mechanisms) and is not flexible enough that it can be adapted to be deployed as one. In contrast, IDDBs are database systems. They are also so designed as to be easily and cleanly extended. Thus, endowing IDDBs with more than just a single induction scheme is straightforward as discussed in Section 5. It is difficult to judge the degree of effort required to extend Mobal with the database features that it currently lacks. A final point of contrast is that while this research is based on Datalog, the logic underlying MOBAL goes beyond the tractability boundaries that Datalog was carefully designed not to cross. The practical implications of this fact could be significant, insofar as, while there is no meta-logical impediment for the database engines envisaged in this report to be efficiently implemented, the opposite can be said of systems such as MOBAL that are based on more expressive logics.

Recon is a data mining system described in [18]. The architecture of Recon includes a deductive database, a rule induction component (which outputs deductive database clauses) and a visualization component. Data is stored in relational databases and SQL is used to retrieve the data used as the inductive basis for the rule induction component. The rules generated in this way can then be tested against the source database and against the knowledge stored in the deductive database. This means that rules can be refined before being assimilated into the deductive database for further use. Visualization plays an important role in refining discovered knowledge. Query results can also be materialized temporarily for improved performance. Recon's main contribution is an environment that allows an interactive discovery and refinement of knowledge before assimilation, if needed. Recon, like MOBAL, offers a fixed and closed set of functionalities that are hard coded behind a user interface, hence, Recon exhibits many of MOBAL's shortcomings. The user must intervene explicitly to move permanent data to temporary stores before knowledge discovery can be performed. Discovered knowledge is also placed in temporary stores before it can be assimilated. Therefore, it is up to users to manage the flow of data and knowledge between physical stores, and this can compromise the fluency with which they can perform complex task flows.

Seminal ideas on *inductive databases* were proposed in [2, 12]. The contributions of this paper differ from those in that they are formulated from a logical perspective in which databases are seen as sets of logical clauses, whereas the inductive databases of [2, 12] conceive of knowledge stocks as patterns, more generally, and not logical clauses, more specifically, as this paper does. Also, [2, 12] make explicit an evaluation function on acquired knowledge and endow the former with queryable status. Such an evaluation function could be easily incorporated into the engine described here and making it queryable would be a means to provide evidence upon which assimilation polices might be configured. Another crucial distinction is that while this paper uses Datalog as its representation language, [2, 12] leave the latter unspecified, with most examples taking the form of propositional association rules. While the approach described here stands upon a well-defined and well-behaved logical framework, it is not clear at this stage what foundations the *inductive databases* proposed in [2, 12] stand upon.

7 Future Work and Conclusions

Work is already underway to extend the contributions described here in two main directions. On the other hand, the authors are also finalizing a generalized framework for specifying different policies for assimilation and exploitation of inductive and deductive outcomes. This will allow IDDBs to be seen as a class of systems whose instances can be determined by particular choices of inferential capabilities on the one hand, and assimilation and exploitation policies on the other.

Future issues that will be explored include studying, the formal, as well as empirical, motivations for different assimilation and exploitation policies; exploring the benefits of a tightly-coupled approach in which the two inference engines are subsumed by a unified one; and extending the range of knowledge discovery tasks beyond the modelling stage (to include, e.g., data preparation and model evaluation).

Preliminary though it is, this paper's characterization of a database platform integrating query answering and induction is nevertheless more detailed and concrete than any other past attempt with similar aims and scope. In particular, issues regarding the dynamic epistemological status of subsets of clauses are clearly highlighted here for the first time. The paper makes it clear that studying and developing policies for the assimilation and exploitation of the outcome of both deductive and inductive tasks is likely to be a major issue, albeit one that no previous related work has frontally addressed. For example, depending on the expressiveness of the logical language used, intermediate learning steps may introduce inconsistencies that are avoided in the case of IDDBs by sticking to (potentially stratified) Datalog.

To conclude, the contributions of this paper can be summarized as follows:

1. a formal characterization of IDDBs as a class of Datalog-based logical structures that subsume deductive databases;
2. an algorithm for performing tasks over IDDBs that fluently integrates deductive and inductive inference capabilities based on logic programming;
3. a characterization of the issues arising in the context of attempts to exploit and assimilate induced knowledge in knowledge management workflows;
4. an algorithm that embodies simple policies for exploiting and assimilating data and knowledge as an example of how the broader issues raised in this respect might be tackled;
5. an extended example of how IDDBs could deliver effectiveness and usability in practical knowledge management situations.
6. a description of how IDDBs can comprise more than one inductive framework.

