UDC 004.896, DOI:10.2298/CSIS100209015D

A Layered Rule-Based Architecture for Approximate Knowledge Fusion*

Barbara Dunin-Kęplicz[1,2], Linh Anh Nguyen[1], and Andrzej Szałas[1,3]

[1] Institute of Informatics, University of Warsaw
Banacha 2, 02-097 Warsaw, Poland
{keplicz,nguyen,andsz}@mimuw.edu.pl
[2] Institute of Computer Science, Polish Academy of Sciences
Ordona 21, 01-237 Warsaw, Poland
[3] Dept. of Computer and Information Science, Linköping University
SE-581 83 Linköping, Sweden

Abstract. In this paper we present a framework for fusing approximate knowledge obtained from various distributed, heterogenous knowledge sources. This issue is substantial in modeling multi-agent systems, where a group of loosely coupled heterogeneous agents cooperate in achieving a common goal. In paper [5] we have focused on defining general mechanism for knowledge fusion. Next, the techniques ensuring tractability of fusing knowledge expressed as a Horn subset of propositional dynamic logic were developed in [13,16].

Propositional logics may seem too weak to be useful in real-world applications. On the other hand, propositional languages may be viewed as sublanguages of first-order logics which serve as a natural tool to define concepts in the spirit of description logics [2]. These notions may be further used to define various ontologies, like e.g. those applicable in the Semantic Web. Taking this step, we propose a framework, in which our Horn subset of dynamic logic is combined with deductive database technology. This synthesis is formally implemented in the framework of HSPDL architecture. The resulting knowledge fusion rules are naturally applicable to real-world data.

Keywords: knowledge fusion, multi-agent systems, approximate reasoning, rule-based systems.

1. Introduction

In this paper we investigate a framework for fusing approximate knowledge obtained from various distributed, heterogenous knowledge sources. This issue is substantial in modeling multiagent systems, where a group of loosely coupled heterogeneous and autonomous agents cooperate in achieving a common

* Supported by the Polish MNiSW grant N N206 399334 and by grants from the Swedish Foundation for Strategic Research (SSF), Strategic Research Center MOVIII and the Swedish Research Council (VR) Linnaeus Center CADICS.

goal. Information exchange, leading ultimately to knowledge fusion, is a natural and vital ingredient of cooperation, coordination and negotiations, which constitute paradigmatic activities of advanced multiagent systems. This is particularly visible, when environment model an agent has access to and from which it can reason is assumed to be limited by inherent perceptual limitations. There are many reasons why approximate approaches are needed in this context, including the following:

– sensor measurements, video streams, etc. are always approximate in their very nature – in fact one can never expect precise, accurate data from such sources
– even in the case of idealized perfect perception agents may draw substantially different conclusions, based on their circumstances. For example, due to different camera angles and light reflections one agent may draw a conclusion that a given object is red while another agent may classify the object to be brown. As this is highly contextual, probabilistic sensor models may be of little help and qualitative approximate reasoning may be needed.

As discussed in [7], in the past several years attempts have been made to broaden the traditional definition of data fusion as state estimation via aggregation of multiple sensor streams. There is still a need to broaden the definition to include the many additional processes used in all aspects of data and information fusion identified in large scale distributed systems. One of the more successful proposals for providing a framework and model for this broadened notion of data fusion is the data fusion model [33] and its revisions [29,21]. In [29] for example, data fusion is defined as "the process of combining data or information to estimate or predict entity states" and the data fusion problem "becomes that of achieving a consistent, comprehensive estimate and prediction of some relevant portion of the world state".

There is a variety of possibilities to model approximate knowledge [6,11,10,9,20,23,28,34,35,36]. In this presentation we have chosen a generalization of rough sets and relations [27]. In contrast to [27] where only equivalence relations are considered, our approach depends on allowing arbitrary similarity relations. In order to construct approximations, a covering of the underlying domain by similarity-based neighborhoods is used here. Resulting approximations have been shown to be useful in applications requiring approximate knowledge structures [6].

There are many choices as to possible constraints to be placed on the similarity relation used to define approximations. The basic requirement is that the lower approximation is included in the upper one of any set/relation. This is equivalent to the seriality of similarity relations (see [8]), which we set as the only requirement. On the other hand, one might not want the relation be transitive since similar objects do not naturally chain in a transitive manner (see, e.g., [4,14,6,22,31]). Similarity measures on sets that could be adapted to the context of approximate reasoning we deal with have been intensively studied in the area of computer vision and fuzzy sets (see, e.g., [17,32]).

The focus of this paper is approximate knowledge fusion based on the idea of approximations. Our starting point is [5], where a framework for knowledge fusion in multi-agent systems is introduced. Agent's individual perceptual capabilities are represented by similarity relations, further aggregated to express joint capabilities of teams. The aggregation expressing a shift from individual to social level of agents' activity has been formalized by means of dynamic logic. The approach of [5], as using the full propositional dynamic logic, does not guarantee tractability of reasoning [18]. To overcome this constraint we adapt the techniques of [24,25,26] to provide an engine for tractable approximate database querying restricted to a Horn fragment of serial propositional dynamic logic, denoted by HSPDL.

1.1. Contributions of the Paper

In this paper we substantially extend our work presented in [13], where we have concentrated on techniques allowing one to query HSPDL databases in a tractable manner. Propositional logics have a very limited expressivity and may seem too weak to be useful in real-world applications. For example, one cannot express rules using even very basic arithmetics, like in rules (12) and (13) of Section 5. On the other hand, propositional languages may be viewed as sublanguages of first-order logics which serve as a natural tool to define concepts in the spirit of description logics [2]. Additionally, allowing one to query other modules of the system, not necessarily propositional (but returning Boolean values), provides a powerful tool. Taking this step, we propose a framework, in which our Horn subset of dynamic logic is combined with deductive database technology [1], allowing one to express an advanced knowledge fusion applicable in real-world data.

The synthesis of the two formalisms naturally leads to a layered architecture, with the lowest layer containing raw data and basic knowledge structures, the middle one allowing to express rules specifying knowledge fusion, new concepts and their approximations, and the upper level providing the resulting knowledge database. We provide this architecture with both the formal semantics and tractable querying machinery. This makes the framework a pragmatic, rich formalism to be directly used in the chosen application domain.

The framework we propose can also be adapted to other propositional logics designed as a specification and computation tool, e.g., for multiagent systems as well as other robotics and software systems. This bridges the gap between propositional languages and real-world, usually non-propositional data.

1.2. The HSPDL Architecture

There are three main layers of HSPDL architecture (see Figure 1):

– the lower layer consisting of perception data, knowledge databases, results of classifiers, etc.

– the middle layer containing HSPDL rules to define new concepts and their approximations
– the upper layer using the data resulting from the lower layers, i.e., fused concepts and approximations, to define new advanced rules and obtain new facts.

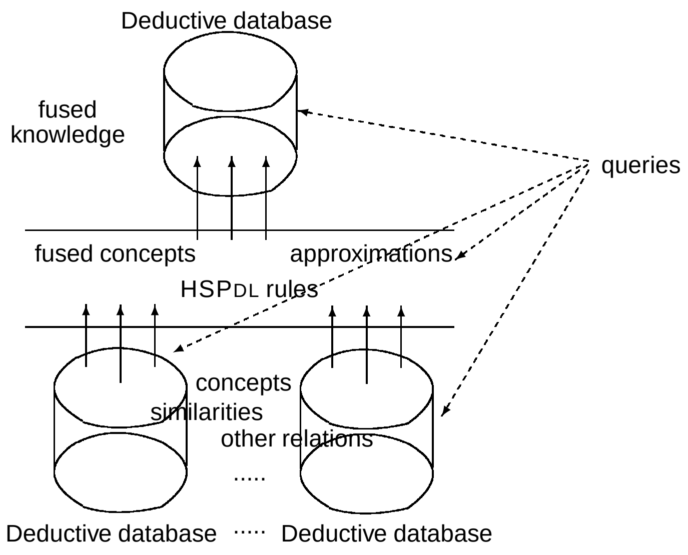

Fig. 1. The HSPDL architecture.

The architecture is highly independent of a particular technology. They can be founded, e.g., on SQL databases or any other software systems.[4] We only make the following assumptions:

– the lower layer is conceptually a database storing relations (but, as indicated, not necessarily a relational database)
– the lower layer provides a programming interface allowing:
  • the middle layer to ask queries about concepts (unary relations) and similarity relations (binary relations)
  • the upper layer to ask queries about any relations represented in the lower layer.
– the lower (respectively, upper) layer computes answers to queries in time polynomial in the size of its domain.

The context in which the middle layer rules appear may be very expressive. It might be the case that all tractable knowledge fusion procedures become

[4] In this paper we shall mainly focus on deductive databases technology using Datalog as its query language (see, e.g., [1]).

expressible without using HSPDL. However, we insist that the introduction of HSPDL rules in the middle layer is both well motivated and intuitively appealing. In the first place, some middle layer constructs are not expressible in many database technologies, including standard SQL and Datalog. Otherwise, the resulting rules happen to be indirect and lead to programs difficult to understand and analyze, while HSPDL rules are fully declarative,

Layered architectures in similar but substantially different contexts have been considered, e.g., in [15,31].

1.3. The Paper Structure

The paper is structured as follows. In Section 2 we recall the serial propositional dynamic logic. Computational aspects of its Horn fragment HSPDL are discussed in Section 3. Section 4 is devoted to combining HSPDL with Datalog. Section 5 illustrates possible applications of the introduced framework on an example. Finally, Section 6 concludes the paper.

2. Serial Propositional Dynamic Logic

2.1. Language and Semantics of SPDL

Let us define *serial propositional dynamic logic* (SPDL). The key idea is to provide calculus on similarity relations rather than on programs. This somehow unusual move allows us to reason about similarities using the whole apparatus of dynamic logic, where "programs" are replaced by similarity relations.

Let SNames denote the set of *similarity relation symbols*, CNames denote the set of *concept names* (i.e., propositions), and INames denote the set of *individuals*. We assume that INames is finite and non-empty. We use letters like σ to indicate elements of SNames, use letters like p, q to indicate elements of CNames, and use letters like a, b, c to indicate elements of INames.

Definition 1. *Formulas* and *similarity expressions* (of SPDL) are respectively defined by the two following BNF grammar rules:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \to \varphi \mid \langle\alpha\rangle\varphi \mid [\alpha]\varphi$$
$$\alpha ::= \sigma \mid \alpha;\alpha \mid \alpha \cup \alpha \mid \alpha^* \mid \varphi?$$

Operator $;$ is called the *composition*, $\cup$ the *union*, * the *iteration* and $\varphi?$ the *test operator*. $\lhd$

We use letters like α, β to denote similarity expressions, and use letters like φ, ψ to denote formulas.

Intuitively,

- $\alpha_1;\alpha_2$ stands for a set-theoretical composition of relations α_1 and α_2
- $\alpha_1 \cup \alpha_2$ stands for set-theoretical union of relations α_1 and α_2
- α^* stands for the reflexive and transitive closure of α

– $\varphi?$ stands for the test operator.

