# Function-free 논리 프로그램의 Bottom-up 수행의 최적화을 위한 정적 필터의 최소 고정점 정형화

○
창병모, 최 광무, 한 태숙
한국과학기술원 전산학과

# Least Fixed Point Formalization of Computation of Static Filters for Optimizing Bottom-up Evaluation of Function-free Logic Programs

*Byeong-Mo Chang Kwang-Moo Choe, and Tai-sook Han*
Department of Computer Science
Korea Advanced Institute of Science and Technology

**ABSTRACT**

필터링을 이용한 논리 프로그램의 bottom-up 수행은 논리 프로그램 수행의 최적화를 위한 강력한 방법이다 본 논문에서는 정적 필터의 계산을 연립 방정식 형태로 정형화하고 그 최소 고정점(least fixed point)이 구하고자하는 정적 필터가 되도록 한다 정적 필터의 계산 과정은 연립 방정식의 최소 고정점을 구하는 과정으로써 설명된다 여기서 구한 정적 필터를 일반화하여 각 rule의 sideways passing graph(SPG)를 고려하여 부분 정적 필터(partial static filter)를 계산할 수 있으며 동적 필터는 부분 정적 필터로부터 시작하여 단계적으로 계산할 수 있다 이를 위한 방법이 [CCH92]에 설명되어 있으며 정적 필터만을 이용할 때보다 효율적임이 증명되었고 그 방법의 completeness가 증명되었다

## 1. 서 론

논리 프로그램의 수행 방법으로써 제안된 bottom-up 수행방법은 논리 프로그램의 최소 고정점(least fixed point)을 계산한다[Rob65, v&K76] 이 방법이 제안된 이래로, 수행중 불필요한 tuple의 생성을 막기위해 많은 최적화 방법들이 제안되었다[ A&U79, BMS86, K&L86b, S&Z86, Ram87] 각 방법은 똑같은 해을 생성함으로써 bottom-up 수행방법과 동등성을 유지한다

Aho 와 Ullman은 순환 질의(recursive query)을 최소 고정점 연산자(least fixed point operator)을 이용하여 선택 연산(selection)을 이동시킴으로써 최적화하였다[A&U79] 또한 naive bottom-up 수행방법에서 tuple의 반복 생성을 막기위해 seminaive bottom-up 수행방법이 제안되었다[B&R86a,B&R86b] 위의 두 방법을 기초로 한 최적화 방법으로써 정적 필터링(static filtering)방법이 제안되었는데 이는 시스템 그래프(system graph)상에서 seminaive bottom-up 수행방법을 기초로 하고있으며 Aho-Ullman방법의 확장이라고 할 수 있다[K&L86b,90] 시스템 그래프는 논리 프로그램의 구문 구조를 나타내며 rule/goal 그래프와 비슷한 구조를 갖는다 필터는 시스템 그래프의 아크를 통과하는 데이타 즉 tuple의 흐름을 제한하기 위해 아크에 부착된 논리식으로써 사실상 리터럴의 호출 형태를 나타내는 논리식이라 할 수있다 각 필터는 수행중에 불필요한 tuple의 생성을 막는다

필터는 컴파일 시간에 계산되면 정적 필터(static filter)라 부르고 수행 시간에 계산되면 동적 필터라 부른다 정적 필터는 수행중 생성된 데이타와 무관한 최적화 방법으로써 질의(query) 혹은 프로그램의 각 논리 규칙(rule)의 바인딩 조건을 전달함으로써 생성된다 정적 필터을 이용한 필터링을 정적 필터링(static filtering)이라 부른다

동적 필터(dynamic filter)는 수행중 실제 생성된 데이타을 이용하여 계산되는데 한 논리 규칙내에서 *sideways information passing* 방법과 논리 규칙사이의 *backward information propagation* 방법을 이용하여 계산된다 각 논리 규칙의 *sideway passing graph*(SPG)는 sideways information passing의 방향을 제한한다

