STUDIA INFORMATICA 2009
Volume 30 Number 2A (83)

Tomasz PIŁKA
Uniwersytet im. Adama Mickiewicza w Poznaniu, Wydział Matematyki i Informatyki
Tadeusz PANKOWSKI
Uniwersytet im. Adama Mickiewicza w Poznaniu, Wydział Matematyki i Informatyki
Politechnika Poznańska, Instytut Automatyki i Inżynierii Informatycznej

ZALEŻNOŚCI FUNKCYJNE W DANYCH XML

Streszczenie. W pracy omawiamy problem występowania i sprawdzania spełniania zależności funkcyjnych w danych XML. Zależności funkcyjne są jednym z głównych elementów zapewniających zachowanie spójności danych. Problem sprawdzania spełniania zależności funkcyjnych był z powodzeniem badany w przypadku danych relacyjnych, jednak w przypadku danych XML jest trudniejszy, gdyż tym razem należy również uwzględnić hierarchiczną strukturę danych. W artykule przedstawiamy metody badania spełniania zależności funkcyjnych w danych XML. Stosujemy przy tym reprezentacje schematu i zależności funkcyjnych w postaci formuł drzewiastych. Umożliwia to przeprowadzenie normalizacji schematu przez zwiększenie szans na poprawienie jakości danych w procesach ich integracji.

Słowa kluczowe: zależności funkcyjne XML, redundancja danych, postać normalna XML

FUNCTIONAL DEPENDENCIES IN XML DATA

Summary. In the work we discuss the problem of appearing and checking functional dependencies in XML data. Examining functional dependencies is one of the main factors witch provide data integrity. This problem was successfully examined for relational data. In the case of XML data we must also take the hierarchical structure of data into consideration.

Keywords: XML functional dependency, data redundancy, XML normal form

1. Wprowadzenie

Ostatnie lata stanowią wzrost popularności stosowania XML jako formatu przechowywania i wymiany danych w sieci. Wiele firm zaadoptowało XML, uznając go jako standard stosowany w procesach wymiany informacji. Również w systemach integracji danych pochodzących z różnych źródeł dane często wymieniane są w tym formacie. Ten niewątpliwy sukces niesie za sobą potrzebę opracowania metod, umożliwiających zapewnienie kontroli nad poprawnością danych. Problem ten jest tym istotniejszy, gdyż obecnie potencjalni klienci systemów wymiany czy integracji danych oczekują, że przesyłane do nich dane pozbawione będą zbędnych powtórzeń (problem redundancji danych [2-4], duplikatów), danych podobnych, a dane zostaną przesłane w krótkim czasie przy niewielkim nakładzie pracy.

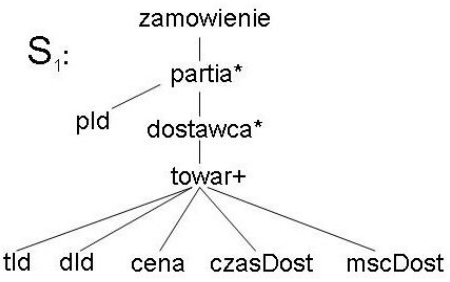
Istotne jest zatem wprowadzenie węzłów integralności (ang. *integrity constraints*) dla danych XML. W dotychczasowych pracach możemy znaleźć propozycję badania zachowania spójności dokumentów XML-owych z wykorzystaniem: kluczy [6], zależności pomiędzy ścieżkami [6], różnych form zależności funkcyjnych [8, 10]. Pracę nad badaniem zależności funkcyjnych dla danych relacyjnych prowadzone były już w latach 70. ubiegłego wieku [11, 1].

W pracy tej omawiamy problem występowania zależności funkcyjnych w danych XML oraz metody sprawdzania, czy w instancjach schematów zdefiniowane zależności funkcyjne są spełnione. Proponujemy, aby do sprawdzania spełniania zależności funkcyjnych zależności przedstawiać jako formuły drzewiaste. Układ dalszej części pracy jest następujący: w kolejnym rozdziale przedstawiamy przykład motywujący do prowadzenia badań. W trzecim paragrafie podajemy definicje stosowane do definiowania zależności funkcyjnych. Czwarty paragraf stanowi omówienie metod notacji i badania poprawności XML-owych zależności funkcyjnych. oraz wprowadzenie postaci normalnej dla XML. Pracę podsumowuje rozdział piąty.

