Clustering relations into abstract ER schemas for database reverse engineering

Pedro Sousa\textsuperscript{a,b}, Lurdes Pedro-de-Jesus\textsuperscript{a,b,*}, Gonçalo Pereira\textsuperscript{b},
Fernando Brito e Abreu\textsuperscript{b}

\textsuperscript{*}Technical University of Lisbon, Av. Rovisco Pais, 1049-001 Lisboa, Portugal
\textsuperscript{b}INESC, Rua Alves Redol 9, 1017 Lisboa, Portugal

Abstract

Database reverse engineering (DBRE) methods recover conceptual data models from physical databases. The bottom-up nature of these methods imposes two major limitations. First, they do not provide an initial high-level abstract schema suitable for use as a basis for reasoning about the application domain: a single detailed schema is only produced at the very end of the project. Second, they provide no support for a divide-and-conquer approach: the entire database schema must be analysed and processed as a unit. This paper presents a simple solution to overcome both limitations. In our proposal, relations are grouped based on their primary keys. Each group can be perceived in two ways: as a relational schema that can be reversed engineered as a standalone DBRE project; and as an element, either an entity or a relationship, of a high-level abstract schema that provides initial insight about the application domain. We also present examples from actual large database systems. © 2002 Elsevier Science B.V. All rights reserved.

1. Introduction

Database reverse engineering (DBRE) is the process of understanding legacy databases and representing their domain semantics as conceptual schemas. This is far more difficult than simple logical-to-conceptual database schema conversion, because semantic knowledge must be obtained by interpreting constructs found in different information sources, such as data definition language (DDL) statements, data manipulation language (DML), statements embedded in application source code, database extension (data values), reports, screens or forms.

\* Corresponding author.

E-mail addresses: pedro.sousa@link.pt (P. Sousa), maria.jesus@acm.org (L. Pedro-de-Jesus), goncalo.pereira@ien.siemens.de (G. Pereira), fba@inesc.pt (F.B. e Abreu).

0167-6423/02/$-see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0167-6423(02)00057-6
Many DBRE methods have been proposed in the literature [12,11,3,5,4,15,14,6,9]. Each of these methods explores different implementation constructs that can be found in different information sources. However, DBRE methods follow a traditional bottom-up methodology by nature: they gather low-level implementation constructs and try to formulate high-level concepts. In practice, DBRE methods inherit two problems associated with bottom-up methodologies [3]: no global view of the schema is produced until the very end of the project; and, there is no support for dividing the initial problem into smaller and simpler ones.

The first problem concerns the acquisition of application domain knowledge that is necessary as input for DBRE methods, either to confirm or reject proposed constructs. High-level abstract schemas have been proposed as a mean to gain a general view and to recognise the global context of a large conceptual schema [7,16,1,10,8]. Entities and relationships of abstract schemas represent sets of entities and relationships of the large conceptual schema. The existence of an initial high-level abstract schema at the beginning of a DBRE project, even if incomplete, would be very helpful to DBRE team members in understanding and reasoning about the application domain. To our knowledge, such abstract schema can only be obtained after the conclusion of a DBRE project, by clustering elements of the resulting conceptual schema into a high-level abstract schema, according to specific criteria [7,16,1,10,8].

The second problem concerns the management of complexity in large DBRE projects. DBRE methods provide no support for dividing the initial problem into smaller and independent ones. The entire database schema must be analysed and processed as a unit. However, it would be desirable to be able to split the initial relational schema into smaller schemas, reverse engineer each schema independently, and then integrate the resulting schemas. The integration may not be straightforward and will probably imply some restructuring, but the fact is that current DBRE methods provide no clues on how such division should be made to minimise the need for schema restructuring.

We present an approach to overcome both issues in relational DBMS. Section 2 presents our approach. We describe the method for clustering relations into an abstract schema; show how the abstract schema establishes a division of the initial DBRE problem into smaller ones; and describe how to merge the solutions for the smaller problems into the final solution. In Section 3 we present results from actual projects. We present related work in Section 4 and our conclusions in Section 5. Finally, we present a formal description of the clustering algorithm in Appendix A.

