The Relative Complexity of Constraints on Database Views

Stephen J. Hegner
Umeå University
Department of Computing Science
Umeå, Sweden

http://www.cs.umu.se/~hegner

Statement of the Problem

- Two fundamental services to be provided by a database system:

  Support for updates: The data which are stored in the system may be altered.

  - ➪ The term *update* is used here in a general sense, to denote *insertions* and *deletions*, as well as alterations of existing values.

  Support via views: Access to parts of the database, via windows which are known as *views*, is allowed.

- Determining that a proposed update is acceptable involves verifying that it satisfies the extant database *constraints* or *dependencies*.
- This verification process has a *computational complexity*.
- The view inherits its *implied constraints* from the main schema.

Problem: Characterize the complexity of verifying the implied constraints on the view as a function of the complexity of verifying the constraints on the main schema.

The Relational Model and Functional Dependencies

Example relational schema and instance:

$\mathcal{F}_{\mathsf{P}} = \{\mathsf{Name} \longrightarrow \mathsf{SalCl},\ \ \mathsf{SalCl} \longrightarrow \mathsf{Salary}\}$

Rel P :

Name	SalCl	Salary	Proj
Smith	9	50000	A
Jones	13	125000	A
Jones	13	125000	B
Wilson	11	90000	C

$\mathcal{F}_{\mathsf{Q}} = \{\mathsf{Proj} \longrightarrow \mathsf{Budget}\}$

Rel Q :

Proj	Budget
A	100
C	300
D	300

- The *schema* is shown in goldenrod .
  - The schema defines which instances are legal via the *dependencies* ($\mathcal{F}_{\mathsf{P}}$ and $\mathcal{F}_{\mathsf{Q}}$).
  - The schema does not change with updates.
- The *instance* is shown in blue-grey . The instance changes with updates.
- The *functional dependency* $\mathsf{Name} \longrightarrow \mathsf{SalCl}$ expresses that any two tuples in the instance of relation P with the same value for attribute Name must also have the same value for attribute SalCl.

The k-Submodel Property

Observation: A relation r satisfies a set of functional dependencies $\mathcal{F}$ iff every subrelation of r consisting of two tuples satisfies $\mathcal{F}$. $\square$

- The complexity of checking for satisfaction of functional dependencies may thus be viewed as $O(n^2)$, with n the number of tuples in the relation.

Definition: Let $\mathbf{R}$ be a relational database schema with constraints $\mathcal{C}$.

(a) An instance M of of $\mathbf{R}$ is called a *k-model* if every subset of M consisting of at most k tuples satisfies the constraints in $\mathcal{C}$.

(b) $\mathbf{R}$ is said to have the *k-submodel property* if for every instance M, M satisfies the constraints in $\mathcal{C}$ iff M is a k-model.

- The complexity of constraint checking in the context of the k-submodel property is $O(n^k)$.
  - Thus, a schema constrained by functional dependencies has the 2-submodel property.
  - The converse also holds within a wide class of database dependencies.

An Example View in the Relational Context

Example base schema and instance:

$\mathcal{F}_{\mathsf{P}} = \{\mathsf{Name} \longrightarrow \mathsf{SalCl},\ \ \mathsf{SalCl} \longrightarrow \mathsf{Salary}\}$

Rel P :

Name	SalCl	Salary	Proj
Smith	9	50000	A
Jones	13	125000	A
Jones	13	125000	B
Wilson	11	90000	C

$\mathcal{F}_{\mathsf{Q}} = \{\mathsf{Proj} \longrightarrow \mathsf{Budget}\}$

Rel Q :

Proj	Budget
A	100
C	300
D	300

Example view $\Gamma = \pi_{\mathsf{Name,Salary}}(P)$

$\mathcal{F}_{\Gamma} = \{\mathsf{Name} \longrightarrow \mathsf{Salary}\}$

Name	Salary
Smith	50000
Jones	125000
Wilson	90000

- The constraint $\mathsf{Name} \longrightarrow \mathsf{Salary}$ is a logical consequence of those in $\mathcal{F}_{\mathsf{P}}$, even though it is not included explicitly in that set.
- It is the only constraint which holds on the view.

An Example from Relational Folklore

Conjecture: The projection of a relational schema constrained by functional dependencies is itself constrained by functional dependencies.

- This conjecture is **false**. A counterexample provides insight into why it fails.

Main schema:

$$(R[ABCD],\ \mathcal{F} = \{A \to D,\ B \to D,\ CD \to A\})$$

Fact: The main schema has the 2-submodel property.

View:

$$\Pi_{ABC} = (R[ABC], \pi_{ABC})$$

Fact: The view does not have the k-submodel property for any $k \geq 0$.

Illustration of the Example

Claim: With distinct symbols identifying distinct values:

- The structure below is not a legal state of Π_{ABC}.
- Any proper subset is a legal state of Π_{ABC}.

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_2	c_3
a_2	b_3	c_4
a_3	b_3	c_5
a_3	b_4	c_6
a_4	b_4	c_7
a_4	b_5	c_8
$\vdots$	$\vdots$	$\vdots$
a_{k-1}	b_{k-1}	c_{2k-3}
a_{k-1}	b_k	c_{2k-2}
$\vdots$	$\vdots$	$\vdots$
a_{m-1}	b_{m-1}	c_{2m-3}
a_{m-1}	b_m	c_{2m-2}
a_m	b_m	c_1

Illustration of the Example

$$\{A \to D,\ B \to D,\ CD \to A\}$$

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_1
a_2	b_2	c_3	d_2
a_2	b_3	c_4	d_2
a_3	b_3	c_5	d_3
a_3	b_4	c_6	d_3
a_4	b_4	c_7	d_4
a_4	b_5	c_8	d_4
$\vdots$	$\vdots$	$\vdots$	$\vdots$
a_{k-1}	b_{k-1}	c_{2k-3}	d_{k-1}
a_{k-1}	b_k	c_{2k-2}	d_{k-1}
$\vdots$	$\vdots$	$\vdots$	$\vdots$
a_{m-1}	b_{m-1}	c_{2m-3}	d_{m-1}
a_{m-1}	b_m	c_{2m-2}	d_{m-1}
a_m	b_m	c_1	d_m

Illustration of the Example

$$\{A \to D,\ B \to D,\ CD \to A\}$$

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_1
a_2	b_2	c_3	d_1
a_2	b_3	c_4	d_1
a_3	b_3	c_5	d_1
a_3	b_4	c_6	d_1
a_4	b_4	c_7	d_1
a_4	b_5	c_8	d_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
a_{k-1}	b_{k-1}	c_{2k-3}	d_1
a_{k-1}	b_k	c_{2k-2}	d_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
a_{m-1}	b_{m-1}	c_{2m-3}	d_1
a_{m-1}	b_m	c_{2m-2}	d_1
a_m	b_m	c_1	d_1

Illustration of the Example

$$\{A \to D,\ B \to D,\ CD \to A\}$$

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_1
a_2	b_2	c_3	d_1
a_2	b_3	c_4	d_1
a_3	b_3	c_5	d_1
a_3	b_4	c_6	d_1
a_4	b_4	c_7	d_1
a_4	b_5	c_8	d_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
a_{k-1}	b_{k-1}	c_{2k-3}	d_1
a_{k-1}	b_k	c_{2k-2}	d_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
a_{m-1}	b_{m-1}	c_{2m-3}	d_1
a_{m-1}	b_m	c_{2m-2}	d_1
$a_1 = a_m$	b_m	c_1	d_1

Illustration of the Example

$$\{A \to D,\ B \to D,\ CD \to A\}$$

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_1
a_2	b_2	c_3	d_1
a_2	b_3	c_4	d_1
a_3	b_3	c_5	d_1
a_3	b_4	c_6	d_2
a_4	b_4	c_7	d_2
a_4	b_5	c_8	d_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$
a_{k-1}	b_{k-1}	c_{2k-3}	d_2
a_{k-1}	b_k	c_{2k-2}	d_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$
a_{m-1}	b_{m-1}	c_{2m-3}	d_2
a_{m-1}	b_m	c_{2m-2}	d_2
a_m	b_m	c_1	d_2

Major Questions Addressed in this Work

Question: To what extent does this explosion of complexity impact the support for updates to views of database schemata?

