Preserving F - Exercises

SAPIENZA
UNIVERSITÀ DI ROMA

The content of the slides is authored by Prof. Maria de Marsico, Sapienza University of Rome
Part of the material is taken from the book: Atzeni, Ceri, Paraboschi, Torlone Basi di dati - Modelli e linguaggi di interrogazione , McGraw-Hill, 2002.

Example 1

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Given the following schema

R = (A, B, C, D)

and the following set of functional dependencies

F = { AB $\rightarrow$ C, D $\rightarrow$ C, D $\rightarrow$ B, C $\rightarrow$ B, D $\rightarrow$ A }

say if the decomposition = { ABC, ABD }

preserves the dependencies in F

it is enough to verify that $F \subseteq G^+$, that is, **every** functional dependency in F is also in G^+

important: it is useless to check the for which **the union of the left and right parts is contained entirely in a subscheme**, because, according to the **definition:**

$\pi_{R_i}(F) = \{ X \rightarrow Y \in F^+ \mid XY \subseteq R_i \}$

these dependencies are already part of G

important 2: the way the algorithm is structured, we can only add attributes to Z (i.e., it never happens that an attribute is deleted from Z), so if Z already contains the **right part of the dependency**, we can be sure that the dependency itself is preserved and stop the algorithm

Example 1

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

R = (A, B, C, D) F = { AB → C, D → C, D → B, C → B, D → A } = { ABC,ABD }

it is enough to check that the dependency D → C is preserved:

warning!

obviously, **(∅)** $^{+}_{F}$ **=** ∅ **whatever** F

Z = D

S = ∅

external for loop on subschemes ABC and ABD

S = S ∪ (D ∩ ABC)$^{+}_{F}$ ∩ ABC = ∅ ∪ (∅)$^{+}_{F}$ ∩ ABC = ∅ ∪ ∅ ∩ ABC =∅

S = S ∪ **(**D ∩ ABD)$^{+}_{F}$ ∩ ABD = S ∪ **(**D)$^{+}_{F}$ ∩ ABD) = **...**

by applying the algorithm on the closure of a set of attributes we have (D) $^{+}_{F}$ = DCBA

... = S ∪ **(**D)$^{+}_{F}$ ∩ ABD = ∅ ∪ DCBA ∩ ABD = ABD

intersection **first**!

∅ ∩ **X =** ∅
whatever X is!!!

Example 1

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

$R = (A, B, C, D)$ $F = \{ AB \rightarrow C, D \rightarrow C, D \rightarrow B, C \rightarrow B, D \rightarrow A \} = \{ ABC, ABD \}$

$ABD \not\subset D$ ($S \not\subset Z$) then **we enter the while loop**

$Z = Z \cup S = ABD$

loop on subschemas ABC and ABD:

Intersection first!

$S = S \cup (ABD \cap ABC)^+_F \cap ABC = S \cup (AB)^+_F \cap ABC = ABD \cup ABC \cap ABC = ABCD$

$S = S \cup (ABD \cap ABD)^+_F \cap ABD = S \cup (ABD)^+_F \cap ABD = ABCD \cup ABCD \cap ABD = ABCD \cup ABD = ABCD$

by applying the algorithm: $(AB)^+_F = ABC$ and $(ABD)^+_F = ABCD$

$ABCD \not\subset ABD$ so, we re-enter the while loop

$Z = Z \cup S = ABCD$

Example 1

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

R = (A, B, C, D) F = { AB $\rightarrow$ C, D $\rightarrow$ C, D $\rightarrow$ B, C $\rightarrow$ B, D $\rightarrow$ A } = { ABC,ABD }

ABCD $\not\subset$ ABD so, we re-enter the while loop

Z = Z $\cup$ S = ABCD

loop on subschemas ABC and ABD:

S = S $\cup$ (ABCD $\cap$ ABC) $\cap$ ABC^+_F = S $\cup$ $(ABC)^+_F$ $\cap$ ABC = ABCD $\cup$ ABC $\cap$ ABC = ABCD

S = S $\cup$ (ABCD $\cap$ ABD) $\cap$ ABD^+_F = S $\cup$ $(ABD)^+_F$ $\cap$ ABD = ABCD $\cup$ ABCD $\cap$ ABC = ABCD $\cup$ ABC = ABCD

the algorithm stops, but **the contents of Z must be checked**

S $\subset$ Z so, stop

Z= $(D)^+_G$= ABCD C $\in$ $(D)^+_G$ so, the dependency is preserved

since **$(D)^+_G$= ABCD** we observe that **D $\rightarrow$ B** and **D $\rightarrow$ A** are also preserved

