Coherent Composition of Distributed Knowledge-Bases through Abduction

Ofer Arieli, Bert Van Nuffelen, Marc Denecker, and Maurice Bruynooghe

Department of Computer Science, University of Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium

Abstract. We introduce an abductive method for coherent composition of distributed data. Our approach is based on an abductive inference procedure that is applied on a meta-theory that relates different, possibly inconsistent, input databases. Repairs of the integrated data are computed, resulting in a consistent output database that satisfies the meta-theory. Our framework is based on the $\mathcal{A}$-system, which is an abductive system that implements SLDNFA-resolution. The outcome is a robust application that, to the best of our knowledge, is more expressive (thus more general) than any other existing application for coherent data integration.

1 Introduction

In many cases complex reasoning tasks have to integrate knowledge from multiple sources. A major challenge in this context is to compose contradicting sources of information such that what is obtained would properly reflect the combination of the distributed data on one hand[1], and would still be *coherent* (in terms of consistency) on the other hand.

Coherent integration and proper representation of amalgamated data is extensively studied in the literature (see, e.g., [1, 3, 7, 13, 14, 20–23, 26, 29]). Common approaches for dealing with this task are based on techniques of belief revision [20], methods of resolving contradictions by quantitative considerations (such as "majority vote" [21]) or qualitative ones (e.g., defining priorities on different sources of information or preferring certain data over another [2, 4, 5]). Other approaches are based on rewriting rules for representing the information in a specific form [14], or use multiple-valued semantics (e.g., annotated logic programs [28, 29] and bilattice-based formalisms [12, 22]) together with non-classical refutation procedures [11, 19, 28] that allow to decode within the language itself some "meta-information" such as confidence factors, amount of belief for/against a specific assertion, etc.

Each one of the techniques mentioned above has its own limitations and/or drawbacks. For instance, in order to properly translate the underlying data to a specific form, formalisms that are based on rewriting techniques must assume

[1] This property is sometimes called *compositionality*; see, e.g., [30].

that the underlying data (or some part of it, such as the set of integrity constraints) has a specific syntactical structure. Other formalisms (e.g., that of [20]) are based on propositional languages, and so in both cases the expressiveness is limited. In some of the non-classical formalisms mentioned above (e.g., those that are based on annotated logics and several probabilistic formalisms), semantical notions interfere with the syntax. Moreover, in many of these frameworks syntactical embeddings of first-order formulae into non-classical languages are needed. Such translations may damage or bias the intuitive meaning of the original formulae. Finally, some of the approaches mentioned above are not capable of resolving contradictions unless the reasoner specifies his/her preferences. In other approaches, the mechanism of resolving contradictions is determined in advance, or is ad-hoc (thus it is oriented towards specific kinds of problems). This interference necessarily reduces the flexibility and the generality of the corresponding mediative engine.

In this paper we start from the perspective of a pure declarative representation of the composition of distributed data. This approach is based on a metatheory relating a number of different (possibly inconsistent) input databases with a consistent output database. The underlying language is that of ID-logic [9], which can be embedded in an abductive logic program. Our composing system is implemented by the abductive solver the, $\mathcal{A}$-system [18]. In the context of this work, we extended this system with an optimizing component that will allow us to compute preferred coherent solutions to restore the consistency of the database.

Our approach is related to other work on the use of abduction in the context of databases. [16] proposed to use abduction for database updating. [15, 27] developed a framework for explaining or unexplaining observations by an extended form of abduction in which arbitrary formulas may be added or formulas of the theory may be removed. In this paper, the focus is on a different application of abduction, namely composition and integrity restoration of multiple databases.

By this declarative approach we are able to overcome some of the shortcomings of the amalgamating techniques mentioned above. In particular, our system has the following capabilities:

1. *Any* first-order formula may be specified for describing the domain of discourse (as part of the integrity constraints). Thus, to the best of our knowledge, our approach is more general and expressive than any other available application for coherent data integration.
2. No syntactical embeddings of first-order formulae into different languages nor any extensions of two-valued semantics are necessary. Our approach is based on a pure generalization of classical refutation procedures.
3. The way of keeping the data coherent is encapsulated in the component that integrates the data. This means, in particular, that no reasoner's input nor any other external policy for making preferences among conflicting sources is compulsory in order to resolve contradictions.

In the sequel we show that our system is sound, complete, and supports various types of special information, such as timestamps and source tracing. We also discuss implementation issues and provide some experimental results.

2 Coherent composition of knowledge-bases

2.1 Problem description

Definition 1. A *knowledge-base* $\mathcal{KB}$ is a pair $(\mathcal{D}, \mathcal{IC})$, where $\mathcal{D}$ (the *database*) is a set of atomic formulae, and $\mathcal{IC}$ (the set of *integrity constraints*) is a finite set of first order formulae.

As usual in such cases, we apply the closed world assumption on databases, i.e., every atom that is not mentioned in the database is considered false. The underlying semantics corresponds, therefore, to minimal Herbrand interpretations.

Definition 2. A formula ψ *follows* from a database $\mathcal{D}$ if the minimal Herbrand model of $\mathcal{D}$ is also a model of ψ.

Definition 3. A knowledge-base $\mathcal{KB} = (\mathcal{D}, \mathcal{IC})$ is *consistent* if all the integrity constraints are consistent, and each one follows from $\mathcal{D}$.

