프롤로그에서 증명의 수를 효과적으로 제어하기 위한 방법

남 영 광†

요 약

본 논문에서는 프롤로그 프로그램의 수행을 제어하기 위한 하나의 기능으로서 수항목(count term)이라는 새로운 기능을 제안한다. 그 목적은 사용자와 프로그래머에게 답 또는 증명의 수를 제한하는 기능과 프롤로그 프로그램의 수행을 제어하는 데 편리함을 주기 위한 것이다. 따라서 그를 위한 구문과 작동적 의미를 제시하였으며 WAM(Warren Abstract Machine)에서 백트랙킹시 관련된 명령어를 수정하여 수항목의 구현 방법을 제시한다.

An Efficient Method for Controlling the Number of Proofs in Prolog

Young Kwang Nam†

ABSTRACT

We propose an extension to Prolog called the *count term* for controlling Prolog execution. The purpose is to allow the programmers as well as the users to have greater flexibility in controlling the execution behavior of Prolog programs and for limiting the number of answers or proofs retrieved when Prolog is used as a database query language. Both syntax and operational semantics of the count term are defined. An implementation strategy based on WAM (Warren Abstract Machine) by modifying instructions related to backtracking behavior has been suggested.

1. 서 론

지난 약 10년 동안 논리언어 중의 하나인 프롤로그(Prolog) 는 프로그램의 논리로부터 제어의 분리, 분명하고 선언적인 특성 때문에 많은 발전을 거듭해왔다. 그러나 프롤로그의 비효율적인 점을 극복하기 위하여 제시된 컷(!)과 fail과 같은 비논리적인 연산자 때문에 선언적인 프롤로그의 장점을 많이 삭감시키고 있다[7, 11]. 이러한 제어 연산자 이외에도 프롤로그의 비결정성(undecidibility)으로 인하여 형, 모드(mode)[6, 15], 그리고 프로그램의 의미(semantics)[13] 등을 분석하기가 어렵다. 만약 이러한 것을 쉽게 분석할 수 있다면 사용자나 프로그래머에게 좀더 나은 프로그램 환경을 제공할 수 있을 것이다.

한편 논리언어로서의 사용 이외에도 프롤로그는 많은 학자들에 의해서 연역 데이타베이스(deductive database)를 위한 질의어로서 적당하다고 주장되어 왔다[2]. 프롤로그는 그것이 질의어로서 사용되었을 경우 SLD search tree[9]에 있는 모든 가지(branch)에 대한 백트랙킹(bactracking)을 통해 주어진 질의에 대한 모든 답을 제시할 것이다. 때때로 데이타 베이스에서는 주어진 질의어에 대해 모든 가능한 답을 제시하는 것이 실현 가능하나 바람직하지 않은 경우가 있다. 현재로서는 사용자나 프로그래머가 주어진 질의

※본 논문은 연세대학교 교내 학술 연구비에 의하여 연구되었음.

† 정 회 원:연세대학교 전산학과 교수

논문접수:1996년 10월 1일, 심사완료:1997년 4월 24일

에 대해 제시된 답의 수를 제한하기 위한 방법으로서는 컷과 fail을 이용하여야 하는데 이는 프로그램의 작성이나 의미 파악이 힘들다. 대부분의 경우에 *setof*나 *bagof*[12]와 같은 비논리적인 연산자에 의해서 원하지 않는 답을 걸러 내거나 모든 답을 검색하는 방법을 취하고 있다.

프롤로그는 대체적으로 명시적(explicit) 혹은 암시적 (implicit)의 두가지 방법으로 수행을 제어한다. 해석기(interpreter)가 프로그래머에게 추가의 답을 원하는지 묻기 위해서 프롬프터(prompter)를 내보내고 프로그래머가 세미콜론(;)을 입력하는 경우는 명시적인 제어 방법이고 컷이나 fail등을 이용해서 제어하는 방법을 암시적 제어방법이다. 본 논문에서는 보다 선언적이며 분명하게 명시적, 암시적 제어를 대치함과 동시에 프로그램의 작성이 쉽고 의미를 쉽게 파악하게 할 수 있는 수항목(count terms)이라는 새로운 술어를 소개한다.

본 논문의 구성은 다음과 같다;2 절에서는 수항목의 비정형적인 의미를 소개하고, 3절과 4절에서는 수항목의 구문적 특성과 작동적 의미를 설명하며, 5절에서는 WAM(Warren Abstraction Machine) 환경하에서 어떻게 수항목을 구현할 것인가에 대해 설명한다.