Operators $\langle\alpha\rangle$ and $[\alpha]$ are modal operators of the dynamic logic with the following intended meaning:

– $\langle\alpha\rangle\varphi$: "there is an object similar w.r.t. α to a given object and satisfying formula φ"
– $[\alpha]\varphi$: "all objects similar w.r.t. α to a given object satisfy φ".

The following definitions naturally capture these intuitions. Observe, however, that rather than possible worlds or states, objects are used as elements of domains of Kripke structures.

Definition 2. A *Kripke structure* is a pair $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ is a non-empty set of *objects*, and $\cdot^{\mathcal{I}}$ is an interpretation function that maps each individual a to an element $a^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, each concept name p to a subset $p^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and each similarity relation symbol σ to a binary relation $\sigma^{\mathcal{I}}$ on $\Delta^{\mathcal{I}}$. $\triangleleft$

The interpretation function is extended for all formulas and similarity expressions as follows:

$$\begin{array}{lll}
\top^{\mathcal{I}} = \Delta^{\mathcal{I}} & (\neg\varphi)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus \varphi^{\mathcal{I}} & \\
(\varphi \wedge \psi)^{\mathcal{I}} = \varphi^{\mathcal{I}} \cap \psi^{\mathcal{I}} & (\varphi \vee \psi)^{\mathcal{I}} = \varphi^{\mathcal{I}} \cup \psi^{\mathcal{I}} & (\varphi \to \psi)^{\mathcal{I}} = (\neg\varphi \vee \psi)^{\mathcal{I}}
\end{array}$$

$$\begin{array}{l}
(\langle\alpha\rangle\varphi)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y\, [\alpha^{\mathcal{I}}(x,y) \wedge \varphi^{\mathcal{I}}(y)]\} \\
([\alpha]\varphi)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y\, [\alpha^{\mathcal{I}}(x,y) \to \varphi^{\mathcal{I}}(y)]\}
\end{array}$$

$$\begin{array}{lll}
\multicolumn{3}{l}{(\alpha;\beta)^{\mathcal{I}} = \alpha^{\mathcal{I}} \circ \beta^{\mathcal{I}} = \{(x,y) \mid \exists z\, [\alpha^{\mathcal{I}}(x,z) \wedge \beta^{\mathcal{I}}(z,y)]\}} \\
(\alpha \cup \beta)^{\mathcal{I}} = \alpha^{\mathcal{I}} \cup \beta^{\mathcal{I}} & (\alpha^*)^{\mathcal{I}} = (\alpha^{\mathcal{I}})^* & (\varphi?)^{\mathcal{I}} = \{(x,x) \mid \varphi^{\mathcal{I}}(x)\}.
\end{array}$$

We sometimes write $\mathcal{I}, x \models \varphi$ to denote $x \in \varphi^{\mathcal{I}}$. For a set Γ of formulas, we write $\mathcal{I}, x \models \Gamma$ to denote that $\mathcal{I}, x \models \varphi$ for all $\varphi \in \Gamma$. If $\mathcal{I}, x \models \Gamma$ for all $x \in \Delta^{\mathcal{I}}$ then we call $\mathcal{I}$ a *model of* Γ. If $\varphi^{\mathcal{I}} = \Delta^{\mathcal{I}}$ then we say that φ is valid in $\mathcal{I}$.

When dealing with the data complexity of the instance checking problem, without loss of generality we can assume that both the sets SNames and CNames are finite and fixed.

Definition 3. The *size of a Kripke structure* $\mathcal{I}$ is defined to be

$$|\Delta^{\mathcal{I}}| + \Sigma_{p \in \mathrm{CNames}} |p^{\mathcal{I}}| + \Sigma_{\sigma \in \mathrm{SNames}} |\sigma^{\mathcal{I}}|.$$

The *length of a formula* is the number of symbols occurring in it. The *size of a set of formulas* is defined to be the sum of the lengths of its formulas. $\triangleleft$

Lemma 1. *Given a Kripke structure $\mathcal{I}$ with domain of size n and a formula φ with length m, the set $\varphi^{\mathcal{I}}$ can be computed in $O(m \times n^3)$ steps.*

Proof. Just notice that the complexity of computing the transitive closure of a binary relation is $O(n^3)$ (see, e.g., [3]). $\triangleleft$

For every $\sigma \in \mathrm{SNames}$, we adopt the axioms

$$[\sigma]\varphi \to \langle\sigma\rangle\varphi \tag{1}$$

(or $\langle\sigma\rangle\top$, equivalently). It is well known (see, e.g., [8,30]) that (1) corresponds to the *seriality property*:

$$\forall x \exists y\ \sigma^{\mathcal{I}}(x, y). \tag{2}$$

Therefore we have the following definition.

Definition 4. By an *admissible interpretation for* SPDL we understand any Kripke structure $\mathcal{I}$ with all similarities $\sigma \in \mathrm{SNames}$ satisfying (2). We call such Kripke structures *serial*. $\lhd$

Note that we do not require a serial Kripke structure to satisfy the seriality condition $\forall x \exists y\, \alpha^{\mathcal{I}}(x, y)$ for every similarity expression α. This condition holds when α does not contain the test operator, but does not hold, e.g., for $\alpha = ((\neg\top)?)$.

2.2. Expressing Approximations in SPDL

Let us now explain how SPDL is used as a query language involving approximate concepts. First, observe that interpretations assign sets of objects to formulas. Therefore, it is natural to identify any formula with a query selecting all objects satisfying this formula.

In order to explain the role of similarities and modal operators, let us first recall the notion of approximations.

Definition 5. Let Δ be a set of objects and α be a similarity expression representing a serial binary relation on Δ. For $a \in \Delta$, by the *neighborhood of a w.r.t. α*, we understand the set of elements similar to a : $n^{\alpha} \stackrel{\text{def}}{=} \{b \in \Delta \mid \alpha(a, b)\}$.

For $A \subseteq \Delta$, the *lower and upper approximations of A w.r.t. α*, denoted respectively by A^{+}_{α} and $A^{\oplus}_{\alpha}$, are defined by

$$A^{+}_{\alpha} = \{a \in \Delta \mid n^{\alpha}(a) \subseteq A\}$$
$$A^{\oplus}_{\alpha} = \{a \in \Delta \mid n^{\alpha}(a) \cap A \neq \emptyset\}.$$ $\lhd$

The meaning of those approximations is illustrated in Figure 2. Intuitively, assuming that the perception of an agent is modeled by similarity expression α,

– $a \in A^{+}_{\alpha}$ means that all objects indiscernible from a are in A
– $a \in A^{\oplus}_{\alpha}$ means that there are objects indiscernible from a which are in A.

Note that seriality guarantees that the lower approximation of a set is included in its upper approximation. In fact, this is the weakest requirement regarding approximations. The following is often desirable in many applications

$$A^{+}_{\alpha} \subseteq A \subseteq A^{\oplus}_{\alpha}, \tag{3}$$

as, in fact, shown in Figure 2. This property corresponds to the reflexivity of the similarity relation expressed by α (see, e.g., [8,37,30]) and guarantees the following:

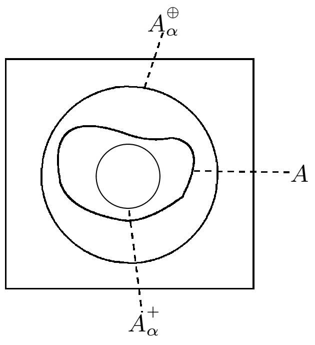

Fig. 2. Lower approximation A^{+}_{α} and upper approximation $A^{\oplus}_{\alpha}$ of a set A.

- $a \in A^{+}_{\alpha}$ means that, from the point of view of the agent, a surely is in A, since all objects indiscernible from a are in A
- $a \in A^{\oplus}_{\alpha}$ means that, from the point of view of the agent, a possibly is in A, since there are objects indiscernible from a which are in A.

Unfortunately, in some applications the set A is given solely via its approximations, so constraints (3) cannot be checked automatically. This is often the case of vague concepts lack precise definitions or lead to definitions unacceptable in applications due to its complexity or other issues. For example one could define a concept of a “dog” via genetic code what is not that much of help when classifying dogs in everyday life. Also, machine learned concepts are often approximated, as e.g., in version spaces (see [12]).

As an immediate consequence of Definitions 5 and 2 we have that:

$$[\alpha]A \text{ expresses the lower approximation of } A \text{ w.r.t. } \alpha\text{, i.e., } A^{+}_{\alpha}, \tag{4}$$

$$\langle\alpha\rangle A \text{ expresses the upper approximation of } A \text{ w.r.t. } \alpha\text{, i.e., } A^{\oplus}_{\alpha}. \tag{5}$$

Remark 1. In the view of (4) and (5), axiom (1) expresses the property that the lower approximation of a set A w.r.t. any similarity expression α is included in its upper approximation. As indicated before, axiom (1) is equivalent to seriality expressed by (2). This justifies seriality to be the key requirement based on approximations. $\triangleleft$

2.3. The Horn Fragment HSPDL

In order to express tractable queries we restrict the query language to the Horn fragment HSPDL, defined below.

Definition 6. *Positive formulas* (of PDL), φ_{pos}, are defined by the following BNF grammar:

$$\begin{aligned}
\varphi_{pos} &::= \top \mid p \mid \varphi_{pos} \land \varphi_{pos} \mid \varphi_{pos} \lor \varphi_{pos} \mid \langle\alpha_{pos_\Diamond}\rangle\varphi_{pos} \mid [\alpha_{pos_\Box}]\varphi_{pos} \\
\alpha_{pos_\Diamond} &::= \sigma \mid \alpha_{pos_\Diamond} ; \alpha_{pos_\Diamond} \mid \alpha_{pos_\Diamond} \cup \alpha_{pos_\Diamond} \mid \alpha^{*}_{pos_\Diamond} \mid \varphi_{pos}? \\
\alpha_{pos_\Box} &::= \sigma \mid \alpha_{pos_\Box} ; \alpha_{pos_\Box} \mid \alpha_{pos_\Box} \cup \alpha_{pos_\Box} \mid \alpha^{*}_{pos_\Box} \mid (\neg\, \varphi_{pos})?
\end{aligned}$$

HSPDL *program clauses*, φ_{prog}, are defined by the following BNF grammar:[5]

$$\begin{aligned}
\varphi_{prog} &::= \top \mid p \mid \varphi_{pos} \to \varphi_{prog} \mid \varphi_{prog} \land \varphi_{prog} \mid \langle \alpha_{prog_\diamond} \rangle \varphi_{prog} \mid [\alpha_{prog_\Box}]\varphi_{prog} \\
\alpha_{prog_\diamond} &::= \sigma \mid \alpha_{prog_\diamond} ; \alpha_{prog_\diamond} \mid \varphi_{prog}? \\
\alpha_{prog_\Box} &::= \sigma \mid \alpha_{prog_\Box} ; \alpha_{prog_\Box} \mid \alpha_{prog_\Box} \cup \alpha_{prog_\Box} \mid \alpha^*_{prog_\Box} \mid \varphi_{pos}?
\end{aligned}$$

An HSPDL *logic program* is a finite set of HSPDL program clauses. The *Horn fragment* HSPDL for the problem of checking whether $\langle \mathcal{P}, \mathcal{A} \rangle \models_s \varphi(a)$ consists of HSPDL logic programs for $\mathcal{P}$ and positive formulas for φ. $\lhd$

Example 1. The following formulas are HSPDL program clauses:

$$\begin{aligned}
&p \land q \land r \to s \\
&[\sigma_1]p \land \langle \sigma_2 \rangle q \to \langle \sigma_3 \rangle (r \land [\sigma_4]s) \\
&[(\sigma_1 \cup \sigma_2)^*]\big(\langle (\sigma_3 \cup \sigma_4)^* \rangle (p \lor q) \to [\sigma_3]\langle \sigma_4 \rangle r\big),
\end{aligned}$$

while the following formulas are not:

$$\begin{aligned}
&p \land q \to r \lor s \\
&p \to \langle \sigma_1 \lor \sigma_2 \rangle q \\
&p \to \langle \sigma^* \rangle q.
\end{aligned}$$

$\lhd$

Let us now formally link SPDL with databases.

