멀티미디어 상연그래프 질의언어와 대수를 이용한 질의처리방법

(A Query Language for Multimedia Presentation Graphs and Query Processing Techniques with Algebra)

이 태 경 †

(Tae-kyong Lee)

요 약 최근 폭발적인 증가를 보이고 있는 멀티미디어 자료의 양과 그 자료들을 이용할 수 있는 하드웨어의 발전은 멀티미디어 상연물을 이용하는 여러 응용 분야에 대한 관심을 촉발시키고 있다. 이에 멀티미디어 상연물의 효과적인 이용을 위해서는 멀티미디어 상연물과 DBMS와의 통합이 필요하다. 이 논문에서는 내용(content)에 근거한 상연물 검색과 검색 처리 기술의 문제를 다룬다.

현재 멀티미디어 상연물 제작 도구(authoring tool)들은 멀티미디어 상연물을 상연 그래프(presentation graph)를 이용하여 표현하고 있으며 상연 그래프는 DAG(directed acyclic graph)이다. 각 노드는 같은 타입의 미디어 스트림을 나타내며 에지는 스트림간의 상연 순서와 동기화(synchronization) 방법을 나타낸다. 각각의 스트림에 포함된 정보, 이 정보들간의 순서, 그리고 스트림간의 상연 순서는 상연의 내용을 구성한다.

GCalculus/S(GCalculus with Set Operators)는 calculus에 바탕을 둔 검색언어이며 멀티미디어 자료들의 물리적 특징과 내용을 다룰 수 있다. 개개의 노드 안에서의 정보의 변화와 노드 사이의 순서는 시간 연산자(temporal operator) *Next, Connected, Until*을 이용하여 표현한다. 검색의 처리를 위하여 객체 대수(object algebra)인 O-Algebra를 확장한다.

Abstract Recently the technological advance in the hardware dealing with multimedia data as well as the explosive increase of the volume of multimedia data bring about new interest in the use of multimedia presentations in many application domains. To use multimedia presentations efficiently, the integration of multimedia presentations into DBMS is necessary. This paper presents a multimedia presentatation query language based on contents and query processing techniques.

Presently, multimedia presentation authoring tools denote a multimedia presentation using a presentation graph which is a DAG. A Node in the graph is a same type of media stream and edges denote a play-out order and a synchronization way among nodes. The contents of presentations graphs are the information of each stream, the sequential order of the information inside each stream and the play-out order among the streams.

GCalculus/S is a calculus-based query language and can deal with the contents of a presentation graph and physical characteristics of multimedia data. It expresses the sequential order of information inside each stream and the play-out order of streams of a presentation graph using temporal operators *Next, Connected* and *Until*. O-Algebra, which is object algebra, is extended to process GCalculus/S queries.

1. 서 론

최근 비전통적인 타입의 자료들을 DBMS에 통합하는 문제에 많은 연구가 이루어지고 있으며 그 연구의 한 부분으로 그래프 타입에 대한 연구가 이루어지고 있다[2, 9,12,18,22]. 그 이유로 그래프 타입은 GIS, network,

† 정 회 원 : 울산대학교 정보디자인학과 교수
tklee@uou.ulsan.ac.kr
논문접수 : 1999년 3월 19일
심사완료 : 2000년 4월 4일

WWW 등과 같은 영역의 문제들을 자연스럽게 표현하기 때문이다. 그래프 타입의 다른 응용 분야로 멀티미디어 상연물[15,16,17]이 있다. 이 논문에서는 멀티미디어 상연물 질의어인 GCalculus/S와 질의의 대수적 처리를 다룬다.

멀티미디어 상연이란 동기화된 방식으로 때로는 대화식의 방법으로 사용자에게 멀티미디어 자료를 전달하는 것이다. 현재 멀티미디어 상연물은 컴퓨터 보조 훈련, 컴퓨터 보조 학습, 온라인 책등에 활용되고 있다. 멀티미디어 상연물[3,13,26]은 앞으로 디지털 도서관, 전자책등의 영역에서도 활용될 것이다.

멀티미디어 자료들을 이용하여 멀티미디어 상연물을 제작하는 제작 도구중 성공적인 것들은 IconAuthor, Authorware, Quest[1,14,23]등이 있다. 이 도구들은 여러 가지 타입의 자료들을 이용하여 멀티미디어 상연물의 구성과 상연 방법을 시각적으로 표현하기 위하여 그래프, 즉 상연물 그래프(presentation graph)를 이용하고 있다. 상연물 그래프의 노드는 같은 타입의 멀티미디어 자료로 만들어진 스트림을 나타내며 노드 사이의 에지는 스트림 사이의 상연 순서와 동기화 방법을 나타낸다. 예를 들어 그림 1의 상연 그래프는 "금강산 관광"이며 두 개의 비디오 스트림과 두 개의 오디오 스트림으로 구성되어 있다. 비디오 스트림 "금강산"과 오디오 스트림 "Promo Song"은 동시에 상연되며 비디오 스트림 "금강산의 사계"는 비디오 스트림 "금강산"과 오디오 스트림 "Promo Song"이 끝난후 순차적으로 상연된다.

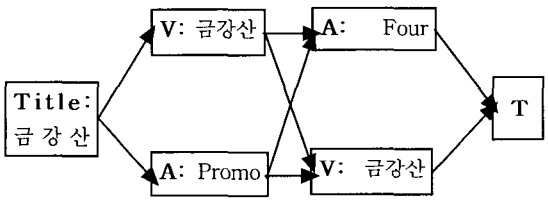


그림 1 멀티미디어 상연물 "금강산 관광"

멀티미디어 자료들이 가지는 물리적 특성은 분할화(segmentation)이다. 멀티미디어 자료들은 의미 있는 내용 이 부여될 수 있는 최소한의 단위가 있다. 예를 들어 비디오 자료들의 경우에는 그 최소의 단위가 프레임(frame)이다. 그리고 연속적인 프레임의 집합은 각각의 프레임보다 큰 단위의 멀티미디어 자료를 만들고 각각의 프레임의 내용들로 구성되는 내용을 가진다. 즉, 멀티미디어 자료들은 큰 자료에서 내용이 주어질 수 있는 최소 단위까지 분할이 가능하다.

멀티미디어 자료들은 내용을 가진다. 내용은 그 자료에 있는 객체, 객체들간의 관계와 같은 정보와 그 정보의 변이(sequential order of the information)이다. 분할화와 관련하여 멀티미디어 상연물의 내용은 분할화의 수준에 따라서 내용이 각각 다르다. 예를 들어 그림 1의 비디오 스트림 "금강산의 사계" 내에 있는 금강산의 계절에 따른 변화와 스트림간의 상연 순서 모두 내용이 될 수 있다.

위에서 언급된 멀티미디어 상연물 제작 도구[1,14,23]들은 초보적인 데이타베이스 기능을 가지고 있다. 그러나 이 도구들과 데이타베이스 사이의 관계는 긴밀하지 못하다. 이에 멀티미디어 상연물의 효과적 사용을 위하여 상연물과 데이타베이스 시스템의 통합이 요구되며 이 통합에는 상연물의 내용과 물리적 특성을 다룰수 있는 멀티미디어 상연물 질의어와 그 처리 기술의 개발이 필요하다.

GCalculus/S(GCalculus with Set Operators)는 calculus에 바탕을 둔 질의어이며 GCalculus(Graph Calculus)를 변형시켜 얻어진다. GCalculus는 시간 논리(Temporal Logic)[24]를 확장하여 연산자 **X**(Next), **U**(Until), **C**(Connected)와 Computational Tree Logic(CTL)[8]의 존재 패스 정량자(existential path quantifer) **E**와 범용 패스 정량자(universal path quantifier) **A**를 사용하여 그래프의 패스를 표현한다. GCalculus에서 범용 정량자 ∀, 범용 패스 정량자 **A**, 그리고 부정 연산자 ¬를 이용한 표현은 복잡하고 이해하기 어려운 표현을 만들어 낸다. 또한 그러한 표현의 처리 비용은 매우 비싸다. 이에 GCalculus를 변형시켜 GCalculus/S로 발전 시킨다. GCalculus/S는 GCalculus에서 정량자 ∀와 **A**를 제거시켜 ∃와 **E**만 가지며 ¬는 base predicate의 왼쪽에서만 나타난다. 그리하여 GCalculus/S는 좀더 쉬운 표현을 가능하게 만들며 그 처리 비용은 좀더 싸다.

GCalculus/S의 질의 처리를 위하여 O-Algebra[19]를 확장하여 대수(algebra)에 의한 "bottom-up" 처리 기술을 제공한다. 이를 위해 O-Algebra[19]에서 정의된 연산자와 Next, Until, Connected를 사용하여 표현된 노드 사이의 관계를 만족하는 패스를 찾기 위하여 새로운 연산자를 정의하여 사용한다. 처리 방법은 GCalculus/S에서 사용된 연산자에 대하여 귀납적인 증명을 이용하여 설명한다.

이 논문의 2장에서는 GCalculus/S를 위한 데이타 모델과 GCalculus/S의 구문(syntax)과 의미(semantics)를 다룬다. 3장에서는 GCalculus/S 질의를 객체 대수를

이용한 처리 방법을 설명한다. 4장은 관련 연구이며 5장은 결론이다.

2. GCalculus/S와 데이타 모델

2.1 데이타 모델

모든 언어는 데이타 모델에 바탕을 두고 있다. 이에 GCalculus/S가 바탕을 두는 데이타 모델이 필요하다. 그 데이타 모델은 앞에서 설명된 멀티미디어 자료들의 물리적 특징과 자료의 내용, 상연 그래프를 효과적으로 표현할 수 있어야 한다. 이에 객체 지향 데이타 모델을 이용하여 모델링을 한다. 이는 다음에 설명되는 이점이 있기 때문이다. 멀티미디어 자료에 나타나는 무수한 종류의 객체들과 그 객체들 사이의 관계는 class-subclass hierarchy와 composition hierarchy 이용하여 처리할 수 있다. 또한, 멀티미디어 자료의 물리적 특징인 분할화도 composition hierarchy를 이용하여 처리한다. 아래의 정의는 GCalculus/S를 위한 데이타 모델이다.

```
class Pres_Graph [
  name: String;
  other attributes;
  Nodes: {Pres_Node};
  Edges: {Pres_Edge}];

class Stream [
  name: String;
  type: String;
  rep_frame: <Frame>;
  other attributes];
class Pres_Edge
  [<Pres_Node>];

class Pres_Node: inherits
      from Stream[
  graph-in: Pres_Graph;
  child-nodes; {Pres_Node};
  parent-nodes; {Pres_Node};
    other attributes];
class Frame[
  name: String;
  objects: {C_Object};
  other attributes];

class C_Object [
  name: String;
  frame-in: Frame;
  other attributes];
```

보기: {}, <>는 set constructor와 sequence constructor를 나타낸다.