본 논문에서는 시스템 그래프를 정의하고 이를 기초로 하여 정적 필터의 계산 과정을 연립 방정식으로 정형화하고 그 최소 고정점(least fixed point)이 정적 필터가 되도록 한다 정적 필터는 그 자체로써 complete한 필터로 사용될 수 있다 이를 일반화하면 각 rule의 SPG을 고려하여 **부분 정적 필터**를 계산할 수 있으며 동적 필터는 여기서 구한 부분 정적 필터로부터 시작하여 단계적으로 계산할 수 있다 이를 위한 방법이 [CCH92]에 설명되어 있다 본 논문에서는 지면의 한계때문에 동적 필터 계산 방법은 설명하지 않는다

본 논문에서는 의미의 정확한 전달을 위해서 정의와 정리들은 영어로 표기하였다 논문의 구성은 아래와 같다 제 2 장에서는 논리 프로그램관한 기본적인 정의에 대해서 기술한다 제 3 장에서는 시스템 그래프를 이용한 논리 프로그램의 수행에 대해서 설명한다 제 4 장에서는 정적 필터 계산 과정을 연립 방정식의 최소 고정점으로 정형화 하고 이에 기초하여 정적 필터를 계산한다 제 5

본 논문은 국방과학연구소 장기기초연구과제 "국방 컴퓨터 언어(안)에 관한 연구" ADD-90-4-07의 부분적인 지원을 받았음을 밝힙니다

장에서는 본 논문을 요약하고 결론을 맺는다.

## 2. 기초 정의

1차 논리(first order logic)의 기본 정의를 살펴보자 atomic formula(atom)는 A, B, C 에 의해서 표시되며 그 안에 변수가 없으면 *ground*된 것이다 *literal*은 atom 혹은 그 역을 말한다 literal들의 논리합을 clause라 한다

*program clause*는 다음 형태의 clause이다

$$A \leftarrow B_1, \cdots, B_n \ (n \geq 0)$$

여기서 A는 head atom 그리고 $B_1, \cdots, B_n$는 body atoms이다 만약 $n = 0$이면, program clause는 단지 A이고 *unit clause* 혹은 *fact*라 부른다 그렇지 않으면 *rule*이라 부른다 program clause는 $\alpha, \beta, \gamma, \delta$에 의해서 표시되며 어떤 rule $\alpha$의 body literal의 수는 $n_\alpha$에 의해서 표시된다

*query* $Q$는 $\leftarrow B_1, \cdots, B_n$ 형태의 clause이다. 여기서 $B_1, \cdots, B_n$은 atom들이다 *Horn clause* 는 program clause 혹은 query이다 (논리) 프로그램 $P$는 program clause들의 유한 집합이다 본 논문에서는 함수가 없는 Horn clause을 가정한다 따라서 어떤 term은 변수 혹은 상수이다

**정의 2.1** A ground atom A is the *hyperresolvent* of a ground instance

$$A \leftarrow A_1, \cdots, A_n$$

of a rule $\alpha$ and ground atoms $A_1, \cdots, A_n$

n 개의 ground term들의 리스트는 n-tuple이라 부르고 $\bar{\mu}$ 혹은 $\bar{\nu}$에 의해서 표시된다. 만약 $p_\alpha(\bar{\mu_0})$이 rule $\alpha$의 ground instance

$$p_0(\bar{\mu_1}) \leftarrow p_1(\bar{\mu_1}), \cdots, p_n(\bar{\mu_n})$$

와 ground atoms $p_1(\bar{\mu_1}), \cdots, p_n(\bar{\mu_n})$의 hyperresolvent이면, $\bar{\mu_i}$, $1 \leq i \leq n$, 는 $\bar{\mu_0}$의 $\alpha$을 통한 *contributor*라고 한다 여기서 $\bar{\mu_0}, \bar{\mu_1}, \cdots, \bar{\mu_n}$의 리스트를 $\alpha$의 *satisfying (tuples) list*라 부른다 릴레이션 *contribute*$_\alpha$는 다음과 같이 정의된다

$\bar{\mu_i}$ *contribute*$_\alpha$ $\bar{\mu_0}$ if $\bar{\mu_i}$ is a contributor of $\bar{\mu_0}$ via a rule $\alpha$

$\bar{\nu_0}$ *contribute*$_{\delta_1}$ $\bar{\nu_1}$ *contribute*$_{\delta_2}$ $\cdots$ $\bar{\nu}_{n-1}$ *contribute*$_{\delta_n}$ $\bar{\nu}_n$ 를

만족하는 일련의 tuple들 $\bar{\nu_0}(=\bar{\nu}), \cdots, \bar{\nu}_n(=\bar{\mu})$이 존재하면 $\bar{\nu}$는 $\bar{\mu}$의 *far contributor*라 한다.

