From Extended Entity-Relationship schemata to illustrative instances

Maria Amalfi and Alessandro Provetti
Physics, Univ. of Messina
M^2AG: Milan-Messina Actions Group

please visit the companion Web site:
http://mag.dsi.unimi.it/EER2instances

May 21, 2008

Illustrative Instances

long-term research topic:

apply logic programming to automated reasoning about projects diagrams: UML and [E]ER.

automated snapshot generation supports the design activity where a formal validation would not be easily available.

Illustrative Instances

long-term research topic:

apply logic programming to automated reasoning about projects diagrams: UML and [E]ER.

automated snapshot generation supports the design activity where a formal validation would not be easily available.

Ornaghi et al. (Milan) work on the validation of UML models, by automated snapshot generation

Illustrative Instances

long-term research topic:

apply logic programming to automated reasoning about projects diagrams: UML and [E]ER.
automated snapshot generation supports the design activity where a formal validation would not be easily available.
Ornaghi et al. (Milan) work on the validation of UML models, by automated snapshot generation

Goal of this work:

Given a set of constraints representing an EER project, generate automatically an example instance showing how the final database will (likely) look like.

Illustrative Instances

long-term research topic:

apply logic programming to automated reasoning about projects diagrams: UML and [E]ER.
automated snapshot generation supports the design activity where a formal validation would not be easily available.
Ornaghi et al. (Milan) work on the validation of UML models, by automated snapshot generation

Goal of this work:

Given a set of constraints representing an EER project, generate automatically an example instance showing how the final database will (likely) look like.
Automated reasoning about project constraints: (relatively) easy to add them to our LP representation
both directions of research showcase DLV*

Informative Armstrong Data Bases (IADBs)

Fabien De Marchi and Jean-Marc Petit
Semantic sampling of existing databases through informative Armstrong databases
Information Systems, 2007.

De Marchi et al. aim to provide information about the structure of the database by letting the user inspect a suitably small instance.
The object database may be very large, legacy indeed hard to visualize.

Informative Armstrong Data Bases (IADBs)

Fabien De Marchi and Jean-Marc Petit
Semantic sampling of existing databases through informative Armstrong databases
Information Systems, 2007.

De Marchi et al. aim to provide information about the structure of the database by letting the user inspect a suitably small instance.
The object database may be very large, legacy indeed hard to visualize.

Informative Armstrong Database

A *small* instance that satisfies exactly the same functional and inclusion dependencies that were found on the given (normally large) database instance.
IADBs consist of *real tuples* that come from an input database which we want to *understand*

Literature: Armstrong Databases

Literature: Armstrong Databases

[Fagin-Vardi]

An Armstrong Database is a *global Witness* from a *relational* schema.

Literature: Armstrong Databases

[Fagin-Vardi]

An Armstrong Database is a *global Witness* from a *relational* schema.

1. Let Σ a set of constraints over the schema
2. Let Σ^* the set of logical consequences of Σ
3. An instance Θ is an ADB if
   a. $\forall\sigma \in \Sigma^*.\ \Theta \models \sigma$
   b. $\forall\sigma \notin \Sigma^*.\ \Theta \not\models \sigma$

Example (Fagin-Vardi): ADB

$\Sigma = \{EMP \rightarrow DEPT, DEPT \rightarrow MGR\}$

Employee	Dept	Manager
Hilbert	Math	Gauss
Pythagoras	Math	Gauss
Turing	Computer Science	von Neumann
Einstein	Physics	Gauss

Table: Armstrong Relation r for Σ

Σ^* contains the FD's in Σ

The FD $MGR \rightarrow DEPT$ is not an FD in Σ^*, and indeed, r does nor obey this FD, since GAUSS is the MANAGER of two distinct departments (MATH and PHYSICS)