Definition 7.

- A *concept assertion* is an expression of the form $p(a)$, where p is a concept name and a is an individual. A *similarity assertion* is an expression of the form $\sigma(a,b)$, where σ is a similarity relation symbol and a, b are individuals.[6] An *ABox* is a finite set of concept assertions and similarity assertions.[7] The *size of an ABox* is the number of its assertions.
- Given a Kripke structure $\mathcal{I}$ and an ABox $\mathcal{A}$, we say that $\mathcal{I}$ is a *model of* $\mathcal{A}$, denoted by $\mathcal{I} \models \mathcal{A}$, if $a^\mathcal{I} \in p^\mathcal{I}$ for every concept assertion $p(a) \in \mathcal{A}$ and $(a^\mathcal{I}, b^\mathcal{I}) \in \sigma^\mathcal{I}$ for every similarity assertion $\sigma(a,b) \in \mathcal{A}$.
- Given an HSPDL logic program $\mathcal{P}$ and an ABox $\mathcal{A}$, we call the pair $\langle \mathcal{P}, \mathcal{A} \rangle$ an HSPDL *database*, with $\mathcal{A}$ as the extensional database and $\mathcal{P}$ as the intensional part. An SPDL *model of* $\langle \mathcal{P}, \mathcal{A} \rangle$ is a serial Kripke structure that is a model of both $\mathcal{P}$ and $\mathcal{A}$.
- Let $\langle \mathcal{P}, \mathcal{A} \rangle$ be an HSPDL database, φ be a positive formula, and a be an individual. We say that a *has the property* φ *w.r.t.* $\langle \mathcal{P}, \mathcal{A} \rangle$ *in* SPDL (or $\varphi(a)$ is a logical consequence of $\langle \mathcal{P}, \mathcal{A} \rangle$ in SPDL), denoted by $\langle \mathcal{P}, \mathcal{A} \rangle \models_s \varphi(a)$, if $a^\mathcal{I} \in \varphi^\mathcal{I}$ for every SPDL model $\mathcal{I}$ of $\langle \mathcal{P}, \mathcal{A} \rangle$. $\lhd$

By the *instance checking* problem for HSPDL we mean the problem of checking whether $\langle \mathcal{P}, \mathcal{A} \rangle \models_s \varphi(a)$. The *data complexity* of this problem is measured when $\mathcal{P}$, φ and a are fixed (and compose a query), while $\mathcal{A}$ varies as input data.

[5] Notice the two occurrences of φ_{pos} in the grammar. We do not allow formulas of the form $\langle \alpha \cup \beta \rangle \varphi$ or $\langle \alpha^* \rangle \varphi$ to be HSPDL program clauses because they cause non-determinism.

[6] Similarity assertions correspond to role assertions of description logic.

[7] In [19], such an ABox is said to be *extensionally reduced*.

3. Computational Aspects of HSPDL

3.1. Ordering Kripke Structures

To construct least models for HSPDL we need the following definitions.

Definition 8. A Kripke structure $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ is said to be *less than or equal to* $\mathcal{I}' = \langle \Delta^{\mathcal{I}'}, \cdot^{\mathcal{I}'} \rangle$, denoted by $\mathcal{I} \leq \mathcal{I}'$, if for every positive formula φ and every individual a, $a^{\mathcal{I}} \in \varphi^{\mathcal{I}}$ implies $a^{\mathcal{I}'} \in \varphi^{\mathcal{I}'}$. $\triangleleft$

Definition 9. Given Kripke structures $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ and $\mathcal{I}' = \langle \Delta^{\mathcal{I}'}, \cdot^{\mathcal{I}'} \rangle$ and a binary relation $r \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}'}$, we say that $\mathcal{I}$ *is less than or equal to* $\mathcal{I}'$ *w.r.t.* r, denoted by $\mathcal{I} \leq_r \mathcal{I}'$, if the following conditions hold for every individual a, every similarity relation symbol σ, and every concept name p :

1. $r(a^{\mathcal{I}}, a^{\mathcal{I}'})$
2. $\forall x, x', y \ \big[[\sigma^{\mathcal{I}}(x,y) \land r(x,x')] \to \exists y' \, [\sigma^{\mathcal{I}'}(x',y') \land r(y,y')]\big]$
3. $\forall x, x', y' \ \big[[\sigma^{\mathcal{I}'}(x',y') \land r(x,x')] \to \exists y \, [\sigma^{\mathcal{I}}(x,y) \land r(y,y')]\big]$
4. $\forall x, x' \ [r(x,x') \to (x \in p^{\mathcal{I}} \to x' \in p^{\mathcal{I}'})]$. $\triangleleft$

In Definition 9, the first three conditions state that r is a kind of bisimulation between the *frames* of $\mathcal{I}$ and $\mathcal{I}'$. Intuitively, $r(x,x')$ states that x has fewer positive properties than x'.

The following lemma is proved in [16].

Lemma 2. *Let $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ and $\mathcal{I}' = \langle \Delta^{\mathcal{I}'}, \cdot^{\mathcal{I}'} \rangle$ be Kripke structures, and $r \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}'}$ be a relation that satisfies conditions 2, 3, 4 of Definition 9. If $r(x,x')$ holds then, for every positive formula φ, $x \in \varphi^{\mathcal{I}}$ implies $x' \in \varphi^{\mathcal{I}'}$.* $\triangleleft$

Corollary 1. *Let $\mathcal{I}$ and $\mathcal{I}'$ be Kripke structures such that $\mathcal{I} \leq_r \mathcal{I}'$ for some r. Then $\mathcal{I} \leq \mathcal{I}'$.* $\triangleleft$

We are now ready to define the least SPDL model of a HSPDL database.

Definition 10. Let $\langle \mathcal{P}, \mathcal{A} \rangle$ be an HSPDL database. We say that a Kripke structure $\mathcal{I}$ is a *least* SPDL *model of* $\langle \mathcal{P}, \mathcal{A} \rangle$ if $\mathcal{I}$ is an SPDL model of $\langle \mathcal{P}, \mathcal{A} \rangle$ and for any other SPDL model $\mathcal{I}'$ of $\langle \mathcal{P}, \mathcal{A} \rangle$ we have that $\mathcal{I} \leq \mathcal{I}'$. $\triangleleft$

3.2. Constructing Least SPDL Models for HSPDL Databases

Now, we are ready to present an algorithm that constructs a finite least SPDL model for a given HSPDL database $\langle \mathcal{P}, \mathcal{A} \rangle$. During execution, the algorithm constructs the following data structures:

– Δ is a set of objects. We distinguish the subset Δ_0 of Δ that consists of all individuals (from INames).
– H is a mapping that maps every $x \in \Delta$ to a set of formulas, which are the properties that should hold for x. When the elements of Δ are treated as states, $H(x)$ denotes the contents of the state x.

- $Next$ is a mapping such that, for $x \in \Delta$ and $\langle\sigma\rangle\varphi \in H(x)$, we have $Next(x, \langle\sigma\rangle\varphi) \in \Delta$. The meaning of $Next(x, \langle\sigma\rangle\varphi) = y$ is that:
  - $\langle\sigma\rangle\varphi \in H(x)$ and $\varphi \in H(y)$,
  - the "requirement" $\langle\sigma\rangle\varphi$ is realized for x by going to y via a σ-transition.

We call the tuple $\langle\Delta, H, Next\rangle$ a *model graph*.
Using the above data structures, we define a Kripke structure $\mathcal{I}$ such that:

- $\Delta^{\mathcal{I}} = \Delta$,
- $a^{\mathcal{I}} = a$ for every $a \in \mathrm{INames}$,
- $p^{\mathcal{I}} = \{x \in \Delta \mid p \in H(x)\}$ for every $p \in \mathrm{CNames}$,
- $\sigma^{\mathcal{I}} = \{(a, b) \mid \sigma(a, b) \in \mathcal{A}\} \cup \{(x, y) \mid Next(x, \langle\sigma\rangle\varphi) = y$ for some $\varphi\}$ for every $\sigma \in \mathrm{SNames}$.

Definition 11. For $x, y \in \Delta$, we say that y is *reachable from* x if there exists a word $\sigma_1 \ldots \sigma_k$ such that $(\sigma_1 \ldots \sigma_k)^{\mathcal{I}}(x, y)$ holds. We say that y is *reachable from* Δ_0 if it is reachable from some $x \in \Delta_0$. $\triangleleft$

Definition 12. The *saturation* of a set Γ of formulas, denoted by $\mathsf{Sat}(\Gamma)$, is defined to be the smallest superset of Γ such that:

- $\top \in \mathsf{Sat}(\Gamma)$ and $\langle\sigma\rangle\top \in \mathsf{Sat}(\Gamma)$ for all $\sigma \in \mathrm{SNames}$,
- if $\varphi \wedge \psi \in \mathsf{Sat}(\Gamma)$ or $\langle\varphi?\rangle\psi \in \mathsf{Sat}(\Gamma)$ then $\varphi \in \mathsf{Sat}(\Gamma)$ and $\psi \in \mathsf{Sat}(\Gamma)$,
- if $\langle\alpha;\beta\rangle\varphi \in \mathsf{Sat}(\Gamma)$ then $\langle\alpha\rangle\langle\beta\rangle\varphi \in \mathsf{Sat}(\Gamma)$,
- if $[\alpha;\beta]\varphi \in \mathsf{Sat}(\Gamma)$ then $[\alpha][\beta]\varphi \in \mathsf{Sat}(\Gamma)$,
- if $[\alpha \cup \beta]\varphi \in \mathsf{Sat}(\Gamma)$ then $[\alpha]\varphi \in \mathsf{Sat}(\Gamma)$ and $[\beta]\varphi \in \mathsf{Sat}(\Gamma)$,
- if $[\alpha^*]\varphi \in \mathsf{Sat}(\Gamma)$ then $\varphi \in \mathsf{Sat}(\Gamma)$ and $[\alpha][\alpha^*]\varphi \in \mathsf{Sat}(\Gamma)$,
- if $[\varphi?]\psi \in \mathsf{Sat}(\Gamma)$ then $(\varphi \rightarrow \psi) \in \mathsf{Sat}(\Gamma)$. $\triangleleft$

Observe that $\mathsf{Sat}(\Gamma)$ is finite when Γ is finite. It can be shown that the size of $\mathsf{Sat}(\Gamma)$ is quadratic in the size of Γ (cf. Lemma 6.3 in [18]).