2. A divide-and-conquer approach

The rationale behind our proposal is to divide the complex problem of reverse engineering a large database system into simpler problems.

In Fig. 1 we present our proposal structured in three phases:

- Phase 1: Database relations are directly grouped into elements (entities and relationships) of a high-level abstract schema.
Fig. 1. A divide-and-conquer approach for reverse engineering relational databases.

- **Phase 2:** For each element of the abstract schema, the corresponding relations are reverse engineered into an intermediate conceptual schema. We call this process refinement of the abstract element. Each abstract element can be refined as a standalone DBRE project and different DBRE methods may be used in different DBRE projects.

- **Phase 3:** Intermediate conceptual schemas are integrated into a single schema and completed with missing elements. This completion is necessary because the knowledge that is possible to extract from processing the relations of each abstract element individually is less than if the whole set of relations is processed simultaneously. Basically, we are looking for foreign keys among relations that belong to different abstract elements, which result in new relationships in the final conceptual schema. To find such foreign keys we must apply a DBRE method to the whole system. As we will see in Section 2.3, conducting a DBRE method to the whole system at this stage is much simpler than it would be at the very beginning of the process.

2.1. **Phase 1: relation clustering**

The high-level abstract schema is produced in two major steps: identification of the primary key of each relation and clustering relations into entities and relationships of an abstract schema.

2.1.1. **Primary key identification**

Primary key identification consists of determining the attributes that are part of the primary key of each relation, and solving potential naming conflicts that may occur among them. Two types of naming conflicts may occur [3]: homonyms and synonyms. Homonym conflicts occur when attributes with the same name have different meanings. Synonym conflicts occur when attributes with the same meaning have different names. This step is normally a simple one because primary key attributes are among the most well documented and known ones.
2.1.2. Clustering relations into abstract entities and relationships

Relations are clustered based on the common attributes of their primary keys. In Appendix A, the detailed algorithm can be found. The clustering is made according to the following procedure:

Step 1: Identify the relations that will be clustered first: pick a set of relations whose primary keys do not contain the primary key of any other relation and are either disjoint or equal among themselves. If several solutions are possible, choose the one that produces the larger set or that uses attributes that occur more frequently.

As an example, we apply the clustering algorithm to the relations \(^1\) presented on left-hand side of Fig. 2. In this first step there are two possible sets: \{R1,R2,R3,R4\} and \{R1,R2,R3,R10\}. We choose the first one because the attributes of relation R4 are more used than those of R10.

Step 2: Cluster relations with equal primary keys in the same group. No group may contain relations with disjoint primary keys. We call these groups entities of the abstract schema or simply abstract entities. Continuing with the example, two groups are created: AE1 = \{R1,R2,R3\} and AE2 = \{R4\}.

Step 3: Add each remaining relation to a given abstract entity if at least one attribute of its primary key belongs to the entity and the remaining primary key attributes do

---

\(^1\) Notice that relation R8 has two attributes with the same name. This is just for simplicity and it means that both attributes have the same meaning.
not appear in any other entity. In this step we add R6 and R8 to AE1 and R5 and R10 to AE2.

Step 4: Create a new group for the relations whose primary key attributes belong to the same abstract entities. These groups are called relationships of the abstract schema or simply abstract relationships. In the example, a single abstract relationship is created: $AR_1 = \{R7, R9, R11\}$, because all relations have attributes that appear in AE1 and AE2 and do not appear in any other abstract entity.

The groups resulting from the clustering process have the following properties:

- The intersection of the primary keys of all relations belonging to an abstract entity is a non-empty set of attributes and constitutes its identification.
- The intersection of the primary keys of any two relations of different abstract entities is empty.
- The intersection of the primary keys of all relations belonging to an abstract relationship is at least the union of the identifiers of the associated abstract entities.

In our example, the resulting groups are presented in Fig. 2 (centre). The first of these two groups correspond to two abstract entities and the third one corresponds to an abstract relationship between them. On the right-hand side, we present the corresponding abstract schema.

