論 文

병렬화를위한 논리 프로그램의 증명 방법

정회원 이 원 석*

A Proof Method of Logic Programs in Parallel Environment

Won Seok Lee* *Regular Member*

要 約

기존의 논리 프로그램을 병렬로 실행하는 방법들은 병렬화의 제약이 되었던 공유 변수들의 생산변수-사용변수의 의존관계로인해 서술적인 표현력이 강한 논리 언어에 잠재된 병렬성을 살리지 못했다. 이 논문에서는 공유 변수의 의존 관계를 제거하기위해 논리 프로그램의 실행을 증명 나무의 생성 단계와 사실을 처리하는 두 단계로 분리하는 방법을 제시한다. 첫단계에서는 변수마다 유일한 번호를 붙여 증명 나무에 생성되는 연구들을 변수 수열로 차별화하고, 사실처리시 각 변수의 값을 차별화된 변수로 구하여 증명 나무의 성공 여부를 확인할 수 있다. 따라서, 생산 변수가 값을 생산한 후 사용 변수가 있는 술어의 처리가 가능했던 기존의 병렬 처리 방식보다 더 높은 병렬성을 이룰 수 있다.

ABSTRACT

Due to the producer-consumer dependency of shared variables, the potential parallelism embeded in the logic programming language has not been fully examined. The method proposed in this paper eliminates the dependency of shared variabls by introducing number-sequenced variables in expanding an AND-OR proof tree. Basically, the execution of a logic program can be divided into two phases : expanding an AND-OR tree and proving the tree by matching facts with leaf nodes. In the coure of the first phase, a set of number-sequenced variables are produced by expanding an AND-OR tree in the breadth-first searching. Based on the information of number-sequence, each of them is verified in the second phase in order to prove the tree. Consequently, the proposed algorithm can explore more parallelism without the dependency of shared variables.

*延世大學校 電算科學科
論文番號 : 93-44

I. 서 론

논리 언어는 사실(fact)과 절(clause)로 구성되며, 사실은 변수 또는 상수 단어(literal)를 인수(argu-

ment)로 갖는 한개의 술어(predicate)이다. 절은 왼편과 오른편으로 나뉘어, 왼편은 한개의 술어로 구성되며 오른편은 여러개의 술어들의 합동(conjunction)으로 이루어진다. 논리 프로그램의 실행은 질의를 사실과 절로 증명하는 과정으로, 이 과정에서 질의의 변수 값을 구한다. 논리 언어는 서술적 표현력이 강하여 LISP과 함께 인공 지능 분야에서 대표적으로 많이 사용되는 프로그래밍 언어이다. 오늘날 컴퓨터 기술의 놀라운 발전으로 인공 지능 분야의 활발한 연구가 가능해 졌지만, 대단히 많은 심볼 데이타를 빠른 시간안에 처리하는데는 한계가 있어 왔다. 이러한 특성때문에 논리 언어는 일본의 5세대 컴퓨터 연구[1]에서 지식 공학의 기본언어로 채택되어 연구되고 있다.

논리 프로그래밍은 순차적인 실행의 절차를 표현하는 다른 프로그래밍 언어에 비해 논리의 서술적인 표현력때문에 많은 병렬성을 갖은 언어이다. 이러한 병렬성을 이용하기위해 데이타프로우(dataflow) 개념을 사용하거나[2,3], 절감(reduction) 기법의 사용[4], 스트림 방식[5]등 여러가지 방법의 연구가 시도되었다. 이중 AND/OR 프로세스 모델을 사용한 Conery[6]의 접근은 논리 프로그램이 실행되는 방식을 따르는 병렬화 방법으로, 이에대한 많은 연구[7,8,9]가 실행되었다. 논리 언어는 여러가지 병렬성을 가지며, 이중 대표적인 것은 AND-parallelism과 OR-parallelism이다[2,3]. OR-parallelism은 프로그램 실행시 여러개의 실행 대안들(altenatives)을 병렬로 처리하는 방법으로, 각각의 대안들은 서로 연관성(의존성)이 없으므로 병렬화가 간단 명료하지만, AND-parallelism은 병렬화의 최대의 난점으로 술어들간에 공유되는 생산 변수-사용 변수의 의존 관계를 효율적으로 찾아내는 방법이 그 주종을 이룬다. 대부분의 연구는 논리 프로그램의 병렬화시 기존의 직렬 처리 방법인 백트랙킹(backtracking)을 단순히 병렬화하여, 병렬 처리시 실행 속도를 저하시키는 공유 변수의 의존 관계를 제거할 수 없었다.

이 논문에서는 기존의 깊이-우선(depth-first) 방법의 병렬화를 피하고, 논리 프로그래밍언어의 특성인 서술적인 표현을 이용한 너비-우선(breadth-first) 방법을 연구하여 병렬 처리의 효율을 높이는 가능성에대해 연구해 보겠다. 기존의 병렬 처리 방법은 공유 변수의 의존 관계때문에 생산 변수가 값을 생산할 때까지 사용 변수가 있는 술어의 실행을 시작할 수 없었다. 이 논문에서 제안된 방안은 기존의 방법이 사실과 절들을 모두 동일하게 처리하는 방법을 피하고, 프로그램의 실행을 두 단계로 분리하여 처리하게 된다. 첫단계는 절들을 호출하여 사실을 비교하기전까지의 증명 나무(AND-OR tree)를 만든다. 이 나무의 잎(leaf node)들은 각각 사실과 비교되는 술어로 구성된다. 질의의 변수들에대한 치환(binding) 조건은 증명 나무의 뿌리(root)에서 각각의 잎까지의 길을 수열로 표시하여 증명 나무를 증명할 수 있는 대안들중, 그잎에 해당되는 대안들에대한 정보를 잎으로 전달한다. 둘째 단계에서는 각각의 잎들을 사실과 비교하여 성공된 잎에 변수들의 치환값을 우선 구한후, 이들 변수들에 있는 수열을 검사하여 최종적으로 질의에대한 답을 구하게 된다. 따라서, 기존의 방법인 변수의 값을 생산하여 다른 변수의 값을 구하는 방법을 배제하고, 변수의 치환을 최대한 늦게 실시하여 변수간의 상호 의존 관계를 최소화하는 방법이다. 따라서, 논리 프로그램의 실행이 변수의 의존 관계없이 분산 처리됨으로 효율적인 병렬 처리가 가능하다.

논리 언어를 너비-우선 방식으로 처리하므로 질의에있는 모든 답을 구하게 된다. 따라서, 직렬 처리 방식처럼 한개의 답을 한번에 구하는 방법을 구현할 수 없다. 하지만, 논리 언어의 병렬화가 그 직렬 처리 방식을 동일하게 모방할 필요는 없다고 본다. 병렬 컴퓨터로 실행되는 병렬 처리 알고리듬은 프로그램에서 제약이 없는 한, 모든 답을 빨리 얻는 방법이 더 합리적일 수 있다. 이 논문은 2절에서 관련 연구를 조사하고, 3절에서는 변수의 수열을 정의하여 유한한 증명 나무를 갖는 프로그램을 처리하는 방법을 소개한다. 4절에서는 3절의 방법을 무한한 증명 나무를 갖는 프로그램에 적용하는 방법을 제시한다. 마지막으로 5절에서 이 논문의 결론을 서술한다.