프로그램 $P$의 *Herbrand base* $B_P$는 $P$의 ground terms( function-free Horn clause경우에는 상수)을 포함하는 모든 ground atom들의 집합이다 *interpretation*은 $B_P$의 부분 집합이다 $P$가 프로그램이고 I가 interpretation이면 $P$와 연관된 변환 $T_P$는 interpretation을 interpretation으로 사상(map)한다.

$$T_P(I) = \{A \mid A \leftarrow A_1, \cdots, A_n \text{ is a ground instance of a clause in } P, \text{ and } A_1, \cdots, A_n \in I\}.$$

$T_P(I)$는 항상 unit clause의 모든 ground instance들을 포함한다.

A ground atom A is *derivable* in a program if $A \in \bigcup_{m=0}^{\infty} T_P^m(\emptyset)$

where $T_P^0(\emptyset) = \emptyset$ and $T_P^{m+1}(\emptyset) = T(T_P^m(\emptyset))$.

이 정의들은 [v&K76]에서 정의되었다

(논리) 프로그램 $P$의 각 n-ary predicate p는 그 predicate p의 n-ary relation을 나타낸다 어떤 predicate p에 해당하는 *relation* R(p)는 다음과 같이 정의된다.

$$R(p) = \{\bar{\mu} \mid p(\bar{\mu}) \text{ is derivable in } P\}$$

어떤 tuple이 $\bar{\mu} \in R(p)$이고 그것이 질의의 어떤 해의 far contributor이면 $\bar{\mu}$는 p 의 useful tuple이라 한다

## 3. 시스템 그래프를 이용한 논리 프로그램의 수행

이 논문에서는 논리 프로그램의 데이터 플로우 수행에 적당한 *AC-notation*[K&L90]으로 표현된 function-free Horn clause만을 고려하겠다 AC-notation을 기술하기 위해, *formula*는 아래와 같이 정의된다

1. An atomic formula is *true*, *false*, or $t\ \theta\ s$ where t and s are terms(variable or constant), and $\theta$ is $<, >, =, \leq$, or $\geq$
2. If $\phi$ and $\phi'$ are formulas, then $\phi \vee \phi'$ and $\phi \wedge \phi'$ are formulas

논리합이 없는 formula는 *conjunctive formula*라 부른다 어떤 formula $\phi$를 만족시키는 모든 substitution이 $\phi'$을 만족하면 $\phi$는 $\phi'$을 *imply* 한다고 하며 $\phi \rightarrow \phi'$로 표시된다

필터의 정의를 위하여 어떤 formula $\phi$의 $\bar{X}$-*consequence*을 정의한다[K&L90] 일반적인 logical consequence에 대해서는 [C&L73]를 참조하기로 한다 $\bar{X}$가 formula $\phi$의 어떤 변수들의 리스트라고 하자. 어떤 formula $\phi'$이 어떤 formula $\phi$의 $\bar{X}$-*consequence*라 함은 $\phi'$의 모든 변수가 $\bar{X}$에 속하고 $\phi$가 $\phi'$을 imply 함을 의미한다. $\phi'$이 $\phi$의 $\bar{X}$-consequence이고 $\phi$의 다른 모든 $\bar{X}$-consequence $\phi''$에 대해서 $\phi'$이 $\phi''$을 imply하면 $\phi'$는 $\phi$의 *most general* $\bar{X}$-consequence이다.

어떤 n-tuple $\bar{\mu} = <\mu_1, \cdots, \mu_n>$은 $\bar{X} = <X_1, \cdots, X_n>$인 formula $\phi[\bar{X}]$를 $\phi\cdot\sigma$이 참이면 만족한다고 한다, 여기서 $\sigma$는 $\{\mu_1/X_1, \cdots, \mu_n/X_n\}$인 substitution이다.

