JOURNAL OF COMPUTER AND SYSTEM SCIENCES **52**, 357–373 (1996)
ARTICLE NO. 0027

On Instance-Completeness for Database Query Languages involving Object Creation

MARC ANDRIES*

Department of Mathematics and Computer Science, Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands

AND

JAN PAREDAENS†

Department of Mathematics and Computer Science, University of Antwerp (UIA), Universiteitsplein 1, B-2610 Antwerp, Belgium

Received December 16, 1991; revised April 30,1993

In 1978 Bancilhon and Paredaens introduced a notion of completeness for relational database languages on instance-level. Their criterion was subsequently called *BP-completeness*. Since then, it was used frequently in the context of other database models. However, its application in the context of languages involving object creation appears to raise some serious problems. In this paper, we use the graph-oriented object database model GOOD as a framework to propose an alternative formulation of the BP-completeness criterion, adapted to the context of languages involving object creation.

1. INTRODUCTION

Over the past few years, database research has been characterized by the development of a variety of data models. These models can be classified according to their underlying paradigm, the most well known of which are probably the relational, the deductive, and the object-oriented. In an object-oriented database, real-world entities are represented by means of objects with a unique identity. Besides, arbitrary relationships between such entities may be stored as links between the corresponding objects, thus organizing the objects in a graph-like structure [5, 8]. For most of these data-modeling paradigms, several data definition and manipulation languages have been proposed.

To demonstrate the viability of such a newly proposed language, its expressive power must be compared to that of other languages. A possible approach towards such a comparison of database languages is the use of so-called *completeness*-criteria. In [7], Codd proposed to call a language for the relational database model complete if its expressive power could be shown equivalent to that of some "standard" query language, like the relational calculus. However, in [4] Bancilhon argued that a completeness-criterion, in order to be sufficiently meaningful, should be language independent. In [4, 13] Bancilhon and Paredaens independently introduced a similar criterion, stating when a query language for flat relational databases is complete on instance-level. The criterion says that, in order to be complete, a query language should express exactly the transformations of a relation R to a relation S that satisfy the following two conditions. First, no new values may be added. Second, each domain permutation that maps R to itself, must also map S to itself. A transformation satisfying these conditions is often called a *generic* transformation. These conditions can be summarized by saying that every *automorphism* of R must be an automorphism of S. It is then shown in these articles that the relational calculus and algebra indeed express exactly these transformations, which in turn justifies Codds choice of the relational calculus as a reference language for testing completeness for relational query languages.

But what does the above criterion *intuitively* signify? The presence of a (nontrivial) automorphism for some relation R can be interpreted as follows: for every value in R, there exists another value which can "take its place" in the relation. Indeed, when each value in R is substituted by its image under the automorphism, R itself is obtained. Consequently, a value and its image under some automorphism are indistinguishable on the basis of their relationships. The criterion states that if such a resemblance exists in the input relation of a database operation, it should still exist in the output relation. Violating this is only possible by manipulating values through more than just their relationships with other values (given in the relations of the database), in other

* E-mail: andries@wi.leidenuniv.nl.
† E-mail: pareda@wins.uia.ac.be.

0022-0000/96 $18.00

words by *interpreting* them. Consequently, this criterion is really very natural and unrestrictive, since it merely prohibits interpreting values, or in other words, to perform calculations on them. This indicates that we will only be concerned with what might be called "abstract databases": of an isolated entity in the database, only its mere existence is significant.

Apart from its theoretical importance, the validity of the criterion for a certain language can also have a practical usage if it is possible to readily check it for two given instances, since this is equivalent to the existence of a transformation between them in the language under consideration.

In [6], the above criterion was named *BP-completeness*. Briefly, a language is BP-complete if it can express exactly all generic transformations. Since its introduction, the notion of genericity has been used frequently in the context of other database models. Naturally, if we want to generalize it to other formalisms besides flat relational databases, we first have to find appropriate definitions for concepts such as "derivation" and "automorphism". An example of this may be found in [9], where this has been done successfully for the nested relational database model.

However, applying the criterion without any change to languages involving object creation, seems to raise some serious problems. Before we can outline these problems, we have to make a note on the concept of automorphisms in the context of object-based languages. On one hand, automorphisms may still be looked upon as permutations of the basic elements of the database (in this case, the objects), that preserve the structure of the instance (in this case, the relationships represented *explicitly* in the instance). In the course of a transformation, however, new objects may be created, while others may be removed. Consequently, we can no longer impose an inclusion relationship on the sets of automorphisms of the input- and output-instance of the transformation. The most natural translation of such a relationship to the context of automorphism groups for object-base instances would therefore be to require the existence of a *mapping* between the respective automorphism groups of two instances, with the additional constraint that an automorphism and its image under the given mapping should coincide on the objects still in common to the input- and output-instance. This correspondence is crucial for the understanding of the remainder of this paper: database transformations for value-based data models *commuting with* permutations corresponds to database transformations for object-based data models *preserving* automorphisms.

We now come to the announced problem. In [1], the identity query language IQL (which is a language involving object creation) is introduced and shown to be very general and powerful. In the same article, however, IQL is shown unable to express exactly the class of transformations that satisfy the above condition. It is therefore our intent in this paper to investigate what modification must be made to the criterion in order to allow us to precisely characterize the set of transformations that is expressible by general languages involving object creation. This modification will be stated in terms of mappings between automorphism groups (cf. Definition 11), since it was shown above that this is the most natural way to go in an attempt to translate the concepts that play a part in the BP-completeness criterion to the context of languages involving object creation.

As a framework for our investigation, we use the graph-oriented object database model (GOOD), introduced in [10, 11], in which graph theory is used to uniformly define an object-oriented data model and data manipulation formalism. In both articles, GOOD is shown to be of significant expressive and modeling power. In [10], it is shown how GOOD can simulate arbitrary recursive functions, while in [11] it is illustrated how the most prominent aspects of object-orientation (such as inheritance of both data and methods, encapsulation, extendibility) can be incorporated in the model.[1] Since at the same time, its data model and manipulation formalism are defined using a limited number of very basic building blocks, GOOD may be regarded as a very general object-oriented database model.

This paper is organized as follows. In Section 2, we recall the aspects of GOOD that are needed in the remainder of the paper. In this section we also introduce the notion of *extension-morphism*, capturing the required modification to the BP-criterion. The main result of this paper is Theorem 8, which shows that the transformation language of GOOD indeed satisfies the adapted BP-criterion. This is stated and proven in Section 3. In Section 4, we first state a theorem that actually captures the same result as the main theorem. Then we see what happens if we release or strengthen some aspects of both theorem and model.

2. THE GRAPH-ORIENTED OBJECT DATABASE MODEL

2.1. *The Data Model*

In GOOD, an object base is conceptually represented as a directed labeled graph. Figure 1 shows a (highly simplified) instance of an object base for some parts and subparts. Of certain parts, the instance contains some structural information (represented by means of `made_of`-relationships). Besides, of some parts, the current location is also given (by means of the `location`-relationships). It can be deduced from the figure that two parts are in a box, while two other parts are on a table.

By means of this figure, we will introduce the different components with which an instance can be constructed. The

[1] Since this paper is concerned with the expressiveness of the GOOD language, the latter modeling issues will not be considered in the remainder.

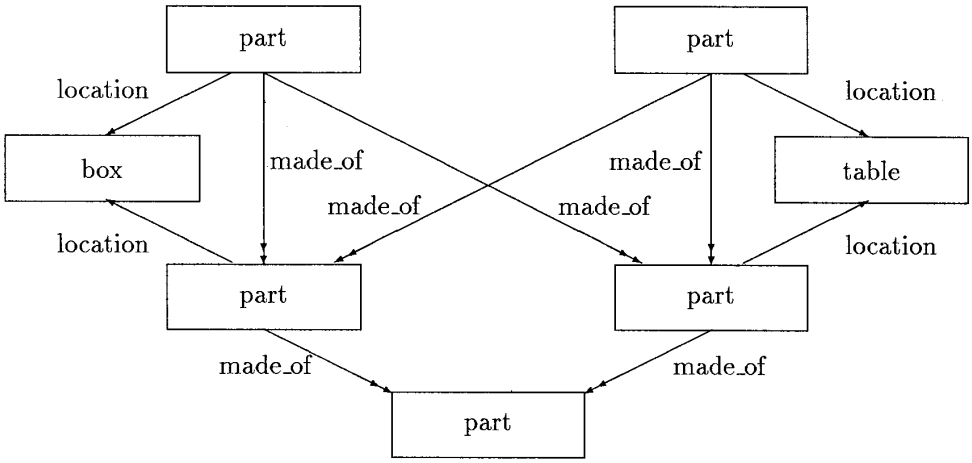

FIG. 1. An instance of an object base for parts and subparts.

nodes of the graph represent the objects of the database. Their labels indicate their class (e.g., `part`, `box`). The edges of the graph represent relationships between objects. A distinction is made between functional and nonfunctional relationships, which are represented respectively by functional edges (with a single arrowhead) and nonfunctional edges (with a double arrowhead).

Since the data model as defined in the previous paragraph is what is commonly called "pure object-based," the representation of certain information, such as an object's location, may seem somewhat awkward. In Section 4, we will show how, by a slight extension of this data model, real-world information may be modeled more directly and elegantly.

For a formal definition of object base instances, we assume the existence of an infinitely enumerable set Ω of *nodes*. Besides, we assume the existence of three infinitely enumerable and pairwise disjoint sets of labels, namely OL of *object labels*, FEL of *functional edge labels*, and NFEL of *nonfunctional edge labels*.

DEFINITION 1 (Object base instance). An object base instance $\mathscr{I}$ is a directed labeled graph (N, E) such that

- N is a finite set of labeled nodes; if $\mathbf{n}$ is a node in N, then its label is denoted by $\lambda_{\mathscr{I}}(\mathbf{n})$; in a graphical representation, the node itself is represented by a rectangle;
- E is a set of labeled edges; if $\mathbf{e}$ is a labeled edge in E, then $\mathbf{e} = (\mathbf{m}, \alpha, \mathbf{n})$ with $\mathbf{m}, \mathbf{n} \in N$ and its label $\lambda_{\mathscr{I}}(\mathbf{e}) = \alpha \in \text{FEL} \cup \text{NFEL}$; if $\lambda_{\mathscr{I}}(\mathbf{e})$ is in FEL (resp. in NFEL), then $\mathbf{e}$ is called a functional edge (resp. a nonfunctional edge);
- if $(\mathbf{m}, \alpha, \mathbf{n}_1)$ and $(\mathbf{m}, \alpha, \mathbf{n}_2) \in E$, then $\lambda_{\mathscr{I}}(\mathbf{n}_1) = \lambda_{\mathscr{I}}(\mathbf{n}_2)$; moreover, if $\alpha \in \text{FEL}$, then $\mathbf{n}_1 = \mathbf{n}_2$.

The set N is often denoted $N(\mathscr{I})$, while E is denoted $E(\mathscr{I})$.

To conclude this section, we define automorphisms in the context of these object-base instances.

DEFINITION 2 (Embedding). Let $\mathscr{I} = (N, E)$ and $\mathscr{J} = (M, F)$ be object base instances. An embedding of $\mathscr{J}$ in $\mathscr{I}$ is a total mapping $i: M \to N$:

1. $\forall \mathbf{n} \in M: \lambda_{\mathscr{J}}(i(\mathbf{n})) = \lambda_{\mathscr{J}}(\mathbf{n})$
2. $\forall \mathbf{n}, \mathbf{n}' \in M, \ \forall \alpha \in \text{FEL} \cup \text{NFEL}: \ (\mathbf{n}, \alpha, \mathbf{n}') \in F \Rightarrow (i(\mathbf{n}), \alpha, i(\mathbf{n}')) \in E$.

DEFINITION 3 (Isomorphism, automorphism). Two instances $\mathscr{I}$ and $\mathscr{I}'$ are isomorphic, if $\mathscr{I}$ can be embedded injectively in $\mathscr{I}'$ and vice versa. An injective embedding of an instance into an isomorphic instance is called an isomorphism. An automorphism of $\mathscr{I}$ is an isomorphism from $\mathscr{I}$ to itself. $\text{Aut}(\mathscr{I})$ is the group of all automorphisms of $\mathscr{I}$.

The following easy-to-verify lemma captures an important relationship between isomorphisms and embeddings.

LEMMA 1. *The composition of an embedding of an instance $\mathscr{I}'$ in an instance $\mathscr{I}$ and an automorphism of $\mathscr{I}$ is itself an embedding of $\mathscr{I}'$ in $\mathscr{I}$.*

2.2. *Data Definition and Manipulation*

In GOOD, data structures are defined and manipulated by means of a uniform language for the transformation of graphs. This language consists of five basic operations. An arbitrary sequence of such operations is called a GOOD *program*. Every operation is based on the notion of *pattern*, which describes the parts of the instance where the operation will be executed. Syntactically, patterns are identical to instances. From the context however, it will always be clear whether a graph satisfying the conditions of Definition 1 is to be considered as the contents of a database (i.e., an instance), or as the descriptive part of an operation (i.e., a pattern).

