공학 연구논문집 제23권 제1호 pp. 39~54. 울산대학교
Journal of Engineering Research Vol. 23, No. 1, pp. 39~54, 1992.
University of Ulsan.

병렬 논리 프로그래밍 시스템에 관한 연구

고재진
전자계산학과

〈요 약〉

이 논문에서는 흐름 AND 병렬성과 확정 선택 비결정성에 의한 병렬 논리 프로그래밍 시스템의 구현을 위한 전반적인 구비 요건에 대해서 서술했다. 여러 논리절의 보호부들을 병렬 처리하는데 있어서 보호부들 사이의 포함 관계 검토에 의한 가장 좁은 범위의 조건을 갖는 보호부를 선택하는 새로운 방법을 제시했고, 릴레이션의 형 선언에 있어서 입력, 출력외에 자유인수를 지정할 수 있는 새로운 방법을 제시했다. 이 시스템의 구현은 AND/OR 프로세스 트리 모델인 가상의 전산모델에서 처리할 수 있도록 제시했다.

A Study on the Concurrent Logic Programming System

Jae Jin Koh
Department of Computer Science, University of Ulsan

〈Abstract〉

In this paper we describe the general components for constructing the parallel logic programming system by a stream AND parallelism and a committed choice non-determinism. We propose two new methods: one is the method that selects the guard who has the narrowest ranges of conditions by considering the subsumption relations among the guards when searching the guards concurrently, the other is the method that can declare the free arguments in addition to the input and output arguments when declare the relation type. We propose the AND/OR process tree model as an abstract computation model on which this system can be implemented.

"이 논문은 1990년도 문교부 지원 한국학술진흥재단의 지방대 육성 학술연구조성비에 의하여 연구되었음"

1. 서 론

1.1 서 론

논리 프로그래밍은 선언적 프로그래밍(declarative programming)의 한 분야로서, 일차 프레디키트 논리(first order predicate logic)의 논리절(clause)들을 명령문으로 다루는 프로그래밍 언어이다.

논리 프로그램(logic program)은 Horn 논리절(Horn clause)과 목표문(goal statement)으로 구성된다. Horn 논리절은 다음의 형태를 갖는다.

B <- A1,A2,....An

여기서 n >= 0이고, B와 각 Ai, 1<= i <= n,는 리터럴(literals)이다. 그리고 B를 논리절의 머리(head)라 하고, A1,....,An을 논리절의 몸체(body)라 한다. 위의 논리절의 의미를 설명하면 B가 성립하기 위해서는 각 Ai가 성립하여야 함을 뜻한다. 즉 if A1 and A2 and ... and An then B를 의미한다. 각 Ai를 B가 성립하기 위해서 만족시켜야 할 목표(goal)라 한다. 리터럴은 R(T1,T2,....,Tk) 형태로서, k > 0이고 R은 k항(k-adic) 릴레이션(relation)이다. 각 Ti, 1 <= i <= k,는 항(term)이다. 항은 상수(constant), 변수(variable), 식(expression)이 될 수 있다. 식은 $f(t_1,\ldots,t_m)$ 형태로서 m > 0이고, f는 m항(m-ary) 함수 기호(function symbol)이고, 각 ti는 항이다.

이 논문에서는 Horn 논리절을 단순히 논리절이라 부르겠다. 목표문은 다음의 형태를 갖는다.

?- G1,G2,....,Gn

여기서 각 Gi ,1 <= i <= n, 는 목표이다.

논리절 B <- A1,....,An 은 어떤 문제를 나타내는 리터럴인 목표와 이 논리절의 머리 리터럴인 B가 대응(match)될 때 문제를 푸는데 사용될 수 있다. 대응이란 두개의 리터럴이 통합(unification)에 의해서 같은 문제를 나타냄을 의미한다. 리터럴의 통합 결과로서 다음과 같은 형태의 변수 치환이 생긴다.

(X1/T1,....,Xm/Tm)

여기서 각 Xi는 변수이고 각 Ti는 항이다.

Kowalski[5]는 논리절에 절차적 해석(procedural interpretation)을 부여하는 기법에 관한 획기적 논문을 발표하였고, 이 논문을 기초로 해서 논리 프로그래밍 언어가 구현되기 시작했는데, 최초의 논리 프로그래밍 언어로서 현재도 널리 쓰이고 있는 것은 1972년에 Marseille 대학에서 Colmerauer 등에 의해서 설계되고 구현된 Prolog[7]이다. 논리 프로그래밍 언어는 인간의 사고를 명백히 표현할 목적으로 고안된 인간 지향적인 언어이다. VLSI 기술이 발전함에 따라서 새로운 병렬 처리 컴퓨터의 개발이 가능하게 되었고, 그에 따라서 선언적 언어에 대한 관심이 많아지게 되었다. 선언적 언어는 원칙적으로 병렬 처리에 적합한 언어이다. 1982년에 시작된 일본의 5세대 컴퓨터 시스템 프로젝트의 목적은 병렬 논리 프로그램의 고속 처리를 위한 병렬 컴퓨터의 개발에 있는 것은 널리 알려진 사실이다. Prolog는 직렬 처리를 전제로 하여 설계되었기때문에 병렬 처리에 적합한 논리 프로그래밍 언어는 아니다. 그래서 논리 프로그램의 병렬 처리에 관한 여러가지의 제안이 있어 왔다.

