Noname manuscript No.
(will be inserted by the editor)

Data model descriptions and translation signatures in a multi-model framework

Paolo Atzeni · Giorgio Gianforme · Paolo Cappellari

Abstract We refer to the problem of translating schemas from a data model to another, in a multi-model framework. Specifically, we consider an approach where translations are specified as Datalog-like programs. In this context we show how it is possible to reason on models and schemas involved as input and output for a translation. The various notions are formalized: (i) concise *descriptions* of models in terms of sets of constructs, with associated propositional formulas; (ii) a notion of *signature* for translation rules (with the property that signatures can be automatically computed out of rules); (iii) the *application* of signatures to models. The main result is that the target model of a translation can be completely characterized given the description of the source model and the signatures of the rules. This result is being exploited in the framework of a tool that implements model generic translations, as the basis for the automatic translations out of a library of elementary ones.

1 Introduction

The translation of schemas from a data model[1] to another has received attention for decades [1,19–21]. An ambitious goal is to consider translations in a model-generic setting [9,11], where the major problem can be formulated as follows:

P. Atzeni supported by MIUR, within the FIRB-MAIS project, and by an IBM Faculty Award.

G. Gianforme supported by Microsoft Research through its European PhD Scholarship Programme. Currently with Almawave.

P. Cappellari supported by MIUR, within the FIRB-MAIS project. Currently with Dublin City University.

Dipartimento di Informatica e Automazione
Università Roma Tre, Italy
[atzeni,cappellari]@dia.uniroma3.it, giorgio.gianforme@gmail.com

[1] In the database field, a *data model* is a set of constructs used to describe the structure of the data of interest. Following common practice in the field, we often abbreviate, and write simply *model*, instead of data model. Clearly, we use the term with a meaning that is different from that used in logic.

"given two data models M_1, M_2 (from a set of models of interest) and a schema S_1 of M_1, translate S_1 into a schema S_2 of M_2 that properly represents S_1."

The goal of this paper is to give formal grounds to the notion of data model and to the management of translations of schemas with reference to a recent approach to this problem, the MIDST proposal [5–7]. Following the MDM approach [9] (which in turn was inspired in this respect by an observation of Hull and King [17]), MIDST assumes that there is a set of *generic* constructs, each with a number of possible variations. Constructs with the "same" meaning in different models are defined in terms of the same generic construct; for example, "entity" in an ER model and "class" in an object model both correspond to the *abstract* construct. Then, a data model is defined by specifying the constructs it includes. The notion of a *supermodel* is introduced: it is a model that includes all constructs (each in its most general version) and so generalizes all the other models (modulo renaming of constructs). The supermodel is not significant in practice as a model, but it plays a major role in our approach as it provides a common framework for the description and comparison of models and for the translation process.

In this framework, the set of models can become very large: each construct has many variations, with many possible combinations. For example, binary relationships (in the family of ER models) can be many-to-many or be limited to one-to-many; they might allow attributes or not; they might involve external identification of entities (and give rise to "weak" entities [24]), or not, and so on. With more constructs (for example nested structures, as appearing in object models or in XML contexts) it is easy to reach hundreds or thousands of different models.[2] With this size for the space of data models, it would be hopeless to have translations between every pair of models. If we refer to the supermodel, then we would need one translation for each model (from the supermodel to it), but with many models this would still be impractical. In order to tackle this problem, MIDST (following MDM [9], and concurrently with other proposals [12,22]) has complex translations that are built as composition of elementary ones. The idea is to have *basic* translations that perform elementary steps that refer to generic constructs, and so can be reused. If the number of models grows because of the possible combinations of constructs, then a relatively small number of basic translations, to be combined, can provide the basis for building many complex translations on the fly, when they are needed, without the need for predefining them. For example, both entities in the ER model and elements in the XSD world can be seen as "abstracts," and so both the translation from the ER and that from XSD to the relational model require the replacement of abstracts with tables. A translation from a model to another would then be obtained as a sequence of basic translations. For example, if we want to translate schemas from an ER model with generalizations and n-ary relationships to an object model with no generalizations, then we would have the following steps (where, for the sake of clarity, we use the specific names, such as "entity," instead of the generic ones, such as "abstract"):

1. replace n-ary relationships with binary ones (adding some entities);
2. eliminate many-to-many relationships (adding entities again);

[2] Obviously, all these models need not be defined at the same time. However, even if few models are used, they are chosen in a very large set. For example, if we have (and we have implemented this) seven aspects in the ER model over which a choice is possible, and the choices are independent from one another, then we have 2^7 variations of the model.

$$\begin{tikzcd}
S_1 \text{ schema for } \mathcal{M}_1 \arrow[rr, "\mathbf{P}"] \arrow[dd, dotted] & {} \arrow[dd, dotted] & S_2 = \mathbf{P}(S_1) \\
 & & S_2 \text{ is a schema for } \mathcal{M}_2 \\
M_1 = \textsc{desc}(\mathcal{M}_1) \arrow[rr, "r_{\mathbf{P}} = \textsc{sig}(\mathbf{P})"] & {} & M_2 = r_{\mathbf{P}}(M_1)
\end{tikzcd}$$

Fig. 1 The main result of the paper

3. replace one-to-many and one-to-one relationships with references;
4. eliminate generalizations (in an object model, new references are added).

With this fine-grained approach, the elementary steps can be reused, but care is needed. For example, steps 1-4 are meaningful in this order, because a step does not introduce constructs removed by one of the previous steps; moreover, step 2 is not needed if the source model has no many-to-many relationships; and the fourth one requires that relationships have been replaced by references, because it does not care about relationships. On the other hand, if step 2 is omitted and there are many-to-many relationships, then the latter are not translated, and so the translated schema would not include concepts corresponding to them. Therefore, a major issue arises: given a set of basic translations, how do we build the actual translations we need? How do we verify that a given sequence of basic translations produces the model we are interested in? And that it does not "forget" any construct?

The contribution of this paper is a formal approach to answering these questions, which exploits the use of Datalog[3] for the implementation of translations, as it is the case in our project, MIDST [5–7]. Let us give an idea of the result. First, we associate with each model a *description* that specifies its constructs. We introduce a similar notion for our Datalog programs, called *signature*, that describes, in a concise way, the behavior of a program, in terms of the involved constructs: given a program, we can derive its signature, and we have defined the application of program signatures to model descriptions. Then, the result is that signatures properly describe the behavior of programs on models, in the sense that the application of signatures provides a "derivation" of models that is sound and complete (in a sense to be clarified shortly) with respect to the schemas generated by programs. Let us describe the main results with the help of Figure 1. Let S_1 be a schema for a model $\mathcal{M}_1$, let $\mathbf{P}$ be a Datalog program implementing a translation, and let S_2 be the schema obtained by applying $\mathbf{P}$ to S_1. Then, let M_1 be the description of $\mathcal{M}_1$ and $r_{\mathbf{P}}$ the signature of $\mathbf{P}$, which corresponds to a function (the *application* $r_{\mathbf{P}}()$ of $r_{\mathbf{P}}$) from descriptions of models to descriptions of models. Soundness and completeness (of signatures with respect to Datalog programs) can be claimed as follows, respectively (the claims will be formalized and proved later in the paper and synthesized in Theorem 3):

1. the application of $\mathbf{P}$ to a schema S_1 of $\mathcal{M}_1$ produces a schema $S_2 = \mathbf{P}(S_1)$ that is legal for the model $\mathcal{M}_2$ whose description $M_2 = r_{\mathbf{P}}(M_1)$ is obtained by applying the signature $r_{\mathbf{P}}$ of program $\mathbf{P}$ to the description M_1 of $\mathcal{M}_1$;
2. among the possible schemas of $\mathcal{M}_1$, there exists a schema S^* such that the application of $\mathbf{P}$ to S^* produces a schema $\mathbf{P}(S^*)$ that (beside belonging to

[3] More precisely, as we will clarify, we use a variant of Datalog. However, for the sake of conciseness, we will always say "Datalog" program or rule, rather than "Datalog-like."

$r_{\mathbf{P}}(M_1)$, as stated by the previous claim) does not belong to any model that is "strictly more restricted" than $r_{\mathbf{P}}(M_1)$.

The two claims together say that the model $\mathcal{M}_2$ whose description M_2 we derive by means of the application of signatures is exactly the model that allows the set of schemas that can be obtained by means of the Datalog programs. Claim (1) says that the derived model is liberal enough (*soundness*) and claim (2) says that it is restricted enough (*completeness*). This can be seen as a "syntactic" aspect of the correctness of rules.

The contents of this paper complement those we have recently published on our MIDST tool [5–7]. In the latter papers, we have shown the overall approach to the problem and how individual translations can be defined and composed. Here we show how it is possible to describe the properties of translations and to reason on them. Various applications are possible for the results we have here. First of all, they can be used to perform simple but useful tasks on the behavior of Datalog programs, without the need to run them on specific schemas. Some of these are (we list them informally here, and we will discuss them in a more technical way later in the paper):

– test for applicability: given a Datalog rule and a model (schema), is the rule applicable to the model (schema)?
– test for membership of a schema to a model: given a schema and a model, does the schema belong to the model?
– inclusion test between schemas and/or models: given two schemas (models), is the first included in the latter or is the second included in the first or neither?
– computation of the result of the application of a Datalog program to a schema (model): we can apply the signature of the program to the description of the schema (model) rather than applying the Datalog program itself.

Moreover, they can be the basis for the automatic generation of complex translations given a library of elementary steps.

The rest of the paper is organized as follows. The main results are in Section 5; they are preceded by an overview of MIDST and an informal presentation of the results (Section 2) and by a formalization of the approach, both for descriptions of models (Section 3) and signatures of rules (Section 4). Then we discuss in Section 6 the interesting consequences of the results in the context of a tool. In Section 7 we extend our approach to a more expressive but concise version of the supermodel. We also discuss some related work (Section 8) and draw our conclusions (Section 9).

2 Context and goals

We set the context for our approach, by discussing the features of the MIDST [5, 7] project that are of interest, together with a running example that will be used to illustrate the technical development.

We assume the availability of a set of constructs, called the *universe*. Each *construct* in the universe has a set of references (which relate its occurrences to occurrences of other constructs; references are acyclic) and Boolean properties. Occurrences of constructs also have names and possibly types (for example, in the ER model each entity, relationship, and attribute has a name, and each attribute has a type), but we need not refer to them here. Let us explain the basic idea by means of the universe we will use in our running example. This universe has the following constructs (just a subset of the universe we are using in the MIDST tool, which allows to handle many versions of the XSD, ER, OO and relational models):

Construct	***References***	***Boolean properties***
ENTITY		
ATTRIBUTEOFENT.	Entity	isKey, isNullable
RELATIONSHIP	Entity1, Entity2	isOpt1, isFunct1, isIdent, isOpt2, isFunct2
ATTRIBUTEOFREL.	Relationship	isNullable
TABLE		
COLUMN	Table	isKey, isNullable

Let us comment on the various aspects. References in the example are rather intuitive: each attribute and column has a reference to the construct it is associated with; each relationship (binary here for the sake of simplicity) has references to the two involved entities. Properties require some explanation: for each attribute of an entity, we can specify, with isKey, whether it belongs to the primary key and, with isNullable, whether it allows null values; each relationship has five properties: isOpt1 (isOpt2) tells us whether the participation of the first (second) entity is optional or compulsory, that is, whether its minimum cardinality is 0 or 1, isFunct1 (isFunct2) tells whether its maximum cardinality is 1 or is unbounded ('N', as we usually write), isIdent tells us whether the first entity has an external identifier to which this relationship contributes (a "weak" entity [24]). In this way, we are able to describe the major aspects of a relationship with respect to its cardinality and external identification (as suggested in major texts on conceptual design, such as Batini et al. [10]).

It is important to note that we have shown this set of constructs to give an idea of how schemas can be described in a model independent way, that is, in a framework that allows for the definition of various models and relates constructs to the framework, rather than to the specific model. Details on how a dictionary that describes data models and schemas in a coherent way can be found in Atzeni et al. [4]. Here the importance is not really in the set of models, but just on the fact that in this way we can describe models in a flexible way and we can reason on them. Indeed, the technical development of this paper is independent of the actual sets of constructs and of their properties—it is generic and extendible.

In this framework, given a set of constructs, references are required to build schemas for meaningful models: in fact, it is often the case that a construct is "subordinate" to another, for example we have columns only if we have tables and relationships only if we have entities. In order to simplify the technical development, we assume here, throughout Section 6,[4] that each construct has a fixed set of required references (in the example, those in the table above). We will use the term *referential constraint* for the requirement that we have in our schemas that each occurrence of a construct does have references to occurrences of other constructs: for example, each occurrence of the construct relationship is required to have references to two occurrences of the construct entity .

[4] In Section 7 we will remove this assumption, in order to allow for more flexibility, and discuss how the approach maintains its validity.