Ⅱ. 관련 연구

논리 언어의 대표적인 병렬성인 AND-parallelism과 OR-parallelism을 구현하기 위한 시도들은 대부분 프로그램을 구성하는 술어간의 데이타 의존 관계를 분석하여, 의존 관계가 없는 술어들을 병렬로 실행하는 방법이 그 주종을 이룬다. 백워드 실행(backward execution) 알고리듬[6]은 절의 술어를 생산 술어와 사용 술어로 구분하여 데이타 의존 그래프를 만든다. 프로그램의 실행은 절의 왼편이 OR-프로세스를, 절의 오른편이 AND-프로세스를 생성하고, AND-프로세스는 해당되는 오른편의 생산 술어를 먼저 실행하

기 위해 생산 술어들의 OR-프로세스를 실행한다. 여기서 생산된 변수의 값으로 나머지 사용 술어를 실행하게 된다. 만약, OR-프로세스가 실패하면, AND-프로세스에 알리게 되며, 되돌아 가는(backtracking) 술어는 변수들의 의존 관계를 나타내는 데이타 의존 그래프와 실행 정보를 저장하는 구조에서 가능한 술어를 선택한다. 실행 정보를 저장하는 구조는 생산 술어에서 사용 술어순으로된 선형 리스트와 실패 리스트등을 사용한다. 이러한 정보는 실행 결과에따라 동적으로 바뀌게 된다. 이 알고리듬은 실패한 사실에 대한 올바른 정보를 갖지 않으므로 AND-프로세스의 답을 정확히 구하지 못한다[7].

대부분의 AND-parallelism의 실현은 [6]에서와 같이 두개의 술어가 변수를 공유할때 이들을 차례로 실행하는 방법과 모든 술어를 동시에 실행하여 그 결과를 다시 동일화(unification)하여 질의에대한 답을 구하는 방법[9]이다. 첫번째 방법에서는 프로그램에 있는 여러가지 의존관계를 먼저 찾아서 일정한 형태로 보관하고, 이를 프로그램의 실행에 적용한다. 이러한 의존관계는 프로그램상에서 변수에 입출력 표시를 하는 방법[6,10,11]과 프로그래머의 작업을 줄이고 실행 오류의 방지를 위해 프로그램을 컴파일하여 술어들간에 존재하는 공유 변수들의 의존관계를 자동으로 찾아내는 방법[8,12]이 있다. [10]에서는 술어에서 사용하는 변수에 서로의 연관 관계를 표시하고, [11]에서는 절마다 보호 술어라는 술어를 만들어 절에있는 보호 술어를 먼저 실행한 후, 절의 나머지 술어를 실행한다. 이때, 보호 술어는 프로그램을 제어하는 술어로 변수간의 관계가 표시된다. 반면에, 변수간의 연관 관계를 자동으로 찾는 방법인[8,12]에서는 컴파일시 술어에 있는 변수들의 공유 형태에대한 정보를 찾아 술어들을 연결하는 정적인 연결 그래프(connection graph)를 만든다. 프로그램의 실행시 이 연결 그래프를 토대로 술어의 실행이 결정된다.

OR-parallelism[13]은 여러개의 독립적인 증명 방안을 동시에 처리하는 방법으로 그 실현이 비교적 간단 명료하다. 하지만, 변수에 치환(binding)된 여러개의 값들이 동시에 생기므로 이들을 관리하는 것이 복잡하다. OR-parallelism에대한 대부분의 연구는 논리적인 방향보다는 이러한 관리의 효율적인 구현이 대부분이다.

기존의 AND-parallelism을 얻는 방법들은 프로그램을 실행하기전에 프로그램에 있는 변수들의 사용용도를 표시하거나, 프로그램을 컴파일하여 술어들의 공유 변수간에 존재하는 의존관계를 찾는다. 따라서, 공유 변수를 갖는 술어들은 일정한 순서로 실행되므로 생산 술어와 사용 술어를 동시에 실행할 수 없다. 이 논문에서 제안된 방법도 절에 존재하는 공유 변수를 찾지만, 이들 공유 변수들은 기존의 방법과는 달리 동시에 처리 될 수 있다. 또한, OR-parallelism은 뿌리부터 앞까지의 길에대한 모든 정보를 잎에 전달하여, 이들 잎들에있는 정보에따라 질의에 대한 여러개의 다른 집합의 답을 찾을 수 있으므로 복잡한 관리를 피할 수 있다.

Ⅲ. 유한한 증명 나무의 처리

논리 프로그램의 실행은 실행 조건에 맞는 절들을 생성하는 증명 나무의 처리로 간주할 수 있다. 이때, 사이클을 갖지않고 유한한 깊이로 생성를 끝내는 증명 나무를 유한한 증명 나무라고 한다. 따라서, 유한한 증명 나무의 잎들은 더이상 다른 술어를 생성시킬 수 없는 술어로 프로그램의 사실들이다. 이절에서 유한한 증명 나무의 처리 방법에대한 연구를 제시하고, 무한한 증명 나무의 처리는 다음절에서 기술한다. 3.1절과 3.2절에서는 제안하는 병렬 처리 기법의 1단계인 증명 나무의 생성 과정을 설명하고, 3.3절에서는 그 2단계인 사실 처리 과정과 증명 나무의 증명 과정을 제시한다.

3.1 변수 번호및 변수 수열

증명 나무를 너비-우선(breadth-first) 방법으로 처리하기위해 다음과 같이 프로그램에서 사용되는 변수에대해 번호를 붙인다.

〈변수 번호〉

(1)각 절의 오른편에 있는 변수들중 동일한 변수에 대해 일련 번호를 붙인다. 즉, 변수 집합 VS(v) ={v1, v2,..., vn}은 변수 v가 절의 오른편에서 n번 사용되며 각각의 변수에 1부터 n까지의 일련 번호를 붙인다.

(2)동일한 왼편 술어를 갖는 절들을 집합으로 모아 일련 번호를 붙인다. 즉, 절 집합 CS(p) ={p1, p2,..., pm}은 왼편의 술어가 p인 절들을 모은 집합으로 각각의 절에 1부터 m까지의 일련 번호를 붙인다. 이 번호는 왼편에서 사용하는 모든 변수와 상수에 붙인다.

다음은 유한한 깊이를 갖는 간단한 프로그램으로,

술어 r, f, g, h는 모두 사실(fact)만을 갖는다.

가) p(X,Y,Z) : -q(X,Z), r(Y,Z)
나) p(a,Y,Z) : -r(a,Y), f(Z)
다) q(X,Y) : -r(X,Y), f(Y)
라) q(X,Y) : -g(X,Y), h(X)

위의 프로그램의 변수에대해 변수 번호를 각각 붙여보면 다음과 같다.

가) p(X1,Y1,Z1) : -q(X1,Z1), r(Y1,Z2)
나) p(a2,Y2,Z2) : -r(a,Y1), f(Z1)
다) q(X1,Y1) : -r(X1,Y1), f(Y2)
라) q(X2,Y2) : -g(X1,Y1), h(X2)

즉, 절 가)에서 오른편에있는 변수 Z는 2번 사용되었으므로 변수 번호 1과 2를 각각 붙였으며, 변수 X와 Y는 한번만 사용되었으므로 변수 번호 1을 각각 붙였다. 또한, 가)와 나)의 왼편 술어가 동일하므로 가)의 p(X,Y,Z)에있는 변수에 모두 변수 번호 1을, 나)의 p(a,Y,Z)에 변수 번호 2를 각각 붙였다. 개념적으로, 절의 왼편은 증명 나무의 OR 노드(node)에, 오른편은 AND 노드에 해당된다. 따라서, 왼편의 변수 번호는 OR parallelism을, 오른편의 변수 번호는 AND parallelism을 이루는데 사용한다. 각각의 변수에 정의된 변수 번호는 변할 수 없으며, 프로그램 실행시 증명 나무를 생성할때 이용된다.

