Towards a Theory of Refinement for Data Migration

ER 2011

Brussels

Nov 2, 2011

(& Qing Wang[1])& Bernhard Thalheim[2]

[1] University of Otago, Dunedin, New Zealand,

[2]Christian-Albrechts-University Kiel, Germany & Lomonosov University Moscow, Russia

qing.wang@otago.ac.nz **thalheim@is.informatik.uni-kiel.de**

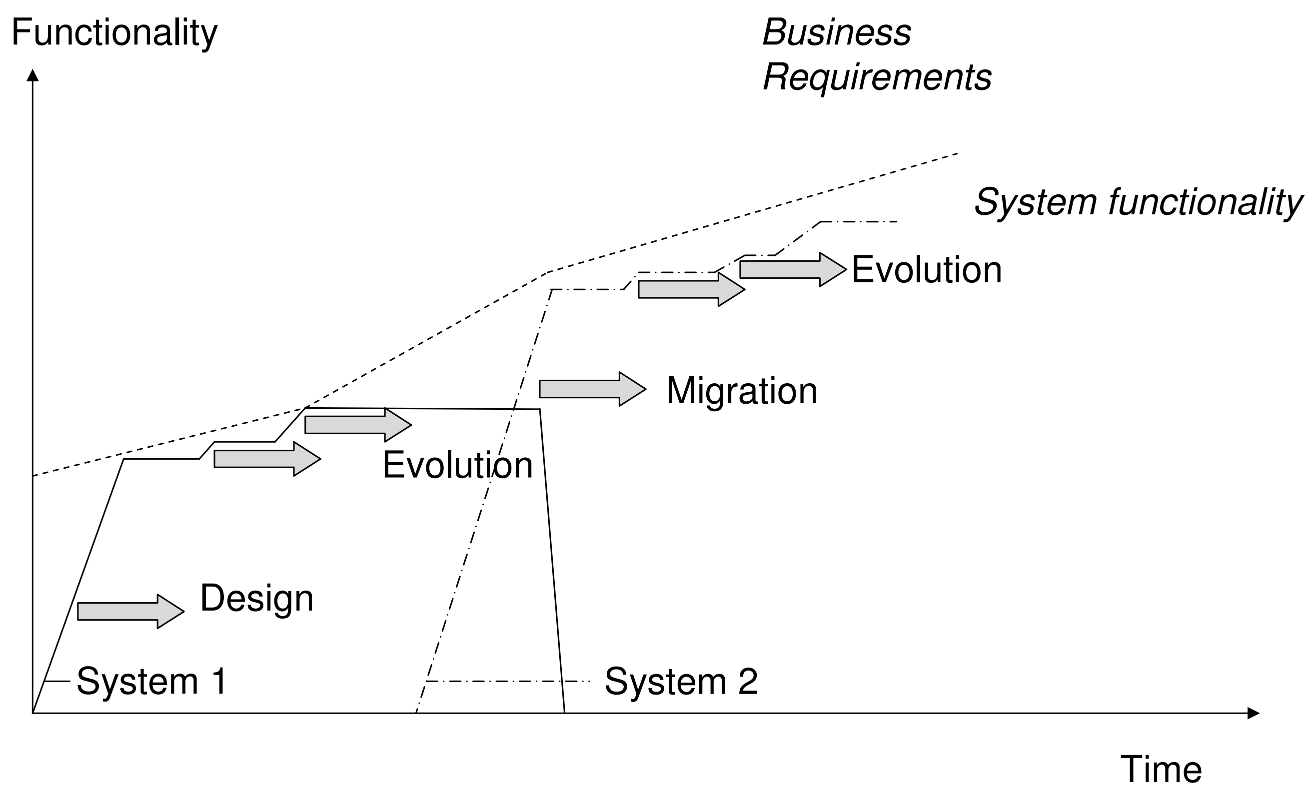
Introduction

- Data migration is a fundamental aspect of projects on modernising legacy systems.
- Business companies annually invest billions of dollars in data migration tasks,
  - e.g., over 5bn from the top 2000 global companies in 2007.
- However, only 16% projects have been successfully accomplished.
- One of main reasons is the lack of a well-defined methodology that can help handle the complexity of data migration tasks.