AC-notation에서 각 literal의 모든 argument는 literal의 predicate와 관련된 서로다른 변수에 의해서 나타내다. n-ary predicate p의 변수는 P1, $\cdots$, Pn로 표현되며 Pj/i는 rule의 i 번째 atom의 j 번째 변수임을 나타낸다 argument들의 바인딩 조건은 각 rule에 하나씩 부착된 conjuctive formula에 의해서 나타낸다 rule $\alpha$의 바인딩 조건은 $\text{Cond}_\alpha$로 표시한다 AC-notation으로 나타내진 rule의 모양은 다음과 같다

$$\alpha.\ p(\bar{P}) \leftarrow p_1(\bar{P_1}/1), \cdots, p_n(\bar{P_n}/n), \text{Cond}_\alpha$$

자세한 설명을 위해서는 [K&L90]을 참조하기로 한다

예 *1*. 아래의 프로그램을 살펴보자

$$p(X,Y) \leftarrow r(X,Y)$$
$$p(X,Y) \leftarrow r(X,Z), p(Z,Y)$$

이들 rule은 AC-notation으로 다음과 같이 표현된다

$$p(P1,P2) \leftarrow r(R1/1, R2/1),\ P1=R1/1,\ P2=R2/1$$
$$p(P1,P2) \leftarrow r(R1/1,R2/1),\ p(P1/2,P2/2),\ P1=R1/1,\ R2/1=P1/2,\ P2=P2/2.$$

어떤 rule $\alpha$에 대해서 함수 $p(\alpha, i)$는 $\alpha$의 body의 i 번째 predicate를 나타내는 함수이고 함수 $h(\alpha)$는 $\alpha$의 head predicate를 나타낸다 함수 $V(\alpha,i)$는 rule $\alpha$의 $i$ 번째 atom내의 모든 변수들의 리스트를 나타낸다

한 프로그램을 위한 어떤 시스템 그래프는 프로그램의 data flow 수행을 위한 도구로써 다음과 같이 정의된다

**정의 *3.1*.** Let SG be a system graph for a program $P$. $SG = (V_P, V_R, E_{P,R}, E_{R,P}, F)$ where

$V_P$ and $V_R$ are the set of predicates and rules in $P$,respectively,

$E_{P,R} = \{(p, \delta, i) \mid \delta \in V_R, p \in V_P, p = p(\delta,i), 1 \leq i \leq n_\delta\}$,

$E_{R,P} = \{(\delta, p) \mid \delta \in V_R, p \in V_P, p = h(\delta)\}$,

$F\ E_{P,R} \rightarrow \{\phi \mid \phi \text{ is a formula on } V(\delta,i), (p,\delta,i) \in E_{P,R}\}$ is a *filter function* that associates a formula with each arc in $E_{P,R}$.

이 정의에서 하나의 프로그램을 위한 여러개의 시스템 그래프가 존재한다는 점을 주의하여야 한다. predicate을 위한 노드는 *pred-node*라 부르고 rule을 위한 노드는 *rule-node*라 부른다 아크는 수행중에 data 포트(port)로 사용됨으로 포트라 부른다 각 노드는 입력 포트(input port)들과 출력 포트*(output port)*들을 갖는다 위의 정의에서 포트 $(p,\delta,i)$는 rule-node $\delta$의 $i$ 번째 입력 포트이고 동시에 pred-node $p$의 출력 포트의 하나이다 여기서는 표현의 간략함을 위해서 $p$는 함수 $p(\delta,i)$이므로 $(p,\delta,i)$는 $(\delta,i)$로도 표시한다 어떤 pred-node $p$의 모든 출력 포트들의 집합은 다음과 같이 표시된다

$$E_{P,R}[p] = \{(p,\delta,i) \mid (p,\delta,i) \in E_{P,R}\}$$

한 포트 $(p,\delta,i)$에 대한 필터 함수 $F$의 값은 $F(p,\delta,i)$이고 그 포트를 통과하기 위해서 tuple들이 만족해야할 변수 $V(\delta,i)$에 대한 하나의 formula이다 $F(p,\delta,i)$는 포트 $(p,\delta,i)$의 필터라 부른다 오직 $E_{P,R}$의 포트들만이 filter을 갖는 점에 주의하기 바란다 filter $F(p,\delta,i)$는 표현의 편의를 위해서 $F_{\delta,i}$로 표시된다 한 rule-node의 입력 포트의 필터는 입력 필터(input filter)라고도 부른다 그림 1은 rule-node $\delta$ 주위의 시스템 그래프를 보여주고 있다

