Using Datalog to provide just-in-time feedback during the construction of concept maps

Post-Print

Francisco J. Álvarez-Montero[1], Fernando Sáenz-Pérez[2], and Antonio Vaquero-Sánchez[3]

[1] Universidad Autónoma de Sinaloa francisco_alvarez_montero@uas.edu.mx
[2] Universidad Complutense de Madrid fernan@sip.ucm.es
[3] Universidad Complutense de Madrid arvsmcmg@telefonica.net

Abstract

Concept maps have been extensively used in education, especially in science teaching. There is strong evidence that their use is associated with increased knowledge transfer and retention across several instructional conditions, settings and methodological features. However, constructing a concept map is complex and difficult for students, especially newbies. Consequently, there is a necessity to provide feedback to the learners during the authoring of their concept maps. There are several concept-mapping tools that provide feedback but none of them provide immediate or just-in-time (JIT) feedback. This kind of feedback is important for two reasons: First, low achieving or low mastery students benefit greatly from this type of feedback. Second, when students start out badly, with incorrect propositions, they tend to continue with further incorrect propositions until the map is grossly incorrect and JIT feedback could prevent this situation. This paper presents a practical application of Ohlsson's theory of learning from performance errors to provide JIT feedback during the construction of concept maps. It is shown that by creating an Entity-Relationship (E-R) schema that incorporates additional elements into the standard schema for concept maps, the schema can be implemented with Datalog, benefiting from the use of its deductive features to provide immediate feedback to the learner. Finally, some field related examples are provided.

Keywords: Concept maps; Just-in-time feedback; Learning from performance errors; Binary relations; Constraints; Entity-Relationship schema; Datalog.

1. Introduction

Concept maps (Novak & Gowin, 1984) are the product of mapping one or more categorical propositions (Hurley, 2010). These propositions are composed of two classes, known as the referent and the relatum, and a term, representing a binary or dyadic relation (Nagel & Cohen, 1993). Graphically, these elements take the form of nodes and labeled directed arcs, respectively. The nodes represent concepts or ideas within a subject area or domain, and the labeled directed arcs are binary relations which explain how two concepts are related.

As an educational tool, concept maps are based on the notion that concept interrelatedness is an essential property of knowledge, and the empirical finding that content understanding (for example, of a school subject) is represented by well-structured knowledge (O'Neil & Klein, 1997; Ruiz-Primo et al., 2001). They have been applied to enhance both individual and collaborative learning, and there is strong evidence that their use is associated with increased knowledge

transfer and retention across several instructional conditions, settings and methodological features (Daley & Torre, 2010; Horton et al., 1993; Nesbit & Adesope, 2006). Additionally, their use in education, according to different researchers (Anohina-Naumeca, Grundspenkis and Strautmane, 2011; Ruiz-Primo et al., 2001; Yin et al., 2005), can be characterized along a continuum from high-directed to low-directed. The more elements are provided to the learners, the higher the degree of directedness and vice versa. Figure 1 shows some of the components of concept maps that can be provided by the teacher or that can be left for the students to create on their own.

However, despite their graphical simplicity, and no matter the degree of directedness, the construction of concept maps is complex and difficult for students, especially for newbies (Chang, Sung and Chen, 2001; Cimolino, Kay & Miller, 2003). Consequently, learner support or feedback during the construction of a concept map is recommended (Coffey et al., 2003). Feedback helps learners determine performance expectations, judge their level of understanding, and become aware of misconceptions (Mason & Bruning, 2001). Without appropriate feedback, as Chang, Sung & Chen (2001) point out, learners have few opportunities to reflect upon their own thinking, and this reduces the beneficial effects of constructing a concept map.

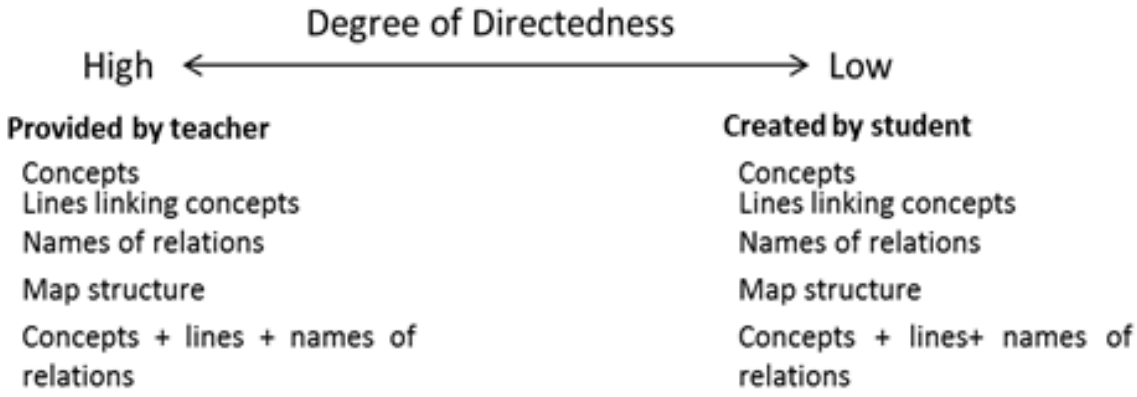

Figure. 1. Degree of directedness in concept mapping tasks.

In this sense, feedback must be differentiated from the assessment or diagnosis of concept maps. The latter addresses the question of how to measure the quality of a concept map by assigning a score, after the time allocated for the concept mapping task is over (Anohina & Grundspenkis, 2009). Feedback, in turn, can be defined as any message generated in response to a learner's action (Mason & Bruning, 2001). There are several types of feedback. In this paper, feedback must be understood as immediate or just-in-time (JIT) feedback, which is a kind of feedback that is automatically given to the learner, when he or she commits an error. Immediate feedback has been successfully implemented in many Intelligent Tutoring Systems (Nwana, 1990; Graesser, Colney & Olney, in press) and there is evidence showing that low achieving or low mastery students benefit greatly from this type of feedback (Mason & Bruning, 2001; Shute, 2008). Additionally, some researchers such as Cimolino, Key and Miller (2003) have found that when students start out badly, with incorrect propositions, they tend to continue with further incorrect propositions until the map is grossly incorrect. Just-in-time, in principle, could help prevent this situation.

There are several concept mapping tools that provide feedback to the learner (Anohina-Naumeca, Grundspenkis & Strautmane, 2011; Chang, Sung & Chen, 2001; Cimolino, Kay & Miller, 2003; Gouli, Gogoulou& Grigoriadou, 2009). However, in all these tools feedback is on demand (explicitly requested by the user) or it is delayed until the concept mapping task is finished. This lack of immediate feedback motivated the following research question: is it possible to provide just-in-time feedback to the learner during the construction of a concept map?

Following our previous work (Álvarez-Montero, Sáenz-Pérez, Vaquero & Jacobo-García, 2012), in this paper, it is shown that by creating a conceptual schema that incorporates two sets of properties into the binary relations of concept maps, and implementing it as a Datalog schema, it is possible to provide just-in-time feedback for high-directed concept mapping tasks, where the concepts and relations have previously been defined. By using Datalog (Ceri, Gottlob & Tanca, 1989), the two sets of properties can be represented as constraints and used to provide immediate feedback to the learner every time he or she makes a mistake, that is, every time a constraint is violated.

This approach to just-in-time feedback is based on the core ideas of the Theory of Learning from Performance Errors (Ohlsson, 1996, 2011) and the Constraint Based Modelling paradigm for Intelligent Tutoring Systems (Chrysafiadi & Virvou, 2013; Graesser, Conley & Olney, in press; Ohlsson & Mitrovic 2007) which focus on what properties a good solution must have and posit that a correct solution can never violate the constraints that follow from these properties.

For implementation purposes, the Entity-Relationship (E-R) model (Chen, 1976) is used to create the conceptual schema, and the Datalog Educational System (DES) (Sáenz-Pérez, 2011) is the deductive system employed to capture the data, the constraints on these data, and provide the feedback. The rationale is that since Datalog and the E-R model are based on the relational data model (Chen, 1976; Ullman, 1988), the conceptual schema can be easily mapped and implemented as a Datalog schema. In addition, thanks to the more expressive data model, more complex constraints (i.e., including non-linear recursion and duplicate elimination) can be stated.

The rest of the article is organized as follows: First, Ohlsson's theory of learning from performance errors is summarized. Second, the properties of binary relations necessary to provide just-in-time feedback, their inclusion in the standard structure of concept maps and its implementation using DES are addressed and presented. Finally, some conclusions and future work are discussed.

2. Learning from performance errors

The theory of learning from performance errors (Ohlsson, 1996, 2011) states that, although humans have the innate ability to catch themselves making errors, this ability has imperfections, as Gilovich (1991) points out. Consequently, anyone can make a mistake. For instance, declaring a false statement or drawing an incorrect conclusion. The explanation is that this happens because there is a disassociation between someone's declarative and practical knowledge. Practical knowledge, also known as procedural (Ohlsson, 1996) or generative knowledge (Ohlsson & Mitrovic, 2007), is a set of rules for generating actions or behaviors that have some probability of being appropriate, correct or useful in a particular context. Declarative knowledge, in turn, enables a person to evaluate the outcome of an action or behavior, and judge it to be correct or incorrect.

Consequently, in order to learn from errors and eliminate the disassociation, a learner needs to reflect on the outcomes of his/her actions. And for that purpose, declarative knowledge in the form of integrity constraints is therefore appropriate. These constraints function as self-monitoring devices by which the learner can evaluate or judge the correctness of the action sequence generated by his/her (possibly incomplete or incorrect) practical knowledge (Ohlsson, 1996). This way, it is possible for a student to modify or update his procedural or generative knowledge base by inserting a new rule that does not violate the constraint.