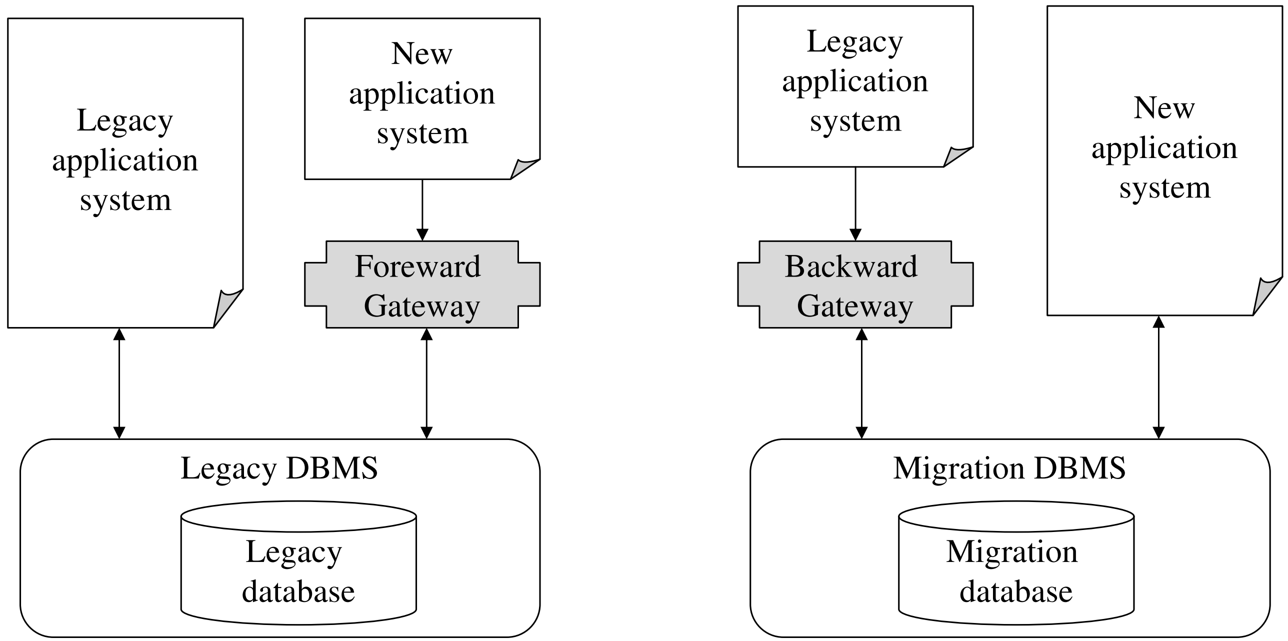
Evolution and Migration are Interwoven

Refinement for Data Migration
Nov 2, 2011
©*Qing Wang/β*

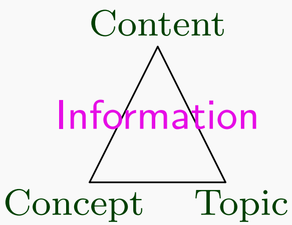
Content
Information
Concept Topic



Wrapper-Based Migration



Why
ETL basis
Refinement schema
Legacy kernel
Transformations
Strategies
Properties
Finally