Our goal is to integrate n consistent knowledge-bases, $\mathcal{KB}_i = (\mathcal{D}_i, \mathcal{IC}_i)$, $i = 1, \ldots n$, to a single knowledge-base in such a way that the data in this knowledge-base will contain everything that can be deduced from one of the sources of information, without violating any integrity constraint of another source. The idea is to consider the union of the distributed data, and then to restore its consistency. A key notion in this respect is the following:

Definition 4. [14] A *repair* of $\mathcal{KB} = (\mathcal{D}, \mathcal{IC})$ is a pair $(\mathsf{Insert}, \mathsf{Retract})$ such that $\mathsf{Insert} \cap \mathsf{Retract} = \emptyset$, $\mathsf{Insert} \cap \mathcal{D} = \emptyset$, $\mathsf{Retract} \subseteq \mathcal{D}$, and every integrity constraint follows from $\mathcal{D} \cup \mathsf{Insert} \setminus \mathsf{Retract}$. [2]
$(\mathcal{D} \cup \mathsf{Insert} \setminus \mathsf{Retract}, \mathcal{IC})$ is called a *repaired knowledge-base* of $\mathcal{KB}$.

As there may be many ways to repair an inconsistent knowledge-base, it is often convenient to make preferences among the repairs and to consider only the most preferred ones. Below are two common preference criteria.

Definition 5. Let $(\mathsf{Insert}, \mathsf{Retract})$ and $(\mathsf{Insert}', \mathsf{Retract}')$ be two repairs of a given knowledge-base.

- *set inclusion preference criterion* :
  $(\mathsf{Insert}', \mathsf{Retract}') \leq_i (\mathsf{Insert}, \mathsf{Retract})$ if $\mathsf{Insert} \subseteq \mathsf{Insert}'$ and $\mathsf{Retract} \subseteq \mathsf{Retract}'$.

[2] I.e., Insert are elements that should be inserted into $\mathcal{D}$ and Retract are elements that should be removed from $\mathcal{D}$ in order to obtain a consistent knowledge-base.

– *cardinality preference criterion*:
$(\mathsf{Insert}', \mathsf{Retract}') \leq_c (\mathsf{Insert}, \mathsf{Retract})$ if $|\mathsf{Insert}|+|\mathsf{Retract}| \leq |\mathsf{Insert}'|+|\mathsf{Retract}'|$.

Let $\leq$ be a semi-order on the set of repairs, expressing a preference criterium.

Definition 6. [14] A $\leq$-*preferred repair* of a knowledge-base $\mathcal{KB}$ is a repair $(\mathsf{Insert}, \mathsf{Retract})$ of $\mathcal{KB}$ s.t. there is no other repair $(\mathsf{Insert}', \mathsf{Retract}')$ of $\mathcal{KB}$ for which $(\mathsf{Insert}, \mathsf{Retract}) \leq (\mathsf{Insert}', \mathsf{Retract}')$. [3]

Definition 7. The set of all the $\leq$-preferred repairs of a knowledge-base $\mathcal{KB}$ is denoted by $!(\mathcal{KB}, \leq)$.

Definition 8. A $\leq$-*repaired knowledge-base* of $\mathcal{KB}$ is a repaired knowledge-base of $\mathcal{KB}$, constructed from a $\leq$-preferred repair of $\mathcal{KB}$. The set of all the $\leq$-repaired knowledge-bases of $\mathcal{KB}$ is denoted by

$$\mathcal{R}(\mathcal{KB}, \leq) = \{\, (\mathcal{D} \cup \mathsf{Insert} \setminus \mathsf{Retract}\,,\, \mathcal{IC}) \mid (\mathsf{Insert}, \mathsf{Retract}) \in\, !(\mathcal{KB}, \leq) \,\}.$$

Note that if $\mathcal{KB}$ is consistent and the preference criterion is a partial order and monotonic in the size of the repairs (as in Definition 5), then $\mathcal{R}(\mathcal{KB}, \leq) = \{\mathcal{KB}\}$, i.e., $\mathcal{KB}$ is the (only) $\leq$-repaired knowledge-base of itself, and so there is nothing to repair in this case, as expected.

Definition 9. For $\mathcal{KB}_i = (\mathcal{D}_i, \mathcal{IC}_i)$, $i = 1, \ldots n$, let $\mathcal{UKB} = (\bigcup_{i=1}^{n} \mathcal{D}_i,\ \bigcup_{i=1}^{n} \mathcal{IC}_i)$.

In the rest of this paper we describe a system that, given n distributed knowledge-bases and a preference criterion $\leq$, computes the set $\mathcal{R}(\mathcal{UKB}, \leq)$ of the $\leq$-repaired knowledge-bases of $\mathcal{UKB}$. The reasoner may use different strategies to determine the consequences of this set. Among the common approaches are the skeptical (conservative) one, that it is based on a "consensus" among all the elements of $\mathcal{R}(\mathcal{UKB}, \leq)$ (see [14]), a "credulous" approach in which entailments are decided by any element in $\mathcal{R}(\mathcal{UKB}, \leq)$, an approach that is based on a "majority vote", etc. A detailed discussion on these methods and ways of assuring the consistency of the composed data in each method, will be presented elsewhere.

We conclude this section by noting that in the sequel we shall assume that $\mathcal{IC} = \bigcup_{i=1}^{n} \mathcal{IC}_i$ is consistent. This is a usual assumption in the literature and it is justified by the nature of the integrity constrains as describing statements that are widely accepted. Thus, it is less likely that integrity constraints would contradict each other. Contradictions between the data in the different $\mathcal{KB}$'s and integrity constraints are more frequent, and may occur due to many different reasons. In the next section we consider some common cases.

[3] In [14] this notion is defined for the specific case where the preference condition is taken w.r.t. set inclusion.

