Reachability Problems in Entity-Relationship Schema Instances

Sebastiano Vigna

Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano
vigna@acm.org

Abstract. Recent developments in reification of ER schemata include automatic generation of web-based database administration systems [1,2]. These systems enforce the schema cardinality constraints, but, beyond unsatisfiable schemata, this feature may create *unreachable instances*. We prove sound and complete characterisations of schemata whose instances satisfy suitable reachability properties; these theorems translate into linear algorithms that can be used to prevent the administrator from reifying schemata with unreachable instances.

1 Introduction

Web-based database administration systems are becoming extremely popular as a mean to access database information without the need to deploy a specific, architecture-bound client. In particular, recent research focuses on systems that are automatically generated from a specification in the language of entity-relationship (ER) schemata[1] [1,2]. The administrator specifies a schema in a suitable language, and the system generates an SQL implementation and an application running server side that presents the user with suitable forms for editing the database.

An important feature of ER schemata is the possibility of specifying *cardinality constraints*, that is, constraints on the number of relationships involving a certain entity. They are particularly important in the case of content management (i.e., database administration for content to be published on the web), because relying on the constraints leads to simpler HTML generation (e.g., special cases or missing relationships do not need to be checked). For this reason, ER-based database administration systems enforce cardinality constraints: when modifying the database, the user is prevented from moving the database form a legal configuration to an illegal one.

Cardinality constraints, however, cannot be imposed lightly, as they can easily lead to problems when the schema is poorly designed. The first systematic study of this issues can be found in a seminal paper by Lenzerini and Nobili [5], which gave necessary and sufficient conditions for a schema to have an instance at all (a condition called *satisfiability*), or to have non-empty instances (*strong satisfiability*). This is the first, necessary step to ensure that the schema definition has any sense. Later, the results were further extended to schemata with ISA [6], albeit the extension did not take into consideration type features, such as disjunctive subtyping.

Even if a schema has non-empty instances, however, there is another orthogonal problem: modifications to the schema instance (i.e., to a database) are usually constrained (e.g., a user can modify just a subset of the instance in a transaction), and thus the existing instances may be "too far" to make the user actually able to modify one into another.

A typical case is one in which we have entity types E and F and mandatory relationship types R from E to F and S from F to E. In this case, there is no way to move from the empty instance (without any entity or relationship) to any nonempty instance *using only local modifications*, that is, inserting, deleting or updating at most one entity and its relationships at a time (as indeed happens in web-based content management). In other word, the schema happily passes all *static* satisfiability conditions, but it is in practice unusable.

[1] ER schemata are a popular conceptual model originally introduced by Chen [3], and later extended in several ways [4].

The purpose of this paper is to present sound and complete algorithms that statically check an ER schema for instance reachability conditions. In particular, we present an algorithm guaranteeing that all instances are mutually reachable, and an algorithm guaranteeing that all *everywhere nonempty* instances are mutually reachable. The second case is useful when a pre-populated database needs very stringent constraints that do not pass the first check (as it happens in the previous example). These results complements the static satisfiability aforementioned results: all instances of a schema *with no instance at all*, for instance, are by definition mutually reachable, so satisfiability tests are necessary, but, as shown in our previous examples, instances of fully satisfiable schemata may be mutually unreachable.

To make our reasoning precise, we need a completely formal definition and semantics of ER schemata that refers only to sets, elements and functions (as proposed, for instance, in [7]); as a by-product, our results are valid independently of the kind of DBMS that is used to implement the schema semantics.[2]

The main motivation for this work is ERW (`http://erw.dsi.unimi.it/`), a reification tool developed by the author that creates a complete web-based database administration system starting from a schema definition. ERW supports sophisticated features such as weak entities, multiple inheritance, user authentication and authorisation, etc. It has been in use for the last two years to manage the web site of the Computer Science Department of the Università degli Studi di Milano; recently, other departments and universities in Italy have refactored their databases to use ERW. Since its public release in February 2002 there have been more than 10 000 accesses to the ERW home page.

ERW is based on the abstract semantics described in this paper. In particular, before creating the actual database administration system it performs a number of checks on the validity of the schema. One of the algorithms solves the problem of *double ownership*, that is, the problem of identifying weak entities, and has been described in [2]. In this paper, we describe the techniques by which we prevent administrators from creating databases that can lock the user in unmodifiable configurations.

