Big Data Integration:
from scalability to accessibility

Angela Bonifati
University Lyon 1 & CNRS Liris (France)

DSC/e
Eindhoven
12 June 2017
Data Science is evolving into a set of principles to make sense of data:

- domain expertise must intervene in data processing
- datasets become larger, with more complex structure
- need of integrating heterogeneous data sources
Big Data Integration (DI) in Data Science

**Interoperability** in data science increasingly important

- seeking expertise in social and professional networks
- integrating related literature in a citation network and in personal repositories
- exchanging large-scale scientific datasets
- exploiting healthcare data across different formats (EHR, data marts, textual data)

....
Interoperability in data science increasingly important

- seeking expertise in social and professional networks
- integrating related literature in a citation network and in personal repositories
- exchanging large-scale scientific datasets
- exploiting healthcare data across different formats (EHR, data marts, textual data)

We need DI principles and tools to accommodate increasing data variety and volume.
Data Exchange as a core process of DI

Data Exchange = translating data under schema $S$ into data under schema $T$ using a set of $s$-$t$ constraints $\Sigma$

Hot research topic in the Database community since its seminal paper\(^1\).

Basic component in commercial data integration products \(^2\).

---


\(^2\) such as IBM InfoSphere DataStage, Altova MapForce, Adeptia, Informatica etc.
Big Data Integration: from scalability to accessibility

**Scalability** – scalable and controllable data exchange
- our first contribution is a **chase-based** algorithm for efficient data exchange with target constraints

**Accessibility** – creation of data exchange constraints
- our second contribution is a **interactive method** for constraint specification suited for domain experts
Table of contents

1. Data Exchange: Background

2. Data Exchange: towards Scalability (Lecture Part I)

3. Data Exchange: towards Accessibility (Lecture Part II)
Background: The Data Exchange Setting

Constraints in $\Sigma$ ($\Sigma_{st} + \Sigma_t$) are first-order formulas that specify the semantic relationship between schemas S and T.

An example of s-t TGD (tuple-generating dependency) is reported below:

$$\forall x \forall y \ S(x, y) \land U(x, z) \rightarrow \exists v \ T(v, y) \land T'(v, z)$$
S-t tgdts correspond to SQL insert statements

A s-t TGD (tuple-generating dependency) of the kind:

$$\forall x \forall y \ S(x, y) \land U(x, z) \quad \rightarrow \quad \exists v \ T(v, y) \land T'(v, z)$$

corresponds (for instance) to an SQL query as follows:

**SQL Insert Command for table** **T**:

```
INSERT INTO T
SELECT append('f_v('', s.y, u.z,')', s.y)
FROM S s, U u
WHERE s.x = u.x;
```

where a skolem function $f_v$ allows value invention in the target table T.
Introduction: constraints in Data Exchange

The Relational Case

Fagin et al.: Data exchange: semantics and query answering. TCS 2005

\[ S \text{ source schema} – \text{a finite collection of relations } R_1, \ldots, R_k \]

\[ T \text{ target schema} – \text{a finite collection of relations } R_1, \ldots, R_l \]

\[ \Sigma_{st} \text{ s-t tgds} – \forall \overline{x}. (\phi_S(\overline{x}) \rightarrow \exists \overline{y}. \psi_T(\overline{x}, \overline{y})) \]

\[ \Sigma_{t} \text{ t tgds} – \forall \overline{x}. (\phi_T(\overline{x}) \rightarrow \exists \overline{y}. \psi_T(\overline{x}, \overline{y})) \]

\[ \Sigma_{t} \text{ t egds} – \forall \overline{x}. (\psi_T(\overline{x}) \rightarrow (x_1 = x_2)) \]

Goal: Compute a target solution \( J \) starting from a source instance \( I \) and \( (S, T, \Sigma_{st} + \Sigma_{t}) \)

- s-t tgds and t tgds: adding missing facts to the target instance \( J \);
- t egds: equate terms in the target instance \( J \) or fail (due to clashes)
Data Exchange: Finding solutions

The Chase in general
- the Chase is a straightforward procedure of constraint enforcement

The Chase in Data Exchange – in reality a two-phase chase

![Diagram showing source-to-target and target chase]

Figure: Exchanging data from source to target.

- source-to-target chase: apply $\Sigma_{st}$ to $I$ and obtain $J^{pre}$ (also called a pre-solution);
- target chase: apply $\Sigma_t$ to $J^{pre}$ and obtain $J$ (also called a solution);
The Classical DE chase procedure for tgds and egds [Fagin et al. TCS’05].

Given a source instance \( I \)

1. Use the naïve chase to chase \( I \) with \( \Sigma_{st} \) and obtain a target instance \( J^{pre} \) (pre-solution).

2. Chase \( J^{pre} \) with the target tgds and the target egds in \( \Sigma_{t} \) to obtain a target instance \( J \) as follows:

   2.1 For target tgds introduce new facts in \( J \) as dictated by the right-hand-side of the s-t tgd and introduce new values (variables) in \( J \) each time existential quantifiers need witnesses.