2.2 Examples

In all the following examples we use set inclusion as the preference criterion.[4]

Example 1. [14, Example 1] Consider a distributed knowledge-base with relations of the form $teaches(\texttt{course\_name}, \texttt{teacher\_name})$. Suppose also that each knowledge-base contains a single integrity constraint, stating that the same course cannot be taught by two different teachers:

$$\mathcal{IC} = \{\, \forall X \forall Y \forall Z \, (teaches(X,Y) \wedge teaches(X,Z) \rightarrow Y = Z) \,\}.$$

Consider now the following two knowledge-bases:

$$\mathcal{KB}_1 = (\, \{teaches(c_1, n_1),\ teaches(c_2, n_2)\},\ \mathcal{IC}\,),$$
$$\mathcal{KB}_2 = (\, \{teaches(c_2, n_3)\},\ \mathcal{IC})$$

Clearly, $\mathcal{KB}_1 \cup \mathcal{KB}_2$ is inconsistent. Its preferred repairs are $(\emptyset,\ \{teaches(c_2, n_2)\})$ and $(\emptyset,\ \{teaches(c_2, n_3)\})$. Hence, the two repaired knowledge-bases are:

$$\mathcal{R}_1 = (\, \{teaches(c_1, n_1),\ teaches(c_2, n_2)\},\ \mathcal{IC}\,), \text{ and}$$
$$\mathcal{R}_2 = (\, \{teaches(c_1, n_1),\ teaches(c_2, n_3)\},\ \mathcal{IC}\,).$$

Example 2. [14, Example 2] Consider a distributed knowledge-base with relations of the form $supply(\texttt{supplier}, \texttt{department}, \texttt{item})$ and $class(\texttt{item}, \texttt{type})$. Let

$$\mathcal{KB}_1 = (\{supply(c_1, d_1, i_1),\ class(i_1, t_1)\},\ \mathcal{IC}), \text{ and}$$
$$\mathcal{KB}_2 = (\{supply(c_2, d_2, i_2),\ class(i_2, t_1)\},\ \emptyset), \text{ where}$$
$$\mathcal{IC} = \{\, \forall X \forall Y \forall Z \, (supply(X,Y,Z) \wedge class(Z, t_1) \rightarrow X = c_1) \,\}$$

states that only supplier c_1 can supply items of type t_1.
$\mathcal{KB}_1 \cup \mathcal{KB}_2$ is inconsistent and has two preferred repairs: $(\emptyset,\ \{supply(c_2, d_2, i_2)\})$ and $(\emptyset,\ \{class(i_2, t_1)\})$. Hence, there are two ways to repair it:

$$\mathcal{R}_1 = (\, \{supply(c_1, d_1, i_1),\ class(i_1, t_1),\ class(i_2, t_1)\},\ \mathcal{IC}\,),$$
$$\mathcal{R}_2 = (\, \{supply(c_1, d_1, i_1),\ supply(c_2, d_2, i_2),\ class(i_1, t_1)\},\ \mathcal{IC}\,).$$

Example 3. [14, Example 4] Let $\mathcal{D}_1 = \{p(a),\ p(b)\}, \mathcal{D}_2 = \{q(a),\ q(c)\}$, and $\mathcal{IC} = \{\forall X (p(X) \rightarrow q(X))\}$. Again, $(\mathcal{D}_1, \emptyset) \cup (\mathcal{D}_2, \mathcal{IC})$ is inconsistent. The corresponding preferred repairs are $(\{q(b)\},\ \emptyset)$ and $(\emptyset,\ \{p(b)\})$. The repaired knowledge-bases are therefore the following:

$$\mathcal{R}_1 = (\, \{p(a),\ p(b),\ q(a),\ q(b),\ q(c)\},\ \mathcal{IC}\,),$$
$$\mathcal{R}_2 = (\, \{p(a),\ q(a),\ q(c)\},\ \mathcal{IC}\,).$$

3 Knowledge integration through abduction

In this section we introduce an abductive method for a coherent integration of knowledge-bases. Our framework is composed of a language for describing the

[4] Generally, in what follows we shall fix a preference criterion for choosing the "best" repairs and omit its notation whenever possible.

problem domain (ID-logic, [9]), an abductive solver that is based on an abductive refutation procedure (SLDNFA, [10]), and a computational model for controlling the search ($\mathcal{A}$-system [18]).

3.1 ID-logic and abductive logic programming

ID-logic [9] is a framework for declarative knowledge representation that extends classical logic with inductive definitions. This logic incorporates two types of knowledge: definitional and assertional. Assertional knowledge is a set of first order statements, representing a general truth about the domain of discourse. Definitional knowledge is a set of rules of the form $p \leftarrow \mathcal{B}$, in which the head p is a predicate and the body $\mathcal{B}$ is a first order formula. A predicate that does not occur in any head is called *open* (sometimes called *abducible*).

Below we present an ID-logic meta-theory describing the composition of databases in terms of open predicates `insert` and *retract*. The key property of this theory is that its *abductive* solutions describe the coherent compositions. Abductive reasoning on an ID-logic theory can be performed by mapping it into an abductive logic program [8] under the extended well-founded semantics [24] and applying an abductive inference procedure to it. An *abductive logic program* (ALP)) is a triple $\mathcal{T} = (\mathcal{P}, \mathcal{A}, \mathcal{IC})$, such that

- $\mathcal{P}$ is a logic program, the clauses of which are interpreted as definitions for the predicates in their head,
- $\mathcal{A}$ is a set of predicates, none of which occurs in the head of a clause in $\mathcal{P}$. The elements in $\mathcal{A}$ are called the *abducible predicates*.
- $\mathcal{IC}$ is a set of first-order formulae, called the *integrity constraints*.

Constants, functors and predicate symbols are defined as usual in logic programs.

Definition 10. An (abductive) *solution* for a theory $(\mathcal{P}, \mathcal{A}, \mathcal{IC})$ and a query $\mathcal{Q}$ is a set Δ of ground abducible atoms, all having a predicate symbols in $\mathcal{A}$, together with an answer substitution θ, such that: (a) $\mathcal{P} \cup \Delta$ is consistent, (b) $\mathcal{P} \cup \Delta \models \mathcal{IC}$, and (c) $\mathcal{P} \cup \Delta \models \forall \mathcal{Q}\theta$.