A simple way to show the meaning of the abstract schema is to compare it with a conceptual schema (in entity relationship model) obtained from reverse engineering the same set of relations. In Fig. 3, we present both the abstract and the conceptual schemas that are obtained by processing the relations presented in Fig. 2. The entities
and relationships of the conceptual schema are named after the relations that implement them.\textsuperscript{2}

The layout of the conceptual schema was chosen to make obvious the correspondence between the entities and relationships of the conceptual and abstract schemas. Our goal is to show intuitively that the information necessary to refine an abstract entity exists within itself. In our example, to refine abstract entity EA1, one needs only to apply a DBRE method to relations R1, R2, R3, R6 and R8. In the next section we present the refinement phase in more detail.

2.2. Phase 2: refinement of abstract elements

In this section, we show that relations that implement concepts such as strong entities, weak entities, generalisations, relationships and aggregations are grouped in the same abstract element.\textsuperscript{3} Therefore, such concepts can be found during the independent reverse engineering of the relations of each abstract element, regardless of the DBRE method used.

We begin with abstract entities. For each ER modelling concept we show that, if it exists within an abstract entity, it can be recognised solely by processing its relations.

- **Strong entities**: recognition of a strong entity occurs whenever there is a relation with a primary key that does not contain primary keys of any other relations [3,5] Therefore, to properly recognise a strong entity\textsuperscript{4} one must process all relations with equal or included primary keys. Since these relations belong necessarily the same abstract entity, strong entities can be found by processing only relations of corresponding abstract entities.

- **Generalisation hierarchies**: recognition of a generalisation hierarchy is based on the existence of relations with equal primary keys [13,3]. Since these relations belong to the same abstract entity, any generalisation hierarchies that may exist in an abstract entity can be found processing its relations.

- **Weak entities**: recognition of weak entities is based on the existence of a relation with a primary key formed by one or more primary keys of other relations and by extra attributes that do not appear as key attributes elsewhere [3,5]. Since the intersection of primary keys of relations belonging to different abstract entities is always empty, only the weak entities that are dependent of a single strong entity\textsuperscript{5} can exist in an abstract entity. Therefore, all relations needed to discover a weak entity belong to a single abstract entity.

- **Relationships**: we have to distinguish two types of relationships: those implemented as a relation, normally a many-to-many relationship; and those implemented using buried foreign keys, normally one-to-many and one-to-one. Recognition of the first type of relationships is based on the existence of relations whose primary key is composed exclusively by the concatenation of primary keys

\textsuperscript{2} Notice that entity named E-Kc has no direct implementation on any relation.

\textsuperscript{3} With a few exceptions that are resolved in phase 3.

\textsuperscript{4} Regardless the number of relations that were used to implement it.

\textsuperscript{5} Or several strong entities under the same generalisation hierarchy.
of strong or weak entities [3,5,6]. Since there is no common sub-set of primary key attributes between any two relations from different abstract entities, all relationships in an abstract entity must refer to entities in the same abstract entity, and therefore can be properly recognised.

Recognition of the second type of relationships is based on the recognition of the foreign keys. These relationships can only be found during refinement of abstract entities if both referring and referred entities belong to the same abstract entity. If not, they can only be found in the third phase of the process.

- Aggregations: aggregations are recognised by the existence of a foreign key to a relation that implements a relationship [5]. Therefore, only aggregations whose foreign key refers to weak or strong entities of the same abstract entity can be discovered in this stage.

A similar reasoning can be made to show that the set of relations that belong to each abstract relationship can be reverse engineered processing only the relations of that abstract relationship together with the relations of the associated abstract entities. In fact, since relations of an abstract relationship can only refer to relations from the participating abstract entities, relations from any other abstract elements are not necessary.

It is worth mentioning that our clustering is particularly robust to optimisations and implementation decisions, namely those concerned with the third normal form, because any combination or permutation of non-key attributes among a set of relations yields the same abstract schema. Partitioned tables, either vertically or horizontally, can also be recognised within the same abstract entity or relationship because all fragments of that table have the same key, therefore belong to the same abstract entity.