2.2 GCalculus와 GCalculus/S

GCalculus에는 두 가지 종류의 기호(symbol)가 있다. 하나는 논리 기호(logical symbol)이고 다른 하나는 비논리 기호(non-logical symbol)이다. 논리기호는 <, >, ≤, ≥와 같은 연산 비교 기호와 ⊂, ⊃, ⊆, ⊇, ∈, ∋와 같은 집합 비교 기호로 구성되어 있다. 각 기호의 의미는 술어 해석에서 사용되는 것이다. 비논리 기호는 괄호, 상수, 그리고 변수이다. GCalculus에서 사용되는 정량자들은 술어 해석에서 사용되는 존재 정량자 ∃, 범용 정량자 ∀, 그리고 새로이 도입되는 존재 패스 정량자 **E**와 범용 패스 정량자 **A**가 사용된다.

2.2.1 GCalculus의 구문

간단히 GCalculus의 구문을 기술하며 약식으로 의미를 설명한다. 포뮬라(formula) A1 -A4의 의미는 술어 해석에서 사용하는 것이다. 포뮬라 A5, A6, B1 - B3의 의미는 약식으로, 또는 예를 사용하여 설명한다.

A. State Formulas

(A1) proposition(0-ary 술어(predicate))

(A2) ϕ(t1,..,tn), ϕ는 n-ary 술어이며 t1,..,tn은 항(term)이다.

(A3) t1 $\emptyset$ t2, $\emptyset \in$ {set of logical operators}, t1과 t2는 항이다.

(A4) (∃y)p, (∀y)p, p는 스테이느 포뮬라이며 y는 p안에 있는 변수이다.

(A5) <<g,x,s>>p, g는 그래프 타입의 값을 가지는 항, x는 순서 타입의 값을 가지는 항(sequence-valued term), s는 순서를 나타내는 x의 처음 노드(starting node)를 나타내는 노드 항이다. 이 표현은 패스 포뮬라 p를 x가 만족한다는 표현이다.

(A6) **E**(x)<<g,x,s>>p, **A**(x)<<g,x,s>>p, **E**와 **A**는 ∃와 ∀의 의미와 유사한 의미를 가진다. **E**(x)<<g,x,s>>p의 의미는 어떤 노드 s에서 시작하는 패스 x가 존재하며 그 패스는 패스 포뮬라 p를 만족한다는 것이다. **A**(x)<<g,x,s>>p의 의미는 어떤 노드 s에서 시작하는 모든 패스 x는 패스 포뮬라 p를 만족한다는 것이다.

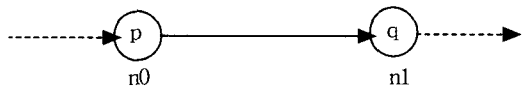
B. Path Formulas

(B1) [[n]]p, n은 노드 변수이며 스테이트 포뮬라 p를 만족한다.

(B2) (p), p는 패스 포뮬라.

(B3) p Θ q, p와 q는 패스 포뮬라, x는 순서 타입의 값을 가지는 항, $\Theta \in$ {**X**(next), **C**(connect), **U**(until)}, **X**, **C**, **U** 연산자의 의미는 아래의 그림으로 설명한다.

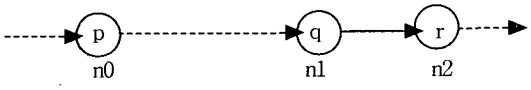
Example 2.1: **E**(x)<<g, x, n0>>[[n0]]p **X** [[n1]]q



p를 만족하는 하나의 노드(n0)로 이루어진 패스에 q를 만족하는 하나의 노드(n1)로 이루어진 패스가 인접하고 있으며 그 패스는 변수 x로 표현된다. 달리 설명하면 패스 x가 존재하며 x의 처음 노드(no)는 p를 만족하며 다음 노드(n1)는 q를 만족한다.

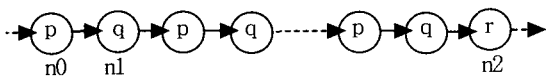
Example 2.2: **E**(x)<<g,x,n0>>[[n0]]p **C** ([[n1]]q **X** [[

n2]]r)



처음의 패스는 하나의 노드(n0)로 구성되어 있다. 두 번째의 path는 두 개의 노드로 구성되어 있으며 서로 인접하고 있다. 그 노드들(n1과 n2)은 각각 q와 r을 만족한다. 노드 n0와 n1사이에는 0개 또는 복수의 노드가 존재할 수 있다. 패스 변수 x가 나타내는 패스는 n0에서 시작하여 n2에서 끝난다.

Example 2.3: **E**(x)<<g,x,n0>>([[n0]]p **X** [[n1]]q) **U** [[n2]]r



노드 n0는 p를 만족한다. 다음의 노드 n1는 q를 만족한다. 계속하여 p와 q를 만족시키는 두 노드가 계속 나타나며 r을 만족 시키는 노드 n2가 나타날 때까지 계속된다.

2.2.2 GCalculus의 GCalculus/S로의 변형

관계 해석(Relational Calculus)에서 범용 정량자 ∀와 부정 연산자 ¬의 사용은 복잡한 형태의 관계 해석 질의 표현을 만든다.[21] 마찬가지로 GCalculus에서 범용 패스 정량자 **A**와 부정 연산자 ¬의 사용은 매우 복잡한 표현을 만들어 낸다. 이에 GCalclus에서 ∀, **A** 정량자들을 없애고 GCalclus/S(GCalculus with Set Operators)로 변형시킨다. GCalculus/S는 ∃와 **E** 정량자들만 가지며 ¬는 base predicate의 왼쪽에서만 사용된다. 이 변형은 좀더 쉬운 질의 표현을 가능하게 하여 준다. 아래에 GCalculus/S의 구문을 간단하게 설명한다.

(1) 정량자 **A**는 사용되지 않는다.

(2) 정량자 ∀는 사용되지 않는다.

(3) 부정 연산자 ¬는 base predicate의 왼쪽에만 나타난다. (즉, 여러개의 술어들과 논리 연산자 ∨ 또는 ∧로 만들어지는 포뮬라의 왼쪽에는 나타나지 않는다.)

Lemma 2.1: **A**(x)<<g, x, s0>>P는 스테이트 포뮬라이다. P는 패스 포뮬라이다. 그 패스 포뮬라를 만족하는 패스의 처음 노드와 마지막 노드는 s0와 sj이다. P안에 있는 p는 s0와 관계가 있는 단정(assertion)이다. 그리하면, **A**(x)<<g, x, s0>>P는 Sp ⊆ Sq와 동일(equivalent)하다 Sp = {x1 | <<g,x1,s0>>[[s0]]p **C** [[sn]]sn.type = ''Terminate''} and Sq = {x1 | <<g,x1,s0>>Q}. 만약, sj에 관한 스테이트 포뮬라가 sj.type = ''Terminate''의 단정을 포함하고 있으면 Q = P, 아니면 Q = P **C** [[sn]] sn.type = ''Terminate''.

Proof: S = {x}는 s0 노드에서 시작하여 Terminate 노드 sn에서 끝나는 패스의 집합이라 하자. 정의에 의하여 S = Sp.

(1) Case 1: sj.type = ''Terminate''가 사실일때.

(a) **A**(x)<<g,x,s0>>P가 사실이라고 가정하자. Sq는 {x1 | <<g,x1,s0>>P)로 표현될 수 있기 때문에 S = Sp = Sq. 그러므로 Sp ⊆ Sq.

(b) **A**(x)<<g,x,s0>>P가 사실이 아니라고 가정하자. 그러면, Sp와 Sq의 정의에 의하여 sn이 마지막 노드인 패스 xi가 Sp안에 포함된다. 그러나 이 패스는 Sq에는 포함되지 않는다. 그러므로 Sp ⊄ Sq.

(2) Case 2: sj.type = ''Terminate''가 사실이 아닐때

(a) **A**(x)<<g,x,s0>>P가 사실이라고 가정하자. 그러면 Sq = {x1 | <<g, x1,s0>>(P **C** [[sn]]sn.type = ''Terminate'')}, S = Sp = Sq. 그러므로 Sp ⊆ Sq. (b) **A**(x)<<g,x,s0>>P가 사실이 아니라고 가정하자. 그러면 Sp에 포함되어 있는 어떤 패스 xi는 Sq에 포함되지 않는다. 즉, Sp ⊄ Sq. □

Lemma 2.2.: **A** 또는 ∀ 정량자들을 가지고 있는 GCalculus의 어떠한 formula도 **E**와 ∃ 정량자, 집합 연산자를 이용하는 GCalculus/S의 포뮬라로 변형 시킬수 있다.

Proof: Lemma 2.1에 의하여 정량자 **A**를 가지고 있는 포뮬라는 정량자 **E**와 집합 연산자를 가지고 있는 포뮬라로 변형 시킬 수 있다. 범용 정량자 ∀와 부정 연산자 ¬를 가지고 있는 포뮬라는 [14]에 설명되어 있는 렘마와 알고리즘을 이용하여 존재 정량자 ∃와 집합 연산자를 가지고 있는 포뮬라로 변형 시킬 수 있다. □

아래에 GCalculus/S를 이용한 질의 표현의 예를 소개한다. 2.1에서 정의된 데이타 모델를 이용하여 class-name(x)로 표현되는 것은 변수 x가 class-name의 클래스에 속한다는 의미이다. Example 2.5는 범용 정량자 ∀가 포함된 질의를 두 개의 질의와 집합 연산자로 표현한 예이며 Example 2.6은 범용 패스 연산자 **A**가 포함된 질의를 두 개의 질의와 집합 연산자로 표현한 예이다.