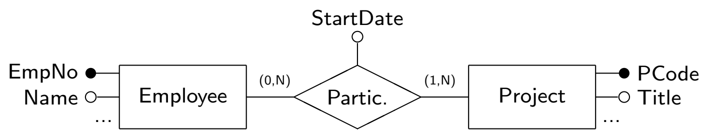

Fig. 2 A schema in the ER model $\mathcal{M}_{\text{ER}}$

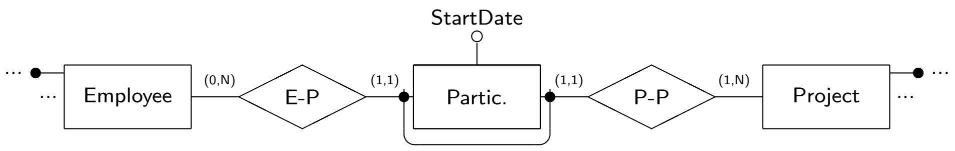

Fig. 3 A schema in the ER model with no many-to-many relationships $\mathcal{M}_{\text{ERNoM2N}}$

Employee		
<u>EmpNo</u>	Name	...
...	...	...
...	...	...

Participation		
<u>EmpNo</u>	<u>PCode</u>	StartDate
...	...	...
...	...	...

Project		
<u>PCode</u>	Title	...
...	...	...
...	...	...

Fig. 4 A relational schema

On the other hand, properties could be restricted in some way. For example, we can think of models where all cardinalities for relationships are allowed and models where many-to-many relationships are not allowed.

Therefore, we can think that models are defined by means of the constructs, each with a condition on its properties; this idea will be formalized in Section 3. Let us list a set of models to be used in the discussion:

- $\mathcal{M}_{\text{REL}}$: a relational model, with tables and columns (and no restrictions);
- $\mathcal{M}_{\text{RELNoN}}$: a relational model with no null values: all columns must have a value *false* for property isNullable;
- $\mathcal{M}_{\text{ER}}$: an ER model with all the available features;
- $\mathcal{M}_{\text{ERSIMPLE}}$: an ER model with no null values on attributes (all attributes have a value *false* for isNullable) and no attributes on relationships;
- $\mathcal{M}_{\text{ERNoM2N}}$: an ER model with no many-to-many relationships (all relationships have a value *true* for isFunct1 or isFunct2).

As we said in the introduction, translations in this approach are specified by means of sequences of elementary steps. A translation from the ER model $\mathcal{M}_{\text{ER}}$ to the relational model $\mathcal{M}_{\text{REL}}$ could be composed of the basic translations:

- $\mathbf{P}_1$: eliminate many-to-many relationships;
- $\mathbf{P}_2$: translate (i) entities into tables, (ii) attributes and one-to-many (and one-to-one) relationships into columns.

For example, given the ER schema sketched in Figure 2, program $\mathbf{P}_1$ translates it into the schema in Figure 3 and then program $\mathbf{P}_2$ translates the latter into the relational schema sketched in Figure 4, by using well known techniques, as follows. First program $\mathbf{P}_1$ eliminates the many-to-many relationship Participation by replacing it with an entity, connected to Employee and Project by one-to-many relationships. The new entity is a weak one, identified by means of the participation to the two relationships. Then program $\mathbf{P}_2$ translates from the ER world to the relational one, by introducing a table for each entity and an attribute for each one-to-many relationship. In this case, the two attributes EmpNo and PCode form the key for Participation (Figure 4) because of the weakness of the corresponding entity.

In MIDST translations are specified in a Datalog variant with OID invention, with ideas from ILOG [18]. The predicates correspond to constructs and their arguments may be (beside OIDs, names and possibly types) references and properties. The variant we adopt with respect to standard Datalog consists in the use of Skolem functors for OID invention, due to the fact that we need to generate new (occurrences of) constructs at each step of the translation, each identified by a newly created OID.

A Datalog program for $\mathbf{P}_1$ includes eight rules (shown in Figure 5). Let us briefly comment on the rules (while we refer to our previous work for more details [7]). We use a non-positional notation for them, so we indicate the names of the fields, and omit those that are not needed (rather than using anonymous variables). Rules generate constructs for a target schema (tgt) from those in a source schema (src), and we may assume that variables tgt and src are bound to constants when the rule is executed. Each predicate has an OID argument, used for unique identification of its occurrences. For each schema we have different identifiers, and so, when a construct is produced by a rule, it has to have a "new" identifier, which is generated by means of a Skolem functor (denoted by the # sign in the rules). Skolem functors are also used for references, in order to correlate newly created occurrences of constructs. In general, there are several functors (with disjoint ranges) for each construct.

Let us comment on the specific rules in the figure. The first four rules copy, from the source schema to the target one, entities and (if there are any) one-to-one and one-to-many relationships, with their respective attributes. Indeed, in most of our programs we have many "copy" rules, because each translation is usually concerned with a very specific task, such as eliminating a construct or some of its variants. In our example above, this happens for entities Employee (with EmplNo, Name, ...) and Project (with PCode, Title, ...). In the example, the only relationship is many-to-many, so rules $R_{1,3}$ and $R_{1,4}$ are not used.

Then, the rules that actually perform the translation are the last four. Let us comment on them, especially $R_{1,5}$, $R_{1,6}$ and $R_{1,7}$, in order to discuss a few important issues:

- These rules are applied only if both isFunct1 and isFunct2 are *false*: so, if the source model has no many-to-many relationships (for example, $\mathcal{M}_{\mathrm{ERNoM2N}}$), they have no effect, that is, they generate no elements in the target schema.
- $R_{1,6}$ and $R_{1,7}$ always generate relationships with isFunct1 = *true* (because of the `true` value for isFunct1 in the head); so, they do not generate many-to-many relationships; constants in the head tell us some restrictions on the construct that is generated; in the example, these rules generate, from the schema in Figure 2, the two relationships E-P and P-P in Figure 3, which are both one-to-many.
- The repeated variable isOpt in $R_{1,6}$ transfers the value of isOpt1 in the source to the value of isOpt2 in the target; similarly, the variable with the same name in $R_{1,7}$ transfers the value of isOpt2 in the source to the value of isOpt2 in the

Rule $R_{1,1}$: *copy entities*
ENTITY(OID: #entity_0(eOid), sOID: tgt, Name: n)
←ENTITY(OID: eOid, sOID: src, Name: n)

Rule $R_{1,2}$: *copy attributes of entities*
ATTRIBUTEOFENT(OID: #attribute_0(aOid), sOID: tgt, Name: n, IsKey: isK,
IsNullable: isN, EntityOID: #entity_0(eOid))
←ATTRIBUTEOFENT(OID: aOid, sOID: src, Name: n, IsKey: isK,
IsNullable: isN, EntityOID: eOid)

Rule $R_{1,3}$: *copy one-to-one and one-to-many relationships*
RELATIONSHIP(OID: #relationship_0(rOid), sOID: tgt, Name: n,
Entity1: #entity_0(eOid1), isOpt1: isO1, isFunct1: `true`, isIdent: isId,
Entity2: #entity_0(eOid2), isOpt2: isO2, isFunct2: isF2)
←RELATIONSHIP(OID: rOid, sOID: src, Name: n, Entity1: eOid1, isOpt1: isO1,
isFunct1: `true`, isIdent: isId Entity2: eOid2, isOpt2: isO2, isFunct2: isF2)

Rule $R_{1,4}$: *copy attributes of one-to-one and one-to-many relationships*
ATTRIBUTEOFREL(OID:#attributeOfRel_0(arOid), sOID: tgt, Name: n,
IsNullable: isN, RelationshipOID: #relationship_0(rOid))
←ATTRIBUTEOFREL(OID: arOid, sOID: src, Name: n,
IsNullable: isN, RelationshipOID: rOid),
RELATIONSHIP(OID: rOid, sOID: src, isFunct1: `true`)

Rule $R_{1,5}$: *generate an entity for each many-to-many relationship*
ENTITY(OID: #entity_1(rOid), sOID: tgt, Name: n)
←RELATIONSHIP(OID: rOid, sOID: src, Name: n, isFunct1: `false`, isFunct2: `false`)

Rule $R_{1,6}$: *for each entity generated by* $R_{1,5}$*, generate a relationship between it and the copy of the first entity involved in the many-to-many relationship*
RELATIONSHIP(OID: #relationship_1(eOid,rOid), sOID: tgt, Name: eN+rN,
Entity1: #entity_1(rOid), isOpt1: `false`, isFunct1: `true`, isIdent: `true`,
Entity2: #entity_0(eOid), isOpt2: isOpt, isFunct2: `false`)
←RELATIONSHIP(OID: rOid, sOID: src, Name: rN, Entity1: eOid, isOpt1: isOpt,
isFunct1: `false`, isFunct2: `false`),
ENTITY(OID: eOid, sOID: src, Name: eN)

Rule $R_{1,7}$: *for each entity generated by* $R_{1,5}$*, generate a relationship between it and the copy of the second entity involved in the many-to-many relationship*
RELATIONSHIP(OID: #relationship_1(eOid,rOid), sOID: tgt, Name: eN+rN,
Entity1: #entity_1(rOid), isOpt1: `false`, isFunct1: `true`, isIdent: `true`,
Entity2: #entity_0(eOid), isOpt2: isOpt, isFunct2: `false`)
←RELATIONSHIP(OID: rOid, sOID: src, Name: rN, Entity2: eOid,
isOpt2: isOpt, isFunct1: `false`, isFunct2: `false`),
ENTITY(OID: eOid, sOID: src, Name: eN)

Rule $R_{1,8}$: *for each attribute of each many-to-many relationship, generate an attribute for the entity generated by* $R_{1,5}$
ATTRIBUTEOFENT(OID: #attributeOfEnt_1(arOid), sOID: tgt, Name: n,
IsKey: `false`, IsNullable: isN, EntityOID: #entity_1(rOid))
←ATTRIBUTEOFREL(OID: arOid, sOID: src, Name: n, IsNullable: isN,
Relationship: rOid)
RELATIONSHIP(OID: rOid, sOID: src, isFunct1: `false`, isFunct2: `false`)

Fig. 5 The rules that form program $\mathbf{P}_1$

target. Let us recall that properties isOpt1 and isOpt2 describe the minimum cardinality of a relationship. Indeed, in the example, for the relationship Participation in the source, we would have isOpt1 $=$ *true* (because of the minimum cardinality set to 0 for Employee) and isOpt2 $=$ *false* (minimum cardinality set to 1 for Project). Then, $R_{1,6}$ generates isOpt2 $=$ *true* for E-P (minimum cardinality 0 for Employee, as shown in Figure 3) and $R_{1,7}$ generates isOpt2 $=$ *false* for P-P (minimum cardinality 1 for Project).

It turns out that, if we apply program $\mathbf{P}_1$ to a schema in model $\mathcal{M}_{\text{ERNoM2N}}$ (that is, ER with no many-to-many relationships), we obtain the same schema,[5] because only copy rules are applicable, and so, if we want to translate a schema from $\mathcal{M}_{\text{ERNoM2N}}$ to $\mathcal{M}_{\text{Rel}}$, then it suffices to apply program $\mathbf{P}_2$. In this paper we show that it is possible to formalize these ideas, by means of the notions of descriptions of models and signatures of programs mentioned in the introduction. In fact, if, for each of the models listed above, M_α is the description of model $\mathcal{M}_\alpha$ and $r_{\mathbf{P}_1}$ and $r_{\mathbf{P}_2}$ are the signatures of programs $\mathbf{P}_1$ and $\mathbf{P}_2$, respectively, then:

(i) $r_{\mathbf{P}_1}(M_{\text{ERNoM2N}}) = M_{\text{ERNoM2N}}$;
(ii) $r_{\mathbf{P}_1}(M_{\text{ER}}) = M_{\text{ERNoM2N}}$;
(iii) $r_{\mathbf{P}_2}(r_{\mathbf{P}_1}(M_{\text{ERsimple}})) = M_{\text{RelNoN}}$.

The results in Section 5 guarantee that the "syntactic" properties (i)-(iii) correspond indeed to the "semantics": for example, from (iii) we will be able to know that the application of the sequence of programs $\mathbf{P}_1$ and $\mathbf{P}_2$ to a schema of $\mathcal{M}_{\text{ERsimple}}$ produces a schema of $\mathcal{M}_{\text{RelNoN}}$. Moreover, the technical development will allow us to say that if we apply program $\mathbf{P}_2$ to a schema of $\mathcal{M}_{\text{ERsimple}}$ or $\mathcal{M}_{\text{ER}}$ that contains many-to-many relationships, which are allowed for them, then some constructs (in fact, many-to-many relationships) are ignored[6] and so get lost.

