Data & Knowledge Engineering 87 (2013) 260–278

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Editorial

Data migration: A theoretical perspective ☆

Bernhard Thalheim [a], Qing Wang [b,*]

[a] Department of Computer Science, Christian-Albrechts-University Kiel, Germany
[b] Research School of Computer Science, The Australian National University, Australia

ARTICLE INFO

Article history:
Accepted 25 September 2012
Available online 23 December 2012

Keywords:
Data migration
Refinement
Transformation
Legacy system
ETL

ABSTRACT

In this paper we investigate data migration fundamentals from a theoretical perspective. Following the framework of abstract interpretation, we first discuss models and schemata at different levels of abstraction to establish a Galois connection between abstract and concrete models. A legacy kernel is discovered at a high-level abstraction which consolidates heterogeneous data sources in a legacy system. We then show that migration transformations can be specified via the composition of two subclasses of transformations: property-preserving transformations and property-enhancing transformations. By defining the notions of refinement correctness for property-preserving and property-enhancing transformations, we develop a formal framework for refining transformations occurring in the process of data migration. In order to improve efficiency of static analysis, we further introduce an approach of verifying transformations by approximating abstraction relative to properties of interest, meanwhile preserving the refinement correctness as accurately as possible. The results of this paper lay down a theoretical foundation for developing data migration tools and techniques.

1. Introduction

Modernising legacy systems is one of the most challenging problems we often face when engineering information systems [7,8,27,30]. With new technologies emerging and application domains evolving, legacy systems need to be migrated into new systems at some point, to support enhanced functionality and re-engineered business models. Data migration, as a fundamental aspect of projects on modernising legacy systems, has been recognised to be a difficult task that may result in failed projects as a whole [23,38]. Industry survey results [23] reveal that the data migration market is rapidly growing and business companies annually invest billions of dollars in data migration tasks (e.g., over 5 bn from the top 2000 global companies in 2007); nevertheless, only 16% of projects successfully accomplish their data migration tasks (i.e., being delivered on time and on budget) — 64% of data migration projects failed to be delivered on time and 37% were over-budget. A main reason for time and budget overrun is the lack of a well-defined methodology that can help handle the complexity of data migration tasks.

In general data migration is the process of moving data from legacy data sources of a legacy system into new data sources of a target system, in which legacy and new systems have different data structures. There are several issues that may considerably complicate this process. First, legacy systems often have a number of heterogeneous data sources designed by using different data modelling tools or interpreted under different semantics. This requires a thorough understanding of legacy data sources from various aspects, such as explicit or implicit data constraints, interrelationships across different data sources, and data availability. Second, legacy systems may have inaccurate, incomplete, duplicate or inconsistent data. On the other side, new systems often require additional semantic constraints on data after being migrated. Thus, bringing the quality of data up to the standard of new systems can be costly and time-consuming. A previous study [1] showed that 62% of data migration projects have significant data

☆ A preliminary version of this paper was presented at the 30th International Conference on Conceptual Modeling (ER 2011), Brussels, Belgium.

* Corresponding author.
E-mail addresses: thalheim@is.informatik.uni-kiel.de (B. Thalheim), qing.wang@anu.edu.au (Q. Wang).

0169-023X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.datak.2012.12.003

quality problems in new systems. Third, many data migration tasks such as data profiling, validating, and cleansing need to be iteratively executed in a project and specification changes frequently happen in order to repair detected problems. It is estimated [1] that 90% of the initial specifications change and over 25% of the specifications change more than once during the life of a data migration project. These issues highlight the importance of methodologies and best practice approaches that can be used to guide through the process of data migration.

In this paper we investigate data migration fundamentals from a theoretical perspective. In particular, we aim to develop a general refinement scheme for migration transformations, linking high-level specifications to executable codes in a way practitioners can systematically verify properties of data migration. Our study provides answers to the following questions arising from data migration in practice.

- How can we control data quality within a data migration process to prevent the propagation of "dirty" data into a new system, which often requires external data sources to be supplied for cleansing and enriching data?
- How can we react to specification changes in a way that keeps track of all relevant aspects on which the changes may impact, such as inconsistencies between specifications, interrelated data and correctness of implementation?
- How can we compare legacy data sources with the migrated data in new systems to ensure that data was migrated properly in terms of desired data semantics and integrity?

Data migration is usually undertaken as part of a larger project in modernising legacy systems (e.g., a system migration or upgrade project). In such cases, transformations on data may lead to transformations on views, functionality and user interfaces [27]. Nevertheless, we keep the scope of this paper restricted to transformations on data models and their corresponding schemata.

1.1. Our contributions

The first contribution in this paper is the understanding of links between abstract and concrete models by extending Galois connections studied in abstract interpretation [15,35]. We first formalise model space to capture models at the same level of abstraction, in which models are partially ordered in terms of a specific logic for expressing queries. This provides us the ability to compare models that may associate with different schemata in a similar way that relative information capacity [24] was studied for handling semantic heterogeneity. A Galois connection between two model spaces at different levels of abstraction is established by specifying abstraction and concretization functions that translate concrete models into abstract models and vice versa.

From a traditional perspective, the process of data migration involves three stages: Extract, Transform, and Load (ETL). However, different from a conventional ETL used for data warehousing that only deals with analytic data, ETL in data migration has to handle data in an operational environment, which is much more complicated. Therefore, our second contribution is the formal development of the following ETL processes for data migration.

- *Extract*: A legacy kernel is first "extracted" at a high-level of abstraction, consolidating heterogeneous data sources in a legacy system.
- *Transform*: We then "transform" a legacy kernel into a new kernel by specifying migration transformations that may involve validating, cleansing and mapping data.
- *Load*: As "loading" a new kernel into targeted data sources is often straightforward, we will omit the discussion of this stage in this paper.

We also analyse specific migration strategies [7,8,27] that can be applied in the ETL process of data migration.

The third contribution of this paper is a general refinement scheme that specifies the refinement correctness in terms of two fundamental subclasses of migration transformations — property-preserving transformations and property-enhancing transformations. By using our refinement scheme, the above ETL processes can be stepwise refined from high-level abstractions into real-life implementations. As illustrated in Fig. 1, models in an abstract transformation (e.g., $M_{legacykernel}$ and $M_{newkernel}$) can be

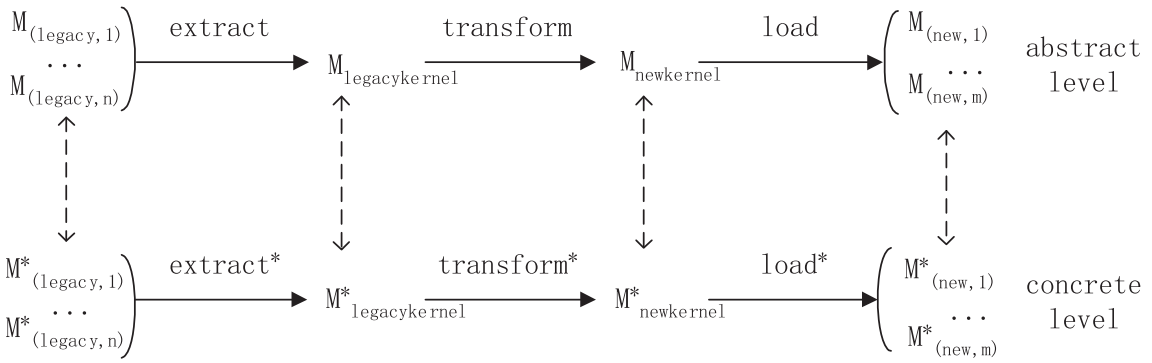

Fig. 1. ETL in data migrations.

refined into more concrete models (e.g., $M^*_{legacykernel}$ and $M^*_{newkernel}$) of the corresponding transformation at a concrete level, and similarly, computation segments of interest (e.g., *extract*, *transform* and *load*) at an abstract level can be refined into the corresponding computation segments (e.g., *extract**, *transform** and *load**) at a concrete level.

Our last contribution is to extend the generic proof method proposed by Schellhorn [32] to verify properties of transformations in real-life implementations. The key idea is to approximate abstraction of transformations in real-life implementations relative to properties of interest in a way of preserving the refinement correctness as accurately as possible. In doing so, this generic proof method can provide us with the ability to control complexity of data migration tasks and develop efficient verification techniques.

1.2. Outline

The rest of the paper is structured as follows. Section 2 presents a running example. We then introduce the definitions for schema and model in Section 3. In Section 4 the connections between abstract and concrete models are exploited as Galois connections. Section 5 presents the definitions for migration transformations and their two fundamental subclasses: property-preserving transformations and property-enhancing transformations. After that, we consider three migration strategies in Section 6 and the issues of refinement, abstraction and verification in Section 7. In particular, we introduce the notions of refinement correctness for property-preserving and property-enhancing transformations, discuss the simplification of abstraction and propose a generic proof method for verifying properties of migration transformation. After presenting a brief discussion on the related work in Section 8, we conclude the paper in Section 9.

2. A running example

We will use the following running example to illustrate the theoretical framework and concepts developed in this paper.

Consider a shipping company that needs to migrate two legacy transport applications into a new transport system. We refer the data sources of two legacy transport applications as RDM_A and OO_B, respectively. RDM_A is associated with a relational schema, while OO_B has an object-oriented schema. The initial analysis by using data profiling tools shows that the design information of the legacy data sources RDM_A and OO_B is out-of-date and incomplete due to various reasons. In order to improve data quality in the new system, the shipping company requests three external insurance data sources (with possibly different data formats) from insurance companies for cleansing up the insurance information, and six external transport data sources (also with possibly different data formats) from associated transport companies for cleansing up the transport carrier data stored in the legacy data sources.

Assume that the shipping company chooses the ER modelling to build abstract models for data sources. The original design information of two legacy data sources needs to be recovered as much as possible before representing them as abstract models at a high-level of abstraction. From a traditional viewpoint, approaches for database reverse engineering [20] can be used for this purpose, which may involve two successive sub-steps — based on the analysis of implementation at the physical level, the design of data sources is first recovered as a logical schema then transformed into a conceptual schema.

Suppose that RDM_A is abstracted to the model ER_A with the schema S_A depicted in Fig. 2 while OO_B is abstracted to the model ER_B with the schema S_B depicted in Fig. 3. In addition, Figs. 4 and 5 present the ER schemata S_{legacy} and S_{new} of the legacy kernel for the legacy data sources and of the new kernel for the new transport system, respectively, determined by the shipping company. Although the legacy data sources RDM_A and OO_B are heterogeneous data models, their ER models can effectively serve as a bridge to represent heterogeneous legacy data sources in a unifying data modelling framework, i.e., at the same level of abstraction. The formal definitions of schema, model and connections between models at different levels of abstraction will be presented in Sections 3 and 4.

Then the shipping company starts to design transformations in the ETL process of data migration. Same as software design, migration transformations are designed by refining a macroscopic statement of rules at a high-level of abstraction in a stepwise fashion, until executable operations are reached. In each step, more details are added into the specification of a transformation.