   2.2 For target egds \( \phi(\bar{x}) \rightarrow x_1 = x_2 \)

      2.2.1 If a variable is equated to a constant, replace the variable by that constant;

      2.2.2 If one variable is equated to another variable, replace one variable by the other variable.

      2.2.3 If one constant is equated to a different constant, stop and report "failure".
The naïve (or Oblivious) chase procedure in pseudocode

---

Input: $M = (S, T, \Sigma_{st}), \text{SrcInstance } I$

Output: $\text{TgtInstance } J$: a universal target solution for $I$ w.r.t. $M$

$J = \emptyset$

for $\forall x. (\phi_S(x) \rightarrow \exists y. \psi_T(x, y)) \in \Sigma_{st}$ do
  for all tuples of constants $\bar{a}$ such that $I \models \phi_S(\bar{a})$ do
    for each $y_i \in \bar{y}$, pick a fresh null value $\bar{N}_i$ for $y_i$
    add the facts in $\psi_T(\bar{a}, \bar{N})$ to $J$
  end for
end for

Return $J$

---
Classical DE Chase - An example

Schemas and source instance

**Source schema:** \{Active_ACTors(name, surname, age),
   Actor_COLL(name_1, surname_1, name_2, surname_2)\}

**Target schema:** \{Actor(name, surname, idRewarding, idClub)\}

**Source instance:**

<table>
<thead>
<tr>
<th>Active_ACTors</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>surname</td>
<td>age</td>
</tr>
<tr>
<td>Leonardo</td>
<td>Di Caprio</td>
<td>42</td>
</tr>
<tr>
<td>John</td>
<td>Redmayne</td>
<td>33</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actor_COLL</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name_1</td>
<td>surname_1</td>
<td>name_2</td>
<td>surname_2</td>
</tr>
<tr>
<td>Leonardo</td>
<td>Di Caprio</td>
<td>Matthew</td>
<td>David</td>
</tr>
<tr>
<td>Fredric</td>
<td>March</td>
<td>Miriam</td>
<td>Hopkins</td>
</tr>
</tbody>
</table>

S-t tgds

\[m_1 : Active_ACTor(n, s, a) \rightarrow Actor(n, s, Y_1, Y_2)\]
\[m_2 : Actor_COLL(n', s', n'', s'') \rightarrow Actor(n', s', E_1, E_2) \land Actor(n'', s'', E_3, E_2)\]

Target egd (fd)

\[e_1 : Actor(n, s, p, w) \land Actor(n, s, p', w') \rightarrow (p = p') \land (w = w')\]
Classical DE Chase - Phase 1 - tgds

Source Instance

<table>
<thead>
<tr>
<th>Active_Actors</th>
<th>Actor_Coll</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>surname</td>
</tr>
<tr>
<td>Leonardo</td>
<td>Di Caprio</td>
</tr>
<tr>
<td>John</td>
<td>Redmayne</td>
</tr>
</tbody>
</table>

S-t tgds

\[ m_1 : \text{Active\_Actor}(n, s, a) \rightarrow \text{Actor}(n, s, Y_1, Y_2) \]
\[ m_2 : \text{Actor\_Coll}(n', s', n'', s'') \rightarrow \text{Actor}(n', s', E_1, E_2) \land \text{Actor}(n'', s'', E_3, E_2) \]

Pre-solution

<table>
<thead>
<tr>
<th>Actor</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>surname</td>
</tr>
<tr>
<td>Leonardo</td>
<td>Di Caprio</td>
</tr>
<tr>
<td>John</td>
<td>Redmayne</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Classical DE Chase - Phase 2 - egds (fds)

Pre-solution - **Actor** table

<table>
<thead>
<tr>
<th>name</th>
<th>surname</th>
<th>idRewarding</th>
<th>idClub</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leonardo</td>
<td>Di Caprio</td>
<td>$N_1$</td>
<td>$N_2$</td>
</tr>
<tr>
<td>Leonardo</td>
<td>Di Caprio</td>
<td>$N_5$</td>
<td>$N_6$</td>
</tr>
<tr>
<td>John</td>
<td>Redmayne</td>
<td>$N_3$</td>
<td>$N_4$</td>
</tr>
<tr>
<td>Matthew</td>
<td>David</td>
<td>$N_7$</td>
<td>$N_6$</td>
</tr>
<tr>
<td>Frederick</td>
<td>March</td>
<td>$N_8$</td>
<td>$N_9$</td>
</tr>
<tr>
<td>Miriam</td>
<td>Hopkins</td>
<td>$N_{10}$</td>
<td>$N_9$</td>
</tr>
</tbody>
</table>

Target egd (functional dependency)

$e_1 : Actor(n, s, p, w) \land Actor(n, s, p', w') \rightarrow (p = p') \land (w = w')$

Solution - **Actor** table

<table>
<thead>
<tr>
<th>name</th>
<th>surname</th>
<th>idRewarding</th>
<th>idClub</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leonardo</td>
<td>Di Caprio</td>
<td>$N_5$</td>
<td>$N_6$</td>
</tr>
<tr>
<td>John</td>
<td>Redmayne</td>
<td>$N_3$</td>
<td>$N_4$</td>
</tr>
<tr>
<td>Matthew</td>
<td>David</td>
<td>$N_7$</td>
<td>$N_6$</td>
</tr>
<tr>
<td>Frederick</td>
<td>March</td>
<td>$N_8$</td>
<td>$N_9$</td>
</tr>
<tr>
<td>Miriam</td>
<td>Hopkins</td>
<td>$N_{10}$</td>
<td>$N_9$</td>
</tr>
</tbody>
</table>
Pros:

- **Sound and complete**: provides a correct Data Exchange Solution iff a solution exists (and catches all no solution cases).