논리 프로그램의 병렬 제어 전략에는 기본적으로 다음과 같은 4가지의 방법[4]이 있다.

(1) 제한된 AND 병렬성(Restricted AND parallelism)
공통 변수를 갖지 않는 여러개의 목표들의 병렬 처리를 의미한다.

(2) 흐름 AND 병렬성(Stream AND parallelism)
공통 변수를 갖는 두개의 목표를 병렬 처리하는데 있어서, 공통 변수의 값은 두개의 목표사이에 점진적으로 서로 교통하게 된다.

(3) OR 병렬성(OR parallelism)
어떤 목표를 만족시키기 위해서 여러 논리절들을 병렬적으로 적용하는 방법이다.

(4) 전해 AND 병렬성(All-solutions AND parallelism) 여러 목표들을 병렬 처리하는데 있어서, 각 목표는 서로 다른 해를 구한다.

현재 까지의 본 논제와 관련된 문제점은 확정 선택 비결정성(committed choice non-determinism)에 의한 논리절의 선택에 있어서, 보호부(guard)를 병렬처리하는 방법의 비효율성에 있는데, 본 논문에서는 보호부들 사이의 포함 관계 검토에 의한 가장 좁은 범위의 조건을 갖는 보호부를 선택하는 새로운 방법을 제시했다. 그리고 릴레이션의 형 선언에 있어서 기존의 입력 인수, 출력 인수 지정외에 자유 인수를 지정할 수 있는 새로운 방법을 제시했다. 본 논문은 흐름 AND 병렬성을 중심으로 전개해 나간다.

1.2 관련 연구

이 절에서는 본연구와 관련된 여러 연구들에 대해서 고찰한다. 병렬 논리 프로그램밍 언어에 관한 일반적인 문헌 조사 연구(survey)는 논문[9]에 상세하게 서술되어 있다. 본 연구에 관련된 연구의 하나로서 Concurrent Prolog[8]가 있다. 이것은 Relational Language[2]을 기초로 해서 개발한 것으로서, 흐름 AND 병렬성을 구현하기 위해서 보호된 논리절(guarded clauses)과 확정 선택 비결정성(committed choice non-determinism)을 원용하고 있다. Concurrent Prolog에서는 본 연구의 입력 형에 해당하는 변수들에 read-only annotations을 부처서 사용하고 있다. 만약 read-only로 지정된 변수를 머리 통합(head unification)에서 비변수항(non-variable term)으로 항결합(bind)하려면, 그 프로세스는 read-only 변수가 다른 프로세스에 의해서 상수배정될 때 까지 유보된다. 다음의 예를 보도록 한다.

```
merge([A|X],Y,[A|Z])<-true merge(X?,Y,Z).
merge(X,[A|Y],[A|Z])<-true merge(X,Y?,Z).
merge([ ],Y,Y)<-true true.
merge(X,[ ],X)<-true true.
```

여기서, 변수에 ? 기호를 부친 X?, Y? 등이 read-only 변수들이다. 이 merge 릴레이션에 대한 질의의 목표는 merge(X?, Y?,Z)형태가 되겠다. 그리고 Concurrent Prolog는 다중 환경(multiple environment) mechanism을 적용하고 있다. 각 보호부 실행에서 국소 항결합 환경(local binding environment)을 제공하고 있고, 어떤 변수의 값은 각 환경(environment) 마다 다를 수 있다. 목표에 있는 변수들은 보호부를 실행하는 중에 국소 환경(local environment)에 복사되고, 어떤 논리절에 확정(commit)된 후에는 범용 환경(global environment)으로 복귀된다.

그 다음의 관련된 연구로는 GHC[10]가 있다. GHC에서는 목표에 있는 변수들이 보호부를 실행에서 항결합(bound)되지 않는다. 만약 목표에 있는 변수들이 보호부 실행에서 항결합할 시도가 있다면, 그 변수들

이 다른 프로세스에 의해서 항결합될 때 까지 보호부 실행은 유보된다. 이 방법은 실행시에 변수의 항결합 여부를 시험하는 것이어서 보호부를 실행하는데 많은 시간이 소요된다. GHC도 보호된 논리절(guarded clause)의 사용과 확정 선택 비결정성(committed choice non-determinism) 방법에 의한 흐름 AND 병렬성을 구현하고 있다.

그 다음의 관련된 연구로는 Parlog[3] 가 있다. Parlog에서는 릴레이션의 인수들에 대해서 미리 입력 모드와 출력 모드를 지정하고 있고, 입력 모드로 지정된 변수가 머리 통합(head unification)을 할 때 비변수항으로 상수 배정될려는 시도가 있으면, 그 변수가 다른 프로세스에 의해서 상수 배정될 때까지 머리 통합은 유보된다. Parlog도 보호된 논리절의 사용과 확정 선택 비결정성 방법에 의한 흐름 AND 병렬성을 구현하고 있다.