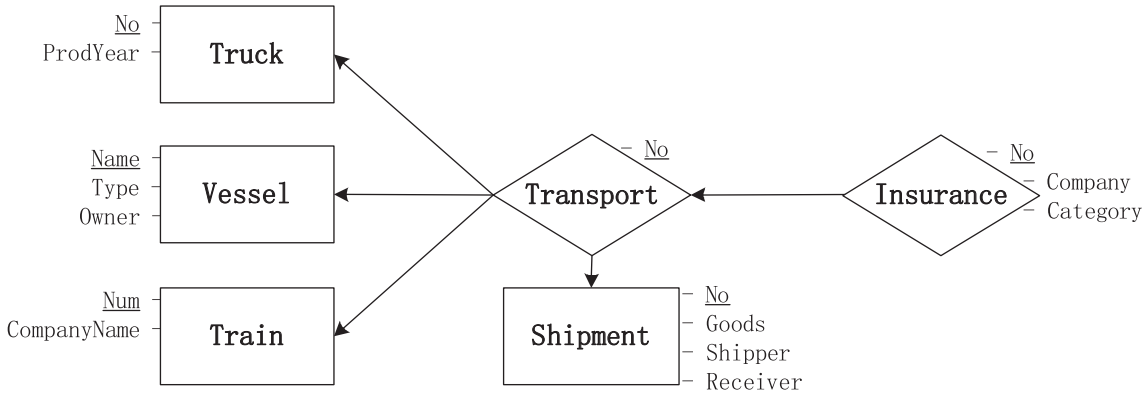

Fig. 2. S_A — an ER schema of legacy data source RDM_A.

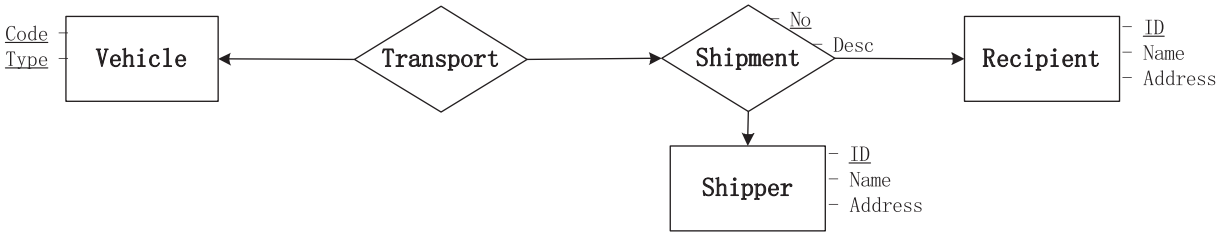

Fig. 3. S_B — an ER schema of legacy data source oo_B.

In the following we use abstract state machines (ASMs) — a practical and scientifically well-founded specification language [10] to illustrate two transformations: the first is to extract data from the legacy data source RDM_A into the legacy kernel (i.e., in Example 1), and the second is to clean up insurance data in the legacy kernel (i.e., in Example 2). Correspondingly, they represent two basic building blocks for migration transformations: i) property-preserving transformations preserve data and their desired semantics among models that may have different schemas; ii) property-enhancing transformations enhance the semantics of data in a given model by imposing new properties, which leads to improved data quality. The formal definitions of ASMs and transformations will be presented in Section 5.

Example 1. The following transformation P_A serves as a blueprint for designing how data should be extracted from the legacy data source RDM_A into the legacy kernel.

```
seq
  forall x with x∈TRUCK do
    EXTRACTEDTOCARRIER(x, "truck", "A")
  enddo
  forall x with x∈VESSEL do
    EXTRACTEDTOCARRIER(x, "vessel", "A")
  enddo
  forall x with x∈TRAIN do
    EXTRACTEDTOCARRIER(x, "train", "A")
  enddo
  forall x with x∈SHIPMENT do
    EXTRACTEDTOSHIPMENT(x, "A")
  enddo
  forall x with x∈TRANSPORT do
    EXTRACTEDTOTRANSPORT(x, "A")
  enddo
  forall x with x∈INSURANCE do
    EXTRACTEDTOINSURANCE(x)
  enddo
endseq
```

There are two properties that P_A must satisfy: 1) each forall rule extracts out all objects of one type in S_A and stores them in S_{legacy}; 2) the model over S_{legacy} must maintain the constraints defined on these objects over S_A.

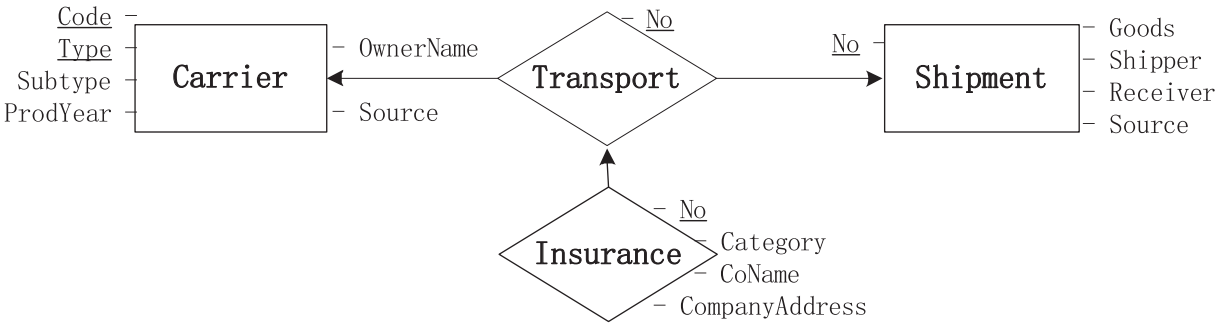

Fig. 4. S_{legacy} — an ER schema of legacy kernel.

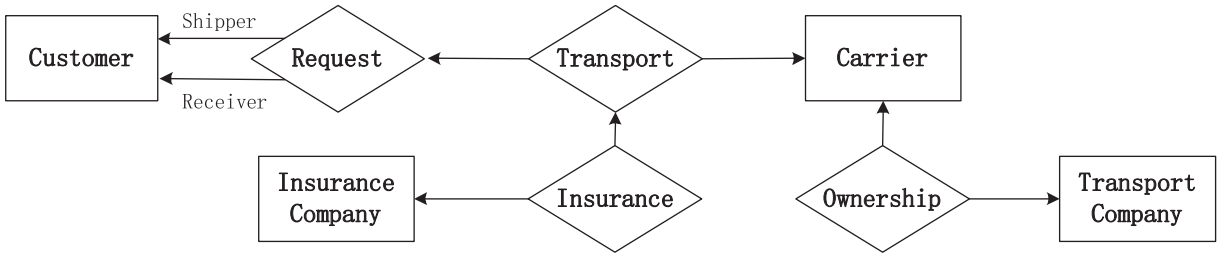

Fig. 5. S_{new} — an ER schema of new kernel.

Suppose that we want to reflect a design decision on EXTRACTEDTOCARRIER of P_A, which leads to a refined transformation P_A^* by replacing EXTRACTEDTOCARRIER(x,y,z) of P_A with the following rule (assume that trains in RDM_A are only provided by "NZ Trans").

par
  if y = "truck" ∧ z = "A" **then**
    seq
      o := SEARCHTRANSPORTCOMPANY(x.NO, "NZ Trans")
      INSERTINTOCARRIER(x.No, "truck", o.TYPE, x.PRODYEAR, "NZ Trans", z)
    endseq
  endif
  if y = "vessel" ∧ z = "A" **then**
    seq
      o := SEARCHTRANSPORTCOMPANY(x.NAME, x.OWNER)
      INSERTINTOCARRIER(x.NAME, "vessel", x.TYPE, o.PRODYEAR, x.OWNER, z)
    endseq
  endif
  if y = "train" ∧ z = "A" **then**
    seq
      o := SEARCHTRANSPORTCOMPANY(x.NUM, x.COMPANYNAME)
      INSERTINTOCARRIER(x.NUM, "train", " ", o.YEAR, x.COMPANYNAME, z)
    endseq
  endif
  if z = "B" **then**
    …
  endif
endpar

Truck	
No	ProdYear
tr1	2001
tr2	2003

Vessel		
Name	Type	Owner
v200	tanker	Arina

Train	
Num	CompanyName
t77	KiwiRail
t88	CSXCorporation

Shipment			
No	Shipper	Receiver	Goods
sh1	BY Trading Co.	J&J Warehouse, Dunedin	household
sh2	Mitsubishi Motors	Automoto Ltd, Wellington	Cars
sh1	BY TradingCo.	Warehouse Dunedin	household

Transport				
No	ShipNo	TNo	VName	TNum
ts1	sh1	tr1	v200	
ts2	sh2	tr2		t88

Insurance			
No	TranNo	Company	Category
in1	ts1	State	Total Loss Coverage
in2	ts2	AA	Basic Risk
in3	ts2	P.A.F.	All Risk Coverage

Fig. 6. M_A — a simple relational model of S_A in Fig. 2.

P_A^* enriches data by adding relevant information from external transport data sources into the legacy kernel. Similarly, we can further refine EXTRACTEDTOSHIPMENT(x,y), EXTRACTEDTOTRANSPORT(x,y) and EXTRACTEDTOINSURANCE(x) until the resulting transformation is executable. Given the relational model M_A of S_A in Fig. 6, the transformation P_A can extract out data in M_A and transform them into M_E as shown in Fig. 7. □

Example 2. Suppose that we want to use the external insurance data sources provided by insurance companies to validate and enrich insurance data in the legacy kernel of our running example.

- ϕ_1: Each insurance in INSURANCE must be found in one of three external insurance data sources;
- ϕ_2: The details of each insurance in INSURANCE must match the information found in three external insurance data sources.

Moreover, we add two constraints on S_{legacy} to filter out "dirty data":

- ϕ_3: Each transport must have at least one insurance.
- ϕ_4: Each transport has no more than one insurance in the same category.

The following transformation P_B describes a blueprint for cleaning up insurance data in the legacy kernel, consisting of two steps: a) identifying dirty data in accordance with each constraint, then b) applying different cleansing strategies CLEANUPINSURANCE$_i(x)$ for dirty data corresponding to different constraints.

seq
– Clean up invalid insurances violating ϕ_1
forall x **with** INSURANCE$(x) \wedge$ INSURANCECOMPANIES(x)
do
CLEANUPINSURANCE$_1(x)$
enddo
– Clean up incorrect insurances violating ϕ_2
forall x **with** INSURANCE$(x) \wedge$ INSURANCECOMPANIES$(x) \wedge$ MATCH(x)
do
CLEANUPINSURANCE$_2(x)$
enddo

Carrier

Code	Type	Subtype	ProdYear	OwnerName	Source
tr1	truck	flatbed	2001	NZTrans	A
tr2	truck	refrigerator	2003	NZTrans	A
v200	vessel	tanker	2000	Arina	A
t77	train	null	2007	KiwiRail	A
t88	train	null	1998	CSX Corporation	A

Shipment

No	Shipper	Receiver	Goods	Source
sh1	BY Trading Co.	J&J Warehouse, Dunedin	household	A
sh2	Mitsubishi Motors	Automoto Ltd, Wellington	Cars	A
sh1	BY Trading Co.	Warehouse Dunedin	household	A

Transport

No	ShipNo	CarrierCode	CarrierType
ts1	sh1	tr1	truck
ts1	sh1	v200	vessel
ts1	sh1	null	train
ts2	sh2	tr2	truck
ts2	sh2	null	vessel
ts2	sh2	t88	train

Insurance

No	TranNo	Category	CoName	CompanyAddress
in1	ts1	Total LossCoverage	State	35 King St, Dunedin
in2	ts2	Basic Risk	AA	6 Kiler Place, Auckland
in3	ts2	All Risk Coverage	P.A.F.	9 Arnold St, Wellington

Fig. 7. M_E — a simple relational model of S_{legacy} in Fig. 4.

