Introducing Cooperation and Actions in Amalgamated Knowledge Bases

Elisa Bertino
University of Milano, Italy
bertino@dsi.unimi.it

Barbara Catania
University of Genova, Italy
catania@disi.unige.it

Paolo Perlasca
University of Milano, Italy
perlasca@dsi.unimi.it

Abstract

The theory of Amalgamated Knowledge Bases [7] represents a formal logical foundation for heterogeneous databases. In an Amalgamated Knowledge Base, data sources are modeled by generalized annotated logic. Moreover, an Amalgamated Knowledge Base is equipped with a supervisor acting as a mediator for amalgamating knowledge from the local databases. Even if the framework is quite appealing, it does not model dynamic aspects. Moreover, no communication channels among local databases are supported and cooperation is provided only through the supervisor. In this paper, we extend the theory of Amalgamated Knowledge Bases to deal with actions and cooperation among local databases.

1. Introduction

In a multidatabase system one of the central problems is how knowledge from multiple and heterogeneous sources can be integrated. Two different types of integration can be devised: *static integration*, concerning the ability to model heterogeneous data sources and to integrate several data management systems to collectively answer user queries; *dynamic integration*, concerning the ability to update data and to propagate updates among the various data sources, possibly through the use of active rules. Integration can be provided both at the global level or at the local one. In the first case, some software components, such as *mediators* [8], can be used, providing users with an integrated view of multiple sources, making transparent the underlying data heterogeneity. In the second case, local databases can be extended to directly interoperate, so that cooperation among local sources can be supported. In both cases, unlike centralized database systems, there is the need of modeling knowledge which is often uncertain.

In designing a multidatabase system, we believe that an important aspect is the definition of a formal framework by which integration among heterogeneous systems can be modeled according to a declarative style. This consideration calls for a declarative approach allowing one to fully model all aspects of integration described above and to provide the basis by which properties of heterogeneous information systems can be proved. To this purpose, the use of a logical approach seems very promising.

Among the logical approaches that have been proposed, the *amalgamated knowledge base* framework proposed by Subrahamanian [7] is one of the few proposals providing a formal logical foundation to cooperative knowledge bases. In this framework, generalized annotated logic [6] is used to model data sources. The use of annotated logic provides the right formalism for modeling inconsistencies and uncertain knowledge, typical of heterogeneous environments. The notion of supervisor is thus introduced as a *mediator*, amalgamating the knowledge from the various local databases. Even if the amalgamated knowledge based framework is quite appealing, it does not model other aspects, that are very relevant in the context of multidatabase systems. In particular, it does not models dynamic aspects and cooperation between local databases and thus it does not support reactive behavior.

In [2], we have proposed a logical framework, called *Heterogeneous U-Datalog* (HU-Datalog for short), supporting both static and dynamic integration and cooperation between data sources. Such framework, however, does not support the amalgamation of the knowledge from the local sources and it does not model uncertainty. More precisely, in an *HU-Datalog system*, each local source consists of an extensional database and an intensional database represented as a set of U-Datalog rules [4]. Sources cooperate through message passing through labeled atoms in rule bodies. This means that, during the evaluation process, a source may require the evaluation of a subquery to another source. Labels can also be variables, thus supporting both static and dynamic communication channels. In order to exhibit a reactive behavior, active rules in the style of Active-U-Datalog [1] are also included both in each data source and at the global level, to provide update propagation among heterogeneous databases, according to the PARK semantics [5]. Under the PARK semantics, conflicts, i.e., situations where two or more active rules are fired and require both

0-7695-0996-7/01 $10.00 © 2001 IEEE

the insertion and the deletion of the same information, are solved according to specific application requirements.

The aim of this paper is to combine the Amalgamated Knowledge Based Approach proposed in [7] (but without considering negation) with Heterogeneous U-Datalog, in order to obtain a framework, called *Annotated Heterogeneous U-Datalog* (AHU-Datalog for short), modeling static and dynamic integration of heterogeneous databases, in a cooperative environment modeling uncertainty. In the proposed framework, each source is an AHU-Datalog database, i.e., an HU-Datalog database in which each atom is annotated with a specific truth value, taken from a complete lattice. Local databases can be integrated through the use of a *supervisor*. Differently from what has been presented in [7], in our framework the supervisor can execute updates and supports global active rules. Such rules can propagate updates depending on the whole status of the integrated system. The result is a language modeling static and dynamic behavior, uncertainty and cooperation, representing a formal basis for analyzing properties of heterogeneous systems. A fixpoint semantics is proposed for the amalgam, integrating those presented in [7] and in [2].

The paper is organized as follows. Section 2 introduces the syntax of AHU-Datalog whereas the semantics is presented in Section 3. Due to space constraints, proofs of the proposed results are not included but can be found in [3]. Finally, Section 4 presents some conclusions and outlines future work.

2. AHU-Datalog: the syntax

In the following we introduce the notions at the basis of the proposed framework. All the proposed concepts refers to a complete lattice $\mathcal{T}$, containing a least element $\bot$.

Annotations Annotations are constructed upon a specific $\mathcal{T}$-language, composed of: (i) the union $\mathcal{F}$ of sets $\mathcal{F}^i$ of total continuous functions (thus, $\mathcal{F} = \bigcup_{i \geq 1} \mathcal{F}^i$), each of type $(\mathcal{T})^i \rightarrow \mathcal{T}$, such that each $f \in \mathcal{F}^i$ is computable and each set $\mathcal{F}^i$ contains an i-ary function $\sqcup_i$[1] that, given $\mu_1, \ldots, \mu_i$ as input, returns the *least upper bound* (*lub*) of $\{\mu_1, \ldots, \mu_i\}$; (ii) a set $\mathcal{C}$ of constant symbols. We assume that the $\mathcal{T}$-language also contains an infinite set V_a of variable symbols. An annotation is a $\mathcal{T}$-term, that is: (i) an element of the lattice $\mathcal{T}$ ("simple" annotation term); (ii) a variable ("simple" annotation term); (iii) a "complex" annotation term defined as follows: if $f \in \mathcal{F}^i$ and $\mu_1 \ldots, \mu_i$ are annotation terms, then $f(\mu_1 \ldots, \mu_i)$ is a complex annotation term.