논리 프로그램의 수행에서 시스템 그래프는 pred-node들과 rule-node들 사이의 데이타의 흐름을 나타낸다 각 pred-node $p$는 $p$를 위한 relation $R(p)$를 저장하고 각 rule node $\alpha$는 입력 포트들로부터 tuple들을 받고 $\text{Cond}_\alpha$을 만족하는 새 tuple을 생성한다

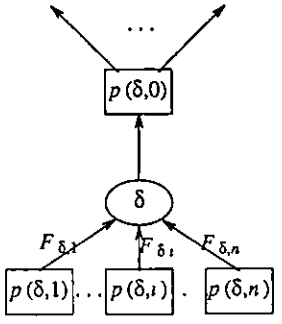


그림 1 rule-node $\delta$ 주위의 시스템 그래프

질의는 새로운 predicate ans를 이용하여 $Q'$ = ans $\leftarrow Q$처럼 나타낸다 $Q'$을 위한 rule-node는 특별히 *query-node*라 부른다 다음의 Algorithm Simplistic seminaive evaluation는 시스템 그래프를 이용한 seminaive bottom-up evaluation이다 필터 함수가 각 포트에 대해서 *true*라고 가정하면 아래의 알고리듬은 논리 프로그램의 최소고정점을 계산한다[v&K76]

**Algorithm Simplistic seminaive evaluation**

1 Every base pred-nodes send out its tuples via each output port if they saufy its filter.

**repeat**

2 Each rule-node store ***new*** tuples from input ports

3 If a rule-node can generate ***new*** tuples, ***new*** tuples are generated and sent to its output ports.

4. Each pred-nodes stores ***newly*** arrived tuples from its input ports and sends them to each output port if they satisfy its filter

until no changes

***정리 3.1*** *On the system graph with the filter function which returns true for each port,* **Algorithm Simplistic seminaive evaluation** *is sound and complete*

위 정리의 증명은 최소 고정점 semantics와 모델 이론(model theoretic) semantics가 동등하다는 증명과 같다 자세한 사항은 [v&K76, Llo84]를 참조하기로 한다

## 4. 정적 필터

이 절에서는 구하고자 하는 정적 필터가 연립 방정식의 최소 고정점이 되도록 정적 필터 계산 과정을 정형화한다

정적 필터 계산을 위한 연립 방정식의 정의를 위하여 한 rule-node의 필터 벡터는 다음과 같이 정의된다 $SG = (V_P, V_R, E_{P,R}, E_{R,P}, F)$는 어떤 프로그램을 위한 하나의 시스템 그래프라고 가정하자 어떤 rule-node $\alpha$의 필터 벡터*(filter vector)* $\vec{F}_\alpha$는 $\alpha$의 모든 입력 필터의 벡터로 정의된다

$$\vec{F}_\alpha = \langle F_{\alpha,1}, .., F_{\alpha n_\alpha}\rangle$$

만약 $n_\alpha = n_\beta$이고 모든 $1 \le i \le n_\alpha$에 대해서 $F_{\alpha,i} \rightarrow F_{\beta,i}$이면, $\vec{F}_\alpha = \langle F_{\alpha,1}, \cdots, F_{\alpha,n_\alpha}\rangle$는 $\vec{F}_\beta = \langle F_{\beta,1}, \cdots, F_{\beta,n_\beta}\rangle$를 *imply*한다고 하며 $\vec{F}_\alpha \le \vec{F}_\beta$로 표시된다

또한 어떤 시스템 그래프 SG의 필터 벡터 $\vec{F}$는 다음과 같이 정의된다

$$\vec{F} = \langle \vec{F}_{\delta_1}, \cdots, \vec{F}_{\delta_m}\rangle \text{ where } V_R = \{\delta_1, \cdots, \delta_m\}$$

$SG = (V_P V_R, E_{P,R}, E_{R,P}, F)$ 와 $SG' = (V_P, V_R, E_{P,R}, E_{R,P}, F')$을 하나의 프로그램을 위한 두 시스템 그래프라고 가정하자 이들 사이의 관계는 다음과 같다

$\vec{F}$ *implies* $\vec{F}'$, denoted by $\vec{F} \le \vec{F}'$, if $\vec{F}_{\delta_i} \le \vec{F}'_{\delta_i}$ for all $1 \le i \le m$ where $V_R = \{\delta_1, \cdots, \delta_m\}$

$\vec{F}$ is *equivalent* to $\vec{F}'$, denoted by $\vec{F} \equiv \vec{F}'$, if $\vec{F} \le \vec{F}'$ and $\vec{F} \Longrightarrow \vec{F}'$.