2. 수항목의 비정형적 의미

수항목은 콜론(:)과 정수로 구성된다 (예 : p(X, Y): 3). 수항목의 의미는 질의(query) 혹은 절 (clause) 안에 있는 종속고울(subgoal)을 독립적인 질의로 생각했을 때 그 종속고울에 붙여져서 검색되어야 할 답의 수를 제한하는 역할을 한다. 그러나 종속고울이 질의에 대한 답을 구하는 길에 있을 경우도 있으므로 수항목은 다음과 같은 두가지 의미를 가질 수 있다. 첫번째는 수항목이 질의에 붙여졌을 경우이다. 예를 들어서 다음과 같은 질의가 있을 경우를 생각해보자.

: −p(X, Y): n

만약 주어진 프로그램이 n개보다도 더 많은 답을 가지고 있을 경우에는 오직 n개의 답만이 검색된다. 그렇지 않으면 모든 답이 검색된다. 만약 답의 갯수가 n개보다 작더라도 프로그램은 성공적으로 수행된 것이라 간주한다. 예를 들어 고용자 데이타 베이스에서 처음부터 100명의 남자 고용자에 대한 이름과 그들의 고유번호를 얻고자 할 경우 p(SSN, Name, male): 100와 같이 질의를 표현하면 원하고자 하는 답을 얻을 수 있다.

두번째 의미는 절의 몸체(body)에 수항목을 가지고 있는 경우인데 이 경우는 SLD search tree 에서 백트랙킹하는 수를 제어한다. 다음과 같이 절의 p(X, Y, Z)와 절의 몸체에 수항목이 있는 프로그램을 생각하여 보자.

```
p(X, Y, Z): − ..., r(X, Y), ...
r(X, Y): − ..., q(X, Y):2, ...
q(X, Y): − ...
q(X, Y): − ...
q(X, Y): − ...
⋮
: −p(X, Y, Z)
```

종속고울 q(X, Y)에 붙어 있는 ":2"는 SLD search tree에서 만약 종속고울 q(X, Y)를 통과하는 가지가 2개 이상 있으면 그 수를 2개로 제한한다. 만약 수항목이 질의에도 있고 절의 몸체에도 있으면 우선순위는 질의에 있는 수항목에 주어진다. 그 의미는 만약 질의에 대한 답의 수가 만족이 되면 SLD search tree에서 더 방문되어야 할 길이 있더라도 주어진 프로그램은 거기서 멈추어져야 한다. 수항목은 단지 답의 개수를 제어하기 위한 것이므로 SLD search tree에서 같은 답이 얻어질 경우 수항목을 이용하여 얻어진 답은 다르지 않을 수 있다. 프롤로그에서 진정한 의미에서의 답은 증명(proof)을 의미하나 편의상 답으로 모두 표시한다.

한 절에 수항목을 가진 술어(predicate)가 두개 이상이 있을 경우의 의미는 나중에 호출된 술어가 주어진 수항목 만큼의 가지나 답을 갖지 못했더라도 주어진 질의에 대하여 하나 이상의 답이 있을 때에는 그 상위의 술어의 수항목의 값을 1만큼 감소할 수 있다. 예를 들어 질의 p(X, Y)에 대하여 다음과 같은 절이 있다고 가정하자.

p(X, Y): −q(X, Y):2, r(Y), s(Y):3. (1)

만약 술어 s에 대하여 답이 하나 밖에 없다고 하더라도 술어 q의 첫번째 호출에 대하여 만족된 것이므로 q에 대한 수항목을 1만큼 감소시킨다.

3. 수항목의 구문

이 절에서는 수항목을 포함한 논리언어에 대한 문법을 소개한다. 언어 L은 상수 기호와 함수 기호(function symbols), 술어(predicate) 기호 그리고 무한의 변수 V로 구성되어 있다. 아톰(atom)은 기존에 정의된 것과 같으며(Chang과 Lee[3] 참조) 언어 L은 연결자 ←와 &를 포함한다. 본 논문에서 다루는 언어는 또한 정수의 집합 $\mathcal{N}\cup\mathrm{T}$로 나타내어 지는 N_T을 포함한다. 여기에서 T 기호는 무한대(∞)를 의미한다.

정의 3.1 만약 A가 아톰이고 $\mu\in N_\mathrm{T}$, $A:\mu$는 *count annotated* 아톰이며, 약하여 *ca-atom*이라 부른다.

정의 3.2 *ca-clause*는 $L_0\leftarrow L_1\&\cdots\&L_n$ 형식의 공식이다. L_0는 아톰이며 각각의 L_1, ..., L_n은 아톰이거나 *ca-atom*이다.