논리 프로그램 실행시, 질의는 술어의 합동(conjunction)이므로 (1)의 방법으로 질의에 있는 변수들에 변수 번호를 붙인다. 예를들면, 질의 ?p(a,X,Y), q(X,Y)은 ?p(a,X1,Y1), q(X2,Y2)로 각각의 변수에 변수 번호를 붙인다. 질의를 증명하기 위해 증명 나무가 생성되며, 이때 변수를 사용할때마다 그 변수에 정의된 변수 번호를 연결(concatenation)한다. 변수의 이름은 동일화(unification) 법칙에 따라 변할 수 있지만, 변수 번호는 변하지 않고 증명 나무가 생성되는 길을 따라 연결된 수열을 만든다. 따라서, 사실을 비교하기 전까지 증명 나무를 생성하면, 잎이 논문에서는 논리 프로그램의 사실과 절의 처리를 분리하기위해 증명 나무를 절들만으로 생성하므로 증명 나무의 잎들은 프로그램의 사실을 처리하기 바로 전 단계의 술어들이 된다. 따라서, 이러한 증명 나무는 기존의 나무(tree)에대한 정의로부터 변형된 구조를 갖는다. 이는 동일한 술어가 절의 왼편과 사실에 사용되었을때, 이 술어에대한 노드는 사실을 처리하기 바로전의 잎노드도 되며 동시에 절을 처리하는 중간

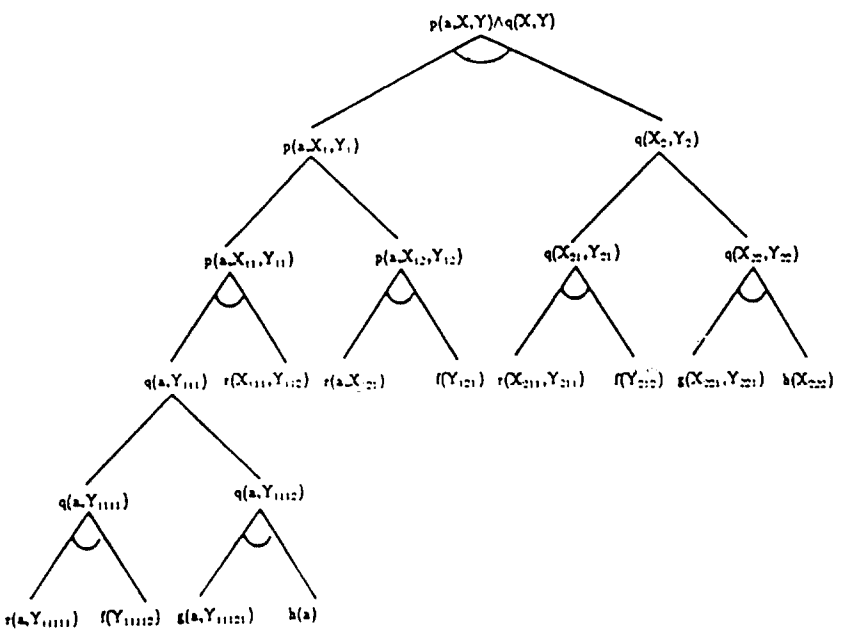


그림 1. ?p(a,X1,Y1), q(X2,Y2)의 증명 나무

노드도 되기 때문이다. 이 논문에서는 증명 나무를 이렇게 변형된 구조를 갖는 나무로 정의하고, 4절에서 잎노드와 중간 노드의 역할을 동시에하는 노드를 갖는 증명 나무에대한 예제를 든다.

질의에대한 답은 질의의 변수가 사실의 상수와 치환된 결과로써, 증명 나무의 OR 노드때문에 여러개의 다른 대안들을 가질 수 있다. 각 대안은 변수들이 치환되는 조건을 가지며, 이 조건을 변수의 치환 조건이라고 정의한다. 예를들면, 그림 1의 증명 나무에서, 변수 X와 Y는 다음과 같은 치환 조건을 갖는다.

변수 X :

BX1) X111, X211

BX2) X111, X221, X222

BX3) X121, X211

BX4) X121, X221, X222

변수 Y :

BY1) Y11111, Y11112, Y112, Y211, Y212

BY2) Y11111, Y11112, Y112, Y221

BY3) Y11121, Y112, Y211, Y212

BY4) Y11121, Y112, Y221

BY5) Y121, Y211, Y212

BY6) Y121, Y221

그림 2. 변수의 치환 조건들

위와 같이 수열로 표시된 변수들을 수열 변수라고 정의하고, 각 수열 변수의 수열을 변수 수열이라고 정의한다.

사실의 상수가 질의의 변수를 치환하기 위해서는 변수의 치환 조건들중의 적어도 어느 한 조건을 만족하여야 한다. 즉, 상수 a가 변수 X를 BX2로 치환하려면, r(a,−), g(a,−) 그리고 h(a)가 사실로 증명되어야 한다.("−"은 변수 Y에 해당됨). 그리고, 질의의 변수들 모두가 각각의 치환 조건을 만족할때, 질의에대한 답을 구할 수 있다.

개념적으로, 변수 수열은 증명 나무의 뿌리에서 잎까지의 길에대한 정보를 갖는다. 변수 수열에서 i번째 수는 만약 i가 홀수이면, 증명 나무의 AND 노드에 해당하며, i가 짝수이면, OR 노드에 해당한다. 이러한 성질을 이용하여, 변수에대한 치환 조건을 변수 수열들의 집합에서 정할수 있다. 위의 예제에서 변수 X에대한 모든 수열을 원소로하는 집합을 변수 X의 수열 집합이라고 한다. 그리고, 각 수열의 첫번째 수가 동일한 수열끼리 겹끼운(nested) 소집합을 만든다. 이런 과정을 더이상 소집합을 만들수 없을때까지 반복하여 만든 집합을 변수의 최대 AND 집합이라고 한다. 이때, 증명 나무의 AND 노드에 해당되는 소집합을 AND 집합, OR 노드에 해당되는 소집합을 OR 집합이라고 정의한다. 변수 X의 최대 AND 집합과 그 소집합들을 다음과 같이 만들수 있다.

X의 수열 집합 {111, 121, 211, 221, 222}
1st level classification : {[111,121], [211,221,222]}
2nd level classification : {[{111},{121}], [{211},{221,222}]}
3nd level classification : {[{111},{121}], [{211},{[221],[222]}]}

여기에서 {...}은 AND 집합을, [...]은 OR 집합을 나타내며, 한개의 원소만을 갖은 집합은 의미가 없으므로 이를 제거하면 X와 Y의 최대 AND 집합은 다음과 같다.

X : {[111, 121], [211, {221, 222}]}
Y : {[{[{11111, 11112}, 11121], 112}, 121], [{211, 212}, 221]}

AND 집합이 수열만을 갖고 각각의 수열이 사실과 치환할때 같은 상수로 치환되면, 그 AND 집합을 참(true) 집합이라고 정의한다. 반면에, 수열만 갖는 OR 집합에서 어느 한 수열만이라도 상수와 치환되면, 그 OR 집합을 참집합이라고 정의한다. 반복적으로, AND 집합에 있는 모든 소집합이 참집합일때, 그 AND 집합은 참집합이고, OR 집합에 있는 어느 한 소집합만이라도 참집합이면 그 OR 집합은 참집합이다. AND 집합이나 OR 집합은 겹끼워진(nested) 형태로 정의 되었으므로 이러한 집합의 원소는 반복적으로(recursive) 정의할 수 있다. 즉, 이러한 집합의 원소는 다음 깊이(next depth)의 소집합을 원소로 갖고, 반복적으로 그 소집합의 원소도 집합의 원소로 갖는다. 따라서, 이러한 집합의 부분 집합은 반복적으로 정의된 원소로 구성된 집합이다.

3.2 증명 나무의 증명 대안들을 찾는 방법

증명 나무는 OR 노드때문에 여러개의 대안으로 증명할 수 있으며, 이들 대안들은 각 변수의 치환 조건들의 결합으로 이루어진다. 즉, 나무 생성시 각 변수간의 연관 관계에따라 여러개의 대안이 생기며 각 대안을 이루는 변수들의 치환 조건을 동시에 만족하였을때 그 대안의 증명을 성공할 수 있다. 그림 2의 변수 X와 Y의 치환 조건인 BX1~BX4과 BY1~BY6은 다음의 관계를 갖는다.