Cons:

- **Overhead**: To apply fds one needs to examine the entire pre-solution, often very large. The larger the fd application scope, the more expensive the fds application.
First Part: Scalable Data Exchange with Target Constraints $^a$ $^b$


Target functional dependencies

Our setting of interest

DE with s-t tgds and target fds, i.e. scenarios where $\Sigma_t$ is a set of functional dependencies.

Such scenarios are:

- widely occurring in practice
- still an issue for state-of-the-art DE engines:
  - *SQL-based* engines offer limited target fd support (often need additional input: source constraints)
  - *Custom chase-based* engines support arbitrary target fds, but at the price of significant overhead.
Our setting of interest

DE with s-t tgds and target fds, i.e. scenarios where $\Sigma_t$ is a set of functional dependencies.

Such scenarios are:

- widely occurring in practice
- still an issue for state-of-the-art DE engines:
  - *SQL-based* engines offer limited target fd support (often need additional input: source constraints)
  - *Custom chase-based* engines support arbitrary target fds, but at the price of significant overhead.

**Goal:** efficient, unlimited support for target fds.
The Interleaved Chase in a nutshell

- Issue: the Classical Data Exchange Chase suffers from **fd overhead**, due to the often **very large fd application scope**.
- Q: How can we mitigate this overhead?
The Interleaved Chase in a nutshell

- Issue: the Classical Data Exchange Chase suffers from **fd overhead**, due to the often **very large fd application scope**.
- Q: How can we mitigate this overhead?

A: The Interleaved Chase

**Exploits the relations among s-t tgds and target fds to:**
- Infer efficient chase orders interleaving s-t tgds and fds
- Thus, tame the size of the intermediate chase result
- Thus, reduce the fd application scope.
DE Chase with assignments

S-t tgd assigment

A mapping from the tgd’s variables to constants or labeled nulls.

- First, build initial assignments for all rules, by taking all the premise’s mappings into the source instance, and completing each with fresh labelled nulls for the existentials.

Initial assignments - example

<table>
<thead>
<tr>
<th>Active_Actors</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>surname</td>
<td>age</td>
<td></td>
</tr>
<tr>
<td>Leonardo</td>
<td>Di Caprio</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>John</td>
<td>Redmayne</td>
<td>33</td>
<td></td>
</tr>
</tbody>
</table>

| Actor_Coll |        |        |        |        |
|------------|--------|--------|--------|
| name₁      | surname₁| name₂  | surname₂|
| Leonardo   | Di Caprio | Matthew | David  |
| Fredric    | March   | Miriam | Hopkins |

\[
m₁: \text{Active}_\text{Actors}(n, s, a) \rightarrow \text{Actor}(n, s, Y₁, Y₂):
\]
\[
a₁^{m₁} = \{n:Leonardo, s:Di Caprio, a:42, Y₁:N₁, Y₂:N₂\}...
\]

\[
m₂: \text{Actor}_\text{Coll}(n', s', n'', s'') \rightarrow \text{Actor}(n', s', E₁, E₂) \land \text{Actor}(n'', s'', E₃, E₂)
\]
\[
a₁^{m₂} = \{n':Leonardo, s':Di Caprio, n'':Matthew, s'': David, E₁:N₅, E₂:N₆, E₃:N₇\}...
\]
DE Chase with assignments

- First, build initial assignments.
- Then chase:
  - Tgd steps pick assignments and add them to the intermediate chase result = target set.
  - Egd (fd) steps modify assignments in the target set.
  - At any point, the (intermediate) target instance can be obtained by 
    *materializing* the assignments in the target set.

### Materialization – example

\[ m_1: \text{Active} \_\text{Actors}(n, s, a) \rightarrow \text{Actor}(n, s, Y_1, Y_2): \]
\[ a_1^{m_1} = \{ n:\text{Leonardo}, s:\text{Di Caprio}, a:42, Y_1:N_1, Y_2:N_2 \} \]

<table>
<thead>
<tr>
<th>Actor</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>surname</td>
</tr>
<tr>
<td>Leonardo</td>
<td>Di Caprio</td>
</tr>
</tbody>
</table>
First, build initial assignments.

Then chase:

- Tgd steps pick assignments and add them to the intermediate chase result = target set.
- Egd (fd) steps modify assignments in the target set
- At any point, the (intermediate) target instance can be obtained by materializing the target set.

Upon chase termination:

- Target set = all assignments (but potentially modified)
- Solution = materialization of the target set.
Saturation Sets

- Egd (fd) steps modify assignments in the target set → *keeping target set size low means keeping fd scope low!*

- **Idea:** during the chase, we can *flush the target set from time to time!*

---

**Saturation Set**

A subset $S$ of the set of tgd assignments $A$ for a scenario, such that an assignment in $S$ is guaranteed never to interact via fd application with an assignment in $A - S$.