1.3 병렬 논리 프로그래밍 시스템의 응용

병렬 논리 프로그래밍 시스템 응용의 한 분야는 시스템 프로그래밍(system programming)을 위한 언어로서의 역할이다. 이것을 이용해서 실험적인 운영체제(operating system)를 작성할 수 있다. 병렬 통신 시스템의 운영 규약을 작성하는데도 이 언어를 이용할 수 있다. 이 언어로 작성된 운영 규약은 이 언어 시스템을 통해서 시뮬레이션할 수 있고, 그 시뮬레이션은 이산시간(discrete time)으로 제어할 수 있다. 자연 언어 처리도 이 언어 응용 분야의 하나이다. 이 언어를 사용해서 자연 언어 문장에 대한 선택적인 후보 parsing을 병렬적으로 수행하는 상향(bottom-up) parser를 작성할 수 있다. 그리고 software 설계 규약(specification)을 작성하는 데도 이 언어를 이용할 수 있다.

2. 흐름 AND 병렬성 (Stream AND parallelism)

흐름 AND 병렬성은 공통 변수를 갖는 목표들의 단일해를 구하는데 있어서, 각 목표들이 병렬적으로 처리되고, 어떤 목표의 공통 변수가 항결합(binding)되면 그 변수의 값이 각 목표들 사이에 점진적으로 전달되는 방식이다. 다음의 예를 들어 보자. 질문은 다음과 같고,

```
?-sort(X,Y),writeins(Y).
```

논리절들이 다음과 같을 때,

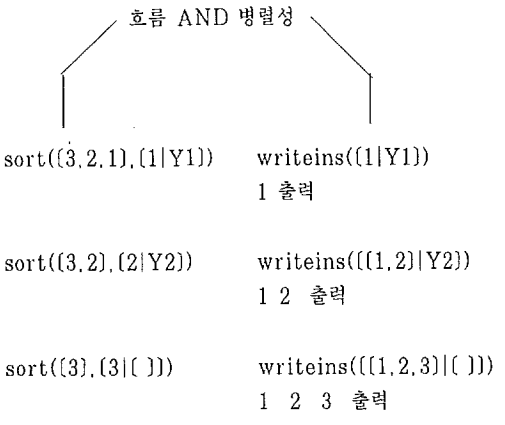
```
sort([ ],[ ]).
sort(X,[A|Y]) <- smallest(X,A,Z),
        sort(Z,Y).
```

sort(X,Y)는 값들의 리스트인 X를 정렬하여, 그 결과 정렬된 리스트 Y를 생성하는 프레디키트이다. smallest(X,A,Z)는 리스트 X 중에서 가장 작은 값을 골라서 A에 넣고 A를 제외한 X의 나머지 리스트를 Z로 하는 프레디키트이다. writeins(Y)는 리스트 Y에서 상수배정(instantiated)된 앞부분의 값들을 출력하는 프레디키트이다. sort(X,Y)와 writeins(Y)는 공통 변수 Y를 갖고 있고, 흐름 AND 병렬성 기법에서는 부분적으로 정렬되어진 Y의 값을 그대로 전달하여서, writeins(Y)에서 부분적으로 정렬 되어진 Y의 앞부분의 내용을 출력한다. 흐름 AND 병렬성에 의하면, 정렬이 완료되기 전에 부분적으로 정렬된 값들을 공통 변수를 갖는 다른 목표에 전달하여서, 부분적으로 완성된 리스트를 병렬적으로 처리함으로써 처리의 효율성을 높일 수 있다.

예를 들면, 질문이 다음과 같을 때

?- sort([3,2,1],Y),writeins(Y).

위의 질문의 처리 과정을 나타낸 것이 그림 1에 있다.



(그림 1) 흐름 AND 병렬성의 처리 예

위의 그림을 설명하면, 앞의 질문에서 sort([3,2,1],Y) 목표와 writeins(Y) 목표는 흐름 AND 병렬성에 의해서 처리되는데, 두 목표사이에는 공통 변수 Y가 있다. 앞의 sort 프로시저에 의해서 Y가 점진적으로 앞부분 부터 상수 배정된다. writeins에서는 부분적으로 상수 배정된 Y의 앞부분의 값을 출력한다.

3. 병렬 논리 프로그래밍 언어

이 장에서는 본논문에서 제시하는 병렬 논리 프로그래밍 언어인 KCURPRO(Korean Concurrent Logic Programming)의 구성에 대해서 기술한다. KCURPRO는 흐름 AND 병렬성을 적용하는 언어이다.

3.1 릴레이션의 형 선언

릴레이션(relation)이란 기본적으로 R(A1,A2,...,An)의 형태를 갖는 것으로서 R은 릴레이션 이름이고 A1,...,An은 릴레이션의 인수들이다. 릴레이션은 각 인수들 사이의 관계를 나타내는 것으로서, 논리 프로그래밍에서 프레디키트(predicate)는 릴레이션의 성립 조건을 표시하는 것이다. 논리 프로그래밍에서 릴레이션의 정의는 두 가지 방법으로 할 수 있는데, 첫째는 단순 사실 릴레이션들을 나열하는 방법이고, 둘째는 릴레이션의 성립 조건을 논리절로 표현해서 정의하는 방법이다. 모든 릴레이션(relation) 정의는 다음과 같은 형태의 형 선언을 선두에 갖는다.

```
type R(T1,T2,...,Tk).
```

여기서, 각 Ti는 세가지의 형인 입력(i), 출력(o), 자유(f) 중의 하나이다.