Atoms We consider a many sorted signature $\Sigma = \{\Sigma_{db}, \Sigma_v, \Sigma_a\}$. Σ_{db} is the set of database identifiers, Σ_v is the set of constant value symbols, and Σ_a is $\mathcal{F} \cup \mathcal{C}$. Sets Σ_{db}, Σ_v, and Σ_a are disjoint. We consider a set of predicate symbols Π, partitioned into extensional predicate symbols Π^e, intensional predicate symbols Π^i, and update predicate symbols Π^u, defined as $\Pi^u = \{+p, -p \mid p \in \Pi^e\}$. A family of sets of variable symbols for each sort $V = \{V_{db}, V_v, V_a\}$ is also considered. We denote with $Term_t$ the set $\Sigma_t \cup V_t$, with $t \in \{db, v, a\}$. We denote with (Π, Σ, V)-atom an atom whose predicate belongs to Π and whose terms are in $\Sigma \cup V$.[2]

Cooperation among databases in the system is represented by *labeled atoms* of the form $db \diamond p(\tilde{t})$, where $db \in Term_{db}$ (*simple label*), meaning that $p(\tilde{t})$ must be solved in db, or $D \diamond p(\tilde{t})$, where $D = \{db_1, ..., db_n\} \subseteq \Sigma_{db}$ (*multiple label*), meaning that $p(\tilde{t})$ must be solved in db_i, $i = 1, ...n$.

Uncertainty is represented by *annotated labeled atoms* of the form $A : \psi$, where A is a possibly labeled atom and ψ is an annotation. $A : \psi$ is *c-annotated* if ψ is a constant, *v-annotated* if ψ is a variable, *t-annotated* if ψ is a complex annotation term. The intuitive meaning assigned to $A : \mu$, where μ is a truth value of a chosen lattice, is: "when $A : \mu$ is true, the ground atom A is true with truth value *at most* equal to μ". We assume that the meaning of an annotated update atom $\alpha_i A_i : \mu_i$ is: "insert/delete atom $A_i : \mu_i$".

Rules Deductive and active rules are defined as follows.

Definition 1 (AHU-Datalog deductive rule) *An* AHU-Datalog deductive rule *is a rule of the form*

$$A_0 : \mu_0 \leftarrow A_1 : \mu_1, \ldots, A_k : \mu_k, db_1 \diamond A_{k+1} : \mu_{k+1},$$
$$\ldots, db_p \diamond A_w : \mu_w, X_1 \diamond A_{w+1} : \mu_{w+1}, \ldots,$$
$$X_q \diamond A_r : \mu_r, \alpha_{r+1} A_{r+1} : \mu_{r+1}, \ldots, \alpha_n A_n : \mu_n$$

where (i) $A_0 : \mu_0$ is an annotated (Π^i, Σ, V)-atom; (ii) $A_l : \mu_l$, $l = 1, \ldots, k$, is a c-annotated or v-annotated $(\Pi^i \cup \Pi^e, \Sigma, V)$-atom, referring to the database where the rule is defined; (iii) $db_1 \diamond A_{k+1} : \mu_{k+1}, \ldots, db_p \diamond A_w : \mu_w$ are simple labeled c-annotated or v-annotated $(\Pi^i \cup \Pi^e, \Sigma, V)$-atoms, referring to specific databases, while $X_1 \diamond A_{w+1} : \mu_{w+1}, \ldots, X_q \diamond A_r : \mu_r$ are simple labeled c-annotated or v-annotated $(\Pi^i \cup \Pi^e, \Sigma, V)$-atoms, referring to databases that are not yet specified; (iv) $\alpha_j A_j : \mu_j$, $j = r + 1, \ldots, n$, is a c-annotated or v-annotated (Π^u, Σ, V)-atom; (v) $X_1, \ldots, X_q$ are variables in V_{db}; (vi) $\alpha_s A_s \neq \alpha_t A_t$ for $k + 1 \leq s, t \leq n$, $s \neq t$.

$A_0 : \mu_0$ is the head of the rule, while the body consists of the query part $A_1 : \mu_1, \ldots, A_k : \mu_k, db_1 \diamond A_{k+1} : \mu_{k+1}, \ldots, db_p \diamond A_w : \mu_w, X_1 \diamond A_{w+1} : \mu_{w+1}, \ldots, X_q \diamond A_r : \mu_r$ and the update part $\alpha_{r+1} A_{r+1} : \mu_{r+1}, \ldots, \alpha_n A_n : \mu_n$,

[1] For simplicity, when there is not ambiguity, we use $\sqcup$ instead of $\sqcup_i$.

[2] In the following we assume that a substitution is a tuple of functions $\theta = \{\theta_{db}, \theta_v, \theta_a\}$, dealing respectively with variables in V_{db}, V_v, and V_a.

that cannot be both empty. For safety, each variable appearing in the head (or in its annotation) must also appear in a deductive atom in the body (or in its annotations); moreover, $X_1, \ldots, X_q$ must appear as arguments of an extensional atom in $A_1 : \mu_1, \ldots, A_w : \mu_w$. □

The intuitive meaning of a deductive rule is: "if A_i, $i = 1, \ldots, k$, is true with truth value at most μ_i in the database where the rule is defined, A_{k+1} is true with truth value at most μ_{k+1} in db_1,…,A_w is true with truth value at most μ_w in db_p, A_{w+1} is true with truth value at most μ_{w+1} in the database to which X_1 is instantiated,…,A_r is true with truth value at most μ_r in the database to which X_q is instantiated, then A_0 is true with truth value at most μ_0 in the database where the rule is defined and, as side effect, the updates $\alpha_{r+1}A_{r+1} : \mu_{r+1}, \ldots, \alpha_n A_n : \mu_n$ are requested". Note that complex annotations can only appear in rule heads and not in rule bodies.

Definition 2 (AHU-Datalog active rule) *An* AHU-Datalog active rule *is a rule of the form*

$$\alpha_1 A_1 : \mu_1, \ldots, \alpha_k A_k : \mu_k, A_{k+1} : \mu_{k+1}, \ldots, A_w : \mu_w,$$
$$db_1 \diamond A_{w+1} : \mu_{w+1}, \ldots, db_p \diamond A_s : \mu_s, X_1 \diamond A_{s+1} : \mu_{s+1},$$
$$\ldots, X_q \diamond A_n : \mu_n \rightarrow \alpha_{n+1} A_{n+1} : \mu_{n+1},$$
$$\ldots, \alpha_{n+m} A_{n+m} : \mu_{n+m}$$

where (i) $\alpha_i A_i : \mu_i$, $i = 1, \ldots, k$, is a c-annotated or v-annotated (Π^u, Σ, V)-atom; (ii) $\alpha_i A_i : \mu_i$, $i = n+1, \ldots, n+m$, is an annotated (Π^u, Σ, V)-atom; (iii) $A_j : \mu_j$, $j = k+1, \ldots, w$, is a c-annotated or v-annotated $(\Pi^i \cup \Pi^e, \Sigma, V)$-atom; (iv) $db_1 \diamond A_{w+1} : \mu_{w+1}, \ldots, db_p \diamond A_s : \mu_s$ are simple labeled c-annotated or v-annotated $(\Pi^i \cup \Pi^e, \Sigma, V)$-atoms referring to specific databases while $X_1 \diamond A_{s+1} : \mu_{s+1}, \ldots, X_q \diamond A_n : \mu_n$ are simple labeled c-annotated or v-annotated $(\Pi^i \cup \Pi^e, \Sigma, V)$-atoms referring to databases that are not yet specified; (v) $X_1, \ldots, X_q$ are variables in V_{db}; (vi) $\alpha_p A_p \neq \alpha_q A_q$ for $1 \leq p, q \leq k$, $p \neq q$, and $\alpha_s A_s \neq \alpha_t A_t$ for $n+1 \leq s, t \leq n+m$, $s \neq t$.