Definition 13. The *transfer of Γ through σ* is defined by:

$$\mathsf{Trans}(\Gamma, \sigma) \stackrel{\text{def}}{=} \mathsf{Sat}(\{\varphi \mid [\sigma]\varphi \in \Gamma\}). \qquad \triangleleft$$

We use procedure $\mathsf{Find}(\Gamma)$ defined as:

if there exists $x \in \Delta \setminus \Delta_0$ with $H(x) = \Gamma$ then return x,
else add a new object x to Δ with $H(x) = \Gamma$ and return x.

Algorithm 1 shown in Figure 3 constructs a least SPDL model for an HSPDL database $\langle\mathcal{P}, \mathcal{A}\rangle$ as follows. At the beginning, Δ starts from $\Delta_0 = \mathrm{INames}$ with $H(x)$, for $x \in \Delta_0$, being the saturation of $\mathcal{P} \cup \{p \mid p(x) \in \mathcal{A}\}$. Then for each $x \in \Delta$ reachable from Δ_0 and for each formula $\varphi \in H(x)$ that does not hold for x, the algorithm makes a change to satisfy φ for x.

There are three relevant forms of φ:[8]

[8] The other possible forms of φ are dealt with by the saturation operator Sat.

```
Algorithm 1
Input: An HSPDL database $\langle \mathcal{P}, \mathcal{A} \rangle$.
Output: A least SPDL model $\mathcal{I}$ of $\langle \mathcal{P}, \mathcal{A} \rangle$.

1. set $\Delta_0 := \mathrm{INames}$, $\Delta := \Delta_0$, $\mathcal{P}' := \mathsf{Sat}(\mathcal{P})$
   for $x \in \Delta$, set $H(x) := \mathcal{P}' \cup \{p \mid p(x) \in \mathcal{A}\}$
2. for every $x \in \Delta$ reachable from $\Delta_0$ and for every formula $\varphi \in H(x)$
   (a) case $\varphi = \langle\sigma\rangle\psi$ : if $Next(x, \langle\sigma\rangle\psi)$ is not defined then
           $Next(x, \langle\sigma\rangle\psi) := \mathsf{Find}(\mathsf{Sat}(\{\psi\}) \cup \mathsf{Trans}(H(x), \sigma) \cup \mathcal{P}')$
   (b) case $\varphi = [\sigma]\psi$ :
         i. for every $y \in \Delta_0$ such that $\sigma^{\mathcal{I}}(x, y)$ holds and $\psi \notin H(y)$
              $H(y) := H(y) \cup \mathsf{Sat}(\{\psi\})$
        ii. for every $y \in \Delta \setminus \Delta_0$ such that $\sigma^{\mathcal{I}}(x, y)$ holds and $\psi \notin H(y)$
            A. $y_* := \mathsf{Find}(H(y) \cup \mathsf{Sat}(\{\psi\}))$
            B. for every $\xi$ such that $Next(x, \langle\sigma\rangle\xi) = y$
                 $Next(x, \langle\sigma\rangle\xi) := y_*$
   (c) case $\varphi = (\psi \to \xi)$ : if $x \in \psi^{\mathcal{I}}$ and $Next(y, \langle\sigma\rangle\top)$ is defined for every $y$
       reachable from $x$ and every $\sigma \in \mathrm{SNames}$ then
         i. if $x \in \Delta_0$ then $H(x) := H(x) \cup \mathsf{Sat}(\{\xi\})$
        ii. else
            A. $x_* := \mathsf{Find}(H(x) \cup \mathsf{Sat}(\{\xi\}))$
            B. for every $y$, $\sigma$, $\zeta$ such that $Next(y, \langle\sigma\rangle\zeta) = x$
                 $Next(y, \langle\sigma\rangle\zeta) := x_*$
3. if some change occurred, go to Step 2
4. delete from $\Delta$ every $x$ unreachable from $\Delta_0$ and delete from $H$ and $Next$
   all elements related to such an $x$.
```

Fig. 3. Constructing a least SPDL model for an HSPDL database.

1. φ is of the form $\langle\sigma\rangle\psi$:
   To satisfy φ for x, we connect x via a σ-transition to an object $y \in \Delta \setminus \Delta_0$ with

   $$H(y) = \mathsf{Sat}(\{\psi\} \cup \{\xi \mid [\sigma]\xi \in H(x)\} \cup \mathcal{P})$$

   by setting $Next(x, \langle\sigma\rangle\psi) := y$.
2. φ is of the form $[\sigma]\psi$:
   We intend to add ψ to $H(y)$ for every y such that $\sigma^{\mathcal{I}}(x, y)$. We do this for the case when $y \in \Delta_0$. However, for $y \in \Delta \setminus \Delta_0$ modifying $H(y)$ has two drawbacks:
   - first, other objects connected to y will be affected (e.g., if p is added to $H(y)$ and $\sigma_2^{\mathcal{I}}(z, y)$ holds, then $\langle\sigma_2\rangle p$ becomes satisfied for z, while x and z may be independent)
   - second, modifying $H(y)$ may cause $H(y) = H(y')$ for some $y' \in \Delta \setminus \Delta_0$ different from y, which we try to avoid.

As a solution, instead of modifying $H(y)$ we replace σ-transitions (x, y) by σ-transitions (x, y_*), where y_* is the object such that

$$H(y_*) = H(y) \cup \mathsf{Sat}(\{\psi\}).$$

3. φ is of the form $\psi \to \xi$ (where ψ is a positive formula):
If ψ "must hold"[9] for x then we intend to add ξ to $H(x)$. We do this for the case $x \in \Delta_0$. However, when $x \in \Delta \setminus \Delta_0$, analogously to the case when φ is of the form $[\sigma]\zeta$, we do not modify $H(x)$, but replace transitions (y, x) by transitions (y, x_*), where x_* is the object such that

$$H(x_*) = H(x) \cup \mathsf{Sat}(\{\xi\}).$$

Example 2. Let $\mathcal{P} = \{p \to [\sigma^*]q,\ [\sigma^*]q \to p\}$ and $\mathcal{A} = \{p(a), s(a), \sigma(a, b)\}$. In Figure 4 we illustrate the construction of a least SPDL model of $\langle \mathcal{P}, \mathcal{A} \rangle$. $\lhd$

The proofs of the following lemma and theorem can be found in [16]. For the theorem we assume that the set of individuals (of INames) that do not occur in the ABox $\mathcal{A}$ is fixed.

Lemma 3. *Let $\mathcal{I}$ be the model constructed by Algorithm 1 for $\langle \mathcal{P}, \mathcal{A} \rangle$, and $\mathcal{I}'$ be an arbitrary* SPDL *model of $\langle \mathcal{P}, \mathcal{A} \rangle$. Let*

$$\begin{aligned} r = \{&(a, a^{\mathcal{I}'}) \mid a \text{ is an individual occurring in } \mathcal{A}\} \cup \\ &\{(x, x') \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}'} \mid x \text{ is not an individual and } \mathcal{I}', x' \models H(x)\}. \end{aligned}$$

Then $\mathcal{I} \leq_r \mathcal{I}'$. $\lhd$

Theorem 1. *For an input* HSPDL *database $\langle \mathcal{P}, \mathcal{A} \rangle$ Algorithm 1 runs in polynomial time in the size of $\mathcal{A}$ and returns a least* SPDL *model $\mathcal{I}$ of $\langle \mathcal{P}, \mathcal{A} \rangle$ of a size polynomial in the size of $\mathcal{A}$.* $\lhd$

Remark 2. The above theorem is central for the querying machinery developed in this paper. According to Definitions 8 and 10, the least model $\mathcal{I}$ has the property that for every positive formula φ and for every individual a, we have that $\langle \mathcal{P}, \mathcal{A} \rangle \models_s \varphi(a)$ iff $a^{\mathcal{I}} \in \varphi^{\mathcal{I}}$. The model is then used to compute answers to queries. $\lhd$

The following corollary follows from the above theorem and Lemma 1.

Corollary 2. *The data complexity of* HSPDL *is in* PTIME. $\lhd$

[9] The statement "ψ must hold for x" intuitively means that "ψ follows from $H(x)$". As it can be seen later, a sufficient condition for the truth of this statement is that $x \in \psi^{\mathcal{I}}$ and $Next(y, \langle\sigma\rangle\top)$ is defined for every y reachable from x and every $\sigma \in \mathrm{SNames}$.