Before concluding the section, let us mention that we have a number of restrictions on our rules, for which we give the main ideas, as needed for stating the hypotheses we use for proving the main results of this paper. It is important to observe that all these assumptions can be checked automatically (actually, we have built a rule editor that guarantees that they are satisfied) and they are all satisfied in our tool, so they do not limit the significance of the approach. First of all, we have the standard "safety" requirements [24]: every variable that appears in a rule has to appear in the body in a (positive) atom that is not a comparison. Second, we assume that Boolean variables for properties cannot be repeated in the head nor in the body (the only allowed repetition is: once in the body and once in the head). Third, repeated variables for oids are allowed only in fields that are subject to referential constraints and in comparison atoms, which may contain only inequality comparisons (equalities are handled by means of repeated variables).

It is also important to observe that we refer to rules without negation. Indeed, negations arise only in a few cases in the translation process, with a restriction

[5] More precisely, we obtain an isomorphic copy of the source schema.

[6] Strictly speaking, such a schema need not contain many-to-many relationships, we just say that it may contain, so we are not guaranteed that everything is kept.

over positive conditions which can be reduced to the positive case (possibly with some growth in the number of rules[7]).

Moreover, our programs are coherent with respect to referential constraints: if there is a rule that produces a construct C that refers to a construct C', then there is another rule that generates a suitable C' that guarantees the satisfaction of the constraint. This property can be checked by analyzing the rules. In the example, rule $R_{1,6}$ is acceptable because rules $R_{1,1}$ and $R_{1,5}$ produce entities and so guarantee that references of relationships generated by $R_{1,6}$ are not dangling.

A final comment on recursion is useful. Most of our rules, such as $R_{1,6}$ above, are recursive according to the standard definition. However, recursion is only "apparent": a literal occurs in both the head and the body, but the construct generated by an application of the rule belongs to the target schema, so the rule cannot be applied to it again, as the body refers to the source schema. A really recursive application happens only for rules that have atoms that refer to the target schema also in their body. In the following, we will use the term *strongly recursive* for these rules. Indeed, in our experiments with MIDST we have needed few strongly recursive rules, the most notable of which are for resolving unbounded chains of external identification (for example a department number is unique within a company, a sub-department number is unique within a department, and so on) and for unnesting complex structures.

3 Descriptions of constructs and models

In this section we formalize the description of models. We define models in terms of their descriptions, blurring the distinction between a model and its description, as descriptions are sufficient for the purpose of this paper. Indeed, there could be different ways to define models within a metamodel framework, but once one is fixed, then it can be used as a formal description.

As we illustrated in the previous section, each construct has a number of properties and references. References are tightly bound to the construct, with no variations. Properties, instead, can be subject to restrictions. Therefore we can give a synthetic description of a model by listing the constructs it involves, each with a propositional formula over its Boolean properties, and ignoring references.

In detail, as we anticipated in Section 2, we fix a **universe** of constructs, each with a set of associated properties:

$$\mathcal{U} = \{C_1(P_1), C_2(P_2), \ldots, C_u(P_u)\}$$

In the examples we will refer to the constructs in the previous section, in abbreviated form as follows (the *abbreviation* denotes the short name of each *construct* with the associated *properties*, as it will be used in the sequel for the sake of compactness):

[7] The growth could be at most quadratic in the number of explicit conditions in the negated literals. We say that it is small because in all practical cases we have considered we have at most one or two rules with negated literals and each with one or two conditions.

Construct	***Boolean properties***	***Abbreviation***
Entity		$\text{E}()$
AttributeOfEnt	isKey,isNullable	$\text{A}(\text{K},\text{N})$
Relationship	isOpt1,isFunct1,isIdent,isOpt2,isFunct2	$\text{R}(\text{O}_1,\text{F}_1,\text{I},\text{O}_2,\text{F}_2)$
AttributeOfRel	isNullable	$\text{AR}(\text{N})$
Table		$\text{T}()$
Column	isKey,isNullable	$\text{C}(\text{K},\text{N})$

Then, **(the description of) a model** is a mapping that associates a proposition (over the associated properties) with each construct in the universe:

$$M = \{C_1(f_1), C_2(f_2), \ldots, C_u(f_u)\}$$

We will use the term **construct description** to refer to $C(f)$, where C is the name of a construct and f a proposition (over the associated properties).

In the definition above, we have that all constructs are mentioned in every model, possibly with a *false* proposition, which would mean that the construct does not belong to the model. In practice, we describe a model by listing only the constructs that really belong to it—those that have a proposition that is *satisfiable*, that is, not identically false; in this way, the models discussed in Section 2 would be as follows (with the propositions that specify the restrictions we have informally described there):

– $M_{\text{Rel}} = \{\text{T}(\textit{true}),\ \text{C}(\textit{true})\}$
– $M_{\text{RelNoN}} = \{\text{T}(\textit{true}),\ \text{C}(\neg\text{N})\}$
– $M_{\text{ER}} = \{\text{E}(\textit{true}),\ \text{A}(\textit{true}),\ \text{R}(\text{F}_1 \vee \neg\text{F}_2),\ \text{AR}(\textit{true})\}$[8]
– $M_{\text{ERsimple}} = \{\text{E}(\textit{true}),\ \text{A}(\neg\text{N}),\ \text{R}(\text{F}_1 \vee \neg\text{F}_2)\}$
– $M_{\text{ERnoM2N}} = \{\text{E}(\textit{true}),\ \text{A}(\textit{true}),\ \text{R}(\text{F}_1),\ \text{AR}(\textit{true})\}$

We can define a partial order on models, as follows:

– $M_1 \sqsubseteq M_2$ (read M_1 is **subsumed** by M_2) if for every $C \in \mathcal{U}$ it is the case that $f_1 \wedge f_2$ is equivalent to f_1 (that is, f_1 implies f_2), where $C(f_1) \in M_1$ and $C(f_2) \in M_2$

It can be shown that $\sqsubseteq$ is a partial order (modulo equivalence of propositions), as it is reflexive, antisymmetric[9] and transitive.

If models are described only in terms of the constructs that have satisfiable properties, then the partial order can be rewritten as:

– $M_1 \sqsubseteq M_2$ if for every $C(f_1) \in M_1$ there is $C(f_2) \in M_2$ such that $f_1 \wedge f_2$ is equivalent to f_1

In words, $M_1 \sqsubseteq M_2$ means that M_2 has at least the constructs of M_1 and, for those in M_1, it allows at least the same variants (that is, the same configuration of values for its properties). For the example models:

[8] Without loss of generality, we assume that in a one-to-many relationship, it is the first entity that has a functional role, and so $\text{F}_1 = \textit{true}$ and $\text{F}_2 = \textit{false}$.

[9] Antisymmetry holds if we consider two models to be the same if their respective propositions are equivalent, which is clearly reasonable in our case. If not, we should say that $\sqsubseteq$ is a preorder.

- $M_{\text{RelNoN}} \sqsubseteq M_{\text{Rel}}$ (and $M_{\text{Rel}} \not\sqsubseteq M_{\text{RelNoN}}$): they have the same constructs, but M_{RelNoN} has a more restrictive condition on construct C than M_{Rel};
- $M_{\text{ERnoM2N}} \sqsubseteq M_{\text{ER}}$ (and $M_{\text{ER}} \not\sqsubseteq M_{\text{ERnoM2N}}$): they have the same constructs, but M_{ERnoM2N} has a more restrictive condition on construct R than M_{ER};
- $M_{\text{ERsimple}} \sqsubseteq M_{\text{ER}}$ (and $M_{\text{ER}} \not\sqsubseteq M_{\text{ERsimple}}$): the constructs in M_{ERsimple} are a proper subset of those in M_{ER} and, for each of them, the condition in M_{ERsimple} is at least as restrictive as the respective one in M_{ER};
- $M_{\text{ERnoM2N}} \not\sqsubseteq M_{\text{ERsimple}}$, $M_{\text{ERsimple}} \not\sqsubseteq M_{\text{ERnoM2N}}$: in fact M_{ERnoM2N} has AR which is not in M_{ERsimple}, but has a more restrictive condition on R.

We can define two binary operators on the space of models as follows:

$$\begin{aligned} M_1 \sqcup M_2 &= \{C(f_1 \vee f_2) \mid C(f_1) \in M_1 \text{ and } C(f_2) \in M_2\} \\ M_1 \sqcap M_2 &= \{C(f_1 \wedge f_2) \mid C(f_1) \in M_1 \text{ and } C(f_2) \in M_2\} \end{aligned}$$

If we refer only to the constructs that have a satisfiable proposition, then we write:

$$\begin{aligned} M_1 \sqcup M_2 = &\{C(f) \mid C(f) \in M_1 \text{ and there is no } C(f') \in M_2\} \cup \\ &\{C(f) \mid C(f) \in M_2 \text{ and there is no } C(f') \in M_1\} \cup \\ &\{C(f_1 \vee f_2) \mid C(f_1) \in M_1 \text{ and } C(f_2) \in M_2\} \\ M_1 \sqcap M_2 = &\{C(f_1 \wedge f_2) \mid C(f_1) \in M_1, C(f_2) \in M_2 \text{ and } f_1 \wedge f_2 \text{ is satisfiable } \} \end{aligned}$$

It can be shown that the space of models forms a lattice with respect to these two operators (modulo equivalence of propositions). The proofs of the claims that guarantee the lattice structure follow the definitions and the fact that the Boolean operators in propositional logic form a lattice. The *supermodel* (the fictitious most general model mentioned in the Introduction) is the top element of the lattice (with the true proposition for every construct). It is worth noting that models obtained as the result of these operations, especially $\sqcap$, could have, in some extreme cases, little practical meaning. For example, the bottom element of the lattice is the (degenerate) empty model, which has all false propositions (or, in other words, no constructs).

We can also define a **difference** operator on models

$$M_2 - M_1 = \{C(f_2 \wedge \neg f_1) \mid C(f_1) \in M_1 \text{ and } C(f_2) \in M_2\}$$

If we refer only to the constructs that have a satisfiable proposition, then we write:

$$\begin{aligned} M_2 - M_1 = &\{C(f_2 \wedge \neg f_1) \mid C(f_1) \in M_1 \text{ and } C(f_2) \in M_2\} \cup \\ &\{C(f_2) \mid C(f_2) \in M_2 \text{ and there is no } C(f_1) \in M_1\} \end{aligned}$$

This operator can also generate models with meaningless conditions, and indeed we will see in Section 6 that we use it only for technical steps in the search for translations.

4 Signatures of Datalog rules and their application

In order to handle rules and to reason on them, in an effective way, we introduce the notion of *signature* of a Datalog rule. The definition gives a unique construction, so the signature can be automatically computed for each rule.

As a preliminary step, let us define the **description of an atom** in a Datalog rule. An *atom* in Datalog has the form $C(\textsc{args})$, where C is a construct name and $\textsc{args}$ is a list of arguments (each of which is a pair composed of a field and a Skolem term or a constant or a variable). For example, the following are the two atoms in the body of rule $R_{1,6}$

Relationship(OID: rOid, sOID: src, Name: rN, Entity1: eOid, isOpt1: isOpt,
isFunct1: `false`, isFunct2: `false`)
Entity(OID: eOid, sOID: src, Name: eN)

Given an atom $C(\textsc{args})$, consider the fields in $\textsc{args}$ that correspond to properties (ignoring the others); let them be $p_1{:}v_1, \ldots, p_k{:}v_k$; each v_i is either a variable or a Boolean constant *true* or *false*. Then, the description of $C(\textsc{args})$ is a construct description $C(f)$, where the proposition f is the conjunction of literals corresponding to the properties in $p_1, \ldots, p_k$ that are associated with a constant; each of them is positive if the constant is *true* and negated if it is *false*. If there are no constants, then the proposition is *true*. For example, the descriptions of the two atoms in the body of rule $R_{1,6}$ are $\textsc{r}(\neg\textsc{f}_1 \wedge \neg\textsc{f}_2)$ and $\textsc{e}(true)$, respectively.

Let us now define the **signature of a Datalog rule**. Let R be a rule, with a head $C(\textsc{args})$ and a body with a list of atoms referring to constructs which need not be distinct $\langle C_{j_1}(\textsc{args}_1), C_{j_2}(\textsc{args}_2), \ldots, C_{j_h}(\textsc{args}_h)\rangle$; comparison terms (with inequalities, according to our hypotheses) do not affect the signature, and so we can ignore them. The signature r_R of R is a triple $(B, H, \textsc{map})$ where:

- B (the **body of** r_R) describes the applicability of the rule, by referring to the constructs in the body of R.
- H (the **head of** r_R) indicates the conditions that definitely hold on the result of the application of R, because of constants in its head.
- $\textsc{map}$ (the **mapping of** r_R) is a partial function that describes where values of properties in the head of the rule originate from.

More formally:

- B is a list of atom descriptions, $\langle C_{j_1}(f_1), C_{j_2}(f_2), \ldots, C_{j_h}(f_h)\rangle$, where $C_{j_i}(f_i)$ is the description of the atom $C_{j_i}(\textsc{args}_i)$.
- H is defined as the description $C(f)$ of the atom $C(\textsc{args})$ in the head.
- $\textsc{map}$ is a partial function whose domain is the set of properties of the construct in the head; $\textsc{map}$ is defined for the properties that are associated, in the head, with a variable (for our assumptions, each variable in the head appears also in the body, and only once). If a variable appears for a property p' in the head and for a property p of a construct C_{j_k} in the body, then $\textsc{map}$ is defined on p' and $\textsc{map}(p') = C_{j_k}(p)$.

