On a Unifying Framework for Comparing Knowledge Representation Schemes

Giorgos Flouris, Dimitris Plexousakis, Grigoris Antoniou
Institute of Computer Science, FO.R.T.H.
P.O. Box 1385, GR 71110, Heraklion, Greece
{fgeo, dp, antoniou}@ics.forth.gr

Abstract

Given the numerous knowledge representation models (KR-schemes) that have been proposed, it would be desirable to have a formal, unifying model for the description of a KR-scheme, as well as a general method of comparing KR-schemes in terms of expressive power. This work attempts to fill this gap, by proposing an elegant, yet very general, model of describing KR-schemes. This formalization is used to describe any knowledge representation model, including databases, logic-based schemes, semantic networks etc. It is also applied to introduce a general comparison method for KR-schemes and a formal definition of the reduction of one scheme to another. Using this model, we can reason about KR-schemes in an abstract manner and to determine whether a certain reduction is possible or not.

1 Introduction

The task of comparing KR-schemes has received substantial attention in the literature [2, 6, 9, 15]. As stated in [2], such a comparison requires that a common framework is established between the compared schemes. Up to now, there have only been independent approaches to the problem, establishing such a framework only for the schemes under comparison. Thus, such methods are applicable (at best) to a small subset of the imaginable KR-schemes.

In this paper, we propose a general comparison method based on a formal definition of what a KR-scheme is. This definition is functional and generalizes the ASK and TELL operators (originally proposed in [11, 13]) to any KR-scheme. We conjecture that this generalization is enough to accommodate any imaginable KR-scheme. Using it, we can propose a method for comparing expressive power which is applicable to any pair of KR-schemes.

2 Definition of a KR-Scheme

2.1 Informal Definitions

The question of what KR is has rarely been directly addressed in the literature [5]. One of the most thorough attempts to define KR and other relevant terms is made in [13]. In that book, KR is defined as "the field of study within AI concerned with using formal symbols to represent a collection of propositions believed by some putative agent".

According to [8], "one can characterize a representational language as one which has (or can be given) a semantic theory". Thus, a KR-scheme (representational language) is a formal description of the symbols used for the representation, as well as their real world semantics. A Knowledge Base (KB) is then simply an instance of a KR-scheme, in the sense that it uses the symbols and the semantics of the given KR-scheme.

2.2 Formalizing a KR-scheme

Despite the vast variety of KR-schemes currently available, some general properties that hold for all such schemes can be identified. As mentioned above, all schemes are meant to be used by an entity, that we will generally refer to as the "user" of the system. The term "user" does not necessarily refer to a human being; a robot or a sophisticated software (e.g. a mediator, an agent, a data mining software, a web crawler etc) that in any way uses the KB can also be classified as a "user" of the system.

Moreover, all KBs store knowledge regarding a domain of interest in a specific and well-defined format. For example, in a propositional KB, the knowledge is represented by a propositional expression or a set of propositional expressions; in a relational database (DB), the knowledge is represented by a set of tables, which are instances of a certain DB schema. So, in any KR-scheme, there exists a (usually infinite) set of available possibilities (states) for the KB. We will call this set the *Knowledge Base set* (or the *KB set*) and denote it by S_K.

In order for the knowledge stored in a KB to be useful, we must equip the system with querying and updating capabilities [12]. As in [11, 13], we will use two functions, namely ASK (queries) and TELL (updates). For the query mechanism, the system designer must have provided the system with a query language; the acceptable expressions of this language form a set, the *Query set*, S_Q. The acceptable replies to a query are likewise predetermined; they form another set, the *Answer set*, S_A. The ASK function is the algorithm that evaluates a query against the current KB and returns the proper answer. Given the above sets, the ASK function (*Query function*) can be formally defined as: $ASK : S_K \times S_Q \rightarrow S_A$. Similarly, to perform updates, a set of acceptable updates must be provided by the system designer, forming the *Update set*, S_U. The TELL function (*Update function*) is the algorithm that maps the current KB to a new one, depending on the update performed. It can be formally defined as: $TELL : S_K \times S_U \rightarrow S_K$.