Next we discuss the basic operations of the GOOD manipulation language, which in essence allows the addition and deletion of objects and relationships. This gives us five operations: one for the addition of objects with only functional properties, one for the addition of objects with only nonfunctional properties, one for the addition of relationships, one for the deletion of objects, and one for the deletion of relationships.

Informally, each operation consists of a pattern and an associated action-part. When an operation is applied to some instance, the instance is scanned, and each time the pattern can be embedded somewhere (cf. Definition 2), the operation is executed for the objects which correspond to the embedding. As a result of Lemma 1, each time an operation is executed for a part of the instance as the result of some embedding, the operation will also be executed for all automorphic images of that embedding.

The semantics of the basic GOOD operations is, however, a bit more involved. As already mentioned, the GOOD language is object-oriented and, thus, offers the possibility of adding new objects to the database. In the foregoing section on the GOOD data model, we postulated the existence of an (infinitely enumerable) universe of nodes. If an object addition operation is executed, for each object that must be added to the object-base instance, a node is chosen at random from this universe by the system. Consequently, object addition operations are in some sense nondeterministic; their outcome is only determined up to *isomorphism*. Since this situation would lead to numerous inelegant constructions and formulations in the remainder of this paper (it would, e.g., not be possible to say that an instance is *the* result of applying a GOOD program to another instance), we slightly modify the semantics of the basic operations (in comparison to their original definition in [10]) as to make them fully deterministic. To this end, the resulting instance of any choice of nodes for newly added objects is considered to be part of the result of a primitive GOOD operation. Consequently, the result of an object addition operation will be an infinite set of instances, unless no new objects are to be added, in which case the result is a singleton containing the input-instance. The result of all other operations (i.e., deletions or additions of relationships) is obviously *always* a singleton.

We next formally define and illustrate two of the addition operations.

DEFINITION 4 (Node addition with functional edges). Let $\mathscr{I}$ be an object base instance and $\mathscr{J}$ a pattern. Let $\mathbf{m}_1, ..., \mathbf{m}_n \in N(\mathscr{J})$, $\mathtt{K} \in OL$ and $\alpha_1, ..., \alpha_n \in$ FEL. The node addition with functional edges NAF$[\mathscr{J}, \mathscr{I}, \mathtt{K}, \{(\alpha_1, \mathbf{m}_1), ..., (\alpha_n, \mathbf{m}_n)\}]$ results in the set of all possible outcomes of the nondeterministic function naf:

function naf$[\mathscr{J}, \mathscr{I}, \mathtt{K}, \{(\alpha_1 \mathbf{m}_1), ..., (\alpha_n, \mathbf{m}_n)\}]$;
$\mathscr{I}' := \mathscr{I}$;
for each *embedding i of* $\mathscr{J}$ *in* $\mathscr{I}$ **do**
  if not exists *a* K*-labeled node* $\mathbf{n}$ *in* $\mathscr{I}'$ *with outgoing edges* $(\mathbf{n}, \alpha_l, i(\mathbf{m}_l))$ $(1 \leqslant l \leqslant n)$
    then *add a new node* $\mathbf{n}' \in \Omega - N(\mathscr{I}')$ *and edges* $(\mathbf{n}', \alpha_l, i(\mathbf{m}_l))$ $(1 \leqslant l \leqslant n)$ *to* $\mathscr{I}'$;
return $(\mathscr{I}')$
end

For a proper understanding of what this *set of all possible results* of the function naf looks like, consider the following definition.

DEFINITION 5 ($\mathscr{I}$-isomorphism). Let $\mathscr{I}$ be an object-base instance. Two object-base instances $\mathscr{J}$ and $\mathscr{J}'$ are $\mathscr{I}$-isomorphic if there exists an isomorphism from $\mathscr{J}$ to $\mathscr{J}'$ that is the identity on $\mathscr{I} \cap \mathscr{J}$, and whose inverse is the identity on $\mathscr{I} \cap \mathscr{J}'$.

It can easily be verified that, given an instance $\mathscr{I}'$ resulting from an application of the non-deterministic function naf, the resulting set of the corresponding node addition with functional properties contains exactly all $\mathscr{I}$-isomorphic instances of $\mathscr{I}'$.

DEFINITION 6 (Edge addition). Let $\mathscr{I}$ be an object base instance and $\mathscr{J}$ a pattern. Let **m**, **m'** be nodes of $\mathscr{J}$ and let α be an edge label. The edge addition EA$[\mathscr{J}, \mathscr{I}, (\mathbf{m}, \alpha, \mathbf{m}')]$ results in the singleton $\{\mathscr{I}'\}$, where $\mathscr{I}'$ is defined by the following function:

function ea$[\mathscr{J}, \mathscr{I}, (\mathbf{m}, \alpha, \mathbf{m}')]$;
$\mathscr{I}' := \mathscr{I}$;
for each *embedding i of* $\mathscr{J}$ *in* $\mathscr{I}$ **do**
  if not exists *an edge* $(i(\mathbf{m}), \alpha, i(\mathbf{m}'))$ *in* $\mathscr{I}'$
    then *add such an edge* $(i(\mathbf{m}), \alpha, i(\mathbf{m}'))$ *to* $\mathscr{I}'$;
return $(\mathscr{I}')$
end

Before we can illustrate these operations, we first have to make a remark on the composition of primitive GOOD operations. Since GOOD operations are in essence set-valued functions, we cannot simply refer to the well-known semantics of function composition. Therefore we use the following formula (in which both f and g are set-valued functions) to extend this semantics in a canonical way to functions resulting in sets:

$$f \circ g(a) := \{b \mid \exists c: c \in g(a) \wedge b \in f(c)\}. \tag{1}$$

Clearly, with this semantics for composition, the result of the application of a GOOD program to an instance $\mathscr{I}$ is either a singleton or an infinitely enumerable set of $\mathscr{I}$-isomorphic instances.

As can be seen in the following figure, GOOD offers graphical representation for each operation. Uniformly,

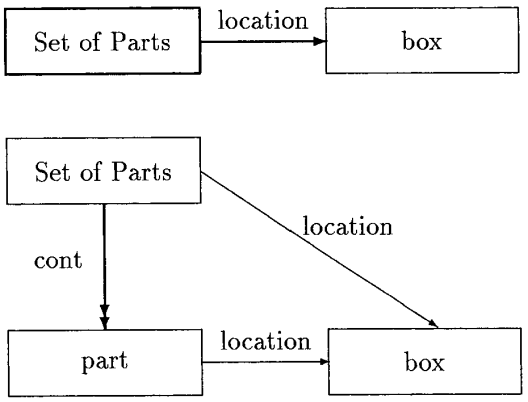

FIG. 2. Grouping the parts in a box.

patterns are indicated in plain line, and what is added is indicated in bold.

Suppose we want to group the parts in the example object base that are currently located in a box. Figure 2 shows how this can be accomplished in the manipulation language of GOOD. First we associate a new object with each box, with the node addition with functional edges NAF[$\mathscr{J}$, $\mathscr{I}$, `SetofParts`, {(`location`, **m**)}], where $\mathscr{J}$ consists of a single node **m** with label `box`. Next, we group all the objects that are in a box, in the object associated with that box by the previous operation. This is done by means of the edge addition EA[$\mathscr{J}'$, $\mathscr{I}'$, **m**, `cont`, **m**′)]. Here, $\mathscr{I}$ is the resulting instance of the previous operation, while $\mathscr{J}'$ consists of all the nodes and the two `location`-edges from the second picture of Fig. 2. The node **m** is the one with label `Set of Parts`, while **m**′ is the node with label `part`.

Now we define and illustrate the addition operation for nodes with nonfunctional edges (also called *abstraction* in [10]).

DEFINITION 7 (Node addition with non-functional edges). Let $\mathscr{I}$ be an object base instance and $\mathscr{J}$ a pattern. Let $\mathbf{n} \in N(\mathscr{J})$, $\mathrm{K} \in OL$, and $\alpha, \beta \in \mathrm{NFEL}$. Let S be the set $\{i(\mathbf{n}) \mid i\colon \mathscr{J} \to \mathscr{I}$ is an embedding$\}$, and let Σ be the partition of S, defined by the following equivalence relation:

$$\mathbf{p} \equiv \mathbf{q} \Leftrightarrow \forall \mathbf{r} \in N(\mathscr{J})\colon (\mathbf{q}, \alpha, \mathbf{r}) \in E(\mathscr{I}) \Leftrightarrow (\mathbf{p}, \alpha, \mathbf{r}) \in E(\mathscr{I}).$$

The node addition with nonfunctional edges NANF[$\mathscr{J}$, $\mathscr{I}$, **n**, K, α, β] then results in the set of all possible outcomes of the nondeterministic function nanf:

function nanf[$\mathscr{J}$, $\mathscr{I}$, **n**, K, α, β];
$\mathscr{I}' := \mathscr{I}$;
for each $T \in \Sigma$ **do**
  if $\nexists \mathbf{q} \in \mathrm{N}(\mathscr{I}')\colon \lambda_{\mathscr{I}'}(\mathbf{q}) = \mathrm{K} \wedge$
    $\{\mathbf{p} \mid (\mathbf{q}, \beta, \mathbf{p}) \in E(\mathscr{I}')\} = T$
  then *add a new node* **q**′ *and edges* (**q**′, β, **p**) ($\forall \mathbf{p} \in T$) *to* $\mathscr{I}'$;
return ($\mathscr{I}'$)
end

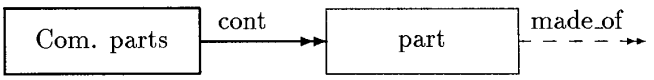

FIG. 3. Groupings parts by their subparts.

Suppose we want to group the parts in the example object base according to the parts they are made of. Figure 3 shows how this can be accomplished in GOOD. By this one operation, a new object of class `Com. parts` is added to the database for each set of parts with common subparts. In the formal notation, introduced in the above definition, this operation is NANF[$\mathscr{J}$, $\mathscr{I}$, **n**, `Com. parts`, `made_of`, `cont`], where $\mathscr{J}$ contains the single node **n** with label `part`. Note that the edge according to which parts are grouped (i.e, `made_of`) is indicated with a dashed line.

Finally we define and illustrate the deletion operations.

DEFINITION 8 (Node deletion). Let $\mathscr{I}$ be an object base instance and $\mathscr{J}$ a pattern. Let **m** be a node in $N(\mathscr{J})$. The node deletion ND[$\mathscr{J}$, $\mathscr{I}$, **m**] results in the singleton $\{\mathscr{I}'\}$, where $\mathscr{I}'$ is defined by the following function:

function nd[$\mathscr{J}$, $\mathscr{I}$, **m**];
$\mathscr{I}' := \mathscr{I}$;
for each *embedding i of* $\mathscr{J}$ *in* $\mathscr{I}$ **do**
  $N(\mathscr{I}') := N(\mathscr{I}') - \{i(\mathbf{m})\}$;
  $\Omega := \Omega - \{i(\mathbf{m})\}$,[2]
return ($\mathscr{I}'$)
end

DEFINITION 9 (Edge deletion). Let $\mathscr{I}$ be an object base instance and $\mathscr{J}$ a pattern. Let (**m**, α, **m**′) be an edge of $\mathscr{J}$. The edge deletion ED[$\mathscr{J}$, $\mathscr{I}$, (**m**, α, **m**′)] results in the singleton $\{\mathscr{I}'\}$, where $\mathscr{I}'$ is defined by the following function:

function ed[$\mathscr{J}$, $\mathscr{I}$, (**m**, α, **m**′)];
$\mathscr{I}' := \mathscr{I}$;
for each *embedding i of* $\mathscr{J}$ *in* $\mathscr{I}$ **do**
  $\mathrm{E}(\mathscr{I}') := \mathrm{E}(\mathscr{I}') - \{(i(\mathbf{m}), \alpha, i(\mathbf{m}'))\}$:
return ($\mathscr{I}'$)
end

In a graphical representation of a deletion operation, the part of the pattern corresponding to the node or edge to be deleted is indicated with double lines.

Suppose we want to remove from the example object base the parts that have no subparts. Figure 4 shows how this can be accomplished in GOOD. In the first three operations we group all those parts in a node of class `At. Parts`. First we add this node, by means of an operation with an empty pattern. Then we link *all* parts to this node, after which we

[2] By also removing a deleted node from the universe Ω of nodes, we avoid the tricky situation where one and the same node is deleted from an instance, and subsequently reinserted by a node addition. This is motivated by the fact that the actual identity of objects or nodes is of no concern when a GOOD program is executed. Hence one cannot and should not be able to make any assumptions about the identity of a newly inserted node.

