Simplification Logic as a tool for manipulation of implications

Angel Mora Bonilla
Department of Applied Mathematics
University of Malaga, Spain

Workshop Information, Uncertainty, and Imprecision
Olomouc, June 2012

Outline

1. **Background**
2. Algebraic framework
3. Logic
   - $\mathbf{SL}_{FD}$ logic
   - Redundancy: Classical logics versus $\mathbf{SL}_{FD}$ logic
   - Closure
   - Minimal Keys
4. Conclusions

Relationships between data

Relational Model: It is easy to represent data

World Gazetteer

Spain: largest cities and towns and statistics of their population

download data for Google Earth

see also the list of metropolitan areas

view more cities and places

no.	name	census 1991	census 2001	estimate 2010	calculation 2012	annual growth
1	Madrid	3 010 492	2 938 723	3 273 049	3 332 646	0.91
2	Barcelona	1 643 542	1 503 884	1 619 337	1 624 598	0.16
3	Valencia	752 909	738 441	809 267	831 261	1.35
4	Sevilla	683 028	684 633	704 198	703 029	-0.08
5	Zaragoza	594 394	614 905	675 121	685 963	0.80
6	Málaga	522 108	524 414	568 507	571 731	0.28
7	Murcia	328 100	370 745	441 345	453 985	1.42
8	Palma	296 754	333 801	404 681	419 285	1.79
9	Las Palmas	354 877	354 863	383 308	385 973	0.35
10	Bilbao	369 839	349 972	353 187	351 864	-0.19

Relational model

But, be careful

	Subject	Identity Card	Surname	Name	Course
t1	Algebra	22222222A	SMITH	RALPH	3
t2	Algebra	33333333A	ROSE	PETER	1
t3	Calculus	22222222A	SMITH	RALPH	3
t4	Calculus	44444444B	BRANDON	ANNE	4
t5	Calculus	11111111C	BUGLE	LOUISE	2
t6	Numerical Methods	33333333A	ROSE	PAUL	1

RELATIONAL DATA BASE

Studying the relations between the data, we avoid the anomalies, inconsistencies, redundancies, ...

Relational model

But, be careful

	Subject	Identity Card	Surname	Name	Course
t1	Algebra	22222222A	SMITH	RALPH	3
t2	Algebra	33333333A	ROSE	PETER	1
t3	Calculus	22222222A	SMITH	RALPH	3
t4	Calculus	44444444B	BRANDON	ANNE	4
t5	Calculus	11111111C	BUGLE	LOUISE	2
t6	Numerical Methods	33333333A	ROSE	PAUL	1

RELATIONAL DATA BASE

Studying the relations between the data, we avoid the anomalies, inconsistencies, redundancies, ...

Functions

	Subject	Identity Card	Surname	Name	Closed Call
t1	Algebra	22222222A	SMITH	RALPH	4
t2	Algebra	33333333A	ROSE	PETER	1
t3	Calculus	22222222A	SMITH	RALPH	4
t4	Calculus	44444444B	BRANDON	ANNE	5
t5	Calculus	11111111C	BUGLE	LOUISE	3
t6	Numerical Methods	33333333A	ROSE	PETER	1

RELATIONAL DATA BASE

Valuable Functions: $f(\textbf{Closed Call}) = \textbf{Registration Fee}$

Functions for extension: *(using the table)*

Functional Dependencies

Defines FDs (1972) and normalization.

Functional Dependencies

	Subject	Identity Card	Surname	Name	Closed Call
t1	Algebra	22222222A	SMITH	RALPH	4
t2	Algebra	33333333A	ROSE	PETER	1
t3	Calculus	22222222A	SMITH	RALPH	4
t4	Calculus	44444444B	BRANDON	ANNE	5
t5	Calculus	11111111C	BUGLE	LOUISE	3
t6	Numerical Methods	33333333A	ROSE	PETER	1

RELATIONAL DATA BASE

Functional Dependencies (FDs)

$t_1/_{idCard} = t_3/_{idCard}$ implies that $t_1/_{surname} = t_3/_{surname}$ y $t_1/_{name} = t_3/_{name}$

$t_2/_{idCard} = t_6/_{idCard}$ implies that $t_2/_{surname} = t_6/_{surname}$ y $t_2/_{name} = t_6/_{name}$

idCard $\mapsto$ Surname, Nane

Functional Dependencies

	Subject	Identity Card	Surname	Name	Closed Call
t1	Algebra	22222222A	SMITH	RALPH	4
t2	Algebra	33333333A	ROSE	PETER	1
t3	Calculus	22222222A	SMITH	RALPH	4
t4	Calculus	44444444B	BRANDON	ANNE	5
t5	Calculus	11111111C	BUGLE	LOUISE	3
t6	Numerical Methods	33333333A	ROSE	PETER	1

RELATIONAL DATA BASE

Functional Dependencies (FDs)

$t_1/_{idCard} = t_3/_{idCard}$ implies that $t_1/_{surname} = t_3/_{surname}$ y $t_1/_{name} = t_3/_{name}$
$t_2/_{idCard} = t_6/_{idCard}$ implies that $t_2/_{surname} = t_6/_{surname}$ y $t_2/_{name} = t_6/_{name}$

idCard$\mapsto$Surname, Nane

Functional Dependencies

Definition

Let R be a relation over $\mathcal{A}$. Any affirmation of the type $X{\mapsto}Y$, where $X, Y \subseteq \mathcal{A}$, is called **functional dependency** (henceforth FD) over R. We say that R **satisfies** $X{\mapsto}Y$ if, for all $t_1, t_2 \in R$ we have that: $t_{1/X} = t_{2/X}$ implies that $t_{1/Y} = t_{2/Y}$.

Functional Dependencies

idCard$\mapsto$Surname, Name
Surname, Name$\mapsto$Antiqueness, Degree
Antiqueness, Degree$\mapsto$Salary

From FDs the following information can be deduced:

idCard$\mapsto$Salary

$\Gamma \models X \mapsto Y$

- All models of Γ satisfy $X \mapsto Y$?
- To analyze this question in all the relations is intractable from the point of view of the semantics.
- A syntactic method to deduce information is required.

Functional Dependencies

idCard$\mapsto$Surname, Name
Surname, Name$\mapsto$Antiqueness, Degree
Antiqueness, Degree$\mapsto$Salary

From FDs the following information can be deduced:

idCard$\mapsto$Salary

$\Gamma \models X \mapsto Y$

- All models of Γ satisfy $X \mapsto Y$?
- To analyze this question in all the relations is intractable from the point of view of the semantics.
- A syntactic method to deduce information is required.

Axiomatic System of Armstrong

Definition

Let Ω be a set of atoms and let $\mapsto$ be a binary connective, the language of the functional dependencies logic is defined as:

$$\mathcal{L} = \{X \mapsto Y \mid X, Y \in 2^{\Omega} \text{ y } X \neq \varnothing\}$$

Definition

L is the logic given by $(\mathcal{L}, \mathcal{S})$ where $\mathcal{S}$, has the unique axiom

$\lfloor Axiom \rfloor : \quad \vdash_{\mathcal{S}_{Par}} X \mapsto Y, \quad \text{if } Y \subseteq X$

and the following inference rules:

$\lfloor Trans \rfloor : \quad X \mapsto Y, \ Y \mapsto Z \vdash X \mapsto Z$ .... **Transitivity**

$\lfloor Aug \rfloor : \quad X \mapsto Y, \vdash X \mapsto XY$ .... **Augmentation**

What about FDs now?