Notice that we pose no restrictions on the contents of the above sets, but

we will require that they are non-empty. The functions ASK and TELL must be total, but they can be freely defined otherwise; no rationality constraints are imposed upon them. Thus:

Definition 1 A KR-scheme is a 6-tuple (S_K, S_U, S_Q, S_A, ASK, TELL), where S_K, S_U, S_Q, S_A are non-empty sets and $ASK : S_K \times S_Q \rightarrow S_A$, $TELL : S_K \times S_U \rightarrow S_K$, are total functions.

One may argue that this definition, though general, does not take into account some of the most sophisticated advances in DB technology, such as triggers, user profiles, active rules or integrity constraints. In fact, it does! As will be made clear in the following examples, the concept of the set is so general that virtually any type of (typically homogeneous) information can be in a set. Regarding the user interaction, it can be classified in two types. Any operation that only reads data from the KB ("read operation"), can be modelled using the ASK function in our framework, even though it may not be a query in the conventional sense (such as the query: "what integrity constraints are there in the DB?"). Any operation that writes data to the KB ("write operation") can be modelled using the TELL function, even though it may not be an update in the conventional sense (such as the update: "insert the integrity constraint C to the DB").

The four sets of our structure contain all the possible *symbols* (or symbol sequences) that represent the *knowledge* of the real world. So the four sets are at the symbol level of our schema, describing the way that the real world will be represented. The two functions are the ones that actually give meaning to the symbols, representing the knowledge level of the schema [14].

Our model unifies several diverse ways of thinking about how knowledge should be represented. In [16], natural language and logic are presented as universal languages for all KR-schemes. We conjecture that all KR-schemes and meta-models, including natural language and logic, can be modelled using this 6-tuple. A few examples will provide some insight on the reasons behind this conjecture.

2.3 Formalization Examples

Initially, we will try to model propositional KBs under this formalism. We will use the AGM paradigm [1]. Assume a propositional language L and the set of all the wff in L, denoted by L^*. Then a KB is a set of propositions, i.e. an element of $P(L^*)$ (the powerset of L^*); not all elements of $P(L^*)$ apply, as the set must be closed under logical consequence. Let us denote by D the subset of $P(L^*)$ whose elements (sets of propositions) are closed under logical consequence. Then $S_K = D$. Any proposition in L^* can be an update or a query, so $S_U = S_Q = L^*$. We will use the Closed World Assumption, where the possible answers to any query are either YES or NO, so $S_A = \{YES, NO\}$ and the Query function ASK is defined as: ASK(K,Q)=YES iff $K \models Q$; else ASK(K,Q)=NO. The TELL function can be any belief revision function (algorithm) satisfying the AGM postulates.

The case of the relational model is somewhat more complex. We will

initially take the simple model, where only tables are used (integrity constraints, triggers etc are not allowed). We will assume that SQL is the data manipulation language. Each DB schema that can be defined is a KR-scheme of its own. Thus for a DB schema S, we can say that it defines a KR-scheme where S_K contains all the possible instances of S, S_U contains all acceptable UPDATE, INSERT and DELETE operations upon S and S_Q contains all the possible SELECT operations. The answer to a query can be any set of tuples. The ASK function is the algorithm that evaluates user queries and TELL function is the algorithm that performs the updates upon the DB; both are implemented in all DBMSs.

A more complex system should also include user profiles, triggers, integrity constraints, views etc. A KB should include this kind of information; thus S_K should be expanded to include such objects. Similarly S_U should be expanded to allow changes upon such objects (for example UPDATE TRIGGER operations) and S_Q should be expanded to allow questions upon such objects (for example: "what are the preconditions of trigger T?"). The implementations of SQL that appear in commercial DBMSs include such facilities. Similarly, the Answer set should be expanded to be able to answer questions regarding these objects and the ASK and TELL functions should implement all the above expansions.

Proceeding one step further, we could also consider each DB schema as a KB on its own right (at a meta-level). Commercial DBMSs allow the creation and manipulation of the DB schema, by allowing the user to create tables or change the definition of existing ones. The KR-scheme of DB schemes can also be modelled using our approach. In this case, it is important to discriminate between a KR-scheme and a DB scheme: we are trying to model a KR-scheme whose contents are DB schemes. The KB set (S_K) of this scheme should contain all the possible sets of tuple definitions (DB schemas) that can be created using elements from a given, fixed domain D. The Update set comprises all the commands that alter a schema (implementations of SQL in commercial DBMSs include this facility) and the Query set should contain all the possible questions upon the schema. The Answer set should contain all the possible answers, while the ASK and TELL functions represent the algorithms implementing these operations.