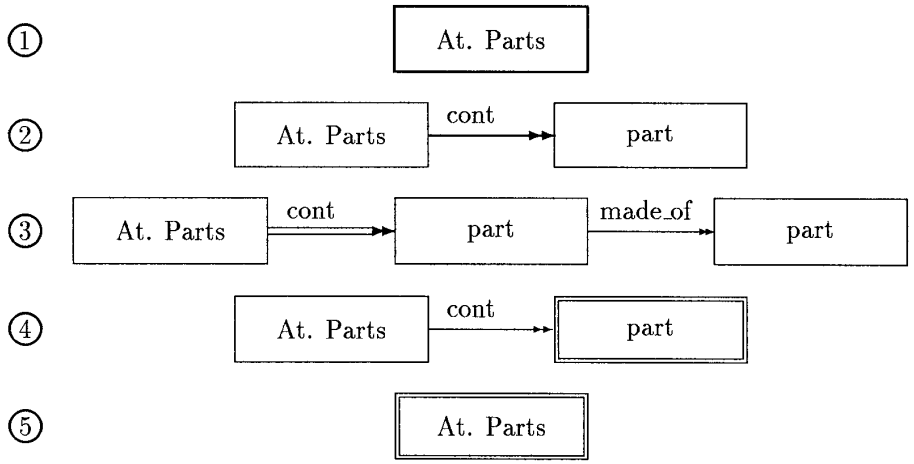

FIG. 4. Removing parts without subparts.

remove the links to those parts that *are* made of one or more parts. This is accomplished by means of an edge deletion. Finally we delete the parts that are still in the set, as well as the set itself.

To conclude this section, we introduce a notation to indicate that an instance is in the result of applying a GOOD program (i.e., a sequence of basic GOOD operations) to another instance.

DEFINITION 10 (GOOD-implication). Let $\mathscr{I}$, $\mathscr{I}'$ be object base instances. $\mathscr{I} \overset{\text{GOOD}}{\Longrightarrow} \mathscr{I}'$ indicates the existence of a GOOD program that, when applied to $\mathscr{I}$, results in a set of instances containing $\mathscr{I}'$.

3. THE GOOD LANGUAGE IS BP-COMPLETE

In this section, we first rephrase the BP-completeness criterion (cf. Section 1) in the context of languages involving object creation. Then we show that the GOOD language is BP-complete. Recall that proving completeness implies giving necessary and sufficient conditions for the existence of a GOOD program, mapping one given instance to another, or (because of the set-valued semantics we gave to the basic GOOD operations) mapping a given instance to a given *set* of instances.

First we define *extension morphisms* as the central means to capture the required modification to the BP-criterion in the context of languages involving object creation. An extension morphism is a group homomorphism between the automorphism groups of two instances, that naturally extends an automorphism of one instance to an automorphism of the other instance.

DEFINITION 11 (Extension morphism). Let $\mathscr{I}$ and $\mathscr{I}'$ be two object base instances. A group homomorphism h: $\mathrm{Aut}(\mathscr{I}) \to \mathrm{Aut}(\mathscr{I}')$:

$$\forall \mathbf{n} \in N \cap N', \forall a \in \mathrm{Aut}(\mathscr{I}): a(\mathbf{n}) = h(a)(\mathbf{n}) \qquad (2)$$

is called an extension morphism of type $(\mathscr{I}, \mathscr{I}')$.

We will often call property 2 the *extension property*. To see the significance of this name, consider the case where $\mathscr{I}$ is a subinstance of $\mathscr{I}'$. The condition then simply says that the image under h of any automorphism should coincide with that automorphism on all nodes of $\mathscr{I}$.

The step from extension morphisms to the adapted BP-completeness criterion is simple. We recall that a language is BP-complete if it can express exactly all generic transformations. Hence the following definition, in which we immediately deal with the set-valued semantics of basic GOOD operations.

DEFINITION 12 (Generic transformations). Let $\mathscr{I}$ be an instance and let Γ be a set of instances. The pair $(\mathscr{I}, \Gamma)$ is a generic transformation if the following three conditions are satisfied:

1. Every two elements of Γ are $\mathscr{I}$-isomorphic;
2. If $\mathscr{J} \in \Gamma$, and $\mathscr{J}'$ is $\mathscr{I}$-isomorphic to $\mathscr{J}$, then $\mathscr{J}'$ is also in Γ;
3. For all $\mathscr{J} \in \Gamma$, there exists an extension morphism of type $(\mathscr{I}, \mathscr{J})$.

Note that by the first condition, the last condition is equivalent to the requirement that *there exists* an instance $\mathscr{J} \in \Gamma$ for which there exists an extension morphism.

With a first proposition, we will show that *GOOD only* expresses generic transformations.

PROPOSITION 2. *If Γ is the resulting set of a GOOD program applied to an instance $\mathscr{I}$, then the pair $(\mathscr{I}, \Gamma)$ is a generic transformation.*

Proof. The first item from the definition of genericity follows from the fact that the difference between two instances in the outcome of a program can only be caused by

different choices of new nodes by the node additions in the program (cf. the explanation of the semantics of the set-valued function naf (Definition 4) and the definition of composition of primitive GOOD operations (cf. Eq. (1)).

The second item from the definition of genericity follows from the fact that the outcome of a node addition operation in a program is defined using *all* possible choices of new nodes.

The fact that the existence of a GOOD program implies the existence of an extension morphism h of the appropriate type is proved by induction on the number of operations in the given program.

Let us first assume that the given GOOD program consists of zero operations, so Γ equals $\{\mathscr{I}\}$. Naturally, for all instances in Γ there exists an appropriate extension morphism, namely the identity function on $\mathrm{Aut}(\mathscr{I})$.

Second, the induction hypothesis is as follows: for each pair of instances $(\mathscr{I}, \mathscr{I}')$ such that $\mathscr{I}'$ is in the outcome of the application of a GOOD program consisting of at most l basic GOOD operations to the instance $\mathscr{I}$, there exists an extension morphism of type $(\mathscr{I}, \mathscr{I}')$.

Suppose we apply a GOOD program ∇ of $l+1$ steps to an instance $\mathscr{I}$. Let the result of the first l operations contain an instance $\mathscr{I}'$, and let the result of applying the final operation to $\mathscr{I}'$ contain an instance $\mathscr{I}''$. $\mathscr{I}''$ is then also in the result of applying ∇ to $\mathscr{I}$. We have to prove that there exists an extension morphism h' of type $(\mathscr{I}, \mathscr{I}'')$. From the induction hypothesis, we already know that there exists an extension morphism h of type $(\mathscr{I}, \mathscr{I}')$.

For node addition with functional edges, we show how to change the extension morphism h into an extension morphism h' of type $(\mathscr{I}, \mathscr{I}'')$, more precisely, how each automorphism of $\mathscr{I}$ can be mapped to an automorphism of $\mathscr{I}''$. For the other four basic operations, the proof is very similar.

Suppose the last operation of ∇ is a node addition with functional edges. If no new nodes are added, we define $h' = h$. If the addition adds a node $\mathbf{p}$ with outgoing functional edges labeled $\alpha_1, ..., \alpha_k$ $(k \geqslant 0)$ to respective nodes $\mathbf{p}_1, ..., \mathbf{p}_k$ of $\mathscr{I}'$, then for all $h(a) \in h(\mathrm{Aut}(\mathscr{I}))$, a node $\mathbf{q}$ is also added with outgoing functional edges labeled $\alpha_1, ..., \alpha_k$ to the nodes $h(a)(\mathbf{p}_1), ..., h(a)(\mathbf{p}_k)$ (cf. Lemma 1). We define $h'(a)(\mathbf{p}) = \mathbf{q}$. Furthermore, for all nodes $\mathbf{n}$ of $\mathscr{I}'$, we define $h'(a)(\mathbf{n}) = h(a)(\mathbf{n})$. To see that h' is an extension morphism of type $(\mathscr{I}, \mathscr{I}'')$, let e be an embedding of the pattern of the node addition in $\mathscr{I}'$, and let a_1 and a_2 be two automorphisms of $\mathscr{I}$. Suppose three nodes $\mathbf{m}_1$, $\mathbf{m}_2$, and $\mathbf{m}_3$ are added to $\mathscr{I}'$ as a result of the respective embeddings e, $h(a_1) \circ e$ and $h(a_2) \circ h(a_1) \circ e$. Then $h'(a_1)(\mathbf{m}_1) = \mathbf{m}_2$ and $h'(a_2)(\mathbf{m}_2) = \mathbf{m}_3$, so $h'(a_2) \circ h'(a_1)(\mathbf{m}_1) = \mathbf{m}_3$. But since the node added by $h(a_2) \circ h(a_1) \circ e$ is $\mathbf{m}_3$, $h'(a_2 \circ a_1)(\mathbf{m}_1) = \mathbf{m}_3$, so h' is still a group homomorphism. Because of its definition in terms of h, h' still satisfies the extension property. ▮

Next, we show that the GOOD language can express *any* generic transformation. This will be proved in two steps. First we study the special case where all instances of Γ are superinstances of $\mathscr{I}$ (i.e., of *monotonic* transformations). We give a GOOD program that, when applied to $\mathscr{I}$, results in a set of superinstances of the elements of Γ, that contain information derived from the extension morphism h. Then we state how these may be restricted to the elements of Γ (cf. Proposition 6).

In the second step, we consider arbitrary instances. We therefore first describe an extension of $\mathscr{I}$ that also includes $\mathscr{I}'$, as well as an adaptation of the extension morphism h to this superinstance. This way we can apply the result of the first step, showing that $\mathscr{I}$ GOOD-implies this superinstance. Finally we show how this superinstance can be restricted to $\mathscr{I}'$ (cf. Proposition 7).

First we introduce some additional concepts to be used in the construction of the superinstances mentioned above.

DEFINITION 13 (Orbit). Let $\mathscr{I}$ be a subinstance of $\mathscr{I}'$, and let h be an extension morphism of type $(\mathscr{I}, \mathscr{I}')$. Let $\mathbf{n}$ be a node of $\mathscr{I}'$. We call the orbit of $\mathbf{n}$ w.r.t. h the set

$$\mathrm{orb}_h(\mathbf{n}) = \{\mathbf{n}' \in N(\mathscr{I}') \mid \exists a \in \mathrm{Aut}(\mathscr{I}) : h(a)(\mathbf{n}) = \mathbf{n}'\}.$$

In each orbit we choose an arbitrary but fixed node, called the *representative* of the orbit. $\mathrm{Orbits}_h(\mathscr{I}' - \mathscr{I})$ is the set of all the orbits of nodes of $\mathscr{I}' - \mathscr{I}$ w.r.t. h.

It can easily be seen that $\mathrm{Orbits}_h(\mathscr{I}' - \mathscr{I})$ is a partition of $N(\mathscr{I}' - \mathscr{I})$.

DEFINITION 14 (Coset). Let G be a subgroup of $\mathrm{Aut}(\mathscr{I})$ and let $a \in \mathrm{Aut}(\mathscr{I})$. We define a coset of G as $a \circ G = \{a \circ b \mid b \in G\}$. $\mathrm{CosetAut}(\mathscr{I})$ is the set of all cosets of all subgroups of $\mathrm{Aut}(\mathscr{I})$.

DEFINITION 15 (Stabilizer). Let $\mathscr{I}$ be a subinstance of $\mathscr{I}'$, and let h be an extension morphism of type $(\mathscr{I}, \mathscr{I}')$. Let $\mathbf{n} \in \mathrm{N}(\mathscr{I}' - \mathscr{I})$. The stabilizer of $\mathbf{n}$ w.r.t. h is the set

$$\mathrm{st}_h(\mathbf{n}) = \{a \in \mathrm{Aut}(\mathscr{I}) \mid h(a)(\mathbf{n}) = \mathbf{n}\}.$$

It can easily be seen that $\mathrm{st}_h(\mathbf{m})$ is a subgroup of $\mathrm{Aut}(\mathscr{I})$.

Next we introduce an extension for an arbitrary instance in which it is explicitly indicated that all nodes are different.

DEFINITION 16 ($\mathscr{I}_{\mathbf{diff}}$). Let $\mathscr{I} = (N, E)$ be an object base instance. We define $\mathscr{I}_{\mathtt{diff}}$ as the instance (N, E) with

$$E' = E \cup \{(\mathbf{n}, \mathtt{diff}, \mathbf{m}) \mid \mathbf{n}, \mathbf{m} \in N, \lambda_{\mathscr{I}}(\mathbf{n}) = \lambda_{\mathscr{I}}(\mathbf{m}), \mathbf{n} \neq \mathbf{m}\}.$$

We assume that diff is a nonfunctional edge label, not occurring in $\mathscr{I}$.

If diff-edges are present in both an instance and a pattern to be matched to that instance, only *injective* embeddings of the pattern are possible.

Given an instance $\mathscr{I}$ and a superinstance $\mathscr{I}'$ such that there exists an extension morphism h of type $(\mathscr{I}, \mathscr{I}')$, we describe an instance $\langle \mathscr{I}' \rangle$ which is an extension of $\mathscr{I}'$, based on the extension morphism h, which in turn will be extended into a group homomorphism $\langle h' \rangle$: $\mathrm{Aut}(\mathscr{I}) \to \mathrm{Aut}(\langle \mathscr{I}' \rangle)$.