입력형의 인수는 어떤 목표가 어떤 논리절의 머리와 통합할 때, 그 목표의 입력 형의 인수는 논리절의 머리에 있는 비변수항(non-variable term)으로 상수배정(instantiation)할 수 없음을 의미한다.

출력형의 인수는 머리 통합할 때 비변수항으로 상수배정할 수 있음을 의미한다.

자유형의 인수는 머리 통합할 때 제한이 없음을 의미한다.

예를 들면,

```
type conc(i,f,o).
conc([ ],L,L).
conc([X|L1],L2,[X|L3])
      <- conc(L1,L2,L3).
```

위에서 conc 릴레이션의 첫번째 인수는 입력, 두번째 인수는 자유, 세번째 인수는 출력으로 형 선언이 되었다. 이런 경우 가능한 질의의 형태는 다음과 같다.

```
?- conc([1,2,3],L1,L2).
?- conc([2,3,4],[5,6],L).
```

입력형 인수들에 대한 통합을 입력대조(input matching)라 하는데, 목표에 있는 입력형 인수가 논리절의 머리에 있는 해당하는 인수의 비변수항으로 상수배정 하여 대조와 보호부 실행이 병렬적으로 이루어진야 할 경우에는 그 논리절에 대한 통합은 목표의 해당하는 입력형 인수가 다른 프로세스(process)에 의해서 상수배정될 때까지 유보(suspend)된다.

3.2 보호된 논리절 (guarded clauses)

KCURPRO에서 논리절의 문법은 다음과 같다.

R(T1,T2,....,Tk) <- G1,G2,....,Gm : B1,B2,....,Bn.

여기서, R(T1,T2,....,Tk) 은 머리 릴레이션이고,
G1,G2,....,Gm 은 목표들의 나열로서 보호부(guard)이고,
B1,B2,....,Bn 은 목표들의 나열로서 몸체(body)이다.
: 은 확정 연산자(commit operator)이다.

위의 논리절의 뜻은 G1,G2,....,Gm, B1,B2,....,Bn이 만족되면 R(T1,T2,....,Tk)가 만족된다는 것이다. 보호부가 있기 때문에 보호된 논리절이라 부른다. 보호부란 논리절이 선택되기 위해서 만족되어야 할 시험을 뜻한다. 보호부가 만족되면 확정 연산자 :가 수행되고, 그 논리절은 선택되어지며, 다른 논리절에의 대조(matching) 시도는 금지된다.

이 논문에서 제시한 병렬 논리 프로그래밍 언어인 KCURPRO에서는 확정 선택 비결정성(committed choice non-determinism) 기법[4]을 원용한다.

어떤 목표 R(T1,T2,....,Tk)의 만족은 릴레이션 R(T1,T2,....,Tk)을 정의하는 여러 논리절들을 병렬적으로 탐색해서 후보 논리절을 찾는 방식으로 진행된다. 어떤 후보 논리절이 발견되면 그 논리절이 선택되고, 다른 논리절들의 탐색은 중지된다. 이것을 확정 선택(committed choice)이라 한다. 그러면 원래의 목표는 선택된 논리절의 몸체 부분의 목표들로 대체된다. 후보 논리절의 선택은 입력대조(input matching)와 보호부 실행(guard evaluation)의 성공에 의해서 결정된다. 논리절이 선택된 후에 출력형 인수들에 대한 출력 통합(output unification)이 행해진다.

예를 들어 보면, 다음과 같은 릴레이션 정의에서

```
type searchtree(i,o,i).
searchtree(Key,Value,tree(Left,
rootpair(Key,Value),Right).
searchtree(Key,Value,tree(Left,
rootpair(Dkey,Dvalue),Right) <-
Key < Dkey : searchtree(Key, Value,
Left).
searchtree(Key,Value,tree(Left,
rootpair(Dkey,Dvalue),Right) <-
Dkey < Key : searchtree(Key,
Value,Right).
```

searchtree 릴레이션은 키값에 의해서 순서대로 배치된 이진 트리를 탐색하는 것으로서, 이진 트리의 각 노드는 키값과 그에 해당하는 어떤 값으로 구성되어 있다. 키값과 트리가 주어지면 그 키값에 해당하는 노드의 해당하는 값을 구하는 것이다.

첫번째 논리절은 주어진 키값이 근노드의 키값과 일치하는 경우로서 근노드에서 바로 해당하는 값을 구하고 있다.

두번째 논리절은 보호부에서 주어진 키값과 근노드의 키값을 비교해서 주어진 키값이 근노드의 키값보다 작으면 확정 연산자(commit

operator)가 실행되어서, 그 논리절이 선택되고 다른 논리절에의 대조(matching) 시도는 금지되고, 왼쪽 트리에 대해서 재귀적으로 트리 탐색을 하게 된다.

본 논문에서 제시한 병렬 논리 프로그래밍 언어인 KCURPRO의 실행논리를 설명하면, 처음에 각 목표의 실행은 유보(suspend)된다. 그런 다음에 각 목표의 실행을 위해서 해당 논리절들에 대해서 병렬적인 적용이 시작된다. 각 논리절의 적용에서는 입력 밍 언어인 KCURPRO의 실행논리를 설명하면, 처음에 각 목표의 실행은 유보(suspend)된다. 그런 다음에 각 목표의 실행을 위해서 해당 논리절들에 대해서 병렬적인 적용이 시작된다. 각 논리절의 적용에서는 입력 다. 각 적용 논리절은 다음과 같은 세가지 중의 하나로 분류된다.