If the target set (intermediate result) is a Saturation Set, we can:

- apply fds
- materialize the target set to get *part of the solution*
- ...then clear/flush the target set!
Finding Saturation Sets

Saturation Set
A subset $S$ of the set of tgd assignments $A$ for a scenario, such that an assignment in $S$ is guaranteed never to interact via fd application with an assignment in $A - S$.

- Goal: clever chase order = structure the chase sequence such that consecutive tgd steps correspond to Saturation Sets.
- Q: How does one find (small) Saturation Sets?
- A: Group together assignments that:
  - interact now (because they map to the same constants)
  - or are suspected to interact later on during the chase (because of mutable existential variables whose assigned values are prone to change)
Overlapping assignments

- To build Saturation Sets, group together assignments that interact now or are suspected to interact later
- **Overlap of two assignments** = pairs of equal (or prone to be equal) variables and corresponding involved fd
- When two assignments are overlapping they are placed in the same Saturation Set!

**Overlapping assignments - example**

\[m_1: \text{Active\_Actors}(n, s, a) \rightarrow \text{Actor}(n, s, Y_1, Y_2)\]
\[m_2: \text{Actor\_Coll}(n', s', n'', s'') \rightarrow \text{Actor}(n', s', E_1, E_2) \land \text{Actor}(n'', s'', E_3, E_2)\]
\[e_1: \text{Actor}(n, s, p, w) \land \text{Actor}(n, s, p', w') \rightarrow (p = p') \land (w = w')\]

\[a_1^{m_1} = \{n:\text{Leonardo}, s:\text{Di Caprio}, a:42, Y_1:N_1, Y_2:N_2\}\]
\[a_1^{m_2} = \{n':\text{Leonardo}, s':\text{Di Caprio}, n'':\text{Matthew}, s'':\text{David}, E_1:N_5, E_2:N_6, E_3:N_7\}\]

\[a_1^{m_1} \text{ and } a_1^{m_2} \text{ overlap on } \langle n, n'\rangle, \langle s, s'\rangle \text{ and } e_1.\]
They must therefore belong to the same Saturation Set.
Efficient overlap search

Saturation Set construction using overlap

- Pick an arbitrary assignment \( a \in A \)
- Initialize the set \( S \) with \( a \)
- While \( \exists b \in S, b' \in A - S \) s.t. \( b \) overlaps with \( b' \)
  - add \( b' \) to \( S \)

- Unoptimized search through all the set of assignments \( A \) can be very long.
- Idea: prune and guide this search by exploiting the relations among constraints!
Conflict areas and conflict masks

- **Conflict area** = a set of tgd variables that may occur in an overlap, together with the corresponding fd
- Conflict area + assignment $\rightarrow$ **Conflict mask** = the values for the area variables as given by the respective assignment.

**Conflict areas and masks - example**

$m_1$: Active-Actors($n, s, a$) $\rightarrow$ Actor($n, s, Y_1, Y_2$)

$e_1$: Actor($n, s, p, w$) $\land$ Actor($n, s, p', w'$) $\rightarrow$ ($p = p'$) $\land$ ($w = w'$)

$a_1^{m1} = \{ n : Leonardo, s : Di Caprio, a : 42, Y_1 : N_1, Y_2 : N_2 \}$

$ca_{11} = \langle n, s \rangle$, $e_1$ is a conflict area for $m_1$.

$m = \langle Leonardo, Di Caprio \rangle$ is the mask of $a_1^{m1}$ on $ca_{11}$.
Conflicts and the Conflict Graph

- A **conflict** between two s-t tgds = pair of conflict areas on the same fd.

**Conflicts - example**

$m_1$: Active_Actors($n, s, a$) $\rightarrow$ Actor($n, s, Y_1, Y_2$)

$m_2$: Actor_Coll($n', s', n'', s''$) $\rightarrow$ Actor($n', s'$, $E_1, E_2$) $\land$ Actor($n'', s''$, $E_3, E_2$)

$e_1$: Actor($n, s, p, w$) $\land$ Actor($n, s, p', w'$) $\rightarrow$ ($p = p'$) $\land$ ($w = w'$)

$cal_1 = \langle n, s \rangle$, $e_1$ is a conflict area for $m_1$

$cal_2 = \langle n', s' \rangle$, $e_1$ is a conflict area for $m_2$

$\langle cal_1, cal_2 \rangle$ is a conflict between $m_1$ and $m_2$ because both conflict areas are for $e_1$.

- **The Conflict Graph** accounts for all conflicts in a DE scenario:
  - vertices represent s-t tgds and are each adorned with their conflict areas
  - an edge states the existence of at least one conflict.

DSC/e Lecture

Angela Bonifati
Conflicts and the Conflict Graph

Conflict Graph - example

\[ m_1: \text{Active\_Actors}(n, s, a) \rightarrow \text{Actor}(n, s, Y_1, Y_2) \]

\[ m_2: \text{Actor\_Coll}(n', s', n'', s'') \rightarrow \text{Actor}(n', s', E_1, E_2) \land \text{Actor}(n'', s'', E_3, E_2) \]

\[ e_1: \text{Actor}(n, s, p, w) \land \text{Actor}(n, s, p', w') \rightarrow (p = p') \land (w = w') \]
The Conflict Graph and overlaps

Property: overlap = conflict + matching mask!