B1: BX1 <——> BY1

B2: BX1 <——> BY3

B3: BX2 <——> BY2

B4: BX2 <——> BY4

B5: BX3 <——> BY5

B6: BX4 <——> BY6

그림 3. 증명 나무의 증명 대안들

질의의 변수 X와 Y는 B1~B6의 여섯가지 대안을 갖고, 각각의 대안을 구성하는 변수들의 치환조건은 동시에 만족되어야 한다. 즉, 대안 B1을 만족하려면 변수 X는 BX1를, 변수 Y는 BY1을 동시에 만족하여야 한다.

변수간의 연관 관계는 앞에있는 수열 변수간의 관계와 증명 나무를 생성하면서 생기는 수열 변수간의 관계로 구분된다. 각각의 앞에는 서로 다른 수열 변수가 존재할때, 이들 변수들은 하나의 투풀(tuple)을 형성한다. 예를 들면, 앞 r(X111, Y112)에서 수열 변수 X111과 Y112는 동시에 한 사실과 비교(matching)된다. 이러한 조건을 투풀링(tupling) 조건이라고 정의한다. 그림 1에서는 X111-Y112와 X211-211, X221-Y221의 투풀링 조건이 존재한다.

증명 나무를 생성하면서 발생하는 수열 변수간의 연관 관계는 나무의 뿌리에서 앞까지의 길을 따라서 생기는 다른 수열 변수들과의 관계이다. 이러한 연관 관계에대한 정보를 앞에 전달하기 위해 노드 N에대한 수열 변수 집합 R(N)을 다음과 같이 정의한다.

R(N)=delta(R(M))

노드 M : 노드 N의 부모(parent)

R(뿌리)={질의의 모든 수열 변수}

위의 식에서 delta는 노드 M에서 노드 N으로 증명 나무가 생성될때 각 변수 수열의 변화를 표시한다. 노드 N의 수열 변수 집합은 뿌리에서부터 자신까지의 길을 따라가는 수열 변수의 변화를 저장한다. 또한, 앞의 수열 변수 집합은 그 앞까지의 길을 따라오는 도중에 다른 길을 따라 간 수열 변수들을 저장한다. 예를들면, 그림 1의 수열 변수 집합은 다음과 같다.

R(뿌리)={X1, Y1, X2, Y2}
R(r(a, Y11111))={X11, Y11111, X2, Y2}
R(r(X111, Y112)={X111, Y112, X2, Y2}
R(r(X211, Y211)={X1, Y1, X211, Y211}
R(h(a))={X11, Y1112, X2, Y2}
R(q(a, Y111))={X11, Y111, X2, Y2}

앞의 수열 변수 집합으로 그 앞을 포함하는 증명 나무의 증명 대안들을 찾을 수 있다. 예를들면, 그림 1에서 다음의 수열 변수 관계를 갖고 앞 r(X111, Y112)가 포함되는 증명 나무의 대안들을 찾아 보자.

투풀링 조건 : X111-Y112, X211-211, X221-Y221
R(r(X111, Y112))={X111, Y112, X2, Y2}
변수의 최대 AND 집합 :
X : {[111, 121], [211, {221, 222}]}
Y : {[{[{11111, 11112}, 11121], 112}, 121], [{211, 212}, 221]}

우선, 변수 X의 최대 AND 집합에서 R(r(X111, Y112)에 있는 X의 변수 수열을 원소로 갖는 가장 작은 AND 집합을 찾는다. 이때, 만약 R의 변수 수열이 최대 AND 집합에있는 수열의 부분인 경우에는 R의 변수 수열을 갖는 가장 큰 OR 집합을 찾는다. 개념적으로 앞에있는 변수 수열은 항상 그 변수의 최대 AND 집합의 원소이고, 그 변수 수열의 마지막 수는 항상 뿌리에서 그 앞까지의 길에있는 마지막 AND 노드에 해당된다. 따라서, 그 앞이 갖는 대안을 성공하려면, 이 마지막 AND 노드가 성공하여야 한다. 이 마지막 AND 노드는 최대 AND 집합에서 그 앞의 수열을 갖는 가장 작은 AND 집합에 해당한다. 반면에, 앞에 있지 않은 변수 수열은 뿌리에서 앞까지의 중도에 있는 AND 노드에서 다른 길에있는 OR 노드를 따라간 수열 변수이다. 그러므로, 이 변수 수열이 성공하려면 그 OR 노드가 성공하면 되므로, 그 수열을 부

분으로 갖는 가장 큰 OR 집합에 해당된다.

다음은 수열 변수 집합 R(r(X111, Y112))에있는 변수 수열들을 증명 나무에대한 변수 X의 최대 AND 집합에서 그 잎에대한 변수의 최대 AND 집합을 구한 결과이며, 잎에대한 변수의 최대 AND 집합은 증명 나무에대한 최대 AND 집합의 부분 집합이 된다.

잎 r(X111,Y112)의 X 최대 AND집합 : {111,[211,{221,222}]}
R의 수열 변수 X111 → 111
R의 수열 변수 X2 → [211, {221, 222}]

마찬가지로, Y의 최대 AND 집합에서 노드 r(X111, Y112)에서의 변수 Y에대한 최대 AND 집합을 구하면 다음과 같다.

Y의 최대 AND 집합 : {{[{11111,11112},11121],112},[{211,212},221]}
R의 수열 변수 Y112 → {[{11111,11112},11121],112}
R의 수열 변수 Y2 → [{211,212},221]

이제 투풀링 조건을 적용하여, 앞에대한 변수들의 최대 AND 집합에서 서로 동시에 만족되어야 하는 관계를 찾는다. 이를 위해, 투풀링 조건의 변수 수열을 원소로하는 가장 작은 AND 집합을 각각의 최대 AND 집합에서 찾는다. r(X111, Y112)의 변수 X와 Y가 동시에 만족되어야 하는 소집합들을 그 잎에서의 각각의 최대 AND 집합에서 구하면 다음과 같다.

투플링 조건	X의 소집합	Y의 소집합
X111−Y112 :	111	{[{11111,11112},11121],112}
X211−Y211 :	211	{211, 212}
X221−Y221 :	{221, 222}	221

이들 소집합들간의 결합으로 변수들의 치환 조건이 결합되어 증명 나무의 대안을 만든다.

노드 r(X111,Y112)의 X와 Y의 최대 AND 집합에서 OR 집합을 제거하여 다시 표시하면 다음과 같이 변수의 치환 조건을 구할 수 있다.

X : {111,211}, {111,221,222}
Y : {11111,11112,112,211,212}, {11111,11112,112,221}
{11121,112,211,212}, {11121,112,221}

위의 X와 Y의 치환 조건들을 수열 변수들의 투풀링 조건에따라 결합하여, 다음과 같이 증명 나무의 여러 개의 대안 B1~B6중에서 노드 r(X111, Y112)가 성공할때 증명될 가능성이 있는 대안들을 찾을 수 있다.

X:{111,211} Y:{11111,11112,112,211,212} →B1
X:{111,211} Y:{11121,112,211,212} →B2
X:{111,221,222} Y:{11111,11112,112,211} →B3
X:{111,221,222} Y:{11121,112,221} →B4

대안 B1은 X111−Y112와 X211−211의 투풀링 조건에따라 결합되었고, 이와 같은 방법으로 다른 대안들을 구한다.