– Clean up missing insurances violating ϕ_3

forall x **with** TRANSPORT(x) $\wedge$ INSURANCED(x)
do
CLEANUPINSURANCE$_3$(x)
enddo

– Clean up multiple insurances paid for the same transport violating ϕ_4

forall x **with** TRANSPORT(x) $\wedge$ MULTIPLEINSURANCES(x)
do
CLEANUPINSURANCE$_4$(x)
enddo
endseq

To ensure the desired data quality after this data cleaning task, P_B must satisfy two properties: 1) every model obtained after executing the kth forall rule must satisfy the condition φ_k, where $\varphi_1 = \phi_1, \varphi_2 = \phi_1 \wedge \phi_2, \varphi_3 = \phi_1 \wedge \phi_2 \wedge \phi_3$ and $\varphi_4 = \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4$; 2) each rule CLEANUPINSURANCE$_i$(x) is only allowed to delete objects x or the ones that refer to x.

Suppose that we want to refine CLEANUPINSURANCE$_i$(x) to reflect more specific strategies for dirty data cleaning. The refined transformation $P_B{}^*$ is obtained by replacing CLEANUPINSURANCE$_i$(x) ($1 \le i \le 4$) of P_B as follows.

- CLEANUPINSURANCE$_1$(x) = DELETEINSURANCE(x)
- CLEANUPINSURANCE$_2$(x) =

seq
DELETEINSURANCE(x)
output := FETCHRECORDSFROMINSURANCECOMPANY(x, 1)
forall y **with** $y \in$ *output* **do**
INSERTINSURANCE(y)
enddo
endseq

- CLEANUPINSURANCE$_3$(x) =

Seq
output := FETCHRECORDFROMINSURANCECOMPANY(x, 2)
forall y **with** $y \in$ *output* **do**
INSERTINSURANCE(y)
enddo
endseq

- CLEANUPINSURANCE$_4$(x) =

Seq
output := FINDINSURNACE(x)
DELETETRANSPORT(x)
forall y **with** $y \in$ *output* **do**
DELETEINSURANCE(y)
enddo
INSERTDIRTYINSURANCE(x, y)
endseq

where FETCHRECORDFROMINSURANCECOMPANY(x, *status*) =

if *status* = 1 **then**
FETCHBYINSURANCECODE
endif if
if *status* = 2 **then**
FETCHBYTRANSACTIONNO
endif

In $P_B{}^*$, invalid insurances with respect to ϕ_1 are deleted from INSURANCE, dirty data with respect to ϕ_2 and ϕ_3 are cleaned up by using external insurance data sources, e.g., in the refinement of CLEANUPINSURANCE$_3$(x) new insurances from external insurance data sources are inserted into INSURANCE. These added insurances may be dirty with respect to ϕ_4; nevertheless, any dirty data with respect to ϕ_4 can be cleaned up in the refined CLEANUPINSURANCE$_4$(x), which removes the related transports and their insurances from TRANSPORT and INSURANCE into a place specific for holding dirty data under review. □

In both the examples described above, we start with specifying a transformation at a high-level abstraction in which desired properties are made explicitly, then refine the transformation until it is executable. Apparently, there are many ways of refining a transformation. How can we decide whether a refinement is correct in terms of certain desired properties? In Section 7 we will develop a general refinement scheme and the notions of correctness for these two subclasses of migration transformations. They can support a disciplined use of refinements so as to correctly reflect intended design decisions. Because the specification of a transformation often changes in a data migration task, maintaining the links between high-level specifications and their refinements become particularly important for systematically verifying properties of data migration. When such links are accurately maintained, the impacts of a specification change can be easily verified on the high-level specifications, meanwhile the correct refinement can ensure that the verified properties are actually preserved in the corresponding executable transformations at the implementation level.

3. Schemata and models

In a data migration project, data sources of legacy and new systems are often designed by using different data modelling approaches. Nevertheless, components supported by many data modelling approaches can be viewed as objects, e.g., entities and relationships in entity–relationship models, tuples in relational data models, and elements in XML. In this paper we take an object-based view on models, which gives us the flexibility to relate different models to each other, regardless of their levels of abstraction.

Let us fix a family $\mathcal{D}$ of basic domains and a set $\mathcal{C}$ of constructors (e.g., record, list, set, multiset, and array). Then an *object type* over $(\mathcal{D}, \mathcal{C})$ can be inductively defined by applying a subset of constructors in $\mathcal{C}$ over a subset of basic domains in $\mathcal{D}$. For example, a relation can be regarded as an object type defined by applying a record constructor over a finite number of basic domains. Let τ be an object type. Then an *object* of τ is a mapping that assigns for each basic domain of τ a value from that domain.

In order to capture additional semantic restrictions on data models, we need a suitable logic[1] L for expressing their properties (i.e., constraints). We use the notations $\mathcal{F}(L, T)$ (or simply $\mathcal{F}(T)$ when there is no ambiguity about L) referring to the set of formulae of L inductively defined by applying all rules of L for building formulae over a set T of object types, and $fr(\varphi)$ referring to the set of free variables in formula φ. A formula φ with $fr(\varphi) = \varnothing$ is called *Boolean formula*. A *constraint* over T is a Boolean formula $\varphi \in \mathcal{F}(L, T)$ of the form $\forall x_1,\ldots,x_n.(\psi_1 \Rightarrow \psi_2)$, where $fr(\psi_1) \cup fr(\psi_2) = \{x_1,\ldots,x_n\}$. In the following we exemplify several types of constraints existing in most relational data models [12], where, for simplicity, we choose the first-order logic.

- Primary key constraints: for relation $r(A_1,\ldots,A_n)$ with the primary key on attribute A_1, we have

$$\forall x_1,\ldots,x_n,y_1,\ldots,y_n.((r(x_1,\ldots,x_n) \wedge r(y_1,\ldots,y_n) \quad \wedge \vee_{1<i\le n} x_i \neq y_i) \Rightarrow x_1 \neq y_1).$$

- Foreign key constraints: for relation $r_1(A_1,\ldots,A_n)$ with the primary key on attribute A_1, and relation $r_2(A_1',\ldots,A_m')$ with a foreign key on attribute A_1' referring to A_1 of r_1, we have

$$\forall y_1,\ldots,y_m.(r_2(y_1,\ldots,y_m) \Rightarrow \exists x_1,\ldots,x_n.(r_1(x_1,\ldots,x_n) \wedge x_1 = y_1)).$$

- Check constraints: for relation $r_1(A_1,\ldots,A_n)$ with a condition on attribute A_1 so that each value of A_1 needs to satisfy a formula φ, we have

$$\forall x_1,\ldots,x_n.(r(x_1,\ldots,x_n) \Rightarrow \varphi).$$

- Functional dependencies: for relation $r_1(A_1,\ldots,A_n)$ with a functional dependency $A_1 \to A_2$, we have

$$\forall x_1,\ldots,x_n,y_1,\ldots,y_n.((r(x_1,\ldots,x_n) \wedge r(y_1,\ldots,y_n) \wedge x_1 = y_1) \Rightarrow x_2 = y_2).$$

A *schema* $S = (T, \Sigma)$ consists of a finite, non-empty set T of object types and a finite, possibly empty set Σ of constraints such that $\Sigma \subseteq \mathcal{F}(L, T)$. A model over schema $(T, \varnothing)$ consists of a finite, non-empty set of objects whose types are in T. We use $[[\varphi]]^M$ referring to the interpretation of a formula φ in a model M. A constraint $\varphi \in \mathcal{F}(L, T)$ is said to be *satisfied* by a model M over schema $(T, \varnothing)$, denoted as $M \models \varphi$, if $[[\varphi]]^M$ is true. A *model* over schema (T, Σ) is a model over schema $(T, \varnothing)$, satisfying every constraint in Σ. For convenience, we use the notations $\mathcal{M}(S)$ to denote the set of all models over schema S and $\mathcal{M}(T)$ to denote the set of all models over schema $S = (T, \varnothing)$. Clearly, $\mathcal{M}(S) \subseteq \mathcal{M}(T)$ for $S = (T, \Sigma)$, and $\mathcal{M}(S) = \mathcal{M}(T)$ if $\Sigma = \varnothing$.

Example 3. The schema S_A as shown in Fig. 2 has the object types TRUCK, VESSEL, TRAIN, SHIPMENT TRANSPORT and INSURANCE, and the following constraints:

- φ_1: the unique constraint on NO of TRUCK:

$$\forall x_1, x_2, y_1, y_2 \cdot ((\mathrm{TRUCK}(x_1, x_2) \wedge \mathrm{TRUCK}(y_1, y_2) \wedge x_2 \neq y_2) \Rightarrow x_1 \neq y_1),$$

[1] The suitable logic is determined by what kinds of constraints we want to capture. For example, first-order logic can express many integrity constraints but not cardinality constraints. However, cardinality constraints can be captured by a more expressive logic such as fixed point logic with counting quantifiers or counting terms [26,29]. We may also use Monadic second-order logic [14] to define constraints of XML tree structures.

- φ_2: the foreign key constraint on INSURANCE:

$$\forall x_1, x_2, x_3, x_4 \cdot (\text{INSURANCE}(x_1, x_2, x_3, x_4) \Rightarrow \exists y_1, y_2, y_3, y_4 \cdot \text{TRANSPORT}(x_2, y_1, y_2, y_3, y_4)).$$

Hence, $S_A = (\{\text{TRUCK, VESSEL, TRAIN, SHIPMENT, TRANSPORT, INSURANCE}\}, \{\varphi_1, \varphi_2\})$. Every model over S_A must satisfy the constraints φ_1 and φ_2. □

4. Levels of abstraction

Each model is represented at a certain level of abstraction. In this section, we discuss relationships between models at the same level of abstraction, and connections between model spaces at different levels of abstraction.

4.1. Model space

Analogous to the logic L used for expressing constraints, we can choose a logic for expressing queries. When such a logic L' is determined, models at the same level of abstraction can be regarded as being partially ordered. We formalise this by using the notion of reflecting relation.