2.3. Phase 3: completion of the final schema

In the third phase we integrate the different intermediate conceptual schemas into a final schema and complete it with missing concepts.

The integration of the different intermediate conceptual schemas is straightforward because no concept restructuring is required. In fact, elements of intermediate schemas resulting from abstract entities are disjoint, and elements of intermediate schemas resulting from abstract relationships need only be associated with the corresponding elements of the abstract entities.

Since all primary keys have been already processed, the missing concepts can only result from foreign keys that are not simultaneously primary keys of any relation. Furthermore, since those concepts were not captured while processing abstract elements independently, they can only be:

- relationships that may exist between entities from different abstract elements.
- aggregations that may exist between relationships from different abstract elements.

Since the entities that participate in the relationships and aggregations were already established in the schema during the second phase, completion is not destructive. In fact, in the vast majority of the cases, we simply add relationships to the schema. To
be able to find such concepts, it is necessary to apply a DBRE method to the entire database schema. However, the effort in conducting such project is much simpler at this stage of the project because much of the work is already done, application domain is better understood and we are only looking for two specific concepts.

3. Experience in real DBRE projects

We have used our approach in actual reverse engineering projects, and have benefited from the advantages already described, most of them consequences of an early abstract schema production and an incremental application of DBRE methods.

We present the application of our approach to a large legacy database system for customer billing services in a telecommunications company, containing 261 database tables, from which 91 are related to customer data and the remaining 170 are related to reference and processing options.

3.1. Phase 1: relation clustering

In Fig. 4 we present the abstract schema obtained from clustering only the customer-related tables. We have omitted seven tables that were not connected with other elements.

The quality of the resulting abstract schema can be evaluated by the semantics of abstract elements. We can confirm that the abstract schema has meaningful elements, each aggregating tables with a common subject area, as described next:

- E1 represents error information.
- E2 represents transactions related to payments processing and adjustments.
- E3 represents information related to customer accounts, sub-accounts, products and services.
- E4 represents the transaction used to change the length of the directory system.
- E5 represents the splitting of files in packets for processing.
- E6 represents those accounts that are going to be retyped in the formatting process.
- E12 represents accounts that were not processed during the normal billing cycle.
• E13 represents current batch information for an online user (teller).
• E14 represents special locations.
• R15 represents a bill’s printing and retyping.
• R16 represents the control of extraction of packets for processing.
• R17 represents a bill’s correctness verification.
• R18 represents the occurrence of an error in the processing of a directory advertising transaction.
• R19 represents information that has been entered through the online entry system at account and sub-account levels.
• R20 represents the audit of the transactions made by a specific teller.

This abstract schema was used for initial understanding of application domain knowledge and for establishing smaller DBRE projects.

3.2. Phase 2: refinement of abstract elements

In the second phase, relations of each abstract entity are processed independently into conceptual schemas, using the entity relationship model. Here, we present only the conceptual schemas obtained from reverse engineering relations of abstract elements E3, R15 and E6 of Fig. 4. Each grey box of Fig. 5 represents a conceptual schema. We do not describe the DBRE method used to reverse engineering each abstract element since it is outside the scope of this paper. Suffice is to say that they were processed as independent DBRE projects.

The apparent small number of concepts produced obtained from the 47 relations of abstract entity E3 resulted from the fact that many relations were used to implement vertical fragmentation, and do not correspond to new concepts.

The meaning of the entities of presented in the conceptual schemas of Fig. 5 is as follows:

• E1 represents the accounts for domestic households.
• E2 represents the accounts for business.
• E3 represents the accounts for services internal to the operator.
• E4 represents the accounts for government use.
• E5 represents the remarks related to subscription of services.
• E6 represents service orders used for changing the products/services used.
• E7 represents additional addresses a customer can have.
• E8 represents the customer.
• E9 represents a customer sub-account; each sub-account represents a different service.
• E10 represents the format used for producing the bill.
• E11 represents a pricing plan.
• E12 represents products and services.
• E13 represents products and service providers.
• E14 represents a bank account used for paying the bill.
• E15 represents a termination point (e.g. a telephone number) involved in a pricing plan.
Fig. 5. Refinement of abstract elements E3, R15 and E6.