$\vec{F}$ *strictly implies* $\vec{F}'$, denoted by $\vec{F} < \vec{F}'$, if $\vec{F} \le \vec{F}'$ but $\vec{F}$ is not equivalent to $\vec{F}'$

필터 벡터에 대한 릴레이션 $\le$은 reflexive, transitive, antisymmetric 성질을 만족함으로 partial order 릴레이션이다

앞으로는 프로그램 $P$가 rule 형태 질의도 포함한다고 가정하겠다 한 프로그램의 모든 시스템 그래프의 필터 벡터들의 집합을 다음과 같이 정의한다

$$S(P) = \{\vec{F} \mid \vec{F} \text{ is a filter vector of a system graph for } P\}$$

$\alpha$가 프로그램 P의 시스템 그래프의 rule-node이면

$$S(P,\alpha) = \{\vec{F}_\alpha \mid \langle \cdots, \vec{F}_\alpha, \cdots\rangle \in S(P)\}$$

한 rule-node $\alpha$의 입력 필터를 변환시키는 변환 $\text{PUSH}_\alpha$는 정의 3.1에서 정의된다

***정의 4.1.*** Let $SG = (V_P, V_R, E_{P,R}, E_{R,P}, F)$ be a system graph for a program $P$ and $\alpha$ be a rule-node in SG A transformation $\text{PUSH}_\alpha$ $S(P) \rightarrow S(P,\alpha)$ is $\vec{F}'_\alpha = \text{PUSH}_\alpha(\vec{F})$ such that for each port $(\alpha,i)$,

$$F'_{\alpha,i} = (\text{Cond}_\alpha \wedge \bigvee_{(p,\delta,k)\in E_{p,\delta}[p]} F_{\delta,k})[V(\alpha,i)]$$

where $p = \text{head}(\alpha)$, and for each closed port, $F' = F$.

위의 정의에서, $\alpha$가 query-node $Q'$이면 $\text{PUSH}_\alpha(\vec{F})$내에서 $F'_{\alpha,i} = F_{\alpha,i} \vee (\text{Cond}_\alpha)[V(\alpha,i)]$ 형태를 갖는다

시스템 그래프에 대한 변환 PUSH는 아래와 같이 정의된다

***정의 4.2.*** Let $SG = (V_P, V_R, E_{P,R}, E_{R,P}, F)$ be a system graph for a program $P$ A transformation $\text{PUSH} : S(P) \rightarrow S(P)$ is

$$\text{PUSH}(\vec{F}) = \langle \text{PUSH}_{\delta_1}(\vec{F}), \cdots, \text{PUSH}_{\delta_m}(\vec{F})\rangle$$

where $V_R = \{\delta_1, \cdots, \delta_m\}$

어떤 rule-node의 입력 필터는 그 rule-node의 출력 필터와 관련된 tuple의 생성을 막지 않으면 그 출력 필터에 대해서 safe 하다고 한다.

**정의 4.3.** Let $SG = (V_P, V_R, E_{P,R}, E_{R,P}, F)$ be a system graph, and $\alpha$ be a rule-node in SG. Let $\overline{\mu_i} \in R(p(\alpha,i))$ *contribute* $_\alpha$ $\overline{\mu_0} \in R(h(\alpha))$ A filter $F_{\alpha,i}$ *is safe w.r.t* $\bigvee_{(p,\delta,k)\in E_{P,R}(p)} F_{\delta,k}$ where $p = head(\alpha)$, iff if $\overline{\mu_i}$ does not satisfy $F_{\alpha,i}$, then $\overline{\mu_0}$ does not satisfy $\bigvee_{(p,\delta,k)\in E_{P,R}(p)} F_{\delta,k}$.