DEFINITION 17. Let $\mathscr{I}$ be a subinstance of $\mathscr{I}'$, such that there exists an extension morphism h of type $(\mathscr{I}, \mathscr{I}')$. We define the extension $\langle \mathscr{I}' \rangle$ of $\mathscr{I}'$ w.r.t. $\mathscr{I}$ and h as follows:

Consider $\mathrm{Orbits}_h(\mathscr{I}' - \mathscr{I})$ as a set of nodes not in $\mathscr{I}'$, labeled by a unique name for that orbit. Consider $\mathrm{Aut}(\mathscr{I})$ as a set of nodes not in $\mathscr{I}'$, labeled by AUT. Consider $\mathrm{CosetAut}(\mathscr{I})$ as a set of nodes not in $\mathscr{I}'$, labeled by a unique name for their associated subgroup. We assume that all these labels are new. We then define:

$$\begin{aligned}
\mathrm{N}(\langle \mathscr{I}' \rangle) &= \mathrm{N}(\mathscr{I}') \cup \mathrm{Orbits}_h(\mathscr{I}' - \mathscr{I}) \cup \mathrm{Aut}(\mathscr{I}) \cup \\
&\qquad \mathrm{CosetAut}(\mathscr{I}) \\
\mathrm{E}(\langle \mathscr{I}' \rangle) &= \mathrm{E}(\mathscr{I}') \cup \\
&\qquad \mathrm{E}(\mathscr{I}_{\texttt{diff}}) \cup \\
&\qquad \{(a, \mathbf{n}, a(\mathbf{n})) \mid a \in \mathrm{Aut}(\mathscr{I}), \mathbf{n} \in \mathrm{N}(\mathscr{I})\} \cup \\
&\qquad \{(C, \xi, a) \mid C \in \mathrm{CosetAut}(\mathscr{I}), a \in C\} \cup \\
&\qquad \{(\mathbf{m}, \mu, A) \mid A \in \mathrm{Orbits}_h(\mathscr{I}' - \mathscr{I}), \mathbf{m} \in A\} \cup \\
&\qquad \{(h(b)(\mathbf{m}_0), v, b \circ \mathrm{st}_h(\mathbf{m}_0)) \mid \mathbf{m}_0 \text{ an orbit-} \\
&\qquad\quad \text{representative}, b \in \mathrm{Aut}(\mathscr{I})\}.
\end{aligned}$$

We assume that $\mathbf{n}$ ($\mathbf{n} \in \mathrm{N}(\mathscr{I})$), μ, and v are functional edge labels and that ξ is a nonfunctional edge label, all not occurring in $\mathscr{I}'$.

LEMMA 3. *$\langle \mathscr{I}' \rangle$ is a well-defined instance.*

Proof. It can easily be seen that the edges labeled by $\mathbf{n}$ ($\mathbf{n} \in \mathrm{N}(\mathscr{I})$) or μ represent functional relationships. We show that v also represents a functional relationship. Let $(h(b)(\mathbf{m})_0), v, b \circ \mathrm{st}_h(\mathbf{m}_0))$ and $(h(b')(\mathbf{m}'_0), v, b' \circ \mathrm{st}_h(\mathbf{m}'_0))$ be two edges in $\langle \mathscr{I}' \rangle$, with $h(b)(\mathbf{m}_0) = h(b')(\mathbf{m}'_0)$. Applying $h(b)^{-1}$ to both sides of this equation yields $h(b^{-1} \circ b')(\mathbf{m}'_0) = \mathbf{m}_0$. Since we have chosen a unique representative for each orbit, it follows that $\mathbf{m}_0 = \mathbf{m}'_0$, so $b^{-1} \circ b'$ is a member of $\mathrm{st}_h(\mathbf{m}_0)$. As a result, $(b^{-1} \circ b') \circ \mathrm{st}_h(\mathbf{m}_0) = \mathrm{st}_h(\mathbf{m}_0)$. Applying b to both sides of this last equation yields $b' \circ \mathrm{st}_h(\mathbf{m}_0) = b \circ \mathrm{st}_h(\mathbf{m}'_0)$, which proves the functionality of the v-edges. ∎

Before we define the extended group homomorphism $\langle h' \rangle$, we first prove a lemma concerning $\langle \mathscr{I}' \rangle$ which is of critical importance to the proof of the main theorem of this article. It shows that, for each pair consisting of an orbit and some coset of the stabilizer of the representative of that orbit, there corresponds exactly one node of $\mathscr{I}' - \mathscr{I}$ (by means of the μ- and v-edges in $\langle \mathscr{I}' \rangle$).

LEMMA 4. 1. *Let $\mathbf{m}_1, \mathbf{m}_2$ be nodes of $\mathscr{I}' - \mathscr{I}$, and let $(\mathbf{m}_1, \mu, A)$, $(\mathbf{m}_2, \mu, A)$, $(\mathbf{m}_1, v, C)$, $(\mathbf{m}_2, v, C)$ be edges of $\langle \mathscr{I}' \rangle$. Then $\mathbf{m}_1 = \mathbf{m}_2$.*

2. *Let $\mathbf{m}_0$ be the representative of some orbit and let a be an automorphism of $\mathscr{I}$. Then there exists a node $\mathbf{n}$ of $\mathscr{I}'$, such that $(\mathbf{n}, \mu, \mathrm{orb}_h(\mathbf{m}_0))$ and $(\mathbf{n}, v, a \circ \mathrm{st}_h(\mathbf{m}_0))$ are edges of $\langle \mathscr{I}' \rangle$.*

Proof. 1. Let $\mathbf{m}_1 = h(b_1)(\mathbf{m}_0)$, $\mathbf{m}_2 = h(b_2)(\mathbf{m}'_0)$. Then, by the definition of orbit, $\mathbf{m}_1$, $\mathbf{m}_2$, $\mathbf{m}_0$, $\mathbf{m}'_0$ all belong to A. By the uniqueness of representatives of orbits, $\mathbf{m}_0 = \mathbf{m}'_0$. Consequently, and by the definition of the v-edges, it follows that $C = b_1 \circ \mathrm{st}_h(\mathbf{m}_0) = b_2 \circ \mathrm{st}_h(\mathbf{m}_0)$. By applying b_1^{-1} to this equation and since stabilizers are subgroups of $\mathrm{Aut}(\mathscr{I})$, it follows that $b_1^{-1} \circ b_2$ is a member of $\mathrm{st}_h(\mathbf{m}_0)$. By the definition of stabilizer, it follows that $h(b_1^{-1} \circ b_2)(\mathbf{m}_0) = \mathbf{m}_0$. Applying $h(b_1)$ to both sides of this equation, we conclude that $\mathbf{m}_1 = h(b_1)(\mathbf{m}_0) = h(b_2)(\mathbf{m}_0) = \mathbf{m}_2$.

2. The node satisfying the requirements is $h(a)(\mathbf{m}_0)$. ∎

DEFINITION 18. Let $\mathscr{I}$ be a subinstance of $\mathscr{I}'$ and let h be an extension morphism of type $(\mathscr{I}, \mathscr{I}')$. Let $\langle \mathscr{I}' \rangle$ be the extension of $\mathscr{I}'$ according to Definition 17. The group homomorphism $\langle h' \rangle$: $\mathrm{Aut}(\mathscr{I}) \to \mathrm{Aut}(\langle \mathscr{I}' \rangle)$ is defined as follows. Let $a \in \mathrm{Aut}(\mathscr{I})$.

1. $\langle h' \rangle(a)(\mathbf{n}) = h(a)(\mathbf{n})$, for $\mathbf{n} \in N(\mathscr{I}')$
2. $\langle h' \rangle(a)(O) = O$, for $O \in \mathrm{Orbits}_h(\mathscr{I}' - \mathscr{I})$;
3. $\langle h' \rangle(a)(b) = a \circ b$, for $b \in \mathrm{Aut}(\mathscr{I})$;
4. $\langle h' \rangle(a)(b \circ G) = a \circ b \circ G$, for $b \circ G \in \mathrm{CosetAut}(\mathscr{I})$.

The instances $\mathscr{K}'_1, \ldots, \mathscr{K}'_7$ are defined as follows.[3]

$$\begin{aligned}
\mathscr{K}'_1 &:= \mathscr{I} \\
\mathscr{K}'_2 &:= \mathscr{K}'_1 \cup (\varnothing, E(\mathscr{I}_{\texttt{diff}})) \\
\mathscr{K}'_3 &:= \mathscr{K}'_2 \cup (\mathrm{Aut}(\mathscr{I}), \{(a, \mathbf{n}, a(\mathbf{n})) \mid a \in \mathrm{Aut}(\mathscr{I}), \mathbf{n} \in N(\mathscr{I})\}) \\
\mathscr{K}'_4 &:= \mathscr{K}'_3 \cup (\mathrm{CosetAut}(\mathscr{I}), \{(C, \xi, a) \mid C \in \mathrm{CosetAut}(\mathscr{I}), \\
&\qquad a \in C\}) \\
\mathscr{K}'_5 &:= \mathscr{K}'_4 \cup (\mathrm{Orbits}_h(\mathscr{I}' - \mathscr{I}), \varnothing) \\
\mathscr{K}'_6 &:= \mathscr{K}'_5 \cup (N(\mathscr{I}'), \quad \{(\mathbf{m}, \mu, A) \mid A \in \mathrm{Orbits}_h(\mathscr{I}' - \mathscr{I}), \\
&\qquad \mathbf{m} \in A\} \cup \{(h(b)(\mathbf{m}_0), v, b \circ \mathrm{st}_h(\mathbf{m}_0)) \mid \mathbf{m}_0 \text{ an orbit-} \\
&\qquad \text{representative}, b \in \mathrm{Aut}(\mathscr{I})\}) \\
\mathscr{K}'_7 &:= \mathscr{K}'_6 \cup (\varnothing, E(\mathscr{I}')).
\end{aligned}$$

[3] The union of graphs is defined componentwise.

The mappings $\langle h' \rangle_1, ..., \langle h' \rangle_7$ are defined as

$$\langle h' \rangle_j : \mathrm{Aut}(\mathscr{I}) \to \mathrm{Aut}(\mathscr{K}'_j) : a \mapsto \langle h' \rangle(a)|_{N(\mathscr{K}'_j)} \quad (j = 1 \cdots 7);$$

i.e., the mapping $\langle h' \rangle_j(a)$ is defined as the restriction of the mapping $\langle h' \rangle(a)$ to the set of nodes $N(\mathscr{K}'_j)$.

LEMMA 5. *Let $\mathscr{I}'$ be a superinstance of $\mathscr{I}$ such that there exists an extension morphism h of type $(\mathscr{I}, \mathscr{I}')$. Then the mappings $\langle h' \rangle_j$ $(j = 1 \cdots 7)$ are group isomorphisms.*

Proof. This proof is structured as follows. We first show that each $\langle h' \rangle_j$ is a well-defined, injective group homomorphism. For each pair of instances $\mathscr{K}'_i$ and $\mathscr{K}'_{i+1}$ $(i = 1 \cdots 6)$, we then give a bijection between their automorphism groups. Given the algebraic property which says that an injective group homomorphism between two finite groups with equal cardinality is a group isomorphism, it follows that each $\langle h' \rangle_j$ is a group isomorphism.

We first prove that $\langle h' \rangle_1$ is well defined, in other words, that for each $a \in \mathrm{Aut}(\mathscr{I})$, $\langle h' \rangle_1(a) = \langle h' \rangle(a)|_{N(\mathscr{K}'_1)}$ is in $\mathrm{Aut}(\mathscr{K}'_1)$. To show that $\langle h' \rangle_1(a)$ is well defined, note that, because $\langle h' \rangle(a) \in \mathrm{Aut}(\langle \mathscr{I}' \rangle)$, it has to preserve μ-edges. Since these edges start at every node of $\mathscr{I}' - \mathscr{I}$, it must map nodes of $\mathscr{I}' - \mathscr{I}$ to nodes of $\mathscr{I}' - \mathscr{I}$ and, hence, nodes of $\mathscr{I}$ to nodes of $\mathscr{I}$. Since $\langle h' \rangle(a)$ is an automorphism, $\langle h' \rangle_1(a)$ is also injective and surjective and preserves node labels. To show that $\langle h' \rangle_1(a)$ also preserves edges, let $(\mathbf{x}, \alpha, \mathbf{y})$ be an edge in $\mathscr{K}'_1$, i.e., in $\mathscr{I}$. As already shown, $\langle h' \rangle(a)(\mathbf{x})$ and $\langle h' \rangle(a)(\mathbf{y})$ are still nodes of $\mathscr{K}'_1$. But by the definition of $\langle h' \rangle$ and since h is an extension morphism, $\langle h' \rangle(a)(\mathbf{x}) = a(\mathbf{x})$. Since $a \in \mathrm{Aut}(\mathscr{I})$, $(a(\mathbf{x}), \alpha, a(\mathbf{y}))$ is still an edge of $\mathscr{K}'_1$. So $\langle h' \rangle_1$ is well defined.

We show that $\langle h' \rangle_j$ is also well defined for $1 < j \leqslant 7$. Since for all $j = 2 \cdots 6$, a node in $\mathscr{K}'_j - \mathscr{K}'_{j-1}$ has either a node label not in $\mathscr{K}'_{j-1}$ or an outgoing edge with a label not in $\mathscr{K}'_{j-1}$, while $\mathscr{K}'_j$ always contains *all* nodes or edges of $\langle \mathscr{I}' \rangle$ with these new labels, $\langle h' \rangle_j(a)$ always maps nodes of $\mathscr{K}'_j$ to nodes of $\mathscr{K}'_j$ and also preserves edges. $\mathscr{K}'_7$ equals $\langle \mathscr{I}' \rangle$, so $\langle h' \rangle_7(a)$ equals $\langle h' \rangle$. From the first item of the definition of $\langle h' \rangle$, it follows immediately that also $\langle h' \rangle$ is well defined.