- Updates to views generally follow some prescribed *strategy*.
  - Arbitrary updates are not allowed.
  - Those updates which are allowed arise from specific changes to a database which are already known to satisfy the constraints.

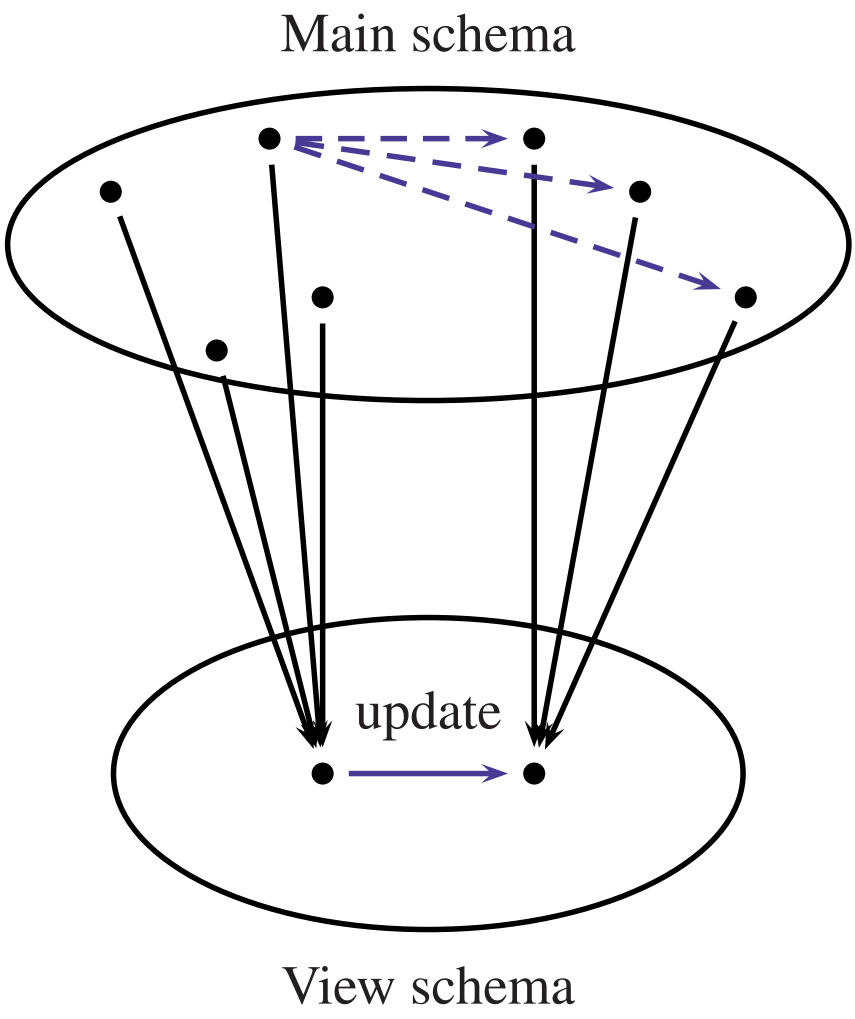
Question: Do these limitations result in a more manageable complexity for determining whether proposed updates to views satisfy the implied constraints?

Answer: For the class of *closed update strategies* on schemata characterized by the *k-submodel property*, the answer is **yes**.

- The complexity of testing a proposed update for correctness is no more complex than a corresponding test in the main schema.
- This is true even though the full axiomatization of the view schema may be much more complex than that of the main schema.

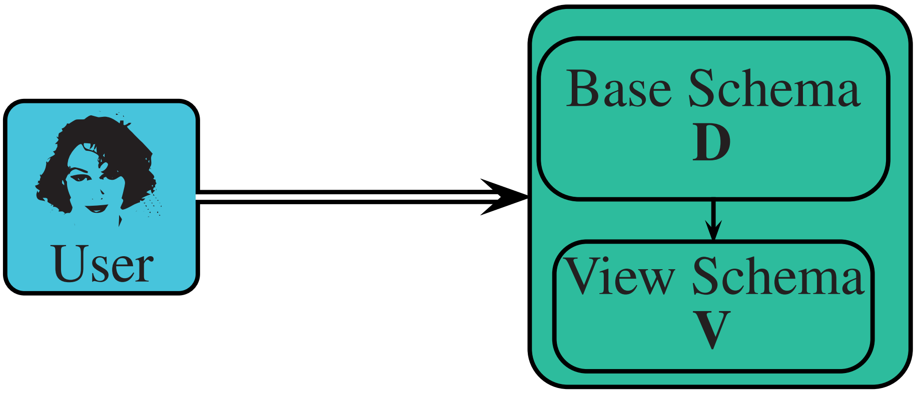
The View-Update Translation Problem

- For a potential update to a view, there are generally many possible *translations* of that update to the main schema.
- For a potential update to the view, two questions must be considered:
  - Should the update be allowed at all?
  - If so, which of the possible translations to the main schema should be selected?



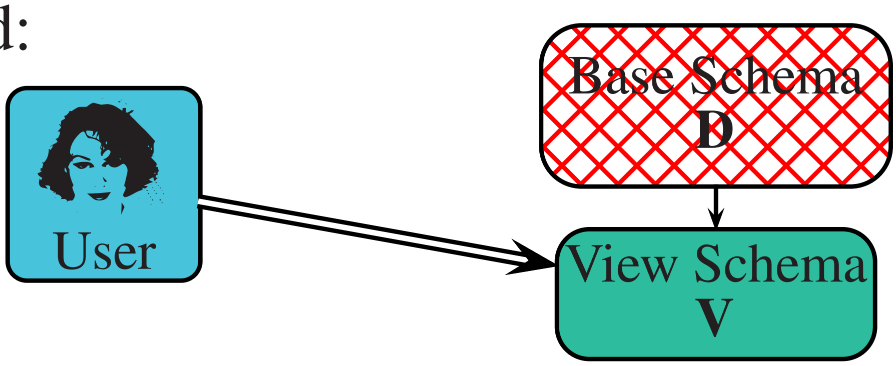
Open vs. Closed Views

Open:



- The user has access to both the view and the base schema.
- The view is provided as a convenience.
- Uniqueness criteria suffice for update translation.