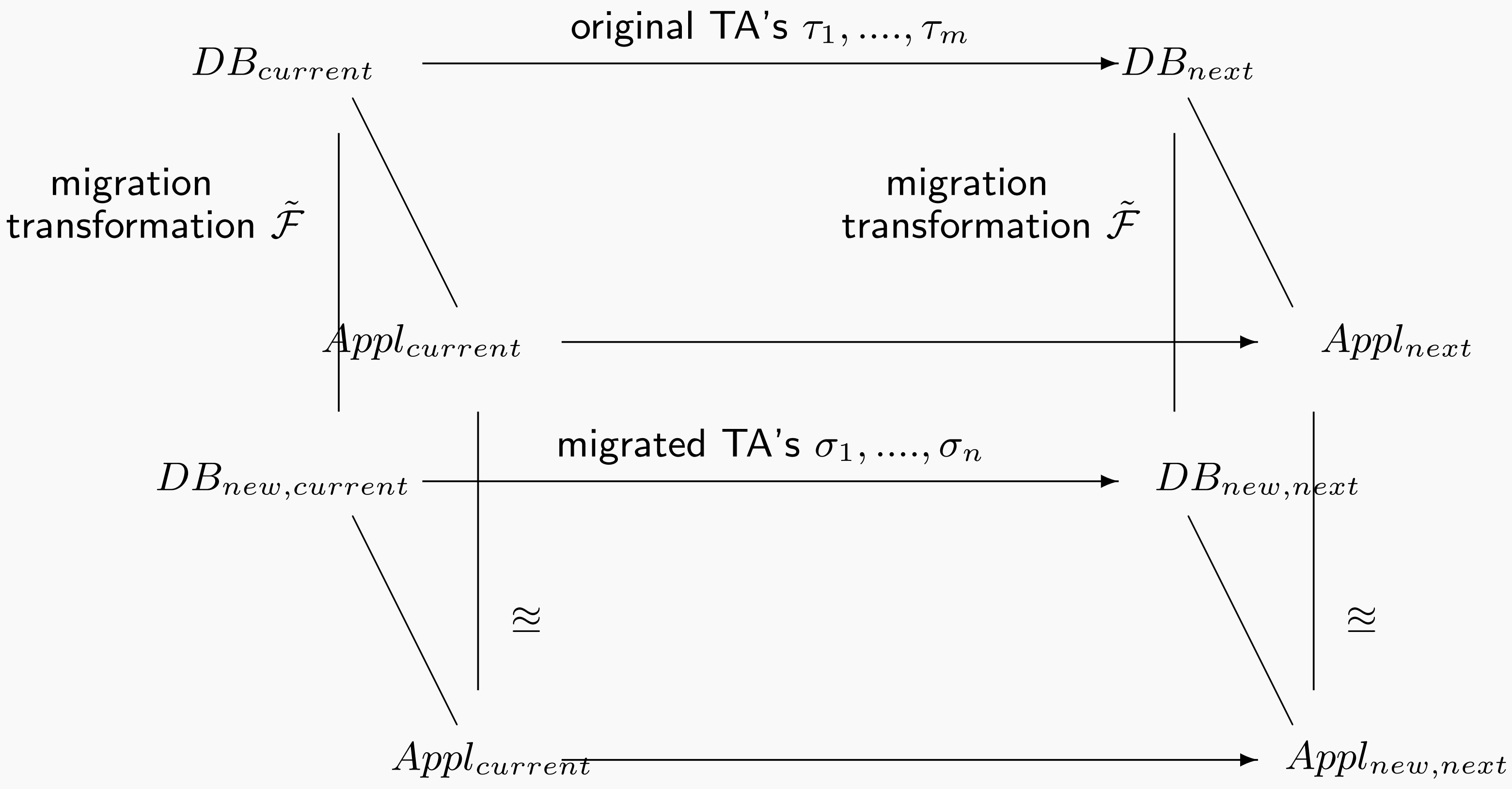


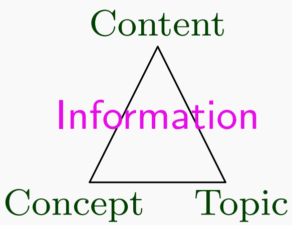
Problems of Data Migration

- Heterogeneous data sources that are
  - designed by using different data modelling tools, or
  - interpreted under different semantics;
- Inaccurate, incomplete, duplicate or inconsistent data;
- Additional semantic constraints on data after being migrated;
- Specification changes in order to repair detected problems:
  - about 90% of initial specifications change, and
  - over 25% of specifications change more than once.

Invisibility to Users: Preservation of Database Behaviour

Conservative migration: invisible changes to the database system without impact on the application





Goals of this Paper

- A theoretical framework of refinement for data migration, which provides answers to the following questions.
  - **How can we react to specification changes in a way of keeping track of all relevant aspects the changes may impact on?**

    For example,
    - 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 data was migrated properly?**

    For example,
    - preserving desired data semantics and integrity.

