XPath 패턴들간의 준동형 정보를 효율적으로 유지하기 위한 래티스 구조

(A Lattice Structure for Efficiently Maintaining Homomorphism Information Among XPath Patterns)

유 상 현 † 손 진 현 †† 김 명 호 †††
(Sanghyun Yoo) (Jin Hyun Son) (Myoung Ho Kim)

요 약 많은 XML 응용들은 XML 문서에 대한 질의 언어로 XPath 패턴을 사용한다. XPath 패턴들 사이에는 포함 관계가 존재할 수 있으며, 하나의 XPath 패턴이 다른 XPath 패턴을 포함하는지를 결정하는 문제를 포함 문제라고 한다. 포함 문제는 많은 응용들에서 발생하고 있지만 co-NP complete 문제로 알려져 있다. 한편 XPath 패턴들 사이의 준동형 관계는 포함 관계의 충분 조건이면서 다항 시간에 얻을 수 있다. 본 논문에서는 준동형 문제가 포함 문제를 대체하여 유용하게 쓰일 수 있는 응용들에 대해 논의하고, XPath 패턴들 사이의 준동형 정보를 유지하면 많은 이점을 얻을 수 있다는 사실에 대해 논의한다. 그리고 XPath 패턴들 사이의 준동형 관계를 유지하기 위하여 POX(Partially Ordered Set of XPath Patterns)라는 래티스 구조를 제안하고, 그것을 유지할 수 있는 알고리즘을 개발한다. 알고리즘 분석을 보면 알 수 있듯이, 본 논문에서 제안하는 알고리즘은 다항 시간에 POX를 효율적으로 유지할 수 있다.

키워드 : XML, XPath 패턴, 포함, 준동형

Abstract Many XML applications use XPath patterns as a query language for XML documents. Two XPath patterns may have containment relationship, and the containment problem between two XPath patterns is a problem that determines whether one XPath pattern contains another XPath pattern. Although the containment problem occurs in many applications, it is known as a co-NP complete. A homomorphism problem, which is a sufficient condition for the containment problem, is solved in polynomial time. We first discuss applications that replace the containment problem with the homomorphism problem, and maintaining homomorphism information among XPath patterns will benefit those applications. Then, we propose a lattice structure, called POX (Partially Ordered Set of XPath Patterns), and develop algorithms for maintaining it. As our analyses show, the algorithms can efficiently maintain POX in polynomial time.

Key words : XML, XPath pattern, Containment, Homomorphism

1. 서 론

XPath[1]는 XSLT[2]와 XPointer[3]에서 사용하기 위해 고안된 언어로, XML 문서를 탐색하고 엘리먼트 노드(element node)들의 집합을 선택하거나, 주어진 조건을 만족하는 XML 문서를 찾아내는 데에 사용된다. 하지만 많은 XML 응용들은 XPath 문법 가운데에서 중요한 요소만을 취한 XPath 단편(XPath fragment)만을 지원한다. XPath 단편이란 노드 테스트(node test)와 자식 축(child axis : "/"), 자손 축(descendant axis : "//"), 와일드카드(wildcard : "*"), 그리고 서술어(predicate : "[...]")만으로 구성되는, XPath의 부분 집합을 의미한다[4]. XPath 단편은 트리 패턴(tree pattern)으로 나타낼 수 있으며, 트리 패턴의 각 노드는 XML 문서에 대한 조건을 명시한다. 문서가 트리 패턴에 나타난 모든 조건을 만족하면 결과는 참(true)이 되고, 하나라도 만족하지 못하면 거짓(false)이 된다.

임의의 두 XPath 트리 패턴 p와 p'이 있을 때, p'의 모든 조건을 만족하는 임의의 XML 문서가 항상 p의

· 본 연구는 대학 IT연구센터 육성 지원사업의 부분적인 지원과 한국과학재단 목적기초연구(R08-2003-000-10464-0) 지원을 받아 수행되었음.
† 비 회 원 : 한국과학기술원 전산학과
shyoo@dbserver.kaist.ac.kr
†† 종신회원 : 한양대학교 컴퓨터공학과 교수
jhson@cse.hanyang.ac.kr
††† 종신회원 : 한국과학기술원 전산학과 교수
mhkim@dbserver.kaist.ac.kr
논문접수 : 2004년 2월 4일
심사완료 : 2005년 1월 19일

모든 조건을 만족한다면, "p는 p'을 포함한다(p contains p')"라고 하고, $p' \sqsubseteq p$으로 표기한다[4,5]. 이렇듯 두 개의 패턴이 있을 때 하나의 패턴이 다른 하나의 패턴을 포함하는지의 여부를 알아내는 것을 포함 문제(containment problem)라고 한다. 포함 문제는 많은 XML 애플리케이션에서 빈번히 발생하고 있지만, co-NP complete 문제로 알려져 있다[4].

한편, [4]에 정의된 XPath 트리 패턴들 사이의 준동형(homomorphism)은 포함 관계의 충분 조건으로써, 다항 시간에 찾을 수 있다. 본 논문에서는 하나의 패턴 p에서 다른 패턴 p'으로의 준동형이 존재할 때, p를 정의역 패턴(domain pattern), p'를 공변역 패턴(codomain pattern)이라고 정의한다. 정의역 패턴은 항상 공변역 패턴을 포함한다.

본 논문은 많은 응용들이 co-NP complete 문제인 포함 문제를 다항 시간 문제인 준동형 문제로 대체할 수 있음을 찾아냈다. 뿐만 아니라 패턴들 사이의 준동형 관계를 유지할 필요성을 발견하고, POX(partially ordered set of XPath patterns)와 inPOX(index for POX) 트리와 같은 자료 구조를 이용하여 패턴들 사이의 준동형 정보를 저장하고 유지할 수 있는 효율적인 알고리즘을 제안하고 있다.