Definition 1. Fix a logic L suitable for expressing queries. Let M_1 and M_2 be two models over the schemata $S_1 = (T_1, \Sigma_1)$ and $S_2 = (T_2, \Sigma_2)$, respectively. Then M_1 *reflects* M_2 with respect to L, denoted as $M_2 \preceq_L M_1$ (L may be omitted when there is no ambiguity), if, for any query $\varphi_2 \in \mathcal{F}(L, T_2)$ with $fr(\varphi_2) = \{x_1, \ldots, x_n\}$, whenever $M_2 \models \exists x_1, \ldots, x_n . \varphi_2$, there exists a formula $\varphi_1 \in \mathcal{F}(L, T_1)$ such that $[[\varphi_1]]^{M_1} = [[\varphi_2]]^{M_2}$ holds.

Note that, M_1 and M_2 in the above definition may associate with different schemata. It is possible that $T_1 \neq T_2$ or $\Sigma_1 \neq \Sigma_2$. Intuitively, if $M_2 \preceq_L M_1$, then it means that, regardless of whether or not M_1 and M_2 have the same schema, data captured by using a logical formula of L over model M_1 can be equally captured by using a logical formula of L over model M_2. However, this does not generally hold the other way round. The reflecting relation provides us the ability to compare models that may associate with different schemata in a similar way that relative information capacity was studied for handling semantic heterogeneity [24,25,28]. Since a reflecting relation is defined with respect to a specific logic, given two different logics L_1 and L_2, $M_2 \preceq_{L_1} M_1$ does not necessarily imply $M_2 \preceq_{L_2} M_1$ unless L_1 is strictly expressive than L_2.

Example 4. Consider the models M_A and M_E shown in Figs. 6 and 7. We have $M_A \preceq_L M_E$ when L is chosen to be the first-order logic. It means that for every first-order query over M_A, we can always obtain the same query result by using a first-order query over M_E. This can be alternatively illustrated by constructing M_A using the following queries over M_E.

- $Q_{\text{TRUCK}}(x_1, x_2) = \exists y_1, y_2 \cdot \text{CARRIER}(x_1, \text{“truck”}, y_1, x_2, y_2, \text{“A”})$
- $Q_{\text{VESSEL}}(x_1, x_2, x_3) = \exists y_1 \cdot \text{CARRIER}(x_1, \text{“vessel”}, x_2, y_1, x_3, \text{“A”})$
- $Q_{\text{TRAIN}}(x_1, x_2) = \exists y_1, y_2 \cdot \text{CARRIER}(x_1, \text{“train”}, y_1, y_2, x_2, \text{“A”})$
- $Q_{\text{SHIPMENT}}(x_1, x_2, x_3, x_4) = \text{SHIPMENT}(x_1, x_2, x_3, x_4, \text{“A”})$
- $Q_{\text{TRANSPORT}}(x, y, z_1, z_2, z_3) = \text{TRANSPORT}(x, y, z_1 \text{“truck”}) \wedge \text{TRANSPORT}(x, y, z_2, \text{“vessel”}) \wedge \text{TRANSPORT}(x, y, z_3, \text{“train”})$
- $Q_{\text{INSURANCE}}(x_1, x_2, x_4, x_3) = \exists x_5 \cdot \text{INSURANCE}(x_1, x_2, x_3, x_4, x_5)$ □

For models at the same level of abstraction, we use the notion of model space to describe their relative data capacity in terms of a specific logic.

Definition 2. A *model space* $\mathbb{M}$ is a lattice $(\mathcal{M}, \preceq, \bot, \top, \sqcup, \sqcap)$ consisting of

- a set $\mathcal{M}$ of models,
- a reflecting relation $\preceq$ defined on $\mathcal{M}$ with respect to a certain logic,
- a greatest model $\top$ satisfying $M \preceq \top$ and a smallest model $\bot$ satisfying $\bot \preceq M$ for all $M \in \mathcal{M}$,
- a least upper bound operation $\sqcup$ and a greatest lower bound operation $\sqcap$ such that, for any $\mathcal{M}' \subseteq \mathcal{M}$, there exist M_{meet} and $M_{join} \in \mathcal{M}$ satisfying $M_{meet} \preceq \sqcap \mathcal{M}'$ and $\sqcup \mathcal{M}' \preceq M_{join}$.

Thus, given a subset $\mathcal{M}' \subseteq \mathcal{M}$, a least upper bound for $\mathcal{M}'$ is a model M_{join} that reflects every model in $\mathcal{M}'$. Conversely, a greatest lower bound for $\mathcal{M}'$ is a model M_{meet} that is reflected by every model in $\mathcal{M}'$.

4.2. Abstract and concrete models

Now we consider models that are defined at two different levels of abstraction — abstract models and concrete models. The notions of “abstract” and “concrete” are relative, depending on which levels are chosen. For example, models at the logical level can be seen as being “abstract” when compared to models at the physical level, but being “concrete” when compared to models at the conceptual level.

Following the framework of abstract interpretation [15,35], we establish the connection between abstract and concrete models as a Galois connection. Let $\mathbb{A} = (\mathcal{M}_A, \preceq_A, \bot, \top, \sqcup, \sqcap)$ be an abstract model space, where $\mathcal{M}_A$ is a set of abstract models and $\preceq_A$ is a reflecting relation with respect to a logic for abstract models. Analogously, let $\mathbb{C} = (\mathcal{M}_C, \preceq_C, \bot, \top, \sqcup', p')$ be a concrete model space that corresponds to $\mathbb{A}$, where $\mathcal{M}_C$ is a set of concrete models at a lower level of abstraction and $\preceq_C$ is a reflecting relation defined over $\mathcal{M}_C$. Then there is a *Galois connection* between $\mathbb{A}$ and $\mathbb{C}$, denoted as $\mathcal{M}_C\langle\alpha,\gamma\rangle\mathcal{M}_A$, where

- α: $\mathcal{M}_C \to \mathcal{M}_A$ is an *abstraction function* that translates a concrete model in $\mathcal{M}_C$ to its most precise approximation in $\mathcal{M}_A$, and
- γ: $\mathcal{M}_A \to \mathcal{M}_C$ is a *concretization function* that translates an abstract model in $\mathcal{M}_A$ to its most general refinement in $\mathcal{M}_C$.

Both α and γ are monotonic; furthermore, $M \preceq_C \gamma\circ\alpha(M)$ and $\alpha\circ\gamma(M') \preceq_A M'$ are satisfied. In other words, $\alpha(x)$ is the most precise abstract model in $\mathcal{M}_A$ whose concretization approximates x, while $\gamma(y)$ is the concrete model in $\mathcal{M}_C$ that is represented by y at the abstract level (Fig. 8).

Alternatively, the abstraction function α and concretization function γ may be defined at the schema level. Let $\mathcal{S}_A$ and $\mathcal{S}_C$ be two sets of schemata associated with the model spaces $\mathbb{A}$ and $\mathbb{C}$, respectively. That is, we consider $\mathcal{S}_C$ as "concrete" schemata and $\mathcal{S}_A$ as "abstract" schemata. Then from the schema-mapping point of view, the abstract function $\alpha{:}_C \to \mathcal{S}_A$ translates a concrete schema to an abstract schema such that $\alpha = (S_C) = (\alpha(T_C), \alpha(\Sigma_C)) = (T_A, \Sigma_A)S_A$ for $S_C = (T_C, \Sigma_C)$, while the concretization function $\gamma{:}\mathcal{S}_A \to \mathcal{S}_C$ translates an abstract schema to a concrete schema such that $\gamma(S_A) = (\gamma(T_A), \gamma(\Sigma_A)) = (T_C, \Sigma_C) = S_C$ for $S_A = (T_A, \Sigma_A)$. These translations can be further extended to their models, i.e., $\alpha{:}\mathcal{M}(S_C) \to \mathcal{M}(S_A)$ and $\gamma{:}\mathcal{M}(S_A) \to \mathcal{M}(S_C)$ in a canonical way.

Note that, a Galois connection between models at two different levels of abstraction can be implemented in various ways. For example, database reverse and forward engineering approaches (e.g., [20]) may be regarded as translations between conceptual models and physical models captured by a Galois connection.

5. Migration transformations

In this section we formalise migration transformations occurring between legacy data sources of a legacy system and new data sources of a target system. In order to have sufficient expressive power for specifying complex transformations in a comprehensive manner, we adopt abstract state machines (ASMs), a high-level specification language introduced by Gurevich [10]. As ASMs are distinguished from other computation models by the power of modelling algorithms at arbitrary levels of abstraction, we have the capability of modelling each migration transformation as a deterministic computation executed at a flexible but fixed level of abstraction.

Definition 3. A *transformation* is a tuple $(\mathcal{M}, M_0, M_n, \delta)$ consisting of a non-empty set $\mathcal{M}$ of models together with an initial model $M_0 \in \mathcal{M}$ and a final model $M_n \in \mathcal{M}$, and a one-step transition function δ over $\mathcal{M}$, i.e., $\delta : \mathcal{M} \mapsto \mathcal{M}$.

The one-step transition function δ is determined by a transition rule inductively defined by

- *update rule*: the value of the object $\tau(t_1,\ldots,t_n)$ is updated to t_0,

$$\tau(t_1, \ldots, t_n) := t_0$$

- *conditional rule*: if ϕ is true, then execute the rule R; otherwise do nothing,
  if ϕ **then** R **endif**
- *block rule*: the rules $R_1,\ldots,R_n$ are executed in parallel,
  par $R_1 \ldots R_n$ **endpar**
- *sequential rule*: the rules $R_1,\ldots,R_n$ are executed sequentially,
  seq $R_1 \ldots R_n$ **endseq**

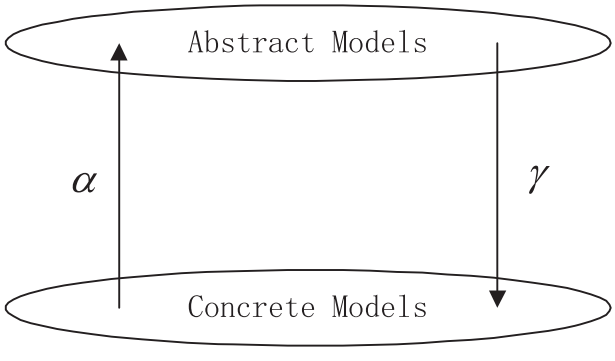

Fig. 8. Galois connection between abstract and concrete models.

- *forall rule*: the rule R is executed in parallel for each $x_1,\ldots,x_n$ satisfying φ,
  forall $x_1,\ldots,x_n$ **with** φ **do** R **enddo**
- *call rule*: the rule R is called with parameters $t_1,\ldots,t_n$.

$$R(t_1,\ldots,t_n)$$

A *rule declaration* is an expression $R(x_1,\ldots,x_n)=R'$ for a rule R' and the free variables of R' are $x_1,\ldots,x_n$. In a rule $R(t_1,\ldots,t_n)$, the variables $x_1,\ldots,x_n$ in R' are replaced by the parameters $t_1,\ldots,t_n$, respectively.

An *update* is a pair (ℓ, b), where ℓ is an object and b is called the *value* of ℓ. An *update set* Δ is a set of updates. An update set Δ is *consistent* if it does not contain conflicting updates, i.e., for all (ℓ, b), $(\ell, b')\in\Delta$ we have $b=b'$.