Scope of This Paper: Correct Transformations

Database Schema transformation

Integrity constraint transformation

Corresponding database transformations

Outside Scope

Functionality transformation

View transformation

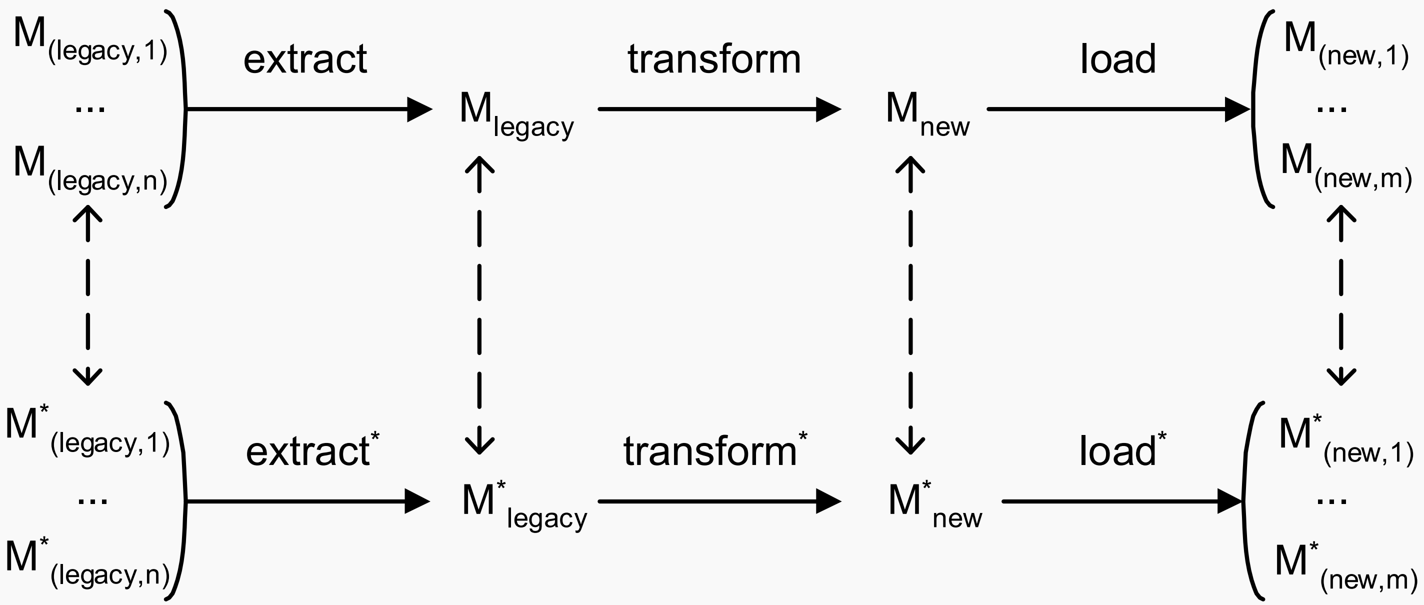
Interface transformation

First: internal variability.

Assuming: conservative migration without cannibalism.

Contributions: ETL in Data Migration

- Our first contribution is the formal development of the ETL processes for data migration.