To prove that for all j, $\langle h' \rangle_j$ is injective, let $a \neq b \in \mathrm{Aut}(\mathscr{I})$. Consequently, there is a node $\mathbf{n}$ of $\mathscr{I}$ (and thus of $\mathscr{K}'_j$ for each j) for which $a(\mathbf{n}) \neq b(\mathbf{n})$. By Definitions 11 and 18, it follows that $\langle h' \rangle(a)(\mathbf{n}) \neq \langle h' \rangle(b)(\mathbf{n})$ and, thus, $\langle h' \rangle(a)_j(\mathbf{n}) \neq \langle h' \rangle(b)_j(\mathbf{n})$.

Since $\forall a, b \in \mathrm{Aut}(\mathscr{I})$, $\forall \mathbf{n} \in \mathscr{K}'_j$, $\langle h' \rangle_j(a \circ b)(\mathbf{n}) = \langle h' \rangle(a \circ b)(\mathbf{n}) = \langle h' \rangle(a) \circ \langle h' \rangle(b)(\mathbf{n}) = \langle h' \rangle_j(a) \circ \langle h' \rangle_j(b)(\mathbf{n})$, $\langle h' \rangle_j$ is a group homomorphism for all j.

Recall from the beginning of this proof that the only thing left to be done is to give a bijection between the automorphism groups of all pairs of instances $\mathscr{K}'_i$ and $\mathscr{K}'_{i+1}$ $(i = 1 \cdots 6)$. We will only give the details for the first two pairs of instances. The other bijections can be constructed analogously.

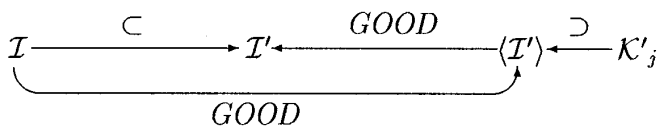

FIG. 5. An overview of instances used in the proof of Theorem 8.

Since in $\mathscr{K}'_2$, no nodes are added to $\mathscr{I}$, an automorphism of $\mathscr{I}$ can be applied to $\mathscr{K}'_2$. Since such an automorphism is an injective mapping, it preserves `diff`-edges, so it is an automorphism of $\mathscr{K}'_2$. Obviously, every automorphism of $\mathscr{K}'_2$ is also an automorphism of $\mathscr{I}$, so $\mathrm{Aut}(\mathscr{I}) = \mathrm{Aut}(\mathscr{K}'_2)$ (hence the bijection is trivial).

An automorphism a of $\mathscr{K}'_2$ can be extended so it maps an `AUT`-node b in $\mathscr{K}'_3$ to the `AUT`-node $a \circ b$. Naturally, this extension is unique. Conversely, if an automorphism of $\mathscr{K}'_3$ maps an `AUT`-node a to a node b, it means that $b = c \circ a$, where c is the restriction of that automorphism to $\mathscr{I}$. Consequently, if two automorphisms of $\mathscr{K}'_3$ are equal on $\mathscr{K}'_2$, they must be equal, so restricting such an automorphism yields a unique automorphism of $\mathscr{K}'_2$. ∎

This lemma concludes the extension of a given instance $\mathscr{I}$ into a superinstance $\langle \mathscr{I}' \rangle$ of another given instance $\mathscr{I}'$ for which there exists an extension morphism h of type $(\mathscr{I}, \mathscr{I}')$. Note that we presented this extension purely *descriptive*, independent of the GOOD transformation language. Then recall that our current aim is to show that GOOD can expresses any generic transformation, or in other words, that the existence of an extension morphism for an instance $\mathscr{I}$ and a superinstance $\mathscr{I}'$ is a sufficient condition for the existence of a GOOD program that, when applied to $\mathscr{I}$, results in a set of instances containing $\mathscr{I}'$. The GOOD program to be constructed in the proof of the following proposition contains $\langle \mathscr{I}' \rangle$ as an intermediate result; hence the proof contains a kind of *constructive* definition for $\langle \mathscr{I}' \rangle$ (cf. Fig. 5).

PROPOSITION 6. *Let $\mathscr{I}$ be an instance and let Γ be a set of superinstances of $\mathscr{I}$, satisfying the following properties*:

1. *Each two elements of Γ are $\mathscr{I}$-isomorphic*;
2. *If $\mathscr{J} \in \Gamma$ and $\mathscr{J}'$ is $\mathscr{I}$-isomorphic to $\mathscr{J}$, then $\mathscr{J}'$ is also in Γ*;
3. *For all $\mathscr{J} \in \Gamma$, there exists an extension morphism h of type $(\mathscr{I}, \mathscr{J})$.*

Then there exists a GOOD program that when applied to $\mathscr{I}$, results in Γ.

Proof. This proof is structured as follows. First we fix an instance $\mathscr{I}'$ in Γ. Since for this instance, there exists an

extension morphism h of type $(\mathscr{I}, \mathscr{I}')$, we can define a corresponding superinstance $\langle \mathscr{I}' \rangle$ according to Definition 17. We then build a GOOD program that, when applied to $\mathscr{I}$, results in a set of instances containing $\langle \mathscr{I}' \rangle$. It is shown that the intermediate resulting sets of instances of this stepwise construction contain the instances $\mathscr{K}'_j$ (where j indicates the number of the step) from Definition 18. Then we give a GOOD program that, when applied to such a superinstance $\langle \mathscr{I}' \rangle$, results in a set containing the corresponding element of Γ. Given the characterization of the resulting set of a node addition with functional edges (cf. Definition 6), it follows that the outcome of the composition of these two GOOD programs is exactly Γ.

Step 1. The input to the program is the instance $\mathscr{I}$, which equals $\mathscr{K}'_1$.

Step 2. We next add the `diff`-edges. First we add a `diff`-edge between each two nodes with the same label. Next, we delete any `diff`-edge with identical source and target. Obviously, the set of instances, resulting from the application of these two operations to $\mathscr{K}'_1$, is the singleton containing the instance $\mathscr{K}'_2$.

Step 3. We next add `AUT`-nodes with outgoing edges, labeled by nodes of $\mathscr{I}$. This operation can be accomplished with a single node addition with functional edges. We use $\mathscr{K}'_2$ as pattern and, add a node labeled by `AUT` with outgoing edges to each node $\mathbf{n}$ of $\mathscr{K}'_2$, labeled by $\mathbf{n}$. To see that this operation has the desired effect, reconsider Lemma 1. Since the identity function on $\mathscr{K}'_2$ is an embedding of the pattern of this operation, each automorphism of $\mathscr{K}'_2$ is in fact an embedding. By the presence of the `diff`-edges, these automorphisms are *all* the possible embeddings of the pattern. Hence precisely one `AUT`-node will be added for each element of $\mathrm{Aut}(\mathscr{K}'_2)$, which by Lemma 5 equals $\mathrm{Aut}(\mathscr{I})$. Consequently, $\mathscr{K}'_3$ is in the resulting set of this operation. In $\mathscr{K}'_3$, the `AUT`-nodes are actually the automorphisms of $\mathscr{I}$ *themselves*. Thus, by the choice of the pattern and the edge labels of the operation, if $(a, \mathbf{n}, \mathbf{m})$ is a new edge of $\mathscr{K}'_3$, added as the result of an embedding $a \in \mathrm{Aut}(\mathscr{I})$, then indeed $a(\mathbf{n}) = \mathbf{m}$.

Step 4. We next add nodes for the elements of $\mathrm{CosetAut}(\mathscr{I})$, with outgoing ξ-edges. Therefore, for each subgroup of $\mathrm{Aut}(\mathscr{I})$ named `D`, the following five operations are applied consecutively to $\mathscr{K}'_3$:

- a node addition with functional edges with $\mathscr{K}'_3$ as pattern, of a node labeled by `D′` (which we assume to be a new label), with outgoing functional edges, all with different new labels, to all the `AUT`-nodes that correspond to an automorphism of `D`;
- an edge addition of nonfunctional edges labeled by ξ', for each functional edge added in the previous step;
- the node addition with nonfunctional edges $\mathrm{NANF}[\{\mathbf{n}\}, \varnothing), \mathscr{H}, \mathbf{n}, \mathtt{D}, \xi', \xi'']$, where $\mathbf{n}$ is a node labeled `D′` and $\mathscr{H}$ is the resulting instance of the foregoing operation;
- an edge addition of nonfunctional edges labeled by ξ, each time there is a ξ''-edge, followed by a ξ'-edge;
- a node deletion of all `D′`-nodes.

By the same observation, used in the explanation of the correctness of Step 3, one can see that the first operation of this step results in the addition of a node $\hat{D}$ for each subgroup named `D` (with outgoing functional edges to its members), but also in the addition of a node $\widehat{a \circ D}$ for each $a \in \mathrm{Aut}(\mathscr{K}'_3)$, which is (group-)isomorphic to $\mathrm{Aut}(\mathscr{I})$. By the use of functional edges with all different labels, however, in general *several* nodes are added for one coset; e.g., if some subgroup contains n automorphisms, then the first operation adds n nodes for that particular subgroup, since an embedding of the pattern, followed by an automorphism of that subgroup, results in the addition of another `D′`-node, which corresponds to the same subgroup.

However, the resulting instance should contain exactly one node for each coset. Recalling Section 2, node addition with nonfunctional edges allows grouping objects according to common nonfunctional properties. Hence the following four operations group `D′`-nodes that represent the same set, thereby adding a unique `D`-node. Consequently, $\mathscr{K}'_4$ is in the resulting set of instances of this operation. In $\mathscr{K}'_4$, the newly added nodes are actually the cosets themselves.

We conclude this step with a calculation, which is used later on in this proof:

$$\forall a \in \mathrm{Aut}(\mathscr{I}),\ \forall D \text{ subgroup of } \mathrm{Aut}(\mathscr{I}):$$
$$\langle h' \rangle_j (a)(D) = \langle h' \rangle (a)(D) = a \circ D. \tag{3}$$

Step 5. We next add nodes representing orbits. An isolated node can be added very easily by means of a node addition with functional edges using an empty pattern. So we apply one node addition to $\mathscr{K}'_4$, using an empty pattern, for each element of O of $\mathrm{Orbits}_h(\mathscr{I}' - \mathscr{I})$, of a node labeled by a unique name for that orbit. Thus the instance $\mathscr{K}'_5$ is in the resulting set of this operation. In $\mathscr{K}'_5$, the newly added nodes are actually the orbits O themselves.

We also conclude this step of the construction with a calculation, which is used later on in the proof.

$$\forall a \in \mathrm{Aut}(\mathscr{I}),\ \forall O \in \mathrm{Orbits}_h(\mathscr{I}' - \mathscr{I}):$$
$$\langle h' \rangle_j (a)(O) = \langle h' \rangle (a)(O) = O. \tag{4}$$

Step 6. We next add nodes representing the nodes of $\mathscr{I}' - \mathscr{I}$, with outgoing μ- and ν-edges. Therefore, one-node addition with functional edges is applied to $\mathscr{K}'_5$ for each

element of $\mathrm{Orbits}_h(\mathscr{I}' - \mathscr{I})$. Given the representative $\mathbf{m}_0$ of an orbit, the operation has $\mathscr{K}'_5$ as pattern and adds a node with the same label as $\mathbf{m}_0$ and two outgoing functional edges. One μ-labeled edge arrives at the orbit $\mathrm{orb}_h(\mathbf{m}_0)$, while the other edge, labeled by ν, arrives at the coset $\mathrm{st}_h(\mathbf{m}_0)$.

To see that these operations have the desired effect, recall that $\langle h' \rangle_5$ is surjective (cf. Lemma 5), so for each $b \in \mathrm{Aut}(\mathscr{K}'_5)$, there is an $a \in \mathrm{Aut}(\mathscr{I})$ such that $\langle h' \rangle_5 (a) = b$. As a result, if $\mathbf{m}_0$ is added with outgoing edges $(\mathbf{m}_0, \mu, O)$ and $(\mathbf{m}_0, \nu, \mathrm{st}_h(\mathbf{m}_0))$, then a node $b[\mathbf{m}_0]$ is also added with outgoing edges $(b[\mathbf{m}_0], \mu, \langle h' \rangle(a)(O))$ and $(b[\mathbf{m}_0], \nu, \langle h' \rangle(a)(\mathrm{st}_h(\mathbf{m}_0)))$. The notation $b[\mathbf{m}_0]$ indicates that this is the node, added by the same node addition as $\mathbf{m}_0$ as a result of the automorphism b of $\mathscr{K}'_5$. Recalling Eqs. (3) and (4), the outgoing edges of $b[\mathbf{m}_0]$ equal $(b[\mathbf{m}_0], \mu, O)$ and $(b[\mathbf{m}_0], \nu, a \circ \mathrm{st}_h(\mathbf{m}_0))$. Consequently, we have added a node for each pair, consisting of an orbit and an arbitrary coset of the stabilizer of the representative of that orbit. We now want to apply Lemma 4 to $\mathscr{K}'_6$. In fact, this lemma concerns the instance $\langle \mathscr{I}' \rangle$, but the only difference between $\langle \mathscr{I}' \rangle$ and $\mathscr{K}'_6$, is that $\mathscr{K}'_6$ lacks the edges of $\mathscr{I}' - \mathscr{I}$. Since the absence of these edges does not invalidate the proof of Lemma 4, we may apply it here. Recalling the introduction to Lemma 4, it follows that for each node of $\mathscr{I}' - \mathscr{I}$ *exactly* one node is added. Hence, the instance $\mathscr{K}'_6$ is in the resulting set of this operation. In $\mathscr{K}'_6$, the newly added nodes are actually the nodes of $\mathscr{I}' - \mathscr{I}$ themselves.