2. Motywacja badań

Występowanie zależności funkcyjnych dla schematów danych XML często poprawia jakość danych występujących w instancjach tych schematów. Fakt ten nie gwarantuje jednak, że instancja takiego schematu będzie wolna od redundantnych danych, czy też będzie odporna na anomalię - aktualizację. W przypadku danych XML dodatkowym utrudnieniem podczas projektowania schematu danych jest hierarchiczna struktura danych.

Aby wyjaśnić ten problem, przyjrzyjmy się poniższemu przykładowi:

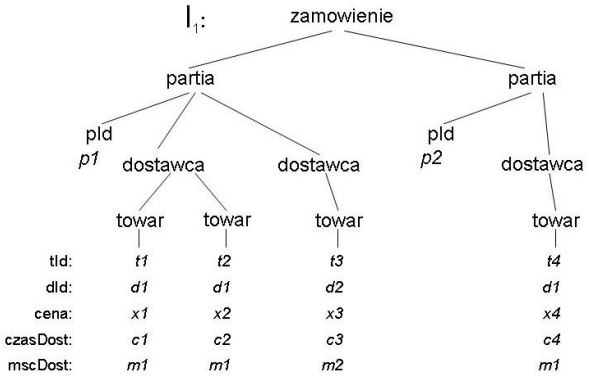


Rys. 1. Przykładowy schemat danych XML
Fig. 1. Sample XML schema tree

zamowienie	→	partia*
partia	→	pId dostawca*
pId	→	ε
dostawca	→	towar+
towar	→	dId tId cena czasDost msc Dost
dId	→	ε
tId	→	ε
cena	→	ε
czasDost	→	ε
mscDost	→	ε

Rys. 2. Definicja DTD
Fig. 2. DTD Definition

Schemat na rys. 1. oznaczony jako S_1 opisuje fragment bazy danych, w której przechowywane są informacje, dotyczące zamówień i dostawców oferujących pewne towary wchodzące w skład zamówień. Schemat ten jest zgodny z DTD umieszczonym na rys. 2, a jego przykładowa instancja została przedstawiona na rys. 3.



Rys. 3. Przykładowy instancja schematu S_1
Fig. 3. Sample instance of schema S_1

W dokumencie tym zamówienia składają się z partii, gdzie każdy element *partia* posiada własny identyfikator – *pId*. Dla każdej partii może istnieć zero bądź więcej dostawców, przy czym każdy dostawca musi zawierać przynajmniej jeden element *towar*. Każdy towar opisany jest przez następujący zestaw danych: identyfikator towaru, identyfikator dostawcy, cenę, czas i miejsce dostawy. Informacje te przechowywane są odpowiednio w elementach: *tId, dId, cena, czasDost, mscDost*.

Przyjmijmy, że w strukturze tej występują następujące więzy integralności:

- wszystkie elementy towar będące potomkami tego samego elementu dostawca muszą mieć taką samą wartość elementu tId,
- element mscDost zależy funkcyjnie od wartości identyfikatora danej partii (pId) oraz identyfikatora dostawcy (dId); oznacza to, że wszystkie towary danego dostawcy dostarczane w ramach tej samej partii muszą mieć to samo miejsce dostarczenia;

- wartość elementu mscDost determinuje wartość identyfikatora dostawcy (dId). Oznacza to, że jeżeli znamy miejsce dostawy, to dokładnie wiemy, który dostawca dostarczał tam towar, dopuszczamy tu, że jeden dostawca może dostarczać towary w wiele miejsc. Ilustruje to rys. 3, gdzie wartość miejsca dostarczenia towaru m1 jest przypisana do dostawcy o identyfikatorze d1.

Zaprezentowany powyżej zestaw danych zawiera redundancje danych. Zwróćmy uwagę, iż informację o dostawcy towarów (*dId*) trzymane są przy wszystkich towarach, które dany dostawca oferuje. Jeśli zestaw informacji o dostawcy rozszerzymy np. o jego dane adresowe – to dane te zostaną powtórzone przy każdym towarze tego dostawcy. Podobna sytuacja występuje z miejscem dostawy danego towaru (*mscDost*), jak wiemy (patrz uwagi powyżej) wszystkie towary dostawcy dostarczane w ramach jednej partii dostarczane są w to samo miejsce, stąd też informacja ta przechowywana przy towarach jest nadmiarowo powtarzana.

W tym zestawie danych występuje również problem wstawiania danych, załóżmy, że do danej partii zamówienia chcielibyśmy dodać nowy towar, nie możemy wykonać jednak tej operacji, dopóki nie będziemy posiadali informacji, jaki dostawca dostarcza ten towar.