Acknowledgement Marcelo A. T. Aragão is on leave from the Central Bank of Brazil and gratefully acknowledges the scholarship awarded to him by the Department of Computer Science of the University of Manchester.

References

[1] Francesco Bergadano, 'Inductive Database Relations', *IEEE TKDE*, **5**(6), 969–971, (1993).

[2] Jean-François Boulicaut, Mika Klemettinen, and Heikki Mannila, 'Querying Inductive Databases: A Case Study on the MINE RULE Operator', in *Proc. 2nd PKDD*, eds., J. Zytkow and M. Quafalou, volume 1510 of *LNCS*, pp. 194–202. Springer, (1998).

[3] Stefano Ceri, Georg Gottlob, and Letizia Tanca, *Logic Programming & Databases*, Springer, 1990.

[4] Lotus Corporation. Lotus discovery server. more information available at `http://www.lotus.com/products/discserver.nsf`.

[5] Subrata Kumar Das, *Deductive Databases and Logic Programming*, Addison Wesley, 1992.

[6] Thomas H. Davenport and Laurence Prusak, *Working Knowledge*, HBS Press, 2000.

[7] Sašo Džeroski and Ivan Bratko, 'Handling noise in inductive logic programming', in *Proceeding of the 2nd International Workshop on Inductive Logic Programming*, ed., Stephen Muggleton, Tokyo, Japan, (June 1992). ICOT. TM-1182.

[8] Alvaro A. A. Fernandes, 'Combining Inductive and Deductive Inference in Knowledge Management Tasks', in *Proc. 1st Workshop on the Theory and Practice of Knowledge Management*, pp. 1109–1114. IEEE Press, (2000). In Proc. 11th Intl. Workshop DEXA.

[9] D. H. Fisher, 'Conceptual Clustering, Learning from Examples, and Inference', in *Proceedings of the 4th International Workshop on Machine Learning*, (1987).

[10] Nada Lavrač and Sašo Džeroski, *Inductive Logic Programming: Techniques and Applications*, Ellis Horwood, 1994.

[11] Ann Macintosh. Adaptive workflow to support knowledge intensive tasks. `http://www.aiai.ed.ac.uk/~alm/KAMSLIDES/index.htm`, September 1998.

[12] Heikki Mannila, 'Inductive Databases and Condensed Representations for Data Mining', in *Proc. 1997 ILPS*, ed., J. Maluszynski, volume 13, pp. 21–30. MIT Press, (1997).

[13] Jack Minker, 'Logic and databases: A 20 year retrospective', in *Proceedings of the International Workshop on Logic in Databases (LID'96)*, volume 1154 of *Lecture Notes in Computer Science*, pp. 3–57, San Miniato, Italy, (July 1996). Springer-Verlag.

[14] MLnet. Resource: Software. available at `http://www.mlnet.org/`.

[15] K. Morik, S. Wrobel, J.-U. Kietz, and W. Emde, *Knowledge Acquisition and Machine Learning - Theory, Methods, and Applications*, Academic Press, 1993.

[16] Stephen Muggleton, 'Inverse Entailment and Progol', *New Generation Computing*, **13**, 245–286, (1995).

[17] Stephen Muggleton and Luc de Raedt, 'Inductive Logic Programming: Theory and Methods', *JLP*, **19**(20), 629–679, (1994).

[18] Evangelos Simoudis, Brian Livezey, and Randy Kerber, 'Integrating inductive and deductive reasoning for data mining', in *Advances in Knowledge Discovery and Data Mining*, eds., Usama Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, 353–373, MIT Press, Cambridge, (March 1996).

[19] Eric Tsui, Brian J. Garner, and Steffen Staab, 'The role of artificial intelligence in knowledge management', *Knowledge Based Systems*, **13**(5), 235–239, (May 2000).

[20] Stefan Wrobel, 'Inductive Logic Programming', in *Principles of Knowledge Representation*, ed., G. Brewka, 151–189, CSLI Pub., (1996).