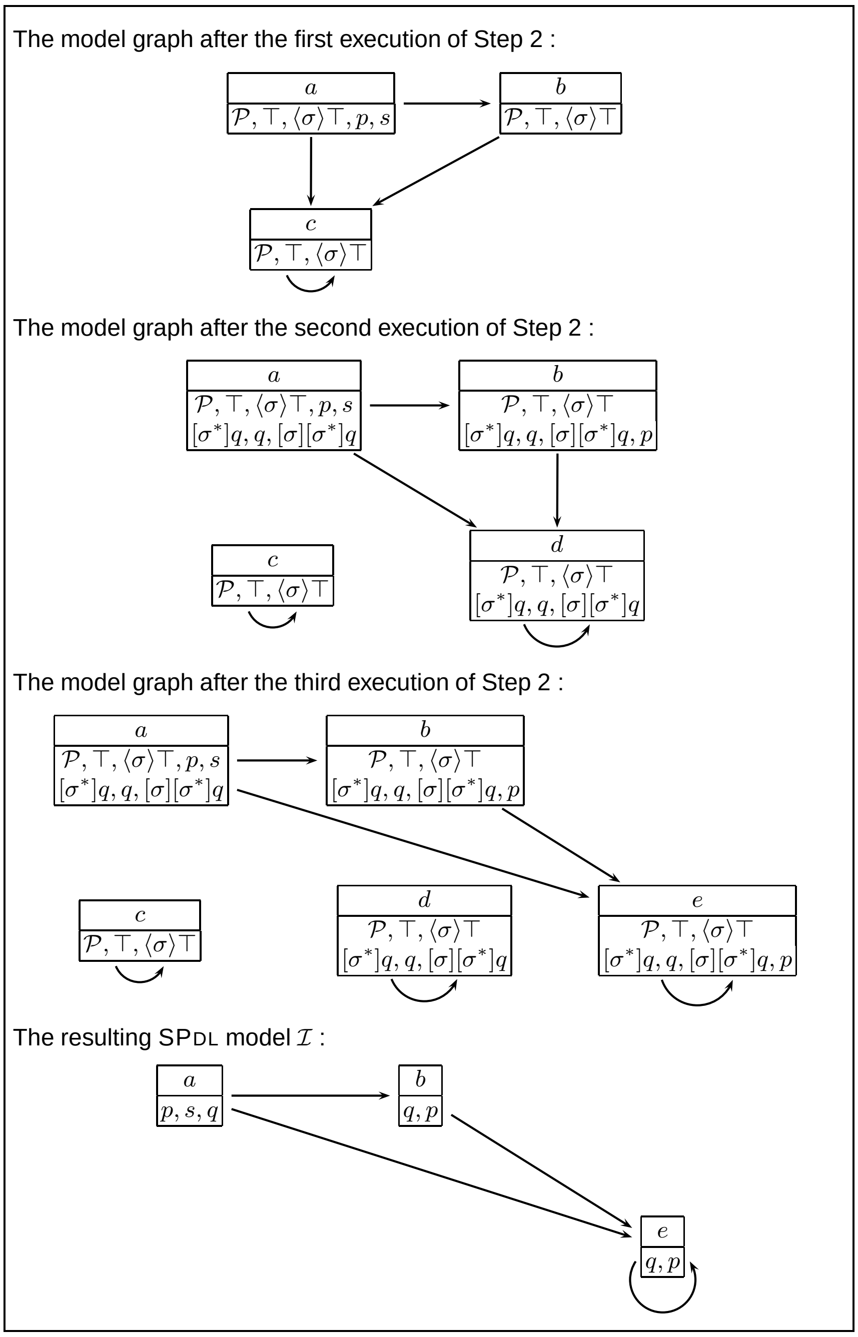
The model graph after the first execution of Step 2 :
a
$\mathcal{P}, \top, \langle\sigma\rangle\top, p, s$
b
$\mathcal{P}, \top, \langle\sigma\rangle\top$
c
$\mathcal{P}, \top, \langle\sigma\rangle\top$
The model graph after the second execution of Step 2 :
a
$\mathcal{P}, \top, \langle\sigma\rangle\top, p, s$
$[\sigma^*]q, q, [\sigma][\sigma^*]q$
b
$\mathcal{P}, \top, \langle\sigma\rangle\top$
$[\sigma^*]q, q, [\sigma][\sigma^*]q, p$
c
$\mathcal{P}, \top, \langle\sigma\rangle\top$
d
$\mathcal{P}, \top, \langle\sigma\rangle\top$
$[\sigma^*]q, q, [\sigma][\sigma^*]q$
The model graph after the third execution of Step 2 :
a
$\mathcal{P}, \top, \langle\sigma\rangle\top, p, s$
$[\sigma^*]q, q, [\sigma][\sigma^*]q$
b
$\mathcal{P}, \top, \langle\sigma\rangle\top$
$[\sigma^*]q, q, [\sigma][\sigma^*]q, p$
c
$\mathcal{P}, \top, \langle\sigma\rangle\top$
d
$\mathcal{P}, \top, \langle\sigma\rangle\top$
$[\sigma^*]q, q, [\sigma][\sigma^*]q$
e
$\mathcal{P}, \top, \langle\sigma\rangle\top$
$[\sigma^*]q, q, [\sigma][\sigma^*]q, p$
The resulting SPDL model $\mathcal{I}$:
a
p, s, q
b
q, p
e
q, p

Fig. 4. An illustration of the run of Algorithm 1 for $\mathcal{P} = \{p \to [\sigma^*]q,\ [\sigma^*]q \to p\}$ and $\mathcal{A} = \{p(a), s(a), \sigma(a,b)\}$. We have that $\Delta_0 = \{a, b\}$. In the shown model graphs, an edge from a node x to a node y means $Next(x, \langle\sigma\rangle\top) = y$. The edges in the resulting model $\mathcal{I}$ represent the similarity relation $\sigma^{\mathcal{I}}$.

3.3. Important Consequences of the Construction of Least Models for HSPDL

Some steps of Algorithm 1 add new objects to satisfy certain formulas. This is a new phenomenon, comparing to more traditional rule languages, where, e.g., existential quantification in heads of program clauses is forbidden. Such an addition of new objects sometimes occurs as a result of application of procedure $Find$, e.g., in Step 2a of Algorithm 1. On the other hand, this phenomenon seriously affects similarity relations. The following example illustrates the problem.

Example 3. Consider an ABox $\{p(a), \sigma(a, a)\}$ and a rule $p \to \langle\sigma\rangle q$. In such a case, during construction of a least model, Algorithm 1 adds to its universe two new objects, say b and c, for which $\sigma(a, b)$, $\sigma(a, c)$, $\sigma(b, c)$, $\sigma(c, c)$ and $q(b)$ additionally hold. Extending the domain by artificially created objects might be seen as a rather unexpected side-effect of the construction of a least model. The explanation is that new objects are sometimes added to satisfy certain rules. $\lhd$

The above example shows that rules do add new objects which are not grounded in the ABox of the database. Methodologically, such a situation is doubtful, as such artificially added objects are not as strongly justified as objects "observed" and directly described in terms of facts. In fact, these artificial objects are only possible explanations of rules, so have a rather weak status. We address this point in our layered architecture.

Another important issue is that the construction of a least model provided by Algorithm 1 may result in unexpected consequences due to identifying certain new objects. The following example illustrates this problem.

Example 4. Consider the rule $\sigma(a, x) \wedge \sigma(b, x) \to p(a, b)$, where σ is a similarity relation. In general $p(a, b)$ may not be a consequence of the rule. However, Algorithm 1 might have identified two objects, say c, d such that $\sigma(a, c)$ and $\sigma(b, d)$ hold. In such a case $p(a, b)$ would become deducible from the considered rule. $\lhd$

Observe that in the light of Theorem 1, Algorithm 1 constructs a least model for the input program. This means (see Remark 2) that the method we propose is correct. The above discussion applies to the case when queries of the highest level directly refer to similarity relations constructed in the middle layer.

Summing up, it is not safe to use similarity relations that have been changed by the HSPDL layer in higher-level queries. In the following sections we define a query language which allows one to ask only "safe" queries, free of the unwanted side-effects discussed above and guaranteeing the correctness of reasoning.

4. Combining HSPDL With Datalog

In the presence of ABoxes, a concept name can be viewed as a unary predicate, and a similarity relation symbol can be viewed as a binary predicate. In this section we extend our language HSPDL with external capabilities offered by database technologies and/or other software systems. The idea is quite general. However, in what follows we focus on Datalog as a possible instantiation of the idea. Therefore, in what follows we consider combination of HSPDL and Datalog and external data types.[10]

To solve problems discussed in Section 3.3, we introduce a special unary predicate $\overline{\mathrm{IName}}$ (treated as a concept name) with the semantics that, in every interpretation $\mathcal{I}$, $\overline{\mathrm{IName}}^{\mathcal{I}}$ consists of all objects which are not assigned to any individual from INames.

We use two basic types $\mathcal{O}$ and $\mathcal{D}$, where $\mathcal{O}$ is called the *individual type* (or *object type*) and $\mathcal{D}$ is called the *data type*. We assume that $\mathcal{D}$ is a fixed non-empty set, which may be the set of real numbers, the set of natural numbers, the set of strings, or a mixture of them. For simplicity we do not divide $\mathcal{D}$ into components.

An individual $a \in \mathrm{INames}$ has type $\mathcal{O}$, a concept name $p \in \mathrm{CNames} \cup \{\overline{\mathrm{IName}}\}$ has type $P(\mathcal{O})$ (the powerset type of $\mathcal{O}$), and a similarity relation symbol $\sigma \in \mathrm{SNames}$ has type $P(\mathcal{O} \times \mathcal{O})$. Apart from CNames and SNames, we use also a set OPreds of *ordinary predicates* and a set $\mathrm{ECPreds}$ of *external checkable predicates*. A k-ary predicate of OPreds has type $P(T_1 \times \ldots \times T_k)$, where each T_i is either $\mathcal{O}$ or $\mathcal{D}$. A k-ary predicate of $\mathrm{ECPreds}$ has type $P(\mathcal{D}^k)$. We assume that each predicate of $\mathrm{ECPreds}$ has a fixed interpretation which is checkable in the sense that, if p is a k-ary predicate of $\mathrm{ECPreds}$ and $d_1, \ldots, d_k$ are elements of $\mathcal{D}$, then the truth value of $p(d_1, \ldots, d_k)$ is fixed and computable. For example, when $\mathcal{D}$ is the type of real numbers, we may want to use the binary predicates $>$, $\geq$, $<$, $\leq$ on $\mathcal{D}$ with the usual semantics.

We assume that the sets INames, CNames, SNames, OPreds, $\mathrm{ECPreds}$ and $\mathcal{D}$ are pairwise disjoint and do not contain $\overline{\mathrm{IName}}$. Let $\mathrm{Preds} = \mathrm{CNames} \cup \mathrm{SNames} \cup \mathrm{OPreds} \cup \mathrm{ECPreds} \cup \{\overline{\mathrm{IName}}\}$. It is the set of all predicates of our language.

Definition 14. An *interpretation* is a pair $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ is a non-empty set of objects and $\cdot^{\mathcal{I}}$ is an interpretation function that maps each individual a to an element $a^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, maps $\overline{\mathrm{IName}}$ to $\overline{\mathrm{IName}}^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus \{a^{\mathcal{I}} \mid a \in \mathrm{INames}\}$, and maps each predicate $p \in \mathrm{CNames} \cup \mathrm{SNames} \cup \mathrm{OPreds}$ of type $P(T_1 \times \ldots \times T_k)$ to a subset $p^{\mathcal{I}}$ of $D_1 \times \ldots \times D_k$, where $D_i = \Delta^{\mathcal{I}}$ if $T_i = \mathcal{O}$, and $D_i = \mathcal{D}$ if $T_i = \mathcal{D}$, for $1 \leq i \leq k$. $\triangleleft$

An interpretation $\mathcal{I}$ can be treated as a Kripke structure by restricting $\cdot^{\mathcal{I}}$ to $\mathrm{INames} \cup \mathrm{CNames} \cup \mathrm{SNames} \cup \{\overline{\mathrm{IName}}\}$, especially when interpreting formulas of PDL. Given a formula φ of PDL built from concept names of $\mathrm{CNames} \cup \{\overline{\mathrm{IName}}\}$ and similarity relation symbols of SNames, the set $\varphi^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ is defined as usual. For $a \in \mathrm{INames}$, we write $\mathcal{I} \models \varphi(a)$ to denote that $a^{\mathcal{I}} \in \varphi^{\mathcal{I}}$.

[10] In the literature (e.g., [1]), Datalog programs consist of an extensional database (facts) and an intensional database (rules). In this paper we refer to the extensional part of the database as to its ABox.

Definition 15. A *term* is either an individual or an element of $\mathcal{D}$ (called a *data constant*) or a *variable* (of type $\mathcal{O}$ or $\mathcal{D}$). If p is a predicate of type $P(T_1 \times \ldots \times T_k)$, and for $1 \leq i \leq k$, t_i is a term of type T_i, then $p(t_1, \ldots, t_k)$ is an *atomic formula* (also called an *atom*). $\triangleleft$

From now on we use letters like x, y, z to denote variables, and letters like t to denote terms. We assume that the types of used predicates are given, and each used variable has a unique type $\mathcal{O}$ or $\mathcal{D}$, known from the context.