이와 같이 증명 나무가 생성된 정보를 잎으로 보내어 잎의 변수들이 사실과 비교될때, 만족되어야 하는 대안들을 찾는다. 각각의 잎들이 자신과 연관된 대안들을 구하여, 사실과의 비교가 실패했을때, 다른 잎들이 자신과 연관된 대안들을 증명하는 불필요한 노력을 줄일 수 있다. 그림 1의 잎 h(a)와 같이 아무런 변수를 갖지 않는 잎도 다른 노드의 처리와 같다. 즉, R(h(a))에서 잎 h(a)의 대안들인 B2와 B4를 찾을 수 있다. 만약, h(a)가 사실과 비교하여 성공하지 않으면 대안 B2와 B4의 증명은 실패한다. 이런 방법으로 각 잎에 연관된 대안들을 구하여, 중복되는 대안을 제거하면 그림 1의 증명 나무가 갖는 모든 대안 B1~B6들을 구할 수 있다.

〈동일화 (unification)〉

질의에대한 증명 나무를 생성하면서 동일화 작업(unification)이 실행되며, 변수가 절의 오른편에서만 사용될때 이런 변수를 숨은(hidden) 변수라고 한다. 한 절은 증명 나무에서 여러번 호출될 수 있고, 각각의 호출에서 동일화되지 않는 이러한 숨은 변수들은 새로운 변수로 간주됨으로 호출될때마다 새로운 이름과 수열이 시작되어야 한다. 기존의 방법에서는 이러한 변수들도 사실과 비교되어 상수로 치환되므로 새로운 변수 이름을 만들 필요가 없었지만, 이 논문에서 제안된 방법은 증명 나무가 생성된 후 치환을 실행하므로 변수의 이름과 수열을 정확히 유지하여야 한다. 예를들면, 다음의 프로그램에서 변수 Y는 호출시 동일화되지않음으로 새로운 변수로 간주하여야 한다.

p(X) : −q(X,Y), r(Y) ⇒ p(Xt) : −q(X1,Y1), r(Y2)

따라서, 변수 Y는 p(X)를 호출할때마다 새로운 이름과 새 수열을 갖는다. 증명 나무가 생성된 후, 잎에는 질의에 사용된 수열 변수와 숨은 변수들의 수열 변수가 생기며, 숨은 변수의 처리는 질의에있는 변수와 동일하며 단지 치환값을 답으로 출력하지 않고, 그 노드의 대안을 찾는데 사용된다.

동일화시 변수와 상수와의 치환은 앞으로 전달되어 사실과 치환된 다른 변수 수열과함께 그 잎의 대안들을 증명하는데 사용된다. 이렇게 동일화시 상수로 치환되는 수열 변수는 오직 한개의 치환값을 가지며 수열이 짝수 번째(OR node)로 끝나게 된다. 따라서, 증명 나무의 대안을 찾을때 다른 변수의 수열 변수들중에 증명 나무의 뿌리에서부터 동일한 길을 따라내려온 수열 변수와 결합된다. 또한, 절의 왼편에 있는 한개 이상의 변수들이 각각 상수로 치환되면, 이들 수열 변수들에대한 투풀링 조건이 생기게 된다. 다음의 절에 ?p(X1, c, Y1)을 실행하면, 그림 4의 증명 나무를 생성한다.

p(X1, Y1, Z1) : −q(X1, Z1, b), r(Y1, Z2)
p(a2, X2, Y2) : −h(X1, Y1)

p(X1, c, Y1)과 절의 왼편인 p(a2, X2, Y2)가 동일화될때, 수열 변수 X12=a의 치환이 생기고 잎 h(c, Y121)의 수열 변수 집합은 R(h(c, Y121))={X12, Y121}이 된다. 따라서, 변수 X와 Y의 최대 AND 집합은

X={[111, 12]}
Y={[{111, 112}, 121]}

이 되며, 투풀링 조건이 X111−Y111이므로 다음의 대안들을 갖는다.

X:{111} Y:{111, 112} →C1
X:{12} Y:{121} →C2

대안 C1은 투풀링 조건 X111−Y111으로 결합되었고, 대안 C2는 동일화시 생긴 수열 변수 X12의 수열 12와 동일한 길로 생성된 수열 변수 Y121를 결합하였다.

3.3 증명 나무의 증명

3.2절의 1단계에서 각각의 잎에대한 증명 나무의 증명 대안들을 구했다. 2단계에서는 각 잎을 사실과 비교한 결과로 질의를 증명하는 과정이다. 증명 나무를 증명하는 과정은 증명 나무에 존재하는 대안들중 적어도 한 대안에대한 변수들의 치환 조건과 투풀링 조건을 만족하는 상수들을 구하는 과정이다. 예를 들면, 대안 B1에서 변수 X의 치환 조건은 X:{111, 211}이므로 수열 변수 X111과 X211을 동시에 만족하는 상수가 존재해야 하며, 마찬가지로 변수 Y의 치환 조건에 있는 수열 변수를 동시에 만족하는 상수가 존재해야 한다. 또한, X와 Y의 수열 변수간에 존재하는 모든 투풀링 조건을 만족하는 상수 투풀이 존재할 때 대안의 증명이 성공한다. 우선, 잎들을 사실과 비교하여 성공 여부를 확인한다. 한개 이상의 사실이 치환될 수 있으므로, 각각의 잎의 변수들은 여러개의 치환값을 가질 수 있다. 2단계의 과정을 설명하기 위해 다음과 같은 사실들을 사용하여 그림 1의 증명 나무를 처리해 보겠다.

g(a,b)	r(a,b)	f(a)	h(a)
g(c,e)	r(c,d)	f(e)	h(b)

우선, 각각의 잎들을 사실과 비교하여 변수에 치환되는 값을 구하면 다음과 같다.

X121	X222	Y11111	Y11112	Y11121	Y121	U212
b	a b	b	a e	b	a e	a e

X111−Y112		X211−Y211		X221−Y221	
a	b	a	b	a	b

만약 동일화에의해서 수열 변수의 치환값이 생기면, 그 수열 변수는 오직 하나의 치환값을 가지며 사실과 비교하여 구한 다른 수열 변수의 치환값과 동일하게

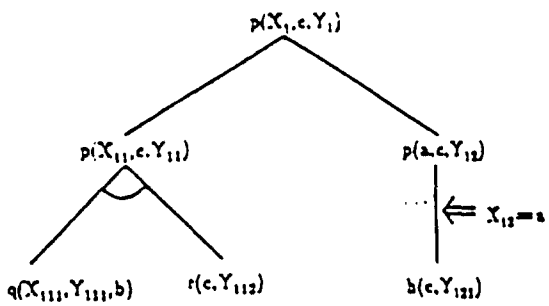


그림 4. 동일화 증명 나무

증명 나무를 증명하는데 사용한다.

각각의 대안은 수열 변수들의 치환 조건과 투풀링 조건을 갖는다. 우선, 투풀링 조건이 한개 이상 존재하면, 각각의 투풀링 조건에 치환된 값들을 투풀링 조건의 공유 변수를 기준으로 조인(join) 작업을 실행한다. 조인 작업은 데이타 베이스의 실행 방법중의 하나로 교집합과 유사하지만, 데이타를 투풀 형태로 유지하여 처리 한다. 예를 들면, 대안 B4는 X111-Y112와 X221-Y221의 투풀링 조건을 갖고 있으므로, 이들의 치환 값을 투풀링 조건의 공유 변수인 XY로 조인한다. 그 결과는 {XY|ab}가 된다. 여기서, 만약 Y221이 다른 변수 Z의 수열 변수라고 가정하고 공유 변수인 X로 조인하면, 그 결과는 {XYZ|abb, cde}가 된다. 이렇게 조인 작업을 실행하는 이유는 사실을 나타내는 기본 단위가 투풀이므로 한개 이상의 변수를 동시에 한 사실의 값으로 치환할때, 이들 값들은 투풀로 존재해야 그 의미를 갖는다.

각 대안 별로 투풀링 조건에대한 조인 작업의 결과와 대안을 이루는 나머지 수열 변수들의 치환된 값들을 수열 변수의 변수를 기준으로 조인 작업을 실행하여 성공되는 투풀이 변수들의 최종값(instantiation)이 된다. 조인의 결과가 공집합이되면, 변수들의 최대 AND 집합이 참집합이 안되므로 그 대안은 증명이 실패한다. 모든 대안이 실패했을때 증명 나무의 증명은 실패한다. 다음은 각각의 대안에대한 증명 과정을 보여 준다.