위의 정의에서 출력 필터는 애매성이 없으면 생략될 것이다

**보조정리 4.1.** *Let* $SG = (V_P, V_R, E_{P,R}, E_{R,P}, F)$ *be a system graph, and* $\alpha$ *be a rule-node in* SG *If* $\vec{F}' = PUSH(\vec{F})$, *then every filter* $F'_{\alpha,i}$ *is safe w.r.t* $\bigvee_{(p,\delta,k)\in E_{P,R}(p)} F_{\delta,k}$ *where* $p = head(\alpha)$

**증명.** 생략([CCH92] 참조) □

위의 정리는 PUSH후의 각 얼린 입력 필터가 PUSH 전의 출력 필터들에 대해서 safe 함을 보였다

아래의 두 개의 보조 정리와 하나의 정리는 변환 PUSH의 최소 고정점 정형화로써 정적 필터의 계산 과정을 보인다.

**보조정리 4.2.** Let P be a program. A pair $(S(P), \leq)$ is a partially ordered set(*poset*) with a unique minimum element $\vec{F}^{min} = \langle\langle false, \cdots, false\rangle, \cdots, \langle false, \cdots, false\rangle\rangle$. A poset $(S(P), \leq)$ satisfies the *ascending chain condition* that there cannot be an infinite sequence of strictly ascending(strictly imply in this) elements of $(S(P), \leq)$

**증명** 프로그램의 상수와 변수들의 수가 유한함으로 이들로 이루어진 *unequivalent* formula의 수가 유한하다 따라서 가능한 필터 함수와 필터 벡터의 갯수도 유한하다 위의 이유와 릴레이션 $\leq$는 partial order이므로 증명됨. □

**보조정리 4.3** Let P be a program. The transformation PUSH is a monotone increasing function in the poset $(S(P), \leq)$

**증명** $SG = (V_P, V_R, E_{P,R}, E_{R,P}, F)$와 $SG' = (V_P, V_R, E_{P,R}, E_{R,P}, F')$이 한 프로그램을 위한 두 시스템 그래프라고 가정한다 $\vec{F} \leq \vec{F}'$가 만족되면, $PUSH(\vec{F}) \leq PUSH(\vec{F}')$ 역시 만족됨은 Lemma A2[K&L90]을 이용하면 쉽게 증명된다 Lemma A2는 formula $\phi \rightarrow$ formula $\phi'$이면, $\phi[\vec{X}] \rightarrow \phi'[\vec{X}]$임을 나타낸다 여기서 $\vec{X}$는 $\phi$와 $\phi'$내의 어떤 변수들의 리스트이다 □

변환 PUSH의 고정점은 $\vec{F} = PUSH(\vec{F})$를 만족하는 필터 벡터을 의미한다

**정리 4.4** Let *P be a program If* LFP(PUSH) *is the least fixed point of* PUSH *in the poset* $(S(P), \leq)$ *with the minimal element* $\vec{F}^{min}$, *then* $LFP(PUSH) = PUSH^n(\vec{F}^{min})$ *for some finite* n

**증명.** **보조정리 4.2** , **보조정리 4.3** 와 Tarski theorm[Tar85]에 의해서 증명됨 □

각 rule-node에 대해서 하나의 SPG를 가정하면, 어떤 프로그램 P의 정적 필터를 갖는 시스템 그래프는 $SG^{static} = (V_P, V_R, E_{P,R}, E_{R,P}, SF)$인데 여기서 SF는 poset $(S(P), \leq)$에서 $LFP(PUSH)$이다 여기서 구한 정적 필터는 [K&L90]에서 가술된 정적 필터와 같다 알고리듬적인 설명을 위해서는 [K&L90]을 참조하기로 한다.