본 논문의 구성은 다음과 같다. 2장에서 관련된 연구 및 동기를 소개하고 3장에서 본 연구가 해결하고자 하는 문제에 대한 정의를 내린 뒤, 4장에서는 알고리즘에서 사용할 주요 개념들에 대한 정의를 내린다. 5장에서 본 논문에서 제시하는 알고리즘에 대해서 설명한다. 6장에서는 알고리즘의 분석을 하고, 마지막으로 7장에서 결론을 맺고 추후 연구 방향을 제시한다.

2. 관련 연구 및 동기

SIENA[6]는 다수의 분산된 클라이언트(client)들에게 선택적으로 정보를 배포할 수 있는 publish/subscribe 시스템이다. 네트워크(network)로 연결된 클라이언트들은 시스템에 정보를 제공하는 publisher나, 혹은 그 정보를 받아 사용하는 subscriber로써 통신에 참여할 수 있으며, 브로커(broker)는 새로운 정보가 들어왔을 때 클라이언트가 등록한 subscription과 일치하는가를 판단하여 적당한 클라이언트들에게 그 정보를 전달한다. 확장성을 위해서 다수의 브로커가 네트워크로 연결된 경우에, 하나의 클라이언트가 subscription을 브로커에게 등록하면, 각 브로커는 새로 등록된 subscription을 인접한 다른 브로커들에게 전달(propagation)하여야 한다.

그림 1은 publish/subscribe 시스템에서 subscription을 전달하는 과정을 보여준다. 그림 1의 (a)는 B_5에 의해 subscription s_1이 나머지 브로커로 전달되는 모습을

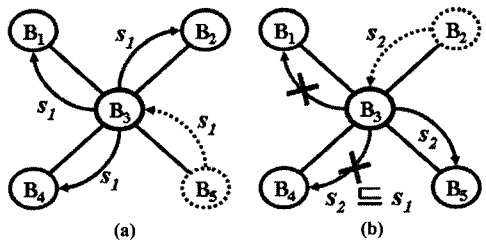

그림 1 Publish/subscribe 시스템의 subscription 전달

나타내고 있다. 이 상황에서, 그림 1의 (b)와 같이 B_2가 $s_2 \sqsubseteq s_1$인 subscription s_2를 전달할 경우, B_3은 B_1과 B_4에게는 s_2를 전달하지 않고 B_5에게만 s_2를 전달할 것이다. s_2를 만족하는 문서의 집합은 항상 s_1을 만족하는 문서 집합의 부분 집합이기 때문이다. 만일 B_1과 B_4에 다른 많은 브로커들이 맞물려 있다면 그 브로커들은 모두 s_2의 전달 대상에서 제외되며, 이는 상당히 많은 네트워크 비용을 줄일 수 있음을 의미한다.

최근 XML이 인터넷에서 문서 교환을 위한 표준으로 등장하게 되면서, XML 문서를 전달하는 publish/subscribe 시스템들이 많이 등장하게 되었다[7-9]. 이때 위에서 설명한 라우팅 개념을 사용하기 위해서는 그 subscription 언어인 XPath 패턴들 사이의 포함 관계를 구해야 하는, 포함 문제가 발생하게 된다. 하지만 이미 밝혔듯 XPath 패턴들간의 포함 관계를 구하는 문제는 co-NP complete 문제이므로, 포함 관계에 기반하여 subscription의 전달 여부를 결정하는 데에는 상당히 많은 시간이 소요된다.

한편, 준동형 관계는 곧 포함 관계를 의미하므로, 새로운 패턴이 subscription으로 등록된 경우, 기존에 등록된 패턴과의 준동형 관계에 기반하여 전달 여부를 결정할 수 있다. 포함 관계가 존재함에도 불구하고 준동형 관계가 존재하지 않는 경우에는, 시스템은 포함 관계가 존재하지 않는다고 판단하고 새로운 패턴을 전달하게 될 것이다. 그러나 이 경우에는 불필요한 메시지가 발생할 뿐이지 시스템의 잘못된 동작을 야기하지는 않는다.

3. 문제 정의

준동형 관계는 부분 순서(partial order)이다[10]. XP를 XPath 패턴들의 집합이라고 하자. 이진 관계(binary relation) $\sqsubseteq_h$를 두 패턴 사이의 준동형을 나타낸다고 하면 부분 순서 집합 $<XP, \sqsubseteq_h>$를 생각할 수 있다. 모든 패턴들의 정의역 패턴을 의미하는 가상의 패턴 TOP를 최소 상계(least upper bound)로, 모든 패턴들의 공변역 패턴을 의미하는 가상의 패턴 BTM을 최대 하계(greatest lower bound)로 $<XP, \sqsubseteq_h>$에 추가하면, $<XP, \sqsubseteq_h>$는 래티스(lattice) 구조가 된다. 본 논문에서

는 이렇게 구성한 래티스를 *POX*(partially ordered set of XPath patterns)라고 정의한다. 그림 2는 POX를 나타낸다.