Applying a transition rule R over a model M yields a set $\Delta(R, M)$ of updates on objects. If $\Delta(R, M)$ is consistent, then updates in $\Delta(R, M)$ lead a transformation from the current model M to its successor model $\delta(M)$ such that there exists a unique model $S'=S+\Delta$ resulting from updating S with Δ:

$$val_{S+\Delta}(\ell)=\begin{cases} b & \text{if } (\ell,b)\in\Delta \\ val_S(\ell) & \text{else}\end{cases}.$$

A *run* is a finite sequence $M_0,\ldots,M_n$ of models with $M_i\neq M_n$ for $0<i<n$, and $\delta(M_i)=M_{i+1}$ for all $i=0,\ldots,n-1$.

In the rest of this section, we will formally present two fundamental subclasses of migration transformations.

5.1. Property-preserving transformations

The first subclass of migration transformations is property-preserving transformations, which transform data and their description from one model (i.e., an initial model) to another model (i.e., a final model) in a way that preserves data and constraints of the initial model.

Suppose that a logic L is used for expressing queries over models of property-preserving transformations. If Ψ is a set of formulae, then $\Psi[\tau_1\leftarrow\varphi_1,\ldots,\tau_k\leftarrow\varphi_k]$ is also a set of formulae, which is obtained by substituting each object type $\tau_i(x_1,\ldots,x_k)$ in Ψ by a formula φ_i where $fr(\varphi_i)=\{x_1,\ldots,x_k\}$.

Definition 4. A *property-preserving transformation* (PPT) is a transformation $\Pi=(\mathcal{M}, M_0, M_n, \delta)$, where $M_0\in\mathcal{M}(S_0)$ for $S_0=(T_0,\Sigma_0)$ and $M_n\in\mathcal{M}(S_n)$ for $S_n=(T_n,\Sigma_n)$, satisfying the following conditions:

- $\bigwedge_{0\le i<j\le n} M_i\preceq_L M_j$, and
- there exists a set $\Phi\subseteq\mathcal{F}(L,T_n)$ of constraints that is equivalent to Σ_0, i.e., $\Phi=\Sigma_0[\tau_1\leftarrow\varphi_1,\ldots,\tau_k\leftarrow\varphi_k]$, and $M_n\models\bigwedge_{\phi\in\Phi}\phi$ holds.

In the above definition, the first condition ensures that data in a model M_i can always be retrieved from its successor model M_j even when their schemata are different, i.e., M_j reflects M_i. The second condition requires an equivalent form of constraints imposed on the initial model M_0 to be satisfied by the final model M_n.

Property-preserving transformations in data migration include extracting a legacy kernel, mapping data between models with different schemata, loading a new kernel, etc.

Example 5. The transformation P_A in Example 1 is a PPT that extracts the legacy kernel. If the constraints φ_1 and φ_2 (over S_A) in Example 3 are imposed on the initial model of P_A, then the following constraint φ_1', which is in an equivalent form of φ_1 over S_{legacy}, needs to be satisfied by the final model of P_A.

$$\varphi'_1\equiv\forall x_1,x_3,y_1,y_3\cdot((\exists x_2,x_4\cdot \mathrm{CARRIER}(x_1,\text{“truck”},x_2,x_3,x_4,\text{“A”})\wedge \exists y_2,y_4\cdot \mathrm{CARRIER}(y_1,\text{“truck”},y_2,y_3,y_4,\text{“A”})\wedge x_3\neq y_3)\Rightarrow x_1\neq y_1).$$

Note that the constraint corresponding to φ_2 of S_A becomes trivial in S_{legacy} after the transformation. □

In general a legacy system may have a number of legacy data sources managed in heterogeneous environments, e.g., Oracle databases, flat files, XML documents, Excel spreadsheets, and Access databases. Therefore, the first step in the ETL process of data migration is to extract a legacy kernel that consolidates all legacy data sources. In order to handle heterogeneity of legacy data sources, such task should be performed at a high level of abstraction. The resulting legacy kernel must preserve data of each legacy data source without loss of information. Therefore, a legacy kernel is the least upper bound of a set of abstract models corresponding to legacy data sources in terms of a reflecting relation determined by the specific logic for querying these abstract models.

Definition 5. Let $\mathcal{M}'=\{M_1,\ldots,M_n\}$ be a set of abstract models from the same model space $(\mathcal{M},\preceq,\bot,\top,\sqcup,\sqcap)$, i.e., $\mathcal{M}'\subseteq\mathcal{M}$. Then the *legacy kernel* M_K of $\mathcal{M}'$ is the least upper bound of models in $\mathcal{M}'$ satisfying $\sqcup\mathcal{M}'\preceq M_K$.

Transformations for mapping data between two models with different schemata are generally considered as PPTs. According to Definition 4, a PPT requires that data in the initial model of a PPT must be preserved in its final model without loss of information.

Note that, although transformations for mapping data often operate at the schema level, e.g., specifying mappings between schema components of two models, not every transformation for mapping data can be handled as a schema transformation. We allow transformations for mapping data between two models to leverage the semantics defined at the data level, so such transformations are not always generic [4]. For example, depending on data in a model, different mapping functions can be initiated for transforming different types of data.

Loading a new kernel is a PPT that can often be straightforwardly defined. We thus skip the detailed discussion on such transformations.

5.2. Property-enhancing transformations

The second subclass of migration transformations we capture is property-enhancing transformations, which transform models violating a certain set of properties into models satisfying these properties. During such transformations, data in an initial model have to be amended so as to ensure that the properties are satisfied by a final model.

For simplicity, we will take a set-theoretic viewpoint to discuss relationships between models defined over the same set of object types. Given two models M_1 and $M_2 \in \mathcal{M}(T)$, we have

- $M_2 \subseteq M_1$ if $[[\tau]]^{M_2} \subseteq [[\tau]]^{M_1}$ holds for each $\tau \in T$;
- $M_1 - M_2 = M_3$ if $[[\tau]]^{M_1} - [[\tau]]^{M_2} = [[\tau]]^{M_3}$ holds for each τ;
- $M_1 \cap M_2 = M_3$ if $[[\tau]]^{M_1} \cap [[\tau]]^{M_2} = [[\tau]]^{M_3}$ holds for each τ;
- $M_1 \cup M_2 = M_3$ if $[[\tau]]^{M_1} \cup [[\tau]]^{M_2} = [[\tau]]^{M_3}$ holds for each τ.

We say that M_2 is a *submodel* of M_1 if $M_2 \subseteq M_1$ holds. When imposing a constraint φ on M_1, M_2 is said to be the *valid submodel* of M_1 with respect to φ if M_2 is a submodel of M_1 and $M_2 \models \varphi$ holds, and M_2 is said to be the *greatest valid submodel* of M_1 with respect to φ if M_2 is the valid submodel of M_1 satisfying the following two conditions:

- $M' \not\models \varphi$ for every M' with $M' \subseteq M_1$ and $M_2 \subseteq M'$.

Basically, the above condition defines that M_2 is greatest in the sense that there does not exist any other models, which subsume M_2 but are subsumed by M_1. Note that, it is possible to find a number $\{M_2^1,\ldots,M_2{}^k\}$ of *greatest valid submodels* of M_1 with respect to φ. The intuition behind the notion of greatest valid submodel is, whenever imposing a constraint φ on a model M_1, we consider the model M_1 to be divided into two parts: $M_1{}^{\varphi}$ and $\overline{M}_1^{\varphi}$ such that $\overline{M}_1^{\varphi} = M_1 - M_1^{\varphi}$. The part $M_1^{\varphi} = \bigcap_{1 \le i \le k} M_2^i$ contains objects from M_1 which do not violate the constraint φ in any case, i.e., objects appearing in every greatest valid submodel of M_1. The part $\overline{M}_1^{\varphi}$ contains all the other objects in M_1, relating to the violation of the constraint φ. We call $M_1{}^{\varphi}$ the *definite* part of M_1 against φ, and respectively, $\overline{M}_1^{\varphi}$ the *indefinite* part of M_1 against φ. When φ is valid over the whole model M_1, $M_1^{\varphi} = M_1$ and $\overline{M}_1^{\varphi} = \varnothing$. Given a set Φ of constraints, we use the notations M_1^{Φ} to represent $\bigcap_{\varphi_i \in \Phi} M_1^{\varphi_i}$ and $\overline{M}_1^{\Phi}$ to represent $M_1 - M_1^{\Phi}$.

Example 6. Let us consider the model M_A depicted in Fig. 6. Suppose that we want to add the following constraint φ over SHIPMENT to validate the model M_A:

$$\varphi = \forall z, x_1, x_2, x_3, y_1, y_2, y_3 \cdot \left((\text{SHIPMENT}(z, x_1, x_2, x_3) \wedge \text{SHIPMENT}(z, y_1, y_2, y_3)) \Rightarrow \bigwedge_{1 \le i \le 3} x_i = y_i \right).$$

That is, all shipments should have their own unique shipment numbers except for duplicate orders.

Then we would have M_A^{φ} and $\overline{M}_A^{\varphi}$ as shown in Figs. 9 and 10, respectively. For simplicity, we omit objects over TRUCK, VESSEL and TRAIN in M_A^{φ} and $\overline{M}_A^{\varphi}$ as they remain the same as those in M_A. □

Definition 6. Let Ψ be a set of constraints. Then a *property-enhancing transformation* (PET) over Ψ is a transformation $\Lambda = (\mathcal{M}, M_0, M_n, \delta)$, where $\mathcal{M} \subseteq \mathcal{M}(S)$ for $S = (T, \Sigma)$, satisfying the following conditions for each run $M_0, M_1, \ldots, M_n$ of Λ,

- $\bigwedge_{0 \le i < j \le n} M_i^{\Sigma \cup \Psi} \subseteq M_j^{\Sigma \cup \Psi}$, and
- $M_n^{\Sigma \cup \Psi} = M_n$.

In the above definition, the first condition states that for each model in a property-enhancing transformation the objects in its definite part must be preserved in its successor model, while the second condition implies that M_n must satisfy each constraint in Ψ.

In data migration, PETs are widely used for specifying data cleaning tasks in terms of predefined business rules (or constraints). As an integral part of data cleaning, validation checking must be first performed, which identifies dirty data existing in a model. Then, for each business rule φ, an appropriate data cleansing strategy is chosen to eliminate dirty data (i.e., the indefinite part of a model against φ).

Shipment			
No	Shipper	Receiver	Goods
sh2	Mitsubishi Motors	Automoto Ltd, Wellington	Cars

Transport				
No	ShipNo	TNo	VName	TNum
ts2	sh2	tr2	null	t88

Insurance			
No	TranNo	Company	Category
in2	ts2	AA	Basic Risk
in3	ts2	P.A.F.	All Risk Coverage

Fig. 9. The definite part M_A^{φ}.

Example 7. The transformations P_B and $P_B{}^*$ in Example 2 are both a PET. For each constraint φ_i, there is a corresponding validation condition that specifies dirty data w.r.t. φ_i (i.e., x that satisfies the corresponding validation condition is to be cleaned up).