No published solution to these problems is known to the author. To be true, only a small part of the work on conceptual modeling is formalised mathematically, and this makes it difficult, if not impossible, to develop algorithms and prove their correctness. Moreover, the problem of reachability was born with stateless web interactions: the usual notion of database transaction is stateful, so reachability is not a problem in that case.

Note that general techniques for satisfiability problems in first-order (but possibly more general than entity-relationship schemata) theories are useless here, because we the problem we study is inherently of higher order—we want to prove statements about the *models* of the theory.

First, we introduce briefly the semantics for schemata with binary relationship types that we proposed in [2]; it extends the original one given by Chen [3], but adds typing and inheritance (the original ER proposal did not include explicit subtyping information). Moreover, it introduces *multirelations* as the abstract semantics of relationship types (with respect to the formulation given in [2], we add a new notion, that of *abstract entity types*, motivated partly by ontological reasons and partly by the need of representing complete disjunctive types).

Once the semantics is set up, we introduce a notion of *isomorphism of instances* that allows us to define precisely when two schema instances are equivalent modulo the particular elements used in representing them, and of *local modification*, which models stateless interaction with a web client.

At this point, we discuss two notions of reachability and characterise them mathematically in a sound and complete way. From the characterisation it is immediate to derive linear check algorithms. However, the characterisation is given for schemata without `ISA` relationship types (i.e., without subtyping). Thus, we conclude discussing some techniques that should be applied when subtyping is present. We also discuss extensions to n-ary relationship types.

A Java implementation of the algorithms described in this paper and in [2] is available as free software by the author at the URI above.

[2] In general, ER-based management systems should be based on a formal, abstract semantics of schema instances and transactions so that different database models (i.e., relational or object-oriented) and different DBMS can be used to implement the abstract semantics.

2 Schemata

We note first that we do not need to introduce attributes in our schema definition. Since we have to discuss just cardinality issues, it is sufficient to be able to speak of sets and elements (one can of course add an attribute map for every set involved and easily discuss keys, attribute constraints and so on). Moreover, for simplicity we present the algorithm for binary relationship types, and discuss at the end of the paper the (easy) extension for types of arbitrary arity.

An EER schema (of binary relations) $\mathscr{S}$ is given by a set $\mathscr{E}$ of entity types, a set $\mathscr{R}$ of relationship types, a source function $s : \mathscr{R} \to \mathscr{E}$ and a target function $t : \mathscr{R} \to \mathscr{E}$ (note that in order to give a formal semantics to an EER schema without roles you need to consider relationship types as directed). An entity type may be *abstract*. Moreover, each relationship type has a source and a target *cardinality constraint*, which is a symbol out of `(0:1)`, `(1:1)`, `(0:N)`, `(1:N)`, `(0:M)`, `(1:M)`. The ordered pair of cardinality constraint of a relationship type is usually written as `(-:-)`$\to$`(-:-)`. Finally, a relationship type may be marked optionally either `ISA`, in which case its constraint must be `(1:1)`$\to$`(0:1)`[3].

Whenever there is an `ISA` relationship type from E to F, E is said to be a *direct subtype* of F, and F a *direct supertype* of E. A *subtype* of E is either E or a direct subtype of a subtype of E (analogously for supertypes).

Note that there are two new cardinality types: `(0:M)` and `(1:M)`. The `M` indicates that we are actually requiring a multirelation.

The notion of abstract entity type is a new extension with respect to the definition presented in [2], and it is borrowed from object-oriented languages (notably Java). The idea is that it should be used for types which are necessary for a correct structuring of the type hierarchy, but that are "universals" (in the ontological sense) and thus have no instance themselves—they can be just instantiated through their subtypes. We shall see that this extension allows one to implement some additional type constructs (at the price of a more complex interaction between types and constraints).