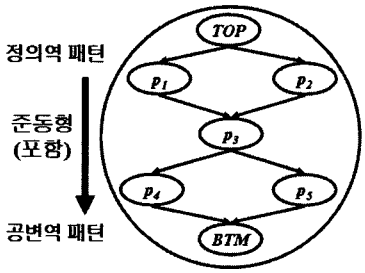

그림 2 POX

준동형 문제가 발생하는 많은 XML 응용에서 POX를 이용하여 준동형 정보를 유지한다면 다음과 같은 이점이 있다. 우선, 유지하고 있는 패턴간의 준동형 정보가 필요한 경우에, 매번 모든 패턴들에 대해서 준동형 정보를 계산하지 않고 바로 정보를 얻을 수 있다. 또한 새로운 패턴이 들어왔을 때에도 준동형 정보를 유지하지 않는 경우에 비해서 다른 패턴들과의 준동형 정보를 빠르게 계산할 수 있다. 마지막으로 문서의 필터링을 보다 효율적으로 할 수 있다. XML 문서가 들어왔을 때, 시스템이 *TOP*부터 시작해서 그것의 공변역 패턴으로 이동하면서 문서를 필터링한다면, 문서가 비교한 패턴을 만족하지 않는 경우 그 패턴의 모든 공변역 패턴들에도 만족할 수 없으므로 비교 대상을 줄일 수 있다.

POX를 준동형 정보를 찾는데 이용하기 위해서는 POX를 항상 최신의 정보로 유지할 수 있어야 한다. 새로운 XPath 패턴 p_{new}가 추가되었을 때, POX의 유지는 (i) 준동형에 대해서 p_{new}의 정의역 패턴 집합 P를 구하는 단계와 (ii) 공변역 패턴 집합 P'을 구하는 단계, 그리고 (iii) $p_i \in P$에서 p_{new}로 에지를 연결하고, p_{new}에서 $p_j' \in P'$로 에지를 연결하는 단계로 이루어진다. 이 경우에, 새로운 패턴 p_{new}를 POX 내의 모든 패턴들과 각각 비교를 해서 정의역 패턴 집합 P와 공변역 패턴 집합 P'을 찾아낼 수도 있지만, 이렇게 하면 POX를 유지하는 데에는 너무 오랜 시간이 걸리게 된다. 따라서 본 논문에서는 POX를 보다 효율적으로 유지하는 알고리즘을 제안하고자 한다.

한 XPath 패턴 p에서 다른 XPath 패턴 p'으로의 준동형이 존재한다면 다음과 같은 속성이 있다.

프로퍼티 1. 만일 XPath 패턴 p에서 XPath 패턴 p'으로의 준동형이 존재한다면, p의 모든 가지(branch)들은 각각 p'의 가지로의 준동형을 갖는다. □

본 논문이 제안하는 알고리즘은 프로퍼티 1을 이용하여 모든 패턴을 가지들로 나누어 가지들 사이의 준동형을 구한다. 그리고 그것을 기반으로 p_{new}의 후보 정의역 패턴들과 후보 공변역 패턴들을 먼저 구한 뒤 일련의 정제 작업을 거쳐 진정한 정의역 패턴들과 공변역 패턴들을 구한다. 그리고 최종적으로 구한 정의역 패턴들과 공변역 패턴들 사이에 p_{new}를 위치시키면 알고리즘은 종료된다.

4. 노드간의 매칭

4.1 매칭의 정의

이번 절에서는 가지들 사이의 준동형을 구하는 데에 중요한 개념으로써 노드간의 매칭(matching)을 정의한다. b를 임의의 트리 패턴 p에 존재하는 가지라 하고, b'을 또 다른 트리 패턴 p'에 존재하는 가지라고 하자. 그리고 b_x는 b의 루트 노드에서부터 $x(\in \text{NODES}(b))$까지의 노드들로 이루어진 선형 패턴이라고 하고, $b'_{x'}$는 b'의 루트 노드에서부터 $x'(\in \text{NODES}(b'))$까지의 노드들로 이루어진 선형 패턴이라고 하자. 여기에서 NODES(p)는 임의의 패턴이나 가지 p의 모든 노드들의 집합을 의미한다. 이 때, b_x에서 $b'_{x'}$으로의 준동형이 존재한다면 "x는 x'에 매치한다(match)"라고 정의하고, x를 매칭 노드(matching node), x'을 매치드 노드(matched node)라고 정의한다. 그림 3은 노드들간의 매칭을 나타낸다.

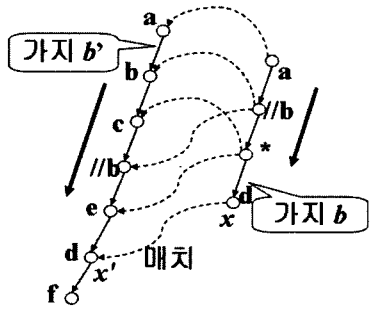

그림 3 노드 x'에 매치하는 노드 x

그림 3에서처럼, 가지 b의 단말 노드(leaf node)가 가지 b'의 임의의 노드에 매치하면, b에서 b'으로 준동형이 존재하게 된다. 우리는 두 가지의 루트 노드에서부터 단말 노드까지 노드들을 차례로 매치시켜 나가면서 가지들 사이의 준동형을 구할 수 있다. 이 때 b를 정의역 가지(domain branch), b'을 공변역 가지(codomain branch)라고 정의한다.

4.2 매칭 노드 및 매치드 노드 찾기

가지들 사이에서 매칭 노드를 찾는 방법은 다음과 같다. 루트 노드에서부터 순서대로 번호를 부여하여, x_i를

가지 b의 i번째 노드라고 하고, x'_j를 가지 b'의 j번째 노드라고 정의하자. b의 루트 노드인 x_1은 b'의 루트 노드인 x'_1과 같다고 가정하자. (무의미 노드(dummy node)를 b와 b'의 루트 노드로 추가하면 가정은 손쉽게 성립한다) 만일 x_i의 매칭 노드가 x'_j라면, x_{i+1} 노드의 매칭 노드는 다음 세 가지 경우가 될 수 있다. 여기에서 LABEL(x)는 임의의 노드 x의 라벨을 의미한다. 그림 4는 세 가지 경우를 나타낸다.