(1) 후보 논리절(candidate clause)
입력 대조와 보호부 실행이 성공한 경우이다.
(2) 비후보 논리절(non-candidate clause)
입력 대조나 보호부 실행이 실패한 경우이다.
(3) 유보된 논리절(suspended clause)
입력 대조나 보호부 실행이 유보되고 둘다 실패하지 않은 경우이다.

유보된 논리절은 결국에는 후보 논리절이 되든지 비후보 논리절이 된다.

3.3 보호부(guard)의 안전성

어떤 목표의 만족을 위해서 후보 논리절들을 찾을 때, 보호부 실행을 행하는데, 이 보호부 실행을 하는 도중에 목표에 있는 어떤 변수라도 항결합(bind)되어서는 안된다. 보호부가 실패할지도 모르니까 목표의 변수들의 항결합은 항상 확정(commit)후에 행해져야 한다. 출력 통합(output unification)은 항상 확정 후에 행해진다. 머리 인수항 내의 변수들은 입력 대조나 보호부 실행에 의해서 상수 배정될 수도 있다. 그러나 출력 통합이 행해질 때 까지 목표의 출력 인수에 영향을 주어서는 안된다. 보호부의 안전성이란 입력 대조(input matching)에 의해서 얻어지는 변수의 값들에 대해서 단지 시험만 행하는 것이다. 즉 논리절의 머리에 있는 입력 변수들을 상수 배정해서는 안된다. 보호부의 안전성은 언어의 구현을 쉽게 해준다. 그리고 안전성 시험은 번역시에 하도록 한다. 그럼으로써 OR 병렬 탐색은 항상 AND 병렬 실행으로 전환할 수 있다.

3.4 정확성(correctness)과 완전성(completeness)

KCURPRO의 추론 구조는 resolution 원리[6]에 기초하고 있다. KCURPRO 프로그램이 생성한 해는 논리 프로그램의 선언적 의미에 의해서 정확하다. 순수한 resolution 원리에서 벗어난 것은 입력 대조라는 특수한 통합을 사용한다는 것이다. 그러나 두개의 항 T1과 T2의 입력 대조가 성공하면 두개의 항의 순수한 통합도 성공한다. 따라서 KCURPRO에서 정확성은 보장된다. KCURPRO 프로그램에 의해서 생성되는 해는 논리 프로그램의 선언적 의미에 의해서 규정된 여러 해 중에서 많아야 한개의 해를 생성한다. 이것은 KCURPRO가 확정 선택 비결정성 원리를 원용하고 있기 때문이다. 논리 프로그램의 선언적 의미에 의해서 해가 존재하더라도 KCURPRO 프로그램의 실행이 실패이거나 무한한 유보에 걸릴 수 있다. 따라서 KCURPRO는 불완전하다. 이것을 극복하는 방법을 강구하도록 하자.

다음의 예를 들어 보자.
merge([U|X],Y,[U|Z]) <- merge(X,Y,Z).

```
merge(X,[V|Y],[V|Z]) <- merge
(X,Y,Z).
merge([ ],Y,Y).
merge(X,[ ],X).
```

다음과 같은 질의가 있을때,

```
?- merge([a,b],[c,d],Z).
```

Z는 다음 리스트 중의 어느 하나가 된다.
[a,b,c,d],[a,c,b,d],[a,c,d,b], [c,a,b,d],[c,a,d,b],[c,d,a,b]
다음과 같은 질의에서

```
?- flatten(T1,X),flatten(T2,Y),
merge(X,Y,Z),writelist(Z).
```

merge(X,Y,Z)라는 질의가 실행되는 과정을 보면, 처음에 각 merge 논리절들을 병렬적으로 적용한다. X나 Y가 다른 목표에 의해서 비변수항으로 항결합될 때까지 입력 제한 조건 때문에 모든 논리절 대조(clause match)가 유보된다. 만약 리스트 X의 첫번째 요소가 다른 목표에 의해서 항결합되어서 [a|X1]로 되었다면 첫번째 merge 논리절의 입력 조건이 만족되어서 첫번째 merge 논리절이 후보 논리절이 되고, Z는 [a|Z1]으로 항결합된다. 만약 Y가 [c|Y1]으로 항결합 된다면 두번째 merge 논리절이 후보가 되고 Z는 [c|Z1]으로 항결합 된다. 만약 X와 Y가 둘다 동시에 항결합된다면 첫번째와 두번째 merge 논리절이 동시에 후보가 되고 둘중에 하나가 선택되는데,이 경우의 선택 기준은 작동의미에 의해서 규정되지는 않는다. 단지 둘중에 하나가 선택될 것이다. 이와 같은 작동 의미의 효과는 어떤 목표의 실행도 그것의 출력 인수에 대해서 한개 이상의 해를 생성하지는 않는다는 것이다. 더구나 변수에 항결합 된 값들은 전혀 철회되지 않는다는 것이다. 그래서 공통변수는 단일 배정성(single-assignment property)을 갖는다. 확정 선택 비결정성은 원활한 병렬 처리를 위해서 비결정성(non-determinism)과 단일 배정성을 결합한 결과로서 나온 것이다.