Let us see the definition on rule $R_{1,6}$ in our running example, repeated here for convenience:

Rule $R_{1,6}$:

```
Relationship(OID: #relationship_1(eOid,rOid), sOID: tgt, Name: eN+rN,
      Entity1: #entity_1(rOid), isOpt1: false, isFunct1: true, isIdent: true,
      Entity2: #entity_0(eOid), isOpt2: isOpt, isFunct2: false)
 ←Relationship(OID: rOid, sOID: src, Name: rN, Entity1: eOid, isOpt1: isOpt,
      isFunct1: false, isFunct2: false),
   Entity(OID: eOid, sOID: src, Name: eN)
```

The body of the signature $r_{R_{1,6}}$ of $R_{1,6}$ is $B_{1,6} = \langle \text{R}(\neg\text{F}_1 \wedge \neg\text{F}_2), \text{E}(\mathit{true})\rangle$; indeed, the rule is applicable only to many-to-many relationships, that is, if both F_1 and F_2 are *false*. Similarly, for $R_{1,3}$ we have $B_{1,3} = \langle \text{R}(\text{F}_1)\rangle$, which is applied only to one-to-many relationships, as the body is the following (note the term isFunct1: true):

```
Relationship(OID: rOid, sOID: src, Name: n, Entity1: eOid1, isOpt1: isO1,
   isFunct1: true, isIdent: isId Entity2: eOid2, isOpt2: isO2, isFunct2: isF2)
```

The head of $r_{R_{1,6}}$ is $H_{1,6} = \text{R}(\neg\text{O}_1 \wedge \text{F}_1 \wedge \text{I} \wedge \neg\text{F}_2)$: the relationships produced by the rule all have O_1 and F_2 equal to *false* and F_1 and I equal to *true*.

The mapping for rule $R_{1,6}$ is $\text{MAP}_{1,6} = \langle \text{O}_2 : \text{R}(\text{O}_1)\rangle$ (we denote the function as a list of pairs, including only the properties on which it is defined). The name of the construct in the head is not mentioned, because it is known, but let us note that it might be different from the one in the body; this is the case for $R_{1,8}$ where $\text{MAP} = \langle \text{N} : \text{AR}(\text{N})\rangle$ and the first N is a property of A, the construct in the head:

Rule $R_{1,8}$:

```
AttributeOfEnt(OID: #attributeOfEnt_1(arOid), sOID: tgt, Name: n,
      IsKey: false, IsNullable: isN, EntityOID: #entity_1(rOid))
 ←AttributeOfRel(OID: arOid, sOID: src, Name: n, IsNullable: isN,
      Relationship: rOid)
   Relationship(OID: rOid, sOID: src, isFunct1: false, isFunct2: false)
```

Figure 6 shows the signatures for all the rules in program $\mathbf{P}_1$ (which we saw in Figure 5).

Then, we can define the *application* of the signature of a rule to a model, a notion that will be essential to show the main results in the paper, as sketched in Figure 1. We need two preliminary notions. First, we say that the signature $r_R = (B, H, \text{MAP})$ of a rule R is **applicable** to a model M if, for each $C_{j_i}(f_i)$ in B, there is $C_{j_i}(f^M_{j_i}) \in M$ such that $f^M_{j_i} \wedge f_i$ is satisfiable. In words, each construct in the body has to appear in the model, and the two propositions must not contradict one another. For example, $R_{1,6}$ is not applicable to M_{ERnoM2N} because we have $\text{R}(\neg\text{F}_1 \wedge \neg\text{F}_2)$ in the body of the rule and $\text{R}(\text{F}_1)$ in the model: the conjunction of $\neg\text{F}_1 \wedge \neg\text{F}_2$ and F_1 is not satisfiable.

Second, let us define the **transformation** $\mu_{\text{MAP}}()$ induced by mapping MAP on literals. In plain words, we use μ_{MAP} to "transfer" constraints on literals over properties in the body to literals in the head according to the MAP of the rule. Let l be a literal for a property p of an atom $C_{j_i}(\ldots)$ in the body of rule R. For example, in rule $R_{1,6}$, we have a literal for each property of atom Relationship; let us denote with o_1 the literal for property isOpt1 and with u_1 the literal for property isFunct1. Then, if $C_{j_i}(p)$ belongs to the range of MAP, with $\text{MAP}(p') = C_{j_i}(p)$, we have that $\mu_{\text{MAP}}(l)$ is a literal for the property p' with the same sign as l; if $C_{j_i}(p)$ does not belong to the range of MAP, then $\mu_{\text{MAP}}(l) = \mathit{true}$. In the example, still referring to

Signature $r_{R_{1,1}}$ for rule $R_{1,1}$:
$H_{1,1} = \langle \mathrm{E}(\mathit{true}) \rangle$
$B_{1,1} = \langle \mathrm{E}(\mathit{true}) \rangle$
$\textsc{map}_{1,1} = \langle \rangle$

Signature $r_{R_{1,2}}$ for rule $R_{1,2}$:
$H_{1,2} = \langle \mathrm{A}(\mathit{true}) \rangle$
$B_{1,2} = \langle \mathrm{A}(\mathit{true}) \rangle$
$\textsc{map}_{1,2} = \langle \mathrm{K} : \mathrm{A}(\mathrm{K}), \mathrm{N} : \mathrm{A}(\mathrm{N}) \rangle$

Signature $r_{R_{1,3}}$ for rule $R_{1,3}$:
$H_{1,3} = \langle \mathrm{R}(\mathrm{F}_1) \rangle$
$B_{1,3} = \langle \mathrm{R}(\mathrm{F}_1) \rangle$
$\textsc{map}_{1,3} = \langle \mathrm{O}_1 : \mathrm{R}(\mathrm{O}_1), \mathrm{I} : \mathrm{R}(\mathrm{I}),$
$\mathrm{O}_2 : \mathrm{R}(\mathrm{O}_2), \mathrm{F}_2 : \mathrm{R}(\mathrm{F}_2) \rangle$

Signature $r_{R_{1,4}}$ for rule $R_{1,4}$:
$H_{1,4} = \langle \mathrm{AR}(\mathit{true}) \rangle$
$B_{1,4} = \langle \mathrm{AR}(\mathit{true}), \mathrm{R}(\mathrm{F}_1) \rangle$
$\textsc{map}_{1,4} = \langle \mathrm{N} : \mathrm{AR}(\mathrm{N}) \rangle$

Signature $r_{R_{1,5}}$ for rule $R_{1,5}$:
$H_{1,5} = \langle \mathrm{E}(\mathit{true}) \rangle$
$B_{1,5} = \langle \mathrm{R}(\neg \mathrm{F}_1 \wedge \neg \mathrm{F}_2) \rangle$
$\textsc{map}_{1,5} = \langle \rangle$

Signature $r_{R_{1,6}}$ for rule $R_{1,6}$:
$H_{1,6} = \langle \mathrm{R}(\neg \mathrm{O}_1 \wedge \mathrm{F}_1 \wedge \mathrm{I} \wedge \neg \mathrm{F}_2) \rangle$
$B_{1,6} = \langle \mathrm{R}(\neg \mathrm{F}_1 \wedge \neg \mathrm{F}_2), \mathrm{E}(\mathit{true}) \rangle$
$\textsc{map}_{1,6} = \langle \mathrm{O}_2 : \mathrm{R}(\mathrm{O}_1) \rangle$

Signature $r_{R_{1,7}}$ for rule $R_{1,7}$:
$H_{1,7} = \langle \mathrm{R}(\neg \mathrm{O}_1 \wedge \mathrm{F}_1 \wedge \mathrm{I} \wedge \neg \mathrm{F}_2) \rangle$
$B_{1,7} = \langle \mathrm{R}(\neg \mathrm{F}_1 \wedge \neg \mathrm{F}_2), \mathrm{E}(\mathit{true}) \rangle$
$\textsc{map}_{1,7} = \langle \mathrm{O}_2 : \mathrm{R}(\mathrm{O}_2) \rangle$

Signature $r_{R_{1,8}}$ for rule $R_{1,8}$:
$H_{1,8} = \langle \mathrm{A}(\neg \mathrm{K}) \rangle$
$B_{1,8} = \langle \mathrm{AR}(\mathit{true}), \mathrm{R}(\neg \mathrm{F}_1 \wedge \neg \mathrm{F}_2) \rangle$
$\textsc{map}_{1,8} = \langle \mathrm{N} : \mathrm{AR}(\mathrm{N}) \rangle$

Fig. 6 The signatures for the rules in program $\mathbf{P}_1$

$R_{1,6}$, we have (see Figure 6) that $\textsc{map}_{1,6} = \langle \mathrm{O}_2 : \mathrm{R}(\mathrm{O}_1) \rangle$ and so property isFunct1 of Relationship does not belong to the range of $\textsc{map}$ (hence, $\mu_{\textsc{map}}(u_1) = \mathit{true}$ where u_1, as we said, is the literal for property F_1 in the source schema) whereas isOpt1 (abbreviated as O_1) does, with $\textsc{map}(\mathrm{O}_2) = \mathrm{R}(\mathrm{O}_1)$. Therefore, $\mu_{\textsc{map}}(o_1) = o_2$, where o_2 is the literal for property isOpt2 of the head construct. If the literal were negated in the source schema (that is, $\neg o_1$), then the application would have led to a negated literal as well (that is, $\mu_{\textsc{map}}(\neg o_1) = \neg o_2$) Let us define $\mu_{\textsc{map}}$ also on constants: $\mu_{\textsc{map}}(\mathit{true}) = \mathit{true}$ and $\mu_{\textsc{map}}(\mathit{false}) = \mathit{false}$. The notion can be extended to disjunctions and conjunctions: (i) $\mu_{\textsc{map}}(f_1 \wedge f_2) = \mu_{\textsc{map}}(f_1) \wedge \mu_{\textsc{map}}(f_2)$ (if $f_1 \wedge f_2$ is satisfiable, otherwise $\mu_{\textsc{map}}(f_1 \wedge f_2) = \mathit{false}$); (ii) $\mu_{\textsc{map}}(f_1 \vee f_2) = \mu_{\textsc{map}}(f_1) \vee \mu_{\textsc{map}}(f_2)$.

We are now ready for the definition of $r_R(M)$, the **application** of the signature of a rule R to a model M. In practice, such function combines constraints expressed in the head of the rule (which the rule forces in the result) and constraints of the source model that could be "transferred" to the output by means of variables in the body that appear also in the head of the rule. Let us see the details. If r_R is not applicable to M, then we define $r_R(M)$ as the empty set $\{\}$. The interesting case is when r_R is applicable to M. Let the signatures of the body and of the head of R be $B = \langle C_{j_1}(f_1), C_{j_2}(f_2), \ldots, C_{j_h}(f_h) \rangle$ and $H = C(f)$, respectively. For every atom $C_{j_i}(f_i)$ in the body, let $f^M_{j_i}$ be the proposition associated with C_{j_i} in the model.

Let us first give the definition in the special case where all the constructs in the source model M have propositions that are just conjunctions of literals, with no disjunctions. In this case,

$$r_R(M) = \{C(f')\} \text{ where } f' = f \wedge (\bigwedge_{i=1}^{h} \mu_{\textsc{map}}(f^M_{j_i} \wedge f_i))$$

Let us note that $(f_{j_i}^M \wedge f_i)$ is satisfiable, since the rule is applicable, and that it is just a conjunction of literals, because this is the case for f_i, by construction, and for $f_{j_i}^M$, by hypothesis. In words, the condition in the result is obtained as the conjunction of the proposition in the head, f, with those obtained, by means of MAP, from those in the source model (the $f_{j_i}^M$'s) and those in the body of the rule (the f_i's).

If the $f_{j_i}^M$'s include disjunctions, then let us rewrite $f_{j_i}^M \wedge f_i$ in disjunctive normal form $g_{i,1} \vee \ldots \vee g_{i,q_i}$. Then f' is built as the conjunction of the disjunctions of the applications of μ_{MAP} to the disjuncts: $f' = f \wedge (\bigwedge_{i=1}^{h}(\bigvee_{t=1}^{q_i} \mu_{\text{MAP}}(g_{i,t})))$.

Let us see three cases of applications of a rule in our running example. First, we have that $r_{R_{1,6}}(M_{\text{ERNoM2N}}) = \{\}$, as the rule is not applicable to the model (as we already saw).