- E16 represents the interval during which the plan is valid.
- E17 represents amounts of money flowing between the company and customers.
- E18 represents security deposits made by customers.
- E19 represents the amounts customers pay to the company.
- E20 represents amounts the company is owing to customers.
- E22 represents the aggregation of several bills for the production of a summary bill.
- E32 represents the bill prepared for printing.
- E33 represents the corrections made to a bill in the retyping process.
- E39 represents the retyped bill.

3.3. Phase 3: completion of the final schema

As an example of the application of the third step of our methodology, we present the result of processing the relations of abstract elements E3, R15 and E6 in a single DBRE project. The new concepts found are the two standalone relationships in Fig. 5:

- relationship R38 is a one-to-many relation an represents an aggregate for the production of a summary bill.
relationship R40 is a one to one conditional relation and represents the changes made to summary bills that were retyped.

As expected the completion of the final schema did not imply any modification of the intermediate schemas but the addition of the concepts found.

3.4. Limitations of the approach

We can also observe limitations of our clustering approach, namely the lack of control over the granularity of the abstract elements: some abstract elements are composed a single table, whereas others have many tables.

In the case being studied, abstract entity E3 of Fig. 4 represents one big cluster, because all tables share a common attribute. However, we can further decompose this entity by applying the clustering method again but considering only tables from that group and removing the attribute causing the aggregation. The resulting abstract schema is presented in Fig. 5, where 17 tables were removed because they were not connected with other tables.

Once again we can confirm that clustered tables have common semantics.

- E7 represents service order information tracking for each customer.
- E8 represents payments processing and adjustments.
- E9 represents amounts and charges due, from customers to product service providers.
- E10 represents the notion of sub-account.
- R11 represents adding or dropping a product by a customer.
- R12 represents services provided to customers.
- R13 represents the changing of the primary provider for a specific customer, through a service order.
- R14 contains the information relating to the coin collection made at a pay-phone.
- R15 represents an adjustment due to a product service provider from a non-sufficient funds payment made by a customer.
Another limitation, is the non-determinism of the Clustering Algorithm, namely in step 1. That means that relations can be clustered in different ways. The immediate impact is on the order relations are processed and therefore on the order concepts are found. More experiences will be conducted to determine the impact on large DBRE projects.

4. Related work

To the best of our knowledge, no attempt has been made so far to produce abstract ER schemas directly from database relations. Although this idea has been mentioned in [10], no actual clues were given on how to perform such clustering nor were requirements stated.

We can relate our work to the work done in ER clustering. Our algorithm has identification as a single criterion to produce abstract entities and relationships, whereas most clustering algorithms proposed the use of additional criteria [7,16,1,10,8]. Nevertheless, Teorey’s notion of dominance and abstraction grouping for clustering entities and relationships into more abstract entities is similar to ours, if relationship cardinality is ignored. We cluster generalisation hierarchies as in Teorey’s abstraction grouping, and we can also cluster weak entities that depend on a single strong entity with the corresponding strong entity, as in Teorey’s dominance grouping. Jäeschk has proposed the notion of complex relationship clustering as a criterion to cluster relationships, entities and aggregations into more abstract relationships [10]. This is similar to our abstract relationships, where we may find everything but strong entities.

The use of clustering techniques in database reengineering was already proposed [10,8]. The idea is to cluster existing ER schemas into high abstraction schemas in a bottom-up activity and then redesign each element of the high abstraction schema through a top-down refinement. However, they start from conceptual schemas and the lack of application domain knowledge was not an issue in their work, whereas we start from the relational schema and consider the lack of application domain knowledge a major issue.