Constraint	Validation condition
ϕ_1	$\text{INSURANCE}(x) \wedge \text{INSURANCECOMPANIES}(x)$
ϕ_2	$\text{INSURANCE}(x) \wedge \text{INSURANCECOMPANIES}(x) \wedge \text{MATCH}(x)$
ϕ_3	$\text{TRANSPORT}(x) \wedge \text{INSURANCED}(x)$
ϕ_4	$\text{TRANSPORT}(x) \wedge \text{MULTIPLEINSURANCES}(x)$

□

6. Migration strategies

In the process of migrating legacy data sources to new data sources, a migration transformation is the composition of a finite sequence of PPTs and PETs.

Definition 7. Let $P_1 = (\mathcal{M}_1, M_0, M_m, \delta_1)$ and $P_2 = (\mathcal{M}_2, M_m, M_n, \delta_2)$ be two transformations. Then the *composition* of P_1 and P_2 is a transformation $P_1 \circ P_2 = (\mathcal{M}_1 \cup \mathcal{M}_2, M_0, M_n, \delta)$ where $\delta : \mathcal{M}_1 \cup \mathcal{M}_2 \to \mathcal{M}_1 \cup \mathcal{M}_2$ such that, if $M \in \mathcal{M}_1$, then $\delta(M) = \delta_1(M)$; if $M \in \mathcal{M}_2$, then $\delta(M) = \delta_2(M)$.

Both PPTs and PETs are closed under composition, which means that a combination of several PPTs (resp. PETs) is another PPT (resp. PET). In the following we discuss migration transformations under different migration strategies [11,8,27].

6.1. Big bang

The strategy of big bang transforms all data from a legacy system into a new data source and takes over all operational data at one time. There are two different approaches to implement this strategy in data migration projects. Fig. 11.a illustrates one approach in which a migration transformation $\Lambda_1 \circ \Pi_1$ starts with Λ_1 to clean up data in the legacy system and then continue with Π_1 to map data into the new data source. Alternatively, a transformation $\Pi_2 \circ \Lambda_2$ can achieve the same effect by swapping the order of data cleansing and mapping. Nevertheless, the latter approach often faces high risk of system failure due to the existence of "dirty" data caused by the long data cleansing process.

6.2. Chicken little

The strategy of chicken little divides a legacy system (including its legacy data) into modules. As few as possible dependencies between modules are retained, and migration takes place by migrating modules step by step. Fig. 11.b shows a two-step data migration process consisting of a transformation $P_1 = \Lambda_{(1,1)} \circ \Pi'_{(1,1)}$ and a transformation $P_2 = \Pi'_{(2,2)} \circ \Lambda_{(2,2)}$. The whole data migration process is the composition of P_1 and P_2. Fig. 11.c shows a three-step data migration in which each step is a transformation that first cleans up the data in a legacy data source and then maps the data into a targeted data source.

6.3. Butterfly

The strategy of butterfly freezes the legacy data sources then uses a crystalliser to transform data from the legacy system to the new system in steps: first transforming data of the read-only legacy data source and then successive temporary data stores. Thus,

Shipment			
No	Shipper	Receiver	Goods
sh1	BY Trading Co.	J&J Warehouse, Dunedin	household
sh1	BY Trading Co.	Warehouse Dunedin	household

Transport				
No	ShipNo	TNo	VName	TNum
ts1	sh1	tr1	v200	null

Insurance			
No	TranNo	Company	Category
in1	ts1	State	Total Loss Coverage

Fig. 10. The indefinite part $\overline{M}_A^{\wp}$.

transformations of data migration in butterfly are similar to the ones in chicken little, with the only difference in separating data sources involved in each step of data migration processes. Fig. 11.d describes a transformation P of butterfly that transforms the read-only legacy data source by $\Lambda_1 \circ \Pi_{(1,1)}$, and two temporary data stores by $\Pi_{(1,2)}$ and $\Pi_{(1,3)}$ during the data migration process, i.e., $P = (\Lambda_1 \circ \Pi_{(1,1)}) \circ \Pi_{(1,2)} \circ \Pi_{(1,3)}$. For simplicity, the two temporary data stores are assumed to be clean without violating any constraints in the new system.

7. Refinement, abstraction and verification

Generally speaking, a refining process is to refine an abstract transformation over abstract models into a concrete transformation over concrete models. In this section, we first introduce the notions of refinement correctness for PPTs and PETs, then discuss an approach of verifying a transformation by simplifying abstraction of the transformation but preserving the refinement correctness in terms of properties of interest.

7.1. Refinement of PPTs

We use the notion of *path* to describe a sequence of models of interest in a run of transformations. Let $(\mathcal{M}, M_0, M_n, \delta)$ be a transformation and δ^k be the k-fold composition of δ for $k \geq 1$. Then a *path* of the run $M_0, M_1, \ldots, M_n$ is a sequence $\langle M_{k_1}, \ldots, M_{k_m} \rangle$ of models satisfying the conditions: $M_{k_1} = M_0, M_{k_m} = M_n$ and $\delta^k(M_{k_i}) = M_{k_{i+1}}$ for $k_{i+1} = k_i + k$. The *length* of a path $\langle M_{k_1}, \ldots, M_{k_m} \rangle$ is the number (i.e., m) of models in the path. The shortest path of a run is the pair $\langle M_0, M_n \rangle$ of initial and final models with the length 2.

Let M and M^* be two models respectively defined over the schemata $S_A \in \mathcal{S}_A$ and $S_C \in \mathcal{S}_C$. A *location invariant* between M and M^*, denoted as $M \approx^{(\gamma)} M^*$, describes that the model M is similar to M^* because objects in M are translated into M^* by a concretization function γ between abstract and concrete models, i.e.,

$$M \approx^{(\gamma)} M^* \equiv M \in \mathcal{M}(S) \wedge M^* \in \mathcal{M}(\gamma(S)) \wedge M^* = \gamma(M).$$

Definition 8. Let Π be a PPT. Then Π^* is a *correct refinement* of Π, denoted as $\Pi \hookrightarrow \Pi^*$, iff Π^* is a PPT and for any run of Π^* with a path $\left\langle M^*_{i_1}, \ldots, M^*_{i_m} \right\rangle$ of interest, there exists a run of Π with a path $\langle M_{j_1}, \ldots, M_{j_n} \rangle$ of the same length n such that, for $k = 1, \ldots, n$,

- $M_{j_k} \approx^{(\gamma)} M^*_{i_k}$.

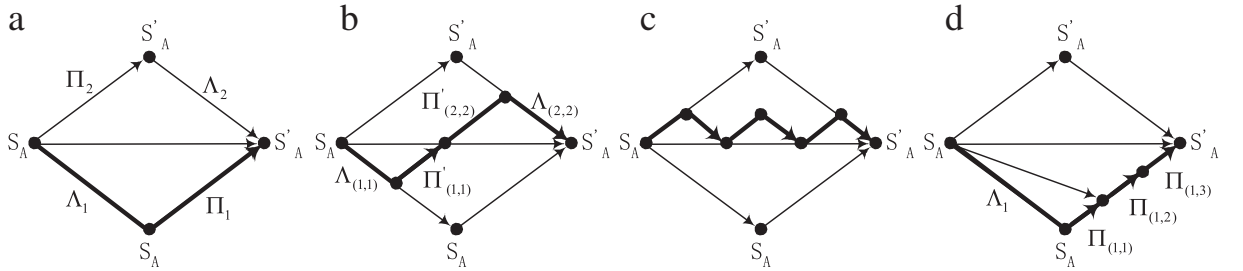

Fig. 11. Refinement of transformations under different migration strategies.

Definition 8 states that for any model of interest in the run of a refined PPT Π^* there is a corresponding model in the run of the abstract PPT Π. A general description of the refinement of PPTs is illustrated in Fig. 12(a). Π transforms a model over schema (T_0, Σ_0) to another model over schema (T_n, Σ_n), and correspondingly, Π^* transforms a model over schema (T_0^*, Σ_0^*) to another model over schema $(T_n{}^*, \Sigma_n{}^*)$. If we describe the mapping from (T_0, Σ_0) to (T_n, Σ_n) as f and the mapping from (T_0^*, Σ_0^*) to $(T_n{}^*, \Sigma_n{}^*)$ as f^*, then we have $\gamma(f(T_0)) = f^*(\gamma(T_0))$ and $\gamma(f(\Sigma_0)) \Leftrightarrow f^*(\gamma(\Sigma_0))$.

Example 8. Consider P_A and $P_A{}^*$ in Example 1 again. Suppose that the path of our interest in P_A is $\langle M_0, M_1, M_2, M_3, M_4, M_5, M_6\rangle$ where M_0 is the initial model, and $M_i (i>0)$ represents the model obtained after sequentially executing the ith forall rule in the run of P_A, and the corresponding path of our interest in $P_A{}^*$ is $\langle M_0^*, M_1^*, M_2^*, M_3^*, M_4^*, M_5^*, M_6^*\rangle$ where M_0^* is the initial model, and $M_i{}^* (i>0)$ represents the model obtained after sequentially executing the ith forall rule in the run of $P_A{}^*$.

Because P_A and $P_A{}^*$ are both a PPT, and for any run of $P_A{}^*$ with a path $\langle M_0^*, \ldots, M_6^*\rangle$ of interest, there exists a run of P_A with a path $\langle M_0, \ldots, M_6\rangle$, $P_A{}^*$ is thus a correct refinement of P_A in terms of the path $\langle M_0^*, M_1^*, M_2^*, M_3^*, M_4^*, M_5^*, M_6^*\rangle$. □

7.2. Refinement of PETs

For the refinement of PETs we need the notion of constraint invariant to capture the invariant of constraints between corresponding models. Let T be a set of object types and Ψ be a set of constraints defined over T. Then a coupling *constraint invariant* between two corresponding models M and M^* with respect to Ψ (denoted as $M \approx^{(\gamma,\Psi)} M^*$) describes that models M and M^* are semantically similar in the sense that both models satisfy constraints in Ψ or corresponding constraints in an appropriate form. Formally speaking, it is defined as

$$\begin{aligned} M \approx^{(\gamma,\Psi)} M^* \equiv\ & M \in \mathcal{M}(S) \wedge M^* \in \mathcal{M}(\gamma(S)) \\ & \wedge M^* = \gamma(M) \\ & \wedge M \vDash \bigwedge_{\varphi \in \Psi} \varphi \wedge M^* \vDash \bigwedge_{\varphi \in \gamma(\Psi)} \varphi. \end{aligned}$$

Definition 9. Let Ψ be a set of constraints and Λ be a PET over Ψ. Then Λ^* is a *correct refinement* of Λ, denoted as $\Lambda \hookrightarrow \Lambda^*$, iff Λ^* is a PET and for any run of Λ^* with a path $\langle M_{i_0}^*, \ldots, M_{i_m}^*\rangle$ of interest, there exists a run of Λ with a path $\langle M_{j_0}, \ldots, M_{j_n}\rangle$ of the same length n such that, for $k = 1,\ldots,n$, $\bigwedge_{0 \le p < q \le n} \Psi_p \subseteq \Psi_q$, $\Psi_0 = \varnothing$ and $\Psi_n = \Psi$,

- $M_{i_k}^* \approx^{(\Psi_k,\gamma)} M_{j_k}$.