Consider an example from the domain of chemistry taken from (Ohlsson, 1996, 2011). A learner is trying to construct the structural formula for an organic molecule (its so-called Lewis structure). Suppose we have a carbon atom that already has 8 valence electrons. Then the learner adds a hydrogen atom to the carbon atom and then, discovers that the carbon atom now has more than eight valence electrons. This is an error because atoms strive toward the noble gas configuration, which, in the case of carbon, requires 8 valence electrons, that is, the carbon atom already was in its noble gas configuration. The constraint that follows from this example is that: if the current number of valence electrons for a particular atom is V and the maximum number of valence electrons for atoms belonging to that substance is N, then it had better be the case that V is smaller than or equal to N (or else some error has been committed).

One can see the parallel here with the notion of integrity checking in the field of deductive databases (Colomb, 2004; Olivé, 1991), which is a process that verifies that a given base update (a set of insertions and/or deletion of base facts) satisfies a set of constraints that have the form of deductive rules, also called integrity rules. Hence, in our particular case, concept mapping can be seen as a jigsaw puzzle where a learner seeks to achieve total integrity, i.e., correctness and validity of his/her concept map, w.r.t. a prefabricated map, and receives feedback every time he/she makes an assertion that violates the constraints imposed to a relation linking a referent and a relatum. Figure 2 shows a very general schema of the proposed notion.

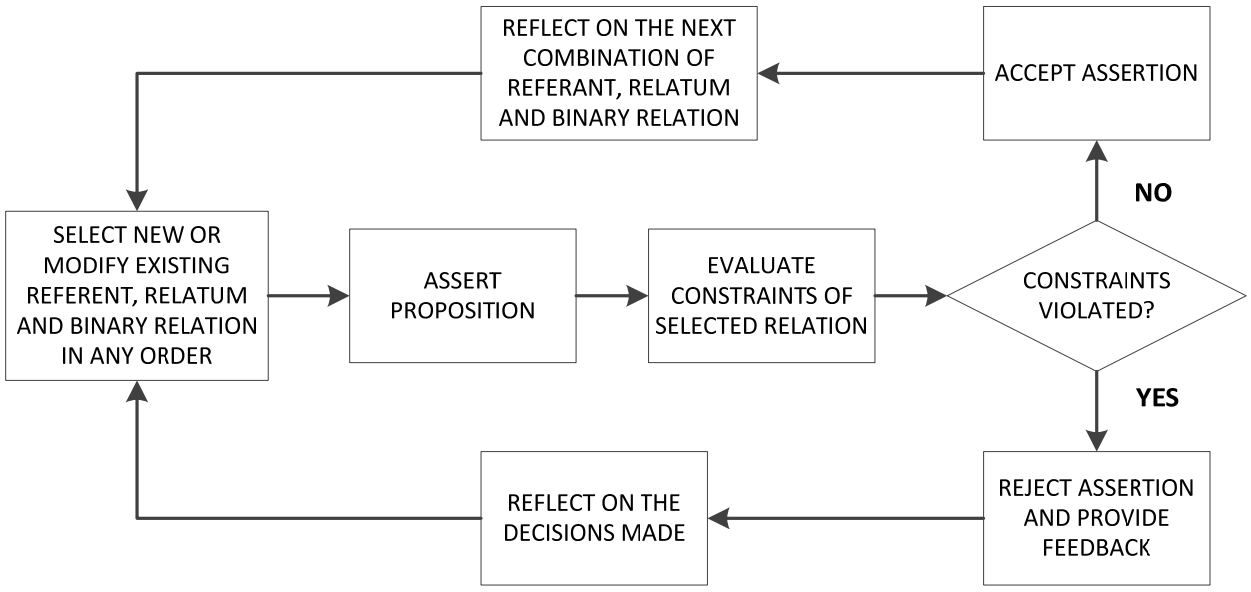

Figure. 2. General schema for the proposed immediate feedback approach.

Additionally, since the style of constraints in this theory, as well as in the Constraint Based Modelling paradigm for Intelligent Tutors (i.e., CBM), is declarative in the sense of logic

programming, as long as the constraints are not violated, the order of operations does not matter (Graesser, Conley & Olney, in press). However, procedural problem-solving can also be modelled. If a domain is governed by principles that pertain to the ordering of problem solving steps, then those principles can be cast as path constraints (Mitrovic & Ohlsson, 2006).

From an implementation perspective, providing feedback using this constraint-based approach offers several advantages over other means of implementing feedback, such as Model Tracing (Graesser, Conley & Olney, in press). Chrysafiadi & Virvou (2013) enumerate the following benefits: 1) It is not necessary to have a runnable expert module, which may be difficult or even impossible to develop for some domains, such as database design or SQL query generation; 2) Extensive studies of typical errors made by students (i.e. bug libraries) are not required. It is even also advantageous over probabilistic methods, such as Bayes networks, which require estimates of prior probabilities (Mitrovic et al., 2001).

In the next section, the notion of properties of binary relations as constraints for concept mapping purposes is developed, focusing on what Hsieh & O'Neil (2002) denominate "knowledge of response feedback", that is, feedback that informs the learner whether the answer is correct.

3. Properties of binary relations

Concept maps, as explained in the introduction, have a pretty simple structure: Nodes, with a tag or name, denoting concepts, and labeled directed arcs representing binary relations. Figure 3 shows an E-R schema of this structure. Its representation using the E-R model, as shown in Figure 3, is also very simple: Two entity types (i.e., Concepts and Relations) and a (ternary) relationship type (i.e., BinaryRelation) relating them. Both entity types are identified by Name attributes.

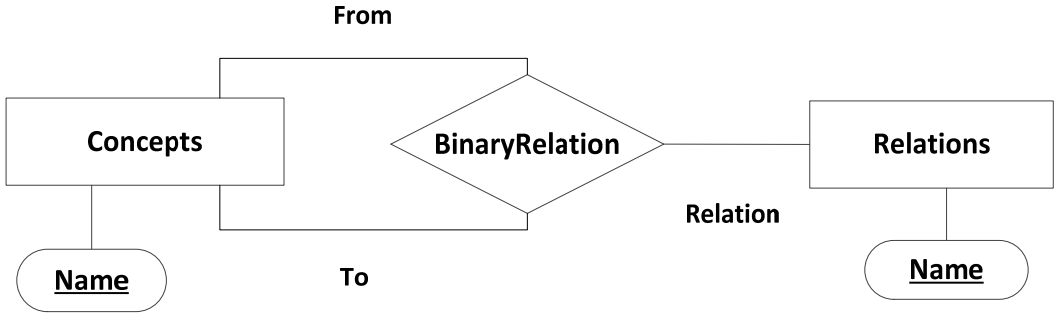

Figure. 3. Standard structure of concept maps.

As it can be seen, the schema does not include any validation or constraint elements beyond the cardinality between entity and relationship types, and primary keys. Therefore, the schema needs to be modified in order to accommodate such elements. To achieve this goal we follow a three step approach: First, we explain the notion, next, we modify the E-R schema, and finally, some Datalog code developed using the Datalog Educational System (Sáenz-Pérez 2011) is presented. All the examples in this paper are bundled in the distribution of DES (Sáenz-Pérez, 2014a), in the folder *examples/ontology*.

3.1. Algebraic properties of binary relations

Many binary relations have a set of properties denominated by some researchers (Álvarez-Montero, Vaquero & Sáenz-Pérez, 2008; Jouis, 2002; Röhrig, 1994) as algebraic properties of relations. These properties, according to Olivé (2007), can be defined as constraints. Consequently, integrity rules can be defined to verify that these properties or constraints are not violated. In particular, these properties are: symmetry, antisymmetry, asymmetry, transitivity, intransitivity, reflexivity and irreflexivity. Using first order logic (FOL) formulas, a relation $R(p_1{:}E, p_2{:}E)$ linking two entities or concepts is:

- Symmetric if: $R(x,y) \rightarrow R(y,x)$.
- Antisymmetric if: $R(x,y) \wedge R(y,x) \rightarrow x = y$.
- Asymmetric if: $R(x,y) \rightarrow \neg R(y,x)$.
- Transitive if: $R(x,y) \wedge R(y,z) \rightarrow R(x,z)$.
- Intransitive if: $R(x,y) \wedge R(y,z) \rightarrow \neg R(x,z)$.
- Reflexive if it can link a concept to itself: $E(x) \rightarrow R(x,x)$.
- Irreflexive if it cannot link an entity to itself: $E(x) \rightarrow \neg R(x,x)$.

Nevertheless, relations with only one property occur very infrequently. The majority of binary relations present a combination of 3 properties. A review of the literature (Badiru & Cheung 2002; Gero, 2000; Goldfarb 2003; Gratton, 2010) shows that there is consensus on six combinations:

1. (Antisymmetric, Reflexive, Transitive). The combination of this triplet typifies any partial order. Examples of this triple are: “is a” and “less or equal than”.
2. (Asymmetric, Irreflexive, Transitive). A binary relation with this triplet is used to state that the referent is greater than or less than the referent, by some objective or subjective scale. Examples of this triplet are: “ancestor of”, “greater than” and “less than”.
3. (Asymmetric, Irreflexive, Intransitive). This triplet asserts that the referent is the agent of the verb or has the property of the noun that describes the relation. Examples of this triplet are: “father of”, “sitting on the legs of” and “starred in”.
4. (Symmetric, Irreflexive, Intransitive). Binary relations expressing kinship or social status, but that do not imply what Lyon (1977) calls converseness, are defined by this triple. For instance: “married to”, “sibling of”, and “first cousin of”.
5. (Symmetric, Reflexive, Intransitive). All compatibility, proximity or tolerance relations are described by this triplet. It declares that both the referent and the relatum are close to each other, share something, or have something in common, by some objective or subjective scale or measure. Examples of these binary relations are: “has/have at least 2 grandparents in common with” and “is within a distance of X kilometers from” and “shares a border with”.
6. (Symmetric, Reflexive, Transitive). This triplet characterizes all equivalence relations such as: “as tall as”, “equal to” and “means the same as”.