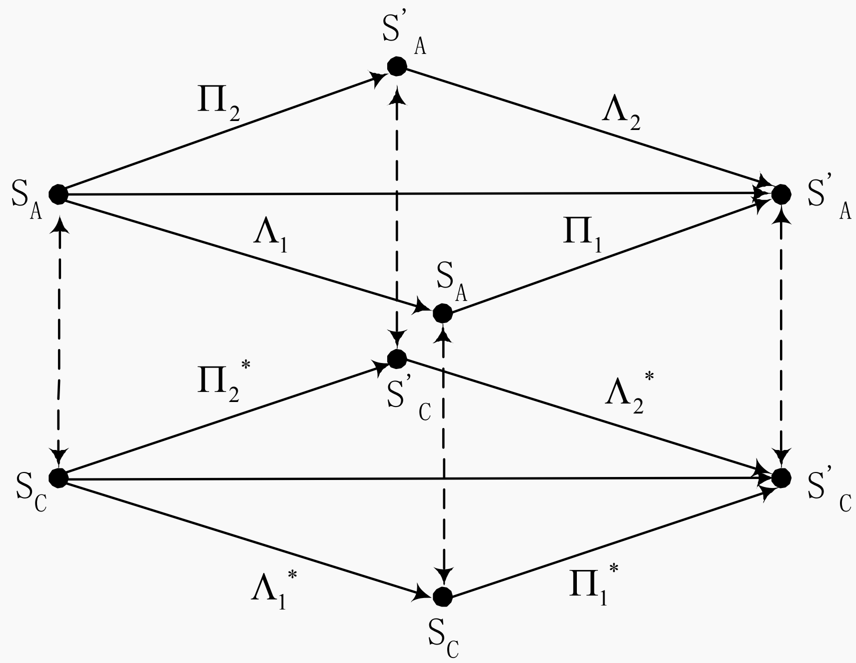


Why
ETL basis
Refinement schema
Legacy kernel
Transformations
Strategies
Properties
Finally

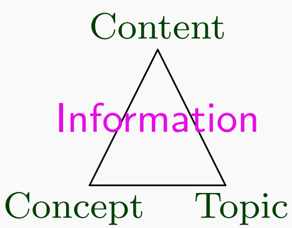
Content
Information
Concept
Topic

Contributions: A Refinement Scheme for Data Migration

- Our second contribution is a refinement scheme specifying the refinement of migration transformations in terms of two types of refinement correctness.



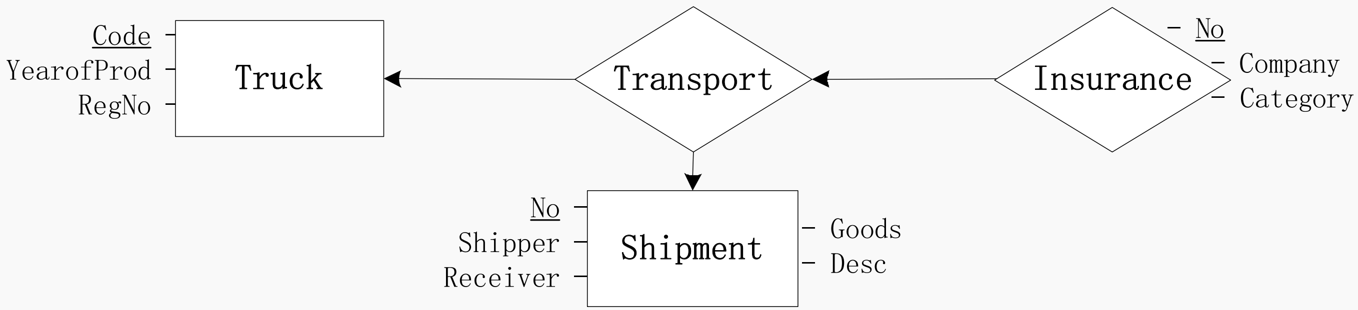
- Target at confluence or Church-Rosser properties.



Schemata and Databases

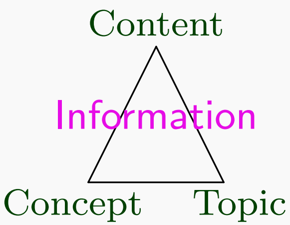
- A **schema** $S = (T, \Sigma)$ consists of a finite, non-empty set T of object types and a finite (possibly empty) set Σ of constraints.
- A **database** over schema (T, Σ) consists of a set of objects, which are of types in T and satisfy every constraint in Σ.

A database is in Tarski sense a **model** of the schema!



$T = \{\textsc{Shipment}, \textsc{Truck}, \textsc{Insurance}, \textsc{Transport}\}$ and $\Sigma = \{\varphi_1, \varphi_2\}$:

- FD: $\varphi_1 \equiv \forall x_1, x_2, x_3, y_1, y_2, y_3.((\textsc{Truck}(x_1, x_2, x_3) \wedge \textsc{Truck}(y_1, y_2, y_3) \wedge x_1 \neq y_1) \Rightarrow x_3 \neq y_3)$,
- InclD: $\varphi_2 \equiv \forall x_1, x_2, x_3, x_4, x_5.(\textsc{Insurance}(x_1, x_2, x_3, x_4, x_5) \Rightarrow \textsc{Transport}(x_4, x_5))$.