3 Comparing Expressive Power

With the above formalism at hand, comparing two given KR-schemes is possible, even if these two schemes seem to have nothing in common. We will introduce two different methods of comparison; deciding on the method that most properly fits our intuitive notion of a scheme being more expressive than another is a matter of current research.

3.1 Normal Reduction

Consider two KR-schemes S_1=(S_{K1}, S_{U1}, S_{Q1}, S_{A1}, ASK_1, $TELL_1$), S_2= (S_{K2}, S_{U2}, S_{Q2}, S_{A2}, ASK_2, $TELL_2$) and suppose that we want to prove

that S_2 is more expressive than S_1. If this is true, then we should be able to "substitute" each symbol (or symbol sequence) in S_1 with a symbol (or symbol sequence) in S_2. This substitution should be made in such a way that there is no loss of information during the transition. If the semantics of the original symbols is preserved, then no loss of information occurs; we may change the symbols, but not the knowledge they represent. If S_2 is strictly more expressive than S_1 then it may be able to express knowledge not available in S_1; in this case the opposite reduction (from S_2 to S_1) is not possible. It might also be the case that S_1 and S_2 are incomparable; this could happen if each scheme contains knowledge which cannot be properly mapped to the other system.

More formally, each possible KB in S_1 (any $K_1 \in S_{K1}$) must be mapped to a KB in S_2 (say $K_2 \in S_{K2}$). This mapping is a function, which will be called *Knowledge Base Reduction function* and denoted by $f_K : S_{K1} \rightarrow S_{K2}$. This mapping is not sufficient, because a KR-scheme also consists of updates, queries and answers to queries. Thus, similar assignments must be made for the Update and Query sets, using the *Update Reduction function*, $f_U : S_{U1} \rightarrow S_{U2}$, and the *Query Reduction function*, $f_Q : S_{Q1} \rightarrow S_{Q2}$, respectively. For the Answer set, the opposite assignment must be made; once the substitution has been made, answers will be obtained by S_2 (function ASK_2), so we need a way to "translate" these answers to answers that S_1 understands (belonging in S_{A1}). So, each possible answer in the substituting system (S_2), will be mapped to one answer in the substitutable system (S_1) by the *Answer Reduction function*, $f_A : S_{A2} \rightarrow S_{A1}$.

Notice that the existence of these four functions by itself should not constitute evidence that S_2 is more expressive than S_1. Moreover, requesting that these are 1-1 and/or onto is irrelevant for our cause, because our goal is to preserve the semantics, not the symbols. The conditions required should be related to the symbol manipulation functions ASK and TELL and guarantee the preservation of semantics during the transition from S_1 to S_2.

The first property is related to the query mechanism. Assume a KB $K_1 \in S_{K1}$, a query $Q_1 \in S_{Q1}$ and their assigned KB $K_2 = f_K(K_1) \in S_{K2}$, and query $Q_2 = f_Q(Q_1) \in S_{Q2}$. Suppose that: $A_1 = ASK_1(K_1, Q_1) \in S_{A1}$ and $A_2 = ASK_2(K_2, Q_2) \in S_{A2}$. These two answers must be the "same", in the sense that A_1 must be the respective answer of A_2 under the Answer Reduction function ($A_1 = f_A(A_2)$). Thus, we get (see also Figure 1):

$$ASK_1(K_1, Q_1) = f_A(ASK_2(f_K(K_1), f_Q(Q_1)))$$
$$\text{for all } K_1 \in S_{K1},\ Q_1 \in S_{Q1}$$

This will be called the *Query Preservation property.*

The second property is related to the update mechanism. Consider, two KBs $K_1, K_2 \in S_{K1}$ and an update $U_1 \in S_{U1}$, as well as their respective KBs $f_K(K_1), f_K(K_2) \in S_{K2}$ and update $f_U(U_1) \in S_{U2}$. We would expect that the KBs K_1 and K_2 are related via the update U_1 and the

$TELL_1$ function ($K_2 = TELL_1(K_1, U_1)$), iff the KBs $f_K(K_1)$ and $f_K(K_2)$ are related via the update $f_U(U_1)$ and the $TELL_2$ function ($f_K(K_2) = TELL_2(f_K(K_1), f_U(U_1))$), that is (see Figure 1):

$$\begin{array}{c} \text{For any } K_1, K_2 \in S_{K1}, U_1 \in S_{U1}: \\ K_2 = TELL_1(K_1, U_1) \text{ iff} \\ f_K(K_2) = TELL_2(f_K(K_1), f_U(U_1)) \end{array}$$