Step 7. Finally, we add the edges of $\mathscr{I}' - \mathscr{I}$. For each such edge, say $(\mathbf{n}, \alpha, \mathbf{m})$, an edge addition is applied with $\mathscr{K}'_6$ as pattern, of an edge $(\mathbf{n}, \alpha, \mathbf{m})$. Obviously, these operations add *at least* enough edges. To see that they do not add too many edges, note that for each such edge and for each $b \in \mathrm{Aut}(\mathscr{K}'_6)$, an edge $(b(\mathbf{n}), \alpha, b(\mathbf{m}))$ is also added. Since $\langle h' \rangle_6$ is surjective (cf. Lemma 5), there exists an automorphism a of $\mathscr{I}$ such that $b = \langle h' \rangle_6 (a)$, so the edge is actually $(\langle h' \rangle_6 (a)(\mathbf{n}), \alpha, \langle h' \rangle_6 (a)(\mathbf{m}))$, or, by the definition of $\langle h' \rangle_6$ and $\langle h' \rangle$, $(h(a)(\mathbf{n}), \alpha, h(a)(\mathbf{m}))$. Since $h(a)$ is an automorphism of $\mathscr{I}'$, this edge must be present in $\mathscr{I}'$, and hence in $\langle \mathscr{I}' \rangle$. Consequently, the resulting instance is $\langle \mathscr{I}' \rangle$, which equals $\mathscr{K}'_7$.

Summarizing, the application to $\mathscr{I}$ of the six GOOD programs described above, results in a set of instances containing $\langle \mathscr{I}' \rangle$. Restricting $\langle \mathscr{I}' \rangle$ to the given instance $\mathscr{I}'$ can be done very easily by deleting all nodes labeled by `AUT` or by some identifier for an orbit or a subgroup of $\mathrm{Aut}(\mathscr{I})$, as well as all `diff`-edges. ▮

We still have to prove that Proposition 6 is still valid if we drop the requirement that Γ must contain nothing but superinstances of $\mathscr{I}$. Fortunately, proving this becomes easy if we use the previous proposition. Before stating the final proposition, leading to the proof of Theorem 8, we define a special kind of superinstance for two instances, containing the "information" of both these instances.

FIG. 6. Instances, used in the proof of Proposition 7.

DEFINITION 19. Let $\bar{\mathscr{I}}$ and $\mathscr{I}'$ be instances with disjoint sets of edge labels. We define the instance $\mathscr{M}_{\bar{\mathscr{I}}, \mathscr{I}'}$ as

$$\mathrm{N}(\mathscr{M}_{\bar{\mathscr{I}}, \mathscr{I}'}) = \mathrm{N}(\bar{\mathscr{I}}) \cup \mathrm{N}(\mathscr{I}') \cup \{\boldsymbol{l}, \text{ a new node with label } \texttt{difference}\}$$

$$\mathrm{E}(\mathscr{M}_{\bar{\mathscr{I}}, \mathscr{I}'}) = \mathrm{E}(\bar{\mathscr{I}}) \cup \mathrm{E}(\mathscr{I}') \cup \{(\boldsymbol{l}, \_\mathtt{K}, \mathbf{n}) \mid \mathbf{n} \in \mathrm{N}(\bar{\mathscr{I}} - \mathscr{I}'), \lambda_{\bar{\mathscr{I}}}(\mathbf{n}) = \mathtt{K}\}.$$

We assume that `difference` is a new element of OL, and that for each $\mathtt{K} \in \mathrm{OL}$, $\_\mathtt{K}$ is a new element of NFEL. ▮

The requirement that the sets of used edge labels should be disjoint, ensures that the superinstance is indeed a well-defined instance, since the union of two instances is in general not an instance; conflicts may arise with the functionality of edges (cf. Definition 1).

PROPOSITION 7. *Let $\mathscr{I}$ be an instance and let Γ be a set of instances, satisfying the following properties*:

1. *Each two elements of Γ are $\mathscr{I}$-isomorphic*;
2. *If $\mathscr{J} \in \Gamma$, and $\mathscr{J}'$ is $\mathscr{I}$-isomorphic to $\mathscr{J}$, then $\mathscr{J}'$ is also in Γ*;
3. *For all $\mathscr{J} \in \Gamma$, there exists an extension morphism h of type $(\mathscr{I}, \mathscr{J})$.*

Then there exists a GOOD program that when applied to $\mathscr{I}$, results in Γ.

Proof. In this proof, we give or prove the existence of three GOOD programs. The first one maps $\mathscr{I}$ to a singleton containing an instance $\bar{\mathscr{I}}$ whose edge labels are all different from those of $\mathscr{I}'$, so we can make use of Definition 19. The second one maps $\bar{\mathscr{I}}$ to a set containing the instance $\mathscr{M}_{\bar{\mathscr{I}}, \mathscr{I}'}$, while the third maps this instance to a set containing $\mathscr{I}'$ (see Fig. 6).

The first GOOD program is straightforward; for each edge label α, that occurs in $E(\mathscr{I})$, two operations are applied to $\mathscr{I}$. First, for each α-edge connecting two nodes, an edge labeled $\_\alpha$ (which is assumed to be an edge label occurring in neither $\mathscr{I}$ nor $\mathscr{I}'$) is added between the same two nodes. Next, all α-edges are deleted. Obviously, $\mathrm{Aut}(\bar{\mathscr{I}}) = \mathrm{Aut}(\mathscr{I})$, so h is also an extension morphism of type $(\bar{\mathscr{I}}, \mathscr{I}')$.

The existence of the second transformation will be shown using Proposition 6. Therefore we define the following mapping h' from $\mathrm{Aut}(\bar{\mathscr{I}})$ to $\mathrm{Aut}(\mathscr{M}_{\bar{\mathscr{I}}, \mathscr{I}'})$. Let $a \in \mathrm{Aut}(\bar{\mathscr{I}})$:

1. $h'(a)(\mathbf{n}) = a(\mathbf{n})$, for $\mathbf{n} \in N(\bar{\mathscr{I}})$;
2. $h'(a)(\mathbf{n}) = h(a)(\mathbf{n})$, for $\mathbf{n} \in N(\mathscr{I}')$;
3. $h'(a)(\boldsymbol{l}) = \boldsymbol{l}$.

For brevity, we omit the tedious but straightforward verification that h' is an extension morphism of type $(\bar{\mathscr{I}}, \mathscr{M}_{\bar{\mathscr{I}}, \mathscr{I}'})$. Applying Proposition 6, we know that there exists a GOOD program that maps $\bar{\mathscr{I}}$ to a set containing $\mathscr{M}_{\bar{\mathscr{I}}, \mathscr{I}'}$.

Finally, the following (third) GOOD program maps $\mathscr{M}_{\bar{\mathscr{I}}, \mathscr{I}'}$ to a set containing $\mathscr{I}'$. First, delete all edges labeled $\_\alpha$. Next, delete all nodes that are linked to $\boldsymbol{l}$. Then delete $\boldsymbol{l}$.

Combining these three GOOD programs, we get the desired GOOD program. ▮

Combining Propositions 2 and 7, the following theorem easily follows.

THEOREM 8. *The GOOD language is BP-complete.*

We conclude this section with a few corollaries, in which some simple *classes* of transformations are shown to be computable in GOOD.

COROLLARY 9. *If $\mathscr{I}$ and $\mathscr{I}'$ are instances with empty intersection, then $\mathscr{I} \overset{\text{GOOD}}{\Longrightarrow} \mathscr{I}'$.*

Indeed, the homomorphism mapping each automorphism of $\mathscr{I}$ to the identity on $\mathscr{I}'$ always satisfies the extension property.

COROLLARY 10. *For an arbitrary instance $\mathscr{I}$, $(\varnothing, \varnothing) \overset{\text{GOOD}}{\Longrightarrow} \mathscr{I}$.*

This is just a specialization of the previous corollary. It shows that any object base instance can be generated starting from scratch.

COROLLARY 11. *Let $\mathscr{I}$ be an instance such that $\mathrm{Aut}(\mathscr{I}) = \{\mathrm{id}_{\mathscr{I}}\}$, and let $\mathscr{I}'$ be an arbitrary instance. Then $\mathscr{I} \overset{\text{GOOD}}{\Longrightarrow} \mathscr{I}'$.*

The intuition behind this corollary is the fact that in such an instance, any node is clearly distinguishable from any other node by means of some pattern (e.g., the instance itself).

4. VARIATIONS ON MODEL AND THEOREM

4.1. *Operations on Equivalence Classes*

A first variation on Theorem 8 has something to do with the fact that GOOD operations are set-valued functions. This already resulted in a special definition for the semantics of operation composition (cf. Eq. (1)). Things become more elegant if we redefine the operations in such a way that they not only *result* in sets of isomorphic instances, but also *operate* on such sets. Using the textual notations introduced for the basic operations in Section 2.2, we define the semantics of an application of such an operation to a set of isomorphic instances as follows.

DEFINITION 20 (Class operations). Let $[\mathscr{I}]$ denote the equivalence class of all instances isomorphic to $\mathscr{I}$. Given an application of some primitive GOOD operation of the form $\mathrm{OP}[\mathscr{J}, \mathscr{I}, \ldots]$, the following formula defines its effect on an equivalence class:

$$\mathrm{OP}[\mathscr{J}, [\mathscr{I}], \ldots] := \bigcup_{\mathscr{H} \in [\mathscr{I}]} \mathrm{OP}[\mathscr{J}, \mathscr{H}, \ldots].$$

After rephrasing the definition of GOOD-implication, we can elegantly rephrase Theorem 8. We omit its straightforward proof.

DEFINITION 21 (Class-GOOD-implication). Let $\mathscr{I}$ and $\mathscr{I}'$ be two object base instances. $[\mathscr{I}] \overset{\text{GOOD}}{\Longrightarrow} [\mathscr{I}']$ indicates that there are a pair of representatives $\mathscr{H}$ of $[\mathscr{I}]$ and $\mathscr{H}'$ of $[\mathscr{I}']$ and a GOOD program that, when applied to $\mathscr{H}$, results in a set of instances containing $\mathscr{H}'$.

THEOREM 12. *Let $\mathscr{I}$ and $\mathscr{I}'$ be object base instances. Then the following properties are equivalent*:

1. $[\mathscr{I}] \overset{\text{GOOD}}{\Longrightarrow} [\mathscr{I}']$.
2. *There exist a pair of representatives $\mathscr{H}$ of $[\mathscr{I}]$ and $\mathscr{H}'$ of $[\mathscr{I}']$ for which there exists an extension-morphism of type $(\mathscr{H}, \mathscr{H}')$.*

4.2. *Introducing Atomic Objects*

When illustrating object base instances in Section 2, we noted that, since the data model is "purely object-based," the representation of certain kinds of information seemed a bit awkward. Also the fact that we had to introduce separate node labels (or *class names*) for boxes and tables, which in the example are all nothing more than places where something may be located, is not very natural. This is due to the fact that, in the current primitive data model, *atomic* information cannot be represented adequately. Therefore, we will show how, by a slight extension of the data model, such "real-world" information may be modeled more directly and elegantly.

First, we redefine object-base instances by making a distinction between nodes that represent atomic information, and therefore have no other properties but a (possible) value, and "general" nodes. In replacement of the set of object labels OL, we postulate the existence of two infinitely enumerable sets of printable, (resp. nonprintable) object labels POL (resp. NPOL). We also assume there is a function π which associates to each printable object label a set of constants (e.g., strings, numbers, booleans, but also drawings, graphics, sound).