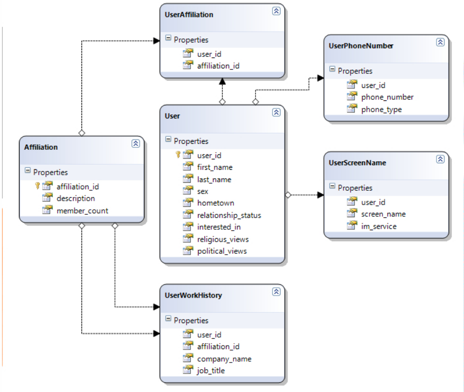


Figure : A normalized database schema for a generic social networking site (`http://www.codinghorror.com`)

What about FDs now?

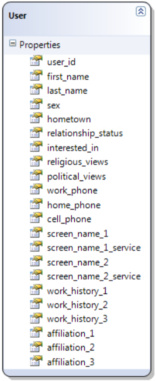


Figure : Non-normalized

This database is faster in the queries

Always? No, in huge database the queries are slower.

And ... you have lost the semantics of the relationship between the data !!!

What about FDs now?

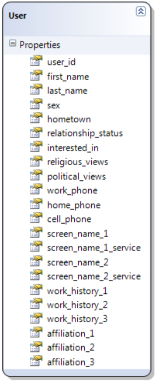


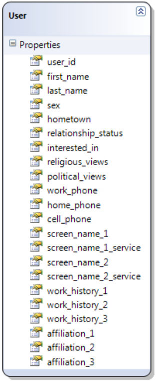
Figure : Non-normalized

This database is faster in the queries

Always? No, in huge database the queries are slower.

And ... you have lost the semantics of the relationship between the data !!!

What about FDs now?



This database is faster in the queries

Always? No, in huge database the queries are slower.

And ... you have lost the semantics of the relationship between the data !!!

Figure : Non-normalized

Towards a new framework

FDs have been left aside!!!

- Armstrong's Axioms are not appropriate to reason.
- Unfortunately, today, normalization is being forgotten in the database design.
- Companies must repair the bad-design (over-cost) when database degenerates .
- Commercial tools do not incorporate FDs because they do not know how manage it.
- In some tools it is possible to specify FDs (Oracle) but none incorporates algorithms for FDs.

It is really necessary an adequate formalization!!!

Our tools: Lattice theory, Logic

Towards a new framework

FDs have been left aside!!!

- Armstrong's Axioms are not appropriate to reason.
- Unfortunately, today, normalization is being forgotten in the database design.
- Companies must repair the bad-design (over-cost) when database degenerates .
- Commercial tools do not incorporate FDs because they do not know how manage it.
- In some tools it is possible to specify FDs (Oracle) but none incorporates algorithms for FDs.

It is really necessary an adequate formalization!!!

Our tools: Lattice theory, Logic

Outline

From Algebra to Logic

***Non-deterministic ideal operators: An adequate tool for formalization in Data Bases*, P. Cordero, et.al. - Discrete Applied Mathematics 156 (6), 2008**

- Characterize the concept of *Armstrong's relation (full-family, f-family)*.
- Formalize database redundancy.
- Propose the algebraic definition of the *normal forms* in database.
- Achieve trivial results about hard problems in functional dependencies.
- Extend the concept of scheme and the study of keys and antikeys.

Non-deterministic operator

Definition

Let A be a non-empty set and $n \in \mathbb{N}$ with $n \geq 1$. If $F : A^n \to 2^A$ is a total mapping, we say that F is a **non-deterministic operator with arity** n in A (henceforth, ndo)

We denote the set ndos with arity n in A by $\mathcal{N}do_n(A)$ and, if F is a ndo, we denote its arity by $\mathtt{ar}(F)$. As usual,

$$F(a_1, \ldots, a_{i-1}, X, a_{i+1}, \ldots, a_n) = \bigcup_{x \in X} F(a_1, \ldots, a_{i-1}, x, a_{i+1}, \ldots, a_n).$$

In **Hyperalgebra Theory** the ndo is known as hyperoperation.

Non-deterministic ideal operator

In database theory, the study of FDs is based on the concept of f-family:

$$DF_{\mathcal{R}} = \{X{\mapsto}Y \mid \Gamma \models X{\mapsto}Y\}$$

Definition

Let $(A, \leq)$ be a poset and $F : A \longrightarrow 2^A$ a ndo in A. We say that F is a **non-deterministic ideal operator** (**nd.ideal-o**) if it is:

- **reflexive**
- **transitive**
- $F(a)$ **an ideal of** $(A, \leq)$ for all $a \in A$

Theorem

F be a unary ndo in a poset $(A, \leq)$. ,

F is a **f-family** in A if and only if is a **nd.ideal-o** in $(2^A, \subseteq)$.

Non-deterministic ideal operator

In database theory, the study of FDs is based on the concept of f-family:

$$DF_{\mathcal{R}} = \{X \mapsto Y \mid \Gamma \models X \mapsto Y\}$$

Definition

Let $(A, \leq)$ be a poset and $F : A \longrightarrow 2^A$ a ndo in A. We say that F is a **non-deterministic ideal operator** (**nd.ideal-o**) if it is:

- **reflexive**
- **transitive**
- $F(a)$ **an ideal of** $(A, \leq)$ for all $a \in A$

Theorem

F be a unary ndo in a poset $(A, \leq)$. ,

F is a **f-family** in A if and only if is a **nd.ideal-o** in $(2^A, \subseteq)$.

Non-deterministic ideal operator

In database theory, the study of FDs is based on the concept of f-family:

$$DF_{\mathcal{R}} = \{X \mapsto Y \mid \Gamma \models X \mapsto Y\}$$

Definition

Let $(A, \leq)$ be a poset and $F : A \longrightarrow 2^A$ a ndo in A. We say that F is a **non-deterministic ideal operator** (**nd.ideal-o**) if it is:

- **reflexive**
- **transitive**
- $F(a)$ **an ideal of** $(A, \leq)$ for all $a \in A$

Theorem

F be a unary ndo in a poset $(A, \leq)$. ,

F is a f**-family** in A if and only if is a **nd.ideal-o** in $(2^A, \subseteq)$.

Generator

Definition

Let $(A, \leq)$ be a lattice and F a nd.ideal-o in A. We say that $G \in \mathcal{N}do_n(A)$ is a **generator** of F if $\widehat{G} = F$.

Definition

Let $(A, \leq)$ be a lattice and $F, G \in \mathcal{N}do_n(A)$. We say that F and G are **equivalent** if $\widehat{F} = \widehat{G}$.

We formalize the definition of the idea of “to have less information than”:

$$G \prec F$$

Definition

Let $(A, \leq)$ bea lattice and $F, G \in \mathcal{N}do_n(A)$. We say that F is **redundant** if there exists G equivalent to F such that $G \prec F$.