Definition 16. A *variable assignment* w.r.t. an interpretation $\mathcal{I}$ is a function that maps each variable of type $\mathcal{O}$ to an element of $\Delta^{\mathcal{I}}$ and maps each variable of type $\mathcal{D}$ to an element of $\mathcal{D}$.

The *value of a term* t w.r.t. a variable assignment ν is denoted by t^ν and defined as follows: if t is a variable then $t^\nu = \nu(t)$; if t is an individual then $t^\nu = t^{\mathcal{I}}$; if t is a data constant (i.e. $t \in \mathcal{D}$) then $t^\nu = t$.

Let $\mathcal{I}$ be an interpretation and ν be a variable assignment w.r.t. $\mathcal{I}$. We say that an atom $p(t_1, \ldots, t_k)$ is *satisfied* in $\mathcal{I}$ using ν, write $\mathcal{I}, \nu \models p(t_1, \ldots, t_k)$, if $(t_1^\nu, \ldots, t_k^\nu) \in p^{\mathcal{I}}$ for the case $p \notin \mathrm{ECPreds}$, and $p(t_1^\nu, \ldots, t_k^\nu)$ holds for the case $p \in \mathrm{ECPreds}$. A ground atom A (i.e. an atom without variables) is *satisfied* in $\mathcal{I}$, write $\mathcal{I} \models A$, if $\mathcal{I}, \nu \models A$ for any ν. $\triangleleft$

Definition 17. An *ABox* (in the extended language) is a finite set of ground atoms of predicates of $\mathrm{CNames} \cup \mathrm{SNames} \cup \mathrm{OPreds}$. An interpretation $\mathcal{I}$ is a *model* of an ABox $\mathcal{A}$ if all atoms of $\mathcal{A}$ are satisfied in $\mathcal{I}$. $\triangleleft$

We now define Datalog extended with external checkable predicates.

Definition 18.

- A *Datalog program clause* is a formula of the form

$$A_1 \wedge \ldots \wedge A_n \rightarrow B$$

  where $n \geq 0$ and $A_1, \ldots, A_n, B$ are atomic formulas with the restriction that:
  - B is an atom of a predicate of $\mathrm{CNames} \cup \mathrm{SNames} \cup \mathrm{OPreds}$
  - every variable occurring in B occurs also in $A_1 \wedge \ldots \wedge A_n$
  - every variable occurring in the clause occurs, amongst others, in an atom of a predicate not belonging to $\mathrm{ECPreds}$.

  The last two restrictions are called the *range-restrictedness* condition. We call B the *head*, and $A_1 \wedge \ldots \wedge A_n$ the *body* of the clause. We omit the implication sign $\rightarrow$ when $n = 0$.
- A *Datalog program* is a finite set of Datalog program clauses.
- An interpretation $\mathcal{I}$ *validates* a Datalog program clause $A_1 \wedge \ldots \wedge A_n \rightarrow B$ if for every variable assignment ν, if $\mathcal{I}, \nu \models A_i$ for all $1 \leq i \leq n$ then $\mathcal{I}, \nu \models B$. An interpretation $\mathcal{I}$ is a *model* of a Datalog program $\mathcal{P}$ if it validates all the clauses of $\mathcal{P}$. $\triangleleft$

We now consider combination of HSPDL and Datalog. In the combined language, a database consists of an ABox $\mathcal{A}$ as the extensional part, and a mixed logic program $\mathcal{P}$ of HSPDL and Datalog as the intensional part. We will study the case when $\mathcal{P}$ consists of three layers, as discussed in Section 1.2 and illustrated in Figure 1. To be now more concrete, consider $\mathcal{P} = \langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3 \rangle$ with the following meaning:

– the lower layer $\mathcal{P}_1$ is a Datalog program intended for specifying the most basic predicates, basic concepts and similarity relations by using the perceptual data of agents and assertions stored in $\mathcal{A}$
– the middle layer $\mathcal{P}_2$ is an HSPDL logic program built on top of $\mathcal{P}_1$ and $\mathcal{A}$ for specifying advanced concepts by using the concepts and similarity relations specified in $\mathcal{P}_1$ and $\mathcal{A}$
– the upper layer $\mathcal{P}_3$ is a Datalog program built on top of $\mathcal{P}_2$, $\mathcal{P}_1$ and $\mathcal{A}$ for defining additional ordinary predicates and for completing definition of concepts and ordinary predicates.

The set $\mathit{InPreds}(\mathcal{P})$ (resp. $\mathit{OutPreds}(\mathcal{P})$) of *input predicates* (resp. *output predicates*) a Datalog program $\mathcal{P}$ is the set of all predicates occurring in the heads (resp. bodies) of program clauses of $\mathcal{P}$.

We define the set $\mathit{OutPreds}(\varphi)$ of *output predicates* of an HSPDL program clause φ recursively as follows:

$$\begin{aligned}
\mathit{OutPreds}(\top) &= \emptyset \\
\mathit{OutPreds}(p) &= \{p\} \\
\mathit{OutPreds}(\psi \to \xi) &= \mathit{OutPreds}(\xi) \\
\mathit{OutPreds}(\psi \land \xi) &= \mathit{OutPreds}(\psi) \cup \mathit{OutPreds}(\xi) \\
\mathit{OutPreds}(\langle\alpha\rangle\psi) &= \mathit{OutPreds}(\alpha) \cup \mathit{OutPreds}(\psi) \\
\mathit{OutPreds}([\alpha]\psi) &= \mathit{OutPreds}(\psi) \\
\mathit{OutPreds}(\sigma) &= \{\sigma\} \\
\mathit{OutPreds}(\alpha;\beta) &= \mathit{OutPreds}(\alpha) \cup \mathit{OutPreds}(\beta) \\
\mathit{OutPreds}(\psi?) &= \mathit{OutPreds}(\psi)
\end{aligned}$$

For example, $\mathit{OutPreds}([\sigma_1]p \to \langle\sigma_3\rangle(\langle\sigma_2\rangle q \to (r \land [\sigma_4]s))) = \{\sigma_3, r, s\}$.

The set $\mathit{OutPreds}(\mathcal{P})$ of *output predicates* of an HSPDL logic program $\mathcal{P}$ is defined to be $\bigcup_{\varphi \in \mathcal{P}} \mathit{OutPreds}(\varphi)$.

Definition 19. A *three-layered* HSPDL-*Datalog program* is a tuple $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3 \rangle$, where $\mathcal{P}_1$, $\mathcal{P}_3$ are Datalog programs and $\mathcal{P}_2$ is an HSPDL logic program using $\mathrm{CNames} \cup \{\overline{\mathrm{IName}}\}$ as the set of concept names, with the property that:

$$\mathit{OutPreds}(\mathcal{P}_2) \cap \mathit{InPreds}(\mathcal{P}_1) = \emptyset \tag{6}$$

$$\mathit{OutPreds}(\mathcal{P}_3) \subseteq \mathrm{CNames} \cup \mathrm{OPreds} \tag{7}$$

$$\mathit{OutPreds}(\mathcal{P}_3) \cap \mathit{InPreds}(\mathcal{P}_1) = \emptyset \tag{8}$$

$$\mathit{OutPreds}(\mathcal{P}_3) \cap \mathit{InPreds}(\mathcal{P}_2) = \emptyset \tag{9}$$

$$\mathit{OutPreds}(\mathcal{P}_2) \cap \mathrm{SNames} \cap \mathit{InPreds}(\mathcal{P}_3) = \emptyset \tag{10}$$
$$\overline{\mathrm{IName}} \notin \mathit{OutPreds}(\mathcal{P}_2) \tag{11}$$

Condition (6) states that the output predicates of $\mathcal{P}_2$ are not used as input predicates of $\mathcal{P}_1$. Conditions (7), (8) and (9) state that the output predicates of $\mathcal{P}_3$ can be only concept names or ordinary predicates which are not used as input predicates of $\mathcal{P}_1$ and $\mathcal{P}_2$. Roughly speaking, these conditions state that $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3 \rangle$ is well-layered. Additionally, Conditions (10) and (6) guarantee that the similarity relations specified by $\mathcal{P}_2$ are not used as input predicates of $\mathcal{P}_1$ and $\mathcal{P}_3$. The reason is that an HSPDL logic program is intended to specify and minimize only (complex) concepts, but not to minimize similarity relations that are specified as side effects of existential modal operators. By (11) and the definition of Datalog program clauses, $\overline{\mathrm{IName}} \notin \mathit{OutPreds}(\mathcal{P}_1) \cup \mathit{OutPreds}(\mathcal{P}_2) \cup \mathit{OutPreds}(\mathcal{P}_3)$.

Definition 20. If $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3 \rangle$ is a three-layered HSPDL-Datalog program and $\mathcal{A}$ is an ABox then the tuple $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{A} \rangle$ is called a *three-layered* HSPDL-*Datalog database* (with $\mathcal{A}$ as a part of the bottom layer). $\triangleleft$

Observe that $\mathcal{A}$ is separated as data complexity takes its size as input. Also, facts from $\mathcal{A}$ are accessible to all layers.

In the following definition we accept the well-known Unique Names Assumption (see, e.g., [1]) for individuals from INames. Furthermore, the interpretation of similarity relations restricted to objects interpreting individuals from INames is computed and fixed by the first layer $\mathcal{P}_1$ using the minimal Herbrand model semantics.