Powerful Trick: Legacy Kernel

- **How to discover a legacy kernel that consolidates legacy data sources while preserving constraints of interest?**
  - First stage: For a number of legacy data sources,
    (a) reverse engineer to recover the original design information,
    (b) approximate as abstract databases at a high level abstraction.
  - Second stage: For abstract databases at the same level of abstraction,
    (a) identify constraints to be preserved between abstract databases,
    (b) base the comparison of abstract databases on these constraints.
- A **legacy kernel** is an abstract database that can reflects the abstract database of every legacy data source in terms of their constraints of interest.

Migration Transformations

- A **transformation** $(\mathcal{M}, M_0, M_n, \delta)$ consists of
  - a non-empty set $\mathcal{M}$ of databases together with an initial database $M_0 \in \mathcal{M}$ and a final database $M_n \in \mathcal{M}$, and
  - a one-step transition function δ over $\mathcal{M}$, determined by a rule inductively defined by
    - **update rule:** $\tau(t_1, \ldots, t_n) := t_0$
    - **conditional rule:** **if** φ **then** r **endif**
    - **block rule:** **par** $r_1 \ldots r_n$ **endpar**
    - **sequential rule:** **seq** $r_1 \ldots r_n$ **endseq**
    - **forall rule:** **forall** x **with** φ **do** r **enddo**

Two Subclasses of Transformations

- Each migration transformation is the composition of a finite sequence of PPTs and PETs.
  - **Property-Preserving Transformation** (PPT) Π: transforms from one database to another database, in which the schemata are different but data properties of interest are preserved.
  - **Property-Enhancing Transformation** (PET) Λ : transforms one database violating a certain set of properties into another database satisfying these properties, while their object types remain unchanged.

Example: Property-Preserving Transformations

par

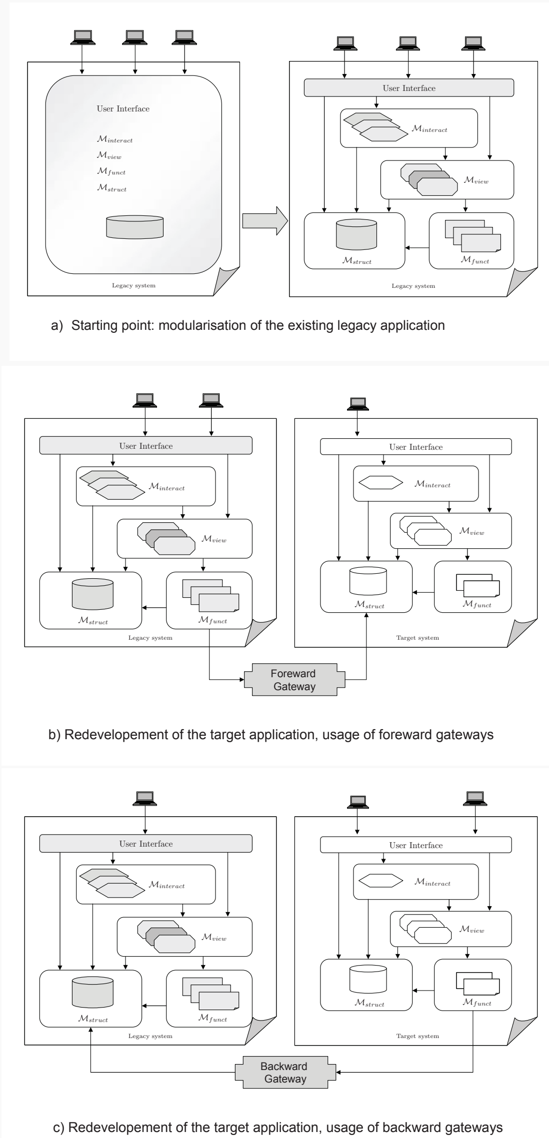
  forall x **with** $x \in$ TRUCK **do**
    MAPPEDTOCARRIER
  enddo
  forall x **with** $x \in$ SHIPMENT **do**
    MAPPEDTOSHIPMENT
  enddo
  forall x **with** $x \in$ TRANSPORT **do**
    MAPPEDTOTRANSPORT
  enddo
  forall x **with** $x \in$ INSURANCE **do**
    MAPPEDTOINSURANCECOMPANYANDINSURANCE
  enddo