각각의 L_i $1\le i\le n$에 대해 우리는 그것이 L_i:T의 약식 표현으로 간주한다. 프로그램은 ca-clause 들의 유한 집합이다.

수항목을 소개하게 된 동기는 논리언어의 수행을 제어하기 위한 것이기 때문에 적절한 의미가 수행을 반영하여야 한다. 따라서 일차논리(first order logic)의 모델 이론에 기초한 집합론적인 의미로서는 그것이 충분하지 못하다.

Debray와 Mishra[6]는 표현적인 의미에 기초한 프롤로그의 의미를 연구하였다. 그들의 의미는 프롤로그의 깊이우선(depth-first)과 좌우(left-to-right) 계산 방법을 고려하였는데 본 논문에서도 그 방법을 취한다. 본 논문에서 나타내는 의미는 [5]에서 정의된 해석기에 기초하고 있다. 직감적으로 의미는 고울(goal) 변수에 대한 유한의 대치(substitution)의 수열로 정의된다. 사용자에게는 그것이 오직 프롤로그 프로그램의 관찰할 수 있는 행동으로 받아들이면 된다. 따라서 주어진 질의 $Q=\leftarrow A_1\&\ldots\&A_n$과 프로그램 P에 대해서, Debray와 Mishra에 의한 의미는 Q가 A_1, ..., A_n에 있는 변수에 대해서 유한의 대치(θ_1, θ_2, ..., θ_n)의 연속이 되도록 정의하였다. θ_1은 프롤로그의 계산 전략으로부터 얻어진 첫번째 대치이다. 여기서는 질의 Q에 대한 유한의 대치를 $seq(A_1\&\ldots\&A_n)$으로 나타낸다.

4. 수항목의 작동적 의미

수항목과 관계되어 가장 손쉽게 나타낼 수 있는 의미는 시도되어야 할 백트랙킹 횟수를 제한하는 일이다. 수항목에 대한 이러한 이해는 주어진 질의에 대해 주어진 데이타베이스에서 원하는 답이 SLD search tree의 처음 n번째 내의 길에 포함이 되어 있다는 것을 알기를 원하거나 알고 있을 경우에 매우 편리하다. 정형적인 수항목의 의미는 [5]에서 정의된 컷이 없는 해석기의 간단한 변경으로 쉽게 설명될 수 있다.

해석기는 스택(stack) 위에서 작동한다. 스택은 해석기의 작동 상황을 잘 설명하여 줄 수 있다. 스택에 있는 각 원소는 FrameList, 대치, 절들과 그리고 현재의 고울의 자손이 답을 얻었는 지를 알려주는 신호변수 *flag*으로 구성되어 있는 레코드로 정의한다. 이 *flag*은 하나의 스택원소가 새로 만들어질 때마다 0으로 초기화된다. Framelist는 Frame의 연속으로 구성되어 있으며 각 Frame은 해결되어야 할 고울 C_atomList와 호출한 고울에 전달되어야 할 바인딩(binding)에 필요한 변수들의 집합을 포함하고 있다. C_atomList는 C_atom의 연속으로 구성되어 있고 각 C_atom은 아톰 또는 ca-atom에 해당되는 아톰 이름과 수항목으로 구성되어 있다.

$C\_atom::=\langle Atom,\ \eta\rangle$, 여기서 $\eta\in N_\mathrm{T}$
$C\_atomList::=nil\,|\,C\_atom::C\_atomList$
$Frame::=\langle C\_atomList,\ Varp\rangle$
$FrameList::=nil\,|\,Frame::FrameList$
$Stack::=nil\,|\,\langle FrameList,\ Subst,\ Clause^*,\ flag\rangle::Stack$
$Stack::=Stack.$

* 표시는 통상의 Kleene closure 연산자를 나타내며, P는 주어진 프롤로그 프로그램을 나타내고, $A::B$는 리스트를 나타내는데 그것의 첫번째 원소는 A이고 B는 리스트의 나머지를 나타낸다. 해석기는 다음과 같

은 함수로 나타내진다.

$interp : State \times Clause^* \times \mathcal{N}_T \rightarrow FSubst^{\infty}$.

주어진 프로그램 P에 있는 절들 C에 있는 모든 술어들을 각각의 독립적인 질의라 생각하여 해석기에 대한 함수로 표현하면 다음과 같이 표현된다.

$interp(\langle\langle\langle A, \mu\rangle :: nil, V_P\rangle :: F, nil, C, 0\rangle :: nil, P, \mu) = \theta_1 :: \theta_2 :: \cdots :: \theta_i$, 여기서 $0 \le i \le \mu$ or $i = \mathsf{T}$

여기서 $FSubst^{\infty}$는 유한 대치의 유한 또는 무한의 연속을 나타내는 집합이다. 해석기에 대해 자세한 것은 (그림 1)에 나타나 있다.