In what follows we use ID-logic to specify the knowledge integration, and implement the reasoning process by an abductive refutation procedure. For this we represent any data in some distributed database by a predicate `db`, and denote the elements in the composed database by the predicate `fact`. The latter predicate is defined as follows:

```
fact(X) :- db(X), not retract(X).
fact(X) :- insert(X).
```

In particular, in order to restore consistency, some facts may be removed and some other facts may be introduced. These facts are represented by the (abducible) predicates `retract` and `insert`, respectively. To assure proper computations of the solutions, the following integrity constraints are also specified:[5]

[5] In what follows we use the notation "`ic :- B`" to denote the denial "`false` ← `B`".

– An element cannot be retracted and inserted at the same time:
`ic :- insert(X), retract(X).`
– An inserted element should not belong to a given database:
`ic :- insert(X), db(X).`

Assuming that all the integrity constraints of the distributed knowledge-bases are compatible and that no distinctions are made among the origins of the composed facts, the following steps are performed:

1. Each database fact X is represented by an atom $\texttt{db}(X)$.
2. Every occurrence of an atom P in some integrity constraint is replaced by $\texttt{fact}(P)$. This is done in order to assure that every integrity constraint would hold for the composed data as well.
3. A solution is computed in terms of the abducible predicates `insert` and `retract`.

3.2 The $\mathcal{A}$-system

The reasoning process of our revision system is performed by the $\mathcal{A}$-system, introduced in [18]. The basic idea of this system is a reduction of a high level specification into a lower level constraint store, which is managed by a constraint solver. The system is a synthesis of the refutation procedures SLDNFA [10] and ACLP [17], together with an improved control strategy. The latest version of the system can be obtained from `http://www.cs.kuleuven.ac.be/~dtai/kt/`. It runs on top of Sicstus Prolog 3.8.5. Below we sketch the theoretical background as well as some practical considerations behind this system. For more information, see [10] and [18].

Abductive inferences Given an abductive theory $(\mathcal{P}, \mathcal{A}, \mathcal{IC})$ as defined above, the logical reduction of a query $\mathcal{Q}$ can be described as a derivation for $\mathcal{Q}$ through a rewriting state process. A state $\mathcal{S}$ consists of two types of elements: a set $\texttt{Pos}(\mathcal{S})$ of literals (possibly with free variables), called *positive goals*, and a set $\texttt{Neg}(\mathcal{S})$ of denials, called *negative goals*. The set $\Delta(\mathcal{S})$ denotes the abducible atoms in $\mathcal{S}$, i.e. positive goal atoms whose predicate is an abducible. $\mathcal{C}(\mathcal{S})$ denotes the set of constraint atoms in $\mathcal{S}$.

A rewriting derivation proceeds from state $\mathcal{S}_i$ by selecting a literal of $\mathcal{S}_i$ and applying a suitable inference rule, yielding a new state $\mathcal{S}_{i+1}$. The main inference rules are given by the following rewrite rules. In the list below we denote by A and B some literals, and by C a constraint literal. $\mathcal{P}$ denotes the theory under consideration. For readability, we do not mention cases in which $\texttt{Pos}(\mathcal{S})$ or $\texttt{Neg}(\mathcal{S})$ is the same in states number i and $i+1$.

– *Rules for defined predicates:*
  - if $A(\overline{X}) \leftarrow B_j[\overline{X}] \in \mathcal{P}$ and $A(\overline{t}) \in \texttt{Pos}(\mathcal{S}_i)$, then $\texttt{Pos}(\mathcal{S}_{i+1}) = \texttt{Pos}(\mathcal{S}_i) \setminus \{A(\overline{t})\} \cup \{B_j[\overline{t}]\}$.
  - if $\leftarrow A(\overline{t}), Q \in \texttt{Neg}(\mathcal{S}_i)$, then $\texttt{Neg}(\mathcal{S}_{i+1}) = \texttt{Neg}(\mathcal{S}_i) \setminus \{\leftarrow A(\overline{t}), Q\} \cup U$, where $U = \{\leftarrow B_j[\overline{t}], Q \mid A(\overline{t}) \leftarrow B_j[\overline{t}] \in \mathcal{P}\}$.

– *Rules for open predicates:*
  - if $\leftarrow A(\overline{t}), Q \in \mathtt{Neg}(\mathcal{S}_i)$ and $p(\overline{s}) \in \Delta(\mathcal{S}_i)$ then $\mathtt{Neg}(\mathcal{S}_{i+1}) = \mathtt{Neg}(\mathcal{S}_i) \setminus \{\leftarrow A(\overline{t}), Q\} \cup \{U\} \cup \{R\}$, where $U = \leftarrow \overline{t} = \overline{s}, Q$, and $R = \leftarrow A(\overline{t}), \overline{t} \neq \overline{s}, Q$.
– *Rules for negations:* Assume that A is not a constraint literal.
  - if $\neg A \in \mathtt{Pos}(\mathcal{S}_i)$ then $\mathtt{Pos}(\mathcal{S}_{i+1}) = \mathtt{Pos}(\mathcal{S}_i) \setminus \{\neg A\}$ and $\mathtt{Neg}(\mathcal{S}_{i+1}) = \mathtt{Neg}(\mathcal{S}_i) \cup \{\leftarrow A\}$.
  - if $\leftarrow \neg A, Q \in \mathtt{Neg}(\mathcal{S}_i)$ then one of the following branches is taken:
    1. $\mathtt{Pos}(\mathcal{S}_{i+1}) = \mathtt{Pos}(\mathcal{S}_i) \cup \{A\}$ and $\mathtt{Neg}(\mathcal{S}_{i+1}) = \mathtt{Neg}(\mathcal{S}_i) \setminus \{\leftarrow \neg A, Q\}$.
    2. $\mathtt{Neg}(\mathcal{S}_{i+1}) = \mathtt{Neg}(\mathcal{S}_i) \setminus \{\leftarrow \neg A, Q\} \cup \{\leftarrow A, \leftarrow Q\}$.
– *Rules for constraint literals:*
  - if $\leftarrow C, Q \in \mathtt{Neg}(\mathcal{S}_i)$ then one of the following branches is taken:
    1. $\mathtt{Pos}(\mathcal{S}_{i+1}) = \mathtt{Pos}(\mathcal{S}_i) \cup \{\neg C\}$, $\mathtt{Neg}(\mathcal{S}_{i+1}) = \mathtt{Neg}(\mathcal{S}_i) \setminus \{\leftarrow C, Q\}$.
    2. $\mathtt{Pos}(\mathcal{S}_{i+1}) = \mathtt{Pos}(\mathcal{S}_i) \cup \{C\}$, $\mathtt{Neg}(\mathcal{S}_{i+1}) = \mathtt{Neg}(\mathcal{S}_i) \setminus \{\leftarrow C, Q\} \cup \{\leftarrow Q\}$.