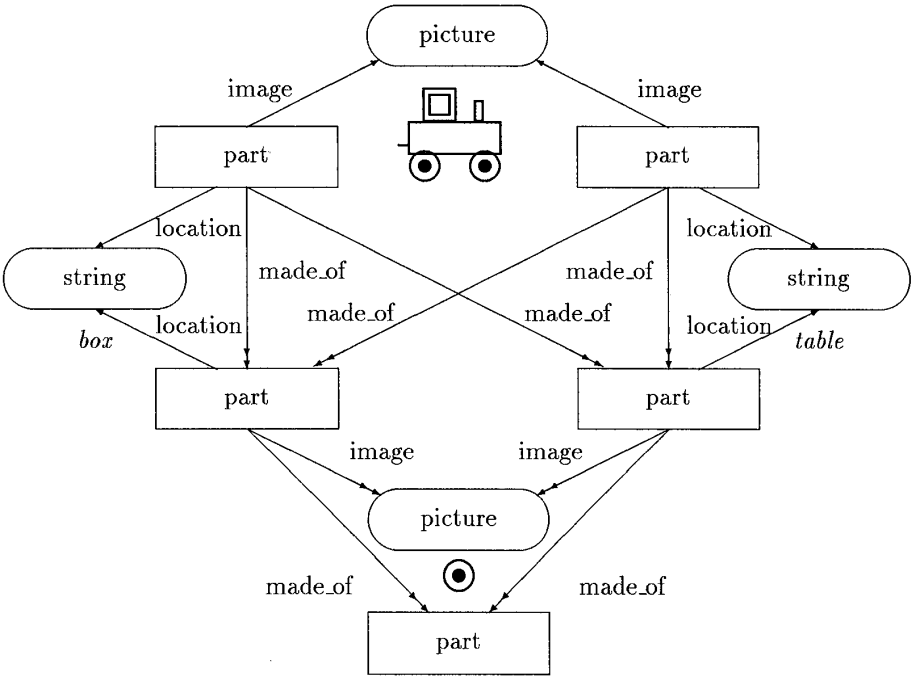

FIG. 7. An extended instance for an object base for parts and subparts.

DEFINITION 22 (Extended object base instance). An extended object base instance $\mathscr{I}$ is a directed labeled graph (N, E):

- N is a finite set of labeled nodes; if $\mathbf{n}$ is a node in N such that its label, denoted by $\lambda_{\mathscr{I}}(\mathbf{n})$, is in NPOL (resp. in POL), then $\mathbf{n}$ is called a nonprintable node (resp. a printable node) and is represented by a rectangular node (resp. an oval node);
- a printable node $\mathbf{n}$ in N may have an additional label, denoted by print($\mathbf{n}$), which is called its *print label*; this must be an element of $\pi(\lambda_{\mathscr{I}}(\mathbf{n}))$;
- E is a set of labeled edges; if $\mathbf{e}$ is a labeled edge in E, then $\mathbf{e} = (\mathbf{m}, \alpha, \mathbf{n})$ with $\mathbf{m}, \mathbf{n} \in N$ and its label $\lambda_{\mathscr{I}}(\mathbf{e}) = \alpha \in$ FEL $\cup$ NFEL; if $\lambda_{\mathscr{I}}(\mathbf{e})$ is in FEL (resp. in NFEL), then $\mathbf{e}$ is called a functional edge (resp. a nonfunctional edge);
- if $(\mathbf{m}, \alpha, \mathbf{n}_1)$ and $(\mathbf{m}, \alpha, \mathbf{n}_2) \in E$, then $\lambda_{\mathscr{I}}(\mathbf{n}_1) = \lambda_{\mathscr{I}}(\mathbf{n}_2) \in E$; moreover, if $\alpha \in$ FEL, then $\mathbf{n}_1 = \mathbf{n}_2$;
- if $\lambda_{\mathscr{I}}(\mathbf{n}_1) = \lambda_{\mathscr{I}}(\mathbf{n}_2)$ is in POL and print($\mathbf{n}_1$) = print($\mathbf{n}_2$) (or if neither node has a print label), then $\mathbf{n}_1 = \mathbf{n}_2$.

As an illustration, Fig. 7 shows an extended object base instance similar to that of Fig. 1. Only here, locations are represented by printable nodes with node label `String`, and with as print label the name of some location (e.g., *box*, *table*). Besides, some pictorial information is also included in the instance, which shows that the instance actually shows the structure of a pair of toy trains.

Extended patterns are defined as an extended object base instance. The definition of embedding (and consequently of isomorphism, $\mathscr{I}$-isomorphism, automorphism, and extension morphism) must also be adapted to incorporate printable nodes. We only redefine embeddings, since the other definitions can straightforwardly be adapted.

DEFINITION 23 (Extended embedding). Let $\mathscr{I} = (N, E)$ be an extended object base instance and let $\mathscr{J} = (M, F)$ be an extended pattern. An extended embedding of $\mathscr{J}$ in $\mathscr{I}$ is a total mapping $i: M \to N$:

1. $\forall \mathbf{n} \in M: \lambda_{\mathscr{I}}(i(\mathbf{n})) = \lambda_{\mathscr{J}}(\mathbf{n})$
2. $\forall \mathbf{n} \in M: \lambda_{\mathscr{P}}(\mathbf{n}) \in \text{POL} \Rightarrow \text{print}(\mathbf{n}) = \text{print}(i(\mathbf{n}))$ (if $\mathbf{n}$ has a print label)
3. $\forall \mathbf{n}, \mathbf{n}' \in M$, $\forall \alpha \in \text{FEL} \cup \text{NFEL}$: $(\mathbf{n}, \alpha, \mathbf{n}') \in F \Rightarrow (i(\mathbf{n}), \alpha, i(\mathbf{n}')) \in E$.

The five primitive GOOD operations can also be redefined straightforwardly by applying them to all extended embeddings of an extended pattern. For the two node addition operations, we impose the restriction that only *nonprintable* nodes may be added. This restriction is motivated by the idea that, since printable nodes actually represent atomic values, one may assume that they are constantly present in an extended object base instance. A sequence of extended operations is called an extended GOOD program.

THEOREM 13. *Let $\mathscr{I}$ be an extended object base instance, and let Γ be a set of extended object base instances. Then the following three properties are necessary and sufficient*

conditions for the existence of an extended GOOD program that, when applied to $\mathscr{I}$, results in Γ:

1. *Each two elements of Γ are extended $\mathscr{I}$-isomorphic*;
2. *If $\mathscr{J} \in \Gamma$, and $\mathscr{J}'$ is extended $\mathscr{I}$-isomorphic to $\mathscr{J}$, then $\mathscr{J}'$ is also in Γ*;
3. *For all $\mathscr{J} \in \Gamma$, there exists an extended extension morphism h of type $(\mathscr{I}, \mathscr{J})$.*

The proof of Theorem 13 is completely analogous to that of Theorem 8: one only has to take care to replace all concepts, such as instances and isomorphisms, by their respective extended counterparts.

4.3. *On Z-Genericity in Pure Object-Based Models*

The respective data models introduced in Sections 2 and 4.2, used in Theorems 8 and 13, differ in their treatment of atomic objects. For Theorem 8, we used a model in which such atomic objects are treated like all other objects, while for Theorem 13, we introduced a model in which certain objects may be designated as representing atomic values (and, hence, carry a print label).

Consequently, in Theorem 8, we considered general GOOD programs that preserve automorphisms that perform arbitrary (relationship-preserving) permutations on atomic objects (represented by general objects and, hence, indistinguishable from other objects) with the same label. On the other hand, in Theorem 13 we dealt with GOOD programs that only preserve automorphisms that leave *all* atomic objects fixed and respect node labels.

In work on the expressiveness of query languages for value-based models, however, one often makes use of so called *Z-generic* transformations [3, 6, 12], yet another notion of database transformations which lies somewhere in between the two notions considered above. In the cited works, a database transformation is called Z-generic if it commutes with any permutation on the set of atomic values in the database *that leaves some set Z of values fixed* and that respects the types of values. Although the essence of genericity, as outlined in the Introduction to this paper, is that database operations should not be allowed to interpret individual values of the database instance, this addition of a special set Z of "privileged" values seems necessary in value-based formalisms in order to allow operations to *name* some of the values explicitly. On the other hand, this set Z may not contain *all* the values, because otherwise *any* transformation would be Z-generic.

How is it then possible that, when we consider a pure object-based model, we can prove the completeness with respect to the set of generic transformations of two languages by using, in one case automorphisms that do not have to deal with atomic objects (since they are indistinguishable from the rest), while in the other case automorphisms treat all such objects uniformly?

In the following, we first adapt the notion of Z-permutation to the context of object-based data models, thus obtaining *Z-automorphisms*. Then we straightforwardly adapt the notion of extension morphism to this new class of automorphisms. We then prove that the existence of an extension morphism between the automorphism groups of two instances is equivalent to the existence of some Z such that there is an extension morphism between the Z-automorphism groups.

First, we must slightly alter Definition 22 of extended object-base instances. Prior to this definition, we namely postulated the existence of, among others, an infinitely enumerable set of nodes. Instead, given the two sets of node labels, we assume that in this "universe" of nodes, every node already carries a unique label. More formally, we assume that for each $a \in \mathrm{POL} \cup \mathrm{NPOL}$, there exists an infinitely enumerable set $\mathscr{N}_a$ of a-nodes, such that for $a' \neq a''$, $\mathscr{N}_{a'}$ and $\mathscr{N}_{a''}$ are disjoint. We then introduce the following notations:

- $\forall a \in \mathrm{POL} \cup \mathrm{NPOL}$, the set $\mathscr{N}_a$ denotes the corresponding set of nodes.
- $\mathscr{P} := \bigcup_{a \in \mathrm{POL}} \mathscr{N}_a$.

DEFINITION 24 (Z-Automorphisms). Let $\mathscr{I} = (N, E)$ be an extended object-base instance, and let $Z \subseteq \mathscr{P}$. A Z-automorphism i of $\mathscr{I}$ is a permutation of N:

1. $\forall \mathbf{n} \in N$: $\lambda_{\mathscr{I}}(i(\mathbf{n})) = \lambda_{\mathscr{I}}(\mathbf{n})$
2. $\forall \mathbf{n} \in Z \cap N$: $i(\mathbf{n}) = \mathbf{n}$
3. $\forall \mathbf{n}, \mathbf{n}' \in N$, $\forall \alpha \in \mathrm{FEL} \cup \mathrm{NFEL}$: $(\mathbf{n}, \alpha, \mathbf{n}') \in E \Leftrightarrow (i(\mathbf{n}), \alpha, i(\mathbf{n}')) \in E'$.

$\mathrm{Aut}_Z(\mathscr{I})$ is the set of all Z-automorphisms of $\mathscr{I}$.

THEOREM 14. *Let $\mathscr{I} = (N, E)$ and $\mathscr{I}' = (N', E')$ be two extended object base instances. Then the following two properties are equivalent*:

1. *There exists an extension morphism of type $(\mathscr{I}, \mathscr{I}')$.*
2. *There exists a finite $Z \subset \mathscr{P}$ and a group homomorphism h: $\mathrm{Aut}_Z(\mathscr{I}) \to \mathrm{Aut}_Z(\mathscr{I}')$:*

$$\forall \mathbf{n} \in N \cap N', \forall a \in \mathrm{Aut}(\mathscr{I}): a(\mathbf{n}) = h(a)(\mathbf{n}).$$

Proof. To prove that property 1 implies property 2, note that if we take Z equal to $\mathscr{P} \cap N$, $\mathrm{Aut}_Z(\mathscr{I})$ equals $\mathrm{Aut}(\mathscr{I})$.

For the other implication, first we remark that if $Z \subset Z'$, then $\mathrm{Aut}_{Z'}(\mathscr{I}) \subset \mathrm{Aut}_Z(\mathscr{I})$. Consequently, for each $Z \subset \mathscr{P}$, $\mathrm{Aut}_{\mathscr{P}}(\mathscr{I}) \subset \mathrm{Aut}_Z(\mathscr{I})$. It follows that we can restrict the given group homomorphism h to $\mathrm{Aut}_{\mathscr{P}}(\mathscr{I})$, of which we already remarked that it equals $\mathrm{Aut}(\mathscr{I})$. The question is: what is the range of this restriction?

Let $a \in \mathrm{Aut}(\mathscr{I})$. First we remark that $N \cap \mathscr{P} \subset N \cap N'$ (node deletions cannot remove printable nodes). By the

properties of h, since a fixes $N \cap \mathscr{P}$, so does $h(a)$. Consequently, the restriction of h to $\mathrm{Aut}(\mathscr{I})$ is a mapping to $\mathrm{Aut}(\mathscr{I}')$. Since the restriction of a group homomorphism is a group homomorphism and since the extra condition on extension morphisms is also preserved under restrictions, the restriction of h to $\mathrm{Aut}(\mathscr{I})$ is an extension morphism of type $(\mathscr{I}, \mathscr{I}')$. ∎

4.4. *Copy Generation*

We next consider what we can still achieve in the GOOD language, given two instances with no known relationship between their respective groups of automorphisms. The following (abstract) example illustrates that, in general, there is no GOOD program that maps one of those instances to the other. This example has been of great use in both formulating and proving some of the results listed in this paper.

Consider an instance $\mathscr{I}$ consisting of two isolated nodes with label A. Then look at the instance $\mathscr{I}'$ of Fig. 8 (the indices are not part of the labels, but will be used to uniquely identify the nodes in this picture). Let us first of all look at the automorphism group of this instance. It contains three mappings besides the identity. First, we can fix the A-nodes and interchange the B-nodes 1 and 3, as well as 2 and 4. Second, if we interchange the A-nodes, we have two possibilities: we can "rotate" the cycle of B-nodes clockwise (mapping node 1 to 2, 2 to 3, etc.) or counterclockwise (mapping node 1 to 4, 4 to 3, etc.). A possible extension morphism of type $(\mathscr{I}, \mathscr{I}')$ would, of course, be completely determined by the image of the automorphism a of $\mathscr{I}$ that interchanges the A-nodes. By the extension property, this image should also interchange the A-nodes. This leaves us two possible extensions for a. Unfortunately, both these automorphisms of $\mathscr{I}'$ have order 4, while a has order 2, and since an extension morphism must still be a homomorphism, there are no viable candidates for extending a. Hence, there exists no GOOD program that maps two isolated A-nodes to the instance of Fig. 8, or in other words, this mapping is not generic.