Example: find assignments overlapping with $a_{^{m1}}^1$

1. Navigate edge from $m_1$ to $m_2$ and look at $m_2$’s areas.
2. Find conflict $\langle{ca_1, ca_2}\rangle$.
3. Pull assignment $a_{^{m2}}^1 = \{n':Leonardo, s':Di Caprio, n'':Matthew, s'':David, E_1:N_5, E_2:N_6, E_3:N_7\}$ since it matches the mask on $ca_2$!

$\begin{align*}
a_{^{m1}}^1 &= \{n : Leonardo, s : Di Caprio, a : 60, Y_1 : N_1, Y_2 : N_2\} \\
\text{Mask of } m_1 \text{ on } ca_1 &= \langle{Leonardo, Di Caprio}\rangle \\
\end{align*}$
The Conflict Graph and parallelization

- The Conflict Graph speeds up the Saturation Set construction.
- But there’s more:
  - (Nice) property: a Saturation Set can never span across two distinct connected components of the Conflict Graph.
  - Therefore, Saturation Sets can be constructed, chased and materialized in parallel for all the connected components of the Conflict Graph.
  - This directly yields a parallelized (therefore faster!) version of the Interleaved Chase!
Summary of the Interleaved Chase

The Interleaved Chase aims for faster DE with target fds:
- Splits the chase into small units called Saturation Sets
- Uses overlaps to build Saturation Sets and the Conflict Graph to speed up overlap assessment
- Comes in a parallel flavour: *The Interleaved Chase with Parallelization*.

Theorem

Both the Interleaved Chase and the Interleaved Chase with Parallelization are sound and complete for providing DE solutions.
- furthermore, solutions are isomorphic to those provided by the Classical Data Exchange Chase
We implemented our algorithm(s) into a DE system called ChaseFUN, and tested using iBench\textsuperscript{3}-generated scenarios:

- $OF$ scenarios – use iBench default primitive Object Fusion
- $OF+$ scenarios – plug in the Vertical Partition primitive
- $OF++$ scenarios – enhanced $OF+$ with 3 atoms in the head.

For context and scale, we compared to one of the best DE engines currently available: the Llunatic system\textsuperscript{4}.

\textsuperscript{3}Patricia C. Arocena, Radu Ciucanu, Boris Glavic, Renée J. Miller. Gain Control over your Integration Evaluations. In PVLDB 8(12), 1960-1963, 2015

## Results

### Scenarios

<table>
<thead>
<tr>
<th>SCENARIO</th>
<th>s-t tgds</th>
<th>OF</th>
<th>OF+</th>
<th>OF++</th>
<th># source tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>15</td>
<td>5 egds</td>
<td>10 egds</td>
<td>15 egds</td>
<td>500K</td>
</tr>
<tr>
<td>C</td>
<td>45</td>
<td>15 egds</td>
<td>30 egds</td>
<td>45 egds</td>
<td>1.5M</td>
</tr>
<tr>
<td>F</td>
<td>90</td>
<td>30 egds</td>
<td>60 egds</td>
<td>90 egds</td>
<td>3M</td>
</tr>
</tbody>
</table>

### Chart

We scale smoothly, efficiently deal with large scenarios, and win the battle against fd overhead!

---

DSC/e Lecture

Angela Bonifati
The Interleaved Chase is:
- A sound and complete approach for Data Exchange with target functional dependencies
- Efficient and scalable w.r.t. the number of constraints and the size of the source.

To go further:
- A plethora of further optimizations
- Extension of our approach beyond target fds?
- Aim towards the smallest (core) Data Exchange solution?
Second Part: Interactive Mapping Specification

Mapping design is an hard task even for expert users:
- Need for domain knowledge and schema understanding
- Need to be acquainted with the syntax and semantics of tgd's
- Need to be able to write queries or customized code

How can we help non-expert users to express correct mappings?
- Need to find mappings as they explore their data
## Existing Paradigms tailored for expert users

### General-purpose approaches
- Rely on model management operators [Bernstein and Melnik, 2007].
  - Operators can be easily adapted to a variety of tools.
  - They are suited for data programmers.

### Graphical approaches
- Rely on a graphical formalism based on schema correspondences (i.e. arrows) [Popa et al., 2002].
- Exhibit limited expressivity and inherent ambiguity of arrows.

### Example-driven approaches
- Require as input a set of representative data examples [Alexe et al., 2011].
- Assume a designer familiar with schema mapping.
Roadmap of our proposal

- Allows a user to provide *arbitrary*\(^5\) exemplar tuples.

- Interacts with the user via simple boolean questions.

- Leads to discover the mapping that the user has in mind.

- Keeps the total number of user interactions as small as possible.

---

\(^{5}\text{i.e. a limited number of raw tuples provided by the non-expert user.}\)
Steps of our Interactive Mapping Specification process

Input: set of input pairs
\[(E^1_1, E^1_1), ..., (E^n_1, E^n_1)\]

Normalization

Atom refinement

Join refinement

Question
Answer: Yes or No

Question
Answer: Yes or No

Output: refined mapping \(\Sigma_{final}\)
Mapping generation from exemplar tuples

**Principle**
- Obtain the canonical mapping.
- Transform every source (target, resp.) exemplar tuple into an atom in the lhs (rhs, resp.) of the canonical mapping.