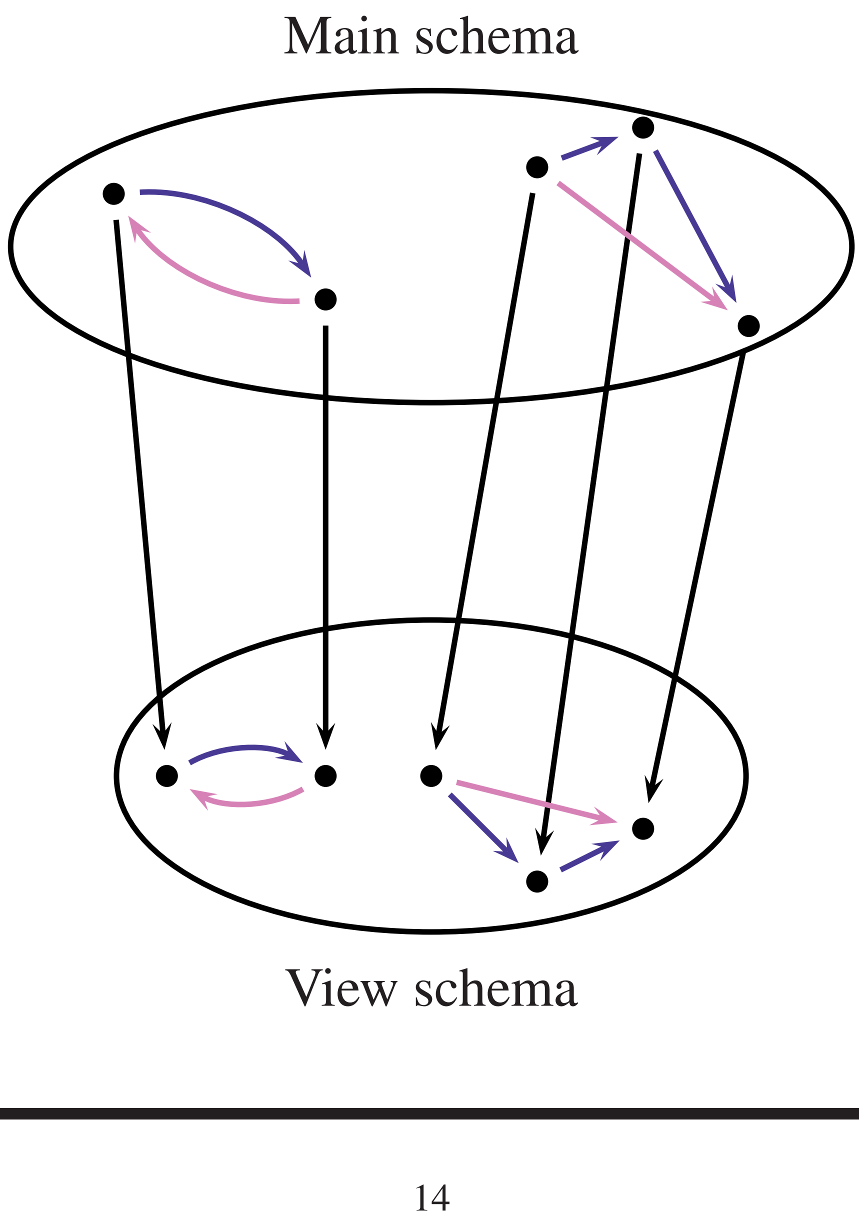
Closed:



- The user has access only to the view.
- The user has no direct knowledge of the base schema.
- The view must be self contained in terms of knowledge needed to effect updates.
- The view should preferably look "just" like a complete base schema.
- Interface criteria for update translations are extremely important.

Closed Update Strategies

- In the work considered here:
  - The admissibility of a view update must depend only upon the state of the view, and not otherwise upon the state of the main schema.
  - View updates and their translations are required to be *reversible* and *transitive*.
- These are called *closed update strategies*.



Constant-Complement Update Strategies

- A *complement* of view Γ is a view Γ' with the property that the state of the main schema is recoverable from the combined states of Γ and Γ'. [$\{\Gamma,\Gamma'\}$ defines a *lossless decomposition*.]
- In other words, for no pair $\{M_1, M_2\}$ of database instances of the main schema, is it the case that M_1 and M_2 map to the same state in both Γ and Γ'.

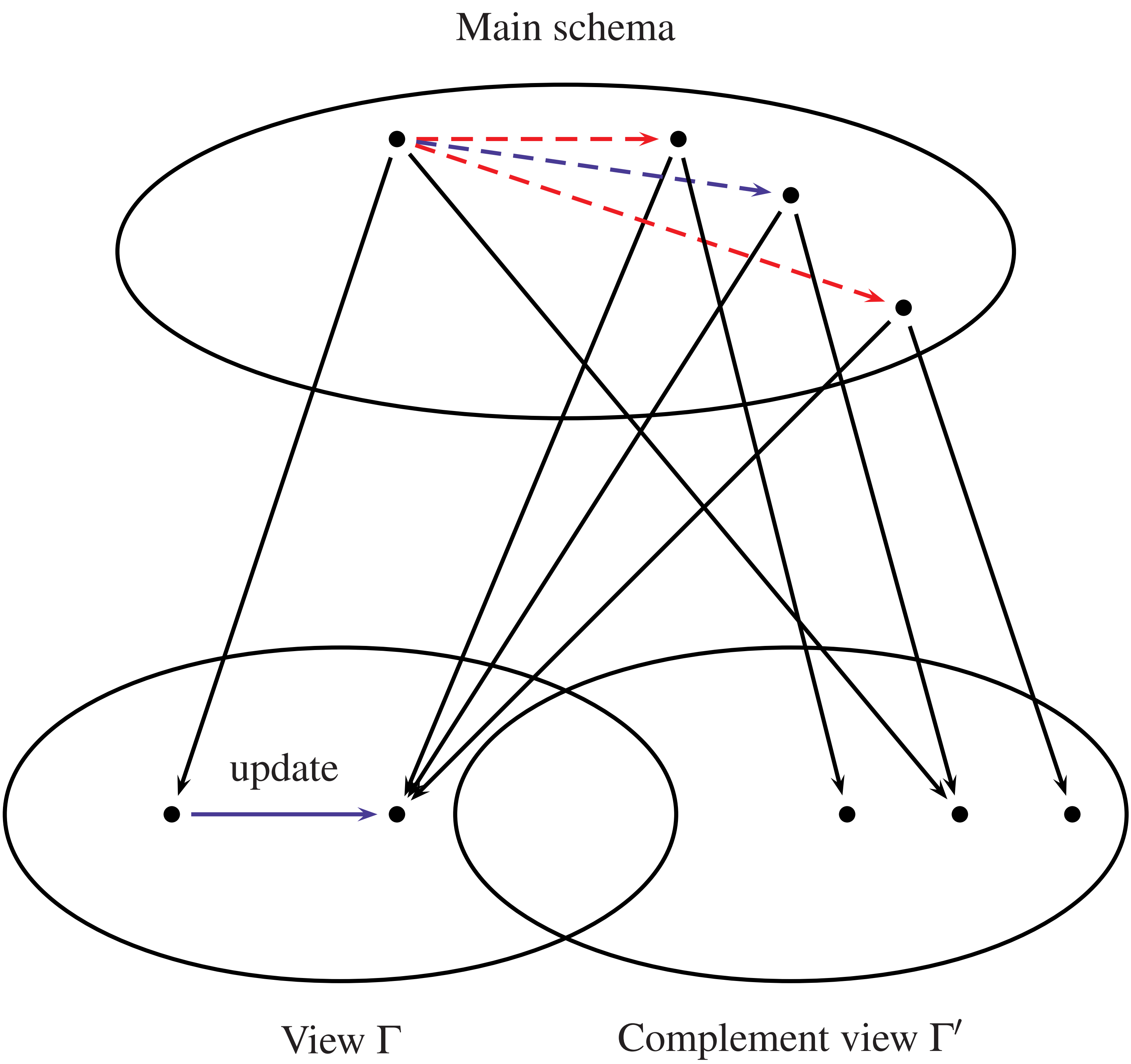
Observation: There is at most one translation of an update on Γ which keeps Γ' constant. $\square$

- A *constant-complement update translation* on Γ is one which keeps a complement Γ' fixed.