Generator

Definition

Let $(A, \leq)$ be a lattice and F a nd.ideal-o in A. We say that $G \in \mathcal{N}do_n(A)$ is a **generator** of F if $\widehat{G} = F$.

Definition

Let $(A, \leq)$ be a lattice and $F, G \in \mathcal{N}do_n(A)$. We say that F and G are **equivalent** if $\widehat{F} = \widehat{G}$.

We formalize the definition of the idea of “to have less information than”:

$$G \prec F$$

Definition

Let $(A, \leq)$ bea lattice and $F, G \in \mathcal{N}do_n(A)$. We say that F is **redundant** if there exists G equivalent to F such that $G \prec F$.

Generator

Definition

Let $(A, \leq)$ be a lattice and F a nd.ideal-o in A. We say that $G \in \mathcal{N}do_n(A)$ is a **generator** of F if $\widehat{G} = F$.

Definition

Let $(A, \leq)$ be a lattice and $F, G \in \mathcal{N}do_n(A)$. We say that F and G are **equivalent** if $\widehat{F} = \widehat{G}$.

We formalize the definition of the idea of “to have less information than”:

$$G \prec F$$

Definition

Let $(A, \leq)$ bea lattice and $F, G \in \mathcal{N}do_n(A)$. We say that F is **redundant** if there exists G equivalent to F such that $G \prec F$.

Example 1

$F : 2^A \to 2^{2^A}$,

$$F(\{a\}) = \{\{a, c\}\}, \quad F(X) = \varnothing \text{ otherwise}$$

is redundant because $G : 2^A \to 2^{2^A}$

$$G(\{a\}) = \{\{c\}\}, \quad G(X) = \varnothing \text{ otherwise}$$

satisfies that $G \prec F$ and, as $F(\{a\}) \subseteq \widehat{G}(\{a\}) = \{\varnothing, \{a\}, \{c\}, \{a, c\}\}$, we have that $\widehat{G} = \widehat{F}$.

Example 2

$F : 2^A \to 2^{2^U}$, $F(\{a\}) = \{\{c\}\}$, $F(\{a,c\}) = \{\{b\}\}$, $F(X) = \varnothing$ otherwise is redundant because $G : 2^A \to 2^{2^A}$

$$G(\{a\}) = \{\{c\},\{b\}\}, \quad G(X) = \varnothing \text{ otherwise}$$

satisfies that $G \prec F$ and $\widehat{G} = \widehat{F}$.

Redundancy

Proposition

- $G(a)$ is given by $G(a) = F(a) \smallsetminus \{b\}$ and $G(x) = F(x)$ otherwise when $b \in \widehat{G(a)}$.

Definition

Let $(A, \leq)$ be a lattice and $F \in \mathcal{O}nd_1(A)$. We say that $G \in \mathcal{O}nd_1(A)$ is a **minimal generator** of F if:

- G is equivalent to F,
- $G \prec F$ and
- G is not redundant.

Redundancy

Proposition

- $G(a)$ is given by $G(a) = F(a) \smallsetminus \{b\}$ and $G(x) = F(x)$ otherwise when $b \in \widehat{G(a)}$.

Definition

Let $(A, \leq)$ be a lattice and $F \in \mathcal{O}nd_1(A)$. We say that $G \in \mathcal{O}nd_1(A)$ is a **minimal generator** of F if:

- G is equivalent to F,
- $G \prec F$ and
- G is not redundant.

Outline

From Algebra to Logic

Stages:

- Firstly, we proposed a new Simplification Rule adequate to remove redundancy in an automatic way.
- Simplification Rule turned the *heart* of a novel logic : **SL**$_{FD}$ logic - Simplification logic for FDs.
- **SL**$_{FD}$ logic turned out to be the *engine* of automated methods: redundancy removal, closure algorithm, minimal keys, etc.

Simplification Logic

Definition: $\mathbf{SL}_{\mathrm{FD}}$ logic

$\lfloor Axiom \rfloor : \quad \vdash_{\mathcal{S}_{\mathrm{FD}}} X \mapsto Y, \quad$ si $Y \subseteq X$

- $\lfloor Frag \rfloor \; X \mapsto Y \vdash_{\mathcal{S}_{\mathrm{FD}}} X \mapsto Y'$ if $Y' \subseteq Y$ .... **Fragmentation**
- $\lfloor Comp \rfloor \; X \mapsto Y, \; U \mapsto V \vdash_{\mathcal{S}_{\mathrm{FD}}} XU \mapsto YV$ .... **Composition**
- $\lfloor Simp \rfloor \; X \mapsto Y, \; U \mapsto V \vdash_{\mathcal{S}_{\mathrm{FD}}} (U\text{-}Y) \mapsto (V\text{-}Y)$ .... **Simplification**
  if $X \subseteq U$, $X \cap Y = \varnothing$

and the following derived rule:
$\lfloor rSimp \rfloor \qquad X \mapsto Y, U \mapsto V \vdash_{\mathcal{S}_{FDS}} U \mapsto (V\text{-}Y)$ .... **r-Simplification**
if $X \subseteq UV, X \cap Y = \varnothing$

Outline

Deduction with classical logics for FDs

Removing redundancy - FD logic of R. Fagin

$\{ab{\mapsto}c,\ c{\mapsto}a,\ bc{\mapsto}d,\ acd{\mapsto}b,\ d{\mapsto}eg,\ be{\mapsto}c,\ cg{\mapsto}bd,\ ce{\mapsto}ag\}$

- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, \underbrace{d{\mapsto}eg}, be{\mapsto}c, \underbrace{cg{\mapsto}bd}, \underbrace{ce{\mapsto}ag}\}$
- Fragmentation 3 times,
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, cg{\mapsto}d, ce{\mapsto}a, ce{\mapsto}g\}$

Deduction with classical logics for FDs

Removing redundancy - FD logic of R. Fagin

$\{ab \mapsto c,\ c \mapsto a,\ bc \mapsto d,\ acd \mapsto b,\ d \mapsto eg,\ be \mapsto c,\ cg \mapsto bd,\ ce \mapsto ag\}$

- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, \underbrace{d{\mapsto}eg}, be{\mapsto}c, \underbrace{cg{\mapsto}bd}, \underbrace{ce{\mapsto}ag}\}$
- Fragmentation 3 times,
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, cg{\mapsto}d, ce{\mapsto}a, ce{\mapsto}g\}$

Deduction with classical logics for FDs

Removing redundancy - FD logic of R. Fagin

{ab↦c, c↦a, bc↦d, acd↦b, d↦eg, be↦c, cg↦bd, ce ↦ag}
Rules: Fragmentation 3 times

- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, \underline{d{\mapsto}eg}, be{\mapsto}c, \underline{cg{\mapsto}bd}, \underline{ce{\mapsto}ag}\}$
- Fragmentation 3 times
- $\{ab{\mapsto}c, \underline{c{\mapsto}a}, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, cg{\mapsto}d, \cancel{ce{\mapsto}a}, ce{\mapsto}g\}$
- Using Augmentation Rule - ce ↦a is redundant
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, cg{\mapsto}d, ce{\mapsto}g\}$