Definition 21. Let $\mathcal{I}$ be an interpretation, $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3 \rangle$ be a three-layered HSPDL-Datalog program, and $\mathcal{A}$ be an ABox. We say that $\mathcal{I}$ is a *model* of the three-layered HSPDL-Datalog database $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{A} \rangle$ if:

- it is a model of $\mathcal{P}_1, \mathcal{P}_3, \mathcal{A}$ and is an SPDL model of $\mathcal{P}_2$ (when restricting $\cdot^{\mathcal{I}}$ to $\mathrm{INames} \cup \mathrm{CNames} \cup \mathrm{SNames} \cup \{\overline{\mathrm{IName}}\}$)
- for every $a \neq b \in \mathrm{INames}$, we have that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$
- for every interpretation $\mathcal{I}'$ satisfying the previous two conditions, for every $\sigma \in \mathrm{SNames}$ and $a, b \in \mathrm{INames}$, if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in \sigma^{\mathcal{I}}$ then $(a^{\mathcal{I}'}, b^{\mathcal{I}'}) \in \sigma^{\mathcal{I}'}$. $\triangleleft$

Definition 22. A *query* to a three-layered HSPDL-Datalog database $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{A} \rangle$ is an atomic formula A of a predicate of $\mathrm{CNames} \cup \mathrm{OPreds}$. A (correct) *answer* to such a query is a substitution $\theta = \{x_1/t_1, \ldots, x_k/t_k\}$ such that:

- $x_1, \ldots, x_k$ are all the different variables occurring in the query
- for $1 \leq i \leq k$, if x_i is a variable of type $\mathcal{O}$ then t_i is an individual, else (x_i is a variable of type $\mathcal{D}$ and) t_i is a data constant of $\mathcal{D}$
- the ground atom $A\theta$ is satisfied in every model of $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{A} \rangle$. $\triangleleft$

Note that more complex queries can be expressed by adding a clause to the intensional part of the database. For example, if φ is a positive formula of PDL without predicates of $OutPreds(\mathcal{P}_3)$ and a is an individual, then to check whether $a^{\mathcal{I}} \in \varphi^{\mathcal{I}}$ in every model $\mathcal{I}$ of a database $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{A} \rangle$ we can check whether $\{x/a\}$ is a correct answer to the query $p(x)$ w.r.t. the extended database $\langle \mathcal{P}_1, \mathcal{P}_2 \cup \{\varphi \to p\}, \mathcal{P}_3, \mathcal{A} \rangle$, where p is a new concept name. Similarly, if $\varphi = A_1 \land \ldots \land A_n$ is a formula such that $x_1, \ldots, x_k$ are all the variables occurring in φ and no predicate of φ belongs to $OutPreds(\mathcal{P}_2) \cap \mathrm{SNames}$, then answers to the query $p(x_1, \ldots, x_k)$ w.r.t. $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3 \cup \{\varphi \to p(x_1, \ldots, x_k)\}, \mathcal{A} \rangle$, where p is a new predicate of OPreds, are exactly the ground substitutions θ such that $A_1\theta$, ..., $A_n\theta$ are satisfied in every model of $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{A} \rangle$.

Definition 23. By the *three-layered* HSPDL*-Datalog query language* we refer to the language of three-layered HSPDL-Datalog databases and their queries. The *data complexity* of this language is the complexity of the problem of finding all answers to a given query A w.r.t. a given three-layered HSPDL-Datalog database $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{A} \rangle$, which is measured in the size of $\mathcal{A}$, when A, $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}_3$ are fixed. $\lhd$

The following theorem is the main result of this section, for which we assume that, given a k-ary predicate p of $\mathrm{ECPreds}$ and elements $d_1, \ldots, d_k$ of $\mathcal{D}$, checking whether $p(d_1, \ldots, d_k)$ holds can be done in polynomial time in the number of bits needed to represent d_1, ..., d_k.

Theorem 2. *The three-layered* HSPDL*-Datalog query language has* PTIME *data complexity.*

Proof. (sketch) Let a three-layered HSPDL-Datalog database $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{A} \rangle$ be given. First, we compute the minimal Herbrand model $\mathcal{I}_1$ of $\mathcal{P}_1$ and $\mathcal{A}$. Treating it as an ABox, we next compute a least model $\mathcal{I}_2$ for $\langle \mathcal{P}_2, \mathcal{I}_1 \rangle$ using Algorithm 1 with the following modification of procedure $\mathsf{Find}(\Gamma)$:

if there exists $x \in \Delta \setminus \Delta_0$ with $H(x) = \Gamma \cup \{\overline{\mathrm{IName}}\}$ then return x,
else add a new object x to Δ with $H(x) = \Gamma \cup \{\overline{\mathrm{IName}}\}$ and return x.

Treating $\mathcal{I}_2$ as an ABox, we now compute the minimal Herbrand model $\mathcal{I}_3$ of $\mathcal{P}_3$ and $\mathcal{I}_2$. It can be shown that, for any query A, a ground substitution θ is an answer to A w.r.t. the database $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{A} \rangle$ iff $A\theta$ is satisfied in the interpretation corresponding to $\mathcal{I}_3$. By Theorem 1 and the fact that the data complexity of Datalog is in PTIME, the above computation runs in time polynomial w.r.t. the size of $\mathcal{A}$. $\lhd$

5. Example

5.1. The Scenario

Consider safety of UGVs' movement on a specific surface (UGV is an acronym for Unmanned Ground Vehicle). For simplicity, we consider two UGVs, denoted by UGV$_1$ and UGV$_2$, operating on the same road segment and exchanging their knowledge, as well as assume that slipperiness of the road and speed of the UGV are the only factors that affect their safety.

Assume that the following equipment and data is available:

- there is an external sensor measuring the slipperiness of the road, evaluated by a real number in the range $[1, 10]$
- each UGV is equipped with sensors detecting its speed, measured in $\frac{km}{h}$, being a real number in $[0, 20]$
- there is a database of facts about situations that led or did not lead to accidents caused by the UGVs.

The main objects are situations, which form the type $\mathcal{O}$. In this example, the type $\mathcal{D}$ consists of real numbers and we use the following predicates, where $i \in \{1, 2\}$:[11]

- ordinary predicates spd_i (speed), slp (slipperiness), $dec\text{-}spd_i$ (decrease speed); the intuitive meaning of these predicates is:
  - $spd_i(x, y)$ holds when the speed of UGV$_i$ in situation x is y
  - $slp(x, y)$ holds when the slipperiness of the considered road segment in situation x is y
  - $dec\text{-}spd_i(x, y)$ holds when the speed of UGV$_i$ in situation x should be decreased by y $\frac{km}{h}$.
- external checkable predicates $>$, $<$, $\leq$ and sim, sim_i, where $>$, $<$, $\leq$ have the standard meaning, as in the arithmetics of reals and

$$sim_i(x_1, x_2) \overset{\text{def}}{\equiv} \frac{abs(x_1 - x_2)}{max(x_1, x_2)} \leq \epsilon_i,$$

  where ϵ_i reflects the accuracy of speed measurements of UGV$_i$; we assume here that $\epsilon_1 \overset{\text{def}}{=} 0.12$ and $\epsilon_2 \overset{\text{def}}{=} 0.09$; for sim (non-indexed) we use $\epsilon \overset{\text{def}}{=} 0.18$
- concept names $h\text{-}spd_i$ (high speed), $h\text{-}slp$ (highly slippery), $unsafe$, $unsafe_i$, $h\text{-}unsafe_i$ (highly unsafe), $accident$
- similarity relation symbols σ_i, where the intended meaning of $\sigma_i(x_1, x_2)$ is that situations x_1 and x_2 are similar w.r.t. UGV$_i$
- auxiliary similarity relation symbol ϱ, used for expressing that one situation is "safer" then the other w.r.t. both speed and slipperiness.

5.2. Exemplary Rules

Consider the following layers of the system, where we assume that $i \in \{1, 2\}$.

[11] The index i indicates subjective knowledge of UGV$_i$, while the lack of index indicates that the respective concepts are related to the external or fused knowledge.

Rules of the Lower Layer

$$spd_i(x,y) \wedge y > 15 \rightarrow h\text{-}spd_i(x) \tag{12}$$

$$slp(x,y) \wedge y > 7 \rightarrow h\text{-}slp(x) \tag{13}$$

$$\begin{aligned}\big(spd_i(x_1,y_1) \wedge slp(x_1,z_1) \wedge spd_i(x_2,y_2) \wedge slp(x_2,z_2) \wedge \\ sim_i(y_1,y_2) \wedge sim(z_1,z_2)\big) \rightarrow \sigma_i(x_1,x_2)\end{aligned} \tag{14}$$

$$\begin{aligned}\big(spd_i(x_1,y_1) \wedge slp(x_1,z_1) \wedge spd_i(x_2,y_2) \wedge slp(x_2,z_2) \wedge \\ y_1 < y_2 \wedge z_1 \leq z_2\big) \rightarrow \varrho_i(x_1,x_2)\end{aligned} \tag{15}$$

$$\begin{aligned}\big(spd_i(x_1,y_1) \wedge slp(x_1,z_1) \wedge spd_i(x_2,y_2) \wedge slp(x_2,z_2) \wedge \\ y_1 \leq y_2 \wedge z_1 < z_2\big) \rightarrow \varrho_i(x_1,x_2)\end{aligned} \tag{16}$$

The meaning of these rules is:

(12): UGV's speed greater than $15\frac{km}{h}$ is considered high
(13): road slipperiness greater than 7 is considered high
(14): two situations are similar w.r.t. σ_i when speeds and slipperiness in these situations are similar (w.r.t. sim_i)
(15) and (16): $\varrho_i(x_1,x_2)$ holds when situation x_1 is "safer" than x_2 from the point of view of UGV$_i$.

Rules of the Middle Layer

$$h\text{-}spd_i \wedge h\text{-}slp \rightarrow unsafe_i \tag{17}$$

$$unsafe_i \rightarrow [\varrho_i]unsafe_i \tag{18}$$

$$[\varrho_i](\overline{\mathrm{IName}} \vee \langle\sigma_i\rangle accident) \rightarrow unsafe_i \tag{19}$$

$$[\varrho_i](\overline{\mathrm{IName}} \vee accident) \rightarrow h\text{-}unsafe_i \tag{20}$$

$$[\varrho_1 \cup \varrho_2](\overline{\mathrm{IName}} \vee (unsafe_1 \wedge h\text{-}unsafe_2)) \rightarrow unsafe \tag{21}$$

$$[\varrho_1 \cup \varrho_2](\overline{\mathrm{IName}} \vee (h\text{-}unsafe_1 \wedge unsafe_2)) \rightarrow unsafe \tag{22}$$

Note that a formula of the form $(\overline{\mathrm{IName}} \vee \varphi)$ represents the set of objects x such that if x is assigned to some individual of INames then x satisfies the property φ. Intuitively, this means that either the situation x is not explicitly presented in the database[12] or it satisfies the property φ.

The meaning of the above rules is:

(17): the situation is unsafe whenever both the speed and the slipperiness are high
(18): if situation s is unsafe then also situations with the speed and slipperiness greater than or equal to those of s, are unsafe
(19): if, for a given situation s, every situation that is explicitly presented in the database and more dangerous than s (w.r.t. ϱ_i) is similar (w.r.t. σ_i) to a situation in which there was an accident, then conclude that s is also unsafe for UGV$_i$

[12] That is, x is added by Algorithm 1.

(20): if, for a given situation s, in every situation that is explicitly presented in the database and more dangerous than s (w.r.t. ϱ_i) there was an accident, then conclude that s is a highly unsafe situation for UGV$_i$

(21) and (22): specify unsafeness by fusing similarity relations of both UGVs.

Rules of the Upper Layer

$$unsafe_i(x) \wedge h\text{-}slp(x) \rightarrow dec\text{-}spd_i(x, 2) \tag{23}$$

$$h\text{-}unsafe_i(x) \wedge h\text{-}slp(x) \rightarrow dec\text{-}spd_i(x, 5) \tag{24}$$

$$unsafe(x) \rightarrow dec\text{-}spd_i(x, 3) \tag{25}$$

The meaning of these rules is:

(23): if x is an unsafe situation for UGV$_i$ in which the road is highly slippery, then UGV$_i$ should decrease its speed by $2\frac{km}{h}$

(24): if x is a highly unsafe situation for UGV$_i$, in which the road is highly slippery, then UGV$_i$ should decrease its speed by $5\frac{km}{h}$

(25): if x is an unsafe situation, then each UGV should decrease its speed by $3\frac{km}{h}$.