Example 2.4: 모든 스트림속에 있는 프레임(frame)중에서 처음 프레임은 ''falcon'' 객체를 가지고 있고 마지막 프레임은 ''mountainlion'' 객체를 가지고 있는 일련의 프레임을 찾으시오.

GCalculus/S: {x | (∃s, ∃n0, ∃n1)(Stream(s) ∧ Frame(n0) ∧Frame(n1) ∧

{n0,n1} ⊆s.rep_frame ∧<<s.rep_frame,x,no>>[[n0]]

((∃o1)(C_Object(o1) ∧o1.name=''falcon'' ∧o1 ∈n0.objects) C [[n1]]((∃o2)(C_Object(o2) ∧
o2.name =''moutainlion'' ∧ o2 ∈ n1.objects))}

Example 2.5: 모든 스트림이 객체 "deer"와 객체 "mountain lion"를 가지는 그래프를 찾으시오.

GCalculus/S: {g | Pres_Graph(g) ∧ All_Stream(g) ≡ Streams_Having_Deer_and_Mountainlion(g)}

All_Stream = {s1 | Stream(s1) ∧ s1 ∈g.Nodes ∧ s1.type ≠Terminate}

Streams_Having_Der_and_Mountainlion = {s1 | Stream(s1) ∧ s1 ∈ g.Nodes ∧
(∃o1, ∃o2)(C_Object(o1) ∧ C_Object(o2) ∧ {o1, o2} ⊆s1.rep_frame.object ∧
o1.name = "deer" ∧ o2.name = "mountain lion")}

Example 2.6: 어떤 스트림이 객체 ''deer''를 가지고 있을 때 그 스트림에서 시작하는 모든 패스위에 객체 ''river''를 가지고 있는 스트림이 있는 그래프를 찾으시오.

GCalculus/S: {g | Pres_Graph(g) ∧ (∃s1)(Stream(s1) ∧ s1 ∈ g.Nodes ∧
(∃o1)(C_Object(o1) ∧ o1.name = ''deer'' ∧ o1 ∈ s1.rep_frame.objects ∧
All_Paths_from_S1≡Paths_Having_Deer_and_River))}

All_Paths_from_S1 = {x | (∃s2) (Stream(s2) ∧ s2 g.Nodes ∧
<<g,x,s1>>[[s1]]True C [[s2]]s2.type = ''Terminate'')}

Paths_Having_Deer_and_River={x | (∃s1, ∃s2)((Stream(s2) ∧ Stream(s3) ∧
{s2,s3} ⊆g.Nodes ∧<<g,x,s1>>[[s1]]TrueC[[s2]](o1)(C_Object(o1) ∧
o1.name =''river'' ∧o1 ∈ s2.rep_frame.objects) C [[s3]] s3.type =''Terminate'')}

3. 대수에 의한 GCalculus/S 질의의 처리

GCalculus/S 질의의 처리는 패스 포뮬라를 만족시키는 자유 변수(free variable)를 찾는 것이다. 패스 포뮬라는 어떤 패스에 대한 단정(assertion)을 표현한다. 패스 포뮬라 α의 처리는 α를 만족시키는 패스 P를 찾는 것이다. 즉, 노드들과 시간 연산자들에 의하여 표시되어 있는 포뮬라를 만족시키는 자유 변수들을 찾는 것이다. 예를 들어 GCalculus/S 검색 표현 $\{x \mid \phi(x)\}$의 처리는 $\phi(x)$를 만족시키는 자유 변수 x를 찾는 것이다.

GCalculus/S의 질의 처리를 위하여 O-Algebra[19]를 확장하여 대수에 의한 "bottom-up" 처리 기술을 제공한다. 이를 위해 O-Algebra[19]에서 정의된 연산자들을 이용하여 GCalculus/S 질의 표현에서 나타나는 포뮬라를 만족시키는 모든 노드들을 찾는다. 또한, O-Algebra[19]에서 정의된 연산자 이외에 새로운 연산자를 정의하여 연산자 **X**, **C**, **U**를 사용하여 표현된 노드 사이의 관계를 만족시키는 노드들을 찾고 그 노드들이 만드는 패스를 찾는다.

3.1 O-Algebra

O-Algebra[19]는 객체 지향 데이타베이스의 질의를 위한 객체 대수(object algebra)이다. O-Algebra는 select (σ), join($\bowtie$), project(π), union($\cup$), difference(-)와 같은 전통적인 연산자와 다른 연산자를 가지고 있다. 언급된 전통적인 연산자 이외의 연산자를 간단히 설명한다.

어떤 객체 타입(object type) τ가 주어지면 타입 τ의 **CO**(collection of objects)는 타입 τ에 속하는 인스탄스의 집합이다. O-Algebra 연산자의 피연산자는 **CO**들이다. O-Algebra에서 연산의 결과는 새로이 만들어지는 **CO**이다. 각각의 객체는 (o, v)의 모양을 가진다. o는 객체 id(oid)이며 v는 특성 튜플(property tuple)이라 불리는 튜플이다.

Remap 연산자 $\Re_{(A,A1,A2)}(S)$: S는 CO이며 S의 속성 A와 S에 있는 객체들의 oid간의 관계를 설정한다. 만약 A의 값의 도메인이 bulk 타입이면 단일화(flattened)되며 그림 2 (b)는 $\Re_{(employee,c,e)}$(Companies) 연산의 결과 CO이다. CO Companies에 있는 객체들의 oid와 속성 employee의 값 간의 관계를 설정한다.

Map 연산자 $\varnothing_{(A,A1,A2)}(S)$: S는 CO이며 S의 속성 A와 S에 있는 객체들의 oid간의 관계를 설정한다. 그러나 *Remap*과는 달리 A가 bulk 타입일지라도 단일화 시키지 않는다. 그림 2 (c)는 $\varnothing_{(employees,c,e)}$(Companies)의 결과 CO이다.

Cap 연산자 $\aleph_{A1}(S)$: 오직 S에 있는 객체들의 oid만 획득한다. 즉, A가 S의 한 속성일 때 $\aleph_{A1}(S)$는 $\pi_{A1}(\varnothing_{(A,A1,A2)}(S))$ 이다. 그림 2 (d)는 $\aleph_b(Companies)$에 해당하는 CO이다.

Group 연산자 $\uplus_{An}(S)$: S가 A_1, ... , A_{n-1}, A_n의 속성을 가지고 A_n이 scalar 타입의 속성일 때, A_1, ... , A_{n-1}의 값이 같은 group으로 분할(partition)된다. 그림 2 (e)는 $\uplus_e(\Re_{(employee,c,e)}(Companies))$에 해당하는 CO이다.

Ungroup 연산자 $\oslash_{An}(S)$: S의 속성 A_n이 bulk 타입일 때, A_n의 값을 단일화 시킨다. 연산의 결과로 나타나는

CO의 속성 A_n은 scalar 타입이다. 그림 2 (f)는 $\oslash_e$ ($\emptyset_{(employee,c,e)}$(Companies))에 해당하는 CO이다.

Extend 연산자 $\oplus_{(op,A1,...Ak,A)}(S)$: A_1, .. , A_k는 S의 속성이며 A는 S에 추가되는 속성이다. 추가되는 속성 A의 값은 op(A_1, .. , A_k)이다. 그림 2 (g)는 $\oplus(\bigcup_{employees,manager,workers})$(Companies)에 해당하는 CO이다.

연속의, 때로는 다른 타입의 객체들로 이루어지는 패스를 *hpath*라 부른다. 즉, *hpath*는 일련의 객체(a sequence of objects)들을 나타낸다. 구체적으로 $S[x_1].A_1...A_k[x_{k+1}]$ 이나 $S[x_1].A_1...Ak[x^*_{k+1}]$는 타입이 T인 CO S에서 출발하여 속성 A1, .. , Ak를 따라서 나타나는 일련의 객체로 이루어지는 패스를 나타낸다. 이때 x_1과 x_{k+1}은 선택적인 (optional) 표현 방법이다. $S.A_1....A_k$는 타입 $T.A_1....A_k$의 외연(extent)을 나타낸다. 예를 들어 Department.manager는 "Department"의 속성 "manager"의 객체 타입의 외연인 CO "Manager"를 나탄낸다. 그리고 *hpath* 표현에 따라서 만들어 지는 CO는 다음의 대수 표현에 의하여 만들어 진다.

$CO\_hpath(S[x_1].A_1...A_k[x_{k+1}]) =_{def} \Re(A1,x1,x2)(S) \bowtie \Re_{(A2,x2,x3)}(S.A1) \bowtie$

$\Re_{(A3,x3,x4)}(S.A1.A2) \bowtie ... \bowtie \Re(A_k,x_k,x_{k+1})(S.A_1.A_2.,,.A_{k-1})$

이며 $S[x_1].A_1...A_k[x_{k+1}]$는 bulk 데이타 타입 $T.A_1.A_2...A_k$를 단일화 시킨다. 즉, 위의 연산으로 만들어지는 CO의 속성 x_{k+1}의 값은 scalar 타입이다.

$CO\_hpath(S[x_1].A_1...A_k[x^*_{k+1}]) =_{def} \Re(A1,x1,x2)(S) \bowtie \Re_{(A2,x2,x3)}(S.A1) \bowtie$

$\Re_{(A3,x3,x4)}(S.A1.A2) \bowtie ..... \bowtie \emptyset(A_k,x_k,x_{k+1})(S.A_1.A_2.,,.A_{k-1})$

이며

$S[x_1].A_1...A_k[x_{k+1}]$에서 x_{k+1}의 값이 scalar 타입것과는 달리 bulk 타입이다.