Deduction with classical logics for FDs

Removing redundancy - FD logic of R. Fagin

{ab↦c, c↦a, bc↦d, acd↦b, d↦eg, be↦c, cg↦bd, ce ↦ag}
Rules: Fragmentation 3 times

- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, \underline{d{\mapsto}eg}, be{\mapsto}c, \underline{cg{\mapsto}bd}, \underline{ce{\mapsto}ag}\}$
- Fragmentation 3 times
- $\{ab{\mapsto}c, \underline{c{\mapsto}a}, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, cg{\mapsto}d, \cancel{ce{\mapsto}a}, ce{\mapsto}g\}$
- Using Augmentation Rule - ce ↦a is redundant
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, cg{\mapsto}d, ce{\mapsto}g\}$

Deduction with classical logics for FDs

Removing redundancy - FD logic of R. Fagin

{ab↦c, c↦a, bc↦d, acd↦b, d↦eg, be↦c, cg↦bd, ce ↦ag}
Rules: Fragmentation 3 times, Augmentation

- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, cg{\mapsto}d, ce{\mapsto}g\}$
- Using Reflexivity Rule - cg ↦c is added as derived FD
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, cg{\mapsto}d, ce{\mapsto}g\} \bigcup \{\underbrace{cg{\mapsto}c}\}$

Deduction with classical logics for FDs

Removing redundancy - FD logic of R. Fagin

{ab$\mapsto$c, c$\mapsto$a, bc$\mapsto$d, acd$\mapsto$b, d$\mapsto$eg, be$\mapsto$c, cg$\mapsto$bd, ce $\mapsto$ag}
Rules: Fragmentation 3 times, Augmentation

- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, cg{\mapsto}d, ce{\mapsto}g\}$
- Using Reflexivity Rule - cg $\mapsto$c is added as derived FD
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, cg{\mapsto}d, ce{\mapsto}g\} \bigcup \{\underbrace{cg{\mapsto}c}\}$

Deduction with classical logics for FDs

Removing redundancy - FD logic of R. Fagin

{ab↦c, c↦a, bc↦d, acd↦b, d↦eg, be↦c, cg↦bd, ce ↦ag}
Rules: Fragmentation 3 times, Augmentation, Reflexivity

- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, cg{\mapsto}d, ce{\mapsto}g\}$
- Using Reflexivity Rule - cg ↦c is added as derived FD
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, cg{\mapsto}d, ce{\mapsto}g\} \bigcup \{cg{\mapsto}c\}$

Deduction with classical logics for FDs

Removing redundancy - FD logic of R. Fagin

{ab↦c, c↦a, bc↦d, acd↦b, d↦eg, be↦c, cg↦bd, ce ↦ag}
Rules: Fragmentation 3 times, Augmentation, Reflexivity

- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, cg{\mapsto}d, ce{\mapsto}g\} \bigcup \{cg{\mapsto}c\}$
- Using Union Rule - $cg{\mapsto}bc$ is added as derived FD
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, \underbrace{cg{\mapsto}b}, cg{\mapsto}d, ce{\mapsto}g\} \bigcup \{\underbrace{cg{\mapsto}c}, cg{\mapsto}bc\}$

Deduction with classical logics for FDs

Removing redundancy - FD logic of R. Fagin

{ab↦c, c↦a, bc↦d, acd↦b, d↦eg, be↦c, cg↦bd, ce ↦ag}
Rules: Fragmentation 3 times, Augmentation, Reflexivity

- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, cg{\mapsto}d, ce{\mapsto}g\} \bigcup \{cg{\mapsto}c\}$
- Using Union Rule - $cg{\mapsto}bc$ is added as derived FD
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, cg{\mapsto}d, ce{\mapsto}g\} \bigcup \{cg{\mapsto}c, cg{\mapsto}bc\}$

Deduction with classical logics for FDs

Removing redundancy - FD logic of R. Fagin

{ab↦c, c↦a, bc↦d, acd↦b, d↦eg, be↦c, cg↦bd, ce ↦ag}
Rules: Fragmentation 3 times, Augmentation, Reflexivity, Union

- $\{ab{\mapsto}c, c{\mapsto}a, \underbrace{bc{\mapsto}d}, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, \cancel{cg{\mapsto}d}, ce{\mapsto}g\} \bigcup \{cg{\mapsto}c, \underbrace{cg{\mapsto}bc}\}$
- Using Transitivity Rule - $cg{\mapsto}d$ is redundant
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, ce{\mapsto}g\} \bigcup \{cg{\mapsto}c, cg{\mapsto}bc\}$

Deduction with classical logics for FDs

Removing redundancy - FD logic of R. Fagin

{ab↦c, c↦a, bc↦d, acd↦b, d↦eg, be↦c, cg↦bd, ce ↦ag}
Rules: Fragmentation 3 times, Augmentation, Reflexivity, Union

- $\{ab{\mapsto}c, c{\mapsto}a, \underbrace{bc{\mapsto}d}, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, \cancel{cg{\mapsto}d}, ce{\mapsto}g\} \bigcup \{cg{\mapsto}c, \underbrace{cg{\mapsto}bc}\}$
- Using Transitivity Rule - $cg{\mapsto}d$ is redundant
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, ce{\mapsto}g\} \bigcup \{cg{\mapsto}c, cg{\mapsto}bc\}$

Deduction with classical logics for FDs

Removing redundancy - FD logic of R. Fagin

{ab↦c, c↦a, bc↦d, acd↦b, d↦eg, be↦c, cg↦bd, ce ↦ag}
Rules: Fragmentation 3 times, Augmentation, Reflexivity, Union, Transitivity

- $\{ab{\mapsto}c, \underline{c{\mapsto}a}, bc{\mapsto}d, \underline{acd{\mapsto}b}, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, ce{\mapsto}g\}$ $\bigcup\{cg{\mapsto}c, cg{\mapsto}bc\}$
- Using Pseudotransitivity Rule - $cd{\mapsto}b$ is added as derived FD
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, ce{\mapsto}g\}$ $\bigcup\{cg{\mapsto}c, cg{\mapsto}bc, cd{\mapsto}b\}$

Deduction with classical logics for FDs

Removing redundancy - FD logic of R. Fagin

$\{ab{\mapsto}c,\ c{\mapsto}a,\ bc{\mapsto}d,\ acd{\mapsto}b,\ d{\mapsto}eg,\ be{\mapsto}c,\ cg{\mapsto}bd,\ ce\ {\mapsto}ag\}$

Rules: Fragmentation 3 times, Augmentation, Reflexivity, Union, Transitivity

- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, ce{\mapsto}g\}$ $\bigcup\{cg{\mapsto}c, cg{\mapsto}bc\}$
- Using Pseudotransitivity Rule - $cd{\mapsto}b$ is added as derived FD
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, acd{\mapsto}b, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, ce{\mapsto}g\}$ $\bigcup\{cg{\mapsto}c, cg{\mapsto}bc, cd{\mapsto}b\}$