3 Instances

To define a schema instance, one introduces multirelations in the spirit of bicategory theory [8,9]:

Definition 1. *A (binary) multirelation from set X to set Y is a set M endowed with two functions, the* left leg *M_0 and the* right leg *M_1:*

$$\begin{tikzcd} & M \arrow[dl, "M_0"'] \arrow[dr, "M_1"] & \\ X & & Y \end{tikzcd}$$

Two elements $x \in X$ and $y \in Y$ are related if there is an $r \in M$ such that $M_0(r) = x$ and $M_1(r) = y$, a condition that we write in short with the notation $r(x, y)$. The reader should notice that it can happen that there are two elements $r, s \in M$ such that $r(x, y)$ and $s(x, y)$. In this case, the elements x and y are related "more than once". If this never happens, then M is a standard relation represented in tabular form (if you start from a relation as a subset of $X \times Y$ the two legs are just the projections).

Definition 2. *An instance σ for a schema $\mathscr{S}$ is given by a map σ assigning to each entity type E in $\mathscr{E}$ a set $\sigma(E)$ and to each relationship type R in $\mathscr{R}$ a multirelation $\sigma(R)$ satisfying the following properties:*

[3] For completeness, we should introduce the `WEAK` labelling for arcs that represent identification functions, but this has no effect on the present discussion.

1. *The left leg of $\sigma(R)$ must end in $\sigma(s(E))$, and the right leg in $\sigma(t(E))$; in other words, if R is a relationship type from entity type E to entity type F, then $\sigma(R)$ must be a multirelation*

$$\begin{tikzcd}
 & \sigma(R) \arrow[dl] \arrow[dr] & \\
\sigma(E) & & \sigma(F)
\end{tikzcd}$$

   that is, a multirelation with a left leg ending in $\sigma(E)$ and a right leg ending in $\sigma(F)$.
2. *A cardinality constraint of the form* `(1:-)` *requires that the corresponding leg be a surjective function.*
3. *A cardinality constraint of the form* `(-:1)` *requires that the corresponding leg be an injective function.*
4. *A cardinality constraint of the form* `(-:N)` *requires that the multirelation is actually a relation (i.e., nothing is related twice).*
5. *Whenever a relationship type is marked* `ISA`*, the first leg is the identity and the second leg is an inclusion map, so that that source entity set is a subset of the target entity set.*
6. *Whenever entity types E and F have a common supertype and $x \in \sigma(E) \cap \sigma(F)$, there is a common subtype G of E and F such that $x \in \sigma(G)$*[4]*.*
7. *If E is abstract, $\sigma(E)$ is exactly the union of $\sigma(F)$ when F ranges through the proper subtypes of E.*

The wording of cardinality constraints may seem a bit unorthodox: however, it is easy to see that it is exactly equivalent to the standard *participation interpretation* of constraints, and moreover extends immediately to n-ary relationship types.

Condition (6) is important as it forces *definite typing*. Essentially, every entity must have a type: if, for instance, `man` and `woman` are subtypes of `person`, it is not possible that x belongs to $\sigma(\texttt{man})$ *and* $\sigma(\texttt{woman})$ (unless you add a common subtype `hermaphrodite` of both `man` and `woman`).

An *entity* is now a pair (E, x) such that E is the type of x. Note that we did not restrain entity sets to be disjoint, so an x whose type is E and an x whose type is F are actually *distinct entities*[5].

Finally, condition (7) gives a precise semantics to abstract entity types: their instances are all subtype instances. In particular, an abstract entity type without subtypes cannot have instances.

Abstract types are useful as they allow one to model *complete disjunctive subtyping*. Getting back to the example above, by making `person` abstract we force every person to be either a man or a woman.

4 Local Modifications

To formalise reachability problems, we introduce *local modifications*, which parallel the action of a user interacting with a database by means of a web browser (and thus, essentially, statelessly).

Definition 3. *A* local modification *of an instance σ of the schema $\mathscr{S}$ is one of the following operations:*

1. *adding an entity e of type E, together with relationships involving e and another entity;*
2. *adding a relationship;*
3. *deleting an entity e of type E and all relationships involving e;*
4. *deleting a relationship.*

If τ is derived by σ by means of a local modification, we write $\sigma \to \tau$. We write $\sigma \Rightarrow \tau$ if there is a chain

$$\sigma = \upsilon_0 \to \upsilon_1 \to \cdots \to \upsilon_n = \tau$$

of instances.

[4] The last two conditions are an elementary phrasing of a *stability* condition on suitably defined maps—see [2].

[5] This feature parallels the common usage of numerical identifiers to represent set elements in SQL databases—the identifiers are not necessarily distinct in different tables.