We can also relate our work to some aspects of DBRE methods, in particular the processing of primary keys. At a first glance, our clustering approach could be judged equivalent to the simpler classification of relations into strong entities, weak entities and relationships [3,5] followed by their clustering into abstract schemas. Our work is different in that we are able to cluster all relations regardless of any optimisation or design decisions that may have been done. In fact, in the systems we analysed, about 20% of the relations could not be classified as above. Others [5] have proposed a more complex classification template, but it requires application domain experts to distinguish between weak entities and specific relationships.

---

6 In fact, we always cluster weak entities with the corresponding strong entities, regardless of their number, as long as all corresponding strong entities belong to the same group.
5. Conclusions

We presented a method for clustering database schemas directly into abstract conceptual schemas with a minimum of input information. This allows to produce abstract schemas without needing to gather the whole set of detailed information that is required to produce a full detailed conceptual schema from a database schema. To the best of our knowledge, no such attempt has yet been made.

The main advantages of our approach are:

- The production of a high-level abstract schema simplifies the understanding of the application domain, because it can be used as a basis for reasoning by DBRE team members. Even when such a schema already exists, our abstract schema is still useful because it is pinned down to actual tables, whereas such a correspondence is difficult to establish in existing high-level schemas.
- DBRE projects become simpler: on the one hand, problem complexity and size are smaller; on the other hand, users acquire more knowledge about the application domain: the problem of reverse engineering becomes inherently simpler.
- Different DBRE methods can be applied to different sets of relations. This is of the utmost importance when the information about the database schema is not homogeneously distributed: for instance, for some relations the candidate keys are known while for others it is known that there is a coherent naming of their attributes. Thus, we are able to choose the most appropriated method for each set of relations.
- Our approach eases project management because it allows different team members to work in parallel on the refinement of different abstract elements and to specialise in different areas of the application domain.

We also envisage the use of abstract schemas for reengineering. In fact, the refinement of the abstract schema can be done either towards the reverse engineering or towards the forward engineering of the system. In the former, the conceptual schema should be compatible with the database schema, and in the latter should be compatible with the new requirements.

Appendix A

In this appendix we present a formal description of the clustering algorithm described in Section 2.1.2. It is written using Sara Baase’s notation [2].

We assume that:

OrderAscPk(S):

given a set of relations S, returns a list with the relations ordered such that the two following properties hold:

1. ascending order regarding the cardinality of the primary keys (first relations with 1 attribute in the primary key, then with 2, and so on);
2. relations having the same primary key are in consecutive positions in the list.
Since there may be different lists satisfying these properties, the function is not deterministic.

$pk(R)$ denotes the primary key of relation $R$.

Through the algorithm, the following variables will be used: $\text{remaining.rels}$, the set of relations that are still remaining to be handled; $\text{cluster}$, the list of sets of relations that is going to be instantiated, throughout the algorithm, with the Abstract Entities and Relationships; $nes$ the Number of Abstract Entities and $nas$ the Number of Abstract Elements (Entities or Relationships).

### A.1. Relations clustering algorithm

**Input:** $rels$, the set of relations to be clustered (with their primary keys defined).

**Output:** $nes$, $nas$ and $\text{cluster}$, such that $\text{cluster}[I]$ to $\text{cluster}[nes]$ are the Abstract Entities and $\text{cluster}[nes+I]$ to $\text{cluster}[nas]$ are the Abstract Relationships.

```plaintext
var remaining.rels: Set of Relations;
cluster: List of Set of Relations;
nes : int;
nas : int;
```

### A.2. Steps 1 and 2

**Input:** $rels$.

**Output:** (i) $nes$ = Number of Abstract Entities formed; (ii) Abstract Entities ($\text{cluster}[I]$ to $\text{cluster}[nes]$) formed with the relations that are clustered first (relations $R$ and $S$ are clustered first only if $(pk(R) = pk(S)) \lor (pk(R) \cap pk(S) = \emptyset)$; (iii) $\text{remaining.rels}$.

```plaintext
var disjoint : boolean;
ordered.rels: List of relations;
remaining.rels := rels;
ordered.rels := OrderAscPk (rels);
insert ordered.rels[I] into cluster[I];
remove ordered.rels[I] from remaining.rels;
nes := 1;
for i := 2 to Length(ordered.rels) do
  R := ordered.rels[i];
  if pk(R) = pk(ordered.rels[i - 1])
    then insert R into cluster[nes];
    remove R from remaining.rels
  else disjoint := true;
    for each $S \in \bigcup_{1 \leq j \leq nes} \text{cluster}[j]$ do
      if $pk(R) \cap pk(S) \neq \emptyset$
```