There is less agreement about Cause-Effect binary relations. In this paper the position of Taylor (1993) is adopted, where this kind of relations is defined by the following triplet: (Antisymmetric, Irreflexive, Intransitive). Additionally, such relations are assumed to be relative to a particular domain, and are normally associated with laws or regularities which govern that domain and act as constraints upon what may happen.

There are approaches to concept mapping, such as the ones depicted in Pirnay-Dummer, Ifenthaler & Spector (2010), Shute et al. (2009), Strautmane (2012), which focus on binary relations. However, the first two seek to analyze the quality of a concept map, and its evolution

through time, w.r.t. several descriptive measures (e.g., connectedness and ruggedness) as well as determine the degree and strength to which binary causal relations among concepts/nodes hold, by using statistical techniques such as Bayesian networks.

The approach that is closest to ours is the one presented in the concept mapping tool IKAS (Strautmane, 2012). Its goal is to expand the prefabricated map with extra binary relations denominated hidden and inverse relations (Anohina, Vilkelis & Lukasenko, 2009), which are logical consequences of linking two or more concepts with a relation. Nevertheless, instead of using algebraic properties, IKAS relies on pre-identified combinations of relations and their corresponding outcomes expressed as rules. Tables 1, 2 and 3, taken from Strautmane (2012), show some of the combinations and rules used in IKAS.

This approach has severe limitations. First, Tables 1, 2 and 3 are expanded only when a new combination of relations is discovered. This makes the construction of the tables a never ending an error-prone effort. Second, some rules are incomplete. For instance, rules R1 and R2 in Table 2 try to obtain the transitive closure (Backhouse 2011) of relations. However, paths involving more than three concepts cannot be calculated. Fourth, symmetry is confused with the inverse of a relation. For example, the "is sibling of" relation depicted in (Strautmane 2012) is symmetric, and is an example of the following triplet: (Symmetric, Irreflexive, Intransitive). This means that if a learner states "A is sibling of B" then it is also stating, although implicitly, "B is sibling of A". Inverse relations are not symmetric and usually represent relations depicting triplets 1, 2 and 3. Finally, it does not include any restrictions on the logical consequences of relations, which opens the possibility for asserting nonsensical propositions such as "A is sibling of A", "Turtle is a Mammal" and "Homo sapiens ancestor of Homo neanderthalensis".

	Rel.1	Rel. 2	Allowed	Rel. 3	Rule No.
1	Is a	Is a	Yes	Is a	R1
2	Is a	Part of	Yes	Cannot be specified*	-
3	Is a	Attribute	Yes	Cannot be specified*	-
4	Is a	Example	Yes	Is a	R2
5	Is a	Value	Yes	No extra relation**	R3
6	Is a	Kind of	Yes	Is a	R4
7	Part of	Is a	Yes	Part of	R5
8	Part of	Part of	Yes	Part of	R6
9	Part of	Attribute	Yes	Cannot be specified*	-
10	Part of	Example	Yes	Part of	R7

Table 1. Combinations of Relations that Produce Hidden Relations

Rule No.	IF..THEN rule
R1	IF Relation(X, Y, "is a") AND Relation (Y, Z, "is a") THEN Relation (X, Z, "is a")
R2	IF Relation (X, Y, "example") AND Relation (Y, Z, "is a") THEN Relation (X, Z, "is a")
R3	IF Relation (X,Y, "is a") AND Relation (X, Z, "value") THEN NOT Relation (Y, Z, "value") AND NOT Relation (Y, Z, "attribute") AND NOT Relation (Z, Y, "is a") AND NOT Relation (Z, Y, "kind of") AND NOT Relation (Z, Y, "part of") AND NOT Relation (Z, Y, "example")
R4	IF Relation (X, Y, "kind of") AND Relation (Y, Z, "is a") THEN Relation (X, Z, "is a")
R5	IF Relation(X, Y, "is a") AND Relation(Y, Z, "part of") THEN Relation(X, Z, "part of")
R6	IF Relation(X, Y, "part of") AND Relation(Y, Z, "part of") THEN Relation(X, Z, "part of")
R7	IF Relation(X, Y, "example") AND Relation(Y, Z, "part of") THEN Relation(X, Z, "part of")

Table 2. Rules for Defining the Outcome of Linking Two Concepts with a Relation

Relation type	Direction	Inverse relation	Corresponding IF..THEN rule
Is-a	Subclass-> Class	Subclass-is	IF Relation (X, Y, "is a") THEN Relation (Y, X, "subclass-is")
Kind-of	Subkind->Kind	Subkind-is	IF Relation (X, Y, "kind of") THEN Relation (Y, X, "subkind is")
Part-of	Part->Whole	Consists-of	IF Relation (X, Y, "part of") THEN Relation (Y, X, "consists of")
Example	Example-> Class	For-example	IF Relation (X, Y, "example") THEN Relation (Y, X, "for example")
Attribute	Object-> Property	Characterizes	IF Relation (X, Y, "attribute") THEN Relation (Y, X, "characterizes")
Value	Property-> Value	Value-for	IF Relation (X, Y, "value") THEN Relation (Y, X, "value-for")
Before	First event-> Second event	After	IF Relation (X, Y, "before") THEN Relation (Y, X, "after")

Table 3. Inverse Relations and their Corresponding Rules

We demonstrate that by extending the standard schema for concept maps, it is possible to easily design and implement a way to overcome most of the limitations presented before and provide just-in-time feedback. Figure 4 shows the extended E-R schema, which now has one more relationship type (i.e., HasAp) and one more entity type (i.e., AlgebraicProperties). From this E-R schema we can now use Datalog to overcome the problems with Strautmane´s approach.

Nevertheless, in order to avoid cluttering the article with Datalog code, we focus on small practical examples of the usage of Datalog rules in a concept-mapping scenario. The relational and Datalog schema obtained from the E-R schema can be consulted in the annex of this article.

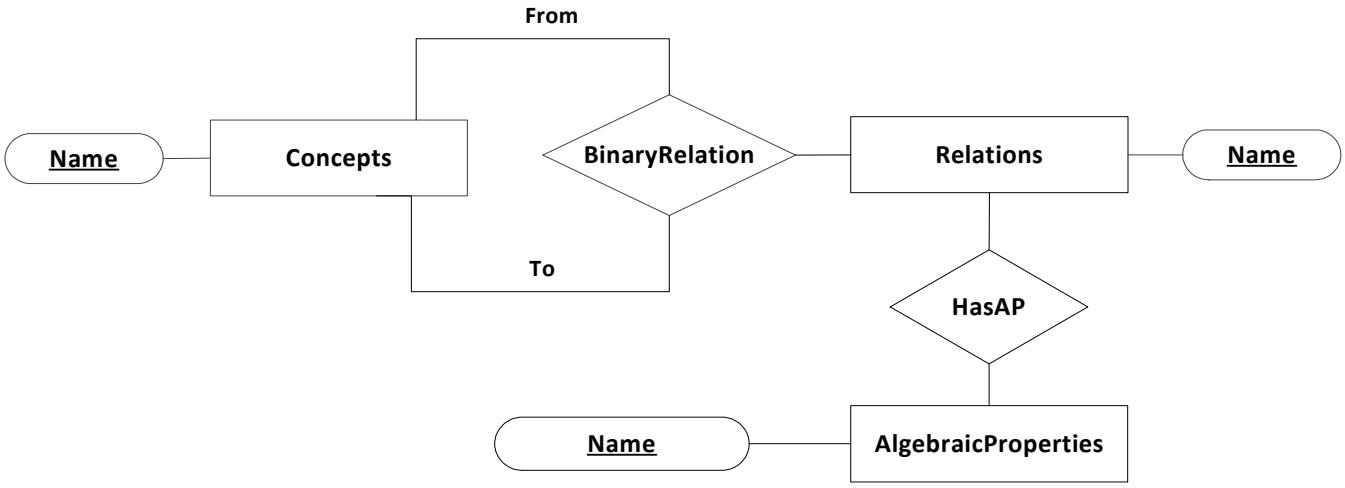

Figure.4. Extended E-R schema with the notion of algebraic properties.