This will be called the *Update Preservation property.*

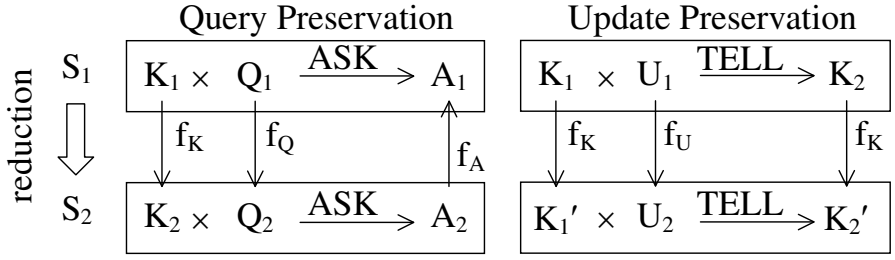

Figure 1: Query and Update Preservation properties

Combining the above, we define:

Definition 2 Consider two KR-schemes S_1=(S_{K1}, S_{U1}, S_{Q1}, S_{A1}, ASK_1, $TELL_1$) and S_2=(S_{K2}, S_{U2}, S_{Q2}, S_{A2}, ASK_2, $TELL_2$). If there exists a 4-tuple of total functions (f_K, f_U, f_Q, f_A), $f_K : S_{K1} \to S_{K2}$, $f_U : S_{U1} \to S_{U2}$, $f_Q : S_{Q1} \to S_{Q2}$, $f_A : S_{A2} \to S_{A1}$, such that the Query and Update Preservation properties hold, then we say that S_1 *can be (normally) reduced* to S_2. The 4-tuple (f_K, f_U, f_Q, f_A) will be called the *(normal) reduction algorithm.* If S_1 can be reduced to S_2 then we say that S_2 *is (normally) at least as expressive as* S_1, denoted by $S_1 \leq_r S_2$. Analogously we define $\cong_r$ (*equivalent expressivity*) and $<_r$ (*strictly greater expressivity*).

3.2 Behavioral Reduction

One may argue that the semantics of a KB is actually determined by the query answers upon the KB. If two KBs give the same answers to all queries, then they carry the same information, as far as the user is concerned. Having this in mind, the Update Preservation property may look a little more restrictive than necessary. It would be enough to demand that the KBs $K_1' = TELL_1(K_1, U_1) \in S_{K1}$ and $K_2' = TELL_2(K_2, U_2) \in S_{K2}$, give the "same" answers to queries, in the sense of the Query Preservation property. In this case the user is not able to notice the transition and the semantics of the original system are preserved. However, this is not enough; the user can perform a sequence of updates (instead of just one) upon the original KB, and the resulting KBs (in the two schemes) must be indistinguishable by the user after this sequence of updates. Indistinguishable in this context means that the two KBs should give the "same" answers to all queries, in

the sense of the Query Preservation property as before. This generalized property will be called the *Behavior Preservation* property.

To formalize this property, we define the *iterated update* (*iterated TELL*) operator, which calculates a sequence of updates upon a KB:

Definition 3 Consider a function $TELL : S_K \times S_U \rightarrow S_K$. We define, for any $m \geq 0$, the iterated TELL function, $TELL^m$ as:
$TELL^0 : S_K \rightarrow S_K$, $TELL^0(K) = K$ for all $K \in S_K$
$TELL^1 : S_K \times S_U \rightarrow S_K$, $TELL^1(K,U) = TELL(K,U)$ for all $K \in S_K$, $U \in S_U$
$TELL^m : S_K \times S_U^m \rightarrow S_K$, $TELL^m(K,U_1,\ldots,U_m) = TELL(TELL^{m-1}(K, U_1,\ldots,U_{m-1}),U_m)$, for all $K \in S_K, U_1,\ldots,U_m \in S_U$