The above definition states that each pair of corresponding models is equivalent with respect to a subset $\Psi_k \subseteq \Psi$ of constraints and a concretization translation γ. The subset of constraints satisfied by a pair of models of interest should also be satisfied by the pair of successor models of interest. Eventually, the final models of both transformations satisfy all the constraints in Ψ or $\gamma(\Psi)$. According to the definition of PET, both Λ^* and Λ remain the valid parts of each model, in terms of a subset of constraints for its successor model, in the successor model. Fig. 12(b) illustrates this refinement process in which Λ transforms a model over schema (T, Σ) to another model over schema $(T, \Sigma \cup \Psi)$, and correspondingly, Λ^* transforms a model over schema (T^*, Σ^*) to another model over schema $(T^*, \Sigma^* \cup \Psi^*)$ where $T^* = \gamma(T)$, $\Sigma^* = \gamma(\Sigma)$ and $\Psi^* = \gamma(\Psi)$.

Example 9. Consider P_B and $P_B{}^*$ for data cleaning in Example 2, the path $\langle M_0, M_1, M_2, M_3, M_4\rangle$ of P_B where M_0 is the initial model and M_i represents the model obtained after applying the 1st forall rule, and similarly the path $\langle M_0^*, M_1^*, M_2^*, M_3^*, M_4^*\rangle$ of $P_B{}^*$. Then $P_B{}^*$ is a correct refinement of P_B in terms of the path $\langle M_0^*, M_1^*, M_2^*, M_3^*, M_4^*\rangle$ because P_B and $P_B{}^*$ are both a PET, and the following conditions hold:

- $M_0^* \approx^{(,\gamma)} M_0$;
- $M_1^* \approx^{(\phi_1,\gamma)} M_1$;
- $M_2^* \approx^{(\phi_1 \wedge \phi_2,\gamma)} M_2$;
- $M_3^* \approx^{(\phi_1 \wedge \phi_2 \wedge \phi_3,\gamma)} M_3$;
- $M_4^* \approx^{(\phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4,\gamma)} M_4$.

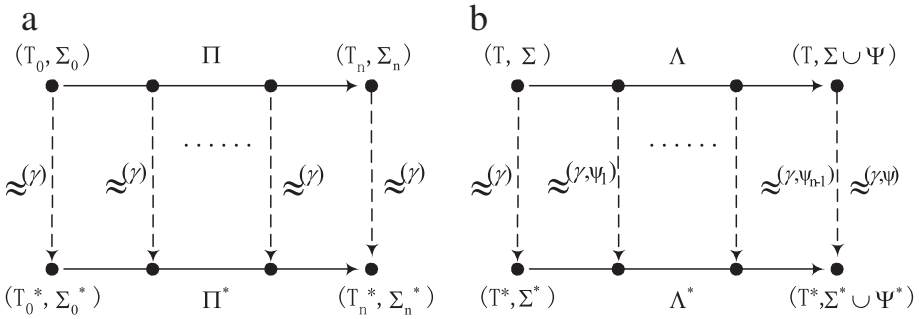

Fig. 12. (a) Refinement of PPTs and (b) refinement of PETs.

However, if the 3rd and 4th forall rules in P_B are swapped and the resulting transformation is called P_C for the convenience of expression, then the refined transformation $P_C{}^*$, that is obtained by replacing $\text{CLEANUPINSURANCE}_i(x)$ $(1 \le i \le 4)$ of P_C with the same rules defined in Example 2, is *not a correct refinement* of P_C. This is because $\text{CLEANUPINSURANCE}_3(x)$ will insert data that violate the constraint φ_4 and thus $M_4^* \approx^{(\phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4, \gamma)} M_4$ does not hold between P_C and $P_C{}^*$. □

7.3. Simplified abstraction

Now we discuss the converse case of refinement, i.e., abstraction. In order to control complexity of data migration tasks by improving efficiency of static analysis, it is important to investigate how to simplify an abstract transformation under the preservation of certain properties of interest. The intuition behind obtaining a simplified abstract transformation is to approximate an abstract transformation by neglecting computation segments that are irrelevant to properties of interest, for example, computation segments that have no effect on objects relating to the properties of interest.

Let M_A be an abstract model over a schema $S_A = (T_A, \Sigma_A)$ and Φ be a set of properties of M_A. Then we say that an abstract model M_A' over a schema $S_A' = (T_A', \Sigma_A')$ keeps *the same set Φ of properties of M_A* if M_A' satisfies the corresponding set Φ' of properties, where Φ' is defined by applying the abstraction function $\alpha' : \mathcal{M}(S_A) \to \mathcal{M}\left(S'_A\right)$ over Φ as appropriate. Then M_A' is said to be a *simpler abstraction* of M_A with respect to Φ, denoted as $M'_A \approx^{(\alpha,\Phi)} M_A$, if M_A' keeps the same set Φ of properties of M_A. Every simpler abstraction of M_A contains equal or less information than M_A. In fact, all simpler abstractions of an abstract model M_A with respect to a set Φ of properties can be viewed as being *partially equivalent* with respect to Φ.

Let $P_A = (\mathcal{M}, M_0, M_n, \delta)$ be an abstract transformation and Ψ be a set of properties of P_A. We use $F(\Psi)$ to denote the set of all subformulae of Ψ. Then an abstract transformation P_A' is a *simpler abstraction* of P_A with respect to Ψ iff for any run of P_A with a path $\langle M_{i_1}, \ldots, M_{i_m} \rangle$ of interest, there exists a run of P_A' with a path $\left\langle M'_{j_1}, \ldots, M'_{j_n} \right\rangle$ of the same length n such that, for $k = 1,\ldots,n$,

- each model M_{j_k}' of P_A' is a simpler abstraction of the corresponding model M_{i_k} of P_A with respect to the set Ψ_A of properties, where $\Psi_A = \{\varphi \in F(\Psi) \mid M_{i_k} \models \varphi\}$.

Fig. 13 shows that the simplification of an abstraction consists of (m,n)-refinements [9] $(m<n)$, i.e., m steps of M corresponds to n steps of M^*, where $M^* = (\mathcal{M}^*, M_0^*, M_n^*, , \delta^*)$ for M_0^* over schema (T_0^*, Σ_0^*) and $M_n{}^*$ over schema $(T_n{}^*, \Sigma_n{}^*)$ is an abstract transformation, and $M = (\mathcal{M}, M_0, M_n, \delta)$ for M_0 over schema (T_0, Σ_0) and M_n over schema (T_n, Σ_n) is another abstract transformation simpler than M^*. Ideally, given a set of properties for an abstract transformation, we want to find the simplest abstraction of such an abstract transformation. However, solutions for finding the simplest abstraction can be potentially complicated (if it exists). We do not further address this issue in this paper.

7.4. Generic proof method

Given two refinements $P_1 \hookrightarrow P_1^*$ and $P_2 \hookrightarrow P_2^*$. If P_1 and P_2 can be composed and respectively P_1^* and P_2^* can be composed, then $P_2^* \circ P_1^*$ is a refinement of $P_2 \circ P_1$. Fig. 14 presents a general refinement scheme for migration transformations. The transformation $\Lambda_1 \circ \Pi_1$ (resp. $\Pi_2 \circ \Lambda_2$) can be refined to $\Lambda_1^* \circ \Pi_1^*$ (resp. $\Pi_2^* \circ \Lambda_2^*$), where $\Lambda_1 \hookrightarrow \Lambda_1^*$ and $\Pi_1 \hookrightarrow \Pi_1^*$ (resp. $\Lambda_2 \hookrightarrow \Lambda_2^*$ and $\Pi_2 \hookrightarrow \Pi_2^*$).

In [32] Schellhorn presented a generic proof method for the correctness of refinements of ASMs, in which invariants are established based on the notion of commuting diagrams. Here we can extend this generic proof method to verify the refinement correctness of transformations in data migration. As it has been well studied [9,10] that the ASM refinement is a practically useful method for proving properties in system design, the theory we developed here thus provides a general scheme for proving the properties of a migration transformation from a legacy system that may have a number of heterogeneous data sources to a unified new system. More specifically, to prove that a migration transformation P^* at the implementation level has certain property φ, we take the following steps:

- specify the abstract transformation P that migrates data from the legacy system to an abstract model of the new system, which is the composition of a number of PPTs and PETs;
- simplify the abstract transformation P into a simpler abstraction P' with respect to the property φ;
- prove that an appropriate abstract form of the property in question holds on the simplified abstract transformation P';

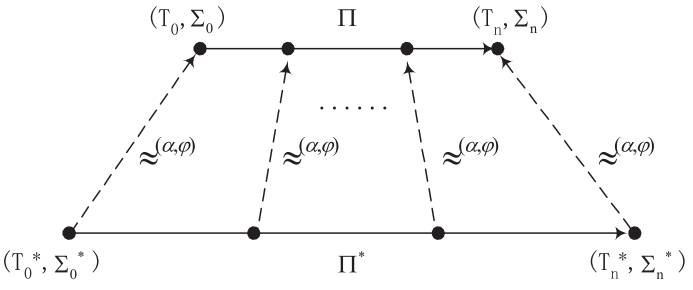

Fig. 13. Simplification of an abstraction.

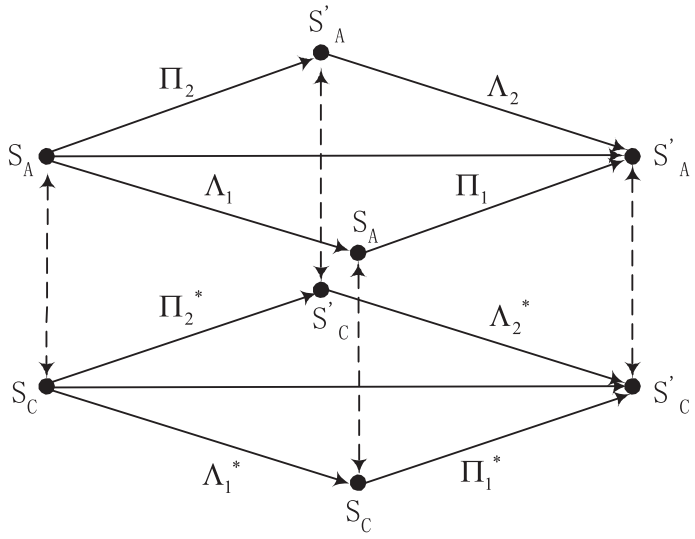

Fig. 14. A refinement scheme for migration transformations.

- prove the transformation P^* in question to be a correct refinement of P', i.e., each PPT Π^* or PET Λ^* included in P^* is a correct refinement of the corresponding PPT Π or PET Λ included in P'.

Example 10. Suppose that we want to verify the following property φ of the transformation $P_A{}^*$ in Example 1:

$$\forall x_1, x_2.\left(\text{TRUCK}^{S_{legacy}}(x_1, x_2) \Rightarrow \exists x_3, x_4.\text{CARRIER}^{S_{new}}(x_1, \text{“truck”}, x_3, x_2, x_4, \text{“A”})\right)$$

where TRUCK^{Slegacy} denotes TRUCK in S_{legacy} and CARRIER^{Snew} denotes CARRIER in S_{new}. The property φ states that all objects in TRUCK^{Slegacy} should be extracted as objects of CARRIER^{Snew} with CODE = TRUCK^{Slegacy} · NO, TYPE = “truck”, PRODYEAR = TRUCK^{Slegacy} · PRODYEAR and SOURCE = “A”.