Companies

oid	cname	employees	manager
c01	GE	{p01,p02}	m1
c02	BP	{p03}	m2
c03	BP	{p03	m3

(a) CO Companies

$\Re_{(employee,c,e)}$(Companies)

oid	c	e
n42	c01	p01
n43	c01	p02
n44	c02	p03
n45	c03	p03

(b) Remap

$\emptyset_{(employees,c,e)}$(Companies)

oid	c	e
k23	c01	{p01,p02}
k24	c02	{p03}
k25	c03	{p03}

(c) Map

$\aleph_b$(*Companies*)

oid	b
m34	c01
m35	c02
m36	c03

(d) Cap

$\uplus_e(\Re_{(employee,c,e)}(Companies))$ $\oslash_e(\emptyset_{(employee,c,e)}$(Companies))

oid	c	e
j78	c01	{p01,p02}
j79	c02	{p03}
j80	c03	{p03}

(e) Group

oid	c	e
h45	c01	p01
h46	c01	p02
h47	c02	p03
h48	c03	p03

(f) Ungroup

$\oplus(\bigcup_{employees,manager,workers})$(Companies)

oid	cname	employees	namager	workers
c01	GE	{p01,p02}	{m1}	{p01,p02,m1}
c02	BP	{p03}	{m2}	{p03,m2}
c03	BP	{p03}	{m3}	{p03,m3}

(g) Extend

그림 2

Department

oid	department	manager
c01	sale	m01
c02	DB	m02
c03	network	m03
c04	OS	nil

(a)

Manager

oid	skill	title
m01	4	{T1,T2}
m02	5	{T1}
m03	6	{T2}

(b)

Department[d].manager.title[t]

oid	d	manager	t
n61	c01	m01	T1
n62	c01	m01	T2
n63	c02	m02	T1
n64	c03	m03	T1

(c)

Department[d].manager.title[t*]

oid	d	manager	t*
p73	c01	m01	{T1,T2}
p74	c02	m02	{T1}
p75	c03	m03	{T2}

(d)

그림 3

hpath의 특별한 경우(special case)로서 $S[b_0]$가 있다. b_0는 S에 있는 속성의 이름이며 S에 있는 객체들의 oid를 얻기 위해서 이용된다. $S[b_0]$는 대수 $\aleph_{b0}(S)$에 해당한다. 그림 3의 (c)와 (d)는 각각 hpath Department [d].manager.title[t]와 Department[d].manager.title[t*]에 해당하는 CO이다.

3.2 새로운 연사자의 추가: X^{Next}, XConnect, 그리고 X^{Until}

GCalculus/S에 사용되고 있는 시간 연산자를 대수적으로 처리하기 위해서 그 연산자가 표현하는 관계를 만족시키는 패스를 구축하는 새로운 O-Algebra 연산자가 필요하다. 이제 GCalculus/S에 있는 시간 연산자와 직접적으로 연계되는 set-at-a-time 연산자를 정의한다. 우선 몇가지의 용어 정의를 한다.

(a) 패스 p_1의 마지막 노드에서 패스 p_2의 처음 노드로 에지가 있을 때, p_1은 p_2에 인접(adjacent)하다고 한다.

(b) 패스 p_1이 패스 p_2에 인접 하거나, 패스 p_1이 1개 또는 여러개의 노드로 이루어진 패스 cp에 인접하고 cp는 p_2에 인접하면, p_1은 p_2에 연결(connected)되어 있다고 한다. 그리고 cp는 연결 패스(connecting path)라 한다.

(c) 여러개의 패스 p_1, ... , p_n이 있을 때, $p_1 \cdot \ldots \cdot p_n$ 은 p_1, ... , p_n의 순서로 연결(concatenation)하여 새로운 패스를 건설하는 것을 말한다. 즉, p_i의 마지막 노드에서 p_{i+1}의 처음 노드로 에지를 만들어 p_i를 p_{i+1}에 인접하게 만든다.

이제 패스 연산자에 대응하는 대수 연산자를 정의한다. 각 연산자의 피연산자는 CO이다.

MakeNext 연산자 ($X^{Next}_{(ai,aj,ak)}(S)$): S는 CO이며 ai와 aj는 S의 두 속성이며 리스트 타입(list type)이다. 리스트 타입은 스칼라 타입으로 생각한다. MakeNext 연산자는 S의 속성 개수보다 1개가 많은 속성을 가진 CO를 만들며 새롭게 첨가되는 속성은 ak이다. p_i와 p_j는 S에 있는 튜플(Tuple) t의 ai와 aj의 값이며 pi가 pj에 인접하고 있으면 ak의 값은 $p_i \cdot p_j$가 된다. 그렇지 않으면 t는 MakeNext연산의 결과 CO에 나타나지 않는다.

MakeConnect 연산자 ($X^{Connect}_{(ai,aj,ak)}(S)$): S는 CO이며 ai와 aj는 S의 두 속성이며 리스트 타입(list type)이다. MakeConnect 연산자는 S의 속성 개수보다 1개가 많은 속성을 가진 CO를 만들며 새롭게 첨가되는 속성은 ak이다. p_i와 p_j는 S에 있는 튜플(Tuple) t의 ai와 aj의 값이며 pi가 pj에 연결되어 있고 cp가 연결 패스이면 ak의 값은 $pi \cdot cp \cdot pj$ 가 된다. 그렇지 않으면 t는 MakeNext연산의 결과 CO에 나타나지 않는다. 연결 패스가 1개 이상일 때 각각의 연결 패스에 대해서 상응하는 튜플이 결과 CO에 나타난다.

Rename 연산자 ($Rename_{(ai,aj)}(S)$): S는 CO이며 a_i는 S에 있는 속성이다. Rename연산자는 속성 a_i가 a_j로 대체된다.

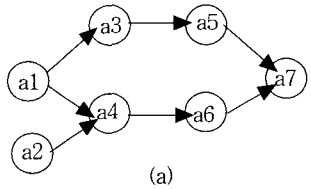
MakeUntil 연산자 ($X^{Until}_{(ai,aj,ak)}(S)$): S는 CO이며 ai와 aj는 S의 두 속성이며 리스트 타입(list type)이다. MakeUntil 연산자는 S의 속성 개수보다 1개가 많은 속성을 가진 CO를 만들며 새롭게 첨가되는 속성은 ak이다. 새롭게 만들어 지는 CO는

$P1 \bigcup \ldots \bigcup Pn$이며 $P1 = X^{Next}_{(ai,aj,ak)}(S)$,

$Pi = (Rename^{(p*,ak)}(\pi_{a1,\ldots,ai,aj,p*}X^{Next}_{(ai,aj,p*)}(S \bowtie \pi_{(aj,ak)}(P_{i-1}))))$

이며 P_{n+1}에는 튜플이 없다.

그림 4에서 (a)는 어떤 상연 그래프이며 (b)는 그 상연 그래프의 노드의 정보를 가지고 있는 CO Pres_Node이다. 그림 5는 CO Nodes와 연산자 X^{Next}, $X^{Connect}$, X^{Until} 연산을 그림 4에 있는 상연 그래프를 가지고 처리한 결과이다.



(a)

oid	child-nodes
a1	{a3,a4}
a2	{a4}
a3	{a5}
a4	{a6}
a5	{a7}
a6	{a7}
a7	nil

(b)

그림 4

oid	s1	s2
o56	a1	a7
o57	a3	a7
o58	a4	a7
o59	a5	a7
o60	a1	a6
o61	a3	a6
o62	a4	a6
o63	a5	a6

(a) Nodes

oid	s1	s2	p1
m23	a5	a7	<a5,a7>
m24	a4	a6	<a4,a6>

(b)$X^{Next}_{(s1,s2,p1)}$(Nodes)

oid	s1	s2	p2
h3	a1	a7	<a1,a3,a5,a7>
h4	a1	a7	<a1,a4,a6,a7>
h5	a3	a7	<a3,a5,a7>
h6	a4	a7	<a4,a6,a7>
h7	a5	a7	<a5,a7>
h8	a1	a6	<a1,a4,a6>
h9	a4	a6	<a4,a6>

(c)$X^{Connect}_{(s1,s2,p2)}$(Nodes)

oid	s1	s2	p3
q2	a5	a7	<a5,a7>
q3	a4	a6	<a4,a6>
q4	a3	a7	<a3,a5,a7>
q5	a1	a6	<a1,a4,a6>
q6	a1	a7	<a1,a3,a5,a7>

(d)$X^{Until}_{(s1,s2,p3)}$(Nodes)

그림 5

3.3 GCalculus/S에서 O-Algebra로의 변환

GCalculus/S 질의의 대수적 처리는 "안에서 바깥쪽"("inside-out")방향을 따라 이루어 진다. 즉, 처리는 안쪽의 기본 포뮬라(atomic formula)에서부터 시작하여 바깥쪽으로 확장한다. 예를 들어 아래의 GCalculus/S 질의를 보자.

{x | (∃g)((∃s0, s1, s2, s3, s4)(<<g, x, s0>>((([[s0]] α X [[s1]] β) U [[s2]] γ) C ([[s3]] δ X [[s4]] ε)))))}

F1: ([[s0]] α X [[s1]] β), F2: F1 U [[s2]] γ,

F3: ([[s3]] δ X [[s4]] ε), F4: F2 C F4

위의 GCaluclus/S 질의는 [[s0]] α, [[s1]] β, F1, [[s2]] γ, F2, [[s3]] δ, [[s4]] ε, F3, F4의 순서로 이루어진다.

3.3.1과 3.3.2에서 GCalculus/S에 사용된 연산자에 대한 귀납적인 증명을 통하여 GCalculus/S 질의가 대수적 표

현으로 전환할수 있음을 증명한다. 3.3.3에서는 질의를 처리할 때 실제로 사용될수 있는 데이타베이스 도메인을 이용하여 대수적 표현으로 전환할수 있음을 증명한다.

3.3.1 기본 포뮬라(Atomic Formula)의 대수로의 변환

GCalculus/S 질의의 예를 가지고 기본 포뮬라를 대수 표현으로 바꾸는 방법을 설명한다. "상연 그래프의 이름이 "wilderness"이며 그 상연 그래프에 있는 인접한 두 개의 노드를 찾으시오. 처음 노드의 이름은 "Yosemite"이며 두 번째 노드의 이름은 "bird"이다. 처음 노드의 상연 시간이 두 번째 노드의 상연 시간보다 길다."