Deduction with classical logics for FDs

Removing redundancy - FD logic of R. Fagin

$\{ab{\mapsto}c,\ c{\mapsto}a,\ bc{\mapsto}d,\ acd{\mapsto}b,\ d{\mapsto}eg,\ be{\mapsto}c,\ cg{\mapsto}bd,\ ce{\mapsto}ag\}$
Rules: Fragmentation 3 times, Augmentation, Reflexivity, Union, Transitivity, Pseudotransitivity

- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, \cancel{acd{\mapsto}b}, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, ce{\mapsto}g\} \bigcup \{cg{\mapsto}c, cg{\mapsto}bc, \underbrace{cd{\mapsto}b}\}$
- Using Augmentation Rule - $acd{\mapsto}b$ is redundant
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, ce{\mapsto}g\} \bigcup \{cg{\mapsto}c, cg{\mapsto}bc, cd{\mapsto}b\}$

Deduction with classical logics for FDs

Removing redundancy - FD logic of R. Fagin

$\{ab{\mapsto}c,\ c{\mapsto}a,\ bc{\mapsto}d,\ acd{\mapsto}b,\ d{\mapsto}eg,\ be{\mapsto}c,\ cg{\mapsto}bd,\ ce{\mapsto}ag\}$
Rules: Fragmentation 3 times, Augmentation, Reflexivity, Union, Transitivity, Pseudotransitivity

- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, \cancel{acd{\mapsto}b}, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, ce{\mapsto}g\}$ $\bigcup\{cg{\mapsto}c, cg{\mapsto}bc, \underbrace{cd{\mapsto}b}\}$
- Using Augmentation Rule - $acd{\mapsto}b$ is redundant
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, ce{\mapsto}g\}$ $\bigcup\{cg{\mapsto}c, cg{\mapsto}bc, cd{\mapsto}b\}$

Deduction with classical logics for FDs

Removing redundancy - FD logic of R. Fagin

{ab↦c, c↦a, bc↦d, acd↦b, d↦eg, be↦c, cg↦bd, ce ↦ag}
Rules: Fragmentation 3 times, Augmentation, Reflexivity, Union, Transitivity, Pseudotransitivity, Augmentation

- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, ce{\mapsto}g\}$ $\bigcup\{cg{\mapsto}c, cg{\mapsto}bc, \underline{cd{\mapsto}b}\}$
- **From the three derived rules, one of them is added to the set of FDs, and the other two are removed.**
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, ce{\mapsto}g, cd{\mapsto}b\}$

Deduction with classical logics for FDs

Removing redundancy - FD logic of R. Fagin

{ab↦c, c↦a, bc↦d, acd↦b, d↦eg, be↦c, cg↦bd, ce ↦ag}
Rules: Fragmentation 3 times, Augmentation, Reflexivity, Union, Transitivity, Pseudotransitivity, Augmentation

- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, ce{\mapsto}g\}$ $\bigcup\{cg{\mapsto}c, cg{\mapsto}bc, \underline{cd{\mapsto}b}\}$
- **From the three derived rules, one of them is added to the set of FDs, and the other two are removed.**
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, ce{\mapsto}g, cd{\mapsto}b\}$

Deduction with classical logics for FDs

From

$\{ab{\mapsto}c,\ c{\mapsto}a,\ bc{\mapsto}d,\ acd{\mapsto}b,\ d{\mapsto}eg,\ be{\mapsto}c,\ cg{\mapsto}bd,\ ce{\mapsto}ag\}$

Using in different ways:

Fragmentation 3 times, Augmentation, Reflexivity, Union, Transitivity, Pseudotransitivity, Augmentation, (*to add one derived FD*)

A set without redundancy

$\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, ce{\mapsto}g, cd{\mapsto}b\}$

Classical logics for FDs: Armstrong's Axioms

Automated manipulation of FDs is not possible using the FDs logics.

- What rules must be applied and in which direction?
- In which order must be selected?
- At the end, we must clean the redundant FDs.

Only one way: Natural Deduction.

Automated method with Simplification logic

Removing redundancy

{ab↦c, c↦a, bc↦d, acd↦b, d↦eg, be↦c, cg↦bd, ce ↦ag}

- $\{ab{\mapsto}c, \underbrace{c{\mapsto}a}, bc{\mapsto}d, \underbrace{acd{\mapsto}b}, d{\mapsto}eg, be{\mapsto}c, cg{\mapsto}bd, ce{\mapsto}ag\}$
- Simplification
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, cd{\mapsto}b, d{\mapsto}eg, be{\mapsto}c, cg{\mapsto}bd, ce{\mapsto}ag\}$

Automated method with Simplification logic

Removing redundancy

$\{ab\mapsto c,\ c\mapsto a,\ bc\mapsto d,\ acd\mapsto b,\ d\mapsto eg,\ be\mapsto c,\ cg\mapsto bd,\ ce\mapsto ag\}$

- $\{ab\mapsto c, \underline{c\mapsto a}, bc\mapsto d, \underline{acd\mapsto b}, d\mapsto eg, be\mapsto c, cg\mapsto bd, ce\mapsto ag\}$
- Simplification
- $\{ab\mapsto c, c\mapsto a, bc\mapsto d, cd\mapsto b, d\mapsto eg, be\mapsto c, cg\mapsto bd, ce\mapsto ag\}$

Automated method with Simplification logic

Removing redundancy

{ab↦c, c↦a, bc↦d, acd↦b, d↦eg, be↦c, cg↦bd, ce ↦ag} Rules: Simplification

- $\{ab{\mapsto}c, \underbrace{c{\mapsto}a}, bc{\mapsto}d, cd{\mapsto}b, d{\mapsto}eg, be{\mapsto}c, cg{\mapsto}bd, \underbrace{ce{\mapsto}ag}\}$
- r-Simplification
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, cd{\mapsto}b, d{\mapsto}eg, be{\mapsto}c, cg{\mapsto}bd, ce{\mapsto}g\}$

Automated method with Simplification logic

Removing redundancy

$\{ab{\mapsto}c,\ c{\mapsto}a,\ bc{\mapsto}d,\ acd{\mapsto}b,\ d{\mapsto}eg,\ be{\mapsto}c,\ cg{\mapsto}bd,\ ce{\mapsto}ag\}$ Rules: Simplification, r-Simplification

- $\{ab{\mapsto}c, c{\mapsto}a, \underline{bc{\mapsto}d}, cd{\mapsto}b, d{\mapsto}eg, be{\mapsto}c, \underline{cg{\mapsto}bd}, ce{\mapsto}g\}$
- r-Simplification
- $\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, cd{\mapsto}b, d{\mapsto}eg, be{\mapsto}c, cg{\mapsto}b, ce{\mapsto}g\}$

Automated method with Simplification logic

Removing redundancy

$\{ab{\mapsto}c,\ c{\mapsto}a,\ bc{\mapsto}d,\ acd{\mapsto}b,\ d{\mapsto}eg,\ be{\mapsto}c,\ cg{\mapsto}bd,\ ce\ {\mapsto}ag\}$ Rules:

Automatically:

Simplification, r-Simplification, r-Simplification

The same set without redundancy

$\{ab{\mapsto}c, c{\mapsto}a, bc{\mapsto}d, d{\mapsto}e, d{\mapsto}g, be{\mapsto}c, cg{\mapsto}b, ce{\mapsto}g, cd{\mapsto}b\}$

$\mathbf{SL}_{\mathrm{FD}}$

logic is adequate to design automated methods to reason with FDs.