For instance, consider a concept map composed of the propositions "Map means the same as Chart", "Chart means the same as Graph" and "Graph means the same as Diagram". Using the Datalog Educational System (DES) this would be represented with the following facts:

```
binary_relation(same_meaning, 'Map',   'Chart').
binary_relation(same_meaning, 'Chart', 'Graph').
binary_relation(same_meaning, 'Graph', 'Diagram').
```

Attaching the transitivity property to this relation can be done with the fact:

```
has_algebraic_property(same_meaning, transitive).
```

and adding the following rule to the definition of binary relations:

```
binary_relation(BinaryRelation, From, To) :-
  has_algebraic_property(BinaryRelation, transitive),
  binary_relation(BinaryRelation, From, Mid),
  binary_relation(BinaryRelation, Mid, To).
```

With this rule, the meaning of a binary relation with the transitivity property attached is intensionally overloaded with all the facts derived by this property, which can be checked at the system prompt (**DES>**) with the following query:

```
DES> binary_relation(same_meaning,F,T)
{
  binary_relation(same_meaning,'Chart','Diagram'),
  binary_relation(same_meaning,'Chart','Graph'),
  binary_relation(same_meaning,'Graph','Diagram'),
  binary_relation(same_meaning,'Map','Chart'),
  binary_relation(same_meaning,'Map','Diagram'),
  binary_relation(same_meaning,'Map','Graph')
}
Info: 6 tuples computed.
```

Using the same facts, symmetry can be handled by applying the next fact and rule:

```
has_algebraic_property(same_meaning, symmetry).
binary_relation(BinaryRelation, From, To) :-
  has_algebraic_property(BinaryRelation, symmetry),
  binary_relation(BinaryRelation, To, From).
```

and obtain the implicit propositions “Chart means the same as Map”, “Graph means the same as Chart” and “Diagram means the same as Graph”:

```
DES> binary_relation(same_meaning,F,T)
{
  binary_relation(same_meaning,'Chart','Map'),
  binary_relation(same_meaning,'Diagram','Graph'),
  binary_relation(same_meaning,'Graph','Chart'),
  .
  .
  .
}
Info: 16 tuples computed.
```

All of the above rules address some of the logical consequences of linking two concepts with a relation that has certain algebraic properties. Nonetheless, some individual properties impose additional restrictions. In particular, the majority of them prohibit the existence of certain types of cycles in a map while the rest imposes restrictions on the transitive closure of a relation.

For example, we could think of a map with the “Homo neanderthalensis is ancestor of Homo sapiens” proposition, which has the “ancestor of” relation defined by the triplet (Asymmetric, Irreflexive, Transitive):

```
binary_relation(ancestor_of, 'Homo neanderthalensis', 'Homo sapiens').

has_algebraic_property(ancestor_of, asymmetric).
has_algebraic_property(ancestor_of, irreflexive).
has_algebraic_property(ancestor_of, transitive).

asymmetric_violation(BinaryRelation, From, To) :-
  has_algebraic_property(BinaryRelation, asymmetric),
  binary_relation(BinaryRelation, From, To),
  From \= To,
  binary_relation(BinaryRelation, To, From).

:- asymmetric_violation(BinaryRelation, From, To).

irreflexive_violation(BinaryRelation, From, From) :-
  has_algebraic_property(BinaryRelation, irreflexive),
  binary_relation(BinaryRelation, From, From).

:- irreflexive_violation(BinaryRelation, From, To).
```

In this case it should not be possible to state “Homo sapiens is ancestor of Homo neanderthalensis” neither “Homo sapiens is ancestor of Homo sapiens”, because of the asymmetry and irreflexivity properties of the relation. With DES, both asymmetry and irreflexivity can be checked with the following (where the command **/assert *Rule*** allows to interactively adding a rule to the deductive database):

```
DES> /assert binary_relation(ancestor_of, 'Homo sapiens',
                             'Homo neanderthalensis')
Error: Integrity constraint violation.
       ic(BinaryRelation,From,To) :-
         irreflexive_violation(BinaryRelation,From,To).
       Offending values in database: [ic(ancestor_of, Homo sapiens, Homo
sapiens), ic(ancestor_of, Homo neanderthalensis, Homo neanderthalensis)]

DES> /assert binary_relation(ancestor_of, 'Homo sapiens', 'Homo sapiens')
Error: Integrity constraint violation.
       ic(BinaryRelation,From,To) :-
         irreflexive_violation(BinaryRelation,From,To).
       Offending values in database: [ic(ancestor_of,Homo sapiens,Homo
sapiens)]
```

Since our approach to concept map construction requires the explicit assignment of algebraic properties to binary relations, we might be interested in letting the concept map author know that not all the propositions required for a binary relation to be transitive are already stated in the map. For example, let us consider the "Map means the same as Chart" example. If we want to state "Map means the same as Graph", it should not be possible to state so unless the proposition "Chart means the same as Graph" is already stated. This goal can be achieved with:

```
transitive_closure(BinaryRelation, From, To) :-
  binary_relation(BinaryRelation, From, To).
transitive_closure(BinaryRelation, From, To) :-
  transitive_closure(BinaryRelation, From, Mid),
  transitive_closure(BinaryRelation, Mid, To).

has_algebraic_property(same_meaning, explicit_transitive).

explicit_transitive_violation(BinaryRelation, From, To) :-
has_algebraic_property(BinaryRelation, explicit_transitive),
lj(transitive_closure(BinaryRelation, From, To),
     binary_relation(B, F, T),
     (BinaryRelation=B, From=F, To=T)),
  is_null(F).

:- explicit_transitive_violation(BinaryRelation, From, To).
```

The predicate **explicit_transitive_violation** looks for the tuples in the transitive closure that are not in the binary relation. To this end, the metapredicate **lj** computes the left outer join of the two input relations (first two arguments) under a given condition (its third argument). If there is a tuple in the first relation which does not find a counterpart in the second relation, the corresponding values (**B**, **F** and **T**) are set to null. The built-in **is_null** is used to check if a value is a null. Note that this is a special feature in DES which is not found in other deductive systems but common in relational databases. If an offending tuple is asserted under this constraint, an error is issued:

```
DES> /assert binary_relation(same_meaning, 'Chart', 'Graph')
Error: Integrity constraint violation.
       ic(BinaryRelation,From,To) :-
         explicit_transitive_violation(BinaryRelation,From,To).
       Offending values in database: [ic(same_meaning,Map,Graph)]
```

Redundant propositions can also be monitored. For instance, given a transitive propositional chain of the type: “Map means the same as Chart” and “Chart means the same as Graph”, then it is possible to avoid redundant propositions such as “Map means the same as Graph” by using the following:

```
redundant_transitive_violation(BinaryRelation, From, To) :-
  has_algebraic_property(BinaryRelation, non_redundant_transitive),
  group_by(transitive_closure(BinaryRelation, From, To),
           [BinaryRelation, From, To],(C=count,C>1)).
has_algebraic_property(same_meaning, non_redundant_transitive).

:- redundant_transitive_violation(BinaryRelation, From, To).
```

The predicate **redundant_transitive_violation** looks for tuples that occur more than once in the transitive closure of binary relations with the property **non_redundant_transitive** attached. The metapredicate **group_by** allows detecting them in an analogous way it could be done in SQL with the clause **GROUP BY**. With duplicates enabled (with the command **/duplicates on**), a group is built for each triple <binary relation, from, to> and if there is more than one tuple for a group, it is collected in **redundant_transitive_violation**. So, it is not possible to state “Map means the same as Graph” in the example above:

```
DES> /assert binary_relation(same_meaning, 'Map', 'Graph')
Error: Integrity constraint violation.
       ic(BinaryRelation,From,To) :-
         redundant_transitive_violation(BinaryRelation,From,To).
       Offending values in database: [ic(same_meaning,Map,Graph)]
```

Intransitivity also poses restrictions on whatever triplet it is used: (Asymmetric, Irreflexive, Intransitive), (Symmetric, Irreflexive, Intransitive) and (Symmetric, Reflexive, Intransitive). In particular, it prohibits the assertion or deduction of additional propositions. For example, in a map with the propositions “A is father of B” and “B is father of C”:

```
binary_relation(father_of, 'A', 'B').
binary_relation(father_of, 'B', 'C').

has_algebraic_property(father_of, asymmetric).
has_algebraic_property(father_of, irreflexive).
has_algebraic_property(father_of, intransitive).
```

It should not be allowed to state that “A is father of C”. That is, the assertion or deduction of the transitive closure, or parts of it, is not a possibility as it would create nonsensical propositions. The following rule expresses that constraint:

```
intransitive_violation(BinaryRelation, From, To) :-
  has_algebraic_property(BinaryRelation, intransitive),
  binary_relation(BinaryRelation, From, Mid),
  From \= Mid,
  binary_relation(BinaryRelation, Mid, To),
  binary_relation(BinaryRelation, From, To).

:- intransitive_violation(BinaryRelation, From, To).
```

Therefore, if we want to assert that “A is father of C” the assertion is rejected:

```
DES> /assert binary_relation(father_of, 'A', 'C').
Error: Integrity constraint violation.
       ic(BinaryRelation,From,To) :-
         intransitive_violation(BinaryRelation,From,To).
       Offending values in database: [ic(father_of,A,C)]
```

We have seen that algebraic properties are useful to prevent the assertion of redundant propositions, and also to avoid the declaration of a range of false propositions. However, they are not enough to prevent certain nonsensical propositions. For instance, let us consider the "Turtle is a Mammal" example cited before in this subsection. We know that the "is a" relation can be defined by the triplet (Antisymmetric, Reflexive, Transitive) but, is any of these properties useful to avoid asserting such a nonsensical proposition?

The answer is obviously no, because science has come to decide that turtles are not mammals by other means other than basic algebraic properties. To avoid asserting this kind of propositions the following question has to be answered: Why is an animal a mammal? And for that purpose we need facts that help answer that question. Luckily, since concept maps are most used in science teaching, these facts should be easily available. This notion is developed in the next subsection.

3.2. Intrinsic properties of binary relations

Intrinsic properties, as explained before, are facts that answer why the referent can be linked to the relatum by a particular binary relation (i.e., is_a, component_of, etc.). For instance, if one of the challenges for the student is to state that "Turtle is a Reptile" in a concept map, then there must be some facts that justify the assertion of such a proposition. For the "Turtle is a Reptile" proposition, the answer to the question could be: because a turtle is cold-blooded and lays eggs. In our approach, this knowledge acts as a constraint that rejects the assertion of these justification facts, if they have not been already stated by the learner in his/her concept map.