Example 1

- **we now check the other dependencies**

Example 1

$R = (A, B, C, D)$ $F = \{ AB \rightarrow C, D \rightarrow C, D \rightarrow B, C \rightarrow B, D \rightarrow A \} = \{ ABC, ABD \}$

let's start with $AB \rightarrow C$

$Z = AB$

$S = \varnothing$

loop on subschemes ABC and ABD:

$S = S \cup (AB \cap ABC)^+_F \cap ABC = S \cup (AB)^+_F \cap ABC$

by applying the algorithm on the closure of a set of attributes we have

$(AB)^+_F = ABC$

$S = S \cup (AB)^+_F \cap ABC = \varnothing \cup ABC \cap ABC = ABC$

$S = S \cup (AB \cap ABD)^+_F \cap ABD = ABC \cup (AB)^+_F \cap ABD = ABC \cup ABC \cap ABD = ABC$

Example 1

R = (A, B, C, D) F = { AB → C, D → C, D → B, C → B, D → A } = { ABC,ABD }

ABC ⊄ AB then we enter the loop

Z = Z ∪ S = ABC

we could stop the algorithm, as C ∈ Z ⊂ $(AB)^+_G$

in solving an examination exercise, you can stop **by providing the above motivation**

loop on subschemes ABC and ABD:

S = S ∪ $(ABC \cap ABC)^+_F$ ∩ ABC = S ∪ $(ABC)^+_F$ ∩ ABC

by applying the algorithm on the closure of a set of attributes we have:

$(ABC)^+_F$ = ABC

S = S ∪ $(ABC)^+_F$ ∩ ABC = ABC ∪ ABC ∩ ABC = ABC

S = S ∪ $(ABC \cap ABD)^+_F$ ∩ ABD = S ∪ $(AB)^+_F$ ∩ ABD = ABC ∪ ABC ∩ ABD = ABC ∪AB = ABC

Example 1

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- $S \subset Z$ so, we stop

- $\mathbf{Z = (AB)^+_G = ABC}$
- $\mathbf{C \in (AB)^+_G}$ so, the dependency is preserved

Example 1

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

$R = (A, B, C, D)$ $F = \{ AB \rightarrow C, D \rightarrow C, D \rightarrow B, C \rightarrow B, D \rightarrow A \} = \{ ABC, ABD \}$

finally, we check that $C \rightarrow B$ is preserved:

$Z = C$

$S = \varnothing$

loop on subchemes ABC and ABD:

$S = S \cup (C \cap ABC)^+_F \cap ABC = S \cup (C)^+_F \cap ABC$

by applying the algorithm on the closure of a set of attributes we have>

$(C)^+_F = BC$

$S = S \cup (C)^+_F \cap ABC = \varnothing \cup BC \cap ABC = BC$

$S = S \cup (C \cap ABD)^+_F \cap ABD = S \cup (\varnothing)^+_F \cap ABD = BC \cup \varnothing \cap ABD = BC$

Example 1

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

$R = (A, B, C, D)$ $F = \{ AB \rightarrow C, D \rightarrow C, D \rightarrow B, C \rightarrow B, D \rightarrow A \} = \{ ABC, ABD \}$

$BC \not\subset C$ so **we enter the while loop**

$Z = Z \cup S = BC$

while internal for loop on subschemes ABC and ABD

$S = S \cup (BC \cap ABC)^+_F \cap ABC^+_F = S \cup (BC)^+_F \cap ABC$

by applying the algorithm on the closure of a set of attributes we have:

$(BC)^+_F = BC$

$S = S \cup (BC)^+_F \cap ABC = BC \cup BC \cap ABC = BC$

$S = S \cup (BC\ ABD)^+_F \cap ABD = S \cup (B)^+_F \cap ABD$

by applying the algorithm on the closure of a set of attributes we have:

$(B)^+_F = B$

$S = S \cup (B)^+_F \cap ABD = BC \cup B \cap ABD = BC$

$S \subset Z$ then we stop

$Z = \mathbf{(C)^+_G = BC}$ i.e. $\mathbf{B \in (C)^+_G}$ so the dependency is preserved

Example 2

Sapienza Università di Roma
Dipartimento di Informatica

given the following schema:

$R = (A, B, C, D, E)$

and the following set of functional dependencies:

$F = \{ AB \rightarrow E, B \rightarrow CE, ED \rightarrow C\}$

say if the decomposition = { ABE, CDE }

preserves dependencies in F

Example 2: Development

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

$R = (A, B, C, D, E)$ $F = \{ AB \rightarrow E, B \rightarrow CE, ED \rightarrow C\} = \{ ABE, CDE \}$

let's check $B \rightarrow CE$

$Z = B$

$S = \varnothing$

loop on ABE and CDE:

$S = S \cup (B \cap ABE)^+_F \cap ABE = (B)^+_F \cap ABE = BCE \cap ABE = BE$

$S = BE \cup (B \cap CDE)^+_F \cap CDE = BE \cup (\varnothing)^+_F \cap CDE = BE$

intersection **first**! or we would eliminate B!