주어진 프로그램은 실행시 사용된 스택에 아무것도 없을때 즉 더 이상 수행될 고울이 없거나 수항목이 0이 되었을때 멈춘다.

$interp(nil, P, \eta) = interp(St, P, 0) = nil$

여기에 나오는 해석기와 수항목이 없는 정상적인 프롤로그를 실행하기 위한 해석기와 다른 점은 (그림 1)의 항목 2와 3의 경우이다. 항목 2의 경우에 있어서 하나의 답이 얻어졌을 때를 나타내는데, 수항목이 있

1. $interp(nil, P, \eta) = interp(St, P, 0) = nil$
2. $interp(\langle nil, \phi, C, 0\rangle :: \langle\langle\langle A, \mu\rangle :: G, V_P\rangle :: F, \theta, C, f\rangle :: St_2, P, \eta(\neq 0)) =$
   $\phi :: interp(\langle\langle\langle A, \mu\rangle :: G, V_P\rangle :: F, \theta, C, 1\ rangle :: St_2, P, \eta - 1)$
3. $interp(\langle\langle\langle A, \mu\rangle :: G, V_P\rangle :: F_0, \phi, nil, 0\rangle :: \langle\langle\langle L, \beta\rangle :: G_1, V_P'\rangle :: F_1, \theta, C_1, f'\rangle :: St_0, P, \eta(\neq 0)) =$
   $interp(\langle\langle\langle L, \beta\rangle :: G_1, V_P'\rangle :: F_1, \theta, C_1, f'\rangle :: St_0, P, \eta)$
4. $interp(\langle\langle\langle A, \mu\rangle :: G, V_P\rangle :: F_0, \phi, nil, 1\rangle :: \langle\langle\langle L, \beta\rangle :: G_1, V_P'\rangle :: F_1, \theta, C_1, f'\rangle :: St_0, P, \eta(\neq 0)) =$
   $interp(\langle\langle\langle L, \beta - 1\rangle :: G_1, V_P'\rangle :: F_1, \theta, C_1, f'\rangle :: St_0, P, \eta)$
5. $interp(\langle\langle\langle A, \mu\rangle :: G, V_P\rangle :: F_0, \phi, (H_0 : -B_0) :: C, f\rangle :: St_0, P, \eta(\neq 0)) =$
   $interp(\langle F_2 :: F_1 :: F_0, \theta \circ \phi, P, 0\rangle :: St_1, P, \eta)$, 여기서
   $\mu \neq 0$;
   $H_1 : - B_1 = rename((H_0 : - B_0), dom(\phi))$;
   $\theta = unify(\phi(A), H_1) \neq fail$;
   $V_P' = dom(\phi)$;
   $F_2 = \langle B_1, V_P'\rangle$;
   $F_1 = \langle G, V_P\rangle$;
   $St_1 = \langle\langle\langle A, \mu\rangle :: G, V_P\rangle :: F_0, \phi, C, f\rangle :: St_0$.
6. $interp(\langle\langle\langle A, \mu\rangle :: G, V_P\rangle :: F_0, \phi, (H_0 : -B_0) :: C, f\rangle :: St_0, P, \eta(\neq 0)) =$
   $interp(\langle\langle\langle A, \mu\rangle :: G, V_P\rangle :: F_0, \phi, C, f\rangle :: St_0, P, \eta)$, 여기서
   $\mu \neq 0$;
   $H_1 : -B_1 = rename((H_0 : -B_0), dom(\phi))$;
   $\theta = unify(\phi(A), H_1) = fail$.
7. $interp(\langle\langle\langle A, 0\rangle :: G, V_P\rangle :: F_0, \phi, C, f\rangle :: \langle\langle\langle L, \mu\rangle :: G_1, V_P'\rangle :: F_1, \theta, C_1\rangle :: St_0, P, \eta(\neq 0)) =$
   $interp(\langle\langle\langle L, \mu - 1\rangle :: G_1, V_P'\rangle :: F_1, \theta, C_1, f'\rangle :: St_0, P, \eta)$
8. $interp(\langle\langle nil, V_P\rangle :: F_0, \phi, C, f\rangle :: St, P, \eta(\neq 0)) = interp(\langle F_0, \phi \downarrow V_P, C, f\rangle :: St, P, \eta)$.