B1 : join(X111-Y112,X211-Y211) on XY → T1={XY|ab,cd}
  join(T1,Y11111) on Y → T2={XY|ab}
  join(T2,Y11112) on Y → T3={ } : 실패
B2 : join(X111-Y112,X211-Y211) on XY → T1={XY|ab,cd}
  join(T1,X222) on X → T2={XY|ab}
  join(T2,Y11112) on Y → T3={ } : 실패
B3 : join(X111-Y112,X221-Y221) on XY → T1={XY|ab}
  join(T1,Y11112) on Y → T2={ } : 실패
B4 : join(X111-Y112,X221-Y221) on XY → T1={XY|ab}
  join(T1,X222) on X → T2={XY|ab}
  join(T2,Y11121) on Y → T3={XY|ab}
  ⇒PROVED (BX2와 BY4의 모든 수열 변수)
B5 : join(X121,X211-Y211) on X → T1={ } : 실패
B6 : join(X121,X221-Y221) on X → T1={ } : 실패

따라서, 그림 1의 증명 나무는 6개의 대안중 오직 B4만이 증명에 성공하였다.

이제까지 증명 나무를 너비-우선 방법으로 생성하여 병렬 처리하는 방법을 제시하였다. 증명 나무의 증명 과정을 두 단계로 나누어, 그 첫 단계에서는 질의에대한 여러개의 대안을 잎별로 구한 후, 이들 대안들의 성공을 증명할 수 있는 조건들을 찾아 낸다. 둘째 단계에서는 각각의 잎을 사실과 비교하여 각 잎의 변수에대한 값을 구한후, 이들 값들이 증명 나무의 대안을 만족하는지를 첫째 단계에서 구한 조건과 비교하여 변수의 최종적인 값(instantiation)을 구한다.

〈CUT의 처리〉

논리 프로그램에서 CUT의 사용은 백트랙킹을 방지하는데 사용한다. 즉, p : -q, r, !, s를 실행할때, q와 r사이는 백트랙킹이 가능하지만, 일단 q와 r을 만족하는 치환을 찾고 CUT인 "!"을 지나면 s의 처리가 실패하더라도 다시 r로 백트랙킹을 할 수 없다. 따라서, CUT을 사용하기전까지의 합동(conjunction)에서 각각의 변수를 치환하는 첫번째 답을 구한후, 이 치환으로 s를 만족하는 최종 답들을 구하게 된다. 첫번째 답이란, 사실과 절이 프로그램에 사용된 순서를 고려하여, 처음으로 그 합동(conjunction)을 만족하는 답이다. 이 논문에서 제안된 방법은 한번에 한개의 답을 구하지 않고, 질의에대한 모든 답들을 동시에 찾는 방법이다. 따라서, 여러개의 답중에 프로그램에 있는 사실과 절의 순서를 고려한 답을 찾기위해서는 절에대한 변수 번호를 절이 프로그램에서 사용된 순서대로 붙여야하며, 사실을 비교할때 각각의 사실의 프로그램에 나타난 순서를 유지하여야 한다. CUT이 사용되기 전까지의 합동에 있는 변수들의 최대 AND 집합을 구한 후, 변수 수열의 OR-노드에 해당되는 수가 가장 적은 치환 조건을 만족하는 변수의 값만을 사용하여 나머지 변수들의 값을 구한다.

Ⅳ. 무한한 증명 나무의 처리

질의를 증명하는 증명 나무가 끝없이 생성될때 이런 증명 나무를 무한한 증명 나무라고 한다. 이와 같은 나무는 생성시 일정한 사이클을 갖고 같은 형태의 부분이 계속하여 무한히 생성된다. 이러한 사이클은 자기 호출(recursive) 절이나 절들간의 호출이 일정한 형태로 반복하여 생길때 발생한다. 사이클은 프로그램에서 잠재적으로 존재하며 질의하는 방법에따라 유한한 나무를 생성할 수도 있다.

무한하게 생성되는 나무의 처리 과정을 논하기전에, 먼저 논리 언어의 기본적인 데이타 구조인 리스트(list)를 나타내는 방법을 기술해 보기로 하자. 논리 언어의 리스트는 [a,b,c] 또는 [a|b,c]로 표시하며, 그림 5와 같은 구조를 갖는다. 이 리스트에서 "a"를 리스트의 머리(head), [b,c]를 꼬리(tail)라고 한다. 변수가 리스트로 치환될때, 각각의 원소를 나타내기 위해서 리스트의 머리를 "h"로, 꼬리를 "t"로 표시하면 다음과 같이 리스트 변수의 원소를 나타낼 수 있다.

$X=[a,b,c] \Rightarrow Xh=a,\ Xth=b,\ Xtth=c,\ Xttt=[\,]$

3.1절의 변수 수열과 같은 방법으로 "h"와 "t"를 연결하여, 각각의 변수를 나타낸다. 이런 변수를 원소 변수라고 정의하고, 원소 변수의 "h"와 "t"의 열을 원소열이라고 정의한다. 따라서, 한 변수의 원소 변수들은 그 변수를 이루는 원소들에대한 변수가 되며, 이들 원소 변수들을 원소열에따라 정리하면 그 변수가 나타내는 리스트가 된다. 같은 변수의 원소 변수들은 원소열이 다르면 각각 다른 값으로 치환될 수 있다. 또한, 원소 변수는 리스트를 치환할 수 있으므로 겹끼운 리스트를 표시할 수 있다. 예를들면, 만약 변수 V의 원소 변수의 값이 각각 Xhh=a, Xht=[ ], Xth=b, Xtt=[ ]이면, V는 리스트 [[a],b]를 치환한다. 원소 변수를 갖는 변수는 각각의 원소 변수들의 치환이 성공했을때 변수의 치환이 성공한다. 이런 원소 변수들로 표시된 절로 증명 나무를 생성하면, 3절에서와 같이 수열을 갖는 원소 변수가 생기며 이를 원소 수열 변수라고 정의한다.

무한한 증명 나무의 기본적인 처리 방법은 무한하게 생성되는 나무를 일정한 깊이를 정하여 그 깊이까지 처리한후, 깊이를 늘려가면서 더이상 새로운 답이 없을 때까지 나무를 증명한다. 그 깊이는 두가지 방법으로 정할 수 있다. 임의의 깊이를 정하는 방법과 사이클을 찾아 그 사이클의 길이를 깊이로 정하는 방법이다. 임의의 깊이를 정하는 방법은 정확한 실패 나무를 찾기 어려우며, 사이클의 길이로 정하는 방법은 그 사이클을 질의가 변할때마다 찾아야 하므로 상당히 비효율적이다. 하지만, 질의의 AND-OR 나무를 생성하기전에 프로그램을 컴파일하여 프로그램에있는 모든 가능한 사이클을 찾는다면 나무 생성시 사이클을 찾는 시간을 제거할 수 있다. 이 논문에서 제안한 방법은 절의 처리와 사실의 처리를 구분하여 실행

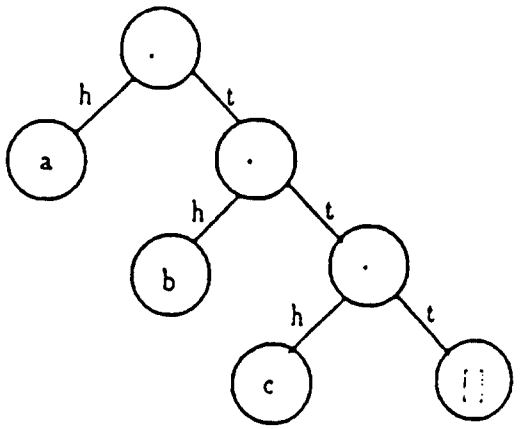


그림 5. 리스트 [a,b,c]의 구조

함으로 컴파일시 찾은 잠재적인 사이클을 나무를 생성하면서 적용할 수 있다. 이러한 사이클을 효율적으로 찾는 방법은 또다른 연구 분야이므로 이 논문에서는 3절에서 제안한 방법을 무한한 증명 나무에 적용하는 방법만을 제시한다.