Using this operator, we can write the Behavior Preservation property as follows (see also Figure 2):

$$f_A(ASK_2(TELL_2^m(f_K(K_1), f_U(U_1),\ldots,f_U(U_m)), f_Q(Q_1))) = \\ = ASK_1(TELL_1^m(K_1,U_1,\ldots,U_m),Q_1) \\ \text{for all } m \geq 0,\ K_1 \in S_{K1},\ U_1,\ldots,U_m \in S_{U1},\ Q_1 \in S_{Q1}$$

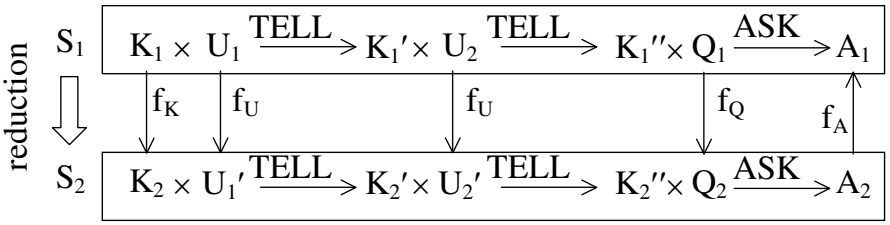

Figure 2: Behavior Preservation property (for m=2)

Notice that the Query Preservation property is a special case of the Behavior Preservation property (for $m = 0$). The latter does not ensure that the transition keeps the knowledge of the original system intact (the KBs need not be the "same"), but it does ensure that the *behavior* of the two systems will be identical and indistinguishable by the user (the answers must be the "same"); the semantics are preserved in both cases. We will call the type of reduction based on this property *behavioral reduction*:

Definition 4 Consider two KR-schemes S_1=(S_{K1}, S_{U1}, S_{Q1}, S_{A1}, ASK_1, $TELL_1$) and S_2=(S_{K2}, S_{U2}, S_{Q2}, S_{A2}, ASK_2, $TELL_2$). If there exists a 4-tuple of total functions (f_K, f_U, f_Q, f_A), $f_K : S_{K1} \rightarrow S_{K2}$, $f_U : S_{U1} \rightarrow S_{U2}$, $f_Q : S_{Q1} \rightarrow S_{Q2}$, $f_A : S_{A2} \rightarrow S_{A1}$, such that the Behavior Preservation property holds, then we say that S_1 *can be behaviorally reduced* to S_2. The 4-tuple (f_K, f_U, f_Q, f_A) will be called the *behavioral reduction algorithm*. If S_1 can be behaviorally reduced to S_2 then we say that S_2 *is behaviorally at least as expressive as* S_1, denoted by $S_1 \leq_b S_2$. Analogously we define $\cong_b$ (*behaviorally equivalent expressivity*) and $<_b$ (*strictly greater behavioral expressivity*).

4 Discussion

4.1 Initial Results and Examples for Reduction

In this section we will state some results related to our framework. The relevant proofs, as well as a more thorough discussion on the issue can be found in [7]. Some intuitively desired properties of expressiveness relations and the connection between the two types of reduction can be initially proved:
Proposition 1 For all KR-schemes S_1, S_2, if $S_1 \leq_r S_2$ then $S_1 \leq_b S_2$. Similarly, if $S_1 \cong_r S_2$ then $S_1 \cong_b S_2$. The opposite entailments do not hold.
Proposition 2 The relations $\leq_r$ and $\leq_b$ are reflexive and transitive. The relations $\cong_r$ and $\cong_b$ are symmetric, reflexive and transitive.