**Example**

<table>
<thead>
<tr>
<th>Company:</th>
<th>IdCompany</th>
<th>Name</th>
<th>Town</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>'C1'</td>
<td>'AA'</td>
<td>'Paris'</td>
</tr>
<tr>
<td></td>
<td>'C2'</td>
<td>'Ev'</td>
<td>'Lyon'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flight:</th>
<th>Departure</th>
<th>Arrival</th>
<th>IdCompany</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>'Lyon'</td>
<td>'Paris'</td>
<td>'C1'</td>
</tr>
<tr>
<td></td>
<td>'Paris'</td>
<td>'Lyon'</td>
<td>'C2'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Travel Agency:</th>
<th>IdAgency</th>
<th>Name</th>
<th>Town</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>'A1'</td>
<td>'TC'</td>
<td>'L.A.'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Carrier:</th>
<th>Id</th>
<th>Name</th>
<th>Town</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>'Id1'</td>
<td>'AA'</td>
<td>'Paris'</td>
</tr>
<tr>
<td></td>
<td>'Id2'</td>
<td>'Ev'</td>
<td>'Lyon'</td>
</tr>
<tr>
<td></td>
<td>'Id3'</td>
<td>'TC'</td>
<td>'L.A.'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Departure:</th>
<th>Town</th>
<th>IdCarrier</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>'Lyon'</td>
<td>'Id1'</td>
</tr>
<tr>
<td></td>
<td>'Paris'</td>
<td>'Id2'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arrival:</th>
<th>Town</th>
<th>IdCarrier</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>'Paris'</td>
<td>'Id1'</td>
</tr>
<tr>
<td></td>
<td>'Lyon'</td>
<td>'Id2'</td>
</tr>
</tbody>
</table>

\[\exists id1, id2, id3,\]
\[
\text{Company}(c1, aa, paris) \land \text{Company}(c2, ev, lyon) \\
\text{Flight}(lyon, paris, c1) \land \text{Flight}(paris, lyon, c2) \\
\text{TravelAgency}(a1, tc, la) \Rightarrow \\
\text{Carrier}(id1, aa, paris) \land \text{Departure}(lyon, id1) \\
\text{Arrival}(paris, id1) \land \text{Carrier}(id2, ev, lyon) \\
\text{Departure}(paris, id2) \land \text{Arrival}(lyon, id2) \\
\text{Carrier}(id3, tc, la) \]
Normalization to make mappings self-explanatory

**Principle**
- Split the canonical mapping into an equivalent set of smaller tgds [Gottlob et al., 2011].

**Example**
- **split-reduction:**
  
  \[
  \phi = \text{Company}(c1, \text{aa}, \text{paris}) \land \text{Company}(c2, \text{ev}, \text{lyon}) \\
  \land \text{Flight}(\text{lyon, paris}, c1) \land \text{Flight}(\text{paris, lyon}, c2) \land \text{TravelAgency}(a1, \text{tc}, la)
  \]
  
  \[
  \{\phi \Rightarrow \exists \text{id1}, \text{Carrier}(\text{id1}, \text{aa}, \text{paris}) \land \text{Departure}(\text{lyon, id1}) \land \text{Arrival}(\text{paris, id1})
  \}
  \]
  
  \[
  \phi \Rightarrow \exists \text{id2}, \text{Carrier}(\text{id2}, \text{ev}, \text{lyon}) \land \text{Departure}(\text{paris, id2}) \land \text{Arrival}(\text{lyon, id2})
  \]
  
  \[
  \phi \Rightarrow \exists \text{id3}, \text{Carrier}(\text{id3}, \text{tc}, \text{la})
  \]

- **σ-redundancy suppression:** either the first or the second tgd is removed.
Atom refinement step

**Principle**
Remove superfluous atoms in the left-hand side of tgd's

**Example**

\[
\text{Company}(c1, \text{aa}, \text{paris}) \land \text{Company}(c2, \text{ev}, \text{lyon}) \\
\land \text{Flight}(\text{lyon}, \text{paris}, c1) \land \text{Flight}(\text{paris}, \text{lyon}, c2) \land \text{TravelAgency}(a1, \text{tc}, \text{la}) \\
\Rightarrow \exists id1, \text{Carrier}(id1, \text{aa}, \text{paris}) \land \text{Departure}(\text{lyon}, id1) \land \text{Arrival}(\text{paris}, id1)
\]
Atom refinement step

**Principle**
Remove superfluous atoms in the left-hand side of tgd's

**Example**

\[
\begin{align*}
\text{Company}(c1, \text{aa, paris}) \land \text{Company}(c2, \text{ev, lyon}) \\
\land \text{Flight}(\text{lyon, paris, c1}) \land \text{Flight}(\text{paris, lyon, c2}) \land \text{TravelAgency}(a1, \text{tc, la})
\end{align*}
\]  
\[
\Rightarrow \exists \text{id1}, \text{Carrier}(\text{id1, aa, paris}) \land \text{Departure}(\text{lyon, id1}) \land \text{Arrival}(\text{paris, id1})
\]
Questioning the user about the atoms to be deleted