$\alpha_1 A_1 : \mu_1, \ldots, \alpha_k A_k : \mu_k$ is the event part, $A_{k+1} : \mu_{k+1}, \ldots, A_w : \mu_w, db_1 \diamond A_{w+1} : \mu_{w+1}, \ldots, db_p \diamond A_s : \mu_s, X_1 \diamond A_{s+1} : \mu_{s+1}, \ldots, X_q \diamond A_n : \mu_n$ is the condition part, and $\alpha_{n+1} A_{n+1} : \mu_{n+1}, \ldots, \alpha_{n+m} A_{n+m} : \mu_{n+m}$ is the action part, that cannot be empty. For safety, each variable occurring in the event part (or in its annotation) should also occur in the event-condition part (or in its annotations); moreover $X_1, \ldots, X_q$ must appear as arguments of an extensional atom in $A_{k+1} : \mu_{k+1}, \ldots, A_s : \mu_s$. □

The intuitive meaning of an active rule is: "if the events $\alpha_1 A_1 : \mu_1, \ldots, \alpha_k A_k : \mu_k$ occur and $A_i, i = k+1, \ldots, w$, is true with truth value at most μ_i in the database where the rule is defined, A_{w+1} is true with truth value at most μ_{w+1} in db_1,…,A_s is true with truth value at most μ_s in db_p, A_{s+1} is true with truth value at most μ_{s+1} in the database to which X_1 is instantiated,…, A_n is true with truth value at most μ_n in the database to which X_q is instantiated, then actions $\alpha_{n+1} A_{n+1} : \mu_{n+1}, \ldots, \alpha_{n+m} A_{n+m} : \mu_{n+m}$ have to be executed in the database in which the rule is defined".

AHU-Datalog database An *AHU-Datalog program* over a lattice $\mathcal{T}$ is defined as follows.

Definition 3 (AHU-Datalog program) *An* AHU-Datalog program or database *is identified by one constant of Σ_{db} and is composed of: (i) a set of c-annotated extensional ground atoms (extensional database); (ii) a set of annotated AHU-Datalog deductive rules (intensional database); (iii) a set of AHU-Datalog annotated active rules. A* transaction *for an AHU-Datalog database is any deductive rule without the head.* □

Example 1 *Consider two sensors monitoring the level in the air of some considered substance (*`s1` *and* `s2`*), expressed as a percentage with respect to a quantity considered as dangerous. We are interested in determining if a monitored substance has a presence above some danger thresholds and, based on this information, locally or globally block car circulation. Information concerning the two local sensors can be modeled by two AHU-Datalog databases, with $\mathcal{T} = \mathcal{R}[0,1]$, identified by db_1 and db_2 (see Figure 1) Information concerning the level of a given substance can be traced by the extensional predicate* `sensor`[3]*. We then consider two level thresholds and we use predicate* `d_lv` *to specify whether the danger thresholds concerning a monitored substance have been exceeded. In this case, an over level warning is generated, by inserting facts for predicate o_lv, with different annotations, depending on the exceeded threshold. If in a local database the percentage of both monitored substances is over 90% and a major over level warning has been generated, a local block is issued, through an active rule. Global blocks will be managed by a supervisor (see Example 2). $T = o\_lv(s1) : 0.5$ is a transaction for DB_1, checking whether* `s1` *has generated an over level warning, with percentage at least equal to 65%* ◇

AHU-Datalog System In order to integrate the static and dynamic knowledge deriving from the local databases, a logical mediator, called *supervisor* [7], is used. Unlike [7], we introduce active rules, besides the deductive ones, in the supervisor. Such rules are a simple and strong mechanism to trigger a global reaction to a complex event that requires checking conditions spanning several databases.

Definition 4 (Supervisor) *A supervisor database $S =< SIDB, SAR >$ is an AHU-Datalog program, identified by $s \in \Sigma_{db}$, composed of:*

[3] For example, the annotated atom `sensor(s1):0.5` means that the level of `s1` is the 50% of the maximal admitted quantity in the air.

```
IDB_DB1:  r1:d_lv(s1):1←sensor(s1):0.65,+o_lv(s1):0.5.
          r2:d_lv(s1):1←sensor(s1):0.85,+o_lv(s1):1.
AR_DB1 :  ar1:+o_lv(s1):1,sensor(s1):0.9,db2◇ sensor(s2):0.9→+l_block(s1,s2):1.
IDB_DB2:  r1:d_lv(s2):1←sensor(s2):0.6,db1◇ sensor(s1):0.4,+o_lv(s2):0.5.
          r2:d_lv(s2):1←sensor(s2):0.8,db1◇ sensor(s1):0.7,+o_lv(s2):1.
AR_DB2 :  ar1:+o_lv(s2):1,db1◇ sensor(s1):0.9,sensor(s2):0.9→+l_block(s1,s2):1.
EDB_DB1:  sensor(s1):0.9,o_lv(s1):1.
EDB_DB2:  sensor(s2):0.6.
```

Figure 1. The local databases.

- *a set of AHU-Datalog deductive rules $SIDB$ of the form $s \diamond A_0 : \mu \leftarrow D_1 \diamond A_1 : \mu_1, \ldots, D_k \diamond A_k : \mu_k$ where each D_i, $i = 1, \ldots, k$, is a simple or multiple label, A_0 is a (Π^i, Σ, V)-atom and A_j, $j = 1, \ldots, k$, is a $(\Pi^i \cup \Pi^e, \Sigma, V)$-atom;*
- *a set of AHU-Datalog active rules SAR of the form*

$$db_1 \diamond \alpha_1 A_1 : \mu_1 \ldots, db_k \diamond \alpha_k A_k : \mu_k,$$
$$D_{k+1} \diamond A_{k+1} : \mu_{k+1}, D_{k+1} \diamond A_{k+1} : \mu_{k+1}, \ldots,$$
$$D_n \diamond A_n : \mu_n \rightarrow db_{n+1} \diamond \alpha_{n+1} A_{n+1} : \mu_{n+1},$$
$$\ldots, db_{n+m} \diamond \alpha_{n+m} A_{n+m} : \mu_{n+m}$$

where: db_i, $i = 1, \ldots, k, n+1, \ldots, n+m$, is a simple label, $db_i \neq s$, D_j, $j = k+1, \ldots, n$, is a simple or multiple label, A_j is a $(\Pi^i \cup \Pi^e, \Sigma, V)$-atom, $j = k+1, \ldots, n$, and A_h is a (Π^e, Σ, V)-atom, $h = 1, \ldots, k, n+1, \ldots, n+m$. □

In order to assign a semantics to the supervisor rules, there is the need of specifying how deductive atoms like $D \diamond A : \mu$ have to be defined, when D is a multiple label. The semantics for such atoms is provided by an additional sets of rules, called *combination axioms* (denoted by C). Such rules specify that an atom $D \diamond A : \mu$, $D = \{db_1, ..., db_n\}$, is true if $db_i \diamond A : \mu_i$, $i = 1, ...n$, is true and $\mu = \sqcup\{\mu_1, ..., \mu_n\}$. As side effect, the truth of atom $D \diamond A : \mu$ may request the execution of a set of updates, precisely those requested as side effect by atoms $db_1 \diamond A : \mu_1$,...,$db_n \diamond A : \mu_n$.