Theorem (Bancilhon & Spyratos 1981): Every closed update strategy is a constant-complement strategy. $\square$

Theorem (Hegner 1990, 2002): A constant-complement update strategy is a closed update strategy iff the congruences of the two views commute. In this case, the composition of the congruences defines a view, called the *meet*, which must be held constant during all updates. $\square$

Visualization of Constant-Complement Update Strategies



Closed Update in the Relational Context

Example base schema and instance:

$(E[\mathsf{ABC}], \{\mathsf{B} \to \mathsf{C}\})$:

A	B	C
a_0	b_0	c_0
a_1	b_1	c_1
a_2	b_1	c_1

View to be updated:

View $\Pi_{\mathsf{AB}} = (E[\mathsf{AB}], \emptyset)$

A	B
a_0	b_0
a_1	b_1
a_2	b_1

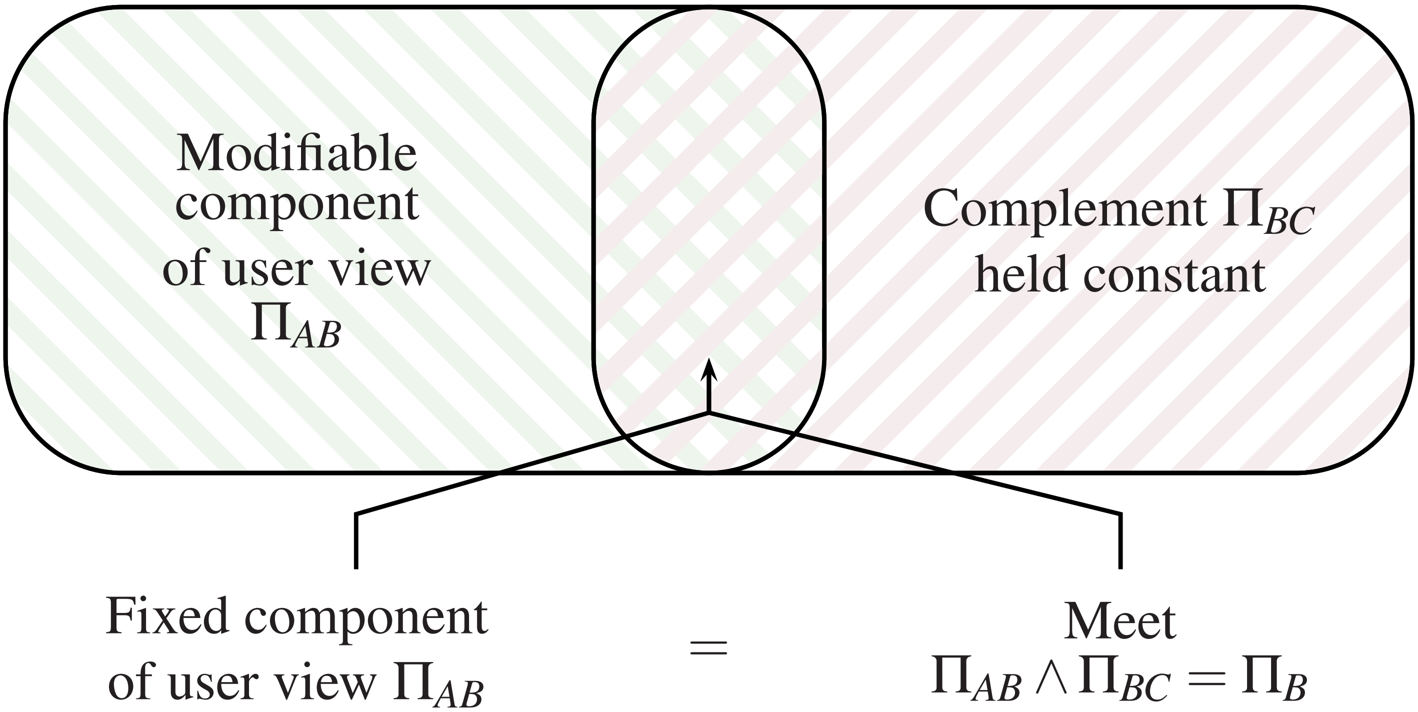
Complementary view:

View $\Pi_{\mathsf{BC}} =$ $(E[\mathsf{BC}], \{\mathsf{B} \to \mathsf{C}\})$

B	C
b_0	c_0
b_1	c_1

- The updates which are allowed to $E[\mathsf{AB}]$ under constant complement Π_{BC} are precisely those which hold the view Π_{B} fixed.
- This is the part of Π_{BC} which is visible within Π_{AB}.
- The allowed updates to Π_{AB} do not depend upon other aspects of the view Π_{BC}.
- The view $\Pi_{\mathsf{B}} = (E[\mathsf{B}], \emptyset)$ is called the *meet* of Π_{AB} and Π_{BC}.

An Alternate Perspective on the Meet



Non-Closed Update in the Relational Context

Example base schema and instance:

$(E[\mathsf{ABC}], \{\mathsf{A} \to \mathsf{C}, \mathsf{B} \to \mathsf{C}\})$:

A	B	C
a_0	b_0	c_0
a_1	b_1	c_1
a_2	b_1	c_1

View to be updated:

View $\Pi_{\mathsf{AB}} = (E[\mathsf{AB}], \emptyset)$

A	B
a_0	b_0
a_1	b_1
a_2	b_1

Complementary view:

View $\Pi_{\mathsf{BC}} =$
$(E[\mathsf{BC}], \{\mathsf{B} \to \mathsf{C}\})$

B	C
b_0	c_0
b_1	c_1

- This decomposition is not dependency preserving.
- The allowed updates to Π_{AB} now depend upon information which is contained in the view Π_{BC} but not in Π_{B}.
- The associated update strategy is not closed.
- The view Π_B is not the meet of $\{\Pi_{AB}, \Pi_{BC}\}$.

Non-Closed Update in the Relational Context

Example base schema and instance:

$(E[\mathsf{ABC}], \{\mathsf{A} \to \mathsf{C}, \mathsf{B} \to \mathsf{C}\})$:

A	B	C
a_0	b_0	c_0
a_1	b_1	c_1
a_2	b_1	c_1
a_0	b_1	c_1

View to be updated:

View $\Pi_{\mathsf{AB}} = (E[\mathsf{AB}], \emptyset)$

A	B
a_0	b_0
a_1	b_1
a_2	b_1
a_0	b_1

Complementary view:

View $\Pi_{\mathsf{BC}} =$
$(E[\mathsf{BC}], \{\mathsf{B} \to \mathsf{C}\})$

B	C
b_0	c_0
b_1	c_1

- This decomposition is not dependency preserving.
- The allowed updates to Π_{AB} now depend upon information which is contained in the view Π_{BC} but not in Π_{B}.
- The associated update strategy is not closed.
- The view Π_B is not the meet of $\{\Pi_{AB}, \Pi_{BC}\}$.

Non-Closed Update in the Relational Context

Example base schema and instance:

$(E[\mathsf{ABC}], \{\mathsf{A} \to \mathsf{C}, \mathsf{B} \to \mathsf{C}\})$:

A	B	C
a_0	b_0	c_0
a_1	b_1	c_0
a_2	b_1	c_0
a_0	b_1	c_0

View to be updated:

View $\Pi_{\mathsf{AB}} = (E[\mathsf{AB}], \emptyset)$

A	B
a_0	b_0
a_1	b_1
a_2	b_1
a_0	b_1

Complementary view:

View $\Pi_{\mathsf{BC}} =$ $(E[\mathsf{BC}], \{\mathsf{B} \to \mathsf{C}\})$

B	C
b_0	c_0
b_1	c_0

- This decomposition is not dependency preserving.
- The allowed updates to Π_{AB} now depend upon information which is contained in the view Π_{BC} but not in Π_{B}.
- The associated update strategy is not closed.
- The view Π_B is not the meet of $\{\Pi_{AB}, \Pi_{BC}\}$.