Example

```
{C_1; C_2; F_1; F_2; TA}
{C_1; C_2; F_1; TA}   {C_1; C_2; F_2; TA}
{C_1; C_2; F_1; F_2}   {C_1; F_1; F_2; TA}
{C_1; C_2; TA}   {C_1; C_2; F_1}   {C_1; F_1; TA}   {C_1; F_1; F_2}   {C_1; F_2; TA}
{C_1; C_2}   {C_1; F_1}   {C_1; F_2}
```

Question:

"Are the tuples \{Company('C1', 'AA', 'Paris'); Company('C2', 'Ev', 'Lyon')\} enough to produce: \{Carrier('ID', 'AA', 'Paris'); Departure('Lyon', 'ID'); Arrival('Paris', 'ID')\}?"
Questioning the user about the atoms to be deleted

Example (cont’d…)

```
{C₁; C₂; F₁; F₂; TA}

{C₁; C₂; F₁; TA}  {C₁; C₂; F₁; F₂}  {C₁; F₁; F₂; TA}

{C₁; C₂; TA}  {C₁; C₂; F₁}  {C₁; F₁; TA}  {C₁; C₂; F₂}  {C₁; F₁; F₂}  {C₁; F₂; TA}

{C₁; C₂}  {C₁; F₁}  {C₁; F₂}
```

Question:

“Are the tuples \{\text{Company}('C₁', 'AA', 'Paris'); \text{Flight}('Lyon', 'Paris', 'C₁')\} enough to produce: \{\text{Carrier}('ID', 'AA', 'Paris'); \text{Departure}('Lyon', 'ID'); \text{Arrival}('Paris', 'ID')\}?"
Questioning the user about the atoms to be deleted

Example (cont’d…)

```
{C1; C2; F1; F2; TA}
{C1; C2; F1; TA} {C1; C2; F1; F2} {C1; F1; F2; TA}
{C1; C2; TA} {C1; C2; F1} {C1; F1; TA} {C1; C2; F2} {C1; F1; F2} {C1; F2; TA}
{C1; C2} {C1; F1} {C1; F2}
```

Question:

"Are the tuples \{Company('C1', 'AA', 'Paris'); Flight('Lyon', 'Paris', 'C1')\} enough to produce:
\{Carrier('ID', 'AA', 'Paris'); Departure('Lyon', 'ID'); Arrival('Paris', 'ID')\}?”
Questioning the user about the atoms to be deleted

Example (cont’d...)

```
{C; C2; F1; F2; TA}
{C; C2; F1; TA}  {C; C2; F1; F2}  {C; F1; F2; TA}
{C; C2; TA}  {C; C2; F1}  {C; C2; F2}  {C; F1; F2; TA}
{C; C2}  {C1; F1}  {C1; F2}
```

**Question:**

"Are the tuples \{Company('C1', 'AA', 'Paris'); Flight('Paris', 'Lyon', 'C2')\} enough to produce:
\{Carrier('ID', 'AA', 'Paris'); Departure('Lyon', 'ID'); Arrival('Paris', 'ID')\}?"
Questioning the user about the atoms to be deleted

Example (cont’d…)

{C_1; C_2; F_1; F_2; TA}

{C_1; C_2; F_1; TA}  {C_1; C_2; F_1; F_2}  {C_1; F_1; F_2; TA}

{C_1; C_2; F_1}  {C_1; F_1; TA}  {C_1; C_2; F_2}  {C_1; F_1; F_2}  {C_1; F_2; TA}

{C_1; C_2}  {C_1; F_1}  {C_1; F_2}

Question:

“Are the tuples \(\{\text{Company}(\text{'C1'}, \text{'AA'}, \text{'Paris'}); \text{Company}(\text{'C2'}, \text{'EF'}, \text{'Lyon'})\};\text{TravelAgency}(\text{'A1'}, \text{'TC'}, \text{'L.A.'})\}\) enough to produce:
\(\{\text{Carrier}(\text{'ID'}, \text{'AA'}, \text{'Paris'}); \text{Departure}(\text{'Lyon'}, \text{'ID'})}; \text{Arrival}(\text{'Paris'}, \text{'ID'})\}\)’?
Questioning the user about the atoms to be deleted

Example (cont’d...)

```
{C1; C2; F1; F2; TA}
{C1; C2; F1; TA}  {C1; C2; F1; F2}  {C1; F1; F2; TA}
{C1; C2; F1}  {C1; C1; F1; TA}  {C1; C2; F2}  {C1; F1; F2}  {C1; F2; TA}
{C1; C2}  {C1; F1}  {C1; F2}
```

**Question:**

“Are the tuples \{Company('C1', 'AA', 'Paris'); Company('C2', 'EF', 'Lyon'); Flight('Paris', 'Lyon', 'C2')\} enough to produce:
\{Carrier('ID', 'AA', 'Paris'); Departure('Lyon', 'ID'); Arrival('Paris', 'ID')\}?"
Questioning the user about the atoms to be deleted

Example (cont’d…)

Question:
Questioning the user about the atoms to be deleted

Example (cont’d…)

Generated mapping:
\[ \Sigma = \{ \text{Company}(c1, aa, paris) \land \text{Flight}(lyon, paris, c1) \} \]
\[ \Rightarrow \exists id1, \text{Carrier}(id1, aa, paris) \land \text{Departure}(lyon, id1) \land \text{Arrival}(paris, id1) \} \]
Join refinement step

Principle
Identify redundant joins entailed by multiple occurrences of a given variable.