{x | (∃g)(Pres_Graph(g) ∧ g.name="wilderness" ∧ Stream(s1) ∧ Stream(s2) ∧

{s1,s2} ⊆g.node ∧ <<g,x,s1>>(([[s1]]s1.name="Yosemite") **X**

([[s2]]s2.name="bird") ∧ (s1.duration<s2.duration))}

위에 GCalculus/S 질의는 기본 포뮬라 Pres_Graph(g), Stream(s1), Stream(s2)를 가지며 g.name, {s1,s2}, g.node, s1.name, s2.name, s1.duration, s2.duration의 항을 가진다. 각각의 기본 포뮬라와 항에 해당하는 대수적 표현은 아래와 같다.

포뮬라	대수 표현
Pres_Graph(g)	$\aleph_g$(Pres_Graph)
g.name	$\mathcal{R}_{(name,g,n)}$(Pres_Graph)
Stream(s1)	$\aleph_{s1}$(Stream)
Stream(s2)	$\aleph_{s2}$(Stream)
{s1,s2}	$\bigoplus(\bigcup_{,s1,s2,y)}(\aleph_{s1}$(Stream)X $\aleph_{s2}$(Stream))
g.node	$\varnothing_{(node,g,s*)}$(Pres_Graph)
s1.name	$\mathcal{R}_{(name,s1,n1)}$(Stream)
s2.name	$\mathcal{R}_{(name,s2,n2)}$(Stream)
s1.duration	$\mathcal{R}_{(duration,s1,d1)}$(Stream)
s2.duration	$\mathcal{R}_{(duration,s2,d2)}$(Stream)

3.3.2 GCalculus/S에서 대수로의 변환

GCalculus/S에 있는 어떠한 포뮬라도 대수적 표현으로 변환될 수 있음을 보인다. 이 증명은 GCalculus/S에 사용되는 연산자에 대한 귀납적인 증명으로 보인다. 우선 GCalculus/S 질의가 처리되는 영역(domain)을 정의한다. 큐리 {t | ϕ}에 대하여 DOM(ϕ)는 ϕ안에 있는 모든 변수의 값(value)이 될수 있는 기호(symbol)들의 집합이라 정의한다.

Definition: ϕ가 GCalculus/S 질의에 있는 포뮬라라 하자. 그러면 DOM(ϕ)는 다음과 같이 정의된다. T1,..,Tn과 {t1,...,tn}이 각각 ϕ안에 있는 항과 상수라 하자. 그리고 Ei는 Ti에 해당하는 대수적 표현이라 하며 COi는 Ei에 해당하는 CO라 한다. 각각의 COi는 속성 i1,...,ik를 가진다. 그리고 S(Ri)는 πi1(Ei) ⋃ ... ⋃ πik(Ei)라 하자. 그러면 DOM(ϕ)는 S(R1) ⋃ ... ⋃ S(Rn) ⋃ {t1,...,tm} 이다.

CO는 속성들을 가지고 있다. CO1(s1,...,sn)은 속성 s1,...,sn을 가지고 있고 이름이 CO1인 CO를 나타낸다. 모든 기본 포뮬라에 대하여 대수 표현이 존재하며 그 대수 표현에 상응하는 CO가 있다. 따라서 서브포뮬라와 연산자를 가지고 만드어지는 포뮬라에 상응하는 대수적 표현이 존재하며 그 대수적 표현에 상응하는 CO가 존재한다.

드모르간의 법칙(DeMorgan's law)을 이용하여 모든 포뮬라를 논리곱 연산자 ∧ 가 없는 포뮬라로 바꿀수 있다.

Theorem 3.1: 모든 GCalculus/S 질의에 대하여 동등한(equivalent) 대수 표현이 존재한다.

Proof: {t | ω}를 GCalculus/S 질의라 하자. 드모르간의 법칙과 [27]에서 변환에이용된 법칙을 이용하면 일반성을 상실함이 없이(without loss of generality) ω는 ∧, ∀와 **A**를 가지고 있지 않다고 가정할 수 있다. 그러면 포뮬라 ω는 ∃, ¬(오직 base predicate의 왼쪽에만), **E**, ∨, 연산 비교 연산자, 집합 비교 연산자와 시간 연산자를 가진다. E는 DOM(ω)에 상응하는 대수 표현이라 하고 E^k는 E X ... X E(k times)를 나타낸다. 이제 포뮬라 ω에 있는 항들과 서브포뮬라에 대한 induction을 이용하여 theorem을 증명한다.

Basis

항(Terms):

Case (a): v, v가 변수일 때.

항 v는 DOM(ω)에 상응하는 대수 표현 E로 변형한다.

Case (b): {v_1, ... , v_n}, v_i가 변수일 때.

항 {v_1, ... , v_n}는 $\bigoplus(\bigcup_{,v1,...,vn,y)}(E^n)$으로 변형된다.

Case (c): v.A_1. ... .A_k, v는 base predicate S(v)에 사용된 변수이며 A_i는 속성 이름일 때.

항 v.A_1. ... .A_k는 CO_hpath(S[v].A_1. ... .A_k)로 변형된다.

기본포뮬라(Atomic Formulas):

Case(a): ω=S(v), S는 base predicate이며 v는 변수일 때.

S(v)는 v가 클래스 S의 멤버임을 말한다. 기본포뮬라 ω는

$\aleph_v$(S)로 변형한다.

Case(b): ω=x θ y, x와 y는 변수혹은 항이며 θ는 산술 비교 연산자 혹은 집합 비교 연산자일 때.

$PE_x(u_1,...,u_p,u_x)$와 $PEy(v_1,...,v_q,v_y)$가 x와 y에 상응하는 대수 표현이다. u_i, v_j, u_x와 v_y는, $1 \le i \le p$, $1 \le j \le q$, PE_x와 PE_y에 상응하는 CO의 속성의 이름이다. u_x와 v_y는 x와 y에 상응하는 값을 가지는 속성이다. 만약 x와 y가 $v_1.A_1$. ... A_p와 $v_2.A_1$. ... A_q의 형태이고 v1=v2이면 포뮬라 ω는 $\sigma_{(ux\ \theta\ uy)}(PEx \bowtie PEy)$로 변형되며 이외의 경우에는 $\sigma_{(ux\ \theta\ uy)}(PEx \times PEy)$로 변형된다.

Case(c): ω=x θ t, x는 변수이며 t는 상수일 때.

PEx(u1,...,up,ux)가 x에 상응하는 대수 표현이라 하자. u_i, $1 \le i \le p$와 u_x는 x에 상응하는 CO의 속성의 이름이며 u_x는 x에 상응하는 값을 가지는 속성이다. 그러면 포뮬라 ω는 $\sigma_{(ux\ \theta\ t)}(PE_x)$로 변형된다.

Induction

항(Terms): $\omega=\{t_1, ... ,t_m \mid \omega_w\}$, t_j는, $1 \le j \le m$, 변수일 때.

$PE_w(q_1,...,q_m,p_1,...,p_u,k_1,...,k_v)$는 ω_w에 상응하는 대수적 표현이라 하자. q_i는 PE_w에 상응하는 CO에 있는 속성이며 그 값들은 t_i에 해당한다. 속성 p_i은 포뮬라 $\{t_1, ... ,t_m \mid \omega_w\}$ 바같에서 정의된 변수에, 예를 들어 V, 상응한다. PE_{wi}는 $\uplus_{(p1,...,pu)}(\pi_{qi,p1,...,pu}(PE_w))$이라 하자. 그러면 PE_{wi}는 각각의 튜플은 변수 t_i의 값과 포뮬라 ω_w를 만족시키는 변수 V의 값을 가지는 CO에 상응하는 대수적 표현이다. q_i의 타입은 set 타입이다. 만약 $m \ge 2$이면 포뮬라$\{t_1, ... ,t_m \mid \omega_w\}$는

$\pi_{p1,...,pu,y}(\bigoplus(\bigcup_{q1,...,qm,y})(PE_{w1} \bowtie ... \bowtie PE_{wm})$으로 변형되며 이외의 경우, 즉, m=1인 경우 포뮬라 $\{t_1 \mid \omega_w\}$는 $\pi_{p1,...,pu,q1}(PE_{w1})$으로 변형된다.

포뮬라(Formulas)

Case(a): $\omega = \omega_1 \vee \omega_2$

이 경우의 증명은 [27]에서 관계 해석을 관계 대수로 변형시키는 theorem의 증명 기법을 사용한다. $PE_1(u_1,...,u_n)$과 $PE_2(v_q,...,v_p)$가 ω_1과 ω_2에 해당하는 대수 표현이라 하자. 그리고 PE_1^*는 $\pi_{i1,...,im}(PE_1 \times E^{m-n})$으로 정의한다. 이때 $u_q = y_j$인 u_q가 존재한다면 i_j은 q에 상응하며 그렇지 않으면 n+1과 m사이에 존재하는 유일한 수(unique integer)이다. 마찬가지의 방법으로 PE_2^*를 정의한다. 즉, PE_2^*는 $\pi_{j1,...,jm}(PE_2 \times E^{m-p})$로 정의되며 $v_q = y_j$인 v_q가 존재한다면 j_i은 q에 상응하며 그렇지 않으면 j_i은 p+1과 m사이에 있는 유일한 수이다. 그러면 포뮬라 ω는 $PE_1^* \times PE_2^*$로 변형된다.

Case(b): $\omega = \neg \omega_1$

$PE_1(u_1,...,u_m)$과 E가 ω_1과 $DOM(\omega_1)$에 상응하는 대수 표현이라 하자. 그러면 포뮬라 ω는 $E^m - PE_1$으로 변형된다.

Case(c): $\omega = \omega_1 \Theta \omega_2$, Θ가 **X**일 때.

$PE_1(u_1,...,u_n)$과 $PE_2(v_1,...,v_p)$는 ω_1과 ω_2에 상응하는 대수 표현이라 하자. u_i와 v_j는 같은 이름일지도 모른다. $\omega_1 \wedge \omega_2$(즉, $\neg(\neg\omega_1 \vee \neg\omega_2)$에 상응하는 대수 표현 PE를 구한다. 그러면 PE는 튜플들이 PE_1과 PE_2를 동시에 만족시키는 CO와 상응한다. 그러면 포뮬라 ω는 $X^{Next}_{(p1,p2,p3)}(PE)$(이 표현을 PE^*라 하자.)로 변형된다. p1과 p2는 값이 포뮬라 ω_1과 ω_2에 해당하는 패스인 속성이다. 만약 ω가 다른 패스 포뮬라의 서브포뮬라가 아니라면

$\uplus_{ui}(\pi_{ui,p3}(PE^*)) \bowtie PE^*$ 연산을 수행한다. 이때 u_i는 포뮬라 ω에 해당하는 패스의 처음 노드이다. 이 연산은 PE^*와 같은 속성들을 가진 CO를 만든다. 그러나 p_3의 값들은 동일한 처음 노드를 가지는 패스의 집합이다.