$BE \not\subset B$ $(S \not\subset Z)$ then **we enter the while loop**

Example 2: Development

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

R = (A, B, C, D, E) F = { AB $\rightarrow$ E, B $\rightarrow$ CE, ED $\rightarrow$ C} = { ABE, CDE }

BE $\not\subset$ B (S $\not\subset$ Z) then **we enter the while loop**

Z = Z $\cup$ S = B $\cup$ BE = BE

loop on ABE and CDE:

Intersection first!

S = BE $\cup$ (BE $\cap$ ABE)^+_F $\cap$ ABE = BE $\cup$ (BE)^+_F $\cap$ ABE = BE $\cup$ BCE $\cap$ ABE = BE

S = BE $\cup$ (BE $\cap$ CDE)^+_F $\cap$ CDE = BE $\cup$ (E)^+_F $\cap$ CDE= BE $\cup$ E $\cap$ CDE = BE

BE $\subseteq$ BE (S $\subseteq$ Z) then the algorithm terminates

Z = (B)^+_G = BE

E $\in$ (B)^+_G **but** C $\notin$ **(B)^+_G**

so, the dependency B $\rightarrow$ CE is not preserved (one of the attributes that should be functionally determined by B is missing in the closure)

Examples

let's go back to the examples we saw earlier, and check whether the algorithm would have detected the loss of some functional dependencies

Initial examples

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

$R = ABC$ with the set of functional dependencies: $F = \{A \rightarrow B, B \rightarrow C\}$

we decompose R into = { AB, AC }

let's start by seeing if it preserves AB

Z = A

S = $\varnothing$

loop on subschemas AB and AC:

$S = S \cup (A \cap AB)^{+}_{F} \cap AB = S \cup (A)^{+}_{F} \cap AB$

$S = S \cup (A)^{+}_{F} \cap AB = ABC \cap AB = AB$

$S = S \cup (A \cap AC)^{+}_{F} \cap AC = S \cup (A)^{+}_{F} \cap AC = AB \cup ABC \cap AC = AB \cup AC = ABC$

$ABC \not\subset A$ so, we should continue with another iteration but Z already contains R, so we stop as we already know that $\mathbf{(A)^{+}_{G} = R}$

and therefore $\mathbf{B \in (A)^{+}_{G}}$

Initial examples

SAPIENZA
Università di Roma
Dipartimento di Informatica

$R = ABC$ $F = \{ A \rightarrow B, A \rightarrow C \}$

let's check BC:

Z = B

$S = \varnothing$

loop on subchemas AB and AC:

$S = S \cup (B \cap AB)^+_F \cap AB = S \cup (B)^+_F \cap AB$

$S = S \cup (B)^+_F \cap AB = BC \cap AB = B$

$S = S \cup (B \cap AC)^+_F \cap AC = S \cup (\varnothing)^+_F \cap AC = B \cap AC = B$

B = B so, the algorithm terminates

$\mathbf{Z=(B)^+_G = B}$

and therefore $\mathbf{C \notin (B)^+_G}$

the algorithm confirms that the decomposition <u>does not preserve</u> F

Initial examples

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- let us consider the schema *R = (Matriculation, Town, Province)* with the set of functional dependencies

  F = {Matriculation → Town, Town → Province}
- let us decompose R *into R1* = {*(Matriculation, Town)*} and R2 = {*(Matriculation, Province)*}
- let us start by checking if the decomposition preserves *Matriculation → Town* but before we start let us replace long names with more "comfortable" letters:

- *R = ABC* with the set of functional dependencies *F = {A → B, B → C}*
- we decompose *R* into ρ = { AB, AC }

- we already verified that *Matriculation → Town (A → B)* is preserved, while *Town → Province* is not; so, the decomposition ρ **does not preserve *F***

Is it all ok?

- we emphasized that a "good" decomposition must have 3 properties:
  - each sub-scheme must be in 3NF
  - the decomposition must preserve all dependencies in F
  - the decomposition must allow to reconstruct a decomposed legal instance without loss of information (lossless join)

- we now know how to check if a decomposition preserves F

- we will now see how we can check if a decomposition has a lossless join