Nevertheless, intrinsic properties, as opposed to algebraic properties, are not general properties of relations. They are properties assigned to a particular combination (Relation, Relatum). For instance, the "cold-blooded and lays eggs" properties are assigned to the (is_a, Reptile) combination. For another proposition such as "Plants eaten by Reptile", then, the intrinsic properties of the (eaten by, Reptile) combination could be: because plants provide minerals, proteins and vitamins. Figure 5 shows the newly extended conceptual schema with the notion of intrinsic properties.

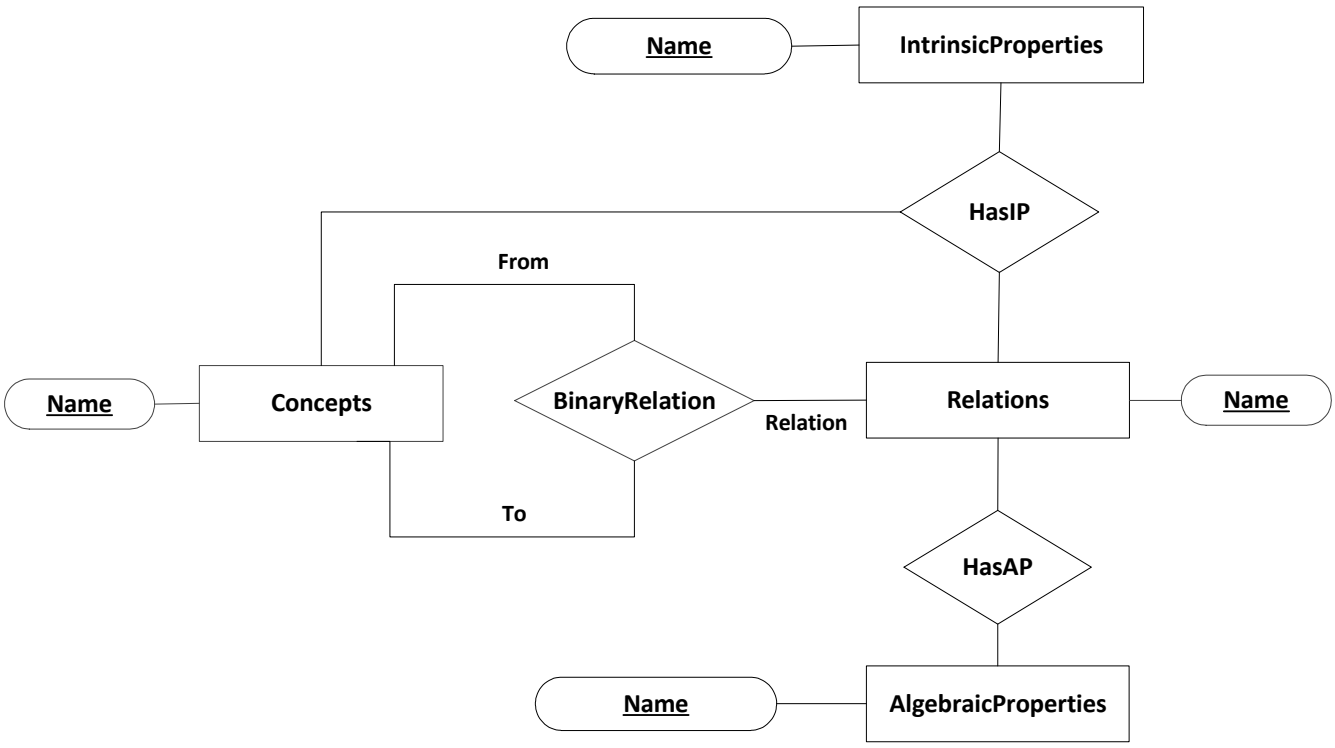

Figure. 5. Extended E-R schema with the notion of intrinsic properties.

Just as with the algebraic properties, the use of intrinsic properties as constraints can be handled by Datalog. For the example involving the "Turtle" and "Reptile" concepts:

```
binary_relation(is_a, 'Turtle', 'Reptile').

has_algebraic_property(is_a, reflexive).
has_algebraic_property(is_a, asymmetric).
has_algebraic_property(is_a, transitive).
```

We can add the intrinsic properties "Turtle lays Eggs" and "Turtle is Cold-blooded" to the (is_a, Reptile) combination:

```
has_intrinsic_property(is_a, 'Reptile', reptile_features).

reptile_features(Reptile) :-
 feature(Reptile,'Lays Eggs'),
 feature(Reptile,'Cold-blooded').
```

and then use the following:

```
reptile_features_violation(Reptile) :-
  binary_relation(is_a,Reptile,'Reptile'),
  not(reptile_features(Reptile)).

:- reptile_features_violation(Reptile).
```

to prevent the assertion of the "Turtle is a Reptile" proposition if the "Turtle lays Eggs" and "Turtle is Cold-blooded" propositions are not stated before:

```
DES> /assert binary_relation(is_a, 'Turtle', 'Reptile')
```

```
Error: Integrity constraint violation.
       ic(Reptile) :-
         reptile_features_violation(Reptile).
       Offending values in database: [ic(Turtle)]
```

The linkage of two concepts by a causal relation is another example of an assertion that needs to be justified and, also, a clear example of path constraints. Following Taylor (1993), this kind of assertions should be preceded by a set of contributory causes (either all positive, all negative, or some of each) that make them true. For instance, consider the "Person moves Table" proposition for the domain of classical physics with its algebraic properties:

```
has_algebraic_property(moves, reflexive).
has_algebraic_property(moves, antisymmetric).
has_algebraic_property(moves, transitive).
```

By adding the following set of intrinsic properties: "Person exerts Force" and "Force applies to Table", to the (moves, Table) combination:

```
has_intrinsic_property(moves, 'Person', move_conditions).

move_conditions(Who,What) :-
  exerts(Who,force),
  applied_to(force,What).
```

and then using the following rules:

```
move_conditions_violation(What) :-
  binary_relation(moves,Who,What),
  not(move_conditions(Who,What)).

:- move_conditions_violation(Object).
```

it is possible to tell the learner that causes must precede effects:

```
DES> /assert binary_relation(moves,'Person','Table')
Error: Integrity constraint violation.
       ic(Object) :-
         move_conditions_violation(Object).
       Offending values in database: [ic(Table)]
```

In relational database parlance (Date, 2008; Thalheim, 2009), we have been representing algebraic and intrinsic properties as hard constraints, that is, as constraints that are checked whenever any change related to the involved data sources for the constraint occurs. However, except for the case of path constraints, in CBM what matters is not the order of the steps leading to the solution, but rather the compliance w.r.t. a set of constraints. This problem is discussed in the next subsection.

3.3. Algebraic and intrinsic properties as soft constrains

Some constraints such as asymmetry and irreflexivity always need to be enforced as hard constraints. Other constraints, if enforced this way, impose an order of operations. For instance, the rule that seeks a missing proposition in a transitive closure demands certain propositions to be stated before others. This is something that unnecessarily limits the learner during concept

mapping activities. In DES, this can be overcome by declaring the constraint as a soft constraint (also known as a deferred constraint (Date, 2008)) by simply removing the hard constraint:

```
:- explicit_transitive_violation(BinaryRelation, From, To).
```

and calling the goal **explicit_transitive_violation(BinaryRelation, From, To)** when the constraint is needed to be checked. By using the "Map means the same as Chart" example, if we wanted to state "Chart means the same as Graph", the system would allow the assertion but it would point out to the missing proposition when requested:

```
DES> /assert same_meaning('Map', 'Chart')
DES> /assert same_meaning('Chart', 'Graph')
DES> explicit_transitive_violation(BinaryRelation, From, To)
{
  explicit_transitive_violation(same_meaning, 'Map', 'Graph')
}
Info: 1 tuple computed.
```

Another alternative is to disable constraint checking with the command **/check off**. Then, the database can be updated and, when it is needed to validate constraints, constraint checking can be enabled again with **/check on**. All the inconsistencies are also displayed. However, this disables checking of *all* of the constraints, not particular ones. Intrinsic properties implemented as constraints can be handled in this same way.

4. Field related examples

In this section, three simplified examples, from the fields of biology, social sciences and astronomy, are presented to highlight both the algebraic and intrinsic properties that can be used. The complete coded examples can be retrieved from Sáenz-Pérez (2014b) by downloading the files **biology.dl** , **countries.dl** and **cosmos.dl** respectively.

4.1. Biology: The human body

Starting from the concepts "body", "head", "trunk", "arm", "leg", "hand", "toe", and "finger" we use the relations "component of" and "part of" to derive a conceptual map for a human body. We say that for a human body ("body" from now on) to be considered as such, required components must be, at least, one head and one trunk; maybe with no arms and no legs. However, these extremities can be part of a body as well. On the one hand, we understand under such semantics that the relation "component of" describes all the required components of a concept. On the other hand, we understand "part of" as the relation that describes which objects can be part of a concept. Both relations are asymmetric, irreflexive, and transitive. The student will be given a specification for these relations along with its properties and will be required to fill the concept map by adding concepts and relations between them. Due to the algebraic properties, the system will avoid entering erroneous inputs as: "arm is part of finger" if "finger is part of hand" and "hand is part of arm" because asymmetry and irreflexivity cancel all loops.