Second, we have $r_{R_{1,6}}(M_{\text{ER}}) = \{\text{R}(\neg\text{O}_1 \wedge \text{F}_1 \wedge \text{I} \wedge \neg\text{F}_2)\}$. The rule is applicable since only construct R has an associated proposition and $(\text{F}_1 \vee \neg\text{F}_2) \wedge (\neg\text{F}_1 \wedge \neg\text{F}_2)$ is satisfiable, as it is equivalent to $\neg\text{F}_1 \wedge \neg\text{F}_2$. Then, applying the definition $f' = f \wedge (\bigwedge_{i=1}^{h}(\bigvee_{t=1}^{q_i} \mu_{\text{MAP}}(g_{i,t})))$ we have that the conjunction of the disjunctions $\mu_{\text{MAP}}(\ldots)$ is *true*, since the only property in the body mapped to the head is O_1, which does not appear in the argument of μ_{MAP}. Therefore, f' equals the condition f in the head of the signature: $f' = f = \neg\text{O}_1 \wedge \text{F}_1 \wedge \text{I} \wedge \neg\text{F}_2$.

As a third example, to see MAP and μ_{MAP} really in action, let us apply $R_{1,6}$ to model $M = \{\text{E}(\textit{true}), \text{R}((\text{F}_1 \vee \neg\text{F}_2) \wedge \neg\text{O}_1)\}$. The rule is applicable and we have $r_{R_{1,6}}(M) = \{\text{R}(\neg\text{O}_1 \wedge \text{F}_1 \wedge \text{I} \wedge \neg\text{F}_2 \wedge \neg\text{O}_2)\}$ as

$$\begin{aligned}
f' &= f \wedge \mu_{\text{MAP}}(((\text{F}_1 \vee \neg\text{F}_2) \wedge \neg\text{O}_1) \wedge (\neg\text{F}_1 \wedge \neg\text{F}_2)) \wedge \mu_{\text{MAP}}(\textit{true} \wedge \textit{true}) \\
&= f \wedge (\mu_{\text{MAP}}(\text{F}_1 \wedge \neg\text{O}_1 \wedge \neg\text{F}_1 \wedge \neg\text{F}_2) \vee \mu_{\text{MAP}}(\neg\text{F}_2 \wedge \neg\text{O}_1 \wedge \neg\text{F}_1 \wedge \neg\text{F}_2)) \wedge \textit{true} \\
&= f \wedge (\mu_{\text{MAP}}(\textit{false}) \vee (\mu_{\text{MAP}}(\neg\text{F}_2) \wedge \mu_{\text{MAP}}(\neg\text{O}_1) \wedge \mu_{\text{MAP}}(\neg\text{F}_2))) \\
&= f \wedge (\textit{false} \vee (\textit{true} \wedge \neg\text{O}_2 \wedge \textit{true})) \\
&= \neg\text{O}_1 \wedge \text{F}_1 \wedge \text{I} \wedge \neg\text{F}_2 \wedge \neg\text{O}_2
\end{aligned}$$

We are now ready for our final definition, that of the application of the signature of a Datalog program (implementing a basic translation) to a model. Let us first consider programs with no strongly recursive rules; we will remove this assumption before the end of the section. Given a program $\mathbf{P}$ consisting of a set of Datalog rules $R_1, R_2, \ldots, R_n$, the **application** of $\mathbf{P}$ to a model M is the least upper bound of the applications of the R_i's to M: $r_{\mathbf{P}}(M) = \bigsqcup_{i=1}^{n} r_{R_i}(M)$. In this way, we have a construct for each applicable rule and, if a construct is generated by more than one rule, the associated condition is the disjunction of the conditions of the various rules.

If we apply the program $\mathbf{P}_1$ in our running example to the ER model M_{ER}, then all constructs get copied and maintain the *true* proposition, except relationships, for which rules $R_{1,3}$, $R_{1,6}$ and $R_{1,7}$ generate, respectively, $\text{R}(\text{F}_1)$, $\text{R}(\neg\text{O}_1 \wedge \text{F}_1 \wedge \text{I} \wedge \neg\text{F}_2)$ (as we saw above) and $\text{R}(\neg\text{O}_1 \wedge \text{F}_1 \wedge \text{I} \wedge \neg\text{F}_2)$. Therefore, as the disjunction of the three formulas is F_1, the target model will have $\text{R}(\text{F}_1)$ and so we can say that the application of the signature of the program to M_{ER} produces M_{ERNoM2N}. The results in Section 5 will tell us that, as a consequence, the application of $\mathbf{P}_1$ to schemas of M_{ER} produce schemas of M_{ERNoM2N}.

Let us now consider also strongly recursive rules, that is, according to our definition, rules whose body includes atoms referring to the target schema. These rules may be applied on the basis of constructs generated by previous applications of other constructs. As a consequence, the application of signatures is also defined recursively, as a minimum fixpoint. We can redefine $r_{\mathbf{P}}$ to have two arguments, the source model and the target one, and its recursive application to a model M_0 is the fixpoint of the recursive expression $M = r_{\mathbf{P}}(M_0, M)$. Since it turns out that $r_{\mathbf{P}}()$ is monotonic, then, by Tarski's theorem [23], we have that the minimum fixpoint exists and can be obtained by computing $M_1 = r_{\mathbf{P}}(M_0, \bot)$ (where $\bot$ is the empty model), $M_{i+1} = r_{\mathbf{P}}(M_0, M_i)$ and stopping when $M_{i+1} = M_i$.

5 Inferring models from rules

In this section we show the main results of this paper, namely the fact that we can characterize the models obtained by applying Datalog rules by means of the syntactical notion of application of the signature of a rule to a model.

We need a few preliminary concepts. A **pseudoschema** is a set of ground atoms (called **ground constructs** hereinafter) each of which has the form $C(\text{OID}\!:\!o,\ p_1\!:\!v_1,\ \ldots,\ p_k\!:\!v_k,\ r_1\!:\!o_1,\ \ldots,\ r_z\!:\!o_z)$, where C is the name of a construct that has exactly the properties $p_1, \ldots, p_k$ and the references $r_1, \ldots, r_z$, each v_i is a Boolean constant and each o_i is an identifier. A **schema** is a pseudoschema that satisfies the referential constraints defined over constructs (see Section 2): that is, if a schema includes a ground construct $C(\ldots)$ with a reference $r\!:o$ (for example, a ground construct RELATIONSHIP(..., Entity1: o, ...)) and r is subject to a referential constraint to a construct C' (in the example, Entity1 has to refer to ENTITY), then the schema has to include a ground atom of the form $C'(\text{OID}\!:\!o, \ldots)$ (in the example, ENTITY(OID: o, ...) has to be in the schema).

Given a pseudoschema S_0, a schema S that contains all ground constructs in S_0 is called a **closure** of S_0. It can be shown that a closure of a pseudoschema can be obtained by applying procedure similar to the chase for inclusion dependencies [14], as follows: the procedure considers in turn each ground construct subject to a referential constraint and, if the constraint is violated, adds to the pseudoschema a new ground construct that repairs the violation (in the example in the previous paragraph, if the constraint is violated, then a ground construct ENTITY(OID: $o, \ldots$) is added, with variables for all properties; in general there can also be references, not here as ENTITY does not have any); the procedure terminates as we have acyclic referential constraints; at the end, variables can be replaced with Boolean values (somehow chosen, details are not relevant here) for all properties and with distinct integers not already appearing in the schema for identifiers.

A schema **belongs to a model** if its predicate symbols (that is, its constructs) belong to the model and, for each ground atom $C(\ldots)$, the Boolean values for its properties satisfy the proposition associated with C in the model.

Given a ground construct $c = C(\text{OID}\!:\!o,\ p_1\!:\!v_1,\ \ldots,\ p_k\!:\!v_k,\ r_1\!:\!o_1,\ \ldots,\ r_z\!:\!o_z)$, we define the **description** of c, denoted with $\text{SIG}(c)$, as $C(f)$ where $f = l_1 \wedge \ldots \wedge l_k$, and each l_j is a literal with the symbol p_j, positive if $v_j = \mathit{true}$ and negated if $v_j = \mathit{false}$.

As an example, $\text{R}(\text{OID}\!:\!o,\ \text{O}_1\!:\!\mathit{false},\ \text{F}_1\!:\!\mathit{false},\ \text{I}\!:\!\mathit{false},\ \text{O}_2\!:\!\mathit{true},\ \text{F}_2\!:\!\mathit{false},\ \ldots)$ is a ground construct (with references omitted as not relevant) describing a many-to-many relationship, optional on one side and not optional on the other and without external identification. Its description is $\text{R}(\neg\text{O}_1 \wedge \neg\text{F}_1 \wedge \neg\text{I} \wedge \text{O}_2 \wedge \neg\text{F}_2)$.

The notion of description can be extended to schemas: given a schema S, we define $\textsc{sig}(S) = \bigsqcup_{c\in S}\{\textsc{sig}(c)\}$. It is interesting to note (even if we will not use this property) that $\textsc{sig}(S) = \sqcap\{M|S \text{ belongs to } M\}$ (that is, $\textsc{sig}(S)$ is the greatest lower bound of the models to which S belongs). Therefore, $\textsc{sig}(S) \sqsubseteq M$ if and only if S belongs to M.

We are now ready to show our results.

Lemma 1 *Let M be (the description of) a model and R a Datalog rule. For each ground construct c in the pseudoschema $R(S)$ produced by the application of R to a schema S that belongs to M, it is the case that $\{\textsc{sig}(c)\} \sqsubseteq r_R(M)$.*

Proof The proof is trivial if R is not applicable to S, because in this case the pseudoschema $R(S)$ is empty and so there is no ground construct c in $R(S)$.
If it is the case that R is applicable to S, let $c = C(\textsc{oid}\colon o, p_1\colon v_1, p_2\colon v_2, \ldots, p_k\colon v_k, r_1{:}o_1, r_2{:}o_2, \ldots, r_z{:}o_z)$ be a ground construct in $R(S)$ (that is, generated by the application of R to S).
The proof proceeds by first showing that r_R is applicable to M: the idea is that if the rule generates a new ground construct, then, for each of its body literals there is some construct in the schema that unifies[10] with it; therefore the schema element satisfies both the condition in the body and that in the model and so their conjunction is satisfiable.
Let R be the rule $C(\ldots) \leftarrow C_{j_1}(\ldots), C_{j_2}(\ldots), \ldots, C_{j_h}(\ldots)$ and let $\langle C_{j_1}(f_1), C_{j_2}(f_2), \ldots, C_{j_h}(f_h)\rangle$ be the body B of the signature r_R of R. As c was generated, then the rule R was applicable, hence S contains at least h ground constructs c_i of the form $C_{j_i}(\ldots)$ (for $i = 1, 2, \ldots, h$) such that each of them unifies with an atom of body of R.
Let be φ_i the assignment of values to properties corresponding to c_i, for each i in $[1, h]$. We have:

- φ_i satisfies f_i because c_i unifies with the ground constructs $C_{j_i}(\ldots)$, as we said above;
- φ_i satisfies $f_{j_i}^M$ (remember that $f_{j_i}^M$ is the formula associated to C_{j_i} in the model M) because the schema S belongs to M and c_i belongs to S.

So we have an assignment of values to properties that satisfy f_i and $f_{j_i}^M$ both, therefore $f_{j_i}^M \wedge f_i \neq F$; this implies r_R is applicable to M.
Then, since r_R is applicable to M, we have that $r_R(M) = C(f')$, with $f' = f \wedge (\bigwedge_{i=1}^{h} \mu_{\textsc{map}}(f_{j_i}^M \wedge f_i))$.
Then, the proof shows that the assignment $\varphi : (p_1 = v_1, p_2 = v_2, \ldots, p_k = v_k)$ (that is, the one with the constants in c) satisfies proposition f', by showing that it satisfies both f and $\bigwedge_{i=1}^{h} \mu_{\textsc{map}}(f_{j_i}^M \wedge f_i)$:

- φ satisfies f because constants in the head of rule R are present also in ground construct generated c by construction (in fact f is generated using such constants of the head) and therefore in φ;

[10] This is the usual notion of unification in logic programming: here it means that there is an occurrence of a construct of the involved type with properties that agree with those that are constrained in the literal.