(그림 1) 처음 n개의 답을 찾는 해석기

는 경우에는 답을 얻기 전의 아톰의 수항목의 값을 1만큼 감소시킴으로써 μ-1의 백트랙킹만이 더 허용된다는 것을 알리는 것을 제외하고는 그 수행 방법이 비슷하다. 동시에 절의에 붙어있던 수항목의 값 η를 1만큼 감소시킨다. 만약 감소된 수항목의 값이 0이면 이 경우에도 프로그램의 수행을 중단한다.

$interp(\langle nil, \phi, C, 0\rangle :: \langle\langle\langle A, \mu\rangle :: G, V_P\rangle :: F, \theta, C, f\rangle :: St_2, P, \eta(\neq 0)) =$
$\phi :: interp(\langle\langle\langle A, \mu\rangle :: G, V_P\rangle :: F, \theta, C, 1\ rangle :: St_2, P, \eta - 1)$

한편 현재의 고울에 대해서 더 이상 수행될 절이 없으면 수행이 실패하여 백트랙킹을 일으키는데, 이 경우 실패한 스택의 원소를 삭제하기 전에 현재의 스택의 신호의 값을 검사하여 만약 그 값이 1이면 다음에 실행될 아톰의 수항목의 값을 1만큼 감소시킨다.

$interp(\langle\langle\langle A, \mu\rangle :: G, V_P\rangle :: F_0, \phi, nil, 0\rangle :: \langle\langle L, \beta\rangle :: G_1, V_P'\rangle :: F_1, \theta, C_1, f'\rangle :: St_0, P, \eta(\neq 0)) =$
$interp(\langle\langle\langle L, \beta\rangle :: G_1, V_P'\rangle :: F_1, \theta, C_1, f'\rangle :: St_0, P, \eta)$

만약 신호의 값이 0이면 현재의 아톰이 답을 얻지 못한 경우가 되므로 그 아톰에 상위의 아톰에 대한 수항목의 값을 감소시키지 않고 스택의 맨 위에 있는 원소를 삭제한다.

$interp(\langle\langle\langle A, \mu\rangle :: G, V_P\rangle :: F_0, \phi, nil, 0\rangle :: \langle\langle L, \beta\rangle :: G_1, V_P'\rangle :: F_1, \theta, C_1, f'\rangle :: St_0, P, \eta(\neq 0)) =$
$interp(\langle\langle\langle L, \beta\rangle :: G_1, V_P'\rangle :: F_1, \theta, C_1, f'\rangle :: St_0, P, \eta)$

(그림 1)의 항목 5는 첫번째 절의 머리 부분을 적당히 이름을 다시 붙인 후에 고울의 제일 왼쪽에 있는 술어와 단일화가 성공하면, Framelist는 그 절의 몸체에 붙어 있는 나머지 종속고울들을 포함하는 새로운 Frame을 생성하는데, 이때 신호 *flag*은 0으로 초기화된다. 단 현재 단일화 할려고 하는 고울의 수항목이 0이 아니여야만 단일화의 수행이 가능하다.

$interp(\langle\langle\langle A, \mu\rangle :: G, V_P\rangle :: F_0, \phi, (H_0 :- B_0) :: C, f\rangle :: St_0, P, \eta(\neq 0)) =$
$interp(\langle F_2 :: F_1 :: F_0, \theta \circ \phi, P, 0\rangle :: St_1, P, \eta)$, 여기서
$\mu \neq 0$;

$H_1 :- B_1 = rename(H_0 :- B_0), dom(\phi))$;
$\theta = unify(\phi(A), H_1) \neq fail$;
$V_P' = dom(\phi)$
$F_2 = \langle B_1, V_P'\rangle$;
$F_1 = \langle G, V_P\rangle$;
$St_1 = \langle\langle\langle A, \mu\rangle :: G, V_P\rangle :: F_0, \phi, C, f\rangle :: St_0$.

만일 단일화가 실패하면 현재의 절은 버려지고 나머지 절에 대해서 이 과정이 반복된다. 마찬가지 로 이 경우도 현재의 고울의 수항목이 0이 아닌 경우에만 수행된다.

$interp(\langle\langle\langle A, \mu\rangle :: G, V_P\rangle :: F_0, \phi, (H_0 :- B_0) :: C, f\rangle :: St_0, P, \eta(\neq 0)) =$
$interp(\langle\langle\langle A, \mu\rangle :: G, V_P\rangle :: F_0, \phi, C, f\rangle :: St_0, P, \eta)$, 여기서
$\mu \neq 0$;
$H_1 :- B_1 = rename(H_0 :- B_0), dom(\phi))$;
$\theta = unify(\phi(A), H_1) = fail$.