If we assume that the assertion "X is component of Y" implies "X is part of Y", then we overload the meaning of the binary relation "part of" with that of "component of", as follows:

```
binary_relation(part_of, X, Y) :-
```

```
  binary_relation(component_of, X, Y).
```

This means that any fact added to "component of" will be intensionally added to "part of". Instead, as an alternative, we can test if students recognize this implication by imposing the following constraint that will reject assertions trying to add redundant facts about "part of":

```
:- binary_relation(component_of, X, Y),
   binary_relation(part_of, X, Y).
```

This way, a fact as "head is part of body" will be rejected if "head is component of body" is already asserted. But in the previous scenario, the fact can be actually added. So, we can check if the student added the redundancy by asking the meaning of the following Datalog relation:

```
redundant(part_of, X, Y) :-
  binary_relation(component_of, X, Y),
  binary_relation(part_of, X, Y).
```

with the query:

```
DES> redundant(R, X, Y)
{
  redundant(part_of,head,body),
  redundant(part_of,trunk,body)
}
Info: 2 tuples computed.
```

The answer to this query in this instance means that the facts **binary_relation(part_of, head, body)** and **binary_relation(part_of, trunk, body)** are redundant.

Another consideration is to identify the concepts that can be part of several other concepts as an intrinsic property. If we set that only the finger can be part of more than one concept, this can be stated as follows:

```
:- binary_relation(part_of, X, Y),
   X \= finger,
   count(direct_part_of(X, _), C),
   C>1.
```

where the aggregate **count** counts the number of components which are not fingers, and if this number is greater than 1, then there is a constraint violation. This count is over the relation **direct_part_of** which includes the facts about "part of" that relates two parts directly connected, i.e., by excluding those intensionally derived by transitivity (as, e.g., "finger" is part of "body"). This relation is defined as:

```
direct_part_of(X,Y) :-
  binary_relation(part_of,X,Y),
  not indirect_part_of(X,Y).

indirect_part_of(X,Y) :-
  binary_relation(part_of,X,Z),
  binary_relation(part_of,Z,Y).
```

So, trying to add that "toe is part of arm" will be rejected if "toe is part of leg" is already asserted.

4.2. Social Sciences: Countries

This example requires the student to classify countries and their assignment either to the European Union as member states, or to the United Kingdom as member countries, or to the United States of America as union states. Students are given with a number of countries/states, including “Germany”, “Spain”, “Iceland”, “Ireland”, “Scotland”, “California”, “Washington”, and “Mexico”, among others. Besides those unions of countries/states, countries must be classified w.r.t. the continent they belong (America and Europe). Students then elaborate the conceptual map by joining all these concepts with the relations “member of”, “component of”, and “is a”. For instance: “A European country is a component of Europe”, “European Union is a component of Europe”, “Germany is a European country”, “Germany is member of the European Union”, “Iceland is a European country” would be valid facts whereas “Iceland is member of the European Union” is not valid because Iceland is a prospective member state yet. In this setting, we can overload the meaning of “component of” with the meaning of “member of”:

```
binary_relation(component_of, X, Y) :-
  binary_relation(member_of, X, Y).
```

In addition to the algebraic properties attached to the relations above, we can specify as intrinsic properties that a European Union member state must be a country of Europe, so that a fact as “Mexico is a European Union member state” would be rejected:

```
is_eu_state(Country) :-
  binary_relation(is_a, Country, european_country).

is_eu_state_violation(Country) :-
  binary_relation(member_of, Country, european_union),
  not is_eu_state(Country).
```

The constraint ensuring this property is:

```
:- is_eu_state_violation(Country).
```

Similar intrinsic properties can be specified for a UK country and a USA state. Note also that additional conditions might be added to the intrinsic property **is_eu_state** (such as that the country has signed the Schengen Agreement).

Additional intrinsic properties include that an American state is not a country and the other way round:

```
:- is_american_state_violation(State).

:- is_american_country_violation(Country).

is_american_state_violation(State) :-
  binary_relation(is_a, State, american_state),
  binary_relation(is_a, State, country).
```

This way, facts as “Washington is a country” would be rejected provided that “Washington” had been stated as an American state already. Also, facts as “Mexico is an American state” would be rejected if “Mexico” had been stated as a country.

4.3. Science: Cosmos[1]

Cosmology deals with a number of objects that can be represented in a hierarchy (Narlikar, 1996). The universe is composed of galaxies, and each galaxy is composed of other objects, as planetary systems, each one, in turn, composed of stars, planets and satellites. Galaxies, stars, planets and satellites are known as cosmological objects (either simply objects or bodies; both terms are interchangeably used). There are objects that are components of bigger ones or even which are not, as, e.g., a star and the universe itself, respectively. In addition, there are objects that orbit others, as the Moon, which orbits the Earth. Several semantic relations naturally emerge from this description: "is a", "part of", "member of", and "orbits", all of which with attached algebraic properties "irreflexive" (but "is a" which is reflexive), "asymmetric", and "transitive". The basic concepts are the cosmological objects, as "star" and "Sun" (we say that the Sun is a star). An instance of this conceptual map includes facts as (where % represents a remark):

```
% A "star" is an "object_type":
binary_relation(is_a, star,  object_type).
% The "Sun" is an "object_instance":
binary_relation(is_a, 'Earth',  object_instance).
% A "star" is part of a "planetary_system":
binary_relation(part_of, star, planetary_system).
% "Earth" is member of the object type "planet"
binary_relation(member_of, 'Earth', planet).
% "Earth" orbits the "Sun":
binary_relation(orbits, 'Earth', 'Sun').
```

Note that we use the concept "object type" and "object instance" analogously to types and values in common typed languages, i.e., an object type represents many possible object instances ("integer" represents 1, 2, …, and "planet" represents "Earth", "Mars", …) We specify that an object instance belongs to an object type with the relation "member of", and we denote object instances and types with the relation "is a" (e.g. "Earth is an object instance" and "Planet is an object type").

Many intrinsic properties can be identified in this example and here we list some of them. For an object to be considered as a planet, it must directly orbit a star, and no intermediate orbiting object can be found in-between. So, a fact as "Moon is member of planet" will be rejected if the facts "The Moon orbits Earth" and "Earth orbits the Sun" are already asserted. This can be stated with:

```
:- is_planet_violation(Planet).

is_planet(X) :-
  binary_relation(member_of,Y,star),
  binary_relation(orbits,X,Y),
  not(intermediate_object(X,Y)).

intermediate_object(X,Y) :-
  binary_relation(orbits,X,Z),
  binary_relation(orbits,Z,Y).
```

[1] This example is taken from Álvarez-Montero, Sáenz-Pérez & Vaquero (2012).

```
is_planet_violation(Planet) :-
  binary_relation(member_of,Planet,planet),
  not(is_planet(Planet)).
```

Analogously, a satellite must directly orbit a planet. Also, for an object instance to be considered as a planetary system, at least it must include a planet and a star:

```
:- is_planetary_system_violation(PlanetarySystem).

is_planetary_system(X) :-
  binary_relation(is_a, X, object_instance),
  binary_relation(member_of, X, planetary_system),
  direct_part_of(S, X),
  binary_relation(member_of, S, star),
  direct_part_of(P, X),
  binary_relation(member_of, P, planet).

is_planetary_system_violation(PlanetarySystem) :-
  binary_relation(member_of, PlanetarySystem, planetary_system),
  not(is_planetary_system(PlanetarySystem)).
```

Here, the predicate **direct_part_of** has been used to state that a star and a planet must be directly connected to the planetary system object. Otherwise, an object as "Milky Way", containing the "Solar System" could be a planetary system as well, which is incorrect. Analogously, a galaxy must include a star, at least.

Another possible consideration is that an object in any cosmological instance must be either a type or an instance, but not both at the same time:

```
:- is_object_violation(Object).

is_object(Object) :-
  concepts(concept, Object, 0),
  not(is_object_violation(Object)).

is_object_violation(Object) :-
  binary_relation(is_a, Object, object_type),
  binary_relation(is_a, Object, object_instance).
```

This avoids asserting facts as "Earth is an object type" if "Earth is an object instance" is already asserted, and the other way round.

5. Conclusions and future work

This paper has shown a way to implement Ohlsson's theory of learning from performance errors to provide JIT feedback during the authoring of concept maps in high directed scenarios, where the concepts and the relations are to be structured in a jigsaw puzzle manner by the learner. The strategy used to provide JIT feedback relies on the use of a deductive system (i.e., Datalog) and on the extension of the standard conceptual schema of concept maps with algebraic and intrinsic properties which can be used as constraints. This is a novel approach that is not present in other concept-mapping tools and has several implications.

First, concept mapping becomes an activity where the semantics of binary relations, defined by algebraic and intrinsic properties, is at the center of the activity. Hence, in principle, the whole

reasoning process through which a learner builds a concept map changes, as the student comes to realize that there a limits to what she or he can do, while constructing a concept map, and that these limits are imposed by the semantic of relations. However, it remains to be seen if this change stimulates reflection and leads to better results.

Second, as stated in the introduction, providing JIT feedback has several advantages but in concept mapping, its main advantage is that it can prevent learners that start out badly with incorrect propositions, to continue doing so until the map is grossly incorrect. This is a problem encountered in concept mapping tools where feedback is on demand (explicitly requested by the user) or it is delayed until the task is finished (Cimolino, Kay & Miller, 2003). Nevertheless, Datalog is mainly text-based, does not support the use of graphics and does not compromise its clean declarative style in any way (Ceri, Gottlob, and Tanca 1989). Therefore, it is necessary on the one hand, a graphic development environment, in the sense of Visual Query Languages (Groppe, 2011) for the teachers or instructors and, on the other hand, a suitable way to translate Datalog query answers into appropriate and meaningful feedback and scaffolding for the learners (Collins et al., 1975; Gagné, 1985; Lesgold et al. 1992; Palincsar & Brown, 1984; Rogoff & Gardner, 1984; Sleeman & Brown, 1982).