Definition 5 (Combination axioms) *Let D be a multiple label, A a deductive atom, and μ an annotation. Atom $D \diamond A : \mu$ is defined by the combination axiom*

$$D \diamond A : \bigsqcup_{db_i \in D} \mu_i \leftarrow \bigwedge_{db_i \in D} db_i \diamond A : \mu_i \text{ where } \mu = \bigsqcup_{db_i \in D} \mu_i. \Box$$

We are now able to define an *AHU-Datalog system.*

Definition 6 (AHU-Datalog system) *An* AHU-Datalog *system SS is a tuple $\langle \{db_1 :: DB_1, \ldots, db_n :: DB_n\}, s :: S, C \rangle$ where: $DB_i = EDB_i \cup IDB_i$, $i = 1, ..., n$ are AHU-Datalog databases, respectively identified by $db_1, \ldots, db_n$, $db_i \in \{\Sigma_{db} \backslash \{s\}\}$; $S =< SIDB, SAR >$ is a supervisor AHU-Datalog program, identified by $s \in \Sigma_{db}$; C is a set of combination axioms. In SS, the extensional part $SSEDB$ is $\langle EDB_1, \ldots, EDB_n \rangle$, the intensional part $SSIDB$ is $\langle IDB_1, \ldots, IDB_n, SIDB, C \rangle$, and the active part $SSAR$ is $\langle AR_1, \ldots, AR_n, SAR \rangle$.* □

In order to ensure encapsulation, transactions to be executed in an AHU-Datalog system cannot contain update atoms. Deductive atoms can be either labeled or unlabeled. Unlabeled atoms require an atom refutation in all the local databases composing the system, while labeled atoms are directed to a specific database. We do not restrict labels in transaction to be constant, thus allowing dynamic cooperation to be established also at transaction level.

Definition 7 (Transaction) *A* transaction[4] *has the form $D_1 \diamond A_1 : \mu_1, \ldots, D_z \diamond A_z : \mu_z, X_1 \diamond A_{z+1} : \mu_{z+1}, \ldots, X_q \diamond A_r : \mu_r$ where $A_1, \ldots, A_r$ are $(\Pi^i \cup \Pi^e, \Sigma, V)$-atoms, $D_1, \ldots, D_z$ are simple or multiple labels, and $X_1, \ldots, X_q$ are variables in V_{db} that must appear as arguments of an extensional atom in $A_1, \ldots, A_z$.* □

Example 2 *Consider Example 1 and assume that car circulation must be globally forbidden, in a partial way, if both* s1 *and* s2 *have a presence above their minor danger level threshold, or globally forbidden, in a total way, if the sensor corresponding to DB_1 has generated a major over-level warning. The detection of partial and total blocks requires a cooperation among the various sensors, therefore this functionality can be assigned to an AHU-Datalog supervisor database. Such database is presented in Figure 2. Information concerning partial and total blocks is represented through predicates* t_block *and* p_block*. In the presented rules, The supervisory database also contains three active rules, avoiding the insertion of information concerning partial blocks if a total block has already been detected. As an example of transaction, T = g_block checks whether some global block has been generated.* ◇

3. AHU-Datalog: the semantics

The semantics of an AHU-Datalog system is defined in three steps, described in the following. In the first step, the semantics of the deductive part of the system is computed, collecting, but not executing, the set of bindings that satisfy

[4] We refer the reader to [3] for the definition of more complex transactions.

```
IDB_S: s◇ g_block(s1,s2):1←db_1◇ d_lv(s1):1,db_2◇ d_lv(s2):1,db_1◇ +p_block(s1):1,db_2◇ +p_block(s2):1.
       s◇ g_block(s1,s2):1←db_1◇ o_lv(s1):1,db_1◇ +t_block(s1):1,db_2◇ +t_block(s2):1.
SAR  : db_i◇ +p_block(s_i):1,db_i◇ t_block(s_i):1→db_i◇ -p_block(s_i):1  i=1,2.
       db_i◇ +t_block(s_i):1,db_i◇ p_block(s_i):1→db_i◇ -p_block(s_i):1  i=1,2,
       db_i◇ +p_block(s_i):1,db_i◇ +t_block(s_i):1→  db_i◇ -p_block(s_i):1  i=1,2.
```

Figure 2. The supervisor database.

the query and the requested updates. In the second step, the semantics of the active part of the system is computed, according to the updates collected in the first step, obtaining a set of update requests. Conflicting update requests are removed by applying a *parametric conflict resolution policy*. In the third step, the updates obtained from the second step are executed against the extensional part of the system.[5]

3.1. Deductive part semantics

In presenting the semantics of the deductive part of a AHU-Datalog system, we consider an *extended Herbrand base* $\overline{\mathcal{CB}_\mathcal{L}}$ composed of *constrained* ground labeled atoms C of the form $C \equiv D \diamond A \leftarrow db_1 \diamond \alpha_1 A_1, \ldots, db_k \diamond \alpha_k A_k$ where $D \diamond A$ is a ground labeled $(\Pi^i \cup \Pi^e, \Sigma, V)$-atom, $db_j \diamond \alpha_j A_j$, $j = 1, \ldots, k$, are ground labeled (Π^u, Σ, V)-atoms. Since the language does not contain function symbols, the extended Herbrand base is a finite set.

An ideal [6] of an upper semilattice[6] $\mathcal{T}$ is any subset K of $\mathcal{T}$ such that: (I) K is downward closed, that is $s \in K, t \leq s \Rightarrow t \in K$; (II) K is closed with respect to the finite least upper bound operation, that is $s, t \in K \Rightarrow s \sqcup t \in K$. If for some $p \in \mathcal{T}$, $K = \{s \mid s \leq p\}$, the ideal is called *principal*. An n-ideal (n-principal) of $\mathcal{T}$ is a tuple composed of n ideals (principals) of $\mathcal{T}$. The set of ideals (principals) of $\mathcal{T}$is denoted by $\mathcal{I}(\mathcal{T})$ ($\mathcal{PI}(\mathcal{T})$) whereas the set of n-ideals (n-principals) is denoted by n-I (n-P).