현재 수행될 고울의 수항목이 0이면 현재의 Frame을 삭제시켜야 하는데 삭제시키기 전에 현재 Frame 다음에 있는 Frame의 가장 왼쪽에 있는 아톰의 수항목을 1만큼 감소시킨다.

$interp(\langle\langle\langle A, 0\rangle :: G, V_P\rangle :: \langle\langle\langle L, \mu\rangle :: G_1, V_P'\rangle :: F_1, \theta, C_1\rangle :: St_0, P, \eta(\neq 0)) =$
$interp(\langle\langle\langle L, \mu - 1\rangle :: G_1, V_P'\rangle :: F_1, \theta, C_1, f'\rangle :: St_0, P, \eta)$

Frame안에 더 이상 수행되어야 할 고울이 없을 때, 현재의 대치값은 기존의 대치 값들과 함께 합성이 이루어지고 그 합성된 대치값을 이용하여 계산이 계속 수행된다.

$interp(\langle\langle\langle nil, V_P\rangle :: F_0, \phi, C, f\rangle :: St, P, \eta(\neq 0)) =$
$interp(\langle F_0, \phi \downarrow V_P, C, f\rangle :: St, P, \eta)$.

질의 G의 절들의 연속 C에 대한 적절한 계산은 다음과 같이 표현된다.
$interp(\langle\langle G, V\rangle :: nil, \varepsilon_V, P, f\rangle :: nil\rangle, P, \eta(neq0))$, 여

기서 $V = var(G)$과 ε_V은 V에 대한 identity finite substitution이다.

5. WAM에서의 수항목의 구현

본 절에서는 WAM[4]에서 어떻게 수항목을 구현할 것인가에 대해 설명한다. 대부분의 해석기나 컴파일러의 근본이 되는 자료 구조로서 스택을 사용하고 있기 때문에 WAM에서의 구현 방법은 다른 곳에서도 쉽게 적용될 수 있다. WAM에는 네가지의 데이타 저장소가 있는데, 프로그램과 명령어를 포함하고 있는 코드 저장소(code area), 그리고 스택과 힙(heap)과 트래일(trail)로 구성되어 있다. 힙은 단일화와 프로시듀어(procedure) 호출에 의해서 만들어진 모든 구조(structure)와 리스트를 포함하고 있다. 트래일은 단일화하는 동안에 결합(bound)되고 백트랙킹시 분해(unbound)되어야할 변수에 대한 주소(reference)를 가지고 있다. 스택은 환경(environment)과 선택점(choice point)를 가지고 있다. 환경은 벡터(vector) 또는 절의 몸체에 있는 변수에 대한 값의 방(cell)과 다른 절의 몸체에 대한 포인터(pointer)와 그와 관계된 환경에 대한 연속점(continuation)을 포함하고 있다. 선택점은 백트랙킹이 일어날 경우 그 전의 상태로 원상 복귀하는데 필요한 모든 정보를 포함하고 있다. 그것은 하나의 프로시듀어를 시작할때 그 프로시듀어를 호출한 고울과 단일화가 가능한 절이 더 있을 경우에 생성된다. 거기에 저장되어 있는 정보는 호출이 가능한 절에 대한 포인터와 여러가지 레지스터들의 값, 즉 스택의 끝(top)이나, 힙의 끝, 바로 전의 선택점과 환경, 트래일의 끝 그리고 프로시듀어의 매개변수에 대한 레지스터에 대한 것이다.