Example: EER schema

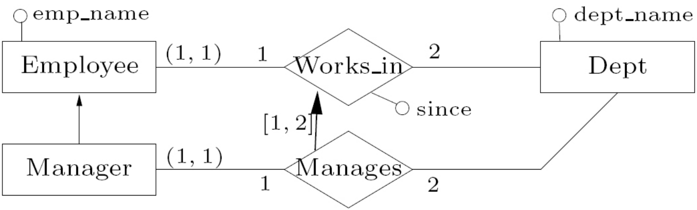
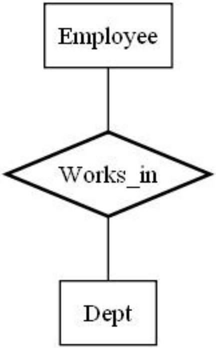


Figure: Extended Entity Relationship schema

The schema describes employees (entity EMPLOYEE) working (relationship WORKS_IN) in departments (relationship DEPT) of a firm, and managers (entity MANAGER) that are also employees, and manage (relationship MANAGES) departments. Managers who manage a department also work in the same department, as imposed by the *is-a* among the two relationships.

FROM EER to ERL




```
<!DOCTYPE erl PUBLIC "-//DSI//DTD ERL V1.2//EN" "file:./ERL/erl.dtd">
<erl id="Example1" title="Example 1 from [Cali, 2007]">

  <ent id="employee" label="Employee">
    <attr id="empname" label="emp_name" size="30" mand="true"/>
  </ent>

  <ent id="dept" label="Dept">
    <attr id="deptname" label="dept_name" type="int" mand="true"/>
  </ent>

  <rel id="worksin" label="Works_in">
    <attr id="since" label="since" type="date" mand="true"
          default="'2001-01-01'"/>
    <leg entref="employee" label="1"/>
    <leg entref="dept" label="2"/>
  </rel>

</erl>
```

A new DTD for handling *is-a* between relationship

Step-by-step generation of the logic program

Given an EER $\mathcal{C}$ our translation creates a Logic program $\pi_{\mathcal{C}}$ with a step-by-step process that applies 15 translation rules.

Step-by-step generation of the logic program

Given an EER $\mathcal{C}$ our translation creates a Logic program $\pi_{\mathcal{C}}$ with a step-by-step process that applies 15 translation rules.

Translation

- **A..D** insert into $\pi_{\mathcal{C}}$ facts and rules that capture the structure of *a* relational schema induced by $\mathcal{C}$;
- **1..11** insert into $\pi_{\mathcal{C}}$ rules that capture two essential aspects of EER:
  - inclusion dependencies, captured by logic programming constraints (on consistency);
  - the inheritance relation between entities (resp. relationships) which is captured by normal deductive rules.

Step-by-step generation of the logic program

Given an EER $\mathcal{C}$ our translation creates a Logic program $\pi_{\mathcal{C}}$ with a step-by-step process that applies 15 translation rules.

Translation

- **A..D** insert into $\pi_{\mathcal{C}}$ facts and rules that capture the structure of *a* relational schema induced by $\mathcal{C}$;
- **1..11** insert into $\pi_{\mathcal{C}}$ rules that capture two essential aspects of EER:
  - inclusion dependencies, captured by logic programming constraints (on consistency);
  - the inheritance relation between entities (resp. relationships) which is captured by normal deductive rules.

Implemented as a ERW interface:

```
>java -cp ER2DL.jar it.unimi.dsi.erw.ERtool < my-eer-project.xml
--datalog
```

From EER to Logic Programs: Modifying [Calì 07]

The semantics of an EER schema is defined by specifying all constraints imposed by EER constructs on databases that satisfy that schema.

From EER to Logic Programs: Modifying [Calì 07]

The semantics of an EER schema is defined by specifying all constraints imposed by EER constructs on databases that satisfy that schema.

First of all, we formally define a database schema from an EER diagram. Such a database schema is defined in terms of predicates: we define a relational database schema that encodes the properties of an EER schema $\mathcal{C}$.

A. Each entity E in $\mathcal{C}$ has an associated predicate E of arity 1.

B. Each attribute A for an entity E in $\mathcal{C}$ has an associated predicate A of arity 2, associating attribute values to entity instances.