    Remark: It is important here to assume that the underlying constraint solver is capable of handling negated constraint literals. This is indeed the case with the constraint solver used by our system (Sicstus).

The *initial state* $\mathcal{S}_0$ for a theory $\mathcal{P}$ and a query $\mathcal{Q}$ consists of the query $\mathcal{Q}$ as a positive goal and the set of all denials in $\mathcal{P}$ as negative goals. A *successful state* $\mathcal{S}$ fulfills the following conditions:

1. $\mathcal{S}$ contains positive goals only of the form of abducible atoms or constraint atoms,
2. negative goals in $\mathcal{S}$ are denials containing some open atom $p(\overline{t})$ which has already been selected and resolved with each abduced atom $p(\overline{s}) \in \mathcal{S}$, and
3. the constraint store $\mathcal{C}(\mathcal{S})$ of $\mathcal{S}$ is satisfiable.

Definition 11. A *successful abductive derivation* of a query $\mathcal{Q}$ w.r.t. $\mathcal{P}$ is a sequence of states $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_n$, where: (a) $\mathcal{S}_0$ is an initial state for $\mathcal{P}$ and $\mathcal{Q}$, (b) For every $0 \leq i \leq n-1$, $\mathcal{S}_{i+1}$ is obtained from $\mathcal{S}_i$ by applying one of the transition rules, and (c) $\mathcal{S}_n$ is a successful state.

Whenever `false` is derived (in one of the constraint domains) the derivation backtracks. A derivation *flounders* when universally quantified variables appear in a selected negated literal in a denial.

Let $\mathcal{S}_n$ be a final state of a successful derivation. Then any substitution θ that assigns a ground term to each free variable of $\mathcal{S}_n$ and which satisfies the constraint store $\mathcal{C}(\mathcal{S}_n)$ is called a *solution substitution* of $\mathcal{S}_n$. Such a substitution always exists since $\mathcal{C}(\mathcal{S}_n)$ is satisfiable for a successful derivation.

Theorem 1. [18] *Let $\mathcal{T} = (\mathcal{P}, \mathcal{A}, \mathcal{IC})$ be an abductive theory s.t. $\mathcal{P} \models \mathcal{IC}$, $\mathcal{Q}$ a query, $\mathcal{S}$ the final state of a successful derivation for $\mathcal{Q}$, and θ a solution substitution of $\mathcal{S}$. Then the pair $\theta(\Delta(\mathcal{S}))$ and θ is an abductive solution for $\mathcal{T}$ and $\mathcal{Q}$.*

Control strategy The selection strategy applied during the derivation process is crucial. A Prolog-like selection strategy (left first, depth first) often leads to trashing, because it is blind to other choices and it does not result in a global overview of the current state of the computation. In the development of the $\mathcal{A}$-system the main focus was on the improvement of the control strategy. The idea is to apply first those rules that have a deterministic change of the state, and so information is propagated. If none of such rules are applicable, then one of the left over choices is selected and a choice is made. This resembles a CLP-solver, in which the constraints propagate their information as soon a choice is made. This propagation yields less amount of choices and thus often dramatically increases the performance.

3.3 Implementation and experiments

In this section we present the structure of our system, discuss a few implementation issues, and give some experimental results.

The structure of the system Figure 1 shows a layered description of the implemented system. The upper most level consists of the data to be integrated, i.e., the database information and the integrity constrains. This layer together with the composer form an ID-Logic theory that is processed by the $\mathcal{A}$-system.

The *composer* consists of the meta-theory for integrating the distributed data in a coherent way. It is interpreted here as an abductive theory, in which the abducible predicates provide the information on how to restore the consistency of the amalgamated data.

The abductive system (enclosed by dotted lines in Figure 1) consists of three main components: A finite domain constraint solver (the one of Sicstus Prolog), an abductive meta-interpreter (described above), and an optimizer.

The *optimizer* is a component that, given a preference criterion on the space of the solutions, computes only the most-preferred (abductive) solutions. Given such a preference criterion, this component prunes "on the fly" those branches of the search tree that lead to worse solutions than what we have already computed. This is actually a branch and bound "filter" on the solutions space that speeds-up execution and makes sure that only the desired solutions will be obtained. If the preference criterion is monotonic (in the sense that from a partial solution it can be determined whether it potentially leads to a solution that is not worse than a current one), then the optimizer is *complete*, that is, it can compute all the optimal solutions (see also Section 3.4).

Note that the optimizer is a general component added to the $\mathcal{A}$-system. Not only this domain benefits, but it is useable in other application domains like e.g. planning.

Experimental study Figure 2 contains the code (data section + composer) for implementing Example 1 (The codes for Examples 2 and 3 are similar). We have executed this code as well as other examples from the literature in our system.