– φ satisfies $\bigwedge_{i=1}^{h} \mu_{\text{MAP}}(f_{j_i}^{M} \wedge f_i)$; we show this by proving that φ indeed satisfies $\mu_{\text{MAP}}(f_{j_i}^{M} \wedge f_i)$, for every $i = 1, \ldots, h$.
  Since R is applicable to S (and remembering R has h atoms in the body), there exist h assignments $\varphi_1, \varphi_2, \ldots, \varphi_h$ to properties of constructs $C_{j_1}, C_{j_2}, \ldots, C_{j_h}$ that satisfy formulas $f_{j_1}^{M} \wedge f_1$, $f_{j_2}^{M} \wedge f_2$, …, $f_{j_h}^{M} \wedge f_h$, respectively.
  Let us consider an assignment φ_i satisfying $f_{j_i}^{M} \wedge f_i$ and rewrite this formula in disjunctive normal form (if necessary), obtaining $g_{i,1} \vee g_{i,2} \vee \ldots \vee g_{i,q_i}$; so we have $f_{j_i}^{M} \wedge f_i \equiv \bigvee_{t=1}^{q_i} g_{i,t}$, hence $\mu_{\text{MAP}}(f_{j_i}^{M} \wedge f_i) \equiv \bigvee_{t=1}^{q_i} \mu_{\text{MAP}}(g_{i,t})$ and there exists at least one term $g_{i,\tau}$ satisfied by φ_i.
  Now we show that φ satisfies $\mu_{\text{MAP}}(g_{i,\tau}) = \mu_{\text{MAP}}(l_1) \wedge \mu_{\text{MAP}}(l_2) \wedge \ldots \wedge \mu_{\text{MAP}}(l_w)$ because it satisfies all terms $\mu_{\text{MAP}}(l_v)$ for $v = 1, 2, \ldots, w$. We have the following two cases:
  1. if $C_{j_i}(p_v)$ (where p_v is the property associated with literal l_v) does not belong to the codomain of the map of the rule, then $\mu_{\text{MAP}}(l_v) =$ `true` and so φ satisfies it;
  2. if $C_{j_i}(p_v)$ belongs to the codomain of the map of the rule, then φ assigns to the property p^* associated with p_v via the map ($\text{MAP}(p^*) = C_{j_i}(p_v)$), by means of repeated variables in the head and the body of the rule R, the same value that φ_i assigns to p_v; φ_i satisfies l_v (because it satisfies $g_{i,\tau}$) and $\mu_{\text{MAP}}(l_v)$ has the same sign of l_v (by definition of $\mu_{\text{MAP}}(l)$), therefore φ satisfies $\mu_{\text{MAP}}(l_v)$.

□

Lemma 2 *Let M be a model and R a Datalog rule. If s is a construct description such that $\{s\} \sqsubseteq r_R(M)$ then there is a schema S that belongs to M such that the application of R to S produces a pseudoschema $R(S)$ that contains exactly one construct c such that* $\text{SIG}(c) = s$.

Proof The proof proceeds by considering a construct c with description $s = \text{SIG}(c)$ and showing that there is a set of ground constructs belonging to S corresponding to the atoms in the body of R out of which c can be produced.
Let $R = C(\ldots) \leftarrow C_{j_1}(\ldots), C_{j_2}(\ldots), \ldots, C_{j_h}(\ldots)$. Consider a pseudoschema S_0 that contains h ground constructs (one for each atom of the body of the rule R), with repeated oids for repeated variables in the body of R and distinct ones elsewhere. The values of Boolean properties of these ground constructs are copied (extracted) from constants in the body of R or traced back from the formula in s, by means of MAP (which gives no ambiguity, as there are no repeated Boolean values in the head); if these constructs have other properties, such properties can have arbitrary values.
The values of properties induced by the body are required to guarantee applicability of R to S_0. So, given an atom $C_{j_i}(f_i)$ of the body of r_R, we use f_i to initialize values of properties of ground constructs c_i corresponding to such atom. In particular we assign a *true* value to properties corresponding to literals positive in f_i and *false* value to ones corresponding to literals negated.
Let us consider values induced by MAP; the assignment φ of values to properties induced by s satisfies the formula f' of $\{C(f')\} = r_R(M)$ and, in particular, it satisfies the conjunction $\bigwedge_{i=1}^{h} \mu_{\text{MAP}}(f_{j_i}^{M} \wedge f_i)$. Let us remember that each formula $f_{j_i}^{M} \wedge f_i$ of the conjunction corresponds to an element of the body of the rule and so it is associated with one of the h ground constructs of S_0 previously introduced,

and let us consider a generic formula $\mu_{\text{MAP}}(f_{j_i}^M \wedge f_i)$ and corresponding ground construct c_i.
If we rewrite $f_{j_i}^M \wedge f_i$ in disjunctive normal form, we have $\mu_{\text{MAP}}(f_{j_i}^M \wedge f_i) \equiv \bigvee_{t=1}^{q_i} \mu_{\text{MAP}}(g_{i,t})$ and φ, obviously, satisfies at least one element $\mu_{\text{MAP}}(g_{i,\tau})$. Let us consider a generic $g_{i,\tau}$ satisfied by φ and let $g_{i,\tau} = l_1 \wedge l_2 \wedge \ldots \wedge l_w$; so φ satisfies all the terms $\mu_{\text{MAP}}(l_v)$ for $v = 1, 2, \ldots, w$ and we can use value assigned to literal $\mu_{\text{MAP}}(l_v)$ by φ to initialize value of property of ground construct c_i corresponding to literal l_v using ${\mu_{\text{MAP}}}^{-1}$.
Then let us assign arbitrary values to other properties of c_i not initialized yet: this is possible because it means these properties are not directly involved in rule R; therefore we found h ground constructs c_i and each of these satisfies a formula $f_{j_i}^M \wedge f_i$ (because they satisfy at least one element $g_{i,\tau}$ by construction) and so pseudoschema S_0, made of these ground constructs, belongs to the model M (because of the $f_{j_i}^M$'s) and rule R is applicable to S_0.
Then, consider a closure $S = S_0^+$ of S_0, which is a schema. Applying R to S, we obtain exactly a construct c' such that $s = SIG(c') = SIG(c)$, that is the only element of the pseudoschema $R(S)$. □

Let us briefly comment on the latter lemma. Given our example model M_{ER} and rule $R_{1,6}$, we have that (as we saw) $r_{R_{1,6}}(M_{\text{ER}}) = \{\text{R}(\neg\text{O}_1 \wedge \text{F}_1 \wedge \text{I} \wedge \neg\text{F}_2)\}$. The lemma says that all construct descriptions s such that $\{s\} \sqsubseteq r_{R_{1,6}}(M_{\text{ER}})$ can be obtained as a result of $R_{1,6}$ applied to some schema of M_{ER}. For example, description $s = \{\text{R}(\neg\text{O}_1 \wedge \text{F}_1 \wedge \text{I} \wedge \neg\text{O}_2 \wedge \neg\text{F}_2)\}$, which satisfies $\{s\} \sqsubseteq r_{R_{1,6}}(M_{\text{ER}})$ can be obtained by applying rule $R_{1,6}$ to a schema which includes (together with other constructs) at least a many-to-many relationship, and all of them with $\text{O}_1 = \textit{false}$; this follows from $\text{MAP}_{1,6} = \langle \text{O}_2 : \text{R}(\text{O}_1)\rangle$.

Lemmas 1 and 2 can be synthesized as the following theorem, which describes the behavior of individual Datalog rules with respect to models and descriptions of schemas. It says that using the signature of a rule we can characterize the descriptions of the constructs that can be generated out of a given model.

Theorem 1 *Let M be a model, R a Datalog rule and s a construct description. Then $\{s\} \sqsubseteq r_R(M)$ if and only if there is a schema S that belongs to M such that $R(S)$ contains exactly one construct c such that* $\text{SIG}(c) = s$.

Let us now extend the results to Datalog programs.

Lemma 3 *Let M be a model and $\boldsymbol{P}$ a Datalog program. For every schema S that belongs to M, the application of $\boldsymbol{P}$ to S produces a schema $\boldsymbol{P}(S)$ that belongs to $r_{\boldsymbol{P}}(M)$.*

Proof Let S be a schema that belongs to model M. The result of the application of $\mathbf{P}$ to S produces a pseudoschema that is indeed a schema because the program is coherent with respect to referential constraints by hypothesis.
Then, the fact this schema belongs to $r_{\mathbf{P}}(M)$ is a consequence of:

– Lemma 1;
– the definition of $r_{\mathbf{P}}(M)$ as $\bigsqcup_{i=1}^n r_{R_i}(M)$ where n is the number of rules composing $\mathbf{P}$;
– the fact that, obviously, $r_{R_j}(M) \sqsubseteq \bigsqcup_{i=1}^n r_{R_i}(M)$ with j in $[1, n]$.

Let us see the details. If a ground construct $c = C_j(f) = r_{R_j}(S)$ is generated by one of the rules R_j of the program $\mathbf{P}$ applied to a a schema S that belongs to M, then, by Lemma 1, $\{\textsc{sig}(c)\} \sqsubseteq r_{R_j}(M)$. But, as we said, R_j is one of the rules of program $\mathbf{P}$, and so, since $r_{\mathbf{P}}(M) = \bigsqcup_{i=1}^{n} r_{R_i}(M)$, we have that $r_{R_j}(M) \sqsubseteq \bigsqcup_{i=1}^{n} r_{R_i}(M)$, and so $\{\textsc{sig}(c)\} \sqsubseteq r_{\mathbf{P}}(M)$. This implies that the assignment of values to properties induced by the formula f of $c = C_j(f)$ satisfies $r_{\mathbf{P}}(M)$. Therefore as c is a generic construct generated by $\mathbf{P}$, we have that all constructs generated by rules of $\mathbf{P}$ satisfy $r_{\mathbf{P}}(M)$ and so $\mathbf{P}(S)$ belongs to $r_{\mathbf{P}}(M)$. □

Lemma 4 *Let M be a model and $\boldsymbol{P}$ a Datalog program. If S' is a schema that belongs to $r_{\boldsymbol{P}}(M)$, then there is a schema S that belongs to M such that* $\textsc{sig}(S') \sqsubseteq \textsc{sig}(\boldsymbol{P}(S))$.

Proof Let us first consider non-recursive programs.
The proof is essentially based on Lemma 2: for every ground construct c in S', we have (partly by hypothesis and partly by definition), that $\{\textsc{sig}(c)\} \sqsubseteq \textsc{sig}(S') \sqsubseteq r_{\mathbf{P}}(M)$; so, there is a construct description $C(f)$ in $r_{\mathbf{P}}(M)$ such that c has the predicate symbol C and its assignment of values to properties satisfies f.
Now, by definition of $r_{\mathbf{P}}(M)$, we have that f is obtained as the disjunction of the formulas associated with the various rules that have C in the head and therefore c (since it satisfies f) has to satisfy one of them. If R is such a rule, it turns out that $\{\textsc{sig}(c)\} \sqsubseteq r_R(M) \sqsubseteq r_{\mathbf{P}}(M)$ and so, by Lemma 2 we have that there is a schema S_c of M such that $R(S_c)$ contains a construct c' such that $\textsc{sig}(c) = \textsc{sig}(c')$.
Then, for each construct c in S', let us consider the corresponding schema S_c of M claimed by Lemma 2 and consider the “union” of such schemas for the various constructs, $S = \bigsqcup_{c \in S'}(S_c)$, and a closure S^+ of it (as defined in Section 5): S^+ is a schema for model M by construction because obtained as the closure of union of schemas belonging to M and $\textsc{sig}(S') \sqsubseteq \textsc{sig}(\mathbf{P}(S^+))$ by construction of S^+.
The proof for recursive rules proceeds by induction on the number of steps needed to reach the fixpoint, with the induction step based on the arguments above. □

Theorem 2 *Let M be a model and $\boldsymbol{P}$ a Datalog program. Then a schema S' belongs to $r_{\boldsymbol{P}}(M)$ if and only if there is a schema S that belongs to M such that* $\textsc{sig}(S') \sqsubseteq \textsc{sig}(\boldsymbol{P}(S))$.

Theorem 2 synthesizes Lemmas 3 and 4 in a direct way. However, there is another point of view, which is more interesting, as follows.

Theorem 3 *Let M be a model and $\boldsymbol{P}$ a Datalog program. Then,*

1. *for every schema S that belongs to M, it is the case that* $\textsc{sig}(\boldsymbol{P}(S)) \sqsubseteq r_{\boldsymbol{P}}(M)$
2. *there is a schema S that belongs to M such that* $\textsc{sig}(\boldsymbol{P}(S)) = r_{\boldsymbol{P}}(M)$

Proof By Lemma 3, we have that $\mathbf{P}(S)$ belongs to $r_{\mathbf{P}}(M)$ and so $\textsc{sig}(\mathbf{P}(S)) \sqsubseteq r_{\mathbf{P}}(M)$, which is Claim 1.
In order to prove Claim 2, let us first argue that, given $r_{\mathbf{P}}(M)$, there exists a schema S' whose description $\textsc{sig}(S')$ is exactly $r_{\mathbf{P}}(M)$. In fact, we can build a schema S' as follows: for each construct description $C(f)$ in $r_{\mathbf{P}}(M)$ such that the construct is not subject to a referential constraint, we have a set of ground constructs for C that show all the possible combinations of properties that satisfy f. Then, for each of the constructs that are subject to referential constraints, we introduce again a set of ground constructs that show all the possible combinations of properties that

satisfy the associated proposition, and have references to constructs that have one of the allowed configuration of properties for the referred construct. Indeed, S' is a schema, because it is constructed satisfying the referential constraints and its description is exactly $r_{\mathbf{P}}(M)$ because for each construct all allowed combinations of properties appear. Then by Lemma 4 we have that there is a schema S such that $\textsc{sig}(S') \sqsubseteq \textsc{sig}(\mathbf{P}(S))$, that is, $r_{\mathbf{P}}(M) \sqsubseteq \textsc{sig}(\mathbf{P}(S))$ and by Lemma 3 we have that $\textsc{sig}(\mathbf{P}(S)) \sqsubseteq r_{\mathbf{P}}(M)$. □

Theorem 3 is our main result. It states that the derivation of model descriptions by means of the application of the signatures of Datalog programs is "sound and complete" with respect to the models generated by the program, in the sense that a Datalog program can generate schemas with all and only the descriptions generated by the application of the signature of rules. In other words, signatures completely characterize the models that can be generated by means of a Datalog program.