W dalszej części pracy podajemy zbiór zależności funkcyjnych, które możemy zdefiniować dla powyższego zestawu danych. Oczywiście, istnienie zależności funkcyjnych oraz ich spełnianie przez instancje danego schematu nie gwarantuje, że dany dokument będzie wolny od redundancji, duplikatów danych czy też odporny na anomalię. Można jednak wykorzystać zależności funkcyjne do określenia postaci normalnej dla XML [4, 8].

3. Podstawowe definicje i oznaczenia

Rozdział ten zawiera podstawowe definicje stosowane w artykule. W dalszym ciągu, dla uproszczenia rozważań, przyjmujemy, że atrybuty w danych XML reprezentowane są za pomocą elementów. Zwróćmy uwagę, iż założenie to nie zawęża klasy dokumentów, które możemy rozważać. Zamiana atrybutów na elementy bądź odwrotnie nie stanowi trudności technicznej, a sam proces nie jest kosztowny obliczeniowo. Zakładamy także, że żaden element w schemacie nie ma definicji rekurencyjnej. Wówczas każdy schemat może być przedstawiony jako drzewo. Na rys. 1 podano reprezentację graficzną danych XML w postaci schematu *S1*, którego przykładowa instancja przedstawiona jest na rys. 3. Schematy danych XML najczęściej definiowane są z wykorzystaniem języków DTD bądź XSD [14], przykład takiej reprezentacji zawiera rys. 2. Alternatywną możliwością definiowania schematów jest wykorzystanie formuł drzewiastych (ang. *tree-pattern formula, TPF*) [13, 14], w taki też sposób schematy będą definiowane w tym paragrafie.

Jeżeli L jest zbiorem etykiet, to formułę drzewiastą możemy zdefiniować w następujący sposób [14]:

Definicja 1 [Formuła drzewiasta]: Formuła drzewiasta π nad zbiorem etykiet L jest wyrażeniem postaci:

$\pi ::= P[E] \mid \boldsymbol{\pi} /P[E]$

$E ::= x \mid C$

$C ::= x \mid \boldsymbol{\pi} \mid C \wedge C$

gdzie P jest ścieżką zdefiniowaną przez gramatykę: $P ::= /l \mid l \mid P/l$, gdzie: $/l$ oznacza zbiór potomków elementu typu l, P/l oznacza zbiór wierzchołków typu l, które są potomkami wierzchołka P.

Definicja 2 [Schemat TPF]: Niech L oznacza zbiór etykiet, a **x** wektor zmiennych tekstowych. Schematem TPF nad zbiorem L i wektorem **x** nazywamy wyrażanie:

$S ::= l\,[E]$

$E ::= /l \mid l[E] \mid E \wedge \ldots \wedge E$

gdzie $l \in L$, i $x \in$ **x**. Zbiór zmiennych, z zachowaniem ich porządku, występujących w schemacie S oznaczamy $S(\mathbf{x})$.

Przykład: Schematem TPF równoważny schematowi przedstawionemu na rys. 1 jest następujący schemat:

$S_1 = /zamowienie[partia[pId=x_1 \wedge dostawca[towar[tId=x_2 \wedge dId=x_3 \wedge cena=x_4 \wedge czasDost=x_5 \wedge mscDost=x_6]]]]$.

Zwróćmy uwagę, że schemat zdefiniowany powyżej posiada następujące własności:

- zachowana jest struktura drzewiasta danych XML,
- zmienne trzymane są z informacją o ścieżkach, które do nich prowadzą,
- wyrażenia TPF są poprawnymi predykatami języka XPath.

Definicja 3 [Typ zmiennej w TPF]: Niech S będzie schematem TPF nad zbiorem zmiennych **x** i niech atom $l = x$ występuje w S. Wówczas ścieżkę P rozpoczynającą się w korzeniu drzewa i kończącą w l określamy jako typ zmiennej x, i oznaczamy $type_S(x) = p$.

Typem zmiennej x_1 w schemacie S_1 jest: $type_{S_1}(x_1) = /zamowienie/partia/pId$.

Drzewa XML

Dane XML możemy zdefiniować jako nieuporządkowane, nieetykietowane drzewo (drzewo XML) nad skończonym zbiorem etykiet *L*, zbiorem wartości tekstowych *Str* ∪ {⊥}, gdzie symbol ⊥ oznacza wartość *null*.