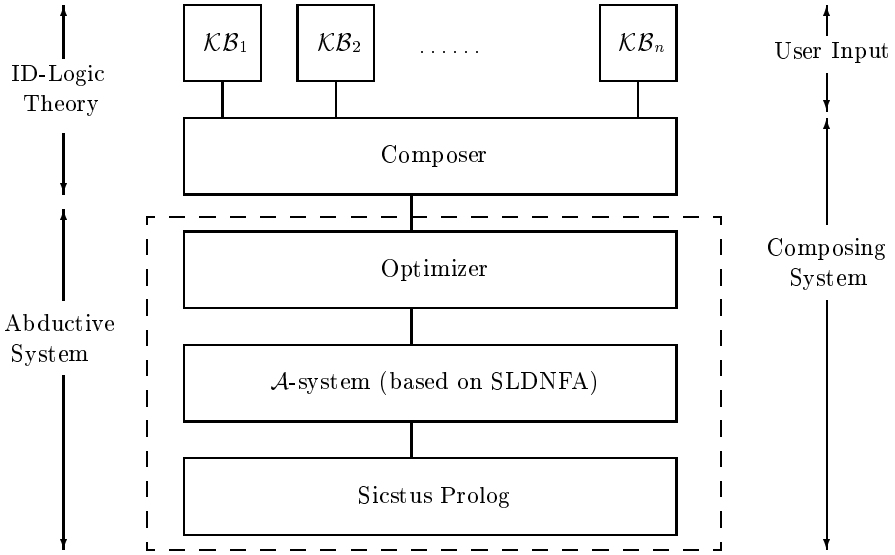

Fig. 1. A schematic view of the system components.

As Theorem 3 below guarantees, the output in each case was the set of the most preferred solutions of the corresponding problem.

3.4 Soundness and completeness

In this section we give some soundness and completeness results for our system. In what follows we denote by $\mathcal{T}$ an abductive theory in ID-logic, constructed as describe above for composing n given knowledge-bases $\mathcal{KB}_1, \ldots, \mathcal{KB}_n$. Also, $\texttt{Proc}_{\texttt{ALP}}$ denotes some sound abductive proof procedure (e.g., SLDNFA [10]).

Proposition 1. *Every abductive solution that is obtained by* $\texttt{Proc}_{\texttt{ALP}}$ *for a theory* $\mathcal{T}$ *is a repair of* $\mathcal{UKB}$.

Proof: By the construction of $\mathcal{T}$ it is easy to see that all the conditions specified in Definition 4 are met: the first two conditions are assured by the integrity constraints of the composer. The third condition immediately follows from the composer's rules. The last condition is satisfied since by the soundness of $\texttt{Proc}_{\texttt{ALP}}$ it produces abductive solutions Δ_i for $\mathcal{T}$, thus by the second property in Definition 10, for every such solution $\Delta_i = (\mathsf{Insert}_i, \mathsf{Retract}_i)$ we have that $\mathcal{P} \cup \Delta_i \models \mathcal{IC}$. Since $\mathcal{P}$ contains a data section with all the facts, it follows that $\mathcal{D} \cup \Delta_i \models \mathcal{IC}$, i.e. every integrity constraints follows from $\mathcal{D} \cup \mathsf{Insert}_i \setminus \mathsf{Retract}_i$. □

Theorem 2. (Soundness) *Every output that is obtained by running* $\mathcal{T}$ *in the* $\mathcal{A}$*-system together with a* $\leq_c$*-optimizer* [*respectively, together with an* $\leq_i$*-optimizer*] *is a* $\leq_c$*-preferred repair* [*respectively, an* $\leq_i$*-preferred repair*] *of* $\mathcal{UKB}$.

Proof: Follows from Proposition 1 (since the $\mathcal{A}$-system is based on SLDNFA that is a sound abductive proof procedure), and the fact that the $\leq_c$-optimizer prunes

```
/* ------- Composer: -------
:- dynamic ic/0, fact/1, db/1.

abducible(insert(_)).
abducible(retract(_)).

fact(X) :- db(X), not(retract(X)).
fact(X) :- insert(X).
ic :- insert(X), db(X).
ic :- insert(X), retract(X).

/* ------- Example 1: -------
db(teaches(1,1)).  db(teaches(2,2)).                    % D1
db(teaches(2,3)).                                       % D2
ic :- fact(teaches(X,Y)), fact(teaches(X,Z)), Y\=Z.     % IC
```

Fig. 2. Code for Example 1

paths that lead to solutions which are not $\leq_c$-preferable. Similar arguments hold for systems with an $\leq_i$-optimizer. □

Proposition 2. *Suppose that the query '$\leftarrow$ `true`' has a finite SLDNFA-tree w.r.t. $\mathcal{T}$. Then every $\leq_c$-preferred repair and every $\leq_i$-preferred repair of $\mathcal{UKB}$ is obtained by running $\mathcal{T}$ in the $\mathcal{A}$-system.*

Outline of proof: The proof that all the abductive solutions with minimal cardinality are obtained by the system is based on [10, Theorem 10.1], where it is shown that SLDNFA^o, which is an extension of SLDNFA, aimed for computing solutions with minimal cardinality, is complete (see [10, Section 10.1] for further details). Similarly, the proof that all the abductive solutions which are minimal w.r.t. set inclusion are obtained by the system is based on [10, Theorem 10.2] that shows that SLDNFA$_+$, which is another extension of SLDNFA, aimed for computing minimal solutions w.r.t. set inclusion, is also complete (see [10, Section 10.2] for further details).

Now, $\mathcal{A}$-system is based on the combination of SLDNFA^o and SLDNFA$_+$. Moreover, as this system does not change the refutation tree (but only controls the way rules are selected), Theorems 10.1 and 10.2 in [10] are applicable in our case as well. Thus, all the $\leq_c$- and the $\leq_i$-minimal solutions are produced. This in particular means that every $\leq_c$-preferred repair as well as every $\leq_i$-preferred repair of $\mathcal{UKB}$ is produced by our system. □

Theorem 3. (Completeness) *In the notations of Proposition 2 and under its assumptions, the output of the execution of $\mathcal{T}$ in the $\mathcal{A}$-system together with a $\leq_c$-optimizer [respectively, together with an $\leq_i$-optimizer] is exactly $!(\mathcal{UKB}, \leq_c)$ [respectively, $!(\mathcal{UKB}, \leq_i)$].*

Proof: We shall show the claim for the case of $\leq_c$; the proof w.r.t. $\leq_i$ is similar.