(i) x_i과 x_{i+1}이 부모-자식 관계일 때, LABEL(x'_{j+1}) = * 이거나 LABEL(x'_{j+1}) = LABEL(x_{i+1})인 x'_{j+1}이 x_{i+1}의 매칭 노드이다.

(ii) x_i와 x_{i+1}이 조상-자손 관계일 때, x'_j와 x'_{j+1}이 조상-자손 관계이고, LABEL(x'_{j+1}) = * 이거나 LABEL(x'_{j+1}) = LABEL(x_{i+1}) 인 x'_{j+1} 이 x_{i+1} 의 매칭 노드이다.

(iii) x'_{k-1}와 x'_k이 조상-자손 관계이고 LABEL(x'_k) = * 이거나 LABEL(x'_k) = LABEL(x_{i+1})인 ($1 < k \leq j$), x'_k가 x_{i+1} 의 매칭 노드이다.

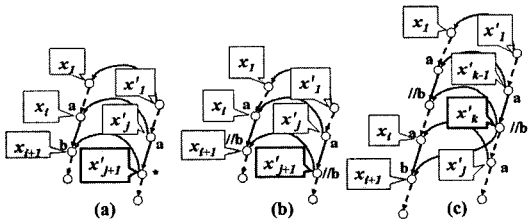

그림 4 x_{i+1} 의 매칭 노드 찾기

가지들 사이에서 매치드 노드를 찾는 방법은 다음과 같다. 역시 루트 노드에서부터 순서대로 번호를 부여하여, x_i 를 가지 b의 i번째 노드라고 하고, x'_j 를 b'의 j번째 노드라고 정의하자. b의 루트 노드인 x_1은 b'의 루트 노드인 x'_1 과 같다고 가정하자. (역시 무의미 노드를 b와 b'의 루트 노드로 추가하면 가정은 손쉽게 성립한다) 만일 x_i의 매치드 노드가 x'_j라면, x_{i+1}의 매치드 노드는 다음 두 가지 경우가 될 수 있다. 그림 5는 두 가지 경우를 나타낸다.

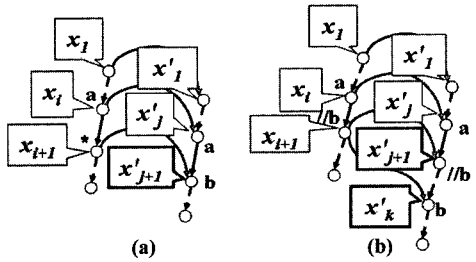

그림 5 x_{i+1}의 매치드 노드 찾기

(i) x_i와 x_{i+1}이 부모-자식 관계일 때, x'_j와 x'_{j+1}이 부모-자식 관계이고, LABEL(x_{i+1}) = * 이거나 LABEL(x_{i+1}) = LABEL(x'_{j+1})인 x'_{j+1}이 x_{i+1}의 매치드 노드이다.

(ii) x_i와 x_{i+1}이 조상-자손 관계일 때, LABEL(x_{i+1}) = * 이거나 LABEL(x_{i+1}) = LABEL(x'_k)인 ($j+1 \leq k \leq n$, $n=|p'|$), x'_k가 x_{i+1}의 매치드 노드이다.

5. POX 유지 알고리즘

5.1 데이타 구조 및 전처리

가지들 사이의 준동형 관계를 보다 효율적으로 구하기 위해서는 인덱스(index) 구조가 필요하다. 본 논문에서 제안하는 인덱스인 *inPOX*(index for POX) 트리의 구조는 다음과 같다. 무의미 노드를 각 패턴의 루트 노드로 추가하고, 모든 패턴을 가지들로 나누고 각각에 고유 ID를 붙인다. 그리고 루트 노드에서부터 시작하여 임의의 노드까지 경로가 같은 가지들이 있다면 그 경로를 하나로 합친다. 그림 6의 inPOX 트리는 왼쪽에 나타난 다섯 개의 패턴들을 하나의 트리 형태로 합친 모습을 나타낸다.

inPOX 트리의 각 노드들은 루트에서부터 그 노드까지의 경로와 일치하는 가지들의 ID를 갖는다. 예를 들어, 그림 6에서 inPOX 트리의 //b 노드는 루트에서부터 /a//b로 이루어진 가지이므로, 이와 일치하는 가지인 $p_{1.1}$을 그 값으로 갖는다. 결국 inPOX 트리는 모든 패턴의 가지들에 대한 정보를 갖게 된다.

XPath 패턴들을 inPOX 트리로 구성하면 정보의 손실이 있을 수 있으므로, XPath 패턴들의 정보를 따로 테이블에 저장하여 유지해야 한다. 각 패턴 트리의 노드들마다 고유한 ID를 부여하고, 가지마다 노드 ID를 루트에서 단말 노드까지 순서대로 나열한다. 표 1은 XPath 패턴들의 정보를 유지하는 테이블의 예이다. *본가지 ID*는 가지가 분기된 본래의 가지를 의미하며 *위치*는 분기된 노드의 위치를 나타낸다. 노드 *순서*는 루트에서 단말 노드까지의 순서를 의미한다. 예를 들어, $p_{4.3}$의 경우, $p_{4.1}$ 가지에서 분기되어 나왔으며, 두 번째 노드가 분기 노드(branching node)이므로, 본가지 ID는 $p_{4.1}$, 위치는 2이 된다.

전처리 단계에서는 새로운 XPath 패턴 p_{new}가 들어오면 그 패턴을 트리 패턴으로 변형하고, 무의미 노드를 루트 노드로 추가하는 작업을 한다. 그리고 나서 p_{new}가 기존에 POX에 등록된 패턴과 동일한 패턴인지 검사한다. 만일 POX에 p_{new}와 동일한 패턴이 존재한다면 알고리즘은 종료된다. 동일한 패턴이 존재하지 않는 경우에 한하여 5.2, 5.3 절의 알고리즘을 진행한다.