The Underlying Syntactic Schemata and Views

- On the *syntactic schemata*, (almost) all candidate databases are allowed; the constraints are ignored.
  - On the main schema, only the *decomposition dependency* is enforced.
  - On the view schemata, no dependencies are enforced.
- The associated *syntactic decomposition* will always be a meet-complementary decomposition (*i.e.*, with commuting congruences), since this property depends only upon the decomposition dependency.

Theorem: The semantic decomposition is meet complementary iff a *cover* of the constraints on the main schema (excluding the decomposition dependency) embeds in the views. □

Illustration: Syntactic Schemata and Views

Example quasi-syntactic base schema and instance:

$(E[\mathsf{ABC}], \{\bowtie[\mathsf{AB}, \mathsf{BC}]\})$:

A	B	C
a_0	b_0	c_0
a_1	b_1	c_1
a_2	b_1	c_2
a_1	b_1	c_2
a_2	b_1	c_1

Syntactic view to be updated:

View $\Pi_{\mathsf{AB}} = (E[\mathsf{AB}], \emptyset)$

A	B
a_0	b_0
a_1	b_1
a_2	b_1

Complementary syntactic view:

View $\Pi_{\mathsf{BC}} = (E[\mathsf{BC}], \emptyset)$

B	C
b_0	c_0
b_1	c_1
b_1	c_2

- The *join dependency* $\bowtie[AB, BC]$ is defined by

$$(\forall..)(E(x_1, y_1, z_1) \wedge E(x_2, y_1, z_2) \Rightarrow E(x_2, y_1, z_1) \wedge E(x_1, y_1, z_2))$$

- $B \rightarrow C \models \bowtie[AB, BC]$.

Illustration: Cover

Example base schema and instance:

$(E[\mathsf{ABC}], \{\mathsf{A} \longrightarrow \mathsf{B}, \mathsf{A} \longrightarrow \mathsf{C}, \mathsf{B} \longrightarrow \mathsf{A}, \mathsf{C} \longrightarrow \mathsf{A}\})$:

A	B	C
a_0	b_0	c_0
a_1	b_1	c_1
a_2	b_2	c_2

Syntactic view to be updated:

View $\Pi_{\mathsf{AB}} =$
$(E[\mathsf{AB}], \{\mathsf{A} \longrightarrow \mathsf{B}, \mathsf{B} \longrightarrow \mathsf{A}\})$

A	B
a_0	b_0
a_1	b_1
a_2	b_2

Complementary syntactic view:

View $\Pi_{\mathsf{BC}} =$
$(E[\mathsf{BC}], \{\mathsf{B} \longrightarrow \mathsf{C}, \mathsf{C} \longrightarrow \mathsf{B}\})$

B	C
b_0	c_0
b_1	c_1
b_2	c_2

- The dependencies $\mathsf{A} \longrightarrow \mathsf{C}$ and $\mathsf{C} \longrightarrow \mathsf{A}$ do not embed in either view.
- The families $\{\mathsf{A} \longrightarrow \mathsf{B}, \mathsf{A} \longrightarrow \mathsf{C}, \mathsf{B} \longrightarrow \mathsf{A}, \mathsf{C} \longrightarrow \mathsf{A}\}$ and $\{\mathsf{A} \longrightarrow \mathsf{B}, \mathsf{B} \longrightarrow \mathsf{C}, \mathsf{B} \longrightarrow \mathsf{A}, \mathsf{C} \longrightarrow \mathsf{B}\}$ are equivalent to each other.
- The family $\{\mathsf{A} \longrightarrow \mathsf{B}, \mathsf{B} \longrightarrow \mathsf{C}, \mathsf{B} \longrightarrow \mathsf{A}, \mathsf{C} \longrightarrow \mathsf{B}\}$ is a *cover* for $\{\mathsf{A} \longrightarrow \mathsf{B}, \mathsf{A} \longrightarrow \mathsf{C}, \mathsf{B} \longrightarrow \mathsf{A}, \mathsf{C} \longrightarrow \mathsf{A}\}$.

The Main Complexity Result

- First, the results are expressed in the context of a single-relation schema which is decomposed into two projections which form a meet complementary pair.

Theorem: A database M of the main schema is a k-model iff its image in each of the two views is a k-model. □

Corollary: If the main schema has the k-submodel property, and the view is to be updated from legal state M to candidate state M' via constant complement (*i.e.*, keeping the meet constant), then for M' to be a legal state of the view, it suffices that it be a k-model. □

Example of Constraint Complexity on a Closed View

Example base schema and instance:

$(E[\mathsf{ABCDE}], \mathcal{F} = \{\mathsf{A} \rightarrow \mathsf{D}, \mathsf{B} \rightarrow \mathsf{D}, \mathsf{CD} \rightarrow \mathsf{A}, \mathsf{A} \rightarrow \mathsf{E}\})$:

A	B	C	D	E
a_0	b_0	c_0	d_0	e_0
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	d_2	e_2

View to be updated:

$\Pi_{\mathsf{ABCE}} = (E[\mathsf{ABCE}], \{\mathsf{A} \rightarrow \mathsf{E}\} \cup \pi_{\mathsf{ABC}}(\mathcal{F})).$

A	B	C	E
a_0	b_0	c_0	e_0
a_1	b_1	c_1	e_1
a_2	b_1	c_2	e_2

Complementary view:

$\Pi_{\mathsf{ABCD}} = (E[\mathsf{ABCD}], \{\mathsf{A} \rightarrow \mathsf{D}, \mathsf{B} \rightarrow \mathsf{D}, \mathsf{CD} \rightarrow \mathsf{A}\}).$

A	B	C	D
a_0	b_0	c_0	d_0
a_1	b_1	c_1	d_1
a_2	b_2	c_2	d_2

- The view Π_{ABCE} is not finitely axiomatizable.
- Any update must hold $\Pi_{\mathsf{ABC}} = \Pi_{\mathsf{ABCE}} \wedge \Pi_{\mathsf{ABCD}}$ fixed.
- The allowed updates to Π_{ABCE} are limited to changes in the values of E, and these are governed solely by $\mathsf{A} \rightarrow \mathsf{E}$.

The General Context of this Work

- The results just described are not limited to the relational model.
- They hold in a very general context which is independent of the specific data model.

The finite-powerset model:

- X = a set (not necessarily finite) of *elements*. (Think of *tuples* in the relational setting).
- *Syntactic databases* = $\mathscr{P}_f(X)$ = all finite subsets of X.
- *Semantic databases* = a subset of $\mathscr{P}_f(X)$ defined by the extant constraints.
- *Morphisms* are defined element-wise:

$$f(M) = \bigcup\{f(\{s\}) \mid s \in M\}$$

  and are required to preserve the empty set and be *basis preserving*:

$$(\mathsf{Card}(M) = 1) \Rightarrow (\mathsf{Card}(f(M)) \leq 1)$$

- Views are defined by surjective morphisms.
  - This recaptures *projection* and *selection* in the classical relational setting, but not *join*.

Generating vs. Denial Constraints

- Classical database constraints fall into two categories:

  Equality-generating dependencies (EGDs) are rules with an equality on the right hand side.

  - Functional dependencies are an example.
  - Here is the logical rule for the FD $A \to B$ on $R(A,B,C)$.

  $$(\forall \ldots)((R(a_1,b_1,c_1) \wedge R(a_1,b_2,c_2)) \Rightarrow (b_1 = b_2))$$

  - If the rule fails, this can be determined by finding a match left-hand side which fails to satisfy the equality.
  - They may thus be viewed as *(existential) denial rules*.
  - Such rules define naturally a k-submodel property.

  (Full) tuple generating dependencies (FTGDs) are rules with a tuple predicate on the right-hand side.

  - Join dependencies and *inclusion dependencies* are natural examples.
  - Such constraints can never be recaptured using the k-submodel property.

- Together, EGDs + FTGDs form the *full implicational dependencies (FIDs)*.