Note that the previous definition is targeted to the applications we have in mind: stateless interaction with a server does not allow to access data stored client-side until there is a commit (if we are interacting to create e we cannot access e as if it was already on the server). This is the reason why the notion of addition and of deletion of an entity are not symmetric.

5 Isomorphism of Instances

To discuss reachability problems, it is important to identify instances which differ only for the set-theoretical identity of their elements, but for not their relational structure, as shown in the example of Fig. 1.

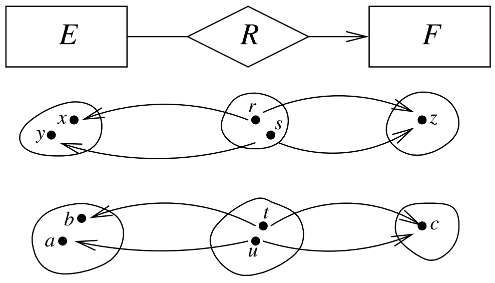

Fig. 1. Isomorphic instances.

Definition 4. *We say that instances σ and τ of a schema $\mathscr{S}$ are* isomorphic *if we have for each entity type $E \in \mathscr{E}$ and for each non-*`ISA` *relationship type $R \in \mathscr{R}$ bijections $\iota_E : \sigma(E) \to \tau(E)$ and $\iota_R : \sigma(R) \to \tau(R)$ such that*

$$r(x, y) \in \sigma(R) \iff \iota_R(r)(\iota_{s(R)}(x), \iota_{t(R)}(y)) \in \tau(R).$$

The definition above could be rephrased by saying that the entity and relationship sets must be in bijection, and the bijections must commute with the multirelations (more precisely, with their legs)[6]. In particular, we notice that every instance in which all entity and relationship sets contain at most one element is unique up to isomorphism.

6 Full Reachability

We can now formalise our problem in its most general form:

Problem 1. Given a class of instances $\mathscr{C}$ of the schema $\mathscr{S}$, is it true that for every pair of instances $\sigma, \tau \in \mathscr{C}$ we have, up to isomorphism, $\sigma \Rightarrow \tau$?

This question is important: if, for instance, there is no path out of the empty instance, then it is impossible to populate the instance by means of local modifications, that is, using a web-based database administration system. As we discussed in the introduction, if we have two entity types E and F, and two relationship types R from E to F and S from F to E, and both R and S have constraints `(1:N)`$\to$`(0:N)`, then no local modification is possible on the empty instance (as adding an entity to E or F would violate a cardinality constraint). However, in this case, it is easy to see that *all nonempty instances are mutually reachable*.

[6] The notion of instance can be easily seen to be a special case of the notion of *pseudo-functor* between bicategories [8,10]; the notion of isomorphism above is just an elementary restatement of the definition of functor isomorphism.

The situation is very different if we instead assume `(1:1)→(1:N)` as constraint of R and S. In this case, *all instances are mutually unreachable*. This happens because every instance contains a cycle of *surjective functions*, and a trivial counting argument shows that *all entity sets along such a cycle must have the same cardinality* (assuming, of course, the entity sets are finite). This forbids *any* local modification. Clearly we want to avoid this case.

To give our results, we set up a few terms that will be useful: a schema is said to be *flat* if it contains neither `ISA` relationship types nor abstract entity types; an *endorelationship* of an entity e is a relationship of the form $r(e, e)$.

Definition 5. *Given a flat schema $\mathscr{S}$, the graph $\Gamma(\mathscr{S})$ is defined as follows: the set of nodes of $\Gamma(\mathscr{S})$ is the set of entity types $\mathscr{E}$, and there is an arc from E to F whenever there is a relationship type with constraint of the form $(- : -) \to (\mathtt{1} : -)$ from E to F or a relationship type with constraint of the form $(\mathtt{1} : -) \to (- : -)$ from F to E.*

The graph $\Gamma(\mathscr{S})$ embodies the mandatoriness constraints in a form that makes it easy to check whether they are contradictory:

Theorem 1. *Let $\mathscr{S}$ be a flat schema. Then all instances of $\mathscr{S}$ are mutually reachable if and only if $\Gamma(\mathscr{S})$ is acyclic (i.e., it contains no cycles).*

Proof. We start by proving the right-to-left implication. It is sufficient to show that from an instance σ we can reach the empty instance by means of reversible local modifications. To do so, it is sufficient to show that we can always reversibly delete an entity from any given instance.