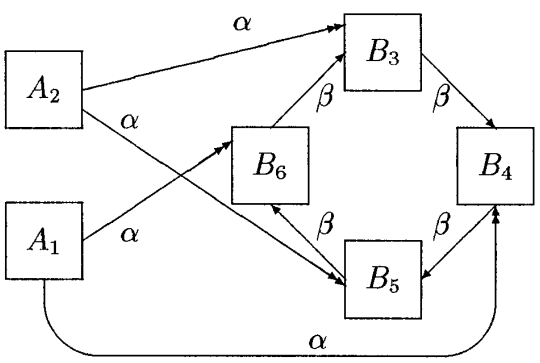

FIG. 8. An instance illustrating a nongeneric mapping.

In what follows, we state a kind of upper bound for what can be obtained in the case of an instance $\mathscr{I}$ and an arbitrary superinstance $\mathscr{I}'$. First, we define an instance containing *copies* of a given instance, identified on the nodes of a given subinstance.

DEFINITION 25 (Instance with copies). Let $\mathscr{I}$ be a subinstance of $\mathscr{I}'$. Let $\mathrm{Aut}(\mathscr{I}) = \{a_1, ..., a_z\}$. Let $\mathscr{I}_1, ..., \mathscr{I}_z$ be instances, satisfying the following properties. First, for all $1 \leqslant i < j \leqslant z$, $\mathscr{I}_i \cap \mathscr{I}_j$ must be exactly $\mathscr{I}$. Second, for each $1 \leqslant i \leqslant z$, there should exist an isomorphism $b_i: \mathscr{I}' \to \mathscr{I}_i$, such that $b_{i \mid \mathscr{I}} = a_i$. An instance $\mathscr{C}_{\mathscr{I}, \mathscr{I}'}$ with copies for the pair $(\mathscr{I}, \mathscr{I}')$ is then defined as $\bigcup_{1 \leqslant i \leqslant z} \mathscr{I}_i$.

Our definition has been inspired by a similar notion which is considered in [1]. Definition 4.2 of that article formalizes the notion of an instance containing a number of *disjoint* copies of a given instance.

The following theorem now states that we can always map an instance to a number of copies of another instance, whatever the relationship may be between their automorphism groups.

THEOREM 15. *Let $\mathscr{I}$ be a subinstance of $\mathscr{I}'$. Then* $\mathscr{I} \xrightarrow{\text{GOOD}} \mathscr{C}_{\mathscr{I}, \mathscr{I}'}$.

Proof. The most elegant way to prove this theorem is to use Theorem 8. Thus we have to prove the existence of an extension morphism of type $(\mathscr{I}, \mathscr{C}_{\mathscr{I}, \mathscr{I}'})$. Therefore, we fix for each subinstance $\mathscr{I}'_a$ of $\mathscr{C}_{\mathscr{I}, \mathscr{I}'}$ (labeled by an automorphism of $\mathscr{I}$) an $\mathscr{I}$-isomorphism i_a of $\mathscr{I}'$ in $\mathscr{I}'_a$. Let $a \in \mathrm{Aut}(\mathscr{I})$. Then define the mapping $h_c: \mathrm{Aut}(\mathscr{I}) \to \mathrm{Aut}(\mathscr{C}_{\mathscr{I}, \mathscr{I}'})$ as follows:

1. $h_c(a)(\mathbf{n}) = a(\mathbf{n})$, for $\mathbf{n} \in N(\mathscr{I})$;
2. $h_c(a)(\mathbf{n}) = i_a \circ i_b^{-1}(\mathbf{n})$, for $\mathbf{n} \in \mathscr{I}'_b$.

By the first condition from the definition of an instance with copies, it follows that $h_c(a)$ is indeed an automorphism of $\mathscr{C}_{\mathscr{I}, \mathscr{I}'}$. By the first item in its definition, h_c obviously satisfies the extension property, while from the second item, it follows straightforwardly that it is also a homomorphism. ∎

An application of Theorem 15 to the pair of instances $(\mathscr{I}, \mathscr{I}')$ tells us we can generate a superinstance of $\mathscr{I}$ containing two (= the cardinality of $\mathrm{Aut}(\mathscr{I})$) copies of $\mathscr{I}'$. This instance is shown in Fig. 9. The fact that this instance can indeed be reached starting from the two isolated A-nodes, can of course also be shown using Theorem 8. Indeed, this instance has a viable extension for the automorphism a namely the automorphism interchanging the A-nodes, as well as the B-nodes i and $i+6$ (for $i = 1, ..., 4$).

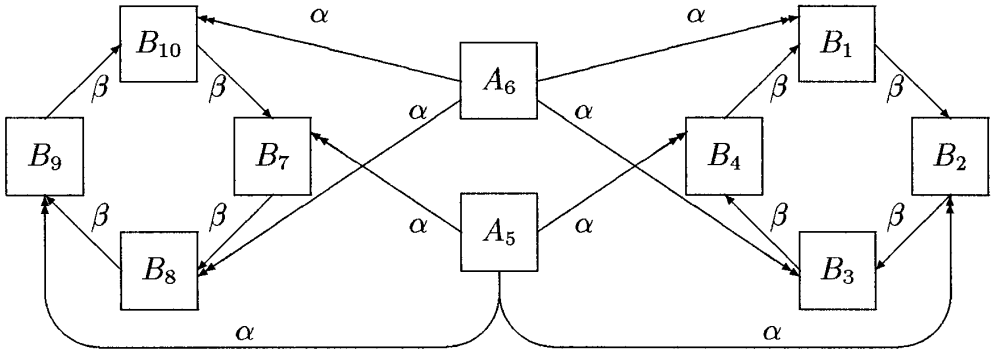

FIG. 9. An instance with copies of the instance of Fig. 8.

4.5. *A Restriction of the GOOD Language*

As a final variation on our main theorem, we consider the language obtained by omitting the operation for adding nodes with outgoing nonfunctional edges. Let us call this language GOOD$^-$. Since it has been shown that this operation cannot be expressed using the other four operations [14], the set of transformations that can be expressed using this restricted language is a strict subset of the set of transformations expressible in the full GOOD language. As for the theorem, this implies that the property, requiring the existence of an extension morphism, has to be strengthened.

THEOREM 16. *Let $\mathscr{I} = (N, E)$ be an instance, and let Γ be a set of instances. Then the following two properties are equivalent*:

1. *Γ is the resulting set of a* GOOD$^-$ *program applied to $\mathscr{I}$.*

2. (a) *Every two elements of Γ are $\mathscr{I}$-isomorphic*;

(b) *If $\mathscr{J} \in \Gamma$, and $\mathscr{J}'$ is $\mathscr{I}$-isomorphic to $\mathscr{J}$, then $\mathscr{J}'$ is also in Γ*;

(c) *For all $\mathscr{J} = (N', E') \in \Gamma$, there exists an extension morphism h of type $(\mathscr{I}, \mathscr{J})$ such that for all $\mathbf{m} \in N' - N$, there exists an ordered list $B(\mathbf{m})$ over N*:

$$\forall a \in \mathrm{Aut}(\mathscr{I})\colon B(h(a)(\mathbf{m})) = a(B(\mathbf{m}))$$

$$\forall a \in \mathrm{Aut}(\mathscr{I})\colon a(B(\mathbf{m})) \neq B(\mathbf{m}) \Leftrightarrow h(a)(\mathbf{m}) \neq \mathbf{m}.$$

In the remainder, we will call the list $B(\mathbf{m})$ the *base* of $\mathbf{m}$. Note first of all that the only difference from the conditions also mentioned in Proposition 6 is precisely this condition concerning bases. Although at first sight this condition may seem somewhat ad hoc, there exists a strong similarity with Lemma 4. The latter lemma states that a node is uniquely determined (besides by its orbit) by some set, while Theorem 16 is based on the fact that a node is determined by some list. The intuition behind this is clearly motivated by the difference between node addition with functional and nonfunctional edges.

Intuitively, the base of a node lists all nodes of $\mathscr{I}$ that "played a part" in the creation of that node. In GOOD$^-$, each newly created node $\mathbf{n}$ actually represents a tuple of other nodes, either nodes of $\mathscr{I}$, or nodes formerly created through other node additions. The base of $\mathbf{n}$ is then the catenation of these nodes of $\mathscr{I}$ and the bases of the formerly created nodes.

Sketch of Proof. The essence of the proof that every program in GOOD$^-$ satisfies condition 2 is once more an induction on the number of operations of the given program. In case of a node addition, the base of newly created nodes is defined as mentioned above.

To show that for every transformation of an instance $\mathscr{I}$ to a set Γ satisfying condition 2 there exists a program in GOOD$^-$ which expresses this transformation, we have to modify the seven-step construction from the proof of Proposition 6. First we again add `diff`-edges and nodes representing orbits. Let us call the resulting instance $\mathscr{K}$. Since the bases of nodes of an instance $\mathscr{J} \in \Gamma$ are already present in $\mathscr{I}$, we can now immediately add these nodes, by means of a single node addition for each orbit O. Suppose that this orbit has representative $\mathbf{m}_0$, with label A and base $[\mathbf{n}_1, ..., \mathbf{n}_k]$. Then the node addition for this orbit is $\mathrm{NA}[\mathscr{K}, \mathscr{K}, A, \{(0, O), (1, \mathbf{n}_1), ..., (k, \mathbf{n}_k)\}]$. It may be verified that such a node addition adds as many nodes as there are mappings in the set $\{a|_{B(\mathbf{m}_0)} \mid a \in \mathrm{Aut}(\mathscr{I})\}$. To see that this is, indeed the required number of nodes (i.e., one for each element of O), observe that the function which maps a mapping $a|_{B(\mathbf{m}_0)}$ from the aforementioned set to the node $h(a)(\mathbf{m}_0) \in O$ is a bijection. To end the construction, the edges of $\mathscr{J}$ are added in an identical way as in the proof of Proposition 6, after which any auxiliary nodes and edges are removed. ∎

ACKNOWLEDGMENTS

We thank the anonymous referee who pointed out some mistakes in the definitions of GOOD programs and $\mathscr{I}$-isomorphisms.

REFERENCES

1. S. Abiteboul and P. C. Kanellakis, Object identity as a query language primitive, *in* "1988 Proceedings SIGMOD International Conference on Management of Data" (H. Boral and P. A. Larson, Ed.) pp. 159–173, ACM Press, New York, 1988.
2. "Proceedings, Ninth ACM Symposium on Principles of Database Systems," ACM Press, New York, 1990.
3. A. V. Aho and J. D. Ullman, Universality of data retrieval languages, *in* "Symposium on Principles of Programming Languages, 1979."
4. F. Bancilhon, On the completeness of query languages for relational data bases, *in* "Proceedings, 7th Symposium on Mathematical Foundations of Computer Science," Lecture Notes in Computer Science, Vol. 64 pp. 112–123, Springler-Verlag, Berlin, 1978.
5. C. Beeri, A formal approach to object-oriented databases, *Data & Knowledge Eng*. **5**, No. 4 (1990), 353–382.
6. A. K. Chandra and D. Harel, Computable queries for relational databases, *J. Comput. System Sci.* **21** (1980), 156–178.
7. E. F. Codd, Relational completeness of data base sublanguages, *in* "Data Base Systems" (R. Rustin, Ed.), Courant Computer Science Symposium, No. 6, pp. 65–98, Prentice–Hall, Englewood Cliffs, NJ., 1972.
8. M. Consens and A. Mendelzon, GraphLog: A visual formalism for real life recursion, *in* [2, pp. 404–416].
9. M. Gyssens, J. Paredaens, and D. Van Gucht, A uniform approach toward handling atomic and structured information in the nested relational database model, (*J. Assoc. Comput. Mach.* **36**, No. 2, (1989), 790–825.
10. M. Gyssens, J. Paredaens, and D. Van Gucht, A graph-oriented object database model, *in* [2, pp. 417–424].
11. M. Gyssens, J. Paredaens, and D. Van Gucht, A Graph-oriented object model for end-user interfaces, *in* "Proceedings, 1990 ACM SIGMOD International Conference on Management of Data" (H. Garcia-Molina and H. V. Jagadish, Ed.) SIGMOD Record, Vol. 19:2 pp. 24–33, ACM Press, New York, 1990.
12. R. Hull, "Relative Information Capacity of Simple Relational Database Schemata," Technical Report TR-84-399, Computer Science Department, University of Southern California, January 1984.
13. J. Paredaens, On the expressive power of the relational algebra, *Inform. Process. Lett*. **7**, No. 2 (1978), 107–111.
14. J. Van den Bussche and J. Paredaens, On the expressive power of structured values in pure OODB's, *in* "Proceedings, Tenth ACM Symposium on Principles of database Systems," pp. 291–299, ACM Press, New York, 1991.