WAM의 수행모형은 바로 앞절에서 정의된 해석기와 그 수행 모형이 비슷한데 둘 다 스택을 사용하고 있으며 스택의 한 원소 안에 백트랙킹과 단일화에 필요한 정보를 포함하고 있기 때문이다. 따라서 WAM에서의 수항목의 구현은 앞절에서 제시된 해석기의 각 항목에 대하여 설명함으로서 쉽게 해결될 수 있다. 앞서 제시한 해석기에 기초하여 설명하기 위해서는 다음과 같은 정보 저장소가 더 필요한데 첫번째로 필요한 것은 CR(Count Register) 레지스터로서 이것은 해석기에서의 η와 같이 질의에 붙어있는 수항목을 나타내며 전역변수(global variable)로서 사용된다. 선택점에는 호출하는 아톰의 수항목을 나타내기 위해서 count_term이라는 새로운 필드를 만들며, 동시에 해석기의 *flag*에 해당되는 flag라는 필드를 만드는데 그 목적은 해석기에서의 목적과 같다. count_term의 값은 호출할 때 그 값이 선택점으로 전달되며 flag 또한 프로시듀어 호출시 만들어지며 그 초기값은 0이다. 수항목의 값을 쉽게 선택점으로 전달하기 위해서 PR(Path Register)이라는 새로운 레지스터를 또 하나 할당하는데, 이것은 다음의 세 명령어에서 쓰인다. 처음의 두 명령어는 순차적인 명령어로서 "call Proc, N"과 "execute Proc"인데, Proc는 수행되어야 할 절의 이름이고, N은 환경속에 있는 지속변수(permenent variable)의 갯수이다. 다른 하나는 "try_me_else Add" 또는 "try Add"라는 명령어로서 Add는 다음에 수행될 절의 주소를 가르친다. WAM에서의 호출순서는 "call Proc, N"이나 "execute Proc" 명령어 다음에 "try_me_else Add" 명령어가 수행되는데, count_term이 저장되어야 할 선택점은 "try_me_else Add" 혹은 "try Add" 명령에 의해서 만들어지므로 ")" 기호나 ":M" 기호를 만남으로서 실행되는 "call Proc, N" 명령어로부터 "try_me_else Add" 명령어에 수항목을 전달하기 위해서는 위의 세명령어를 모두 수정하여야 한다. 그렇게하기 위해서는 위의 세 명령어 모두에 대해서 수항목 부분을 첨가한다. 즉, "call Proc, N"을 "call Proc, N, M"으로, "execute Proc"를 "execute Proc M"으로 "try_me_else Add"나 "try Add"를 "try_me_else Add M"과 "try Add M"으로 각각 수정한다. 동시에 M의 값이 선택점에 새로 만들어진 count_term 필드에 할당되고 flag 필드의 값은 0으로 초기화된다.

항목 1에 대해서는 스택이 비어있거나 CR 레지스터의 값이 0이면 프로그램의 수행을 중단한다.

항목 2의 경우에는 다음과 같은 명령어가 필요하다. 만약 CR 레지스터의 값이 0이 아니고, 프로그램 계수기(PC)의 값이 nil이면 하나의 답이 발견되었다는 것을 의미하므로 CR의 값을 1만큼 감소시키고 flag 필드의 값을 1로 하며, 동시에 바로 전에 있는 선택점의 수항목의 값을 1만큼 감소시킨다. 이러한 일련의 명령들은 "Proceed" 명령어 다음에만 수행이 된다. 그 이유는 한 프로그램이 답을 가질 수 있는 경우

는 오직 단위절(unit clause)과의 단일화가 성공적으로 이루어진 후에만 얻어질 수 있기 때문이다.

현재의 고울에 대해서 다음에 수행될 더 이상의 절이 없을 경우에 현재의 선택점은 삭제되어야 하는데, 삭제하기 전에 반드시 현재의 선택점에 있는 flag 필드의 값을 검사하여야 한다. 만약 그 값이 1이면 바로 전에 있는 선택점의 값을 감소시킨다. 그렇지 않고 0이면 현재의 술어의 자손들 중에서 답을 얻지 못했다는 것을 의미하므로 수항목의 값을 감소시키지 않아도 된다. 위의 두 경우는 해석기의 항목 3과 4에 각각 해당된다.

항목 5에 대해서는 이미 "try_me_else Add" 명령어에 의해서 flag 필드의 값이 0으로 만들어졌으므 로 아무런 행동이 취해질 필요가 없다. "retry_me_else" 명령에 의해서 다음에 수행될 주소가 바뀜으로 항목 6에 대해서도 자동적으로 해결된다.

항목 7에 대해서는 count_term 필드의 값이 0이면 현재 선택점을 삭제하면 되고, 프로그램 계수기의 값을 연속점(CP)의 값으로 대치하는 것은 항목 8에 해당된다. 〈표 1〉는 WAM에서 수항목을 구현하기 위해 변형된 명령어들의 요약을 나타낸다.

〈표 1〉 WAM에서 수항목의 구현을 위하여 변형된 명령어
〈Table 1〉 Modified WAM instructions for count terms

인코딩 시간	수정된 명령어	추가된 연산 명령
) 또는 :M	call Proc, N, M	PR:=M
단위절 종료시	proceed Proc, M	if PC=nil then CR:=CR−1; if CR=0 then stop else flag:=1; B(count_term)= B(count_term)−1;
단일화 전	try_me_else Add, M try Add, M	count_term:=M flag:=0;
단일화 후	trust_me_else fail trust fail	if flag=1 then B(flag)=1; B(count_term):= B(count_term)−1; discard the current choice point;