Definicja 4 [Drzewo XML]

Drzewem XML ***I*** nazywamy układ $I = (r, N^e, N^t, child, \lambda, v)$, gdzie**:**

- r jest wierzchołkiem wyróżnionym – korzeniem drzewa, N e jest skończonym zbiorem wierzchołków elementowych, a N t jest skończonym zbiorem wierzchołków tekstowych;
- $child \subseteq (\{r\} \cup N^{\varepsilon}) \times (N^{\varepsilon} \cup N^{t}))$ – relacja wprowadzająca strukturę drzewiastą do zbioru $\{r\} \cup N^e \cup N^t$, gdzie *r* jest korzeniem drzewa, a każdy wierzchołek elementowy ma przynajmniej jednego potomka (który również jest albo wierzchołkiem elementowym, albo wierzchołkiem tekstowym), wierzchołki tekstowe są liśćmi;
- $\lambda : N^e \to L$ – funkcja etykietującą wierzchołki elementowe ich nazwami (etykietami);
- $v : N^t \to Str \cup \{\perp\}$ – funkcja etykietująca wierzchołki tekstowe wartościami ze zbioru *Str* bądź wartością ⊥.

Dane XML, które reprezentują konkretną reprezentację schematu, wygodnie jest rozpatrywać jako drzewo *I* o schemacie *S* nad zbiorem zmiennych **x** jako parę (*S*, Ω), gdzie *S* jest schematem TPF, a *Ω* jest zbiorem wartościowań zmiennych w **x**. Wartościowanie $\omega \in \Omega$ jest funkcją przydzielającą zmiennym z **x** wartości z $Str \cup \{\perp\}$, tzn. ω: $\mathbf{x} \to Str \cup \{\perp\}$. Parę (*S*, Ω) nazywamy opisem danych XML.

Instancję danych XML I_1 umieszczoną na rys. 3 można zatem przedstawić za pomocą następującego opisu:

I1 : = (S1(x1,x2,x3,x4,x5,x6), {(p1,t1,d1,x1,c1,m1), (p1,t2,d1,x2,c2,m1), (p1,t3,d2,x3,c3,m2), (p2,t4,d1,x4,c4,m1)}).

Drzewo XML *I* spełnia opis (*S*, Ω), co oznaczamy jako I |= (*S*, Ω), jeśli *I* spełnia (*S*, *ω*) dla każdego wartościowania $\omega \in \Omega$. Należy tu podkreślić, że opis (*S*, Ω) reprezentuje pewną klasę instancji schematu *S* dla tego samego zbioru wartościowań Ω . Zatem, ten sam zestaw danych może być przedstawiony w różnych schematach, o różnych opisach, ale o tym samym zbiorze wartościowań. Spowodowane jest to przez elastyczność danych XML i ich hierarchiczną strukturę, która umożliwia przegrupowywanie elementów i zagnieżdżanie ich na różne sposoby.

4. Zależności funkcyjne w danych XML

Definicję zależności funkcyjnych dla danych relacyjnych można znaleźć w każdym podręczniku do baz danych [11, 1], ma ona następującą postać:

Niech dana będzie relacja $R(U)$, i niech $X, Y \subseteq U$. Zależnością funkcyjną nazywamy każdy napis o postaci:

$$X \rightarrow Y$$

Mówimy wówczas, że Y zależy funkcyjnie od X lub że X determinuje funkcyjnie Y.

Mówimy, że w R spełniona jest zależność funkcyjna $X \rightarrow Y$ (oznaczamy to jako $R \models X \rightarrow Y$), jeśli dla każdych dwóch krotek r_1 i $r_2 \in R$ zachodzi:

$$r_1[X] = r_2[X] \Rightarrow r_1[Y] = r_2[Y]$$

Definicja ta określona jest dla płaskich, tabularycznych danych. Nie ma możliwości przeniesienia tej definicji w bezpośredniej postaci dla danych XML, gdyż w przypadku XML musimy uwzględnić również hierarchiczność danych.

4.1. Notacja XML-owych zależności funkcyjnych

Definicje zależności funkcyjnych dla XML korzystają z klasycznej definicji zależności funkcyjnych dla danych relacyjnych. Uwzględnienie struktury danych wymusiło wprowadzenie pewnych modyfikacji. W literaturze wyróżniamy dwa główne nurty badań prowadzonych nad definiowaniem zależności funkcyjnych dla XML. W pierwszej [2-5] definicja zależności funkcyjnych oparta jest na notacji poddrzew budowanych z wykorzystaniem ścieżek (ang. *tree tuple*). W pracach tych Arenas i Libkin proponują, aby zależności funkcyjne definiować w następującej postaci:

(*zamowienie.partia.@pId*, *zamowienie.partia.dostawca.towar.@dId* →

zamowienie.partia.dostawca)