Definition 8 (Interpretations) *An* Herbrand interpretation *(a* restricted Herbrand interpretation*) (r-interpretation) I is any total map from the extended Herbrand Base* $\overline{\mathcal{CB}_\mathcal{L}}$ *to* $\bigcup_n n$-I *(*$\bigcup_n n$-P*). We assume that the first ideal (principal) of the n-ideal (n-principals) is assigned to the head of the constrained atom whereas all the other ideals (principals) are assigned to labeled update atoms in the body, in the order by which they appear.* □

Since $\mathcal{T}$ is a complete lattice with respect to $\leq$, the order $\leq$ can be extended point to point to interpretations in the obvious way. Moreover, n-ideals (n-principals) of $\mathcal{T}$ can be seen as ideals (principals) of the complete lattice $\mathcal{T}^n$, in which the order $\leq$ is defined component by component. It is now possible to define when a constrained ground atom is true in an interpretation (r-interpretation).

Definition 9 (R-satisfaction) *Let I be an interpretation (r-interpretation),* $< \mu_0, \mu_1, \ldots, \mu_k >$ *a list of c-annotations on a lattice* $\mathcal{T}$, $C \equiv D \diamond A \leftarrow db_1 \diamond \alpha_1 A_1, \ldots, db_k \diamond \alpha_k A_k \in \overline{\mathcal{CB}_\mathcal{L}}$. *I satisfies (r-satisfies) the ground constrained annotated atom* $\mathcal{D} \equiv D \diamond A : \mu_0 \leftarrow db_1 \diamond \alpha_1 A_1 : \mu_1, \ldots, db_k \diamond \alpha_k A_k : \mu_k$ *(*$I \models \mathcal{D}$*) (*$I \models^r \mathcal{D}$*) if and only if* $<\mu_0, \mu_1, \ldots, \mu_k> \in I(C)$ *(*$<\mu_0, \mu_1, \ldots, \mu_k> \leq I(C)$*).* □

Based on the previous concepts, we can now define a fixpoint operator computing the semantics of the deductive part of an AHU-Datalog system.

Definition 10 (Operator R_{SS}) *Let SS an AHU-Datalog system and I an interpretation for SS. The operator* $R_{SS}(I)$*, for any* $A \in \overline{\mathcal{CB}_\mathcal{L}}$*, is such that* $R_{SS}(I)(A) = \sqcup T_{SS}(I)(A)$[7] *where* $T_{SS}(I)(A)$ *is such that, given a constrained atom A and an interpretation I, returns the least ideal containing all the ideals for A that can be deduced from the system starting from I.* □

Operator $R_{SS}(I)(A)$ assigns to A a single principal and is a good candidate for defining a bottom-up AHU-Datalog semantics. Unfortunately, differently from T_{SS}, R_{SS} is not continuous and therefore $R_{SS} \uparrow \omega$ in general is not a fixed point for R_{SS}. However, it can be proved that if all the rules in a system contain only c-annotated atoms or only v-annotated atoms in their bodies, the fixpoint exists. A system satisfying the previous condition is said to be *acceptable*.[8] For acceptable systems, the following result holds.

Theorem 1 *Let SS be an acceptable AHU-Datalog system. Then* $R_{SS} \uparrow \omega = lfp(R_{SS}) = \sqcup(lfp(T_{SS}))$*. The* fixpoint semantics $\mathcal{F}$ *of SS is defined as* $\mathcal{F}(SS) = lfp(R_{SS})$*, where lfp is a shorthand for the least fixpoint.* □

Starting from the semantics of an AHU-Datalog system SS, the semantics of each local database can be easily determined by projecting the semantics on the label of the considered database. Thus, the semantics of the local database $DB_i \in SS$ is defined as $\mathcal{F}(SS)(i) = \{D \diamond A : \mu | D \diamond A : \mu \in \mathcal{F}(SS)\}$.

[5] In order to simplify the presentation of the semantics of an AHU-Datalog system, we assume that all atoms appearing in an AHU-Datalog system are labeled.

[6] A set R is an upper semilattice, with respect to a given ordering, if any pair of elements of R has a least upper bound in R.

[7] $\sqcup T_{SS}(I)(A)$ is a shorthand notation for $\sqcup_{t \in T_{SS}(I)(A)} t$.

[8] See [3] for a more general definition of acceptable systems.

The deductive semantics of a simple transaction T with respect to an AHU-Datalog system SS is defined in the usual way by using the fixpoint operator R_{SS}.

Definition 11 (Query answers) *Let SS be an AHU-Datalog system and T a transaction such that $T = D_1 \diamond A_1 : \mu_1, \ldots, D_n \diamond A_n : \mu_n, X_1 \diamond A_{n+1} : \mu_{n+1}, \ldots, X_p \diamond A_{n+p} : \mu_{n+p}$. Query answers for T in SS, denoted by* Set(T, SS), *are defined as follows:*[9]

$$\begin{aligned}
\mathsf{Set}(T, SS) = \{\langle b, \overline{U}\rangle \mid & \textit{ there exist} \\
& \overline{D}_i \diamond \overline{A}_i : \overline{\mu}_i \leftarrow db_1^i \diamond \alpha_1^i A_1^i : \mu_1^i, \ldots, \\
& \quad db_{k_i}^i \diamond \alpha_{k_i}^i A_{k_i}^i : \mu_{k_i}^i \in \mathcal{F}(SS), i = 1, ..., n+p, \\
& \theta = mgu((D_1 \diamond A_1 : \mu_1, \ldots, D_n \diamond A_n : \mu_n, \\
& \quad X_1 \diamond A_{n+1} : \mu_{n+1}, \ldots, X_p \diamond A_{n+p} : \mu_{n+p}), \\
& \quad\quad (\overline{D}_1 \diamond \overline{A}_1 : \overline{\mu}_1, \ldots, \overline{D}_{n+p} \diamond \overline{A}_{n+p} : \overline{\mu}_{n+p})), \\
& b = eqn(\theta)_{|T}, \overline{U} = \left\{ \overset{+}{\bigcup_{\substack{i=1,\ldots,n+p \\ u=1\ldots,k_i}}} (db_u^i \diamond \alpha_u^i A_u^i : \mu_u^i)\theta \right\}\}
\end{aligned}$$

The set $\mathcal{A} \overset{+}{\cup} \mathcal{B}$ is defined as the usual set union with the following exception: if $db \diamond \alpha A : \mu_1 \in \mathcal{A}$ and $db \diamond \alpha A : \mu_2 \in \mathcal{B}$, then $db \diamond \alpha A : \sqcup\{\mu_1, \mu_2\} \in \mathcal{A} \overset{+}{\cup} \mathcal{B}$ and $db \diamond \alpha A : \mu_1$, $db \diamond \alpha A : \mu_2 \notin \mathcal{A} \overset{+}{\cup} \mathcal{B}$. □