Case(d): $\omega = \omega_1 \Theta \omega_2$, Θ가 C일 때.

모든 논의(argument)는 (c)의 경우와 동일하다. 단지 PE^*는 $X^{Connect}_{(p1,p2,pe)}(PE)$에 의하여 구한다.

Case(e): $\omega = \omega_1 \Theta \omega_2$, Θ가 U일 때.

모든 논의(argument)는 (c)의 경우와 동일하다. 단지 PE^*는 $X^{Until}_{(p1,p2,pe)}(PE)$에 의하여 구한다.

Case(f): $\omega = S_1 \subseteq S_2$, S_i는 $\{s_1,..,s_n\}$, $v.A_1$. ... $.A_k$ 또는 $\{t_1,...,t_m \mid \omega_m\}$의 형태이며 s_i, v, t_j는 변수일 때.

$\{s_1,...,s_n\}$, $v.A_1$. ... $.A_k$, $\{t_1,...,t_m \mid \omega_m\}$에 해당하는 대수 표현을 구하는 방법은 앞의 경우에서 설명되었다. $PE_1(y_1,a_1,...,a_m)$과 $PE_2(y_2,b_1,...,b_n)$가 S_1과 S_2에 상응하는 대수 표현이라 하자. y_1과 y_2는 S_1과 S_2에 상응하는 값을 가지는 속성이다. 만약 S_1과 S_2사이에 공통의 속성이 없다면 포뮬라 ω는 $\sigma_{(y1 \subseteq y2)}(PE_1 \times PE_2)$로 변형되며 공통의 속성이 존재하면 ω는 $\sigma_{(y1 \subseteq y2)}(PE_1 \bowtie PE_2)$로 변형된다.

Case(g): ω = S1 $\subset$ S2

연산자가 $\subseteq$대신에 $\subset$가 사용된 것만 제외하고 모든 논의는 동일하다. 즉 포뮬라 ω는 $\sigma_{(y1 \subseteq y2)}(PE_1 \times PE_2)$또는 $\sigma_{(y1 \subseteq y2)}(PE_1 \bowtie PE_2)$로 변형된다.

Case(h) $\omega = v \in S$, S는 $\{v_1,...,v_m\}$, $u.A_1$. ... $.A_k$혹은 $\{t \mid w_1\}$의 형태이며 v, v_i, u는 변수일 때.

$PE_v(p_v)$와 $PE_s(p_s,p_1,...,p_k)$가 v와 S에 상응하는 대수 표현이라 하자. p_v와 p_s는 PE_v와 PE_s에 상응하는 CO에서 v와 S에 상응하는 값을 가지는 속성이다. 그러면 포뮬라 ω는 $\sigma_{(pv \in ps)}(PE_v \times PE_s)$로 변형된다.

아래에 GCalculus/S 질의를 대수를 이용하여 처리하는 예를 보인다.

Example 3.1: 이름(name)이 "deer"인 스트림을 가지는

상연물 (그래프)를 찾으시오

{g | Pres_Graph(g) ∧ (∃s)(Stream(s) ∧ {s} ⊆g.nodes ∧ s.name = "deer")}

포뮬라	대수 표현
Pres_Graph(g)	PE1(g): ℵ g(Pres_Graph)
Stream(s)	PE2(s): ℵ s(Stream)
{s}	PE3(s): ℵ s(Stream)
g.nodes	PE4(g,no): CO_hpath(Pres_Graph[g].nodes[no*])
s.name	PE5(g,na): CO_hpath(Stream[s].nodes[na])

위의 GCalculus/S 질의의 전체 포뮬라, f, Pres_Graph(g) ∧ (∃s)(Stream(s) ∧ {s} ⊆g.nodes ∧ s.name = "deer")는 ¬(¬Pres_Graph(g) ∨ ¬((∃s)(¬(¬Stream(s) ∨(¬({s} ⊆g.nodes) ∨ ¬(s.name = "deer"))))))로 변환된다.

F1: ¬(s.name = "deer"), F2: ¬({s} ⊆g.nodes),

F3: F1 ∨ F2, F4: ¬Stream(s), F5: ¬(F4 ∨ F3). F6: ¬((∃s)(F5), F7: ¬Pres_Graph(g),

F8: ¬(F7 ∨ F6)

이제 theorem에서 설명된 변환 법칙을 적용한다. F1에 해당하는 대수 표현부터 시작하여 차례로 F8에 해당하는 대수 표현을 만든다. 이 처리에서 사용되는 데이타 베이스 도메인(domain), E_i은 DOM(f)이며 π_g(PE1) ∪ π_s(PE2) ∪ π_s(PE3) ∪ π_g(PE4) ∪ π_{no}(PE4) ∪ π_s(PE5) ∪ π_{na}(PE5) ∪ {deer}이다.

F1: E^2 - $\sigma_{(na=deer)}$(PE1) = PE6(s,na)

F2: E^3 - $\pi_{s,g,no}$($\sigma_{(s \subseteq no)}$(PE3 X PE4)) = PE7(s,g,no)

F3: E^4 - ($\pi_{1,2,3,4}$(PE7 X E) ∪ $\pi_{1,3,4,2}$(PE6 X E2)) = PE8(s,g,no,na)

F4: E - PE2 = PE9(s)

F5: E^4 - ($\pi_{1,2,3,4}$(PE9 X E3) ∪ $\pi_{1,2,3,4}$(PE8)) = PE10(s,g,no,na)

F6: E^4 - PE10 = PE11(s,g,no,na)

F7: E - PE1 = PE12(g)

F8: E4 - $\pi_{2,1,3,4}$(PE12 X E3) ∪ $\pi_{1,2,3,4}$(PE11) = PE13(s,g,no,na)

GCalculus/S 질의의 결과: π_g(PE13)

3.4 DOM(ω)을 사용하지 않는 변환

3.3.2에서 GCalculus/S 퓨리 {t | ω}를 대수 표현으로 변환시키는 방법을 기술 하였다. 그리고 그 퓨리가 처리되는 대상이 되는 데이타 베이스로서 DOM(ω)를 이용하였다. 그러나 DOM(ω)의 크기는 매우 방대할 수 있고 이 처리 방법은 비실용적일 수 있다.

GCalculus/S의 포뮬라에 있는 각각의 변수들은 그 변수들이 속하는 클래스에 의하여 구속(restricted)받는다. 즉, 변수 v가 GCalculus/S에서 사용될 때 기본 포뮬라 S(v)의 표현이 GCalculus/S 표현에 항상 존재한다. GCalculus/S의 처리를 위하여 두 번의 패스(two passes)를 가정하면 각 변수의 영역(range)를 결정할수 있으며 이는 DOM(ω)를 사용하지 않고 질의를 처리할 수 있는 가능성을 제공한다.

포뮬라 $\omega = \omega_1 \vee \omega_2$를 대수 표현으로 변환시켜 보자. E를 {*nil*}이라 하자. *nil*은 무 객체(no object) 혹은 {}를 나타낸다. $PE_1(u_1,...,u_n)$과 $PE_2(v_1,...,v_p)$는 ω_1과 ω_2에 상응하는 대수 표현이라 하자. 그리고 PE_1^*를 $\pi_{i1,...,im}(PE_1 \times E^{m-n})$이라 정의한다. 만약 $u_q = y_l$인 u_q가 존재한다면 i_l은 q에 상응하며 그렇지 않으면 i_l은 n+1과 m 사이에 존재하는 유일한 수(*unique integer*)이다. 유사한 방법으로 PE_2^*는

$\pi_{j1,...,jm}(PE_2 \times E^{m-p})$이다. 만약 $v_q = y_l$인 v_q가 존재한다면 j_l은 q에 상응하며 그렇지 않으면 j_l은 p+1과 m 사이에 존재하는 유일한 수(unique integer)이다. 그러면 ω는 $PE_1^* \cup PE_2^*$로 변환된다.

포뮬라 $\omega = \omega_1 \wedge \omega_2$를 대수 표현으로 변환시키기 위하여 $PE_1(u_1,...,u_n)$과 $PE_2(v_1,...,v_p)$는 ω_1과 ω_2에 상응하는 대수 표현이라 하자. 만약 똑같은 이름을 가지는 속성 ui와 vj가 존재한다면 ω는 $PE_1 \bowtie PE_2$로 변환되며 이외의 경우에는 $PE_1 \times PE_2$로 변환된다.

포뮬라 $\omega = \neg \omega_1$일 때 PE $\omega_1(u_1,...,u_m)$이 $\omega_1(u_1,...,u_m)$에 상응하는 대수 표현 이라 하자. u_i는 클래스 S_i에 상응하며 PE_i는 대수 표현 $S_i(u_i)$에 상응한다고 가정하자. 그러면 ω는 $(PE_1 \times ... \times PE_m) - \pi_{u1,...,um}(PE\ \omega_1)$으로 변환된다.

포뮬라 $\omega = \omega_1 \Theta \omega_2$, Θ가 **X**일 때 $PE_1(u_1,...,u_n)$과 $PE_2(v_1,...,v_p)$는 ω_1과 ω_2에 상응하는 대수 표현이라 하자. ui와 vj는 같은 이름을 가질수 있다. 그러므로 우선 $\omega_1 \wedge \omega_2$에 상응하는 대수 표현을 구한다. 그 대수 표현을 PE라 하자. 그러면 ω에 상응하는 대수 표현은 $X^{Next}_{(p1,p2,p3)}(PE)$이다. 이 대수 표현을 PE^*라 하자. 만약 ω가 패스 표뮬라의 서브표뮬라가 아니라면 $\uplus_{ui}(\pi_{ui,p3}(PE^*)) \bowtie PE^*$를 수행한다. u_i는 포뮬라 ω가 나타내는 패스의 처음 노드에 상응하는 속성이다. 이 대수의 수행은 PE^*와 같은 속성을 가지는 CO를 만든다. 그러나 p_3속성의 값은 같은 노드에서 시작하는 패스의 집합이다.