The only local modification that is not reversible in general is deletion of an entity e that has endorelationships. However, since by hypothesis the type of those endorelationships cannot be mandatory (or there would be a loop), we can obtain the same result by first deleting all endorelationships, and then finally deleting e (the last step is reversible, as all relationships at this point involve some other entity).

Consider now a nonempty instance σ, and an entity e of type E of σ. Then, either e can be deleted without breaking any cardinality constraint, or there must be a relationship type, say from E to $F \neq E$ with constraint $(- : -) \to (\mathtt{1} : -)$ satisfied by e (the dual case of a relationship type from F to E with constraint $(\mathtt{1} : -) \to (- : -)$ can be treated analogously). Thus, there must be an arc of $\Gamma(\mathscr{S})$ from E to F, and an entity f of type F which is in relation with e. We can iterate this procedure on f, but since $\Gamma(\mathscr{S})$ contains no cycle after a finite number of steps we must get to an entity that can be deleted.

For the other implication, consider a cycle in $\Gamma(\mathscr{S})$, to which we can associate a sequence E_0, R_0, E_1, $R_1, \ldots, R_n$, E_0 of mandatory relationship types[7]. If $n > 0$, given an instance σ such that $\sigma(E_i) = \varnothing$ for all i, there is no local modification that can change this condition, as adding an entity of any of the $\sigma(E_i)$'s would imply adding at the same time other n entities, which is not allowed by the definition of local modification. If $n = 0$, adding an entity would imply adding at the same time an endorelationship or other entities, which again is not allowed by the definition of local modification.

7 Almost Full Reachability

There are situations in which the previous check is too strict (for instance, if we start from a pre-populated database, which is never meant to be empty). A reasonable relaxing of full reachability is requiring that all instances in which all entity sets are nonempty are mutually reachable. We give a formal definition of this fact:

Definition 6. *A schema instance σ is* everywhere nonempty *if for every $E \in \mathscr{E}$ we have $\sigma(E) \neq \varnothing$, that is, if all its entity sets are nonempty.*

[7] To simplify the proof, we assume without loss of generality that all mandatory types involved are "in the right direction".

In this case, we have to look for *injective mandatory relations*. A relationship that is injective and mandatory, that is, such that every entity is in relation with some other entity and such that two entities are never related to the same entity, imposes a cardinality bound on the size of the source and target entity set (the first one must be smaller than or equal to the second one). If we get a cycle of such constraints, we may be in trouble (the fact that cycles of injective relations can be substituted with cycles of bijections without changing the allowable instances was already noticed in [4]).

Again, we build a graph that embodies the combinatorial bounds imposed by cardinality constraints:

Definition 7. *Given a flat schema $\mathscr{S}$, we define the graph $A(\mathscr{S})$ as follows: the set of nodes of $A(\mathscr{S})$ is the set of entity types $\mathscr{E}$, and there is an arc from E to F whenever there is a relationship type with constraint of the form $(- : \mathtt{1}) \to (\mathtt{1} : -)$ from E to F or a relationship type with constraint of the form $(\mathtt{1} : -) \to (- : \mathtt{1})$ from F to E.*

Theorem 2. *Let $\mathscr{S}$ be a flat schema. Then all everywhere nonempty instances of $\mathscr{S}$ are mutually reachable if and only if $A(\mathscr{S})$ contains no cycles.*

Proof. Analogously to the proof of Theorem 1, for the right-to-left implication we show that from an instance σ we can reach a fixed instance (up to isomorphism) having exactly one entity per entity type and one relationship per relationship type. Such an instance certainly satisfies all cardinality constraints (as all multirelations are bijections).

We notice again that all entity deletions performed in the proof will be reversible. Indeed, if we have to delete $e \in \sigma(E)$, with $|\sigma(E)| > 1$, and e is involved in endorelationships, we can certainly either delete them (because their type is not mandatory), or modify them so to relate to some other element of the same type of e (because they are not injective); otherwise, the types of those relationships would generate a loop in $A(\mathscr{S})$.