다음은 list에 원소를 첨가하는 자기 호출 프로그램과 멤버를 확인하는 프로그램이다.

```
(1) append([ ],L,L).
(2) append([X|U],V,[X|W]) : -append(U,V,W)
(3) member(X,U) : -append(V,[X|W],U)
```

(1)은 사실이므로 (2)의 변수들에 변수 번호와 원소 부호를 다음과 같이 붙인다.

```
(2) append([X1-h|U1-t], V1, [X1-h|W1-t]) :
      -append(U1,V1,W1)
(3) member(X1, U1) : -append(V1, [X1|W1], U1)
```

원소 부호는 변수가 리스트로 치환되어 그 변수를 머리와 꼬리의 원소로 분할할때 사용하므로 절의 왼편에있는 리스트 구조에 붙인다. 따라서, 원소 부호는 변수가 머리와 꼬리로 분할되어 동일화될때 그 변수의 원소열에 연결된다.

위의 프로그램에 ?append([a],[b],X)를 실행하면 그림 6의 증명 나무에서 왼편의 부분 나무(sub-tree)를 생성한다. 나무 생성시 동일화에의해 생기는 원소 수열 변수의 치환 값은 X11-h=a이다. 이 나무에는 두개의 잎이 있으며, 각각의 노드에 전달된 정보는

다음과 같다.

N1 − append([a],[b],X1)
⇒수열 변수 집합 R1＝{X1}
N2 − append([ ],[b],X111−t)
⇒수열 변수 집합 R2＝{X11−h, X111−t}
치환값 : X11−h＝a

잎의 원소 수열 변수를 모아 그 변수의 최대 AND 집합들을 만든다. 여기서 원소열을 갖는 수열은 원소열이 없는 수열의 원소가 되므로, 만약 AND set에 이들이 같이 존재하면 원소열이 없는 수열 변수의 치환값에서 해당되는 원소열의 값이 원소 수열 변수의 치환값과 동일해야 한다.

X＝{[1, 111−t}

위에서는 변수 X와 그의 원소 수열이 OR 집합에 있으므로, 두개의 대안 B1 : X＝{1}과 B2 : X＝{111−t}를 갖는다. 대안 B2가 성공하려면 변수 X가 항상 리스트로 치환된다. 따라서, B2에서는 X가 올바른 리스트로 치환되었는지를 검사한다. 잎 N1과 N2에서의 원소 변수들의 최대 AND 집합은

N1 : X의 최대 AND 집합＝{1}
N2 : X−t의 최대 AND 집합＝{111}

다음 과정으로 각각의 잎들을 사실과 비교하면, N1은 비교에 실패하고 N2는 성공하여 X111−t＝[b]의 치환을 갖는다. 변수 수열의 투풀링 조건이 존재하지 않으므로 N2의 변수인 X−t의 최대 AND 집합들이 참집합이된다. 질의의 변수 X는 X−h＝a와함께 리스트를 형성하므로 값은 노드 N2의 각 원소 변수들을 원소열에따라 재구성한 리스트 [a,b]가 된다.

이제 그림 6의 증명 나무 전체를 고려하여 증명하여 보자. 오른편의 부분 나무는 무한히 생성될 수 있으며 일정한 깊이로 생성했다고 가정한다. 각 잎들에 전달된 정보는 다음과 같다.

N1 append([a],[b],X1) :

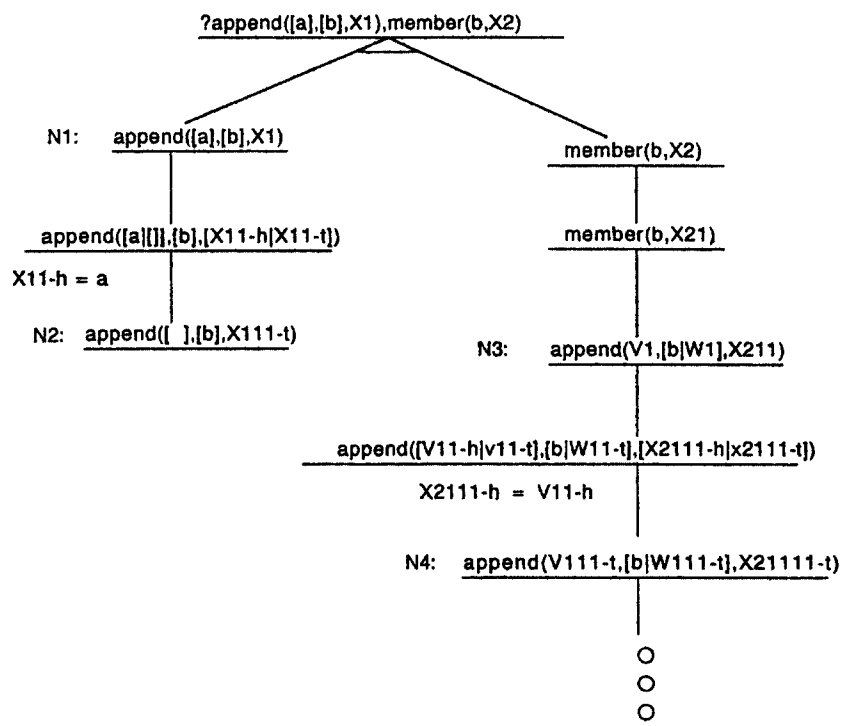


그림 6. 무한한 증명 나무

수열 변수 집합 R1={X1, X2}

N2 append([ ],[b],X111-t) :

수열 변수 집합 R2={X11-h, X111-t, X2}

동일화 과정 치환값 : X11-h=a

N3 append(V1, [b|W1],X21)

수열 변수 집합 R3={X1, X211, V1, W1}

N4 append(V111-t, [b|W111],X21111-t)

수열 변수 집합 R4={X21111-t, X1, V111-t, W111}

동일화 과정 치환값 : X2111-h=V11-h

N1에 N4까지의 잎들에 있는 원소 수열 변수들로 각각의 최대 AND 집합을 구하면, 다음과 같다.

X={[1, 111-t], [211, 21111-t]}
V={[1, 111-t]}
W={[1, 111]}

또한, 각 잎들의 투풀링 조건을 구하면 다음과 같다.

V1_W1_X211, V111-t_W111_X2111-t

이제 각 잎들에대한 변수의 최대 AND 집합을 위에서 구한 증명 나무의 최대 AND 집합에서 찾으면 다음과 같다.

N1 : X={1, [211, 21111-t]}
N2 : X={111-t, [211, 21111-t]}
N3 : X={[1, 111-t] 211}, V={1}, W={1}
N4 : X={[1, 111-t], 21111-t}, V={111-t}, W={111}

투풀링 조건을 사용하여 증명 나무의 대안들을 찾으면,

B1 : X={1, 211}, V={1}, W={1}
B2 : X={1, 21111-t}, V={111-t}, W={111}
B3 : X={111-t, 211}, V={1}, W={1}
B4 : X={111-t, 21111-t}, V={111-t}, W={111}

잎들을 사실과 비교하면, N1은 실패하고 N2, N3, N4의 치환값은 다음과 같다.