5.2 후보 정의역 패턴 및 후보 공변역 패턴 찾기

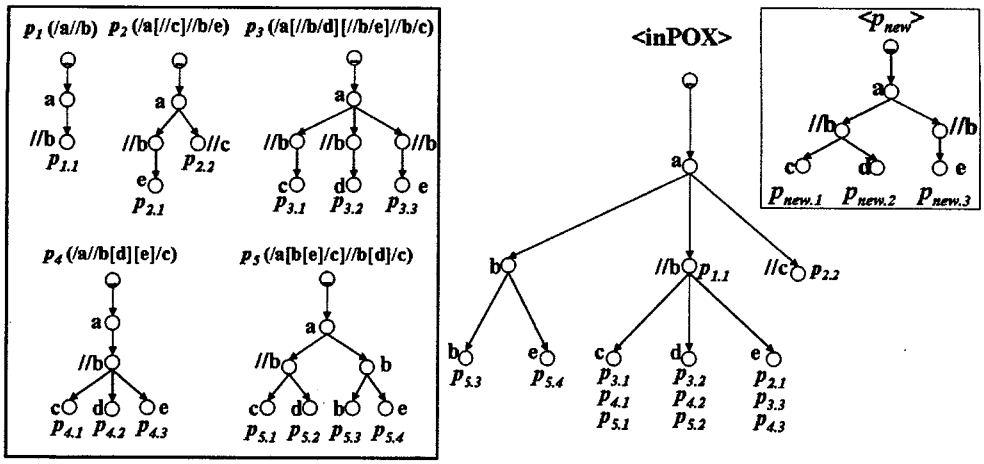

그림 6 다섯 개의 패턴을 하나의 트리 형태로 만든 inPOX 트리와 새로운 패턴 p_{new}

표 1 XPath 패턴들의 정보를 유지하는 테이블

XPath ID	가지 ID	본가지 ID	위치	노드 순서
p_1	$p_{1.1}$	-	-	a_1 - //b_1
p_2	$p_{2.1}$	-	-	a_1 - //b_1 - e_1
	$p_{2.2}$	$p_{2.1}$	1	a_1 - //c_1
p_3	$p_{3.1}$	-	-	a_1 - //b_1 - c_1
	$p_{3.2}$	$p_{3.1}$	1	a_1 - //b_2 - d_1
	$p_{3.3}$	$p_{3.1}$	1	a_1 - //b_3 - e_1
p_4	$p_{4.1}$	-	-	a_1 - //b_1 - c_1
	$p_{4.2}$	$p_{4.1}$	2	a_1 - //b_1 - d_1
	$p_{4.3}$	$p_{4.1}$	2	a_1 - //b_1 - e_1
p_5	$p_{5.1}$	-	-	a_1 - //b_1 - c_1
	$p_{5.2}$	$p_{5.1}$	2	a_1 - //b_1 - d_1
	$p_{5.3}$	$p_{5.1}$	1	a_1 - b_2 - b_3
	$p_{5.4}$	$p_{5.3}$	2	a_1 - b_2 - e_1

후보 정의역 패턴을 찾는 것은 새로운 패턴 p_{new}의 모든 가지들에 대해서 정의역 가지를 찾는 과정으로 이루어진다. 그림 7은 정의역 가지를 찾는 알고리즘의 의사 코드이다.

정의역 가지를 찾는 과정은 p_{new}를 전위 순회(pre-order traversal)하면서 만나는 노드 x의 매칭 노드를 inPOX 트리에서 찾는 과정이다(단계 4-8). 여기에서 찾은 매칭 노드들은 x의 자식 노드들의 매칭 노드를 찾는데 참조되므로, 스택 S에 저장한다(단계 16). 만일, 매칭 노드인지 검사하기 위해 만난 inPOX 트리의 노드 x'이 조상-자손 관계를 갖는 노드이면, x의 자손 노드의 매칭 노드가 될 수 있으므로(단계 9), 차후에 있을지도 모를 퇴각 검색(backtracking)을 용이하게 하기 위해 스택 S_d에 x'을 저장한다(단계 10, 단계 17). S_d의 가장 상단에는 항상 현재까지 비교한 모든 조상-자손 관계를 갖는 노드들이 유지되도록 한다. 그래서 x의 매칭 노드들을 찾을 때에는 S_d의 가장 상단에 있는 노드들 가운데 매칭 노드가 존재하는지를 검사하고(단계 11-15), 여기에서 찾은 매칭 노드들을 역시 S에서 x의 매칭 노드가 저장된 부분에 추가한다(단계 6, 단계 13, 단계 16). 노드 x의 매칭 노드를 찾는 단계가 끝나면, 전위 순회에 따라 x의 자식 노드들을 차례로 재귀적으로 처리하며(단계 18-19), 모든 자식 노드들에 대해 매칭 노드를 찾는 과정이 끝나면 S와 S_d에 대해 팝(pop) 연산을 행한다(단계 20-21). 만일 노드 x의 매칭 노드가 패턴의 가지 ID를 가지고 있다면, 그 ID에 해당하는 가지로부터 p_{new}에서 x를 포함하는 가지로의 준동형이 존재하게 된다. 즉, ID에 해당하는 가지가 p_{new}에서 x를 포함하는 가지의 정의역 가지가 된다는 뜻이다. 따라서, XPath 패턴 테이블에서 이 ID에 해당하는 가지들을 찾아 그

```
Algorithm FindDomainBranch(Node x, Stack S, Stack S_d)
Input: x is a current node in p_new.
Output: Updated Stacks S and S_d, Updated result table RT.
1) Initialize MatchingNodeList to be empty;
2) Initialize DescendantNodeList to be empty;
3) for each m in S.top do
4)    for each x' in a set of children nodes of m do
5)       if (x' is the matching node of x) then
6)          Append x' to MatchingNodeList;
7)          Record all branches correspondent to IDs
            in x' as the domain branches of p_new into RT;
8)          Record the matched nodes of nodes in b into RT where
            b is a path from the root node of inPOX tree to x';
9)       if (x' starts with //) then
10)         Append x' to DescendantNodeList;
11)for each x' in S_d.top do
12)   if (x' is the matching node of x) then
13)      Append x' to MatchingNodeList;
14)      Record all branches correspondent to IDs in x'
         as the domain branches of p_new into RT;
15)      Record the matched nodes of nodes in b into RT where
         b is a path from the root node of inPOX tree to s';
16)Push MatchingNodeList into S;
17)Push S_d.top and DescendantNodeList into S_d;
18)for each c that is a child node of x do
19)   (S, S_d) = FindDomainBranch(c, S, S_d);
20)Pop S;
21)Pop S_d;
22)Return (S, S_d);
```