Consider now an everywhere nonempty instance σ, an entity set $\sigma(E)$ with more than one element and an entity e of type E. If e can be deleted without breaking any cardinality constraint, we are done. If e cannot be deleted, this must happen because it is the only entity associated to one or more other entities $f_0, f_1, \ldots, f_n$ of type $F_0, F_1, \ldots, F_n$ and there are mandatory relationship types $R_0, R_1, \ldots, R_n$ from each of $F_0, F_1, \ldots, F_n$ to E, satisfied by association with e.

However, if these relationship types are not injective (i.e., if their cardinality constraints are *not* of the form $(- : -) \to (- : \mathtt{1})$), then we can first make a finite number of modifications associating $f_0, f_1, \ldots, f_n$ to a different element of E, and then delete e. Otherwise, we take an element f_i of type F_i, where R_i has a constraint of the form $(\mathtt{1} : -) \to (- : \mathtt{1})$, and iterate the above operations (note that $\sigma(F_i)$ must contain more than one element, or we could delete e without breaking any constraint of R_i). In doing so, we have followed an arc of $A(\mathscr{S})$, so we can iterate a finite number of times. At the last step, we obtain an entity that can be deleted.

For the other implication, consider a cycle in $A(\mathscr{S})$, to which we can associate a sequence E_0, R_0, E_1, R_1, $\ldots$, R_n, E_0 of mandatory injective relationship types. If $n > 0$, given an instance σ such that $|\sigma(E_i)| = k > 0$ for all i, there is no local modification that can change this condition, as adding an entity to any of the $\sigma(E_i)$'s would imply either adding a relationship from e to itself (if $n = 0$) or otherwise adding at the same time other $n > 0$ entities. Indeed, e must be associated in R_0 to some other entity, but we cannot use any other previously existing entities, or we would violate injectivity (since all $\sigma(E_i)$'s have the same cardinalities, the $\sigma(R_i)$'s are all bijections).

8 Arbitrary Arity

In general, one would like to handle relationship types of arbitrary arity. To this purpose we slightly (but naturally) extend the definition of schema and instances.

Definition 8. *A (n-ary) multirelation between the sets X_0, $X_1, \ldots, X_{n-1}$ is a set M endowed with n functions, the* legs, M_0, $M_1, \ldots, M_{n-1}$

$$\begin{tikzcd}
 & & M \arrow[dll, "M_0"'] \arrow[dl, "M_1"] \arrow[d, "M_{n-2}"] \arrow[dr] \arrow[drr, "M_{n-1}"] & & \\
X_0 & X_1 & \cdots & X_{n-2} & X_{n-1}
\end{tikzcd}$$

We can now easily extend the definition of schema by allowing n-ary relationship types in the obvious way, and by giving their semantics using n-ary multirelations.

Definition 9. *An n-ary schema is given by a set of entity types $\mathscr{E}$ and a set of relationship types $\mathscr{R}$. Each type $R \in \mathscr{R}$ is endowed with a list E_0, $E_1, \ldots, E_{n-1}$ of $n \geq 2$* component types *and a list of n cardinality constraints. In this case, we say that R is n-ary, or of arity n.*

Extending corresponding the definition of instances is now trivial. Note that the definition of cardinality constraints remains valid also in this case.

To extend our results to n-ary schemata, all we need to do is to rephrase correctly the construction of the graphs $\Gamma(\mathscr{S})$ and $A(\mathscr{S})$. At that point it is not difficult to show that Theorem 1 and Theorem 6 remain true.

Theorem 3. *Given an n-ary schema $\mathscr{S}$, we define the graph $\Gamma(\mathscr{S})$ as follows: the set of nodes of $A(\mathscr{S})$ is the set of entity types $\mathscr{E}$, and there is an arc from F to G whenever there is a relationship type R with component types E_0, $E_1, \ldots, E_{n-1}$, indices $i \neq j$ such that $E_i = F$, $E_j = G$, and the j-th constraint is of the form `(1:-)`. Then, all instances of $\mathscr{S}$ are mutually reachable if and only if $\Gamma(\mathscr{S})$ is acyclic.*

The construction above puts an arc in $A(\mathscr{S})$ from F to G whenever F and G are connected by a relationship type in such a way that to each entity of type G we must associate an entity of type F. In particular, this means that you cannot have a ternary relationship type that is mandatory on *two* component types (as you would create a cycle of length two).