Drugie podejście do definiowania zależności funkcyjnych dla XML możemy znaleźć w pracach Bunemanna [6]. Zależności funkcyjne definiowane są z wykorzystaniem kluczy opartych na ścieżkach. Podana wcześniej zależność funkcyjna w tej notacji ma postać:

(*zamowienie.partia*,{*pId* })

(*zamowienie.partia*,(*dostawca*{*towar*/*dId* }))

W naszych rozważaniach zależności funkcyjne definiujemy z wykorzystaniem TPF. Przyjmujemy następującą definicję:

Definicja 5 [Zależność funkcyjna XML]: Zależnością funkcyjną XML (*XFD*) nad zbiorem etykiet L i zbiorem zmiennych **x** nazywamy wyrażenie o następującej składni:

$f ::= /P_1[C_1]/.../P_n\,[C_n]$,

$P ::= l \mid P/l,$

$C ::= TRUE \mid P = x \mid C_1 \wedge \ldots \wedge C_n$

gdzie $l \in L$, a x jest zmienną w **x**. Zależność funkcyjną możemy więc zapisywać jako wyrażenie o postaci $f(\mathbf{x})$.

Przykładem zależności funkcyjnej dla rozważanego powyżej przykładu jest:

$$f_5(x_1,x_2) = /zamowienie/partia[pId = x_1]/dostawca[towar/dId = x_2]$$

Zależności funkcyjne zgodne z powyższą definicją są wyrażeniami XPath, które dla danego wartościowania zmiennych zwracają sekwencję obiektów. Obiektami zwracanymi w wyniku obliczania zależności funkcyjnych mogą być zarówno wierzchołki drzewa XML, jak i wartości tekstowe. Umożliwia nam to określenie spełniania zależności funkcyjnych, a mianowicie: instancja $I = (S, \Omega)$, spełnia XML-ową zależność funkcyjną $f(x_1,\ldots x_k)$, jeżeli dla każdego wartościowania zmiennych, $\omega \in \Omega$, funkcja $f(x_1,\ldots x_k)$ przy wartościowaniu ω zwraca singleton, tzn. sekwencję złożoną z jednego elementu.

4.2. Spełnianie zależności funkcyjnych

Można zatem wprowadzić następującą definicję spełniania zależności funkcyjnych XML, dla danych, dla których schemat jest TPF.

Definicja 6 [Spełnianie zależności funkcyjnych]: Niech I będzie instancją schematu TPF $S(\mathbf{x})$, a f zależnością funkcyjną XML zdefiniowaną nad S. Instancja danych XML I spełnia zależność funkcyjną f (oznaczamy to jako: $I \models f$), jeżeli dla każdego wartościowania $\boldsymbol{\omega}$ zmiennych w **x** zachodzi następująca implikacja:

$$I \models (S, \omega) \Rightarrow count([\![f(\omega)]\!]) \leq 1$$

gdzie $[\![f(\omega)]\!]$ jest wynikiem f obliczonym przy wartościowaniu ω.

Możemy zdefiniować następujące zależności funkcyjne dla schematu S_1.

$$f_1(x) = /zamowienie/partia[dostawca/towar/tId = x]$$
$$f_2(x) = /zamowienie/partia[dostawca/towar/tId = x]/pId$$
$$f_3(x_1,x_2) = /zamowienie/partia[pId = x_1]/dostawca/towar[dId = x_2]/mscDost$$
$$f_4(x) = /zamowienie/partia/dostawca/towar[mscDost = x]/dId$$
$$f_5(x_1,x_2) = /zamowienie/partia[pId = x_1]/dostawca[towar/dId = x_2]$$

Prześledźmy, w jaki sposób obliczana jest wartość wyrażenia $f_1(x)(\omega)$ dla instancji I_1 umieszczonej na rys. 3: w pierwszym kroku wybieramy zbiór wierzchołków typu */zamowienie/partia* (zgodnie z wcześniejszą definicją), dalej dla każdego wybranego w pierwszym kroku wierzchołka testujemy, czy jest spełniony predykat [*dostawca/towar/tId* = *x*]. Wyrażenie to jest spełnione w wierzchołku *k*, jeżeli w instancji I_1

istnieje co najwyżej jedna ścieżka typu *dostawca*/*towar*/*tId* prowadząca z wierzchołka *k* do elementu tekstowego o wartości ω(*x*). Wartość wyrażenia *count*($[\![f_1(x)(\omega)]\!]$) jest równa 1 dla wszystkich czterech wartościowań dostępnych w instancji I_1. Należy tu podkreślić, że obliczenia te prowadzone są zgodnie z semantyką języka XPath.