Some examples may help uncover some less obvious properties of the above relations. At first, assume any KR-scheme based on some form of logic (say propositional). We will denote by L^* the set of all wffs of the selected logic and S_{OWA}, S_{CWA} the KR-schemes that occur by using L^* and the Open World Assumption (OWA) and Closed World Assumption (CWA) respectively. Then, it can be proven that $S_{OWA} >_r S_{CWA}$. Indeed: S_{OWA}=(S_K, S_U, S_Q, S_{AOWA}, ASK_{OWA}, TELL) and S_{CWA}=(S_K, S_U, S_Q, S_{ACWA}, ASK_{CWA}, TELL), where $S_K = S_U = S_Q = L^*$, and TELL is some function that performs updates (any function would do). For S_{OWA}, we have $S_{AOWA} = \{YES, NO, UNKNOWN\}$, and $ASK_{OWA}(K, Q) = YES$ iff $K \models Q$; $ASK_{OWA}(K, Q) = NO$ iff $K \models \neg Q$; else $ASK_{OWA}(K, Q) = UNKNOWN$. For S_{CWA} we have $S_{ACWA} = \{YES, NO\}$ and ASK_{CWA} (K,Q)=YES iff $K \models Q$; else $ASK_{CWA}(K, Q) = NO$. We take f_K, f_U, f_Q to be the identity functions of their respective domains and $f_A(YES) = YES$, $f_A(NO) = f_A(UNKNOWN) = NO$. Then: $ASK_{OWA}(K, Q) = YES$ iff $K \models Q$ iff $f_K(K) \models f_Q(Q)$ iff $ASK_{CWA}(f_K(K), f_Q(Q)) = YES$. Using this equivalence and the definition of f_A, it is easy to prove that the Query and Update Preservation properties hold. Thus, by definition, $S_{OWA} \geq_r S_{CWA}$. For the opposite reduction, notice that all three answers are needed in S_{OWA}, so we cannot identify any two answers during the transition to S_{CWA} using f_A. On the other hand, f_A cannot be 1-1, because S_{AOWA} is finite and contains more elements than S_{ACWA}. Thus, the opposite reduction is not possible. This argument is a special case of a more complex condition (see proposition 4 below) regarding the "size" of the sets in a KR-scheme and the possibility of reducing a scheme to another. Therefore: $S_{OWA} >_r S_{CWA}$.

In [3, 4], a belief revision algorithm was introduced based on propositional KBs. The papers described the update mechanism, but the underlying KR-scheme was implicitly described. Dalal used propositional KBs, so $S_K = S_U = S_Q = L^*$, where L^* is the set of all wff of the underlying propositional language L. We assume CWA, so $S_A = \{YES, NO\}$ and the ASK function is the CWA query answering function as in the above example. The TELL function is the belief revision algorithm described in Dalal's papers. As proved in [10], Dalal's algorithm satisfies the AGM postulates

[1], so for any two KBs $K_1, K_2 \in S_K$, with $K_1 \equiv K_2$ (where the symbol "$\equiv$" stands for logical equivalence) and any $U_1, U_2 \in S_U$ with $U_1 \equiv U_2$ it holds that $TELL(K_1, U_1) \equiv TELL(K_2, U_2)$. Moreover, by the definition of ASK, for all $K_1, K_2 \in S_K$ with $K_1 \equiv K_2$ and all $Q_1, Q_2 \in S_Q$ with $Q_1 \equiv Q_2$: $ASK(K_1, Q_1) = ASK(K_2, Q_2)$. Thus, it makes sense to restrict ourselves to $L^*/\equiv$ (instead of L^*) for the sets S_K, S_U and S_Q and have an equivalent KR-scheme. This does happen, as the two schemas can be proven behaviorally equivalent.

4.2 Redundancy

In fact, the above equivalence is an application of a more general property having to do with "redundant" elements. We say that redundancy appears when two or more symbols (or symbol sequences) from one of the sets S_K, S_U, S_Q represent the "same" real-world knowledge (KB, update or query respectively). Being able to express the same information with more than one way leads to redundancy. From a practical viewpoint, it might sometimes be desirable to allow the user to express information with different ways; this usually makes the system more user-friendly. However, for any scheme containing redundancy, we should be able to find an equivalent one that does not contain redundancy, thus eliminating redundant elements without loss of expressive power. On the contrary, the elimination of non-redundant elements from a scheme should result in the loss of expressive power. For the Answer set, redundancy appears when S_A contains answers that the system never actually gives. Such answers (only) are useless and could be dropped without loss of expressive power.

To formally express the above considerations, we must initially resolve what we mean by the term "same knowledge". Once again, there are two viewpoints, the "normal" and the "behavioral" one, depending on the comparison method we use:

Definition 5 Let S=(S_K, S_U, S_Q, S_A, ASK, TELL) be a KR-scheme. We say that S contains:

- *Normal KB redundancy* iff there exist K_1, $K_2 \in S_K$ $K_1 \neq K_2$, such that:
  For all $Q \in S_Q$: $ASK(K_1, Q) = ASK(K_2, Q)$
  For all $U \in S_U$: $TELL(K_1, U) = TELL(K_2, U)$
  There is no $K_0 \in S_K, U_0 \in S_U$: $TELL(K_0, U_0) = K_1$
  There is no $K_0 \in S_K, U_0 \in S_U$: $TELL(K_0, U_0) = K_2$
- *Normal update redundancy* iff there exist $U_1, U_2 \in S_U$ $U_1 \neq U_2$, such that for all $K \in S_K$: $TELL(K, U_1) = TELL(K, U_2)$.
- *Normal query redundancy* iff there exist $Q_1, Q_2 \in S_Q$ $Q_1 \neq Q_2$, such that for all $K \in S_K$ $ASK(K, Q_1) = ASK(K, Q_2)$.
- *Normal answer redundancy* iff there exists an $A \in S_A$, such that for all $K \in S_K$, $Q \in S_Q$, $ASK(K, Q) \neq A$.
- *Normal redundancy* iff it contains any type of (normal) redundancy.

- *Behavioral KB redundancy* iff there exist $K_1, K_2 \in S_K$ $K_1 \neq K_2$, such that for all $m \geq 0$, $U_1,\ldots,U_m \in S_U$, $Q \in S_Q$:
  $ASK(TELL^m(K_1, U_1, \ldots, U_m), Q) =$
  $= ASK(TELL^m(K_2, U_1, \ldots, U_m), Q)$
- *Behavioral update redundancy* iff there exist $U_1, U_2 \in S_U$ $U_1 \neq U_2$, such that for all $m \geq 0$, $K \in S_K$, $U'_1, \ldots, U'_m \in S_U$, $Q \in S_Q$:
  $ASK(TELL^{m+1}(K, U_1, U'_1, \ldots, U'_m), Q) =$
  $= ASK(TELL^{m+1}(K, U_2, U'_1, \ldots, U'_m), Q)$
- *Behavioral query redundancy* iff it contains normal query redundancy
- *Behavioral answer redundancy* iff it contains normal answer redundancy
- *Behavioral redundancy* iff it contains any type of behavioral redundancy.

Using this definition we can prove that redundant elements do not add expressive power to a scheme:

Proposition 3 Suppose any schema S. There exist schemas S_r, S_b, such that S_r does not contain normal redundancy, S_b does not contain behavioral redundancy and $S_r \cong_r S$, $S_b \cong_b S$.

This proposition implies that we lose nothing by restricting our attention to non-redundant schemes only. When S does not contain redundancy, then S itself is the schema that satisfies this proposition. However, if S does contain some type of redundancy, then we can get rid of the redundant elements without loss of expressive power. Notice that the proposition can be specialized [7], so that each type of redundancy (KB, update, query, answer) can be checked separately. We could apply this property in all belief revision algorithms that satisfy the AGM postulate of preservation (#5), like Dalal's algorithm described above. By proposition 3, all such algorithms have simpler behavioral equivalents.

The proposition below implies that only the redundant elements can be removed without loss of expressive power:

Proposition 4 Suppose two KR-schemes S_1, S_2. If $S_1 \leq_r S_2$ and S_1 contains no normal redundancy, then for any normal reduction algorithm from S_1 to S_2, (f_K, f_U, f_Q, f_A), it holds that f_K, f_U, f_Q are 1-1 functions and f_A is onto. Similarly, if $S_1 \leq_b S_2$ and S_1 contains no behavioral redundancy, then for any behavioral reduction algorithm from S_1 to S_2, (f_K, f_U, f_Q, f_A), it holds that f_K, f_U, f_Q are 1-1 functions and f_A is onto.

Thus, if the non-redundant scheme S_1 can be reduced to another schema S_2, then the latter is required to have sets of at least equal cardinality, so that each symbol of S_1 can be mapped to a different symbol in S_2. Once again [7], the above proposition can be further specialized: for any type of redundancy that S_1 does not contain, the respective function of the reduction algorithm should be 1-1 or onto (depending on the case). Thus,

cardinality becomes important when determining the feasibility of a certain reduction involving non-redundant schemas (because non-redundant symbol sets cannot be reduced in size during the reduction).

When attempting a reduction we could check the existence of such functions (1-1 or onto, depending on the case) for each set of the two schemes under question. This can be easily done using the results provided by set theory. If such functions do not exist, we can safely conclude by proposition 4 that the reduction is not possible, thus saving some futile search for a reduction algorithm. In such a case, this proposition may prove useful in identifying, using formal methods instead of simple intuition, what the second KR-scheme lacks in order to be more expressive than the first (for example enriching the query language could do, while enriching the KB set could prove useless). This property is especially useful when dealing with finite sets, where the famous "Pigeonhole Principle" applies. On the other hand, the existence of such functions does not guarantee that the reduction is possible; it is merely an indication that it is.