여기서 문제가 되는 것은 불완전성이고, 이를 해결하는 방법을 고찰해보자. 병렬 논리 프로그래밍 언어에서 논리절 선택의 결정 조건은 입력 대조와 안전한 보호부 실행인데, 그런 경우 논리절이 선택되면 다른 논리절에의 적용 시도는 중지되고, backtracking도 허용되지 않는다. 만약 선택된 논리절이 우리가 원하는 해를 생성하지 않으면 어떻게 할 것인가? 이에 대한 해답은 아직까지 제시되고 있지 않다.

예를 들어 보면,

```
class(X,fighter) <- beat(X,_ ),
beat( _,X):.
class(X,winner) <- beat(X,_ ):.
class(X,sportsman) <- beat( _,X):.

beat(tom,jim).
beat(ann,tom).
beat(pat,jim).
```

에서 다음과 같은 질의를 한다면,

```
?- class(tom,C).
```

C는 논리절의 병렬탐색에 의해서 fighter, winner, sportsman 중 어느 하나가 된다.

이것은 우리가 원하는 것이 아니다. 여기서 문제가 되는 것은 각 논리절의 보호부 사이의 포함(subsumption) 관계이다. 즉 만족되는 보호부 사이에 포함 관계가 있으면 가장 좁은 범위의 조건을 만족하는 보호부로 확정하도록 하는 것이다. 그러면 위의 질의에서 변수 C는 fighter 가 되어서 우리가 원하는 대로 결과가 나온다.

그리고 다음과 같은 질의를 한다면,

?- class(tom,sportsman).

응답이 yes 인가 no 인가를 생각해보자. Prolog 방식으로 해석하면 yes가 되지만, 우리가 의도하는 바는 아니다. 여기서 문제가 되는 것은 출력변수에 값이 미리 항결합되어서 질의에 나타날 수 있는가의 문제와 보호부의 성공과 출력변수의 불일치는 어떻게 처리할 것인가의 문제이다. 출력변수에 값을 미리 항결합하는 것은 시험의 문제이기 때문에 보호부의 성공과 출력변수의 불일치는 시험의 실패로 간주하는 것이 타당하다고 생각한다. 이와같이 성공한 보호부 사이의 포함 관계를 조사해서 최소 범위를 갖는 보호부로 선택하는 것도 불완전성을 완화하는 하나의 방법이 되겠다.

3.5 보호부의 충분성

목표를 만족시키기 위한 실행에 있어서, 어떤 논리절이 후보가 된다면, 첫째 그 논리절을 사용해서 목표의 해가 생성될 수 있든지, 둘째 어떤 다른 논리절을 사용해서는 해를 생성할 수 없도록 보장하기 위해서 입력 대조와 각 논리절의 보호부가 충분해야 한다. 이것을 보호부의 충분성이라 한다. 달리 말하면 목표의 해가 존재한다면 그 목표는 실패하지않아야 한다. 보호부의 충분성을 만족하기 위해서 각 논리절의 몸체 부분을 전부 보호부에 포함시키면 가능하지만 이것은 모든 보호부의 실행을 요구하기 때문에 계산량이 너무 증가한다. 따라서 보호부의 충분성을 만족시키는 최소한의 보호부를 선택하는 것이 이상적이지만 간단한 문제는 아니다.

3.6 논리적 변수(logical variable)

논리적 변수란 부분적으로 상수배정(instantiated)된 데이타 구조로서, 항결합되어 있지 않는(unbound) 변수들을 포함하고 있다. 목표의 실행에 의해서 이러한 논리적 변수들이 생성될 수 있다. 논리적 변수안에 포함된 변수들은 다른 목표의 실행에 의해서 나중에 상수배정될 수 있다. 논리적 변수안에 있는 변수들이 차례대로 상수배정됨으로써 논리적 변수의 내용이 점진적으로 수정되어 진다. 논리적 변수는 양방향 통합(bidirectional unification)의 결과로 생긴다. 어떤 목표의 약성 입력 인수란 그 목표의 실행에 의해서 그 입력 인수안에 있는 변수들을 상수배정할 수 있는 것이다. 입력 인수에 약성을 부여함으로써 목표의 실행에 의해서 입력 인수 항에 있는 변수들을 상수배정시킬 수 있다. 즉 입력 인수 항에 논리적 변수를 사용할 수 있다.

예를 들면,

```
type r(i).
r(f(X,Y)) <- s(X).
```

에서 목표 r의 실행은 입력 인수가 f(X,Y) 형태로 상수배정될 때까지 유보된다. 그러나 변수 X는 s(X)에 의해서 상수배정될 수 있다. 이것은 s의 모드가 s(o)이던지 또는 s(i)이지만 입력 인수가 약성일 때 가능하다.