6 Applications of the results

The technical development of the previous sections can be used in various ways to support the activities of an actual tool for schema translation, such as the MIDST tool [5–7] we have developed.

A simple use of the result is the possibility offered to check which is the model obtained as the result of the application of a Datalog program: the results in Section 5 allow the "rule designer"[11] to know the target model without running (or inspecting) Datalog rules, but simply generating the signatures of rules and the description of a schema (model) and running their applications against the schema (model).

A related use, still in rule specification, is the possibility to check whether a Datalog program takes into consideration all the constructs of a given source model, that is whether the application of a Datalog program to a given source model causes a loss of information. Let us say that the **domain** of a rule with respect to a model is the set of constructs of M that are considered by the rule; formally, given a rule R and a model M, if R is applicable to M, then the domain $\textsc{dom}(r_R, M)$ of R with respect to M is the set of the constructs of M that unify with the atoms of the body B of the signature of R; if R is not applicable to M, then $\textsc{dom}(r_R, M)$ is the empty model. We can extend this notion to Datalog programs. Given a program $\mathbf{P}$ and a model M, the **domain** of $\mathbf{P}$ with respect to M, $\textsc{dom}(r_{\mathbf{P}}, M)$ is $\bigsqcup_{R \in \mathbf{P}'} \textsc{dom}(r_R, M)$ where $\mathbf{P}'$ denotes the program that includes only the rules in $\mathbf{P}$ that are applicable to M. Now, constructs (or variant of them) **ignored** by a program $\mathbf{P}$ when applied to a model M are those that do not unify with any atom of the body of any Datalog rule of $\mathbf{P}$, that is, those in the difference between M and $\textsc{dom}(r_{\mathbf{P}}, M)$. Clearly, the presence of ignored constructs is the indication of the fact that a program probably needs additional rules. Our techniques support the rule designer in discovering these situations. For example, if we have a program that ignores many-to-many relationships and one applies it

[11] The rule designer is a high level user, also called *metamodel engineer* [7]. The "standard" user would be a designer, who is interested in translations.

to an ER schema that contains many-to-many relationships, these relationships would get lost.

A more ambitious goal would involve the automatic selection of rules for the generation of complex translations out of a library. A general approach for this, followed also by other authors in similar contexts [12, 22], would be based on the generation of a search tree and on the adoption of heuristics (for example based on A*-type algorithms) that can be satisfactory. The formal system introduced in this paper allows the automatic generation of concise description of translation steps, and then could be the basis for the application of such algorithm but it has several drawbacks:

– it could be computationally unfeasible, because the number of basic steps can grow and its termination in general need not be guaranteed, as multiple application of rules could arise, with no bounds (i.e. a rule could introduce and eliminate the same constructs in turn, producing a loop);
– it could produce a translation plan driven for a specific criterion (it depends on the heuristics adopted) and hence the result plan may differ from the "optimal" one (provided that a notion of optimality can be given, for example in terms of the number of steps).

In the remainder of the section, we propose a different algorithm that, under suitable assumptions, is effective and more efficient. The assumptions are formalizations of some observations derived by our experience with the tool, in terms of both definition of models and specification of rules.

In fact, it can be observed that most translations, when applied to certain models, return a schema that is more restricted than the input, because they just eliminate a feature. Eliminations can be performed by dropping a construct or reducing its variants. We use the term *reduction* to refer to these translations. The others are called *transformation.* They introduce new constructs (or new variants of constructs), beside eliminations of some constructs. With reference to the example of Section 1, the elimination of generalizations is a reduction, performed by dropping the constructs devoted to represent generalizations, substituting them with new references; the elimination of many-to-many relationships is performed adding constraints to the formula of the construct devoted to represent relationships; the replacement of relationships with references is an example of a transformation.

A second observation is that we have few "families" of models (according to a notion to be defined shortly), such as ER, OO and relational, and we manage many variations for each family. With respect to the previous observation, reductions allow to move within a family (i.e. they return a schema or model of the same family), while transformations allow to move toward another family. Again with reference to Section 1, the elimination of generalizations is a reduction within the OO family; the elimination of many-to-many relationships is a reduction within the ER family; the replacing of relationships with references is a transformation from the ER to the OO family.

Formally, a family of models $\mathcal{F}$ is a set of models defined by means of a model M^* (called the **progenitor** of $\mathcal{F}$) and a set of models $M_{*,1}, \ldots, M_{*,k}$ (the **minimal** models of $\mathcal{F}$) and contains all models that are subsumed by M^* and subsume at least one of the $M_{*,i}$'s:

$$\mathcal{F} = \{M \mid M \sqsubseteq M^* \text{ and } \; M_{*,i} \sqsubseteq M, \text{ for some } \; 1 \leq i \leq k\}$$

For example, the model M_{ER} of our previous examples should be the progenitor of the ER family, and a model with Entities and Relationships only should be one of the minimal models of such family.

Let us now formalize the notions of **reduction** and **transformation**. A translation $\mathbf{P}$ is a reduction for a family $\mathcal{F}$ if, when applied to a schema S of a model $M \in \mathcal{F}$, it generates a schema that is subsumed by the input ($\mathbf{P}(S) \sqsubseteq S$). The translations that are not reductions for a certain family are indeed transformations for such family (i.e. they typically eliminate one or more constructs of the input and introduce new ones).

For example, translations $\mathbf{P}_1$ and $\mathbf{P}_2$ of Section 2 are a reduction within the ER family and a transformation from the ER family to the relational one, respectively.

Now we can present our assumptions on the set of basic translations.

Assumption 1 *For each pair of families $\mathcal{F}_1$, $\mathcal{F}_2$, there are a model M_1 in $\mathcal{F}_1$ and a translation $\mathcal{T}$ such that, for each schema S_1 of M_1:*

1. *$\mathcal{T}$ does not ignore any construct of S_1;*
2. *$\mathcal{T}$ produces a schema that belongs to the progenitor M_2^* of $\mathcal{F}_2$.*

This hypothesis requires the existence of F^2 translations, if F is the number of different families. This is not a real problem, as F is reasonably small and, whatever the approach, these rules would be needed. In general there might even be pairs of families with more than one translation, but we ignore this issue, as it would not add much to the discussion. Indeed, these translations could in turn be composed of various basic ones, but this is also not essential here.

Assumption 2 *For each family $\mathcal{F}$, for each minimal model $M_{*,i}$ of $\mathcal{F}$, there is a translation from the progenitor M^* of $\mathcal{F}$ to $M_{*,i}$, entirely composed of reductions that do not ignore constructs.*

For example, this assumption requires that, in the ER family, there is a set of reductions to move from a complete ER with generalizations, many-to-many relationships, attributes on relationships to a minimal ER with entities, attributes of entities and binary relationships only, without ignoring any construct. If there were no reduction to transform many-to-many relationships, it would be violated. The satisfaction of this assumption can be verified by considering the reductions for the family (that is, the basic translations that are reductions for the progenitor of the family) and performing an exhaustive search on them. This is in principle inefficient, but in practice it can be done in a fast way, as the number of reductions in a family is small, and in fact most of them are commutative, because they eliminate variants of constructs, which are often independent of one another.

We proved in [7] that, if the set of basic translations satisfies Assumption 1 and Assumption 2, then, for each family and each pair of models M_1 and M_2 within it, there is a translation from M_1 to M_2 that does not ignore constructs.

On the basis of these assumptions, with the technical machinery developed in this paper we can give the details of an algorithm that always finds a complete transformation with the following structure:

1. a reduction (composed of a sequence of translations that are reductions) within the source family;
2. a transformation from the source family to the target family;

3. a reduction within the target family.

On the basis of Assumption 1, the transformation of step 2 always exists. Then the first set of reductions (step 1) is needed to transform the source schema into another schema belonging to the model M_1 mentioned in Assumption 1. Applying the translations of steps 1 and 2 to the source schema, we obtain a resulting schema that need not belong to the target model, hence the second set of reductions (step 3) is needed to guarantee it.

The previous assumptions and arguments justify the algorithm shown in Figure 7. The algorithm has an input that is composed of a source schema S_1 and a target model M_2, and refers to a given set of families and a given set of rules.

```
findCompleteTranslation($S_1, M_2$)
1.    $\mathcal{F}_1$ = family($S_1$)
2.    $\mathcal{F}_2$ = family($M_2$)
3.    $\mathcal{T}$ = getTransformation($\mathcal{F}_1, \mathcal{F}_2$)
4.    $M_1'$ = getSource($\mathcal{T}$)
5.    $\mathcal{T}_1$ = getReduction($\mathcal{F}_1, M_1'$)
6.    $\mathcal{T}_2$ = getReduction($\mathcal{F}_2, M_2$)
7.    **return** $\mathcal{T}_1 \circ \mathcal{T} \circ \mathcal{T}_2$
```

Fig. 7 Algorithm findCompleteTranslation

Let us comment on the various steps of the algorithm.

Lines 1 and 2 find the families to which the source schema and the target model belong, respectively. In order to find the family to which a schema (model) belongs, it is enough to test the "inclusion" of such schema (model) against the progenitors of the families. This is feasible if the number of families is small, and we assume this to be the case. Let S be a schema and M^* the progenitor of family $\mathcal{F}$, if $S \sqsubseteq M^*$ then $S \in \mathcal{F}$.

Line 3 selects the transformation between the two families, whose existence is guaranteed by Assumption 1. A Datalog program $\mathbf{P}$ is a transformation between two families $\mathcal{F}_1$, $\mathcal{F}_2$ (with progenitors M_1^* and M_2^*, respectively), if $\bigsqcup_{R\in\mathbf{P}} B_R \sqsubseteq M_1^*$ and $r_{\mathbf{P}}(M_1) \sqsubseteq M_2^*$. It is interesting to note that this operation can be performed off-line for each pair of families and not during a transformation process.

Then, line 4 computes the source model M_1' for transformation $\mathcal{T}$. It is the "union" of the bodies of the selected transformation: $\bigsqcup_{R\in\mathbf{P}} B_R$.

Next, line 5 finds the sequence of reductions needed to go from the progenitor of $\mathcal{F}_1$ to M_1' (on the basis of Assumption 2) and line 6 does the same within the target family. The first step to find a reduction (that is, a sequence of reduction programs) toward a model within a family is the search for reductions for that model, testing if the application of a program to the progenitor of the family returns a model subsumed by the progenitor. Then we have to order the reductions to avoid that a program could introduce a construct eliminated by a previous program. Given two Datalog programs $\mathbf{P}_1$ and $\mathbf{P}_2$, $\mathbf{P}_1$ precedes $\mathbf{P}_2$ if $\bigsqcup_{R\in\mathbf{P}_1} H_R \sqcap \bigsqcup_{R\in\mathbf{P}_2} B_R \neq \emptyset$. Obviously we have also to check that, at each step, no information gets lost (this is done by verifying that there are no ignored constructs). Finally, after we found a reduction, we can optimize it with respect to the actual input, which need not be the progenitor of a family. Again we note that the first two steps of this procedure can be done off-line for each family and perform just the optimization step at run-time during the transformation.

Finally, the algorithm returns a translation that is the concatenation of $\mathcal{T}_1$, $\mathcal{T}$, and $\mathcal{T}_2$.

7 An enhanced supermodel

In this section we show that the ideas presented in the paper make sense even if we remove the assumption that references of constructs are always required.

The usefulness of the MIDST proposal depends on the expressive power of its supermodel, that is the set of models handled and accuracy and precision of the representation of such models. In order to improve the expressive power of the supermodel, it has been necessary to introduce new constructs, often just variants of pre-existing ones, thus observing a growth in the number of constructs.

A key observation is the following: many constructs, despite differences in their syntactical structure, are semantically similar or identical, in the following sense. Recalling our running example, in the ER model, attribute of entities and attribute of relationships show some similarity: even if they have different references (toward an entity and toward a relationship, respectively), and the first one has an extra property (isKey), they both represent a lexical value. Moreover, also a column in the Relational model is very similar to an attribute of the ER model, even if it has a reference toward a table. In these cases, two or more constructs can be collapsed into a unique construct with their common semantics and a structure obtained by the union of the structures of the involved constructs. Clearly, constructs thus obtained have some optional references, together with some mandatory ones.

This observation on the one hand allows us to obtain a more compact supermodel (i.e. smaller number of constructs) and cohesive (i.e. one construct to represent all concepts with same semantics) but on the other hand it causes a general complication of Datalog rules: this implies the necessity to refactor the formal system introduced in order to keep on reasoning on data models.