Example 3 *Consider the AHU-Datalog system SS in Example 2 and $T = s \diamond g\_block(s1, s2) : 1$. Figure 3 presents the computation of the deductive part semantics for SS. It is easy to verify that*

$$\begin{aligned}
\mathsf{Set}(T, SS) = \{\langle\{\emptyset\}\{& db_1 \diamond +o\_lv(s1) : 1, db_2 \diamond +o\_lv(s2) : 0.5, \\
& db_1 \diamond +p\_block(s1) : 1, db_2 \diamond +p\_block(s2) : 1, \\
& db_1 \diamond +t\_block(s1) : 1, db_2 \diamond +t\_block(s2) : 1.\}\rangle\}. \diamond
\end{aligned}$$

3.2. Active part semantics

The semantics of the active part of the system is almost similar to that presented for HU-Datalog [2]. Therefore, in the following we just survey the basic aspects, referring the reader to [2, 3] for additional details.

Validity conditions In order to establish when an atom, contained in the event or condition part of an active rule may trigger it, the concept of *restricted-intermediate interpretation* (ri-interpretation) is provided. An ri-interpretation is a function associating with each element of $\mathcal{B}^{\pm}$ a principal of $\mathcal{T}$, where given a system SS, $\mathcal{B}^{\pm} = \mathcal{B} \cup \{db \diamond \alpha A \mid db \diamond A \in \mathcal{B}, pred(A)^{10} \in \Pi^e, \alpha \in \{+, -\}\}$.[11] An ri-interpretation is *consistent* if it does not contain any pair of *conflicting updates*. Two update atoms u_1 and u_2 are conflicting if they require the insertion and the deletion of the same atom in the same database, independently from the considered truth values.

Given an ri-interpretation I and an atom a, the atom is valid in I if $a = db \diamond A : \mu$ and I contains either $D \diamond A : \overline{\mu}$ or $db \diamond +A : \overline{\mu}$, with $\mu \leq \overline{\mu}, db \in D$. On the other hand if $a = a = db \diamond +A : \mu$, I must contain $db \diamond +A : \overline{\mu}$, with $\mu \leq \overline{\mu}$. A similar condition can be given for deletions.

Active rules should take into account the available intensional knowledge when checking conditions, without triggering additional updates that, potentially, would change the state of the database as soon as it is observed. Therefore, we extend the set of active rules with *purified deductive rules* $\widehat{SSIDB}$, obtained from the system deductive rules $SSIDB$ by removing all update atoms appearing in them and reversing the arrow direction, to make them similar to active rules. Given a ri-interpretation I, a rule is fireable if all atoms contained in the event and condition parts are valid in I.

Blocked rule instances Suppose that, given an r-interpretation, more than one rule is fireable. It could happen that the actions to be executed contain some conflicting updates. Such a conflict can always be represented by the pair of conflicting update atoms, together with the set of rule instances generating such updates. It is therefore possible to define a *conflict resolution policy* sel as a total function taking as input an extensional database EDB, a set of AHU-Datalog rules P, a ri-interpretation I, an a conflict c and returning either {insert, delete}, representing which operation (insertion, deletion) has to prevail in the resolution of c, based on the considered system status.

Based on the chosen conflict resolution policy, it is possible to determine which rule instances have to be blocked (denoted by blocked(EDB, P, I, sel)) by considering all rule instances associated with conflicting updates that have been discarded by the conflict resolution policy. We always block an entire rule instance, rather than a single update, so that the set of update requests by the same rule instance exhibits an atomic behavior.

The computation Using the above concepts, given a set of AHU-Datalog rules P, a set of blocked rule instances B, and an ri-interpretation I, we define an immediate consequence operator over ri-interpretations $\Gamma_{P,B}(I)$, similarly to other bottom-up operators defined in the logic programming context. However, differently from them, some rules may not be fired even if their body is valid, due to the

[9] Given a set of bindings b, a transaction T, and a substitution $\theta = \{V_1 \leftarrow t_1, \ldots, V_n \leftarrow t_n\}$, we define $b_{|T} = \{(X = t) \in b \mid X$ occurs in $T\}$ and $eqn(\theta) = \{V_1 = t_1, \ldots, V_n = t_n\}$. With mgu we denote the most general unifier.

[11] $pred(A)$ denotes the predicate on which the atom A is constructed and $\mathcal{B}$ is the set of labeled ground annotated atoms.

$$R^1_{SS}(\emptyset) = \{db_1 \diamond sensor(s1) : 0.9, db_1 \diamond o\_lv(s1) : 1, db_2 \diamond sensor(s2) : 0.6.\}$$

$$\begin{aligned} R^2_{SS}(\emptyset) = R^1_{SS}(\emptyset) \cup \{&db_1 \diamond d\_lv(s1) : 1 \leftarrow db_1 \diamond {+o\_lv}(s1) : 1, db_2 \diamond d\_lv(s2) : 1 \leftarrow db_2 \diamond {+o\_lv}(s2) : 0.5,\\ &s \diamond g\_block(s1, s2) : 1 \leftarrow db_1 \diamond {+t\_block}(s1) : 1, db_2 \diamond {+t\_block}(s2) : 1, \{db_1, db_2\} \diamond sensor(s1) : 0.9, \ldots\} \end{aligned}$$

$$R^3_{SS}(\emptyset) = R^2_{SS}(\emptyset) \cup \{s \diamond g\_block(s1, s2) : 1 \leftarrow db_1 \diamond {+o\_lv}(s1) : 1, db_2 \diamond {+o\_lv}(s2) : 0.5, db_1 \diamond {+p\_block}(s1) : 1, db_2 \diamond {+p\_block}(s2) : 1.\}$$

$$R^4_{SS}(\emptyset) = R^3_{SS}(\emptyset) = \mathcal{F}(SS)$$

Figure 3. The fixpoint computation

blocked set of rules. If P is c-annotated, or the lattice is finite, $\Gamma_{P,B}$ admits a fixpoint, reachable in a finite number of steps.

In general, the application of the function $\Gamma_{P,B}$ to a consistent ri-interpretation does not return a consistent ri-interpretation. Therefore, we cannot compute the semantics of SS as the least fixpoint of $\Gamma_{SS,B}$. We must instead appropriately select rules, that is, we must build a set of blocked rules B such that the least fixpoint of $\Gamma_{SS,B}$ is consistent. Thus, instead of dealing with ri-interpretations, a new operator is defined, dealing with *bi-structures*, defined as pairs composed of a set of blocked rule instances and an ri-interpretation.