Fourth, the developed E-R schema presented in this paper can be easily extended to address other problems related to the use of concept maps in high-directed settings. For example, concepts, and binary relations can only be given one name, described by a term or word. Consequently, the use of synonyms by the learner cannot be handled during evaluation (Kornilakis et al. 2004), producing what Yin et al. (2005) denominate bipolar scores when learners choose a name for a concept or a relation. To address this problem, Kornilakis et al. (2004) propose the use of WordNet (Miller, 1995), a lexical resource, to address this problem and enhance the prefabricated maps with synonyms. However, WordNet has been criticized inside (Strautmane, 2012) and outside (Nirenburg, McShane and Beale, 2004) the concept maps community. We contend that instead of incorporating a resource, such as WordNet, concept maps should have a structure that allows the introduction of terms according to the school level (Latimer 2001) or knowledge field (Rickard 1984; Martin 2006). Figure 6 shows an E-R schema for concept maps that allows synonymy.

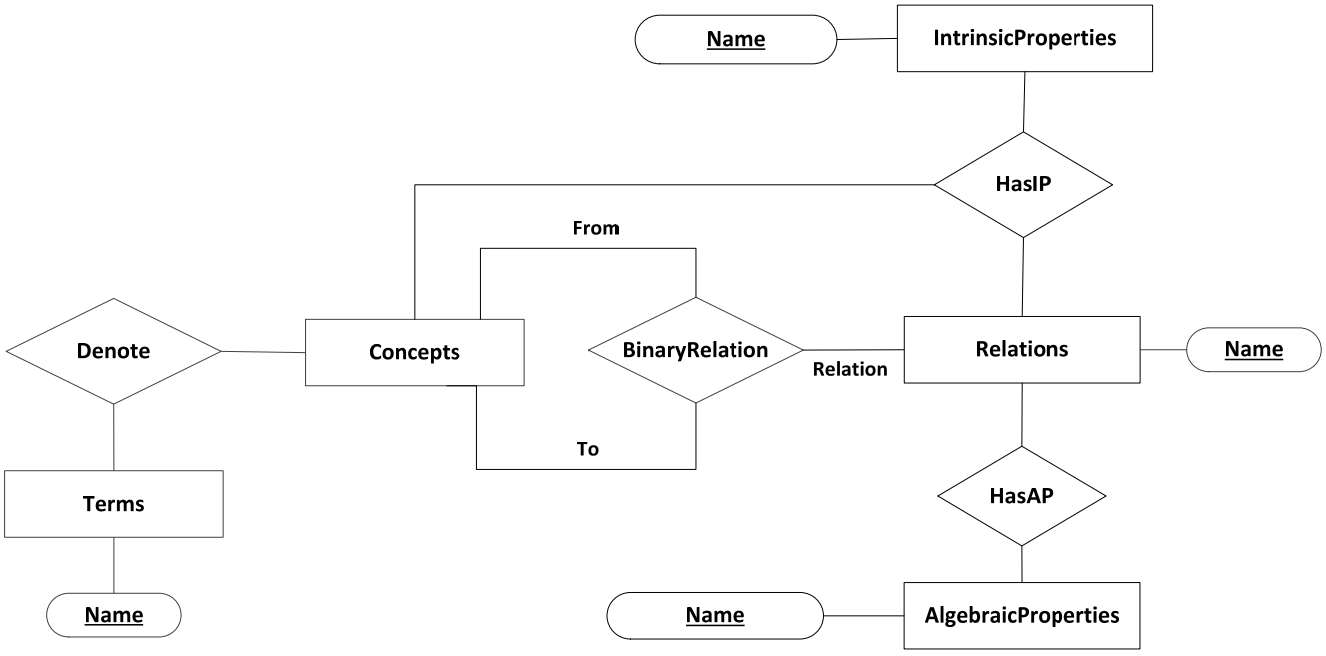

Figure. 6. Extended E-R schema with the notion of synonymy.

6. Acknowledgments

This work has been partially supported by the Spanish MINECO project CAVI-ART (TIN2013-44742-C4-3-R), Madrid regional project N-GREENS Software-CM (S2013/ICE-2731) and UCM grant GR3/14-910502.

7. References

Álvarez-Montero, F.J., Vaquero, A. & Sáenz-Pérez, F. (2008). Conceptual Modeling of Ontology-based Linguistic Resources with a Focus on Semantic Relations. *In Proceedings of the Sixth International Conference on Language Resources and Evaluation* (LREC 2008).

Álvarez-Montero, F.J., Sáenz-Pérez, F. Vaquero, A. & Jacobo-García, H. (2012). Logic Programing and Relational Databases for the Consistent Construction of Knowledge Maps. In *Proceedings of the Global Chinese Conference on Computers in Education* (pp. 83-86).

Álvarez-Montero, F., Sáenz-Pérez, F., & Vaquero, A. Un enfoque declarativo para la construcción de mapas de conocimiento de tipo Fill-In. In *Proceedings of the XIV International Symposium on Computers in Education* (pp. 327-332).

Anohina, A., & Grundspenkis, J. (2009, June). Scoring concept maps: an overview. In *Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing* (p. 78). ACM.

Anohina, A., Vilkelis, M., & Lukasenko, R. (2009). Incremental improvement of the evaluation algorithm in the concept map based knowledge assessment system. *International Journal of Computers, Communication and Control*, *4*(1), 6-16.

Anohina-Naumeca, A., Grundspenkis, J., & Strautmane, M. (2011). The concept map-based assessment system: functional capabilities, evolution, and experimental results. *International Journal of Continuing Engineering Education and Life Long Learning*, *21*(4), 308-327.

Backhouse, R. (2011). *Algorithmic problem solving*. John Wiley & Sons.

Badiru, A. B., & Cheung, J. (2002). *Fuzzy engineering expert systems with neural network applications* (Vol. 11). John Wiley & Sons.

Ceri, S., Gottlob, G., & Tanca, L. (1989). What you always wanted to know about Datalog (and never dared to ask). *Knowledge and Data Engineering, IEEE Transactions on*, *1*(1), 146-166.

Chang, K. E., Sung, Y. T., & Chen, S. F. (2001). Learning through computer□based concept mapping with scaffolding aid. Journal of Computer Assisted Learning, 17(1), 21-33.

Chen, P. P. S. (1976). The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, *1*(1), 9-36.

Chrysafiadi, K., & Virvou, M. (2013). Student modeling approaches: A literature review for the last decade. *Expert Systems with Applications*, *40*(11), 4715-4729.

Cimolino, L., Kay, J., & Miller, A. (2003, July). Incremental student modelling and reflection by verified concept-mapping. In *Supplementary Proceedings of the AIED2003: Learner Modelling for Reflection Workshop* (pp. 219-227).

Coffey, J. W., Carnot, M. J., Feltovich, P. J., Feltovich, J., Hoffman, R. R., Cañas, A. J., & Novak, J. D. (2003). *A summary of literature pertaining to the use of concept mapping techniques and technologies for education and performance support* (Technical Report submitted to the US Navy Chief of Naval Education and Training). Institute for Human and Machine Cognition. Retrieved from:

https://carmenwiki.osu.edu/download/attachments/15599855/IHMC+Literature+Review+on+Concept+Mapping.pdf

Cohen, R. &Nagel, E. (1993). *An Introduction to Logic 2nd Edition*. Hackett Pub Co Inc.

Collins, A., Warnock, E. H., Aeillo, N. &Miller, M. L. (1975). Reasoning from incomplete knowledge. In D. G. Bobrow A. Collins (Eds.), *Representation and understanding* (pp. 453-494). New York: Academic.

Colomb, R. (Ed.). (2004). *Deductive databases and their applications*. CRC Press.

Daley, B. J., & Torre, D. M. (2010). Concept maps in medical education: an analytical literature review. *Medical education*, *44*(5), 440-448.

Date, C. J. (2008). *The relational database dictionary*. Apress.

Gagné, R. M. (1985). *The conditions of learning and theory of instruction* (p. 304). New York: Holt, Rinehart and Winston.

Gero, J. (2000). *Artificial Intelligence in Design*. Kluwer Academic Publishers.

Gilovich, T. (1991). *How we know what isn't so: The fallibility of human reason in everyday life*. New York, NY, US: Free Press.

Goldfarb, W. (2003). *Deductive Logic*. Hacket Publishing Company Inc.

Graesser, A. C., Conley, M. W., & Olney, A. M. (in press). Intelligent tutoring systems. In S. Graham, & K. Harris (Eds.), *APA handbook of educational psychology*. Washington, DC: American Psychological Association.

Gratton, C. (2010). *Infinite Regress Argument*. Springer.

Groppe, S. (2011). Visual Query Languages. In *Data Management and Query Processing in Semantic Web Databases* (pp. 191-201). Springer Berlin Heidelberg.

Horton, P. B., McConney, A. A., Gallo, M., Woods, A. L., Senn, G. J., & Hamelin, D. (1993). An investigation of the effectiveness of concept mapping as an instructional tool. *Science Education*, *77*(1), 95-111.

Hsieh, I., & Gloria, L. (2002). Types of feedback in a computer-based collaborative problem-solving group task. *Computers in Human Behavior*, *18*(6), 699-715.

Hurley, P. J. (2010). *A concise introduction to logic*. Cengage Learning.

Jouis, C. (2002). Logic of relationships. *In The Semantics of Relationships* (pp. 127-140). Springer Netherlands.

Kornilakis, H., Grigoriadou, M., Papanikolaou, K. A., & Gouli, E. (2004, August). Using WordNet to support interactive concept map construction. In *Advanced Learning Technologies, 2004. Proceedings. IEEE International Conference on* (pp. 600-604). IEEE.