Automated method to remove redundancy

INPUT: Γ (a set of FDs)
OUTPUT: Γ' (a FDs set with less redundancy)
BEGIN
1. $\lfloor$*Reduc*$\rfloor$ + $\lfloor$*Axiom*$\rfloor$
2. $\lfloor$*Union*$\rfloor$

REPEAT
3. Simplification $\lfloor$*Simp*$\rfloor$ + $\lfloor$*rSimp*$\rfloor$

UNTIL *more simplifications cannot be applied*
4. Check if it is possible to apply Generalized Transitivity

END

Important improvement with respect the rest of FDs algorithms: all of them apply the rule $\lfloor$*Frag*$\rfloor$ *as their first transformation.*

Outline

1. Background
2. Algebraic framework
3. **Logic**
   - $\mathbf{SL}_{FD}$ logic
   - Redundancy: Classical logics versus $\mathbf{SL}_{FD}$ logic
   - Closure
   - Minimal Keys
4. Conclusions

$\mathbf{SL}_{\mathrm{FD}}$ closure

Closure via functional dependence simplification, A. Mora et.al., IJCM, 89 (4), 2012

- We present an automated method directly based on Simplification Logic to calculate the closure of a set of attributes.
- Fields of application goes from theoretical areas as algebra or geometry to practical areas as databases and artificial intelligence: *data analysis, knowledge structures, knowledge compilation, redundant constraint elimination, query optimization, finding key problem,etc.*

$\mathbf{SL}_{\mathrm{FD}}$ closure

Theorem

- **Equivalency I**: If $U \subseteq W$ then $\{\top \mapsto W, U \mapsto V\} \equiv_{\mathcal{S}_{\mathrm{FD}}} \{\top \mapsto WV\}$
- **Equivalency II**: If $V \subseteq W$ then $\{\top \mapsto W, U \mapsto V\} \equiv_{\mathcal{S}_{\mathrm{FD}}} \{\top \mapsto W\}$
- **Equivalency III**: If $U \cap W \neq \varnothing$ or $V \cap W \neq \varnothing$ then
$$\{\top \mapsto W, U \mapsto V\} \equiv_{\mathcal{S}_{\mathrm{FD}}} \{\top \mapsto W, U - W \mapsto V - W\}$$

Automated Prover to obtain the closure

From Γ and X, calculate X^+ (the closure of X):

- Add $\top \mapsto X$
- Apply systematically the three equivalences based on $\mathbf{SL}_{\mathrm{FD}}$ logic.

Result: $\top \mapsto X^+$

$\mathbf{SL}_{\mathrm{FD}}$ closure

Theorem

- **Equivalency I**: If $U \subseteq W$ then $\{\top \mapsto W, U \mapsto V\} \equiv_{\mathcal{S}_{\mathrm{FD}}} \{\top \mapsto WV\}$
- **Equivalency II**: If $V \subseteq W$ then $\{\top \mapsto W, U \mapsto V\} \equiv_{\mathcal{S}_{\mathrm{FD}}} \{\top \mapsto W\}$
- **Equivalency III**: If $U \cap W \neq \varnothing$ or $V \cap W \neq \varnothing$ then
$$\{\top \mapsto W, U \mapsto V\} \equiv_{\mathcal{S}_{\mathrm{FD}}} \{\top \mapsto W, U - W \mapsto V - W\}$$

Automated Prover to obtain the closure

From Γ and X, calculate X^+ (the closure of X):

- Add $\top \mapsto X$
- Apply systematically the three equivalences based on $\mathbf{SL}_{\mathrm{FD}}$ logic.

Result: $\top \mapsto X^+$

Execution

Closure of $\{$afd$\}$

$\{ak \mapsto bc,\ cd \mapsto gh,\ cij \mapsto kl,\ de \mapsto f,\ g \mapsto de,\ hf \mapsto ia,\ f \mapsto c\}$

Execution

Closure of $\{afd\}$

$\{ak \mapsto bc,\ cd \mapsto gh,\ cij \mapsto kl,\ de \mapsto f,\ g \mapsto de,\ hf \mapsto ia,\ f \mapsto c\}$

$\top \mapsto afd$

$ak \mapsto bc$	$cd \mapsto gh$	$cij \mapsto kl$	$de \mapsto f$	$g \mapsto de$	$hf \mapsto ia$	$f \mapsto c$
$k \mapsto bc$	$c \mapsto gh$	$cij \mapsto kl$	$\times$	$g \mapsto e$	$h \mapsto i$	$\times$
(III)	(III)	none	(II)	(III)	(III)	(I)
						$\top \mapsto afdc$

Execution

Closure of $\{$afd$\}$

$\{$ak$\mapsto$bc, cd$\mapsto$gh, cij$\mapsto$kl, de$\mapsto$f, g$\mapsto$de, hf$\mapsto$ia, f$\mapsto$c$\}$

$\top\mapsto$afd

ak$\mapsto$bc	cd$\mapsto$gh	cij$\mapsto$kl	de$\mapsto$f	g$\mapsto$de	hf$\mapsto$ia	f$\mapsto$c
k$\mapsto$bc	c$\mapsto$gh	cij$\mapsto$kl	$\times$	g$\mapsto$e	h$\mapsto$i	$\times$
(III)	(III)	none	(II)	(III)	(III)	(I)
						$\top\mapsto$afdc

Execution

Closure of $\{afd\}$

$\{ak{\mapsto}bc,\ cd{\mapsto}gh,\ cij{\mapsto}kl,\ de{\mapsto}f,\ g{\mapsto}de,\ hf{\mapsto}ia,\ f{\mapsto}c\}$

$\top{\mapsto}afd$

$ak{\mapsto}bc$	$cd{\mapsto}gh$	$cij{\mapsto}kl$	$de{\mapsto}f$	$g{\mapsto}de$	$hf{\mapsto}ia$	$f{\mapsto}c$
$k{\mapsto}bc$	$c{\mapsto}gh$	$cij{\mapsto}kl$	$\times$	$g{\mapsto}e$	$h{\mapsto}i$	$\times$
(III)	(III)	none	(II)	(III)	(III)	(I)
						$\top{\mapsto}afdc$

Execution

Closure of $\{afd\}$

$\{ak \mapsto bc,\ cd \mapsto gh,\ cij \mapsto kl,\ de \mapsto f,\ g \mapsto de,\ hf \mapsto ia,\ f \mapsto c\}$

$\top \mapsto afd$

$ak \mapsto bc$	$cd \mapsto gh$	$cij \mapsto kl$	$de \mapsto f$	$g \mapsto de$	$hf \mapsto ia$	$f \mapsto c$
$k \mapsto bc$	$c \mapsto gh$	$cij \mapsto kl$	$\times$	$g \mapsto e$	$h \mapsto i$	$\times$
(III)	(III)	none	(II)	(III)	(III)	(I)
						$\top \mapsto afdc$