\[ \text{then disjoint := false fi od} \]
\[ \text{if disjoint = true} \]
\[ \quad \text{then nes := nes + 1;} \]
\[ \quad \text{insert } R \text{ into } \text{cluster}[nes]; \]
\[ \quad \text{remove } R \text{ from } \text{remaining_rels fi} \]
\[ \text{fi} \]
\[ \text{od} \]

A.3. Step 3

Input: \textit{remaining_rels} and Abstract Entities (output of steps 1 and 2)

Output: The same Abstract Entities but, eventually, with more relations inserted into them; \textit{remaining_rels}

\[ \text{for each } R \in \text{remaining_rels do} \]
\[ \quad i := 1; \]
\[ \quad \text{clustered := false}; \]
\[ \quad \text{while } (i \leq \text{nes} \land \neg \text{clustered}) \text{ do} \]
\[ \quad \quad \text{if } ((pk(R) \cap (\bigcup_{S \in \text{cluster}[i]} pk(S))) \neq \emptyset \land \]
\[ \quad \quad \quad (pk(R) \cap (\bigcup_{S \in \text{cluster}[i], j \neq i, 1 \leq j \leq \text{nes}} pk(S))) = \emptyset) \]
\[ \quad \quad \text{then insert } R \text{ into } \text{cluster}[i]; \]
\[ \quad \quad \text{remove } R \text{ from } \text{remaining_rels}; \]
\[ \quad \quad \text{clustered := true fi od od} \]

A.4. Step 4

Input: \textit{remaining_rels} and Abstract entities (output of step 3)

Output: \textit{nas} and Abstract relationships (\textit{cluster}[nes+1] to \textit{cluster}[nas])

\[ \textbf{var} \ argument:\text{Bidimensional Matrix of boolean,}^7 \]
\[ \text{intersects}: \text{List of boolean}; \]
\[ 
\text{nas} := \text{nes + 1}; 
\]
\[ \text{first\_relationship} := \text{true}; \]
\[ \text{for each } R \in \text{remaining_rels do} \]
\[ \quad \text{for } i := 1 \text{ to } \text{nes do} \]
\[ \quad \quad \text{if } \quad pk(R) \cap (\bigcup_{S \in \text{cluster}[i]} pk(S)) \neq \emptyset \]
\[ \quad \quad \quad \text{then intersects[i] := true fi od} \]
\[ \quad \text{if first\_relationship} \]
\[ \quad \quad \text{then for } i := 1 \text{ to } \text{nes do} \]
\[ \quad \quad \quad \text{argument}[1,i] := \text{intersects[i] od;} \]
\[ \quad \text{insert } R \text{ into } \text{cluster}[nas]; \]

---

\[ ^7 \text{Argument}[i,j] \text{ is true if Abstract Entity of } \text{cluster}[j] \text{ is an argument of Abstract Relationship of } \text{cluster}[i], \text{ for } 1 \leq i \leq \text{nas} - \text{nes}, 1 \leq j \leq \text{nes.} \]
remove R from remaining_rels;
first_relationship := false

else
j := 1;
found := false;
while j ≤ nas ∧ ¬found do
  if \( \bigwedge_{1 \leq i \leq nes} \) (intersects[i] = argument[j, i])
    then insert R into cluster[j];
    remove R from remaining_rels
    found := true fi
  j := j + 1 od
if ¬ found
then nas := nas + 1;
for i := 1 to nes do
  argument[nas – nes, i] := intersects[i] od;
insert R into cluster[nas];
remove R from remaining_rels fi
od

References