W podobny sposób można sprawdzić poprawność pozostałych powyższych zależności funkcyjnych. Zwróćmy jednak uwagę, że dla instancji I_1 można zaproponować zależności funkcyjne, które nie będą spełnione. Przykłady takich zależności zostały umieszczone poniżej.

$g_1(x)$ = */zamowienie/partia[dostawca/towar/dId = x]*

$g_2(x)$ = */zamowienie/partia[pId = x]/dostawca/towar/dId*

$g_3(x_1,x_2)$ = */zamowienie/partia[pId = x_1]/dostawca/towar[dId = x_2]*

Prowadząc rozumowanie przedstawione powyżej, można łatwo policzyć, że dla tych zależności funkcyjnych otrzymamy:

count($[\![g_1(x)(x\text{->}d1)]\!]$) = 2

count($[\![g_2(x)(x\text{->}d1)]\!]$) = 2

count($[\![g_3(x_1,x_2)(x_1\text{->}p1, x_2\text{->}d1)]\!]$) = 2

4.3. Postać normalna dla XML

Jedną z powszechniej stosowanych metod eliminowania redundancji w dokumentach XML jest przeprowadzanie procesu normalizacji. W dotychczasowych pracach zaproponowano kilka podejść definiowania postaci normalnej dla XML [2, 4, 9]. Do definicji postaci normalnej dla XML przedstawionej w tym rozdziale wykorzystujemy formuły drzewiaste oraz definicje zależności funkcyjnych przedstawione w poprzednim podpunkcie, w definicji tej wykorzystujemy założenia zaprezentowane w pracy [4].

Przyjmijmy, że przez (*S*, *F*) będziemy oznaczać schemat TPF *S* wraz ze zbiorem zależności funkcyjnych zdefiniowanych nad *S*. Domknięcie (*S*, *F*) oznaczmy przez $(S, F)^+$, jest nim schemat TPF *S* wraz ze zbiorem zależności funkcyjnych, które mogą być wyprowadzone z (*S*, *F*).

Definicja 7 [Postać normalna XML]: Niech *S* będzie schematem TPF, a *F* zbiorem zależności funkcyjnych nad *S*. (*S*, *F*) jest w postaci normalnej XML (*XNF*) wtedy i tylko wtedy, gdy dla każdej zależności funkcyjnej $f/l \in (S, F)^+$ zachodzi również zależność $f \in (S, F)_+$, czyli

(S, F) jest w XNF $\Leftrightarrow$ $(f/l \in (\mathrm{S}, \mathrm{F})^+ \Rightarrow f \in (\mathrm{S}, \mathrm{F})^+)$

Zależność ta rozumiana jest w następujący sposób: przyjmijmy, że $f(x_1,\ldots x_k)/l$ jest zależnością funkcyjną, a *I* jest instancją danych schematu *S*. W instancji *I* spełniona jest zależność funkcyjna $f(x_1,\ldots, x_k)/l$, jeżeli dla każdego wartościowania ω układu zmiennych $(x_1,\ldots, x_k)$ znajduje się co najwyżej jedna wartość typu *type*(*f*/*l*). Zatem, dla każdego

wartościowania $(x_1,\ldots, x_k)$ powinniśmy mieć tylko jedną wartość *f*/*l*, czyli może być tylko jedno poddrzewo typu *type*(*f*) oznaczone przez wyrażenie $f(x_1,\ldots, x_k)$. Zauważmy, że spełnienie warunku XNF pozwala na zniwelowanie problemu redundancji danych.

Przyjrzyjmy się podanej wcześniej zależności funkcyjnej $f_4(x)$ podanej dla instancji *I* schematu S_1:

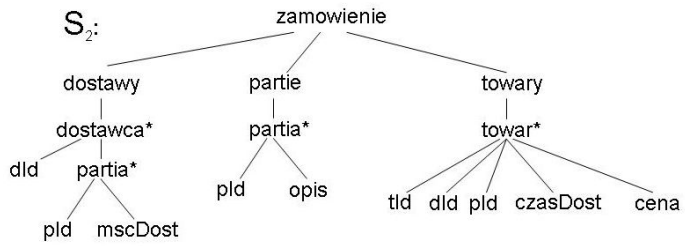
f4(x) = /zamowienie/partia/dostawca/towar[mscDost = x]/dId

zależność ta określa, że miejsce dostawy determinuje dostawcę, dla danych relacyjnych miałaby zatem postać: *mscDost* → *dId*. Jak pokazano wcześniej, zależność ta jest spełniona w instancji *I*, schemat S_1 nie jest jednak w XNF, gdyż nie zachodzi w nim zależność:

$f_{4'}(x)$ = */zamowienie/partia/dostawca/towar[mscDost = x]*

jeśli rozważymy wartościowanie ω, w którym wartości *x* będzie równa *m1* (ω=[x →"*m1*"]), to otrzymamy dwa różne elementy *dostawa* (będące potomkami różnych elementów *partia*) o wartości *mscDost* równej *m1*. Oznacza to, że instancja *I* schematu S_1 nie jest pozbawiona redundancji, nie jest zatem w postaci normalnej dla XML.