PC : 프로그램 계수기　　M : 술어에 붙어 있는 수항목
B : 바로 전의 선택점의 주소　　PR : Path Register
CR : Count Register

6. 결 론

본 논문에서는 ":"과 정수의 결합으로 만들어진 수항목의 의미, 구문, 그리고 사용 방법과 WAM에서의 구현방법등에 대하여 살펴보았다. 수항목은 답의 수를 제한해 줄 수 있는 기능을 가지고 있을 뿐만이 아니라 절의 몸체안에 있는 종속고울에 사용됨으로서 프로그램의 제어의 기능을 제공하여 주고 프롤로그의 디버깅(Debugging) 도구로서도 사용될 수도 있다. 수항목이 프롤로그에서 뿐만 아니라 연역 데이타베이스에서도 매우 유용한 기능이라고 생각되며, 나아가서 수항목은 최상의 답을 찾는 기능 혹은 처음에서부터 몇개의 답 또는 마지막에서부터 몇개의 답을 구하는 등의 질의어 및, $\leftarrow(A_1\&\ldots\&A_n):\mu$나 $\leftarrow B_1\&\ldots\&B_{i-1}\&(A_1\&\ldots\&A_n):\mu\&B_i\&\ldots\&B_m\&$와 같은 답을 구하는데 사용할 수 있도록 하는 것이 향후과제라 하겠다.

참 고 문 헌

[1] M. Bruynooghe and G. Janssens, "An Instance of Abstract Interpretation Integrating Type and Mode Inferencing," in *Proc. of the 5th International Conference and Symposium on Logic Programming*, Seattle, Washington, pp. 669-683, Aug. 1988.

[2] S. Ceri, G. Gottlob, and L. Tanca, *Logic Programming and Databases*, Surveys in Computer Science, New York : Springer-Verlag, 1990.

[3] C. L. Chang and R.C.T. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Orlando : Academic Press, 1973.

[4] W. Clocksin and C. Melish, *Programming in PROLOG*, New York : Springer-Verlag, 1981.

[5] S. K. Debray and P. Mishra, "Denotational and Operational Semantics for Prolog," *J. Logic Programming*, vol. 5, pp. 61-91, 1988.

[6] S. K. Debray and D. S. Warren, "Automatic Mode Inference for Logic Programs," *J. Logic Programming*, vol. 5, pp. 207-229, 1988.

[7] S. K. Debray and D. S. Warren, "Towards Banishing the Cut from Prolog," *IEEE trans. Software Eng.*, vol. 16, Mar. pp. 335-349, 1990.

[8] T. Kanamori and K. Horiuchi, "Type Inference

in Prolog and Its Application," *Proc. of the International Joint Conference on Artificial Intelligence*, pp. 704-707, 1985.

[9] J. W. Lloyd, *Foundations of Logic Programming*, New York : Springer-Verlag, 1988.

[10] P. Mishra, "Towards a Theory of Types in Prolog," in *Proc. of the IEEE International Symposium on Logic Programming*, pp. 289-298, 1984.

[11] R. A. O'Keefe, "On the Treatment of Cuts in Prolog Source-level Tools," in *Proc. 1985 Symposium on Logic Programming*, Boston, MA, pp. 73-77, July 1985.

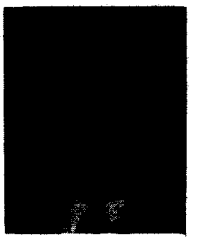
[12] N. Rowe, "Artificial Intelligence through Prolog," Englewood Cliffs, N.J. : Prentice-Hall, 1988.

[13] M. H. Van Emden and R. A. Kowalski, "The Semantics of Predicate Logic as a Programming Language," *J. of ACM*, vol. 23, no. 4, pp. 733-742, 1976.

[14] D. H. D. Warren, *An Abstract Prolog Instruction Set*, SRI International, Technical Note 309, Menlo Park, CA., 1983.

[15] J. Xu and D. S. Warren, "A Type Inference System for Prolog," in *Proc. of International conference and Symposium on Logic Programming*, Seattle, Washington, pp. 604-619, Aug. 1988.

[16] E. Yardeni and E. Shapiro, "A Type System for Logic Programs," *J. Logic Programming*, vol. 10, pp. 125-153, 1991.



남 영 광

1982년 연세대학교 수학과 졸업(이학사)
1985년 한국과학기술원 전산학과 졸업(공학석사)
1992년 Northwestern 대학교 전산학과 졸업(공학박사)
1992년~1994년 시스템공학연구소 선임연구원
1995년~현재 연세대학교 전산학과 조교수
관심분야 : 소프트웨어 공학, 프로그래밍 언어, 정보검색