Definition 12 (Δ operator) *Given a set of AHU-Datalog rules P, a bi-structure $\langle B, I\rangle$ and a conflict resolution policy* sel, *we let $I^e = \{D \diamond A : \mu \in I \mid pred(A) \in \Pi^e\}$ and*

$$\Delta_{P,\mathsf{sel}}(\langle B, I\rangle) = \begin{cases} \langle B, \Gamma_{P,B}(I)\rangle & \textit{if } \Gamma_{P,B} I \textit{ is consistent;} \\ \langle B \cup \mathsf{blocked}(I^e, S, I, \mathsf{sel}), I^e\rangle & \textit{otherwise} \end{cases} \quad \square$$

If no conflict arises, Δ does not change the blocked rule set B and only the ri-interpretation of the bi-structure is changed by adding the immediate consequences of the non-blocked rules. On the other hand, any conflict is solved via the resolution policy sel and all blocked rule instances are collected. Then, the computation of Δ is started again from the ri-interpretation I^e with the augmented set of blocked rules. I^e represents the "safe" starting point of the new computation since it does not contain atoms whose validity depends on actions of rule instances that are now blocked.

If the program is c-annotated or if the lattice is finite, Δ admits a fixpoint which is reachable in a finite number of steps and can be used to assign the active part semantics.

Theorem 2 *Let P be a set of AHU-Datalog rules,* sel *a conflict resolution policy, I a set of ground annotated extensional atoms. Suppose that either P is c-annotated or the lattice is finite. Then, there exists k such that $\Delta^k_{P,\mathsf{sel}}$ is a fixpoint of $\Delta_{P,\mathsf{sel}}$ and if $\Delta^k_{P,\mathsf{sel}}(\langle B, I\rangle) = \langle B', I'\rangle$, then $I' = lfp_I(\Gamma_{P,B'})$[12] and I' is consistent.* □

3.3. Integrating deductive and active semantics

In order to integrate deductive and active semantics, we define a function that, given an ri-interpretation and the current system state, returns the next state obtained by executing the updates in the ri-interpretation. To be more precise, we define as *observable* a triple $\langle$Ans, EDB, Res$\rangle$ where Ans is a set of bindings, EDB is an extensional database and Res $\in$ {Commit, Abort}. The set of observables is Oss. We first define update incorporation as follows.

Definition 13 (Update incorporation) *Given an ri-interpretation I and a AHU-Datalog extensional database $EDB = EDB_1 \cup \ldots \cup EDB_n$, we define $\alpha I = \{db \diamond A : \overline{\mu} \mid db \diamond \alpha A : \overline{\mu} \in I\}$, $\alpha \in \{+, -\}$ and we let* $\mathsf{incorp}(I, EDB) = (EDB_i \setminus^+ -I) \cup^+ +I$, *where $A \overset{+}{\cup} B$ is defined as in Definition 11 and $A \setminus^+ B$ is defined as the usual set difference with the following exception: if A contains $db \diamond D : \mu_1$ and B contains $db \diamond D : \mu_2$, then: (i) if $\mu_1 < \mu_2$, $A \setminus^{inc} B$ does not contain $db \diamond D : \mu_1$; (ii) if $\mu_1 > \mu_2$, $A \setminus^{inc} B$ contains $db \diamond D : \mu_1$.* □

Update incorporation guarantees that the extensional database always contains just one annotation for each labeled atom. If an atom has to be inserted with a different annotation, the lub of both annotations is used to replace the existing one. On the other hand, a labeled annotated atom is deleted only if the annotation of the atom to be deleted is stronger than or equal to the existing annotation.

Definition 14 (AHU-Datalog semantics) *Given an acceptable AHU-Datalog system SS, a transaction T and a conflict resolution policy* sel, *the semantics of T in SS is defined as* $Sem_{SS,\mathsf{sel}}(T) = \mathcal{S}_{SSIDB,SSAR,\mathsf{sel}}(T)(\langle\emptyset, SSEDB, \mathsf{Commit}\rangle)$ *where function* $\mathcal{S}_{SSIDB,SSAR,\mathsf{sel}}(T) : \mathsf{Oss} \to \mathsf{Oss}$ *is defined as follows:*

$$\mathcal{S}_{SSIDB,SSAR,\mathsf{sel}}(T)(\langle\alpha, \varepsilon, \rho\rangle) = \\ = \begin{cases} \langle\emptyset, \varepsilon, \mathsf{Abort}\rangle & \textit{if } \rho = \mathsf{Abort} \\ \langle\mathsf{Ans}, \mathsf{incorp}(I, \varepsilon), \mathsf{Commit}\rangle & \\ \quad \textit{if } \rho \neq \mathsf{Abort} \textit{ and } \overline{U} \textit{ is ground} & \\ \langle\emptyset, \varepsilon, \mathsf{Commit}\rangle & \textit{otherwise} \end{cases}$$

$$\mathsf{Ans} = \{b_j \mid \langle b_j, \overline{u}_j\rangle \in \mathsf{Set}(T, SS)\}$$
$$\overline{U} = \bigcup\{\overline{u}_j \mid \langle b_j, \overline{u}_j\rangle \in \mathsf{Set}(T, SS)\}$$

[12] $lfp_I(f)$ denotes the least fixpoint of f which is greater than or equal to I.

$\langle B, I\rangle = \Delta^{\omega}_{\Xi,\mathsf{sel}}(\langle\emptyset, \varepsilon\rangle)$
$\Xi = (\widehat{SSIDB} \cup SSAR) \cup \{ \rightarrow db \diamond +A : \mu \mid$
$db \diamond +A : \mu \in \overline{U}\} \cup \{ \rightarrow db \diamond -A : \mu \mid db \diamond -A : \mu \in \overline{U}\}$. □

To compute the semantics of a transaction, first we build the set of answers (Definition 11), obtaining a set of bindings for the variables of the transaction (Ans) and a set of annotated updates ($\overline{U}$) which are requested but which will not necessarily be executed. Then, we gather rules in order to apply the Δ operator (Definition 12). Such a set of rules (Ξ) contains the purified deductive rules ($\widehat{SSIDB}$), the rules in SSAR and the annotated updates requested from the deductive part ($\overline{U}$), represented as rules with neither event nor condition. The updates in $\overline{U}$ become the initial events to which the active rules in AR have to react.