Aby doprowadzić schemat S_1 do XNF, musimy dokonać jego transformacji. Chcąc jednak, aby nowy schemat był w postaci normalnej dla XML, musimy dokonać reorganizacji schematu wyjściowego, możemy tego dokonać dzięki hierarchiczności danych XML. W efekcie otrzymamy schemat S_2 umieszczony na rys. 4.



Rys. 4. Schemat S_1 po transformacji do XNF
Fig. 4. Schema S_1 in XNF

Dla powyższego schematu można zdefiniować następujący zbiór zależności funkcyjnych:

$f_1(x)$ = /*zamowienie*/*dostawy*/*dostawca*[d*Id* = *x*]

$f_2(x_1,x_2)$ = /*zamowienie*/*dostawy*/*dostawca*[*dId* = x_1]/*partia*[*pId* = x_2]/*mscDost*

$f_3(x)$ = /*zamowienie*/*dostawy*/*dostawca*[*partia*[*mscDost* = *x*]/*dId*

$f_4(x)$ = /*zamowienie*/*partie*/*partia*[*pId* = *x*]/*opis*

$f_5(x_1,x_2)$ = /*zamowienie*/*towary*/*towar*[*dId* = x_1 ∧ *pId* = x_2]/*tId*

$f_6(x)$ = /*zamowienie*/*towary*/*towar*[*pId* = *x*]/cena

$f_7(x)$ = /*zamowienie*/*towary*/*towar*[*pId* = *x*]/czasDost

oraz zbiór zależności funkcyjnych, które mogą być z nich wyprowadzone:

$f'_2(x_1,x_2)$ = /*zamowienie*/*dostawy*/*dostawca*[*dId* = x_1]/*partia*[*pId* = x_2]

$$f'_3(x) = /zamowienie/dostawy/dostawca[partia[mscDost = x]$$
$$f'_4(x) = /zamowienie/partie/partia[pId = x]$$
$$f'_5(x_1,x_2) = /zamowienie/towary/towar[dId = x_1 \wedge pId = x_2]$$
$$f'_6(x) = /zamowienie/towary/towar[pId = x]$$

Dla schematu S_2 można zbudować instancję danych I_2. Dla instancji tej łatwo można sprawdzić poprawność i spełnianie powyższych zależności, co oznacza, że jest w XNF. Zatem, schemat ten jest pozbawiony redundancji, a zarazem zachowuje zależności funkcyjne. Spełnione są wiec główne założenia procesu normalizacji danych.

5. Podsumowanie

W pracy tej omówiliśmy problem występowania i spełniania zależności funkcyjnych w danych XML. Zaproponowaliśmy metodę definiowania i sprawdzania spełniania zależności funkcyjnych w danych XML. W zaprezentowanych rozważaniach wykorzystaliśmy notację TPF, która korzysta z semantyki języka XPath [15]. Definicja XNF oparta jest na założeniach przedstawionych w pracy [2].

Zależności funkcyjne są jednym z głównych mechanizmów zapewniania spójności w danych XML. Wzrost znaczenia danych XML oraz powszechność ich stosowania w wielu dziedzinach powoduje konieczność opracowania efektywnych metod, pozwalających na zapewnienie poprawności danych. Problem ten jest szczególnie istotny w systemach integracji danych pochodzących z różnych źródeł. Dalsze badania dotyczyć będą konstruowania efektywnych algorytmów sprawdzania spełniania zależności funkcyjnych w danych XML oraz określania postaci normalnych dla schematów XML. Hierarchiczność danych XML umożliwia dokonywanie rekonstrukcji drzew, pozwalającej na przeprowadzenie procesu normalizacji, w którym wynikowy schemat danych odporny będzie na problem redundancji danych z jednoczesnym zachowaniem zależności funkcyjnych.

Praca częściowo realizowana w ramach projektu nr 3695 finansowanego przez Ministerstwo Nauki i Szkolnictwa Wyższego.