노드 N2	노드 N3	노드 N4
X111-t=[b]	X211=[b|W1]	X2111-t=[b|W111]
	V1=[ ]	V111-t=[ ]

잎 N1이 실패하였으므로, 대안 B1과 B2는 증명할 수 없으며, 대안 B3와 B4의 증명을 증명한다. 여기서, 수열 변수와 원소 수열 변수의 조인 검사는 그 원소 열을 맞춰서 실행하며, 조인에 성공하면, 나머지 원소열에대한 검사를 실행해서 변수가 올바른 리스트 구조를 갖는지를 검사해야 한다. 다음은 대안 B3와 B4를 증명하는 과정을 보여 준다. 이때, 각각의 대안에 투풀링 조건이 없으므로 이 과정은 생략되었다.

B3 : join(X111-t, X211) on X-t→T1={X-t|[b]}
(X211-t=W1이므로 W1이 X111-t의 값인 [b]로 치환됨)
리스트 구조 검사 : R2 X11-h=a이고,
X211-h=b이므로 실패함

B4 : join(X111-t, X21111-t) on X-t→T1={X-t|[b]}
(X21111-t=[b|W111-t]이므로 W111가 [ ]으로 치환)
리스트 구조 검사 : X11-h=a X211-h=V11-h ⇒
X211-h=V11-h=a X=[a,b]→성공
변수 V=[a]이고 W=[ ]이므로 모두 성공→증명 성공

따라서, 대안 B4만이 증명 나무의 증명을 성공하였고, 변수 X의 치환값 [a,b]를 구했다.

이절에서는 32절에서 제안한 방법으로 무한히 생성하는 증명 나무를 증명하였다. 잠재적으로 무한히 생성할 수 있는 나무를 일정한 깊이로 너비-우선 방식으로 생성하여, 3절에서와 같이 변수간의 아무런 의존 관계없이 병렬로 생성하며, 2단계에서 각각 노드의 성공 여부또한 병렬로 처리할 수 있다. 여기서 변수의 최대 AND 집합을 구하기 위하여 각 노드의 전달 정보를 취합하는 과정의 동기화만이 필요하다.

V. 결 론

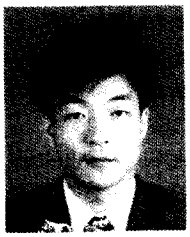
이 논문에서는 변수의 수열을 사용하여 논리 프로그램을 병렬 처리하는 방안을 제시하였다. 제안된 방안은 논리 언어의 서술적인 표현력을 최대한 활용하기 위해 증명 나무를 생성하는 부분과 사실을 비교하는 부분을 분리하여 각각의 부분을 병렬로 처리한다. 이러한 접근은 그간 논리 언어 병렬화의 장애였던 AND-parallelism의 공유 변수 의존 관계를 해소할 수 있을뿐만아니라 증명 나무를 너비-우선 방식으로 처리함으로 증명 나무가 넓게 생성 될수록 그 효과가 더욱 클 것으로 예상된다. 컴퓨터 기술의 놀라운 발전으로 대용량의 병렬 처리 컴퓨터가 상용화 되고 있으며, 이런 시스템을 효율적으로 사용하려면 수많은 프로세서들의 활용율(utilization)을 높여야 한다. 이런 관점에서 너비-우선으로 증명 나무를 처리하는 제안된 방법은 더욱 많은 병렬화를 이룰 수 있고 높은 활용율을 제공할 수 있다. 증명 나무의 처리는 논리

프로그램뿐만 아니라 연역 데이타 베이스 시스템에서 연역법(deductive law)을 처리할때[14], 프로덕션 시스템의 규칙(rule)을 사실과 비교할때 사용되므로 제안된 방법을 이들에 적용하여 질의 처리시 더욱 많은 병렬성을 얻을 수 있다. 제안된 병렬 처리 방법과 다른 방법간의 실행 시간의 실제적인 비교가 현재 없지만, 개념적으로, 늦은 변수 치환을 사용하여 변수간의 의존관계를 제거함으로서 공유되는 변수를 갖는 술어들을 동시에 처리할 수 있다는 점에서 기존의 방법들과 다른 병렬 처리 방식이다.

전통적으로 LISP와 논리 언어는 인터프리터를 사용하여 처리 하였지만, LISP의 유연성을 유지하면서 강한 타입 조사(strong type checking)를 하는 ML 언어의 개발은 논리 언어를 컴파일 언어로 변환하는 여러 가지 연구를 활성화할 것으로 예상된다. 질의 처리를 사실과 분리하는 제안된 방법은 이러한 목적을 이루는데 여러가지 잇점들을 가지고 있다. 즉, 제안된 방법은 논리 프로그램을 두개의 분리된 단계로 실행하므로 가능한 모든 질의 형태(입출력)를 고려하여 프로그램을 컴파일한다. 이때, 프로그램에 잠재하는 사이클을 찾아, 프로그램의 실행시 증명 나무의 생성 깊이를 정한다. 또한, 컴파일된 프로그램은 부분적인 증명 나무를 생성하여 질의 수행시 질의와 부분적으로 생성된 증명 나무간의 동일화(unification)를 실행할 수 있으므로 실행 속도와 빨라질 수 있다. 이렇게 컴파일된 프로그램은 사실과 무관하므로 사실의 변화에 영향을 받지 않고 프로그램 자체로 유지 보수할 수 있다.

참 고 문 헌

1. Fuchi, K., "The Direction the FGCS Project will take," New Generation Computing, Vol.3, Springer-verlag, 1983.
2. Ito, N., at el, "Dataflow Based Execution Mechanism of Parallel and Concurrent Prolog," New Generation Computing, Vol.3, pp.15-41, Springer-verlag, 1985.
3. Hasegawa, R. and Amamiya, M., "Parallel Execution of Logic Programs based on Dataflow Concept," Proc. of Intr. Con. on Fifth Generation Computer Systems, 1984.
4. Onai, R., at el, "Architecture of a Reduction-based Parallel Inference Machine: PIM-R," New Generation Computing, Springer-verlog, 1985.
5. Lindstrom, G. and Panangaden, P., "Stream-based Execution of Logic Programs," Proc. of Symp. on Logic Programming, 1984.
6. Conery, J. and Kibler, D., "AND Parallelism and Nondeterminism in Logic Programs," New Generation Computing, Springer-verlog, 1985.
7. Woo, N.S. and Choe, K.M., "Selecting the Backtracking Literals in the AND/OR tree Process Model," Sum. on Logic Programming, IEEE, 1986.
8. Kim, S.K., Maeng, S.M., and Cho, J.W., "A Parallel Execution Model of Logic Program Based on Dependency Relationship Graph," Int. Conf. on Parallel Processing, IEEE August, 1986.
9. Tung, Y-W, "Parallel Processing Model for Logic Programming," Ph&D dissertation, Dept. of Electrical Engineering and Systems, Univ. of Southern California, May, 1986.
10. Chang, J.H. and Despain, A.M., "Semi-intelligent Backtracking of Prolog based on Static Dependency Analysis," Sym. on Logic Programming, July 1985.
11. Taylor, S., Safra, S. and Shapiro, E., "A Parallel Implementation of Flat Concurrent Prolog," Int. Journal of Parallel Programming, 1987.
12. Juang, J.Y., at el, "Parallelism in Connection-Graph Based Logic Inference," Int. Conf. on Parallel Processing, 1988.
13. Hausman, B., Ciepielewski, A. and Haridi, S., "OR-parallel Prolog Made Efficient on Shared Memory Multiprocessors," Sym. on Logic Programming, IEEE, 1987.
14. Lee, W.S., "Parallel Query Evaluation in Deductive Databases," Ph&D Dissertation, Purdue University, West Lafayette, IN, 1990.



이 원 석(Won Seog Lee) 正會員

1985년 : 미국 보스톤대학교 컴퓨터 공학과 학사 취득

1987년 : 미국 퍼듀대학교 전기공학과 석사 취득

1990년 : 미국 퍼듀대학교 전기공학과 박사 취득

1990년~1992년 : 삼성전자 선임연구원

현재 : 연세대학교 이과대학 전산과학과 조교수 재직중