We observe that objects relevant to this property are in TRUCK^{Slegacy} and CARRIER^{Snew}. Moreover, those objects in CARRIER^{Snew} whose TYPE is not “truck” are also not relevant. Therefore, the following simpler abstract transformation $P_A{}^\circ$ with respect to φ can be obtained, which are simplified by removing irrelevant computation segments and objects. We use λ to indicate a value whose content is trivial in an abstract transformation.

forall x **with** $x \in$ TRUCK **do**
  EXTRACTEDTOCARRIER$^\circ$(x, “truck”, “A”)
enddo
where EXTRACTEDTOCARRIER$^\circ$(x, y, z) =
  if y = “truck” **then**
    INSERTINTOCARRIER(x.NO, “truck”, λ, x.PRODYEAR, “NZ Trans”, z)
  endif

We can prove that the property φ holds on the above transformation $P_A{}^\circ$. Since $P_A{}^*$ is a correct refinement of $P_A{}^\circ$, the property φ also holds on $P_A{}^*$. □

Remark. Designing data migration transformations is challenging and their implementation can be very complicated. Formal methods like ASMs and B provide us a mathematically solid approach for managing the complexity of the data migration process. Nevertheless, it is well-known that the state explosion problem may occur in model checking when verifying complex systems. Therefore, abstraction techniques become particularly useful for controlling the complexity since they remove irrelevant details of the original design with respect to the property under consideration, and we thus only need to verify a simplified transformation which is more efficient than verifying the original transformation.

8. Related work

Data migration is an important but often overlooked aspect of system modernization. In contrast to numerous industry interests (e.g., [1,2,23]), very little research attention has been given to the theoretical foundations of this subject in its own right.

Previous studies [13,31,36] on system modernization mostly consider only the legacy interface to data by using wrapping — a "black-box" reengineering technique, which hides the unwanted complexity by ignoring the internals of data migration. In [3], the formal method B is used for specifying and verifying changes made on a model, which can generate a data migration transformation between the old and the new data representations. There have also been several research efforts [5,16,21] in investigating algorithms for the data migration problem within a network environment by using techniques from a graph theory, which merely focus on moving data stored on devices in a network. In these cases, many important issues of data migration, such as data quality control, have been disregarded.

In this paper we use abstract state machines (ASMs) to express various migration transformations. ASMs are a practical and scientifically well-founded systems engineer method invented by [17], extensively discussed in [10] and applied in solving various database-related problems in [34,37].

One of the two subclasses of migration transformations – property-preserving transformations – captures schema and data translations in general, and is thus related to various approaches for schema mapping or model translations, e.g., model-independent translations [6], and schema transformations [18,19,22]. Moreover, our notion of reflecting a relation between two models has links with the notion of relative information capacity (i.e., information-capacity dominance and equivalence) studied in the area of semantic heterogeneity [24,25,28]. The main difference between them is that we consider not only schema transformations but also translations that take into account concrete data with implicit semantics.

There are several migration strategies proposed to manage data migration projects [7,8,11,39], including the big bang, chicken little and butterfly approaches. In [27], advantages and disadvantages of each migration strategy are discussed in detail, and factors that affect decision making on which strategy is best suited for a specific data migration project have been analysed.

The ASM refinement method and its connection with other refinement approaches have been well studied in [9]. Schellhorn [32,33] proposes a generic proof method for the correctness of refinements of ASMs. We extend his work by simplifying abstractions relative to the properties of interests in this paper.

9. Conclusion

Data migration hardly exists in isolation. Nevertheless, it has severe impacts on an overall project such as system migration and system evolution projects. Our work here is the first step towards the formal development for system modernization. We first established the links between abstract and concrete models by extending Galois connections in abstract interpretation. Two model spaces at different levels of abstraction are connected by a pair of abstraction and concretization functions that translate concrete models into abstract models and vice versa. Then we discussed the ETL process in the setting of data migration, in which two fundamental subclasses of migration transformations are identified. In order to link a high-level specification of migration transformations into the ones at an implementation level, we developed a general refinement theory for data migration. In particular, our refinement theory can be further extended to systematically and efficiently prove properties of migration transformations.

In the future we plan to investigate how our theory can be extended to support system evolution and system migration in a broader sense. Taking functionality transformations, wrapping of old applications, etc. into consideration will certainly bring in additional complexity into the refinement scheme. We will investigate these problems in the future.

References

[1] Rapid Application Development (RAD) for Data Migration — White Paper Solutions by Premier International, http://www.premier-international.com/pdf/Applaud_White_Paper.pdf 2004.
[2] A Roadmap to Data Migration Success — White Paper by SAP, http://whitepapers.technologyevaluation.com/view_document/22180/a-roadmap-to-data-migration-success.html 2008.
[3] M. Aboulsamh, J. Davies, Specification and verification of model-driven data migration, Model and Data Engineering (2011) 214–225.
[4] A.V. Aho, J.D. Ullman, Universality of data retrieval languages, in: Proceedings of Principles of Programming Languages, ACM, 1979, pp. 110–119.
[5] E. Anderson, J. Hall, J. Hartline, M. Hobbes, A. Karlin, J. Saia, R. Swaminathan, J. Wilkes, Algorithms for data migration, Algorithmica 57 (2) (2010) 349–380.
[6] P. Atzeni, P. Cappellari, R. Torlone, P. Bernstein, G. Gianforme, Model-independent schema translation, The VLDB Journal 17 (6) (2008) 1347–1370.
[7] J. Bisbal, D. Lawless, B. Wu, J. Grimson, Legacy information systems: issues and directions, IEEE Software 16 (5) (1999) 103–111.
[8] J. Bisbal, D. Lawless, B. Wu, J. Grimson, V. Wade, R. Richardson, D. O'Sullivan, A Survey of Research into Legacy System Migration, 1997.
[9] E. Börger, The ASM refinement method, FAC 15 (2) (November 2003) 237–257.
[10] E. Börger, R.F. Stärk, Abstract State Machines: A Method for High-level System Design and Analysis, Springer, 2003.
[11] M. Brodie, M. Stonebraker, Migrating Legacy Systems: Gateways, Interfaces & the Incremental Approach, Morgan Kaufmann Publishers Inc., 1998
[12] E.F. Codd, A relational model of data for large shared data banks, Communications of the ACM 13 (June 1970) 377–387.
[13] S. Comella-Dorda, K. Wallnau, R. Seacord, J. Robert, A survey of black-box modernization approaches for information systems, in: Proceedings of Software Maintenance, IEEE, 2000, pp. 173–183.
[14] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, Tree Automata Techniques and Applications, http://www.grappa.univ-lille3.fr/tata 2007.
[15] P. Cousot, R. Cousot, Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, in: Proceedings of Principles of Programming Languages, ACM, 1977, pp. 238–252.
[16] R. Gandhi, J. Mestre, Combinatorial algorithms for data migration to minimize average completion time, Algorithmica 54 (1) (2009) 54–71.
[17] Y. Gurevich, A new thesis (abstracts), AMS 6 (4) (August 1985) 317.
[18] J. Hainaut, Specification preservation in schema transformations — application to semantics and statistics, Data & Knowledge Engineering 19 (2) (1996) 99–134.
[19] J. Hainaut, The transformational approach to database engineering, Generative and Transformational Techniques in Software Engineering (2006) 95–143.
[20] J. Hainaut, J. Henrard, J. Hick, D. Roland, V. Englebert, Database design recovery, in: Advanced Information Systems Engineering, Springer, 1996, pp. 272–300.
[21] J. Hall, J. Hartline, A. Karlin, J. Saia, J. Wilkes, On algorithms for efficient data migration, in: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 2001, pp. 620–629.

[22] J. Hick, J. Hainaut, Database application evolution: a transformational approach, Data & Knowledge Engineering 59 (3) (2006) 534–558.
[23] P. Howard, C. Potter, Data Migration in Global 2000, Research, Forecasts and Survey Results — A Survey Paper by Bloor Research, http://www.bloorresearch.com/research/survey/876/data_migrtaion_survey.html 2007.
[24] R. Hull, Relative information capacity of simple relational database schemata, in: Proceedings of Principles of Database Systems, ACM, 1984, pp. 97–109.
[25] R. Hull, Managing semantic heterogeneity in databases: a theoretical prospective, in: Proceedings of Principles of Database Systems, ACM, 1997, pp. 51–61.
[26] N. Immerman, Expressibility as a complexity measure: results and directions, in: Second Structure in Complexity Conference, 1987, pp. 194–202.
[27] M. Klettke, B. Thalheim, Evolution and migration of information systems, in: The Handbook of Conceptual Modeling: Its Usage and Its Challenges, Springer, 2011, pp. 381–420, (ch. 12).
[28] R. Miller, Using schematically heterogeneous structures, in: SIGMOD Record, ACM, 1998, pp. 189–200.
[29] M. Otto, The expressive power of fixed-point logic with counting, Journal of Symbolic Logic 61 (1) (1996) 147–176.
[30] D. Parnas, Software aging, in: Proceedings of Software Engineering, IEEE, 1994, pp. 279–287.
[31] M. Roth, P. Schwarz, W. Don't Scrap It, A wrapper architecture for legacy data sources, in: Proceedings of Very Large Data Bases, 1997, pp. 266–275.
[32] G. Schellhorn, Verification of ASM refinements using generalized forward simulation, Journal of Universal Computer Science 7 (11) (2001) 952–979.
[33] G. Schellhorn, ASM refinement and generalizations of forward simulation in data refinement: a comparison, Theoretical Computer Science 336 (2–3) (2005) 403–435.
[34] K.-D. Schewe, Q. Wang, A customised ASM thesis for database transformations, Acta Cybernetica 19 (4) (2010) 765–805.
[35] D.A. Schmidt, Binary relations for abstraction and refinement, in: Workshop on Refinement and Abstraction, Elsevier, 1999.
[36] P. Thiran, J. Hainaut, G. Houben, D. Benslimane, Wrapper-based evolution of legacy information systems, ACM Transactions on Software Engineering and Methodology 15 (4) (2006) 329–359.
[37] Q. Wang, Logical Foundations of Database Transformations for Complex-value Databases, Logos Verlag, Berlin, Germany, 2010.
[38] B. Wu, D. Lawless, J. Bisbal, J. Grimson, V. Wade, D. O'Sullivan, R. Richardson, Legacy system migration: a legacy data migration engine, in: DATASEM97, 1997, pp. 129–138.
[39] B. Wu, D. Lawless, J. Bisbal, R. Richardson, J. Grimson, V. Wade, D. O'Sullivan, The butterfly methodology: a gateway-free approach for migrating legacy information systems, in: Proceedings of Engineering of Complex Computer Systems, IEEE, 1997, pp. 200–205.