그림 7 정의역 가지를 찾는 알고리즘

노드 순서에 대응하는 p_{new}에서의 노드를 결과 테이블 RT에 적는다(단계 7-8, 단계 14-15). 이는 나중에 후처리 과정에서 사용된다.

프로퍼티 1에 따라, 모든 가지로부터 p_{new}의 일부 가지로의 준동형을 갖는 패턴들은 p_{new}로의 정의역 패턴이 될 가능성이 있으므로, 그러한 패턴들만을 남기고 나머지는 모두 후처리 대상에서 제외한다. 표 2는 그림 6에 나타난 inPOX 트리와 p_{new} 예제를 이용하여 알고리즘을 적용한 결과 테이블 RT를 나타낸다.

후보 공변역 패턴을 찾는 것은 p_{new}의 모든 가지들에 대해서 공변역 가지들을 찾는 과정이다. 그림 8은 이 과

```
Algorithm FindCodomainBranch(Node x, Stack S')
Input: x is a current node in p_new.
Output: Updated Stack S', Updated result table RT'.
1) Initialize MatchedNodeList to be empty;
2) for each m in S'.top do
3)    for each x' in a set of descendent nodes of m do
4)       if (x' is the matched node of x) then
5)          Append x' to MatchedNodeList;
6) Push MatchedNodeList into S';
7) for each c that is a child node of x do
8)    S' = FindCodomainBranch(c, S');
9) if (x is a leaf node) then
10)   for each m in S'.top do
11)      for each d in a set of descendent nodes of m do
12)         Record all branches correspondent to IDs in x'
            as the codomain branches of p_new into RT';
13)         Record the matched nodes of nodes in b into RT' where
            b is a path from the root node of p_new to x;
14)Pop S';
15)Return S';
```

그림 8 공변역 가지를 찾는 알고리즘

정의 의사 코드를 나타낸다. 후보 정의역 패턴을 찾는 과정과 유사하게, p_{new}를 전위 순회 하면서 만나는 노드 x의 매치드 노드를 inPOX 트리에서 찾는 과정으로 이루어진다. 역시 여기에서 찾은 매치드 노드들은 x의 자식 노드들의 매치드 노드를 찾는 데 참조되므로, 스택 S'에 저장한다(단계 3-6). x의 매치드 노드를 찾은 후에는 x의 모든 자식 노드들을 재귀적으로 처리한다(단계 7-8). 모든 자식 노드들을 처리한 후에는 S'에 대해 팝 연산을 수행한다(단계 14). x가 p_{new}의 단말 노드이고 x'이 x의 매치드 노드라고 하자. b를 x가 속해있는 가지라고 할 때, x'나 x'의 자손 노드가 일부 가지들의 ID를 가지고 있다면, b에서 그 ID에 해당하는 가지로 준동형이 존재한다. 따라서, ID에 해당하는 가지는 b의 공변역 가지가 된다. 찾아진 공변역 가지들에 대해, 그 가지의 노드들과 그 노드들에 대응하는 b의 노드들을 결과 테이블 RT'에 순서대로 적는다(단계 12-13). RT'은 후처리 과정에서 사용된다.

표 2 정의역 가지를 찾는 알고리즘의 결과표 (RT)

XPath ID	정의역가지 ID	본가지 ID	위치	정의역 가지 노드 순서	매치드 노드 순서
p_1	$p_{1.1}$	-	-	a_1 - $//b_1$	a_1 - $//b_1$ a_1 - $//b_2$
p_2	$p_{2.1}$	-	-	a_1 - $//b_1$ - e_1	a_1 - $//b_2$ - e_1
	$p_{2.2}$	$p_{2.1}$	1	a_1 - $//c_1$	a_1 - c_1
p_3	$p_{3.1}$	-	-	a_1 - $//b_1$ - c_1	a_1 - $//b_1$ - c_1
	$p_{3.2}$	$p_{3.1}$	1	a_1 - $//b_2$ - d_1	a_1 - $//b_1$ - d_1
	$p_{3.3}$	$p_{3.1}$	1	a_1 - $//b_3$ - e_1	a_1 - $//b_2$ - e_1
p_4	$p_{4.1}$	-	-	a_1 - $//b_1$ - c_1	a_1 - $//b_1$ - c_1
	$p_{4.2}$	$p_{4.1}$	2	a_1 - $//b_1$ - d_1	a_1 - $//b_1$ - d_1
	$p_{4.3}$	$p_{4.1}$	2	a_1 - $//b_1$ - e_1	a_1 - $//b_2$ - e_1

표 3 공변역 가지를 찾는 알고리즘의 결과표 (RT')