Example
We will focus on variable 'paris' in the obtained tgd:

\[ Company(c1, aa, paris) \land Flight(lyon, paris, c1) \]
\[ \Rightarrow \exists id1, Carrier(id1, aa, paris) \land Departure(lyon, id1) \land Arrival(paris, id1) \]
Example (cont’d…)

- Renaming of occurrences of a variable:

  \begin{align*}
  &\text{Company}(c1, aa, \text{paris}_1) \land \text{Flight}(\text{lyon}, \text{paris}_2, c1) \\
  &\Rightarrow \exists id1, \text{Carrier}(id1, aa, \text{paris}_3) \land \text{Departure}(\text{lyon}, id1) \land \text{Arrival}(\text{paris}_4, id1)
  \end{align*}
Join refinement step

Example (cont’d…)

- Renaming of occurrences of a variable:

  \[ \text{Company}(c1, \text{aa}, \text{paris}_1) \land \text{Flight}(\text{lyon}, \text{paris}_2, c1) \]
  \[ \Rightarrow \exists \text{id}_1, \text{Carrier}(\text{id}_1, \text{aa}, \text{paris}_3) \land \text{Departure}(\text{lyon}, \text{id}_1) \land \text{Arrival}(\text{paris}_4, \text{id}_1) \]

produced:

\[
\begin{align*}
\text{Company}(c1, \text{aa}, \text{paris}_1) & \land \text{Flight}(\text{lyon}, \text{paris}_2, c1) \\
\Rightarrow & \exists \text{id}_1, \text{Carrier}(\text{id}_1, \text{aa}, \text{paris}_3) \land \text{Departure}(\text{lyon}, \text{id}_1) \land \text{Arrival}(\text{paris}_4, \text{id}_1)
\end{align*}
\]
Join refinement step

Example (cont’d…)

- Renaming of occurrences of a variable:

\[ Company(c1, aa, \text{paris}_1) \land Flight(\text{lyon}, \text{paris}_2, c1) \]
\[ \Rightarrow \exists \text{id1}, Carrier(\text{id1}, aa, \text{paris}_3) \land Departure(\text{lyon}, \text{id1}) \land Arrival(\text{paris}_4, \text{id1}) \]

- Produced tgd:

\[ Company(c1, aa, \text{paris}_1) \land Flight(\text{lyon}, \text{paris}_2, c1) \]
\[ \Rightarrow \exists \text{id1}, Carrier(\text{id1}, aa, \text{paris}_1) \land Departure(\text{lyon}, \text{id1}) \land Arrival(\text{paris}_2, \text{id1}) \]
Final mapping

\[ \Sigma_{\text{final}} = \{ Company(c1, aa, paris_1) \land Flight(\text{lyon}, paris_2, c1) \land \exists id1, \ Carrier(id1, aa, paris_1) \land Departure(\text{lyon}, id1) \land Arrival(paris_2, id1); \ \ TravelAgency(a1, tc, la) \Rightarrow \exists id3, \ Carrier(id3, tc, la) \} \]

Theorem

If \( \Sigma_{\text{final}} \) is a refinement of \( \Sigma \) then \( \Sigma_{\text{final}} \models \Sigma \)

Property

Moreover, \( \Sigma_{\text{final}} \) is guaranteed to be in normal form.
Objectives

Evaluate the effectiveness of the interactive process by measuring the number of questions needed to achieve $\Sigma_{final}$

- **7 real mapping scenarios** drawn from iBench [Arocena et al., 2015].
  - exemplar tuples extracted from mappings
  - degradation procedure (addition of atoms and unification of variables)
  - user simulated by leveraging the original mapping

- **Four strategies for exploring semi-lattices**: top-down and bottom-up, combined with breadth-first and depth-first exploration.
Degradation procedure

- Goal: simulate two kinds of end-user errors:
  - Addition of superfluous atoms.
    
    Example:
    
    $S(a, b) \Rightarrow T(a)$ can be degraded in $S(a, b) \land S(a', b') \Rightarrow T(a)$
  
  - Unification of variables as for ambiguous joins.
    
    Example:
    
    $S(a, b, c) \Rightarrow T(a, b, c)$ can be degraded in $S(a, a, c) \Rightarrow T(a, a, c)$

- From 0 to 10 degradations have been applied to the above scenarios.
Experiments on atom refinement

All exploration strategies keep the number of questions (per tgd) low along atom refinement.
Experiments on join refinement

The average number of questions per tgd is kept low along join refinement (but slightly higher than that of the former refinement).
Among all strategies, the top-down breadth-first one exhibits the maximum gain in both refinement steps.
Benefit of (non-universal) exemplar tuples

Arbitrary exemplar tuples allow users to produce smaller target instances than their universal counterparts.
Concluding Remarks

- We have built a mapping specification framework that lets non-expert users tame sophisticated mappings.
- The system requires as input simple exemplar tuples and pursues boolean user interactions.
- The experimental assessment shows the practicality and the quality of mapping specification.

Future directions

- Coverage of existentially quantified variables without a dramatic impact on the number of asked questions.
- Modeling the behavior of real end-users and simulations in a crowdsourcing workbench.
Thanks for your attention.