3.7 후방 통신 (back communication)

어떤 목표의 실행은 다른 목표에 의해서 상수배정될 수 있는 변수를 포함하는 논리적 변수를 생성할 수 있다.그러나 논리적 변수를 생성하는 목표와 그 논리적 변수를 받아서 그안에 있는 변수를 상수 배정하는 목표는 병렬적으로 처리된다. 논리적 변수를 생성하는 프로세스(process)를 생산자 프로세스라하고 논리적 변수를 받아서 그안에 있는 변수를 상수 배정하는 프로세스를 소비자 프로세스라 할 때, 소비자 프로세스에서 논리적 변수안에 있는 변수를 상수배

정하면 소비자로 부터 생산자에로 후방 통신이 일어 난다. 이것은 병렬적으로 처리되는 두개의 목표사이에 한개의 공통 변수를 통해서 양방향 통신(two-way communication)이 실현되는 것이다.

예를 들면,

```
writelist([reply(Prompt, Reply)|List]) <- write(Prompt),read(Reply), writelist(List).
writelist([Message|List]) <- write(Message),nl,writelist(List).
writelist([ ]).
```

에서 writelist 논리절이 reply(Prompt, Reply)라는 메세지(message)를 받게되면 Prompt를 출력하고, read에 의해서 한 항(item)이 입력되어서 Reply에 상수배정되고 이것은 후방 통신에 의해서 원래의 목표에 보내지게 된다. 다른 형태의 메세지들은 두번째 writelist 논리절에 의해서 정상적으로 출력하게 된다.

3.8 직렬 처리

지금까지는 AND 병렬 처리와 확정(committed) OR 병렬 처리를 적용했다. 이론적으로는 이것으로 충분하나, 실용적인 이유 때문에 목표의 직렬 처리와 논리절 탐색의 직렬 처리가 선택사항으로 제공된다. 다음에 병렬과 직렬 처리에 관한 연산자를 기술하겠다.

, : 목표의 병렬 AND 처리 연산자
. : 논리절의 병렬 탐색 연산자
& : 목표의 직렬 AND 처리 연산자
.. : 논리절의 직렬 탐색 연산자

예를 들면,

```
type dbase(i),dbase1(i,i),trans(i,i,o),initialize(o).
dbase(Cmds) <- initialize(Initdb) & dbase1(Cmds,Initdb)..
dbase1([Cmd|Cmds],Db) <- trans(Cmd,Db,Newdb),dbase1(Cmds,Newdb).
dbase1([ ],Db).
```

여기서 두개의 dbase1 논리절은 서로 병렬적으로 탐색되고, dbase 논리절과 두개의 dbase1 논리절은 직렬적으로 탐색된다. dbase 논리절의 몸체에 있는 두개의 목표는 직렬적으로 처리되고, dbase1 논리절의 몸체에 있는 두개의 목표는 병렬적으로 처리된다.

3.9 부정(negation)

목표 p의 부정은 not p로 표현한다. not p는 p의 증명이 실패할 때 true가 된다. 이와 같은 규칙을 실패에 의한 부정(negation as failure) 규칙이라 한다[1]. 일반적으로 p에 대한 증명의 실패가 논리적으로 p가 false라는 것의 증명이 될 수는 없지만, 해당 프로그램에 국한한 세계에서는 false임에 틀림없다. 논리적 false와 증명 실패에 의한 false는 한정된 세계 가정(closed world assumption)이라는 가정하에서만 일치한다. 프로그램이 릴레이션들에 대한 완전한 정의를 갖고 있다고 가정하는 것이 한정된 세계 가정이다.

예를 들면,

```
sibling(X,Y) <- parent(Z,X),parent(Z,Y),diff(X,Y).
diff(X,Y) <- dif1(X,Y).
diff(X,Y) <- dif1(Y,X).
parent(alice,caspar).
parent(caspar,eric).
parent(alice,brian).
parent(alice,dexter).
dif1(brian,caspar).
dif1(brian,dexter).
dif1(caspar,dexter).
```

에서 다음의 implication을 추가하면,

$(\forall X,Y)[parent(X,Y) \rightarrow$
$[X=alice \wedge Y=caspar] \vee$
$[X=caspar \wedge Y=eric] \vee$
$[X=alice \wedge Y=brian] \vee$
$[X=alice \wedge Y=dexter]]$

parent 릴레이션에 대한 해는 오직 여기에 정의된 릴레이션 정의에 의해서 구해진다. 이것을 완성 법칙(completion law)이라 한다. 원래의 parent 릴레이션 정의에 위의 완전 법칙을 결합하면 다음과 같은 'if and only if' 정의를 얻을 수 있다.

$(\forall X,Y)[parent(X,Y) \leftrightarrow$
$[X=alice \wedge Y=caspar] \vee$
$[X=caspar \wedge Y=eric] \vee$
$[X=alice \wedge Y=brian] \vee$
$[X=alice \wedge Y=dexter]]$

위의 정의에 의한 parent 릴레이션의 증명 실패는 논리적 false로 해석할 수 있다. 다음과 같은 KCURPRO의 논리절이 있을 때,

$r(X1,\ldots,Xk) \langle\text{-}\ g_1 : b_1.$
$r(X1,\ldots,Xk) \langle\text{-}\ g_2 : b_2.$
...
$r(X1,\ldots,Xk) \langle\text{-}\ g_n : b_n.$

여기서 gi는 보호부이고 bi는 몸체이다. 각 논리절의 보호부와 몸체에 X1,...,Xk와 다른 변수 Yi,1,...,Yi,mi가 있을 때 위의 논리절들에 대한 완성 법칙은 다음과 같다.