Let $(\mathsf{Insert}, \mathsf{Retract}) \in !(\mathcal{UKB}, \leq_c)$. By Proposition 2, $\Delta = (\mathsf{Insert}, \mathsf{Retract})$ is one of the solutions produced by the $\mathcal{A}$-system for $\mathcal{T}$. Now, during the execution of our system together with the $\leq_c$-optimizer, the path that corresponds to Δ cannot be pruned from the refutation tree, since by our assumption $(\mathsf{Insert}, \mathsf{Retract})$ has a minimal cardinality among the possible solutions, so the pruning condition is not satisfied. Thus Δ will be produced by the $\leq_c$-optimized system. For the converse, suppose that $(\mathsf{Insert}, \mathsf{Retract})$ is some repair of $\mathcal{UKB}$ that is produced by the $\leq_c$-optimized system. Suppose for a contradiction that $(\mathsf{Insert}, \mathsf{Retract}) \notin !(\mathcal{UKB}, \leq_c)$. By the proof of Proposition 2, there is some $\Delta' = (\mathsf{Insert}', \mathsf{Retract}') \in !(\mathcal{UKB}, \leq_c)$ that is constructed by the $\mathcal{A}$-system for $\mathcal{T}$, and $(\mathsf{Insert}', \mathsf{Retract}') <_c (\mathsf{Insert}, \mathsf{Retract})$. But $|\Delta'| < |\Delta|$, and so the $\leq_c$-optimizer would prone the path of the Δ solution once its cardinality becomes bigger than $|\Delta'|$. This contradicts our assumption that $(\mathsf{Insert}, \mathsf{Retract})$ is produced by the $\leq_c$-optimized system. □

4 Handling specialized information

4.1 Timestamped information

Many database applications contain temporal information. This kind of data may be divided to two types: time information that is part of the data itself, and time information that is related to database operations (e.g., records on when the database was updated). Consider, for instance, $\texttt{birth\_day(John,15/05/2001)}_{16/05/2001}$. Here, John's date of birth is an instance of the former type of time information, and the subscripted data that describes the time in which this fact was added to the database, is an instance of the latter type of time information.

In our approach, timestamp information can be integrated by adding a temporal theory describing the state of the database at any particular time point. One way of doing so is by using *situation calculus*. In this approach a database is described by initial information and a history of events performed during the database lifetime (see [25]). Here we use a different approach, which is based on *event calculus*. The idea is to make a distinction between two kinds of events: `add_db` and `del_db` that describe the database modifications, and the composer-driven events `insert` and `retract` that are used for constructing database repairs. In this view, the extended composer has the following form:

```
holds_at(P,T) :- initially(P), not clipped(0,P,T).
holds_at(P,T) :- add(P,E), E<T, not clipped(E,P,T).
clipped(E,P,T) :- del(P,C), E≤C, C<T.

add(P,T) :- add_db(P,T).        add(P,T) :- insert(P,T).
del(P,T) :- del_db(P,T).        del(P,T) :- retract(P,T).
ic :- insert(P,T), retract(P,T).
ic :- insert(P,T), add_db(P,T).
ic :- retract(P,T), del_db(P,T).
```

In this extended context the integrity constrains must be carefully specified. Consider, e.g. the statement that a person can be born only on one date:

```
ic :- holds_at(birth_day(P,D1),T), holds_at(birth_day(P,D2),T), D1≠D2.
```

The problem here is that to ensure consistency this constraint must be checked at every point in time. This may be avoided by a simple rewriting that ensures that the constraint will be verified only when an event occurs:

```
ic(birth,T) :- holds_at(birth_day(P,D1),T),
               holds_at(birth_day(P,D2),T), D1\=D2.
ic :- add_db(birth_day(_,_),T), NT = T+1, ic(birth,NT).
ic :- del_db(birth_day(_,_),T), NT = T+1, ic(birth,NT).
```

4.2 Keeping track of source identities

There are cases in which it is important to preserve the identity of the database from which a specific piece of information was originated. This is useful, for instance, when one wants to make preferences among different sources, or when some specific source should be filtered out (e.g, when the corresponding database is not available or becomes unreliable). This kind of information may be decoded by adding another argument to every fact, which denotes the identity of its origin. This requires minor modifications in the basic composer, since the composer controls the way in which the data is integrated. As such, it is the only component that can keep track to the source of the information.

Suppose, then, that for every database fact we add another argument that identifies its source. I.e., `db(X,S)` denotes that `X` is a fact originated from a database `S`. The composer then has the following form:

```
fact(X,S) :- db(X,S), not retract(X)
fact(X,composer) :- insert(X)
ic :- insert(X), db(X,S)
ic :- insert(X), retract(X)
```

Note that the composer considers itself as an extra source that inserts brand new data facts. Now it is possible, e.g., to trace information that comes from a specific source, make preferences among different sources (by specifying appropriate integrity constraints), and filter data that comes from certain sources. The last property is demonstrated by the following rule:

```
validFact(X) :- fact(X,S), trusted_source(S)
```

where `trusted_source` enumerates all reliable sources of the data.

4.3 Handling quantitative information

Next we consider a potential way of decoding in the integrated data some quantitative information, such as certainty factors or probabilities.

Suppose that `db(X,i)` denotes that fact `X` holds with probability `i`. One can define a strategy on how to reason with this kind of information, and decode

it in the composer. For instance, the composer below uses a conservative policy that takes for each fact its lowest probability:

```
fact(X,i) :- db(X,_), not retract(X), i = min{j | db(X,j)}
fact(X,1) :- insert(X,1)
ic :- insert(X,1), db(X,_)
ic :- insert(X,1), retract(X)
```

For implementing this kind of program the underlying system should be able to compute aggregations (possibly together with recursion). Adding this capability to our system is one of the subjects for a future work.