XPath ID	정의역 가지 ID	본가지 ID	위치	정의역 가지 노드 순서	매치드 노드 순서
p_3	$p_{new.1}$	-	-	a_1 - //b_1 - c_1	a_1 - //b_1 - c_1
	$p_{new.2}$	$p_{new.1}$	2	a_1 - //b_1 - d_1	a_1 - //b_2 - d_1
	$p_{new.3}$	$p_{new.1}$	1	a_1 - //b_2 - e_1	a_1 - //b_3 - e_1
p_4	$p_{new.1}$	-	-	a_1 - //b_1 - c_1	a_1 - //b_1 - c_1
	$p_{new.2}$	$p_{new.1}$	2	a_1 - //b_1 - d_1	a_1 - //b_1 - d_1
	$p_{new.3}$	$p_{new.1}$	1	a_1 - //b_2 - e_1	a_1 - //b_1 - e_1
p_5	$p_{new.1}$	-	-	a_1 - //b_1 - c_1	a_1 - //b_1 - c_1
	$p_{new.2}$	$p_{new.1}$	2	a_1 - //b_1 - d_1	a_1 - //b_1 - d_1
	$p_{new.3}$	$p_{new.1}$	1	a_1 - //b_2 - e_1	a_1 - b_2 - e_1

프로퍼티 1에 따라, 모든 가지로부터 p_{new}의 일부 가지로의 준동형을 갖는 패턴들은 p_{new}로의 공변역 패턴이 될 가능성이 있으므로, 그러한 패턴들만을 남기고 나머지는 모두 후처리 대상에서 제외한다. 표 3은 이 과정에서 작성된 결과 테이블 RT'을 나타낸다.

5.3 후처리

가지들 사이의 준동형 만으로는 패턴들 사이의 준동형을 결정할 수 없다. 그 이유는, 가지들 사이의 준동형만으로 구한 후보 정의역 패턴 및 공변역 패턴은 준동형 관계에 대한 조건 가운데에서 준동형이 함수(function)라는 조건을 만족시키지 못할 수 있기 때문이다. 패턴 p에서 패턴 p'으로의 준동형 함수 h는 NODES(p)→NODES(p')로 나타낼 수 있는데, 이는 함수의 특성상 p의 모든 노드 x에 대해서 반드시 정확히 하나의 노드 x'이 p'에 존재해야 함을 의미한다[11].

따라서 찾아낸 정의역 패턴 p에 대해서, p_{new}에 정확히 한 개의 매핑 노드를 갖지 않는 노드가 존재하는 p를 제거하고, 역시 공변역 패턴 p'에 대해서, p_{new}의 각 노드들에 대해, 그 정확히 한 개의 매핑 노드가 결정되지 않는 p'을 제거해야 한다. 표 2에서, 본 가지 ID와 위치는 각 가지들이 그 위치에는 본 가지 ID의 그 위치에 해당하는 노드와 동일한 노드가 와야 한다는 것을 의미한다. 예를 들어, $p_{4.3}$은 $p_{4.1}$의 두 번째 노드에서 갈라져 나왔으므로 $p_{4.3}$의 공변역 가지의 두 번째 노드도 $p_{4.1}$의 공변역 가지의 두 번째 노드와 같아야 한다. 그러나 표 2의 매치드 노드 순서에서 $p_{4.3}$에 해당하는 가지의 두 번째 노드가 $p_{4.1}$에 해당하는 가지의 두 번째 노드와 다름을 알 수 있다. 따라서 결과표에서 $p_{4.3}$은 적절치 못하므로 제거된다. 제거 작업이 끝나고도 모든 가지들이 p_{new}의 가지들로 준동형을 갖는 패턴이 진정한 정의역 패턴이 된다. 표 2에서는 p_1, p_2, p_3이 p_{new}의 정의역 패턴이다.

정의역 패턴을 구할 때와 유사하게, 표 3에서는 p_3에서 $p_{new.2}$에 대응되는 가지가 적절치 못하므로 제거되며, 따라서 진정한 공변역 패턴은 p_4와 p_5 뿐이다.

정의역 패턴과 공변역 패턴을 구한 이후에는 나머지 데이타 구조를 갱신해야 한다. p_{new}를 inPOX 트리에 추가하는 것은 p_{new}를 전위 순회하면서 만나는 노드 중에서 처음으로 inPOX 트리에 존재하지 않는 노드가 나타나면 그 노드를 루트로 하는 부분 트리를 통째로 inPOX 트리에 삽입하면 된다. 그리고 나서 XPath 패턴의 테이블에 p_{new}의 정보를 추가한다.

POX를 업데이트 하는 과정은 구해진 정의역 패턴들의 집합 P에서 최소 정의역 패턴을 구하고, 구해진 공변역 패턴들의 집합 P' 가운데에서 최대 공변역 패턴을 구한 뒤 그 사이에 p_{new}를 삽입하는 것으로 이루어진다. 최소 정의역 패턴이란, 자신으로부터 P 내의 어떠한 패턴으로도 준동형이 존재하지 않는 패턴을 뜻하며, 유사하게 최대 공변역 패턴이란 P' 내의 어떠한 패턴에도 자신을 향한 준동형이 존재하지 않는 패턴을 뜻한다. 최소 정의역 패턴 p에 대해서, 공변역 패턴을 가리키는 링크를 모두 제거하고 p_{new}만을 공변역 패턴으로 가리키도록 하고, p_{new}는 p를 정의역 패턴으로 가리키도록 한다. 그리고 최대 공변역 패턴 p'에 대해서, 정의역 패턴을 가리키는 링크를 모두 제거하고 p_{new}만을 정의역 패턴으로 가리키도록 하고, p_{new}는 p'을 공변역 패턴으로 가리키도록 하면, POX의 업데이트 과정은 종료된다.