5 Conclusion and Future Work

We introduced a general method for comparing any pair of KR-schemes, no matter how different, in terms of expressive power. It was based on a universal method of formally describing a KR-scheme and used the notion of "reduction", which refers to the "translation" of the knowledge expressed under one scheme in terms of the other without loss of information.

This work is intended as an aid to such attempts of "translations" (like [2, 6, 9, 15]). Ideally, we would like to find an algorithm that could compare two schemes in terms of expressive power; using our framework, we can prove that this is an undecidable problem in the general case and intractable in the special case where the two schemas contain finite symbol sets. Given this negative result, we would like to establish other conditions under which a reduction is or is not possible (like proposition 4). Such results could be used to prove or disprove the existence of a reduction algorithm before any attempt to find one.

We could also expand the notion of reduction; one way to do so is by allowing the mapping of each query/update of the weaker system to a sequence of queries/updates in the stronger one (instead of only one). Another interesting (but less general) method can be found in [16]. The properties of each of the comparison methods is a matter of current research.

Finally, given that each DB schema can be considered a separate KR-scheme, we can study the limitations of the expressive power of each DB schema in terms of some other interesting KR-scheme with known expressive power or with respect to a specific real world application.

References

[1] C. Alchourron, P. Gardenfors, D. Makinson. On the Logic of Theory Change: Partial Meet Contraction and Revision Functions. *The Journal of Symbolic Logic*, 50: 510-530, 1985.

[2] A. Borgida. On the Relative Expressiveness of Description Logics and Predicate Logics. *Artificial Intelligence*, 82(1-2):353-367, 1996.

[3] M. Dalal. Investigations Into a Theory of Knowledge Base Revision: Preliminary Report. In *Proceedings of the Seventh National Conf. on Artificial Intelligence*, 475-479, 1988.

[4] M. Dalal. Updates in Propositional Databases. Technical Report, DCS-TR-222, Dept. of Computer Science, Rutgers Univ., 1988.

[5] R. Davis, H. Shrobe, P. Szolovitz. What is a Knowledge Representation? *AI Magazine*, 14(1): 17-33, 1993.

[6] G. Flouris, D. Plexousakis. Belief Revision Using Table Transformation. Technical Report ICS-FORTH, TR-290, 2001.
URL: http://www.ics.forth.gr/isl/publications/paperlink/flourisTR-290.pdf

[7] G. Flouris, D. Plexousakis, G. Antoniou. Describing Knowledge Representation Schemes: a Formal Account. Technical Report ICS-FORTH, TR-320, 2003.
URL: http://www.ics.forth.gr/isl/publications/paperlink/TR-320April2003.pdf

[8] P. Hayes. The logic of frames. In Metzing, editor, *Frame Conceptions and Text Understanding*. de Gruyter, Berlin, 1979.

[9] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl. RQL: A Declarative Query Language for RDF. In *Proceedings of the 11th International Conf. on the WWW*, Hawaii, 592-603, 2002.

[10] H. Katsuno, A. Mendelzon. Propositional Knowledge Base Revision and Minimal Change. Technical Report KRR-TR-90-3, Technical Reports on Knowledge Representation and Reasoning, Univ. of Toronto, 1990.

[11] H. Levesque. Foundations of a Functional Approach to Knowledge Representation. *Artificial Intelligence*, 23:155-212, 1984.

[12] H. Levesque, R. Brachman. Expressiveness and Tractability in Knowledge Representation and Reasoning. *Computational Intelligence*, 3: 78-93, 1987.

[13] H. Levesque, G. Lakemeyer. *The Logic of Knowledge Bases*. MIT Press, Cambridge, Massachusetts, 2000.

[14] A. Newell. The Knowledge Level. *Artificial Intelligence*, 18(1): 87-127, 1982.

[15] D. Plexousakis. Semantical and Ontological Considerations in Telos: a Language for Knowledge Representation. *Computational Intelligence*, 9(1): 41-72, 1993.

[16] J. Sowa. *Knowledge Representation*. Brooks/Cole, Pacific Grove, California, 2000.