Note also the careful wording of the definition: an apparently similar definition claiming that we should insert an arc whenever there is a relationship type R whose component type include F and G, where the constraint of G is the form `(1:-)`, would erroneously include the case that F is equal to G *and in the same position in the component list*. This of course is wrong, as we would add an an arc from E to E for a relationship type with list of component types E, F which is mandatory in E (the only correct one would be from F to E). The position of a type in the component list act as a *role*; roles are usually specified using identifiers, but from a mathematical viewpoint it is much more manageable to use the position in the component list (this is also the approach taken in [4]).

Analogously, we can give a suitable redefinition of $A(\mathscr{S})$:

Theorem 4. *Given an n-ary schema $\mathscr{S}$, we define the graph $A(\mathscr{S})$ as follows: the set of nodes of $A(\mathscr{S})$ is the set of entity types $\mathscr{E}$, and there is an arc from F to G whenever there is a relationship type R with component types E_0, $E_1, \ldots, E_{n-1}$, indices $i \neq j$ such that $E_i = F$, $E_j = G$, the i-th constraint is of the form `(-:1)`, and the j-th constraint is of the form `(1:-)`. Then, all everywhere nonempty instances of $\mathscr{S}$ are mutually reachable if and only if $A(\mathscr{S})$ is acyclic.*

In this case, we put from F to G whenever F and G are connected by a relationship type in such a way that to each entity of type G we must associate a *distinct entity of type F*.

9 Considerations on Subtypes

In the previous sections valid and complete checks for full and almost full reachability were provided. Turning these checks into algorithms is of course trivial, as cycle detection is linear. However, the theorem were given for schemata *without subtyping*. When subtyping is taken into consideration, the combinatorics of the problem becomes much more entangled, as we will try to explain using a few examples, and obtaining sound and complete results becomes much more difficult.

First of all, *there is no canonical everywhere nonempty instance*. The heart of the proof of Theorem 6 is that we have a unique (up to isomorphism) simple instance to work with. If subtypes are present, this is no longer true.

Consider the schema shown in Fig. 2. If there are k entities in D, there must be at least k entities in both A and B. But this means that there must be at least $2k$ entities in C (because of definite typing) and thus at least $2k$ elements in D (because of T). This set of constraints reduces to $k \leq 0$, and thus it is satisfied *by the empty instance only*.

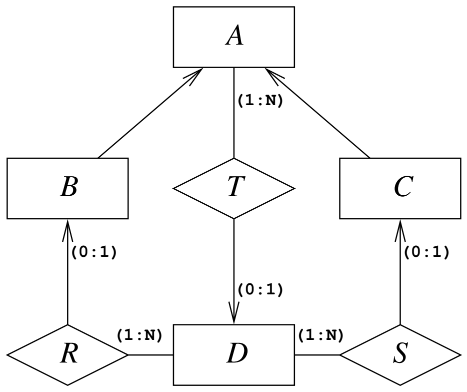

Fig. 2. A diagram without nonempty instances.

If we eliminate T, things get better, but it is easy to see that there is no everywhere nonempty canonical instance, as if we put one entity in B and C we are forced to put two in A; more complex type hierarchies may create extremely entangled combinatorial constraints.

One could think that at least the left-to-right implications of Theorems 1 and 6 should continue to hold (of course, all ISA relationship types will end up in both $\Gamma(\mathscr{S})$ and $A(\mathscr{S})$). However, there are certainly situations in which this is not true. Consider the diagram in Fig. 3, where dashed entities are abstract. It would pass all of our checks. Nonetheless, an instance in which every type has exactly one element (i.e., $\sigma(A) = \sigma(B) = \sigma(C) = \{x\}$, $\sigma(E) = \sigma(F) = \sigma(G) = \{y\}$, $\sigma(T) = \{t(y, x)\}$ and $\sigma(S) = \{s(x, y)\}$) has no legal modification.

The point is that *relationship types are inherited*. Since C is a subtype of A, also elements of type C may participate to a relationship of type T. Moreover, since A is abstract, each entity of type A must also be of type C. All in all, we get a cycle analogous to the ones used in the proof of Theorem 6.