Execution

Closure of $\{$afd$\}$

$\{ak{\mapsto}bc,\ cd{\mapsto}gh,\ cij{\mapsto}kl,\ de{\mapsto}f,\ g{\mapsto}de,\ hf{\mapsto}ia,\ f{\mapsto}c\}$

$\top{\mapsto}afd$

$ak{\mapsto}bc$	$cd{\mapsto}gh$	$cij{\mapsto}kl$	$de{\mapsto}f$	$g{\mapsto}de$	$hf{\mapsto}ia$	$f{\mapsto}c$
$k{\mapsto}bc$	$c{\mapsto}gh$	$cij{\mapsto}kl$	$\times$	$g{\mapsto}e$	$h{\mapsto}i$	$\times$
(III)	(III)	none	(II)	(III)	(III)	(I)
						$\top{\mapsto}afdc$

Execution

Closure of $\{$afd$\}$

$\{ak \mapsto bc,\ cd \mapsto gh,\ cij \mapsto kl,\ de \mapsto f,\ g \mapsto de,\ hf \mapsto ia,\ f \mapsto c\}$

$\top \mapsto afd$

$ak \mapsto bc$	$cd \mapsto gh$	$cij \mapsto kl$	$de \mapsto f$	$g \mapsto de$	$hf \mapsto ia$	$f \mapsto c$
$k \mapsto bc$	$c \mapsto gh$	$cij \mapsto kl$	$\times$	$g \mapsto e$	$h \mapsto i$	$\times$
(III)	(III)	none	(II)	(III)	(III)	(I)
						$\top \mapsto afdc$

Execution

Closure of $\{afd\}$

$\{ak \mapsto bc,\ cd \mapsto gh,\ cij \mapsto kl,\ de \mapsto f,\ g \mapsto de,\ hf \mapsto ia,\ f \mapsto c\}$

$\top \mapsto afd$

$ak \mapsto bc$	$cd \mapsto gh$	$cij \mapsto kl$	$de \mapsto f$	$g \mapsto de$	$hf \mapsto ia$	$f \mapsto c$
$k \mapsto bc$	$c \mapsto gh$	$cij \mapsto kl$	$\times$	$g \mapsto e$	$h \mapsto i$	$\times$
(III)	(III)	none	(II)	(III)	(III)	(I)
						$\top \mapsto afdc$

Execution

Closure of $\{afd\}$

$\{ak\mapsto bc,\ cd\mapsto gh,\ cij\mapsto kl,\ de\mapsto f,\ g\mapsto de,\ hf\mapsto ia,\ f\mapsto c\}$

$\top\mapsto afdc$

$k\mapsto bc$	$c\mapsto gh$	$cij\mapsto kl$ $\times$	$g\mapsto e$	$h\mapsto i$ $\times$
$k\mapsto b$	$\times$	$ij\mapsto kl$	$\times$	$\times$
(III)	(I)	(III)	(I)	(I)
	$\top\mapsto afdcgh$		$\top\mapsto afdcghe$	$\top\mapsto afdcghei$

Execution

Closure of $\{afd\}$

$\{ak \mapsto bc,\ cd \mapsto gh,\ cij \mapsto kl,\ de \mapsto f,\ g \mapsto de,\ hf \mapsto ia,\ f \mapsto c\}$

$\top \mapsto afdc$

$k \mapsto bc$	$c \mapsto gh$	$cij \mapsto kl$	$\times$	$g \mapsto e$	$h \mapsto i$	$\times$
$k \mapsto b$	$\times$	$ij \mapsto kl$		$\times$	$\times$	
(III)	(I)	(III)		(I)	(I)	
	$\top \mapsto afdcgh$			$\top \mapsto afdcghe$	$\top \mapsto afdcghei$	

Execution

Closure of $\{afd\}$

$\{ak\mapsto bc,\ cd\mapsto gh,\ cij\mapsto kl,\ de\mapsto f,\ g\mapsto de,\ hf\mapsto ia,\ f\mapsto c\}$

$\top\mapsto afdc$

$k\mapsto bc$	$c\mapsto gh$	$cij\mapsto kl$	$\times$	$g\mapsto e$	$h\mapsto i$	$\times$
$k\mapsto b$	$\times$	$ij\mapsto kl$		$\times$	$\times$	
(III)	(I)	(III)		(I)	(I)	
	$\top\mapsto afdcgh$			$\top\mapsto afdcghe$	$\top\mapsto afdcghei$	

Execution

Closure of $\{afd\}$

$\{ak \mapsto bc,\ cd \mapsto gh,\ cij \mapsto kl,\ de \mapsto f,\ g \mapsto de,\ hf \mapsto ia,\ f \mapsto c\}$

$\top \mapsto afdc$

$k \mapsto bc$	$c \mapsto gh$	$cij \mapsto kl$	$\times$	$g \mapsto e$	$h \mapsto i$	$\times$
$k \mapsto b$	$\times$	$ij \mapsto kl$		$\times$	$\times$	
(III)	(I)	(III)		(I)	(I)	
	$\top \mapsto afdcgh$			$\top \mapsto afdcghe$	$\top \mapsto afdcghei$	

Execution

Closure of $\{$afd$\}$

$\{ak{\mapsto}bc,\ cd{\mapsto}gh,\ cij{\mapsto}kl,\ de{\mapsto}f,\ g{\mapsto}de,\ hf{\mapsto}ia,\ f{\mapsto}c\}$

$\top{\mapsto}afdcghei$

$k{\mapsto}b$	$ij{\mapsto}kl$
none	none

$\top{\mapsto}afdcghei$

Execution

Closure of $\{afd\}$

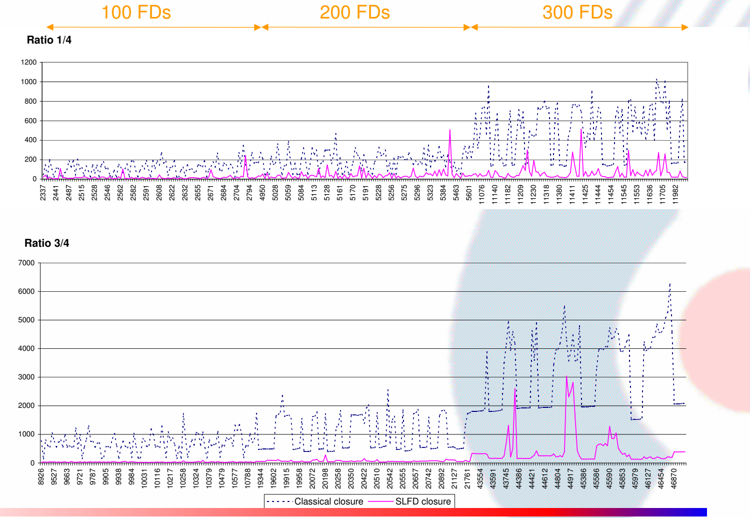
$\{ak \mapsto bc, cd \mapsto gh, cij \mapsto kl, de \mapsto f, g \mapsto de, hf \mapsto ia, f \mapsto c\}$

$\top \mapsto afdcghei$

$k \mapsto b$	$ij \mapsto kl$
none	none

$\top \mapsto afdcghei$

SL_{FD} closure: Results



Outline

Pruning the search space for keys

Ideal non-deterministic operators as a formal framework to reduce the key finding problem, A. Mora et. al., IJCM, 88 (9), 2011

- We have presented a formal method in the framework of the lattice theory to prune the problem of finding all the minimal keys.
- With lineal cost, this prune method provides a longer reduction than the rest of techniques (The %-reduction in an experiment was the 70,52 %).