5 Conclusion and further work

In this paper we have developed a formal declarative foundation for rendering coherent data, provided by different knowledge-bases, and presented an application that implements this approach. Like other systems (e.g., [6, 14, 20, 29]), our system mediates among the sources of information and between the reasoner and the underlying data.

Composing distributed data by a meta-theory in ID-logic yields a robust and easily extendable system. Extra meta information about the data facts, such as time stamps and source, are easily dealt with by extending the meta-theory properly. Due the inherent modularity of the chosen approach, each part is independent and can be adapted according to the needs.

It is important to note that our composing system inherits the functionality of the underlying solver. This implies, in particular, flexibility, modularity, easy interaction with different sources of information, and the ability to reason with *any* set of first order integrity constraints.[6] As such, our system may be easily modified and extended with addition background knowledge.

Among the directions for further exploration are dealing with more general forms of databases, in which views (or rules) are allowed, and lifting the condition that all the integrity constraints are compatible with each other. Another important challenge is to extend the capabilities of the abductive system with aggregation. This would allow us to integrate different types of databases, and would provide means of solving new kinds of problems.

References

1. M.Arenas, L.E.Bertossi, J.Chomicki. Consistent query answers in inconsistent databases. *Proc. PODS'99*, 68–79, 1999.
2. O.Arieli. Four-valued logics for reasoning with uncertainty in prioritized data. In: *Information, Uncertainty, Fusion*, 263–309, Kluwer, 1999.
3. C.Baral, S.Kraus, J Minker. Combining Multiple Knowledge Bases. *IEEE Trans. on Knowledge and Data Enginnering* 3(2), 208–220, 1991.
4. S.Benferhat, C.Cayrol, D.Dubois, J.Lang, H.Prade. Inconsistency management and prioritized syntax-based entailment. *Proc. IJCAI'93*, 640–645, 1993.

[6] Provided that the constraints do not lead to floundering. To the best of our knowledge no other application of data integration has this ability.

5. S.Benferhat, D.Dubois, H.Prade. How to infer from inconsistent beliefs without revising? *Proc. IJCAI'95*, 1449–1455, 1995.
6. L.Bertossi, M.Arenas, C.Ferretti. SCDBR: An automated reasoner for specifications of database updates. *Intelligent information Systems* 10(3), 253–280, 1998.
7. F.Bry. Query Answering in Information Systems with Integrity Constraints. *Proc. IICIS'97*, 113–130, 1997.
8. M. Denecker, A.C. Kakas. Abductive Logic Programming, *Special issue of Journal of Logic Programming*, 44 (1-3), 2000.
9. M.Denecker. Extending classical logic with inductive definitions. *Proc. CL'2000*, J. Lloyd et al., editors, LNAI 1861, Springer, 703–717, 2000.
10. M.Denecker, D.De Schreye. SLDNFA an abductive procedure for abductive logic programs. *Journal of Logic Programming* 34(2), 111–167, 1998.
11. M.Fitting. Negation as refutation. *Proc. LICS'89*, IEEE Press, 63–70, 1989.
12. M.Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming* 11(2), 91–116, 1991.
13. M.Gertz, U.W.Lipeck. An extensible framework for repairing constraint violations. *Proc. IICIS'97*, 89–111, 1997.
14. S.Greco, E.Zumpano. Querying inconsistent databases. *Proc. LPAR'2000*, M.Parigot and A.Voronokov, editors, LNAI 1955, 308–325, Springer, 2000.
15. K.Inoue, C.Sakama. Abductive framework for nonmonotonic theory change. *Proc. IJCAI'95*, 204-210, 1995.
16. T.Kakas, P.Mancarella. Database updates through abduction. *Proc VLDB'90*, 650–661, 1990.
17. T.Kakas, A.Michael, C.Mourlas. ACLP: Abductive constraint logic programming. *Journal of Logic Programming* 44(1–3), 129–177, 2000.
18. T.Kakas, B.Van Nuffelen, M.Denecker. $\mathcal{A}$-System: Problem solving through abduction. Proc. IJCAI'01, 2001.
19. M.Kifer, E.L.Lozinskii. A logic for reasoning with inconsistency. *Journal of Automated Reasoning* 9(2), 179–215, 1992.
20. P.Liberatore, M.Schaerf. BReLS: a system for the integration of knowledge bases. *Proc KR'2000*, 145–152, 2000.
21. J.Lin, A.O.Mendelzon. Merging databases under constraints. *Int. Journal of Cooperative Information Systems* 7(1), 55–76, 1998.
22. B.Messing. Combining knowledge with many-valued logics. *Data and Knowledge Engineering* 23, 297–315, 1997.
23. A.Olivé. Integrity checking in deductive databases. *Proc VLBD'91*, 513–523, 1991.
24. L.M. Pereira, J.N. Aparicio, J.J. Alferes. , Hypothetical Reasoning with Well Founded Semantics , *Proc. of the 3th Scandinavian Conference on AI* , B. Mayoh, IOS Press, 289-300, 1991
25. R. Reiter. On specifying database updates. *Journal of Logic Programming*, 25(1), 53–91, 1995.
26. P.Z.Revesz. On the semantics of theory change: Arbitration between old and new information. *Proc. PODS'93*, 71–82, 1993.
27. C.Sakama, K.Inoue. Updating extended logic programs through abduction. *Proc LPNMR'99*, 147–161, 1999.
28. V.S.Subrahmanian. Mechanical proof procedures for many valued lattice-based logic programming. *Journal of Non-Classical Logic* 7, 7–41, 1990.
29. V.S.Subrahmanian. Amalgamating knowledge-bases. *ACM Trans. on Database Systems* 19(2), 291–331, 1994.
30. S.Verbaeten, M.Denecker, D.De Schreye. Compositionality of normal open logic programs. *Journal of Logic Programming* 41(3), 151–183, 2000.