To obtain the set of updates to be executed, we apply the Δ operator starting from an empty set of blocked rule instances and from the extensional database as initial ri-interpretation ($\langle\emptyset, SSEDB\rangle$). A fixpoint of $\Delta_{\Xi,\mathsf{sel}}$ is reached in a finite number of steps. The new state of the database is then computed by incorporating the updates belonging to I in the current state of the database, following Definition 13.[13]

Example 4 *Consider the execution of transaction* $T = s \diamond g\_block(s1, s2) : 1$ *against the system presented in Example 3. In order to compute* $Sem_{SS,\mathsf{sel}}(T)$*, suppose that our selection function privileges deletions. Now, let* $\Xi = \widehat{SSIDB} \cup SSAR \cup USet(T, SS)$*, where* $USet(T, SS)$ *represents the set of rules generated from the updates contained in* $\mathsf{Set}(T, SS)$*, presented in Example 3. In order to compute the active semantics, we must compute the fixpoint of* $\Delta_{\Xi,\mathsf{sel}}(\langle\emptyset, EDBSS\rangle)$*. We let* $I^e = SSEDB$ *and start by computing* $\Gamma_{\Xi,\emptyset}(I^e)$*:*[14]

$\Gamma_{\Xi,\emptyset}(I^e) = I_1 = I^e \cup \{\ db_1\diamond\, d\_lv(s1){:}1,\ db_2\diamond\, d\_lv(s2){:}1,$
$s\diamond\, g\_block(s1,s2){:}1,\ \{db_1, db_2\}\diamond\, sensor(s1){:}0.9, \ldots,\ db_1\diamond$
$+o\_lv(s1){:}1,\ db_2\diamond\, +o\_lv(s2){:}0.5,\ db_1\diamond\, +p\_block(s1){:}1,\ db_2\diamond$
$+p\_block(s2){:}1,\ db_1\diamond\, +t\_block(s1){:}1,\ db_2\diamond\, +t\_block(s2){:}1\ \}$

Since I_1 *is consistent, we let* $\Delta_{\Xi,\mathsf{sel}}(\langle\emptyset, SSEDB\rangle) = \langle\{\}, I_1\rangle$*. The computation continues by computing* $\Gamma_{\Xi,\emptyset}(I_1)$*:*

$\Gamma_{\Xi,\emptyset}(I_1) = I_2 = I_1 \cup \{\ db_1\diamond\, -p\_block(s1){:}1, db_2\diamond$
$-p\_block(s2){:}1\ \}$

At this step, two conflicts are generated, since there is a request of both inserting and deleting atoms $db_1\diamond$ `p_block(s1):1` *and* $db_2\diamond$ `p_block(s2):1`*. The rule instances generating the conflict for* $db_1\diamond$ `p_block(s1):1` *are* $r_1 = \rightarrow\ db_1\diamond$ `+p_block(s1):1` *and* $r_2 = db_1\diamond$ `+p_block(s1):1,` $db_1\diamond$ `+t_block(s1):1,`$\rightarrow db_1\diamond$ `-p_block(s1):1` *whereas the rule instances generating the conflict for* $db_2\diamond$ `p_block(s2):1` *are* $r_3 = \rightarrow\ db_2\diamond$ `+p_block(s2):1` *and* $r_4 = db_2\diamond$ `+p_block(s2):1,` $db_2\diamond$ `+t_block(s2):1,`$\rightarrow db_2\diamond$ `-p_block(s2):1`. *Since we privilege deletions, we set* $B' = \mathsf{blocked}(I^e, \Xi, I_2, \mathsf{sel}) = \{(r_1, \emptyset), (r_3, \emptyset)\}$ *and we obtain* $\Delta_{\Xi,\mathsf{sel}}(\langle\{\}, I_1\rangle) = \langle B', I^e\rangle$*. We have now to compute* $\Gamma_{\Xi,B'}(I^e)$*:*

$\Gamma_{\Xi,B'}(I^e) = I'_1 = I^e \cup \{\ db_1\diamond\, d\_lv(s1){:}1,\ db_2\diamond\, d\_lv(s2){:}1,$
$s\diamond\, g\_block(s1,s2){:}1,\ \{db_1, db_2\}\diamond\, sensor(s1){:}0.9, \ldots \quad \ldots$
$\ldots,\ db_1\diamond\, +o\_lv(s1){:}1,\ db_2\diamond\, +o\_lv(s2){:}0.5,\ db_1\diamond$
$+t\_block(s1){:}1,\ db_2\diamond\, +t\_block(s2){:}1\ \}$

Additional iterations do not generate any new constrained atom, thus $\Delta^{\omega}_{\Xi,\mathsf{sel}}(\langle\{\}, SSEDB\rangle) = \langle B', I'_1\rangle$ *and* $Sem_{SS,\mathsf{sel}}(T) = < \emptyset, \mathsf{incorp}(I'_1, SSEDB), \mathsf{Commit} >$*, where:*

$\mathsf{incorp}(I'_2, SSEDB) = \{\ db_1\diamond\, sensor(s1){:}0.9,\ db_1\diamond$
$o\_lv(s1){:}1,\ db_2\diamond\, sensor(s2){:}0.6,\ db_2\diamond\, o\_lv(s2){:}0.5,\ db_1\diamond$
$t\_block(s1){:}1,\ db_2\diamond\, t\_block(s2){:}1\ \}$

4. Conclusions and future work

In this paper, we defined a logical framework for modeling queries, updates, update propagation and cooperation against a set of amalgamated knowledge bases. This work can be extended in several ways. Important research topics concern the definition and analysis of transaction execution properties, the extension of the framework with negation, and the usage of our model as a foundation for agent technology dealing with intelligent information management.

References

[1] E. Bertino, B. Catania, V. Gervasi, and A. Raffaetà. Active-U-Datalog: Integrating Active Rules in a Logical Update Languages. In *LNCS 1472: Transactions and Change in Logic Databases*, pages 107–133. Springer Verlag, 1998.

[2] E. Bertino, B. Catania, V. Gervasi, and A. Raffaetà. A Logical Approach to Cooperative Information Systems. *Journal of Logic Programming*, 43(1):15–48, 2000.

[3] E. Bertino, B. Catania, and P. Perlasca. Introducing Cooperation and Actions in Amalagamated Knowledge Bases (Extended Version). Technical report, University of Milano, Italy, 2000.

[4] E. Bertino, M. Martelli, and D. Montesi. Transactions and Updates in Deductive Databases. *IEEE Transactions on Knowledge and Data Engineering*, 9(5):784–797, 1997.

[5] G. Gottlob, G. Moerkotte, and V. Subrahmanian. The PARK Semantics for Active Rules. In *LNCS 1057: Proc. of the Fifth Int. Conf. on Extending Database Technology*, pages 35–55. Springer Verlag, 1996.

[6] M. Kifer and V. Subrahmanian. Theory of Generalized Annotated Logic Programming and its Applications. *Journal of Logic Programming*, 12(4):335–368, 1992.

[7] V. Subrahmanian. Amalgamating Knowledge Bases. *ACM Transactions on Database Systems*, 19(2):291–331, 1994.

[8] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25:38–49, 1992.

[13] Note that, unlike U-Datalog [4], if non-ground updates are requested, we do not abort but we commit the transaction by discarding all required changes.

[14] In the following, to simplify notation, we use a set-based representation of ri-interpretations.