We define $\varrho_a : A \to (a]$ with $\varrho_a(x) = x \wedge a$

- $((a], \leq)$ defines a Boole Algebra
- $\pi : L \to L/_{\equiv_a}$ is the homomorphism that assigns to x its equivalence class $\mathbb{C}_a(x)$
- $\Psi : L/_{\equiv_a} \to (a]$ is the isomorphism defined as $\Psi(\mathbb{C}_a(x)) = \varrho_a(x)$

$$\begin{tikzcd}
L \arrow[rr, "\varrho_a"] \arrow[d, "\pi : x \to \mathbb{C}_a(x)"'] & & (a] \\
L/_{\equiv_a} \arrow[urr, "\Psi : \mathbb{C}_a(x) \to \varrho_a(x)"'] & &
\end{tikzcd}$$

Prunning the scheme

Algorithm: *core* and the *body* of *R*

Let $R =< \mathcal{A}, \Gamma >$ be a relational schema.

1. $Dnt(\Gamma) = \bigcup_{X \mapsto Y \in \Gamma} X$
2. $Dte(\Gamma) = \bigcup_{X \mapsto Y \in \Gamma} Y$
3. core $= \mathcal{A} - Dte(\Gamma)$
4. body $= (Dnt(\Gamma) \cap (\mathcal{A} - core^{+}))$

Theorem

Let $R =< \mathcal{A}, \Gamma >$ be a scheme. Let $\mathcal{K}$ be a minimal key of R, then we have that $\texttt{core}_F \subseteq \mathcal{K} \subseteq (\texttt{core}_F \cup \texttt{body}_F)$.

Wastl Method

- Wastl introduces a Hilbert style inference system, called $\mathbb{K}$, for deriving all keys.
- Wastl builds a tableaux which represents the search space to find all the keys applying the inference system $\mathbb{K}$.

The rules of the $\mathbb{K}$ inference system

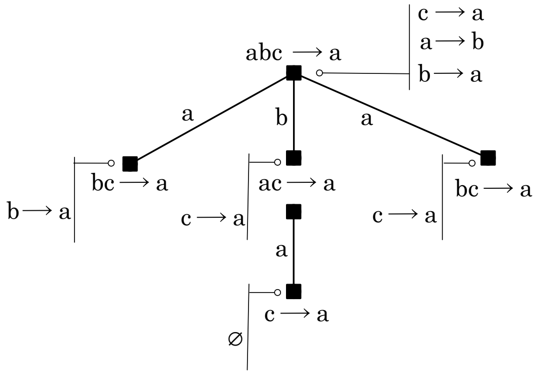
Rules of inference:

$$\mathbb{K}_1 : \quad \frac{X{\mapsto}a \qquad Ya{\mapsto}b}{XY{\mapsto}b}$$

$$\mathbb{K}_2 : \quad \frac{X{\mapsto}a \qquad Y{\mapsto}b}{XY{\mapsto}b}$$

Wastl Method

Let $\mathcal{A} = \{a, b, c\}$ and $\Gamma = \{c\mapsto a, a\mapsto b, b\mapsto a\}$. We build the root of the Wastl tree ($abc\mapsto a$) by applying the $\mathbb{K}_2$ rule. And applying $\mathbb{K}_1$ we build the tableaux.



SL_{FD}-Key Algorithm

Automated reasoning to infer all minimal keys, P. Cordero et.al., Submitted.

Definition: Ψ-Operator

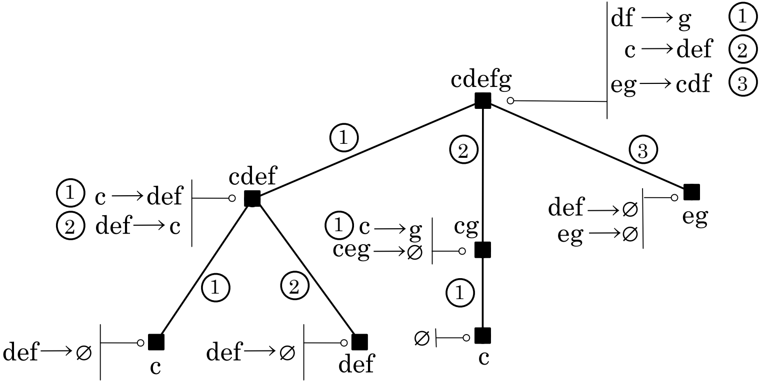
$$\Psi_{X \mapsto Y}(U \mapsto V) = \begin{cases} \text{U} \mapsto \text{V-Y, if } U \cap Y = \varnothing \\ \text{(UX)-Y} \mapsto \text{V-(XY) otherwise} \end{cases}$$

$$\Psi_{X \mapsto Y}(\Gamma) = \{\Psi_{X \mapsto Y}(U \mapsto V) \mid U \mapsto V \in \Gamma\}$$

SL_{FD}-Key Algorithm

Example

Let $\mathcal{A} = \{a, b, c, d, e, f, g\}$ and $\Gamma = \{adf{\mapsto}g, c{\mapsto}def, eg{\mapsto}bcdf\}$.
We have that $\texttt{core}_F = \{a\}$ and $\texttt{body}_F = \{c, d, e, f, g\}$. So, we reduce the problem considering $\mathcal{A}' = \{c, d, e, f, g\}$ and $\Gamma' = \{df{\mapsto}g, c{\mapsto}def, eg{\mapsto}cdf\}$.



Execution

Results:

- Keys in our tableaux are $\{c, def, eg\}$
- $core = \{a\}$
- Thus the set of all the minimal keys is $\{ac, adef, aeg\}$.
- Our tableaux has 7 nodes and 3 levels of depth, while this same example in Wastl's method produces a tableaux of 56 nodes and 5 levels of depth.

Outline

1. Background
2. Algebraic framework
3. Logic
   - $\mathbf{SL}_{FD}$ logic
   - Redundancy: Classical logics versus $\mathbf{SL}_{FD}$ logic
   - Closure
   - Minimal Keys
4. Conclusions

Conclusions

- Formalization of FDs as non-deterministic operators in the algebraic framework has guided us to:
  - A logic for functional dependencies: **Simplification Logic for FDs**.
  - Automated methods based on logic to:
    - **remove redundancy**.
    - **calculate the closure**.
    - **obtain all minimal keys**.
- Simplification Logic can be applied in extensions of classical models: fuzzy extensions, XML extensions, FCA.


Gracias por su atención
Děkuji za pozornost
Thank you for your attention
Спасибо за ваше внимание
Ďakujem za pozornosť
Je vous remercie de votre attention
Vielen Dank für Ihre Aufmerksamkeit
Dziękuję za uwagę