BIBLIOGRAFIA

1. Abiteboul S., Hull R., Vianu V.: Foundations of Databases, Reading, Massachusetts, Addison-Wesley, 1995.
2. Arenas M.: Normalization theory for XML. SIGMOD Record, vol. 35 No. 4, 2006, s. 57÷64.

3. Arenas M., Libkin L.: XML Data Exchange: Consistency and Query Answering. PODS Conference, 2005, s. 13÷24.
4. Arenas M., Libkin L.: A normal form for XML documents. ACM Trans. Database Syst., Vol. 29, 2004, s. 195÷232.
5. Arenas M., Libkin L.: An information-theoretic approach to normal forms for relational and XML data. J.ACM, Vol. 52, No. 2, 2005, s. 246÷283.
6. Buneman P., Davidson S. B., Fan W., Hara C. S., Tan W. C.: Reasoning about keys for XML. Information Systems, Vol. 28, No. 8, 2003, s. 1037÷1063.
7. Jixue L., Millist V., Chengfei L.: Functional Dependencies, from Relational to XML. ser. Lecture Notes in Computer Science, Vol. 2890, 2003, s. 1063÷1079.
8. Kolahi S.: Dependency-Preserving Normalization of Relational and XML Data. DBPL, ser. Lecture Notes in Computer Science, G. M. Bierman and C. Koch, Eds., Vol. 3374, Springer, 2005, s. 247÷261.
9. Kolahi S.: Dependency-preserving normalization of relational and XML data. Journal of Computer and Systems Sciences, Vol. 73, No. 4, 2007, s. 636÷647.
10. Libkin L.: Normalization theory for XML. ser. Lecture Notes in Computer Science, Vol. 4704, 2007, s. 1÷13.
11. Millist V., Jixue L.: Checking Functional Dependency Satisfaction in XML. ser. Lecture Notes in Computer Science, Vol. 3671, 2005, s. 4÷17.
12. Pankowski T.: Podstawy baz danych. Wydawnictwo Naukowe PWN, Warszawa, 1992.
13. Pankowski T.: XML data integration in SixP2P: a theoretical framework. EDBT Workshop on Data Management in P2P Systems DaMaP 2008, ser. ACM International Conference Proceeding Series, Doucet A., Garncarski E., Pacitti E., ACM, 2008, s. 11÷18.
14. Pankowski T., Cybulka J., Meissner A.: XML Schema Mappings in the Presence of Key Constraints and Value Dependencies. ICDT 2007 Workshop EROW, CEUR Workshop Proceedings. CEUR-WS.org, Vol. 229, 2007, s. 1÷15.
15. XML Path Language (XPath) 2.0, www.w3.org/TR/xpath20, 2006.

Recenzent: Dr inż. Piotr Bajerski

Wpłynęło do Redakcji 3 marca 2009 r.

Abstract

In the paper we discuss the problem of defining XML functional dependencies (XFDs) and designing XML normal form (XNF) for XML data. These problems are similar to their counterparts in relational model. The main task in relational schema normalization is producing such a set of schemas that posses the required form, usually 3NF or BCNF. The normalization process consists in decomposition of a given input relational schema. In turns out that in some cases elimination of redundancy implies the lost of functional dependencies. This problem was also investigated for XML data. Thanks to the hierarchical structure of XML data, it is possible to achieve both – elimination of redundancies and preservation of all XNFs.

In this paper we discuss XML functional dependencies and XML normal forms using so-called tree-pattern formulas (TPFs). We use these formulas to express both the structure of XML data (XML schemas) and XFDs. TPFs form in fact a class of XPath predicates, so their semantics is defined precisely within the XPath recommendation. We use this notation to define the satisfaction of a given XFD by an XML instance. This is used do define conditions for XNF. The discussion is illustrated be an running example. This theory is the basis for developing algorithms for checking satisfactions of XFDs by XML instances, and for normalizing XML schemas. These algorithms are currently under development.

Adresy

Tomasz PIŁKA: Uniwersytet im. Adama Mickiewicza, Wydział Matematyki i Informatyki, ul. Umultowska 87, 61-614 Poznań, Polska, tomasz.pilka@amu.edu.pl.
Tadeusz PANKOWSKI: Uniwersytet im. Adama Mickiewicza, Wydział Matematyki i Informatyki, ul. Umultowska 87, 61-614 Poznań, Polska, tadeusz.pankowski@amu.edu.pl; Politechnika Poznańska, Instytut Automatyki i Inżynierii Informatycznej, Pl. M. S-Curie 5, 60-965 Poznań, Polska, tadeusz.pankowski@put.poznan.pl.