Abstractness here plays a fundamental rôle: should not A be abstract, we could add an entity z to A, change the relationship $t(y, x)$ to $t(y, z)$, delete x and its adjacent relationship $s(x, y)$, delete y and its adjacent relationship $t(y, z)$, and finally z, getting to the empty instance.

Indeed, cycles of this kind can be built only using *forced subtypes* (E is a *forced subtype* of F if every entity of type F is also of type E). For a graph-theoretical viewpoint, an entity type E is a forced subtype of F if and only if E is the only minimal (i.e., without proper subtypes) subtype of F and all sequences of ISA relationship types going from E to F traverse abstract entity types only (including F, but except possibly for E).

Forced subtyping is important for reachability because cycles of mandatory (and possibly injective) *inherited* relationships can usually be broken by inserting suitable entities, as we pointed out in our last example,

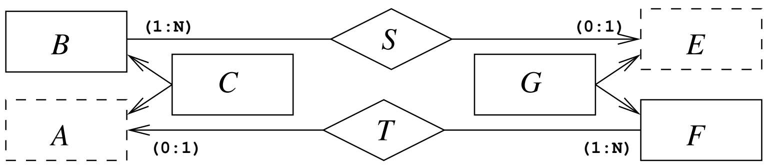

Fig. 3. An apparently innocuous diagram fragment.

but this does not happen if inheritance (on the non-mandatory side of the relationship type) is by forced subtypes.

Nonetheless, forced subtyping is a pathological condition that should not appear in a schema as much as abstract entity types without subtypes. Both pathologies can be filtered before performing the acyclicity check; under the hypothesis that no forced subtype exists, one can prove the following one-sided theorem, using the same techniques of Theorem 1 and 6:

Theorem 5. *If all instances of a schema $\mathscr{S}$ without forced subtypes are mutually reachable, then $\Gamma(\mathscr{S})$ is acyclic. If all everywhere nonempty instances are mutually reachable, then $A(\mathscr{S})$ is acyclic.*

10 Conclusions

We have presented the first sound and complete algorithms to check instance reachability in entity-relationship schemata. Since the algorithms are acyclicity tests on a graph whose size in linearly bounded by the schema size, they are linear by definition.

It would be interesting to extend these results to schemata with ISA arcs *and a type system* as described in Definition 2, or, at least, for type constructors with disjunctive types (albeit the former definition is more general). It would be more cautious, however, to start first with an extension in this direction of the results given in [5], as the problem seems to be already tough enough in that case.

References

1. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): a modeling language for designing web sites. In: Proc. Ninth World Wide Web Conference. (2000)
2. Vigna, S.: Multirelational semantics for extended entity-relationship schemata with applications. In: Conceptual Modeling—ER 2002. 21st International Conference on Conceptual Modeling. Number 2503 in Lecture Notes in Computer Science, Springer–Verlag (2002) 35–49
3. Chen, P.P.S.: The entity-relationship model: Toward a unified view of data. ACM Transactions on Database Systems **1** (1976) 9–36
4. Thalheim, B.: Entity-Relationship Modeling. Springer–Verlag (2000)
5. Lenzerini, M., Nobili, P.: On the satisfiability of dependency constraints in entity-relationship schemata. Information Systems **15** (1990) 453–461
6. Calvanese, D., Lenzerini, M.: On the interaction between ISA and cardinality constraints. In Elmagarmid, A.K., Neuhold, E., eds.: Proceedings of the 10th International Conference on Data Engineering, Houston, TX, IEEE Computer Society Press (1994) 204–213
7. Gogolla, M., Hohenstein, U.: Towards a semantic view of an extended entity-relationship model. ACM Transactions on Database Systems (TODS) **16** (1991) 369–416
8. Bénabou, J.: Introduction to bicategories. In: Reports of the Midwest Category Seminar. Number 47 in Lecture Notes in Mathematics. Springer–Verlag (1967) 1–77
9. Bruni, R., Gadducci, F.: Some algebraic laws for spans (and their connections with multirelations). In Kahl, W., Parnas, D., Schmidt, G., eds.: Relational Methods in Software. Volume 44.3 of Electronic Notes in Theoretical Computer Science., Elsevier (2001)
10. Borceux, F.: Handbook of Categorical Algebra 1. Volume 50 of Encyclopedia of Mathematics and Its Applications. Cambridge University Press (1994)