In the rest of this section we briefly illustrate how the concepts previously introduced need to be changed. The intuition is that, now, for every construct, we have to represent a formula over its properties and references, hence the universe is:

$$\mathcal{U} = \{C_1(P_1 \cup Ref_1), C_2(P_2 \cup Ref_2), \ldots, C_u(P_u \cup Ref_u)\}$$

Recalling our running example, the definition of attribute, in the new approach, should be: $Attribute(\{isKey, isNullable, Entity, Relationship, Table\})$.

The form for the description of a model is still a set of constructs, each with an associated formula. Let us assume that the formula associated with a construct C is in disjunctive normal form. Each disjunct has the following form:

$$p_1 \wedge p_2 \wedge \ldots \wedge p_n \wedge ref_1 \wedge ref_2 \wedge \ldots \wedge ref_m$$

where the p_i's are atoms for the properties and the ref_j's are atoms for references. It states that construct C has non-null values exactly for the references corresponding to positive ref_j's, and its properties must satisfy the constraints expressed by the conjunctions of the p_i's. We want to remark that the meaning of an atom for a reference in a formula is different from that of an atom for a property: a positive atom for a reference states that such references must be non-null; a negative one states that such references must be null.

Now formulas are more complex, but it is due to increased complexity of the structure of the constructs. On the other hand they are more expressive, since with a single formula now we can express constraints not only on properties but also on references. Moreover we can express also constraints on the relationships between references of a construct, in the sense that, for each construct, we can force the presence of specific combinations of references and avoid other ones. For example we can state that both the references of a relationship toward entities must be valued at the same time, while just one of the three references of an attribute can be valued.

Some of the models discussed in Section 2, would be as follows:

- $M_{\mathrm{ER}} = \{\textsc{e}(\mathit{true}), \textsc{r}((\textsc{f}_1 \vee \neg\textsc{f}_2) \wedge \textsc{e}_1 \wedge \textsc{e}_2), \textsc{a}(\textsc{e} \wedge \neg\textsc{r} \wedge \neg\textsc{t} \vee \neg\textsc{isKey} \wedge \neg\textsc{e} \wedge \textsc{r} \wedge \neg\textsc{t})\}$
- $M_{\mathrm{ER}\textsc{simple}} = \{\textsc{e}(\mathit{true}), \textsc{r}((\textsc{f}_1 \vee \neg\textsc{f}_2) \wedge \textsc{e}_1 \wedge \textsc{e}_2), \textsc{a}(\neg\textsc{n} \wedge \textsc{e} \wedge \neg\textsc{r} \wedge \neg\textsc{t})\}$

In order to simplify the notation, in the following we omit the negated references in formulas; hence, the previous model descriptions would be as follows:

- $M_{\mathrm{ER}} = \{\textsc{e}(\mathit{true}), \textsc{r}((\textsc{f}_1 \vee \neg\textsc{f}_2) \wedge \textsc{e}_1 \wedge \textsc{e}_2), \textsc{a}(\textsc{e} \vee \neg\textsc{isKey} \wedge \textsc{r})\}$
- $M_{\mathrm{ER}\textsc{simple}} = \{\textsc{e}(\mathit{true}), \textsc{r}((\textsc{f}_1 \vee \neg\textsc{f}_2) \wedge \textsc{e}_1 \wedge \textsc{e}_2), \textsc{a}(\neg\textsc{n} \wedge \textsc{e})\}$

The notions of $\sqsubseteq$, $\sqcup$, $\sqcap$ and $-$ go unchanged. For example, we can check that also with this new formalism $M_{\mathrm{ER}\textsc{simple}} \sqsubseteq M_{\mathrm{ER}}$. In fact, it results that for $\textsc{e}$ and $\textsc{r}$ the condition is trivially verified, since they have the same formula associated in the two models. For $\textsc{a}$ we have to consider the complete formulas (i.e. the form with also the negated atom for references); simplifying the logical conjunction of the two formulas in the two models, we have:

$$\begin{aligned}
&\neg\textsc{n} \wedge \textsc{e} \wedge \neg\textsc{r} \wedge \neg\textsc{t} \wedge (\textsc{e} \wedge \neg\textsc{r} \wedge \neg\textsc{t} \vee \neg\textsc{isKey} \wedge \neg\textsc{e} \wedge \textsc{r} \wedge \neg\textsc{t})\\
=\ &\neg\textsc{n} \wedge \textsc{e} \wedge \neg\textsc{r} \wedge \neg\textsc{t} \wedge \textsc{e} \wedge \neg\textsc{r} \wedge \neg\textsc{t} \vee \neg\textsc{n} \wedge \textsc{e} \wedge \neg\textsc{r} \wedge \neg\textsc{t} \wedge \neg\textsc{isKey} \wedge \neg\textsc{e} \wedge \textsc{r} \wedge \neg\textsc{t}\\
=\ &\neg\textsc{n} \wedge \textsc{e} \wedge \neg\textsc{r} \wedge \neg\textsc{t}
\end{aligned}$$

Regarding the signature of Datalog rules, we redefine the description of an atom considering also the references. Given an atom $C(\textsc{args})$, the corresponding construct description $C(f)$ is computed as the conjunction of the formula obtained on the basis of properties associated with a constant (like described in Section 4) and of one literal for each reference in $\textsc{args}$. For example, the signatures of the two atoms in the body of rule $R_{1,6}$ are $\textsc{r}(\neg\textsc{f}_1 \wedge \neg\textsc{f}_2) \wedge \textsc{e}_1 \wedge \textsc{e}_2$ and $\textsc{e}(\mathit{true})$, respectively.

Then, the definitions of the three parts $(B, H, \textsc{map})$ of the signature r_R of a Datalog rule R go unchanged. We remark that references are not involved in the $\textsc{map}$ and, hence, the application of $\mu_{\textsc{map}}$ to a literal for a reference always returns *true*. Let us see again the definition on rule $R_{1,6}$ in our running example. The body is $B_{1,6} = \langle \textsc{r}(\neg\textsc{f}_1 \wedge \neg\textsc{f}_2 \wedge \textsc{e}_1 \wedge \textsc{e}_2), \textsc{e}(\mathit{true})\rangle$. The head is $H_{1,6} = \textsc{r}(\neg\textsc{o}_1 \wedge \textsc{f}_1 \wedge \textsc{i} \wedge \neg\textsc{f}_2 \wedge \textsc{e}_1 \wedge \textsc{e}_2)$. The mapping is $\textsc{map}_{1,6} = \langle \textsc{o}_2 : \textsc{r}(\textsc{o}_1)\rangle$. Also the application of the signature of a rule R to a model M goes unchanged.

As the difference in formalization is limited to the way propositions are defined, it turns out that all lemmas and theorems of Section 5 hold in this extended framework as well, with the same formulations and with proofs that are straightforward variations of those shown there.

We have recently described the features of this new version of the supermodel and commented on its experimentation [8].

8 Related work

To the best of our knowledge, there is no approach in the literature that tackles the problem we are considering here. There are pieces of work that consider the translation of schemas in heterogeneous frameworks [5,9,12,22], but none has techniques for inferring high level descriptions of translations from their specification. They all propose some way of generating a complex translation plan but they either handle very simple descriptions of models or have to rely on a hard coding of knowledge of behaviour of transformations in terms of pattern of constructs removed and introduced. [12,22] both use some form of signature to implement an A* algorithm that produces the shortest transformation plan (if it exists) between the source and the target model in terms of number of transformations.

This paper complements our previous piece of work [7], as here we provide the complete development for a result which is indeed used in such a paper (Section 5.3 of it summarize the notions of the present paper, in order to use them).

Translations of schemas by means of Datalog variants have been proposed by various authors [1,13,15], but no explicit reference to models and to the possibility of reasoning on models has been proposed. The latter work includes some reasoning on constraints, but without reference to the features of models.

Various pieces of work exist on the correctness of transformations of schemas, with reference to the well known notion of information capacity dominance and equivalence [2,16,21]. Here we are not studying the correctness of the individual translation steps, but the correctness of complex translations, assumed that the elementary steps are correct, following an “axiomatic” approach [9].

9 Conclusions

We have given the definition of a formal system to infer the model $r_{\mathbf{P}}(M)$ obtained out of a model M by applying the signature $r_{\mathbf{P}}$ of a Datalog program $\mathbf{P}$, and have shown that the derivation is sound and complete: the application of $\mathbf{P}$ to a schema S that belongs to M produces only schemas that belong to $r_{\mathbf{P}}(M)$ and potentially all of them. The techniques developed here are the basis for a formal support to a tool for the automatic generation of translations, because signatures can be obtained directly out of programs. Such a tool was originally developed for the *offline* translation of schemas and data, where both are imported in the tool from a source environment and then, after the translation, exported in a different target one [6,7]. More recently, we have used these techniques also in a tool that supports a *runtime* approach, where data is not moved but views are built to execute the needed transformations in the native environment [3].

References

1. Abiteboul, S., Cluet, S., Milo, T.: Correspondence and translation for heterogeneous data. Theor. Comput. Sci. **275**(1-2), 179–213 (2002)
2. Abiteboul, S., Hull, R.: Restructuring hierarchical database objects. Theor. Comput. Sci. **62**(1-2), 3–38 (1988)
3. Atzeni, P., Bellomarini, L., Bugiotti, F., Gianforme, G.: A runtime approach to model-independent schema and data translation. In: EDBT Conference, ACM, pp. 275–286 (2009)

4. Atzeni, P., Cappellari, P., Bernstein, P.A.: A multilevel dictionary for model management. In: ER Conference, LNCS 3716, pp. 160–175. Springer (2005)
5. Atzeni, P., Cappellari, P., Bernstein, P.A.: Model-independent schema and data translation. In: EDBT Conference, LNCS 3896, pp. 368–385. Springer (2006)
6. Atzeni, P., Cappellari, P., Gianforme, G.: MIDST: model independent schema and data translation. In: SIGMOD Conference, pp. 1134–1136. ACM (2007)
7. Atzeni, P., Cappellari, P., Torlone, R., Bernstein, P.A., Gianforme, G.: Model-independent schema translation. VLDB J. **17**(6), 1347–1370 (2008)
8. Atzeni, P., Gianforme, G., Toti, D.: Polymorphism in datalog and inheritance in a metamodel. In: FOIKS Symposium, LNCS 5956, pp. 114–132. Springer (2010)
9. Atzeni, P., Torlone, R.: Management of multiple models in an extensible database design tool. In: EDBT Conference, LNCS 1057, pp. 79–95. Springer (1996)
10. Batini, C., Ceri, S., Navathe, S.: Database Design with the Entity-Relationship Model. Benjamin and Cummings Publ. Co., Menlo Park, California (1992)
11. Bernstein, P.A.: Applying model management to classical meta data problems. In: CIDR Conference, pp. 209–220 (2003)
12. Bernstein, P.A., Melnik, S., Mork, P.: Interactive schema translation with instance-level mappings. In: VLDB, pp. 1283–1286 (2005)
13. Bowers, S., Delcambre, L.M.L.: The Uni-Level Description: A uniform framework for representing information in multiple data models. In: ER Conference, LNCS 2813, pp. 45–58. Springer (2003)
14. Cosmadakis, S., Kanellakis, P.: Functional and inclusion dependencies - a graph theoretical approach. In: P. Kanellakis, F. Preparata (eds.) Advances in Computing Research, Vol.3, pp. 163–184. JAI Press (1986)
15. Davidson, S.B., Kosky, A.: Wol: A language for database transformations and constraints. In: ICDE, pp. 55–65 (1997)
16. Hull, R.: Relative information capacity of simple relational schemata. SIAM J. Comput. **15**(3), 856–886 (1986)
17. Hull, R., King, R.: Semantic database modelling: Survey, applications and research issues. ACM Computing Surveys **19**(3), 201–260 (1987)
18. Hull, R., Yoshikawa, M.: ILOG: Declarative creation and manipulation of object identifiers. In: Sixteenth International Conference on Very Large Data Bases, Brisbane (VLDB'90), pp. 455–468 (1990)
19. Markowitz, V.M., Shoshani, A.: On the correctness of representing extended entity-relationship structures in the relational model. In: SIGMOD, pp. 430–439 (1989)
20. McGee, W.C.: A contribution to the study of data equivalence. In: IFIP Working Conference Data Base Management, pp. 123–148 (1974)
21. Miller, R.J., Ioannidis, Y.E., Ramakrishnan, R.: The use of information capacity in schema integration and translation. In: VLDB, pp. 120–133 (1993)
22. Papotti, P., Torlone, R.: Heterogeneous data translation through XML conversion. J. Web Eng. **4**(3), 189–204 (2005)
23. Tarski, A.: A lattice-theorethic Fixpoint Theorem and its applications. Pacific Journal of Mathematics **5**, 285–309 (1955)
24. Ullman, J.D., Widom, J.: A First Course in Database Systems. Prentice-Hall, Englewood Cliffs, New Jersey (1997)