endpar

Example: Property-Enhancing Transformations

seq
  forall x **with** $x \in \textsc{Insurance} \wedge \textsc{Invalid}(x)$ **do**
    $\textsc{DeleteInsurance}$
  enddo
  forall x **with** $x \in \textsc{Transport} \wedge \textsc{MissedInsurance}(x)$ **do**
    seq
      $\textsc{SearchRecord}$
      if $\textsc{FoundRecord}$ **then**
        $\textsc{AddIntoInsuranceCompanyAndInsurance}$
      endif
    endseq
  enddo
endseq

Migration Strategy: Chicken Little



(1) Modularise application

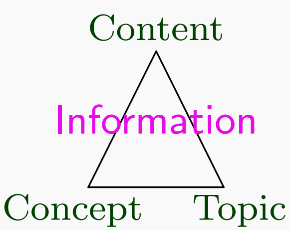
(2) Develop a plan for stepwise migration with running system

(3) Develop forward gateways

(4) Use forward gateways referring to new database

(5) Develop backward gateways

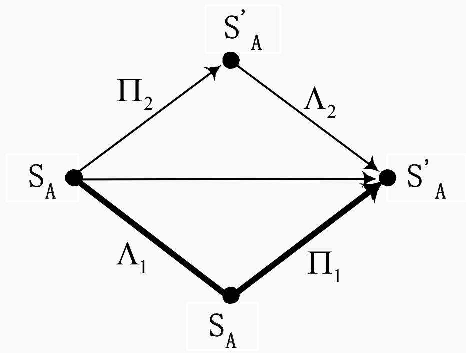
(6) Use backward gateways to old legacy untransferable data



Migration Strategies

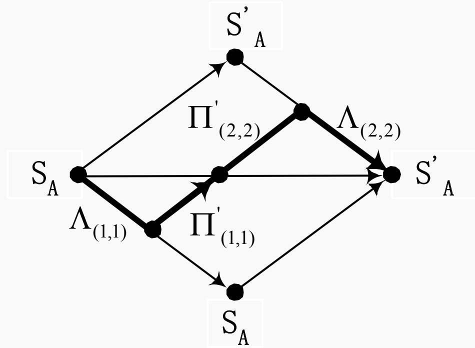
- **Big Bang**

  e.g., $\Lambda_1 \circ \Pi_1$ starts with Λ_1 to cleanse data in the legacy system and then continue with Π_1 to map data into the new data source.



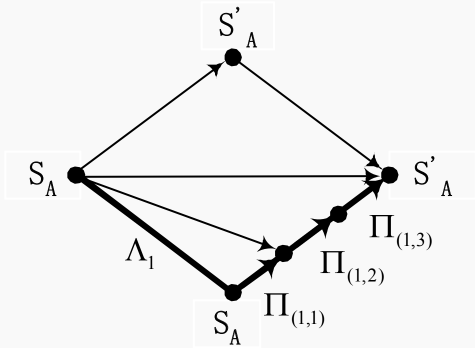
- **Chicken Little**

  e.g., a two-step process consisting of $P_1 = \Lambda_{(1,1)} \circ \Pi'_{(1,1)}$ and $P_2 = \Pi'_{(2,2)} \circ \Lambda_{(2,2)}$.