Let

- $\mathcal{P}_1$ be the Datalog program consisting of the clauses (12) – (16)
- $\mathcal{P}_2$ be the HSPDL program consisting of the clauses (17) – (22)
- $\mathcal{P}_3$ be the Datalog program consisting of the clauses (23) – (25).

Thus, $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3 \rangle$ is a three-layered HSPDL-Datalog program.

Exemplary Facts of the Lower Layer

Let $\mathcal{A}$ be the exemplary ABox consisting of facts presented below:

situation	slp	spd_1	spd_2	$accident$
s_0	10	20	20	yes
s_1	9	19	17	yes
s_2	8	17	18	yes
s_3	7	16	17	yes
s_4	7	16	16	(no)
s_5	6	12	17	(no)
s_6	4	18	15	(no)
s_7	6	16	16	?

This ABox contains only four facts of predicate $accident$, with argument s_0, s_1, s_2 or s_3. In the situations s_4, s_5, s_6, there were no accidents. The situation s_7 is the current situation, for which we want to consider safeness of the UGVs. With respect to the three-layered HSPDL-Datalog database $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{A} \rangle$, the query $dec\text{-}spd_1(s_7, x)$ returns answer $x = 2$.

6. Conclusions

In the paper we have addressed the problem of fusing possibly approximate knowledge from distributed sources. To express fusion rules we have used the Horn fragment of serial propositional dynamic logic combined with Datalog-based deductive databases machinery. As a framework for such a combination we have proposed a three-layered architecture. This allowed us to provide a pragmatic framework for knowledge fusion that can be used in many application areas. We have demonstrated the use of our approach on an example.

The paper can also be considered as an advanced case-study of embedding expressive propositional logics into database environments. Similar methodology can be used for many other logics designed as specification and computational tools for advanced software and robotics systems.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley Pub. Co., 1996.
2. F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *Description Logic Handbook*. Cambridge University Press, 2002.
3. T.H. Cormen, E.H. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.
4. S.P. Demri and E.S. Orłowska. *Incomplete Information: Structure, Inference, Complexity*. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Heidelberg, 2002.
5. P. Doherty, B. Dunin-Kęplicz, and A. Szałas. Dynamics of approximate information fusion. In M. Kryszkiewicz, J. Peters, H. Rybinski, and A. Skowron, editors, *Proc. RSEISP 2007, Rough Sets and Emerging Intelligent Systems Paradigms*, number 4585 in LNAI, pages 668–677. Springer-Verlag, 2007.
6. P. Doherty, W. Łukaszewicz, A. Skowron, and A. Szałas. *Knowledge Representation Techniques. A Rough Set Approach*, volume 202 of *Studies in Fuziness and Soft Computing*. Springer-Verlag, 2006.
7. P. Doherty, W. Łukaszewicz, and A. Szałas. Communication between agents with heterogeneous perceptual capabilities. *Journal of Information Fusion*, 8(1):56–69, 2007.
8. P. Doherty and A. Szałas. On the correspondence between approximations and similarity. In S. Tsumoto, R. Slowinski, J. Komorowski, and J.W. Grzymala-Busse, editors, *Proc. of 4th International Conf. on Rough Sets and Current Trends in Computing, RSCTC'2004*, volume 3066 of *LNAI*, pages 143–152. Springer-Verlag, 2004.
9. D. Dubois, J. Lang, and H. Prade. Fuzzy sets in approximate reasoning, part 2: logical approaches. *Fuzzy Sets and Systems*, 40(1):203–244, 1991.
10. D. Dubois and H. Prade. Fuzzy sets in approximate reasoning, part 1: inference with possibility distributions. *Fuzzy Sets and Systems*, 40(1):143–202, 1991.
11. D. Dubois and H. Prade. *Fuzzy Sets and Systems*. Fuzzy Logic CDROM Library. Academic Press, 1996.
12. V. Dubois and M. Quafafou. Concept learning with approximation: Rough version spaces. In J.J Alpigini, J.F. Peters, A. Skowron, and N. Zhong, editors, *Rough Sets and Current Trends in Computing: Proc. of the Third International Conf., RSCTC 2002*, number 2475 in LNAI, pages 239–246. Springer-Verlag, 2002.

13. B. Dunin-Kęplicz, L.A. Nguyen, and A. Szałas. Fusing approximate knowledge from distributed sources. In G.A. Papadopoulos and C. Badica, editors, *Proc. IDC'09, 3rd International Symposium on Intelligent Distributed Computing*, volume 237 of *Studies in Computational Intelligence*, pages 75–86. Springer, 2009.
14. B. Dunin-Kęplicz and A. Szałas. Towards approximate BGI systems. In H-D. Burkhard, G. Lindeman, L. Varga, and R. Verbrugge, editors, *Proc. CEEMAS 2007, 5th International Central and Eastern European Conf. on Multi-Agent Systems*, number 4696 in LNAI, pages 277–287. Springer-Verlag, 2007.
15. B. Dunin-Kęplicz. An architecture with multiple meta-levels for the development of correct programs. In *Proc. of Fourth International Workshop on Meta Programming in Logic*, number 883 in LNCS, pages 293–310, 1995.
16. B. Dunin-Kęplicz, L.A. Nguyen, and A. Szałas. Tractable approximate knowledge fusion using the Horn fragment of serial propositional dynamic logic. *Int. J. Approx. Reasoning*, 51(3):346–362, 2010.
17. Y. Gdalyahu and D. Weinshall. Measures for silhouettes resemblance and representative silhouettes of curved objects. In Bernard F. Buxton and Roberto Cipolla, editors, *Proceedings of the 4th European Conference on Computer Vision ECCV'96, Volume II*, volume 1065 of *Lecture Notes in Computer Science*, pages 365–375. Springer, 1996.
18. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.
19. U. Hustadt, B. Motik, and U. Sattler. Data complexity of reasoning in very expressive description logics. In L.P. Kaelbling and A. Saffiotti, editors, *Proc. of IJCAI-05*, pages 466–471. Professional Book Center, 2005.
20. R. Kruse, E. Schwecke, and J. Heinsohn. *Uncertainty and Vagueness in Knowledge Based Systems. Numerical Methods*. Springer-Verlag, 1991.
21. J. Llinas, C. Bowman, G. Rogova, A. Steinberg, E. Waltz, and F. White. Revisions and extensions to the JDL data fusion model II. In P. Svensson and J. Schubert, editors, *Proc. of the 7th Int. Conf. on Information Fusion*, 2004.
22. J. Maluszyński, A. Szałas, and A. Vitória. Paraconsistent logic programs with four-valued rough sets. In C.-C. Chan, J. Grzymala-Busse, and W. Ziarko, editors, *Proc. of 6th International Conf. on Rough Sets and Current Trends in Computing (RSCTC 2008)*, volume 5306 of *LNAI*, pages 41–51, 2008.
23. J. McCarthy. Approximate objects and approximate theories. In A.G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proc. 7th International Conf. on Principles of Knowledge Representation and Reasoning, KR2000*, pages 519–526, 2000.
24. L.A. Nguyen. On the deterministic Horn fragment of test-free PDL. In I. Hodkinson and Y. Venema, editors, *Advances in Modal Logic - Volume 6*, pages 373–392. King's College Publications, 2006.
25. L.A. Nguyen. Weakening Horn knowledge bases in regular description logics to have PTIME data complexity. In S. Ghilardi, U. Sattler, V. Sofronie-Stokkermans, and A. Tiwari, editors, *Proc. of Automated Deduction: Decidability, Complexity, Tractability ADDCT'07*, pages 32–47, 2007.
26. L.A. Nguyen. Constructing finite least Kripke models for positive logic programs in serial regular grammar logics. *Logic Journal of the IGPL*, 16(2):175–193, 2008.
27. Z. Pawlak. *Rough Sets. Theoretical Aspects of Reasoning about Data*. Kluwer Academic Publishers, Dordrecht, 1991.
28. H. Prade. A quantitative approach to approximate reasoning in rule-based expert systems. In L. Bolc and M.J. Coombs, editors, *Expert System Applications*, pages 199–256. Springer-Verlag, 1988.
29. A. Steinberg and C. Bowman. Revisions to the JDL data fusion model. In D. Hall and J. Llinas, editors, *Handbook of Multisensor Data Fusion*. CRC Press LLC, 2001.

30. J. van Benthem. Correspondence theory. In D. M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 2, pages 167–247. D. Reidel Pub. Co., 1984.
31. A. Vitória, J. Maluszyński, and A. Szałas. Modeling and reasoning in paraconsistent rough sets. *Fundamenta Informaticae*, 97(4):405–438, 2009.
32. X. Wang, B. De Baets, and E. Kerre. A comparative study of similarity measures. *Fuzzy Sets and Systems*, 73(2):259–268, 1995.
33. F. White. A model for data fusion. In *Proc. of 1st National Symposium for Sensor Fusion*, volume 2, 1988.
34. Yang Xiang. Distributed multi-agent probabilistic reasoning with bayesian networks. In Zbigniew W. Ras and Maria Zemankova, editors, *ISMIS*, volume 869 of *Lecture Notes in Computer Science*, pages 285–294. Springer, 1994.
35. Yang Xiang. Semantics of multiply selectioned bayesian networks for cooperative multi-agent distributed interpretation. In Gordon I. McCalla, editor, *Canadian Conference on AI*, volume 1081 of *Lecture Notes in Computer Science*, pages 213–226. Springer, 1996.
36. Yang Xiang and Victor R. Lesser. On the role of multiply sectioned bayesian networks to cooperative multiagent systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 33(4):489–501, 2003.
37. Y.Y. Yao and T.Y. Lin. Generalization of rough sets using modal logic. *Intelligent Automation and Soft Computing, An International Journal*, 2(2):103–120, 1996.

Barbara Dunin-Kęplicz is a Professor of computer science at the Institute of Informatics of Warsaw University and at the Institute of Computer Science of the Polish Academy of Sciences. She obtained her PhD in 1990 on computational linguistics from the Jagiellonian University, and in 2004 she was awarded her habilitation on formal methods in multi-agent systems from the Polish Academy of Sciences.

She is a recognized expert in multi-agent systems and one of the pioneers in the area of modeling BDI systems.

Linh Anh Nguyen is an assistant professor of computer science at the Institute of Informatics of Warsaw University. He obtained a PhD degree in 2000 and a hablitation degree in 2009 on modal logics from Warsaw University. He has published a considerable number of papers on modal logic programming, Horn fragments of modal logics, and tableau-based automated reasoning for modal and description logics.

Andrzej Szałas is a Professor of computer science at the Institute of Informatics of Warsaw University and at the Department of Computer and Information Science of the Linköping University. He obtained his degrees in computer science from Warsaw University: PhD in 1984 and habilitation in 1991. In 1999, he obtained the scientific title of professor from the President of Poland.

He is an expert in logics in computer science and artificial intelligence.

Received: February 9, 2010; Accepted: May 3, 2010.