6. 알고리즘 분석

본 논문이 제안하는 알고리즘의 건전함(soundness)과 완전함(completeness)는 [10]에 증명되어 있다. 한편, 알고리즘의 시간 복잡도(time complexity)는 $O(n^2m + x(b + b_{max}))$이다[10]. 여기에서 n은 p_{new}의 총 노드의 개수, m은 inPOX 트리의 총 노드의 개수이며, x는 POX 내의 패턴의 수이고 b 는 p_{new}의 가지 수이며, b_{max} 는 POX 내의 각 패턴이 갖는 가지 수의 최대값이다.

패턴 p에서 p'으로의 준동형이 존재하는지를 결정하

는 알고리즘은 $O(|p||p'|^2)$이라고 알려져 있다[4]. 따라서, POX와 inPOX 트리를 이용하지 않는 경우에, 즉 p_{new}가 들어왔을 때 기존의 패턴들과 일일이 준동형을 비교하는 방법은 $O(\max(ns^2, n^2s))$가 된다. 여기에서 n은 p_{new}의 총 노드의 개수, s은 모든 패턴들의 총 노드의 개수이다. 일반적으로 s는 m보다 크기 때문에, POX와 inPOX 트리를 이용한 본 알고리즘이 더 효율적임을 알 수 있다.

7. 결론 및 향후 연구 계획

본 논문은 XPath 패턴들간의 준동형 관계를 유지할 필요성을 최초로 제기하였으며, 이 정보를 저장할 수 있는 데이타 구조인 POX를 제안하였다. 그리고 그것을 유지할 수 있는 방법으로 효율적인 다항 시간 알고리즘을 제안하였다.

궁극적으로 필요한 것은 패턴간의 포함 관계라고 할 수 있다. 하지만 이미 두 패턴들 사이의 포함 관계를 구하는 문제는 co-NP complete 문제임이 증명되었으므로 [4], 준동형 관계보다 포함 관계에 더 가까운 관계가 있는가를 연구해 볼 필요가 있을 것으로 보인다.

참 고 문 헌

[1] World Wide Web Consortium, XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/xpath, W3C Recommendation, November 1999.

[2] World Wide Web Consortium, XSL Transformations (XSLT) Version 1.0, http://www.w3.org/TR/xslt, W3C Recommendation, November 1999.

[3] World Wide Web Consortium, XML Pointer Language (XPointer), http://www.w3.org/TR/xptr, W3C Working Draft, August 2002.

[4] G. Miklau and D. Suciu, Containment and Equivalence for an XPath Fragment, In Proceedings of the 21st Symposium on Principles of Database Systems, pages 65-76, 2002.

[5] C.-Y. Chan, W. Fan, P. Felber, M. Garofalakis, and R. Rastogi, Tree Pattern Aggregation for Scalable XML Data Dissemination, In Proceedings of the 28th VLDB Conference, pages 826-837, 2002.

[6] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, Design and Evaluation of a Wide-Area Event Notification Service, ACM Transactions on Computer Systems, Vol. 19, No. 3, pages 332-383, 2001.

[7] M. Altinel and M. J. Franklin, Efficient Filtering of XML Documents for Selective Dissemination of Information, In Proceedings of the 26th VLDB Conference, pages 53-64, 2000.

[8] C.-Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi, Efficient Filtering of XML Documents with XPath Expressions, The VLDB Journal, Vol. 11, No. 4, pages 354-379, 2002.

[9] Y. Diao, M. Altinel, M. J. Franklin, H. Zhang, and P. Fischer, Yfilter: Efficient and Scalable Filtering of XML Documents, In Proceedings of International Conference of Data Engineering, pages 341-342, 2002.

[10] S. Yoo, J. H. Son, and M. H. Kim, Maintaining Homomorphism Information of XPath Patterns, Korea Advanced Institute of Science and Technology (KAIST) Technical Report (CS-TR-2004-209), 2004. http://cs.kaist.ac.kr/research/technical/Archive/CS-TR-2004-209.pdf

[11] J. A. Dossey, A. D. Otto, L. E. Spence, and C. V. Eynden, Discrete Mathematics, Scott, Foresman and Company, 1987.

유 상 현

2002년 연세대학교 정보산업공학 학사 2004년 한국과학기술원 전산학 석사 2004년~현재 한국과학기술원 전산학과 박사과정 재학 중. 관심분야는 데이타베이스, e-비즈니스, XML

손 진 현

1996년 서강대학교 전산학과 학사. 1998년 한국과학기술원 전산학과 석사. 2001년 한국과학기술원 전자전산학과 박사 2001년 9월~2002년 8월 한국과학기술원 전자전산학과 박사후 연구원. 2002년 9월~현재 한양대학교 컴퓨터공학과 조교수. 관심분야는 데이타베이스, e-비즈니스, 유비쿼터스 컴퓨팅, 임베디드 시스템

김 명 호

1982년 서울대학교 컴퓨터 공학과 학사 1984년 서울대학교 컴퓨터 공학과 석사 1989년 MICHIGAN 주립대 전산학과 박사. 1989년 MICHIGAN 주립대 연구원 1989년~1993년 한국과학기술원 조교수 1993년~1999년 한국과학기술원 부교수 1999년~현재 한국과학기술원 정교수. 1992년~1993년 개방형 컴퓨터 통신 연구회(OSIA) 분산 트랜잭션처리 분과위(TG-TP) 의장. 1993년~1994년 한국통신기술협회(TTA) 분산 트랜잭션처리 실무 위원회 의장. 2004년~현재 데이타베이스연구회 운영위원장. 관심 분야는 데이타베이스, e-비즈니스, 분산트랜잭션, 분산시스템, 워크플로우