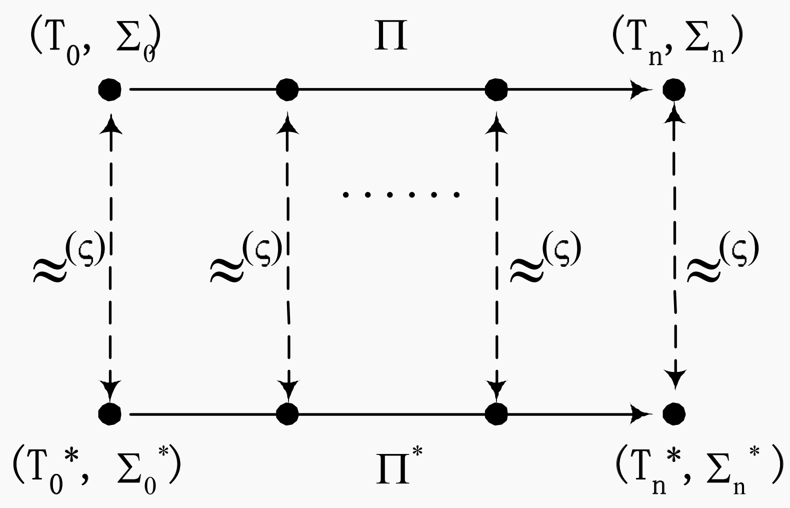
- **Butterfly**

  e.g., $P = (\Lambda_1 \circ \Pi_{(1,1)}) \circ \Pi_{(1,2)} \circ \Pi_{(1,3)}$ transforms the read-only legacy data source and then successively temporary data stores.



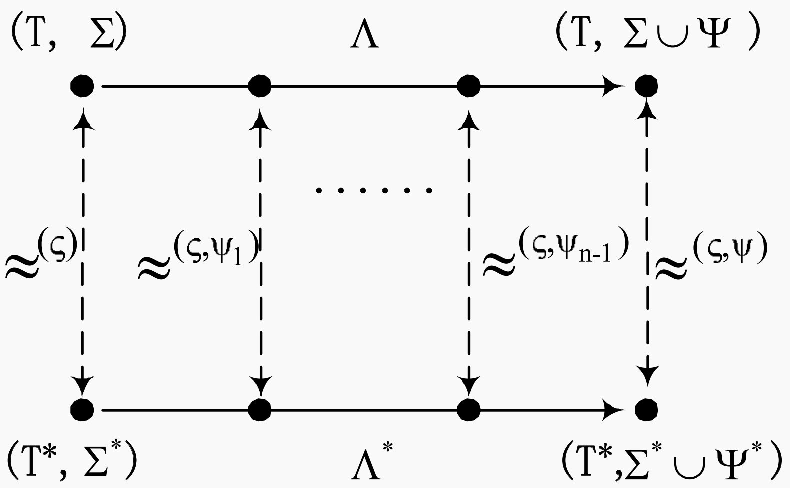
Refinement of Property-Preserving Transformations

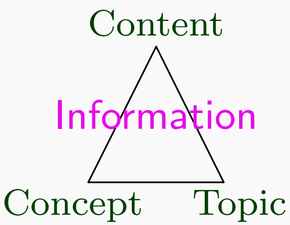
- A **location invariant** between two databases M and M^*, denoted as $M \approx^{(\varsigma)} M^*$, describes that abstract objects in M are translated into more concrete representations in M^*.



Refinement of Property Enhancing Transformations

- A **constraint invariant** between two databases M and M^* with respect to Ψ (denoted as $M \approx^{(\varsigma,\Psi)} M^*$) describes that M and M^* are semantically similar in the sense that they both satisfy constraints in Ψ.





Proving Properties of a Migration Transformation

- To prove that a migration transformation P^* has certain property φ, there are three steps:

(1) Specify the abstract transformation P that migrates data from the legacy system to an abstract database of the new system;

(2) Prove that an appropriate abstract form of the property φ holds on the abstract transformation P;

(3) Prove the transformation P^* to be a correct refinement of P that preserves correctness.

Conclusion

- This paper lays down a formal foundation for investigating data migration.
- We developed a theoretical framework for refining transformations occurring in the process of data migration.
  - A legacy kernel can be discovered at a high-level abstraction which consolidates heterogeneous data sources in a legacy system.
  - Migration transformations can be specified via the composition of two subclasses of transformations at flexible levels of abstraction.
  - With our notions of refinement correctness, the generic proof method by Schellhorn can be used to verify the correctness of properties.
- Our theory can be extended to system migration in a broader sense, which will bring in additional complexity into the refinement scheme.

Thank you!

thalheim@is.informatik.uni-kiel.de