Question: Do results regarding the complexity of constraints on views which are similar to those for denial constraints hold in the context of generating dependencies?

Answer: Not in general.

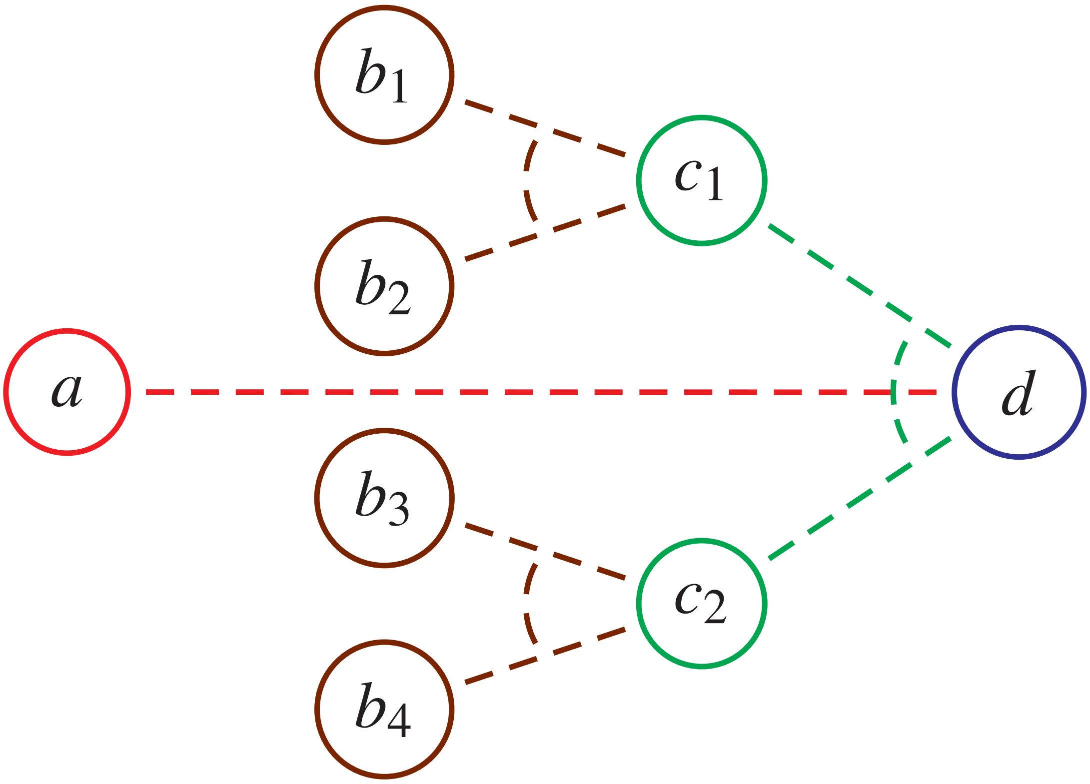
A Simple Counterexample

- $X = \{a, b_1, b_2, b_3, b_4, c_1, c_2, d\}$.
- The legal databases M are those subsets of X which satisfy the following constraints.

$$(a \in M) \Leftrightarrow (d \in M)$$
$$(c_1 \in M) \Leftrightarrow ((b_1 \in M) \wedge (b_2 \in M))$$
$$(c_2 \in M) \Leftrightarrow ((b_3 \in M) \wedge (b_4 \in M))$$
$$(d \in M) \Leftrightarrow ((c_1 \in M) \wedge (c_2 \in M))$$

- Note that the last rule, for example, is an abbreviation for the following three.

$$(d \in M) \Rightarrow (c_1 \in M) \qquad (d \in M) \Rightarrow (c_2 \in M)$$
$$((c_1 \in M) \wedge (c_2 \in M)) \Rightarrow (d \in M)$$

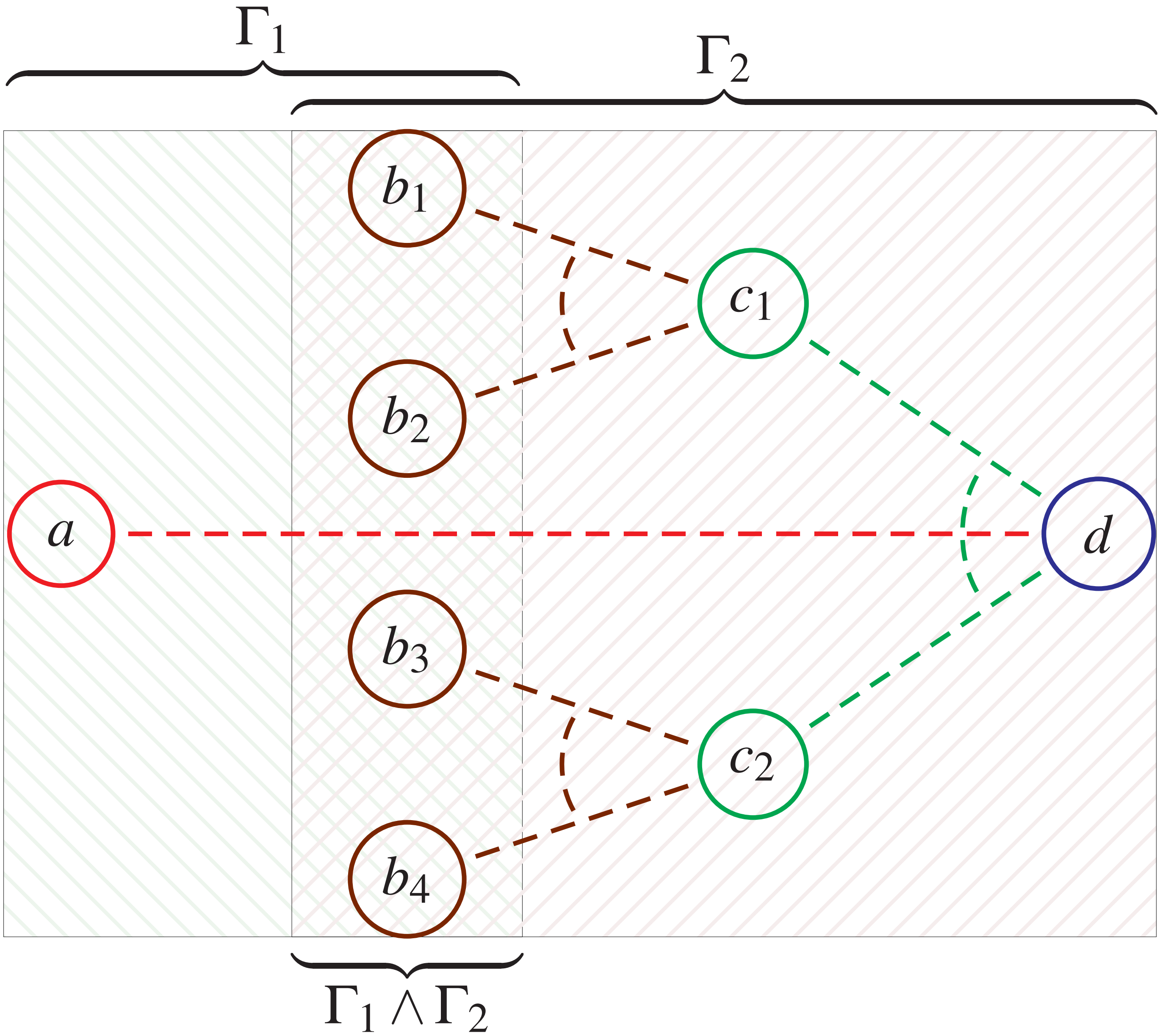


A Simple Counterexample — Continued

- Decompose this schema into meet-complementary views Γ_1 and Γ_2 as follows.
- It is easily seen that the following constraint holds on Γ_1.