포뮬라 $\omega = \omega_1 \Theta \omega_2$, Θ가 **C**혹은 **U**인 경우 모든 논의는 Θ가 **X**인 경우와 동일하다. 단지 PE^*가 각각

$X^{Connect}_{(p1,p2,p3)}$(PE)와 $X^{Until}_{(p1,p2,p3)}$(PE)에 의하여 구해진다.

Example 3.2: 스트림의 이름이 "deer"또는 "buffallo"인 스트림을 가지는 상연물을 찾으시오

{g | Pres_Graph(g) ∧ (∃s)(Stream(s) ∧ (s.name = "deer" ∨ s.name = "buffallo") ∧ s ∈ g.nodes)}

F1: s ∈ g.nodes, F2: s.name = "buffallo", F3: s.name = "deer",

F4: F3 ∨ F2, F5: Stream(s) ∧ F4 ∧ F1

F6: Pres_Graph(g) ∧ (∃s)(F5)

포뮬라	대수 표현
Pres_Graph(g)	PE1(g): $\aleph_g$(Pres_Graph)
Stream(s)	PE2(s): $\aleph_s$(Stream)
s.name	PE3(s,na1): CO_hpath(Stream[s].name[na1])
s.name	PE4(s,na2): CO_hpath(Stream[s].name[na2])
s	PE5(s): $\aleph_s$(Stream)
g.nodes	PE6(g,no): CO_hpath(Pres_Graph[g].nodes[no^*])

앞으로 언급되는 대수 표현에서 E는 {nil}을 나타낸다.

F1 PE7(s,g,no): $\sigma_{(s \in no)}$(PE5 X PE6)

F2 PE8(s,na2): $\sigma_{(na2=buffalo)}$(PE4)

F3 PE9(s,na1): $\sigma_{(na1=deer)}$(PE3)

F4 PE10(s,na1,na2): $\pi_{1,2,3}$(PE9 X E) ∪ $\pi_{1,3,2}$(PE8 X E)

F5 PE11(s,na1,na2,g,no): PE2 ⋈ F4 ⋈ F1

F6 PE12(s,na1,na2,g,no): PE1 ⋈ PE11

GCalculus/S 큐리의 결과: π_g(PE12)

Example 3.3: 한 스트림의 상연 시간이 15이며 그 스트림에 인접하고 있는 모든 스트림의 상연 시간은 15보다 큰 스트림들을 가지는 상연물을 찾으시오.

{g | Pres_Graph(g) ∧ (∃s1, ∃s2, ∃sn)(Stream(s1) ∧ Stream(s2) ∧ Stream(sn) ∧ {s1,s2,sn} ⊆ g.nodes ∧

{x1|E<<g,s1,x1>>[[s1]](s1.duration=15) **C** [[sn]]sn.type = Terminate} ⊆

{x2|E<<g,s1,x2>>[[s1]](s1.duration=15)**X**[[s2]](s1.duration < s2.duration) **C** [[sn]]sn.type = Terminate))}

F1: **X** [[s2]](s1.duration < s2.duration) **C**

[[sn]]sn.type = Terminate

F2: {x2 | F1}

F3: E<<g,s1,x1>>[[s1]](s1.duration=15) **C** [[sn]]sn.type = Terminate

F4: {x1 | F3}

F5: {s1,s2,sn} ⊆ g.nodes

포뮬라	대수 표현
Pres_Graph(g)	PE1(g): $\aleph_g$(Pres_Graph)
Stream(s1)	PE2(s1): $\aleph_{s1}$(Stream)
Stream(s2)	PE3(s2): $\aleph_{s2}$(Stream)
Stream(sn)	PE4(sn): $\aleph_{sn}$(Stream)
{s1,s2,sn}	PE5(s1,s2,sn,s): $\oplus(\bigcup,s1,s2,sn,s)$ (PE2(s1) X PE3(s2) X PE4(sn))
g.nodes	PE6(g,no): CO_hpath(Pres_Graph[g].nodes[no^*])
s1.duration	PE7(s1,d1): CO_hpath(Stream[s1].duration[d1])
s2.duration	PE8(s2,d2): CO_hpath(Stream[s2].duration[d2])
sn.type	PE9(sn,tn): CO_hpath(Stream[sn].type[tn])

PE10(s1,d1,s2,d2,p1,sn,x2):$X^{Connect}_{(p1,sn,x2)}(X^{Next}_{(s1,s2,p1)}(\sigma_{(d1=15)}$(PE7 ⋈ $\sigma_{(d1<d2)}$(PE7 X PE8)) X (PE4 ⋈ $\sigma_{(tn=Terminate)}$(PE9))

F1 PE11(s1,d1,s2,d2,p1,sn,x2): $\uplus$s1($\pi_{s1,x2}$(PE10)) ⋈ PE10

F2 PE12(s1,s2,sn,x2): $\pi_{s1,s2,sn,x2}$(PE11)

PE13(s1,d1,sn): $X^{Next}_{(s1,s2,x1)}(\sigma_{(d1=15)}$(PE6) X PE4)

F3 PE14(s1,d1,sn,x1): $\uplus$s1($\pi_{s1,x1}$(PE13)) ⋈ PE13

F4 PE15(s1,sn,x1): $\pi_{s1,sn,x1}$(PE14)

F4 ⊆ F2 PE16(s1,sn,x1,s2,x2): $\sigma_{(x1 \subseteq x2)}$(PE15 ⋈ PE12)

F5 PE17(s1,s2,sn,s,g,no*): $\sigma_{(s \subseteq no)}$(PE5 X PE6)

GCalculus/S 질의에 있는 전체 포뮬라 PE18(g,s1,s2,sn,s,no^*,x1,x2): PE1 X PE2 X PE3 ⋈ PE17 ⋈ PE16

GCalculus/S 질의의 결과: π_g(PE18)

4. 관련 연구

데이타베이스에서 그래프는 여러 상황(different contexts)에서 여러 연구자들의 관심이 되어 왔다. 이에 DAG로 표현되는 멀티미디어 상연물 질의어인 GCalculus/S와 그래프 타입으로 표현되는 자료를 검색하는 다른 질의어의 표현 능력(expressive power)의 차이점은 다음의 네 가지의 관점에서 비교된다. 첫째, 분할화에서 발생하는 여러 계층을 포함하는 표현이 가능한가, 둘째, 분기를 표현할 수 있는가, 셋째, 그래프의 패스를 구성하는 노드들의 순서를 표현할 수 있는가, 마지막으로 패스 정량자 **A**를 가지는 표현을 할 수 있는가 하는 관점이다. 언급된 네 가지의 관점에 해당하는 실제적인 검색의 보기를 언급된 순서대로 그림 6을 이용하여 예시한다. 한편, GCalculus/S의 패스 표현 능력과 정규식 표현 능력의 차이는 [17]에 기술되어 있다.

Example 4.1: 노드를 찾으시오. 노드는 내용 "b"를 만

족하고 그 노드 안에서 내용 "p" 다음에 내용 "q"가 나타난다.

Example 4.2: 노드를 찾으시오. 노드는 내용 "c"를 만족하고 그 노드에서 분기되는 두 노드는 각각 "e"와 "f"를 만족한다.

Example 4.3: 패스를 찾으시오. 패스의 처음 노드는 "a"를 만족하고, 다음 노드는 "b", 다음에는 "c"를 만족하는 노드들이 "d"를 만족하는 패스의 마지막 노드가 나타나기 전까지 계속하여 나타난다.

Example 4.4: 노드를 찾으시오. 그 노드는 "a"를 만족하며 그 노드 다음에 있는 모든 노드는 "b"를 만족한다.

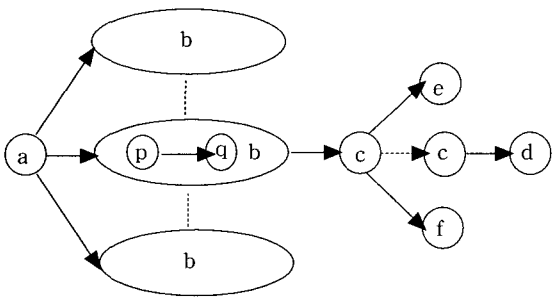


그림 6

G+[6,7]와 Hy+[4,5]와 같은 시스템에서는 데이타베이스는 그래프로 시각화(visualization)되고 질의는 데이타베이스를 대상으로 패턴을 비교하는 그래프의 집합으로 표시된다. 노드들 사이의 관계를 나타내기 위하여 여러 종류의 에지를 사용한다. Example 4.1, 4.2, 4.3에 해당하는 검색 표현이 가능하다. 그러나 Example 4.4에 해당하는 검색 표현은 불가능하다.

Graph-Oriented Object Database 모델[9,10,11]에서는 데이타베이스 스킴(schme)과 데이타베이스의 사례(instance)를 그래프로 표현하는 것에 주안점이 있다. 질의는 5개의 그래프 연산(노드와 에지 첨가(addition), 제거(deletion), 중복 제거(duplicate elimination))을 이용하여 표현되다. 또한 질의와 결과의 시각화를 위한 비쥬얼 사용자 인터페이스가 있다. 이 모델에서는 객체는 그래프의 노드로 표현되며 객체간의 관계와 객체의 성질은 에지로 표현된다. 비쥬얼 인터페이스가 있는 G+[6,7]와 Hy+[4,5]처름 Example 4.1, 4.2, 4.3의 표현은 가능하나 Example 4.4.의 표현은 불가능하다.