C. Each relationship R among the entities $E_1, \ldots, E_n$ in $\mathcal{C}$ has an associated predicate R of arity n.

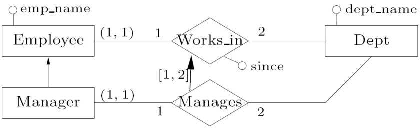
D. Each attribute A for a relationship R among the entities $E_1, \ldots, E_n$ in $\mathcal{C}$ has an associated predicate A of arity n + 1, associating attribute values to n-tuples of entity instances.

Cont'd

Given a database schema R for an EER schema $\mathcal{C}$, Calì gives a semantics to each construct of the EER model by specifying what databases (i.e. extension of the predicates of R) satisfy the constraints imposed by the constructs of the EER diagram.

EER construct	**Relational constraint**
attribute $A/2$ for an entity E	$A[1] \subseteq E[1]$
attribute $A/(n+1)$ for a relationship R/n	$A[1,\ldots,n] \subseteq R[a,\ldots,n]$
relationship R with entity E as i-th component	$R[i] \subseteq E[1]$
mandatory attribute $A/2$ of entity E	$E[1] \subseteq A[1]$
mandatory attribute $A/(n+1)$ of relationship R/n	$R[1,\ldots,n] \subseteq A[1,\ldots,n]$
functional attribute $A/2$ of entity E	$key(A) = \{1\}$
functional attribute $A/(n+1)$ of a relationship R/n	$key(A) = \{1,\ldots,n\}$
is-a relation between entities E_1 and E_2	$E_1[1] \subseteq E_2[1]$
is-a relation between relationships R_1 and R_2, where components $1,\ldots,n$ di R_1 correspond to components $j_1,\ldots,j_n$ of R_2	$R_1[1,\ldots,n] \subseteq R_2[j_i,\ldots,j_n]$
mandatory participation as c-th component of an entity E in a relationship R	$E[1] \subseteq R[i]$
functional participation as c-th component of an entity E in a relationship R	$key(R) = \{c\}$

Our method, By Example



manager/1,
employee/1,
dept/1,
works_in/2,
manages/2,
emp name/2,
dept name/2,
since/3

Manager[1] ⊆ Employee[1]
Manages[1] ⊆ Employee[1]
Manages[2] ⊆ Dept[1]
Works_In[1] ⊆ Employee[1]
Works_In[2] ⊆ Dept[1]
Manages[1,2] ⊆ Works_In[1,2]

Step **A..D**:

Step A.
```
employee(emp).
manager(man).
dept(dept).
```

Step B.
```
emp_name(emp,emp_name).
dept_name(dept,dept_name).
```

Step C.
```
works_in(emp,dept).
manages(man,dept).
```

Step D.
```
since(emp,dept,since).
```

Step **1..11**:
capture special constraints (*FDs*, *IDs* e *KDs*).

Example 1: Defining the constraints

Step 1: emp_name[1] ⊆ employee[1]:

```
:- emp_name(Emp, Emp_name),
   not employee(Emp).
```

Step 2: since[1,2] ⊆ works_in[1,2]:

```
:- since(Emp, Dept, Since),
   not works_in(Emp, Dept).
```

Step 3: works_in[1] ⊆ employee[1]:

```
:- works_in(Emp, Dept), not employee(Emp).
```

Step 4: employee[1] ⊆ emp_name[1]:

```
:- employee(Emp), not
   emp_name(Emp, emp_name_of(Emp)).
```

Step 5: works_in[1,2] ⊆ since[1,2]:

```
:- works_in(Emp, Dept), not
   since(Emp, Dept, works_in_since(Emp, Dept)).
```

Step 6: does not apply to this example (it would simply generate a key constraint)

Step 7: does not apply to this example (it would simply generate a key constraint)

Step 8: Each manager is also an employee:

```
employee(Man) :- manager(Man).
```

Step 9: Each occurrence of Manages is also an occurrence of Works_in:

```
works_in(Emp, Dept) :- manages(Emp, Dept).
```