이렇게 구해진 정적 필터는 safe condition을 만족하며 이는 filtering의 completeness의 기초가 된다

**보조정리 4.5.** *Let* $SG^{static} = (V_P, V_R, E_{P,R}, E_{R,P}, SF)$ *be the system graph with the static filter function* SF *for a program P Every filter* $SF_{\alpha,i}$ *in* $SG^{static}$ *is safe w.r.t* $\bigvee_{(p,\delta,k)\in E_{P,R}(p)} SF_{\delta,k}$ *where* $p = head(\alpha)$

**증명.** 보조정리 4.1과 고정점의 정의에 의해서 증명됨 □

**정리 4.6.** *([K&L90]) Let* $SG^{static}$ *be the system graph with the total static filter function for a program* Algorithm Simplistic seminaive evaluation with $SG^{static}$ *is sound and complete.*

여기서 주의할 점은 정적 필터의 safe condition혹은 필터링의 completeness는 그 필터링이 불필요한(useless) tuple을 생성하지 않는다는 것을 의미하지 않는다는 점이다. 오직 필요한(useful) tuple만을 생성하는 문제는 undecidable 문제임이 이미 증명되어 있다[BKB87] 따라서 이 연구의 목표는 가능한 한 적은 수의 불필요한 tuple의 생성하는 것이다

## 5. 결 론

본 논문에서는 정적 필터의 계산을 연립 방정식 형태로 정형화하고 그 최소 고정점이 구하고자하는 정적 필터가 되도록 하였다 정적 필터의 계산 과정은 연립 방정식의 최소 고정점을 구하는 과정으로써 설명되었다 이 논문에서는 제안된 식을 기초로하여 각 rule의 sideways passing graph(SPG)를 고려하여 부분 정적 필터를 계산할 수 있으며 이를 이용하여 동적 필터를 단계적으로 계산하는 방법이 [CCH92]에 설명되어 있으며 이 방법은 정적 필터만을 이용할 때보다 효율적임이 증명되었고 그 completeness가 증명되었다.

일반적으로 동적 필터의 성능은 SPG에 의해서 크게 좌우된다 효율적인 SPG를 선택하는 문제는 어려운 문제이며 이를 위한 heuristic 방법을 찾는 일은 동적 필터의 성능 향상을 위해서 더 연구되어야 한다 시스템 그래프를 이용한 필터링 방법은 병렬/분산 계산이 용이하며 그 구현은 흥미로운 연구 주제가 될 것이다

## 참고 문헌


[A&U79] Aho, A. and Ullman, J D. Universality of data retrieval languages, *Proc of Sixth ACM Symposium on POPL, ACM, New York, 1979*

[BMS86] Bancilhon, F , Maier, D , Sagiv, Y , and Ullman, J D Magic sets and other strange ways to implement logic programs, *Proc of the ACM SIGACT-SIGMOD Symposium on PODS* ACM, New York, 1986

[B&R86a] Balbin,I and Ramaohanarao, K A differential approach to query optimization in recursive deductive databases, TR-86/7, Dept. of Computer Science, Univ of Melbourne

[B&R86b] Bancilhon, F , and Ramakrishnan, R An amateur's introduction to recursive query processing starategies *Proc of the SIGMOD Conference on Management of Data*, ACM, New York, 1986

[BKB87] Beeri, C , Kanellakis, P , Bancilhon, F , and Ramakrishnan, R Bounds on the propagation of selection into logic programs *Proc of the ACM SIGACT-SIGMOD Symposium on PODS* , ACM, New York, 1987

[CCH92] Chang, B -M , Choe, K -M , Han, T Optimized bottom-up evaluation of function-free logic programs with dynamic filtering CS-TR-92-66, KAIST, Feb 1992

[C&L73] Chang, C -L , Lee, R C T *Symbolic logic and mechanical theorem proving*, Academic Press, 1973

[K&L90] Kifer, M , and Lozinskii, E L On compile time query optimization in deductive databases by means of static filtering *ACM TODS, Vol 15*, No 3, Sep 1990

[K&L86b] Kifer, M , and Lozinskii, E L Filtering data flow in deductive databases *Proceeding of the International Conference on Database Theory* Springer-Verlag, New York, 1986

[Llo84] Lloyd, J W *Foundations of logic programming*, Springer-Verlag, Berlin, 1984

[Rob65] Robinson, J A. Automatic deduction with hyper-resolution *Internat J Comput Math , Vol 1*, pp 227-234, 1965

[Tar85] Tarski, A A lattice theoretical fixpoint theorem and its applications *Pacific Journal of Mathematics, Vol 5 pp 289-321, Sept 1985*

[v&K76] van Emden, M H, Kowalski, R A The semantics of predicate logic as a programming language *JACM Vol 23*, No 4, Oct. 1976