그래프가 데이타를 모델화하기 위해서 필요한 응용분야에서는 그래프를 데이타 모델과 질의에서 명시적으로(explicitly) 지원하는 것이 매우 유용하다. 명시적으로 그래프를 모델화하고 그래프를 검색하는 GraphDB[12]가 좋은 예이다. GraphDB에서는 세가지의 클래스(simple classes, link classes, path classes)가 있다. 링크 클래스는 두 개의 심플 클래스를 "소스(source)"와 "타켓(target)"으로 정하여 연결한다. 패스 클래스는 링크 클래스를 이용한 정규식으로 표현된다. 즉, 패스가 명시적으로 데이타 베이스 시스템안에 저장되어 있다. 검색을 위하여 "on...where...derive" 구절을 제공한다. 내포된 서브질의(nested subquery)의 표현은 없다. 그리고 객체 지향 데이타 모델을 이용하기 때문에 여러 계층을 포함한 질의 표현이 가능하다. 즉, Example 4.1, 4.2, 4.3, 4.4의 표현이 모두 가능하다. 그러나 복잡한 질의는 여러 과정을 거쳐 표현되고 각 과정에서 여러 심플, 링크, 패스 클래스가 표현되어야 한다. 그리고 이 모든 과정은 "derive" 구문에 표현되기 때문에 질의 표현은 혼란스럽거나 복잡할 가능성이 매우 높다.

시간 논리에서는 *next, until, eventually*와 같은 시간 연산자들을 사용하여 시간의 변화에 따른 사실(truth)의 변화를 표현한다. Hierarchical Temporal Logic (HTL)[20,25]에서 위의 연산자들은 비디오를 유사성에 입각한 질의에 이용한다. 여러 계층에 걸친 검색 표현이 가능하나 분기(branch)를 표현하는 방법은 없다. 이것은 이 논문에서 다루고 있는 DAG에 관한 검색 표현을 불가능하게 만든다. Example 4.2에 해당하는 검색 표현이 불가능하다. Propositioanal Linear Termporal Logic(PLTL)은 하나의 무한한 패스(a single infinite path)의 노드들이 만족하여야 하는 사실들과 시간 연산자들로 이루어지는 패스 포뮬라를 사용하여 패스를 표현하고 있다. 그러나 PLTL의 시간 개념은 1차원(linear)이다. 다시 말하여 1차원 시간 개념하에서는 오직 하나의 next 노드만 있으며 이것은 트리나 그래프에 있는 가능한 다수의 next 노드들의 표현을 불가능하게 만든다. 또한 여러 계층에 걸친 검색 표현이 불가능하여 Example 4.1과 4.2에 해당하는 검색 표현이 불가능하다.

5. 결 론

이 논문에서 DAG의 형식으로 표현되는 멀티미디어 상연물의 DBMS과의 통합을 위한 데이타 모델, 내용에 바탕을 둔 질의어, 질의의 대수적 처리를 위한 방법을 논의하였다.

멀티미디어 상연물의 내용은 개개의 상연 스트림에 대한 정보뿐만 아니라 그 정보의 흐름도 내용을 구성하는 요소이다. 즉, 정보의 흐름은 상연물의 주제를 구성한다.

또 멀티미디어 자료의 물리적 특징인 분할화와 결합하여 한 상연물 안에서 분할화의 수준에 따라 여러 가지의 흐름이 있다. 이에 멀티미디어 상연물을 내용을 표현할수 있도록 시간 연산자 Next, Connected와 Until을 이용하였다.

검색 언어 GCalculus/S는 GCalculus에 존재하는 범용 정량자 ∀, 범용 패스 정량자 **A**를 제거하고 부정 연산자 ¬는 base predicate의 왼쪽에만 나타나게 하여 얻어진다. 이에 GCalculus/S에서의 질의 표현은 범용 정량자들이 존재하여 발생할수 있는 복잡한 질의 표현을 제거하였다. 즉, GCalculus/S에서 사용되는 모든 변수는 자유변수이거나 존재 정량자에 의하여 구속(bounded)되며 대수를 통한 처리를 위한 대수 표현으로의 변환을 가능하게 한다.

GCalculus/S 질의의 효과적 처리를 위하여 객체 대수인 O-Algebra[19]를 확장하여 이용하였다. GCalculus/S에서 사용된 모든 연산자에 의하여 만들어 지는 포뮬라를 대수 표현으로 변환할수 방법을 제시 하였으며 GCalculus/S에서 사용된 시간 연산자를 직접 변환할수 있는 대수 연산자를 이용하였다. 대수를 통한 처리는 대수의 동등한 변환(equivalent algebra transformation)을 통한 처리 최적화의 가능성을 열수도 있다.

이 논문에서는 GCalculus/S의 응용 도메인으로 멀티미디어 상연물을 이용하였다. 그러나 그래프 타입을 이용하는 다른 도메인에서의 이용도 가능하다. 한편 GCalculus/S에서는 범용 정량자들의 제거를 통하여 모든 변수들이 존재 정량자들로만 구속되어 있다. 이에 GCalculus/S에 있는 변수들과 시간 연산자에 해당하는 아이콘(icon)을 가지고 같은 표현이 가능하다. 즉, 이 특징은 GCalculus/S를 formal basis로 하는 간단한 비쥬얼 질의어(visual query language)의 개발을 가능하게 할 것이다.

참 고 문 헌

[1] "Authorware Professional for Windows," Macromedia, Inc., 600 Townsend St., San Francisco, CA. 94103, 1993

[2] Biskup, J., U. Rasch, and H. Stiefeling, "An Extension of SQL for Querying Relations," Computer Languages 15(1990), pp 65 - 82.

[3] Blakowski, G., Steinmetz, R., "A Media Synchronization Survey: Reference Model, Specification, and Cast Studies," IEEE Journal on Sel. Areas in Comm., Jan., 1996

[4] Consens, M., Mendelzon, A., "GraphLog: A Visual Formalism for Real-Life Recursion," ACM PODS Conference, 1990.

[5] Consens, M., Mendelzon, A., "Hy$^+$: A Hygraph-based Query and Visualization System,," ACM SIGMOD Conference, 1993.

[6] Cruz, I.F., Mendelzon, A., Wood, P.T., "A Graphical Query Language Supporting Recursion," ACM Sigmod Conference, 1987

[7] Cruz, I.F., Mendelzon, A., Wood, P.T., "G$^+$: Recursive Queries witnhour recursion," 2nd Int. Conference on Expert Database Systems, 1988

[8] Emerson, E., "Temporal and Modal Logic" in Handbook of Theoretical Computer Science, Chapter 16, Leeuwen J., editor, pp 995 - 1072, Elsevier, 1990

[9] Gyssens, M., Paredaens, J., Bussche, J., Gucht D., "A Graph-Oriented Database Model," IEEE Trans. on Knowledge and Data Engineering, vol. 6, No. 4, Aug. 1994, pp 572 - 586

[10] Gyssens, M., Paradaens, J., Gucht, D., "A Graph-Oriented Object Database Model," ACM PODS Conference, 1990

[11] Gyssens, M., Paradaens, J., Gucht, D., "A Graph-Oriented Object Model for Database End-User Interfaces," ACM SIGMOD Conference, 1990.

[12] Guting, R., "GraphDB: Modeling and Querying Graphs in Databases," VLDB, conf., 1994, pp 297 - 308

[13] Haindl, M., "A New Multimedia Synchronization Model", IEEE Journal on Sel. Areas in Comm., Jan., 1996

[14] "IconQuthor," Aimtech Corporations, 20 Trafalgar Square, Nashua, NH. 03063, 1997

[15] Lee, Taekyong, Bozkaya, T., Kuo, H-C., Ozsoyogou, G., Ozsoyoglu, Z.M., "A Scientific Multimedia Database System for Polymer Science Experiments," SSDB Conference, Jun., 1996, pp 86 - 95

[16] Lee, Taekyong, Sheng, L., Bozkaya, T., Ozsoyoglu G., Ozsoyoglu, Z.M., "Querying and Manipulating Multimedia Presentation Graphs," Int. Workshop on Multimedia Information Systems, Sep. 26 - 28, 1996, U.S. Army Research Center, West Point, New York, pp 35 - 40

[17] Lee, Taekyong, Sheng, L., Bozkaya, T., Balkir, N.H., Ozsoyoglu, G., Ozsoyoglu, Z.M., "Querying Multimedia Presentations Based on Content," IEEE Trans on Knowledge and Data Engineering, vol. 11, No. 3, Jun., 1999, pp 361 - 385

[18] Levene, M., Loizou, G., "A Graph-Based Data Model and its Ramifications," IEEE Trans. on Knowledge and Data Engineering, vol. 7, No. 5, Oct., 1995, pp 809 - 823

[19] Lin, J., Ozsoyoglu, Z.M., "Processing OODB queries by O-Algebra" CIKM, Nov. 12 - 16, 1996, Rockville,

MD. pp 134 - 142
[20] Liu, K.L., Sistla, A.P., Ym, C., Rishe, N., "Query Processing in a Video Retrieval System," IEEE Data Engineering Conference, 1998
[21] Ozsoyoglu, G., Wang, H., "A Relational Calculus With Set Operators, Its Safety, and Equivalent Graphical Languages," IEEE Trans. on Software Engineering, Vol. 15, No. 9, Sep. 1989, pp 1038 - 1052
[22] Poulovassilins, A., Levene, M., "A Nested-Graph Model for the Representation and Manipulation of Complex Objects," ACM Trans on Information Systems, Vol. 12, No. 1, Jan. 1994, pp 35-68
[23] "Quest", Allen Communication, Inc., 5 Triad Center, Salt Lake City, UT. 84180, 1996
[24] Richardson, J., "Supporting Lists in a Data Model(A Timely Approach)," VLDB Conf., 1992, pp 127 - 138
[25] Sistla, A.P., Yu, C., Venkatasubrahmanian, R., "Similarity Based Retrieval of Videos," IEEE Data Engineering Conference, 1997
[26] Steinmeta, R., "Synchronization Properties on Multi-media Systems," IEEE Journal on Sel. Areas in Comm., Arp., 1990
[27] Ullman, J.D., "Principles of Database Systems," Computer Science Press, 1982

이 태 경
1978년 고려대학교 경제학과 입학. 1985년 고려대학교 경제학과 졸업(학사) 1988년 Indiana University (Bloomington)경제학 석사. 1998년 Case Western Reserve대학 컴퓨터 공학 박사. 1998년 3월 ~ 2000년 2월 경산대학교 정보처리학과 전임강사 2000년 3월 ~ 현재 울산대학교 정보디자인학과 조교수. 관심분야는 멀티미디어 데이타베이스, 데이타마이닝, 지식기반 데이타베이스, Knowledge Representation