Step 10: employee[1] ⊆ works_in[1]:

```
:- employee(Emp),
   not participates(Emp, works_in, 1).
participates(Emp, works_in, 1) :-
            dept(Dept), works_in(Emp, Dept).
participates(Dept, works_in, 2) :-
            employee(Emp), works_in(Emp, Dept).
```

Step 11: key(works_in) = employee[1]:

```
:- works_in(Emp1, Dept1),
   works_in(Emp2, Dept2),
   Emp1 = Emp2,
   Dept1 != Dept2.
```

Interpretation of the Logic Program

F. Calimeri, S. Cozza, G. Ianni and N. Leone
Functions, Lists and Set in DLV System
CoRR Technical Report, forthcoming. 2008

The *target language* of the mapping described in the previous Section is that of the novel DLV^* inferential engine.

Interpretation of the Logic Program

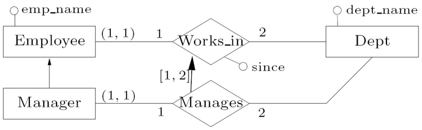
F. Calimeri, S. Cozza, G. Ianni and N. Leone
Functions, Lists and Set in DLV System
CoRR Technical Report, forthcoming. 2008

The *target language* of the mapping described in the previous Section is that of the novel DLV^* inferential engine.

DLV* is an extension of the well-know DLV inferential engine[a] for deductive databases and disjunctive logic programming which supports functions, lists and sets.

[a]DLV and its extensions, and related literature are available from *http://www.dlvsystem.com/*.

Finally...



Relationship predicates reification are reified

```
-->dl my-eer-project.dlv

DLV [build BEN/Feb  8 2008   gcc 3.4.2 (mingw-s
{
 manages(man,dept),
 employee(emp),
 employee(man),
 manager(man),
 dept(dept),
 emp_name(emp,emp_name_of(emp)),
 emp_name(man,emp_name_of(man)),
 dept_name(dept,dept_name_of(dept)),
 works_in(emp,dept),
 works_in(man,dept),
 since(emp,dept,works_in_since(emp,dept)),
 since(man,dept,works_in_since(man,dept)),
 participates(emp,works_in,1),
 participates(man,works_in,1),
 participates(man,manages,1),
 participates(dept,works_in,2),
 participates(dept,manages,2)
}
```

Results

Conceptual

Automated generation of illustrative instances for EER schema:
It assists Database design exp. w.r.t. the non-expert.

Results

Conceptual

Automated generation of illustrative instances for EER schema:
It assists Database design exp. w.r.t. the non-expert.

Technical

Modified XML representation of EER projects.
Modified translation from EER to Logic Programming
First-time application of DLV*

Results

Conceptual

Automated generation of illustrative instances for EER schema:
It assists Database design exp. w.r.t. the non-expert.

Technical

Modified XML representation of EER projects.
Modified translation from EER to Logic Programming
First-time application of DLV*

Integration into ERW

1. Input: EER project in ERL encoding
2. Output:
   2.1 DLV* representation, an illustrative instance;
   2.2 SQL commands for DB creation, ready-made PHP pages and forms needed for populating the database and for maintenance.

Bibliography

William Ward Armstrong and Claude Delobel
Decompositions and Functional Dependencies in Relations
ACM Trans. on Database Systems (TODS). 1990

Andrea Calì
Querying Incomplete Data with Logic Programs: ER Strikes Back
Proc. of ER 2007 Conference

Peter P. Chen
The Entity-Relationship Model: Toward a Unified View of Data
TODS. 1976

Ronald Fagin and Moshe Y. Vardi
Armstrong Databases for Functional and Inclusion Dependencies
Information Processing Letters. 1983

Georg Gottlob and Leonid Libkin
Investigations on Armstrong Relations, Dependency Inference and Excluded Functional Dependencies
Acta Cybernetica, 1990

Sebastiano Vigna
Reachability Problems in Entity-Relationship Schema Instances
ER. 2004