Latimer, J. P. (2001). *Simon & Schuster Thesaurus for Children*. Simon & Schuster Children's Publishing.

Lesgold, A., Lajoie, S. P., Bunzo, M., & Eggan, G. (1992). SHERLOCK: A coached practice environment for an electronics trouble-shooting job. In J. H. Larkin & R. W. Chabay (Eds.), *Computer assisted instruction and intelligent tutoring systems: Shared goals and complementary approaches* (pp. 201–238). Hillsdale, NJ: Erlbaum.

Martin, E. (2006). *Computer Jargon Dictionary and Thesaurus*. Specialist Computing Ltd; 2 edition.

Miller, G. A. (1995). WordNet: a lexical database for English. C*ommunications of the ACM*, 38(11), 39-41.

Mitrovic, A., Mayo, M., Suraweera, P., & Martin, B. (2001). Constraint-based tutors: a success story. In *Engineering of Intelligent Systems* (pp. 931-940). Springer Berlin Heidelberg.

Mitrovic, A., & Ohlsson, S. (2006). A Critique of Kodaganallur, Weitz and Rosenthal," A Comparison of Model-Tracing and Constraint-Based Intelligent Tutoring Paradigms". *International Journal of Artificial Intelligence in Education*, *16*(3), 277-289.

Narlikar, J.V. (1996). *Elements of Cosmology*. Universities Press.

Nesbit, J. C., & Adesope, O. (2006). Learning with concept and knowledge maps: A meta-analysis. *Review of educational research*, 76(3), 413-448.

Nirenburg, S., McShane, M., & Beale, S. (2004). The rationale for building resources expressly for NLP. In *Proc. of the 4th International Conference on Language Resources and Evaluation* (Vol. 218).

Novak, J. D., & Gowin, D. B. (1984). *Learning how to learn*. New York: Cambridge University Press.

Nwana, H. S. (1990). Intelligent tutoring systems: an overview. *Artificial Intelligence Review*, *4*(4), 251-277.

Ohlsson, S. (1996). Learning from performance errors. *Psychological review*, *103*(2), 241.

Ohlsson, S. (2011). *Deep learning: How the mind overrides experience*. Cambridge University Press.

Ohlsson, S., & Mitrovic, A. (2007). Fidelity and Efficiency of Knowledge Representations for Intelligent Tutoring Systems. *Technology, Instruction, Cognition & Learning*, *5*(2).

Olivé, A. (1991, September). Integrity Constraints Checking In Deductive Databases. In *VLDB* (pp. 513-523).

Olivé, A. (2007). *Conceptual modeling of information systems*. Springer.

O'Neil, H. F., & Klein, D. C. D. (1997). *Feasibility of machine scoring of concept maps* (CSE Tech. Rep. No, 460). Los Angeles: University of California, Center for Research on Evaluation, Standards, and Student Testing. Retrieved from: http://files.eric.ed.gov/fulltext/ED418100.pdf

Palincsar, A. S., & Brown, A. L. (1988). Teaching and practicing thinking skills to promote comprehension in the context of group problem solving. *Remedial and Special Education*, *9*(1), 53-59.

Pirnay-Dummer, P., Ifenthaler, D., & Spector, J. M. (2010). Highly integrated model assessment technology and tools. *Educational Technology Research and Development*, *58*(1), 3-18.

Rickards, T. (1984). *Cambridge Illustrated Thesaurus of Physics*. Cambridge University Press; 1st edition.

Röhrig, R. (1994). A theory for qualitative spatial reasoning based on order relation. In *Proceedings of the Twelfth National Conference on Artificial Intelligence AAAI-94* (pp. 1418-1423). AAAI-Press.

Rogoff, B. & Gardner, W. (1984). Adult guidance of cognitive development. In: Rogoff, B. and Lave, J. (Eds.), *Everyday cognition: Its development in social context* (pp. 95–116). Cambridge, MA: Harvard University Press.

Ruiz-Primo, M. A., Shavelson, R. J., Li, M., & Schultz, S. E. (2001). On the validity of cognitive interpretations of scores from alternative concept-mapping techniques. *Educational Assessment*, 7(2), 99-141.

Sáenz-Pérez, F. (2011). DES: A Deductive Database System. *Electronic Notes in Theoretical Computer Science*, 271:63–78.

Sáenz-Pérez, F. (2014a). DES: Datalog Educational System (Version 3.8) [Software]. Available from http://des.sourceforge.net.

Sáenz-Pérez, F. (2014b). Coded examples for the Datalog Educational System (DES). Available from http://www.fdi.ucm.es/profesor/fernan/des/examples/ontology.

Shute, V. J. (2008). Focus on formative feedback. *Review of educational research*, *78*(1), 153-189.

Shute, V. J., Jeong, A. C., Spector, J. M., Seel, N. M., & Johnson, T. E. (2009). Model-based methods for assessment, learning, and instruction: Innovative educational technology at Florida State University. In *Educational media and technology yearbook* (pp. 61-79). Springer US.

Sleeman D. & J. S. Brown. (1982). (Eds.). *Intelligent Tutoring Systems*. Orlando, Florida: Academic Press, Inc.

Strautmane, M. (2012). Usage of graph patterns for concept map extension. *Applied Computer Systems*, *13*(1), 37-43.

Taylor, C. (1993). *A Formal Logical Analysis of Causal Relations*. Unpublished doctoral dissertation, University of Sussex.

Thalheim, B. (2009). Extended Entity-Relationship Model. In *Encyclopedia of Database Systems* (pp. 1083-1091). Springer US.

Ullman, J. (1988). *Principles of Database and Knowledge-Base systems Vol. 1*. Computer Science Press.

Yin, Y., Vanides, J., Ruiz-Primo, M. A., Ayala, C. C., & Shavelson, R. J. (2005). Comparison of two concept-mapping techniques: Implications for scoring, interpretation, and use. *Journal of Research in Science Teaching*, 42(2), 166-184.

Annex: Datalog schema for concept maps

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Relational Schema for Figs. 3, 4
% "A Constraint-based Approach for the Construction of
%  Concept Maps in Fill-In-The-Map Scenarios"
% Date:   16-Oct-2013
% Author: Fernando Saenz-Perez
%
% Note: The entity set Relations IS A entity set Concepts
%   (Instance relationship between Relations and Concepts)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Concepts
%   - Type constraints:
:- type(concepts(type:string, id:string, arity:integer)).
%   - Primary key constraint:
:- pk(concepts,[id]).
%   - Domain constraints:
:- concepts(Type,Id,Arity), Arity < 0.
:- concepts(Type,Id,Arity), Id \= concept, Id \= relation.

% Binary relations
%   - Type constraints:
:-           type(binary_relation(relation:string,           fromConcept:string,
toConcept:string)).
%   - Primary key constraint:
```

```
:- pk(binary_relation,[relation, fromConcept, toConcept]).
%   - Foreign key constraints:
:- fk(binary_relation,[relation], concepts, [id]).
:- fk(binary_relation,[fromConcept], concepts, [id]).
:- fk(binary_relation,[toConcept], concepts, [id]).

% Algebraic properties
%   - Type constraints:
:- type(algebraic_properties(algebraicProperty:string)).
%   - Primary key constraint:
:- pk(algebraic_properties,[algebraicProperty]).

% Has algebraic property
%   - Type constraints:
:- type(has_algebraic_property(relation:string, algebraicProperty:string)).
%   - Primary key constraint:
:- pk(has_algebraic_property,[relation, algebraicProperty]).
%   - Foreign key constraints:
:- fk(has_algebraic_property, [relation], concepts, [id]).
:-   fk(has_algebraic_property,   [algebraicProperty],   algebraic_properties,
[algebraicProperty]).

% Intrinsic properties
%   - Type constraints:
:- type(intrinsic_properties(intrinsicProperty:string)).
%   - Primary key constraint:
:- pk(intrinsic_properties,[intrinsicProperty]).

% Has intrinsic property
%   - Type constraints:
:-         type(has_intrinsic_property(relation:string,         concept:string,
intrinsicProperty:string)).
%   - Primary key constraint:
:- pk(has_intrinsic_property,[relation, concept, intrinsicProperty]).
%   - Foreign key constraints:
:- fk(has_intrinsic_property, [relation], concepts, [id]).
:- fk(has_intrinsic_property, [concept], concepts, [id]).
:-   fk(has_intrinsic_property,   [intrinsicProperty],   intrinsic_properties,
[intrinsicProperty]).

% For the problem constraints below (irreflexive_violation, ...):
% - As offending tuples can be duplicated, no primary constraint is imposed
% - Also, by its definition, only valid domain elements are allowed.
%   There is no need for further domain constraints

% Violation of irreflexive algebraic property
:-     type(irreflexive_violation(binaryRelation:string,     fromConcept:string,
toConcept:string)).
% Violation of asymmetric algebraic property
:-     type(asymmetric_violation(binaryRelation:string,     fromConcept:string,
toConcept:string)).
% Violation of explicit transitive algebraic property
:-                    type(explicit_transitive_violation(binaryRelation:string,
fromConcept:string, toConcept:string)).
% Violation of redundant transitive algebraic property
:-                   type(redundant_transitive_violation(binaryRelation:string,
fromConcept:string, toConcept:string)).
```

```
% Violation of intransitive algebraic property
:-     type(intransitive_violation(binaryRelation:string,     fromConcept:string,
toConcept:string)).
% Violation of reptile feature
:- type(reptile_features_violation(concept:string)).
% Violation of move conditions
:- type(move_conditions_violation(concept:string)).
```