$$(a \in M) \Leftrightarrow ((b_1 \in M) \wedge (b_2 \in M) \wedge (b_3 \in M) \wedge (b_4 \in M))$$

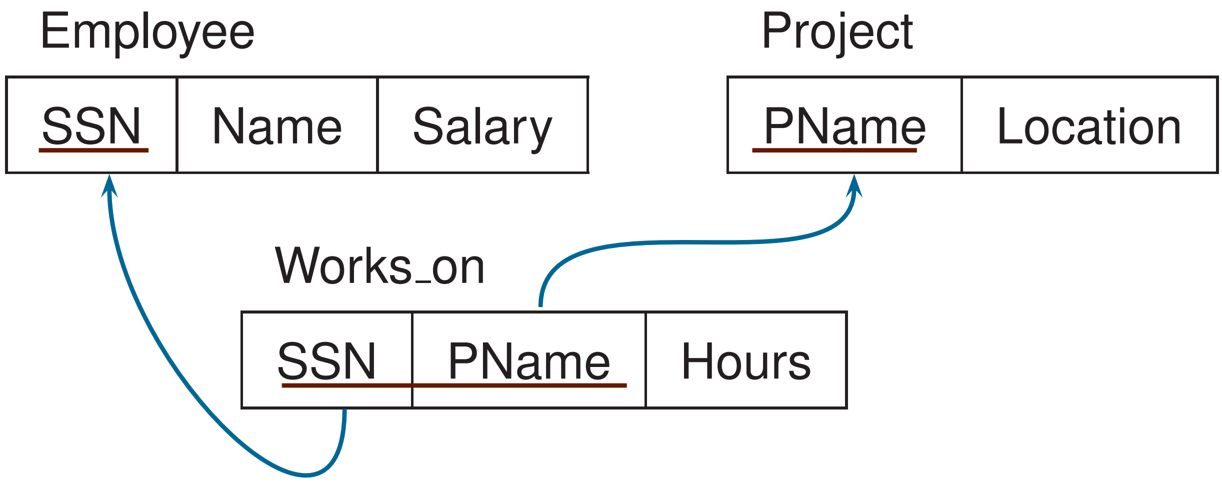
- Thus, the constraints on Γ_1 require antecedents of size four, while the full schema only requires size two.



Counterexamples in "Real" Data Models?

- Similar counterexamples are possible in the classical relational model (even with a single relation) under the following circumstances:
  - If named constants are allowed.
  - If untyped dependencies are allowed. (Counterexamples are more limited.)
- With EGDs and TGDs in a typed relational framework with no named constants, it is unknown whether a counterexample exists.
- A positive result (good complexity bounds) exists in the (multi-)relational framework constrained by EGDs and *fanout-free unary inclusion dependencies (INDs)*.
  - The EGD part is handled by the existing theory using the k-submodel property.
  - The part involving inclusion dependencies is handled using old-fashioned relational database theory.

Illustration of Inclusion Dependencies



- Key constraints (shown underlined in sepia) specify those fields which uniquely determine a tuple.

- Foreign-key constraints (represented as arrows in midnight blue) specify inclusion of key fields.

- Foreign-key constraints are (almost) a special case of *inclusion dependencies*.

- The foreign-key dependency linking SSN of Works_on to SSN of Employee is expressed formally by

$$(\forall s)(\forall p)(\forall h)\mathsf{Works\_on}(s,p,h) \Rightarrow$$
$$((\exists n)(\exists a)(\mathsf{Employee}(s,n,a))$$

- This is a tuple-generating dependency and cannot be checked via a simple denial process, since the matching tuple on the right-hand side does not occur on the left-hand side.

Unary Inclusion Dependencies

General setting: Multi-relation schema with

$$R_1[\mathbf{U}_1],\ R_2[\mathbf{U}_2],\ \ldots\ R_n[\mathbf{U}_n]$$

General inclusion dependency (IND):

$$R_i[\mathbf{W}_i] \subseteq R_j[\mathbf{W}_j]$$

for some $\mathbf{W}_i \subseteq \mathbf{U}_i$ and $\mathbf{W}_j \subseteq \mathbf{U}_j$.

Fact: (Chandra & Vardi 1985) The inference problem for inclusion dependencies and functional dependencies is undecidable. □

Unary inclusion dependency (UIND):

$$R_i[A_i] \subseteq R_j[A_j]$$

for some $A_i \in \mathbf{U}_i$ and $A_j \in \mathbf{U}_j$.

Facts: (Cosmadakis, Kanellakis, & Vardi 1990)

- The inference problem for unary inclusion dependencies and functional dependencies is not only decidable but quite tractable as well.
- This inference problem largely decouples from the inference of other dependencies. □

Inference Rules for FIDs and UINDs

Fact: (Cosmadakis, Kanellakis, & Vardi 1990) For a schema constrained by FIDs (Full Implicational Dependencies) and UINDs, a complete set of inference rules for finite implication is given by any such set of rules for FIDs, plus the rules below.

(a) Reflexivity and transitivity of UINDs:

$$\frac{}{A \subseteq A} \qquad \frac{A \subseteq B \quad B \subseteq C}{A \subseteq C}$$

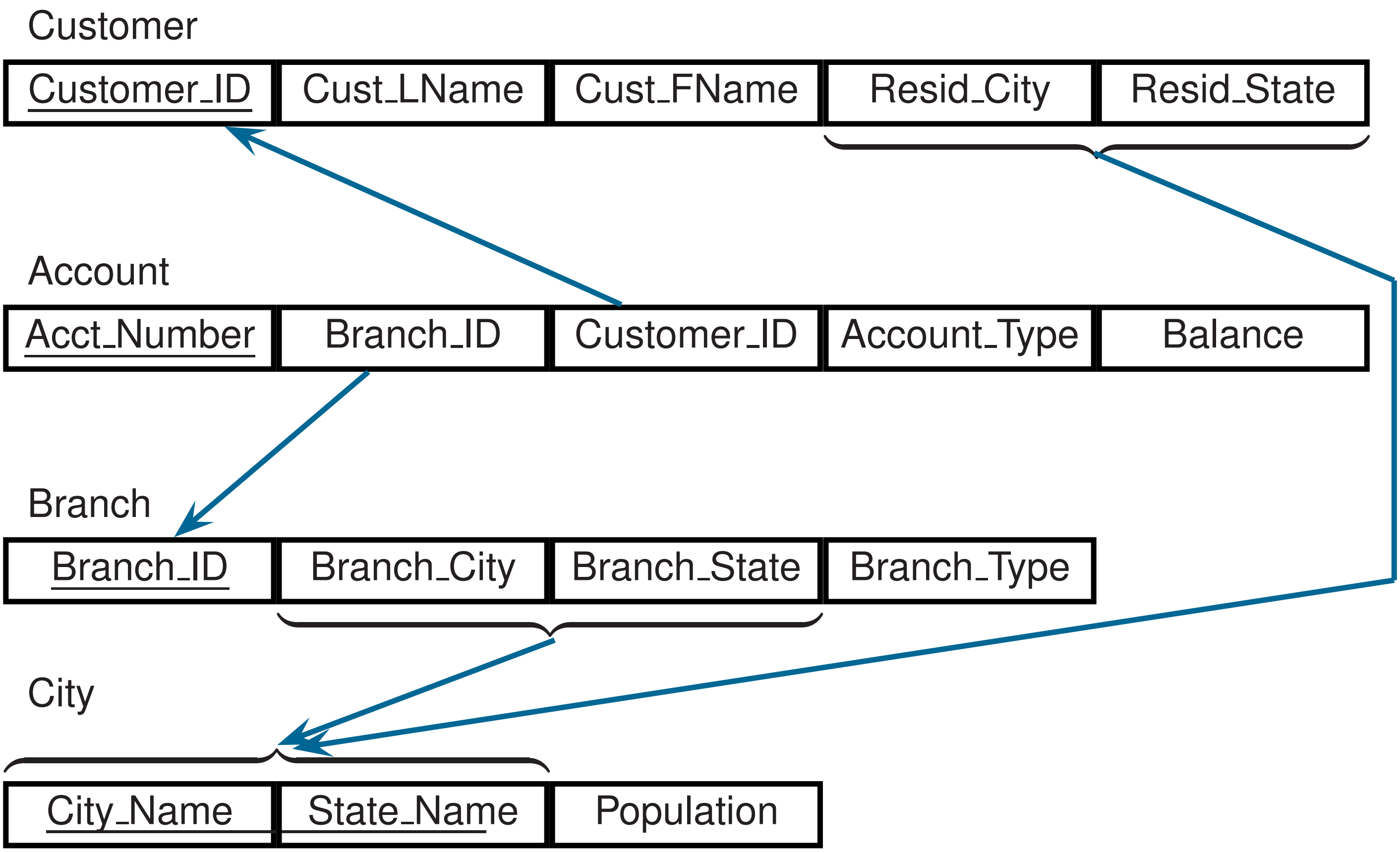
(b) The *cycle rule* for FDs and UINDs on finite databases:

$$\frac{C_0 \rightarrow C_1 \quad C_1 \supseteq C_2 \quad \ldots \quad C_{m-1} \rightarrow C_m \quad C_m \supseteq C_0}{C_1 \rightarrow C_0 \quad C_2 \supseteq C_1 \quad \ldots \quad C_m \rightarrow C_{m-1} \quad C_0 \supseteq C_m}$$

□

Example of Non-Unary Inclusion Dependencies

- Non-unary INDs can occur in realistic examples, as illustrated below.



- However, if the attributes (*e.g.*, (City_Name,State_Name)) can be grouped together in all dependencies, then this group may be treated as a single attribute, and the INDs will be effectively unary.

Fanout-Free Inclusion Dependencies

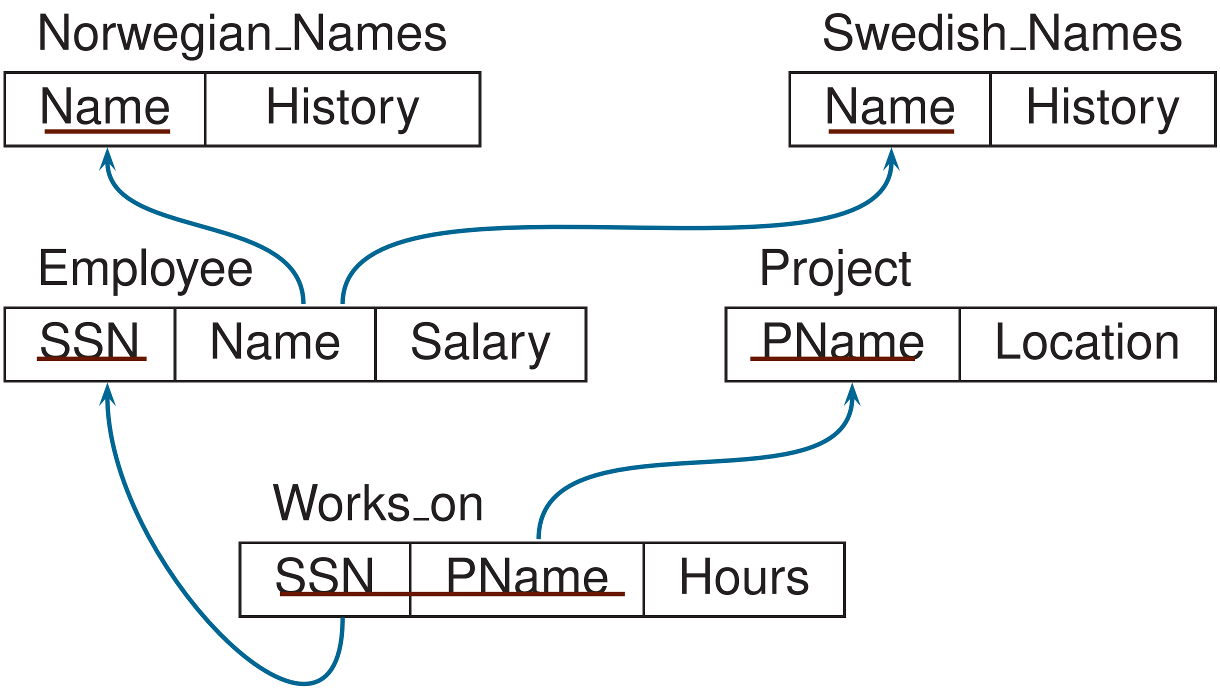
- A set $\mathcal{D}$ of UINDs is *fanout free* if whenever

  $$R_i[A_i] \subseteq R_j[A_j] \qquad R_i[A_i] \subseteq R_k[A_k]$$

  both hold, then one of the following chains must also hold:

  $$R_i[A_i] \subseteq R_j[A_j] \subseteq R_k[A_k] \text{ or } R_i[A_i] \subseteq R_k[A_k] \subseteq R_j[A_j]$$

- It is difficult to imagine practical examples which are not fanout free.



Embedding of UINDs and UIND-Extended k-Models

Proposition:

- Context:
  - $\mathbf{D} = (R[\mathbf{U}], \Phi)$ = relational schema.
  - Φ = FIDs + fanout-free UINDs.
  - $\{(\Pi_{\mathbf{W}_1}, \Phi_1), (\Pi_{\mathbf{W}_2}, \Phi_2)\}$ = meet-complementary decomposition of $\mathbf{D}$ into projections.
- Define:
  - Φ_i^{UIND} = all UINDs which hold on $\Pi_{\mathbf{W}_i}$ for $i = 1, 2$.
- Then for every UIND $\varphi \in \Phi^+$

$$\Phi_1^{\text{UIND}} \cup \Phi_2^{\text{UIND}} \models_f \varphi$$

  using only the transitivity rule for UINDs. $\square$

Definitions:

- A *UIND-extended k-model* of a relational schema $\mathbf{D} = (R[\mathbf{U}], \Phi)$ is a k-model which also satisfies all UINDs in Φ^+.
- $\mathbf{D} = (R[\mathbf{U}], \Phi)$ has the *UIND-extended k-submodel property* if for every instance M, M satisfies the constraints in Φ iff M is a UIND-extended k-model.

The Main Complexity Result in the UIND context

Proposition:

- Context:
  - $\mathbf{D} = (R[\mathbf{U}], \Phi)$ = relational schema.
  - Φ = EGDs + fanout-free UINDs.
  - $\{(\Pi_{\mathbf{W}_1}, \Phi_1), (\Pi_{\mathbf{W}_2}, \Phi_2)\}$ = meet-complementary decomposition of $\mathbf{D}$ into projections.

Theorem: A database M of the main schema is a UIND-extended k-model iff its image in each of the two views is a UIND-extended k-model. $\square$

Corollary: If the main schema has the UIND-extended k-submodel property, and the view is to be updated from legal state M to candidate state M' via constant complement (*i.e.*, keeping the meet constant), then for M' to be a legal state of the view, it suffices that it be a UIND-extended k-model. $\square$

Further Directions

Localized issues:

- Generalize the results for EGDs+UINDs to the more general, data-model-independent framework.
  - UINDs generalize to generating rules with one antecedent.

Longer-range issues regarding views:

- Uniqueness of subdirect complements. (Done; needs to be written.)
- Relax the notion of closed view slightly.
- Views and database components.