$(\forall X1,\ldots,Xk)[r(X1,\ldots,Xk) \leftrightarrow$
$(\exists Y1,1,\ldots,Y1,m1)[g1 \quad b1]$
$(\exists Y2,1,\ldots,Y2,m2)[g2 \quad b2]$
...
$(\exists Yn,1,\ldots,Yn,mn)[gn \quad bn]]$

그리고 확정 선택(committed choice) 방법 때문에 다음과 같은 추가적인 보호부 충분 법칙(sufficient guard law)이 필요하다.

For all i and j such that $1 \leq i,j \leq n$:
$(\forall Yi,1,\ldots,Yi,mi,Yj,1,\ldots,Yj,mj)$
$[gi \wedge gj \rightarrow [bi \leftrightarrow bj]]$

이것은 보호부 충분성을 좀더 상세하게 설명한 것으로서 보호부 gi가 성공하고 몸체 bi가 실패한다면 다른 어떤 논리절의 보호부와 몸체가 성공할 수 없다는 것을 말한다. KCURPRO에서 실패에 의한 부정은 논리절의 직렬 탐색으로 구현할 수 있다.

```
type not i.
not p <- call(p) : fail..
not p.
```

여기서, not p는 첫번째 논리절의 보호부에 있는 목표 p를 실행해서 그것이 성공하면 not p는 확정(commit)되서 fail이 된다. 만약 goal p가 실패하면 not p는 성공한다. not p의 실행에서는 어떤 변수도 항결합(bind)되지 않는다.

4. 병렬 논리 프로그래밍 시스템의 구현

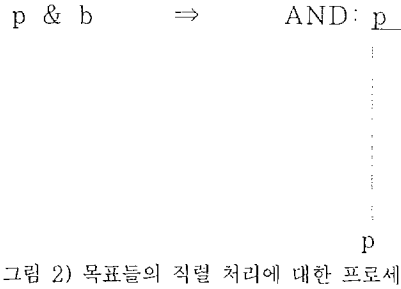
이 절에서는 AND/OR 프로세스 트리 모델(process tree model)에 의한 KCURPRO의 구현에 관해서 논한다. AND/OR 프로세스 트리 모델은 KCURPRO 프로그램의 제어 구조를 직접적으로 지원할 수 있는 가상의 전산 모델(abstract computation model)이다.

4.1 AND/OR 프로세스 트리 모델

AND/OR 프로세스 트리는 어떤 시점에서 KCURPRO 프로그램의 실행 상태를 기록하는 프로세스 트리이다. 트리의 각 노드(node)는 프로세스로서, 그 중 잎노드 프로세스(leaf node process)는 실행가능(runnable) 또는 어떤 변수의 항결합을 기다리며 유보(suspended)상태에 있다. 잎이 아닌(non-leaf) 노드 프로세스는 어떤 프로세스 형(AND 또는 OR)을 갖고 있고, 자식 프로세스들의 실행 결과를 기다리고 있는 상태에 있다. 각 프로세스는 내부적으로 일련의 명령(instruction)들을 실행할 수 있는 능력을 갖고 있다. 이것은 모델의 논리 실행 단위(granularity)를 규정한다. 직렬로 연결된 목표들이 비명령(non-instruction) 목표로 시작할 때, 그 목표를 실행하기 위하여 그 프로세스는 자체분기(fork)되어서 그 목표를 실행하기 위한 자식 프로세스를 생성한다. 만약 어떤 릴레이션 r이 여러 논리절들로 정의되어 있다면, 그 릴레이션의 목표를 만족시키기 위한 실행은 먼저 각 논리절의 보호부를 실행하기 위한 자식 프로세스를 생성하게 된다. 목표들의 병렬 처리와 논리절들의 병렬 탐색을 위한 프로세스는 병렬로 처리될 여러개의 자식 프로세스들을 생성한다.

4.1.1 목표들의 직렬 처리

어떤 프로세스의 상태(state)가 p & b 이고 p가 목표일 때, 프로세스 트리는 다음과 같이 변한다.

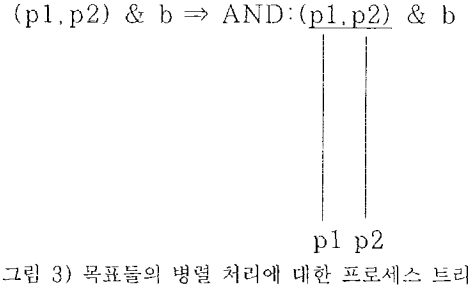


(그림 2) 목표들의 직렬 처리에 대한 프로세스 트리

프로세스 p & b는 AND 프로세스로 변하고, b에 계속 지점(continuation)이 표시된다. 그리고 p를 실행하기 위한 자식 프로세스가 생성된다. 계속지점 앞의 AND 프로세스의 내용에 밑줄이 그어있다. p가 자식 프로세스에 의해서 성공적으로 수행된 후에야 b가 실행될 수 있다.

4.1.2 목표들의 병렬 처리

목표들의 병렬 처리는 자식 프로세스가 여러개 생성된다는 점이외에는 직렬 처리와 동일하다. 예를 들면 (p1,p2) & b 프로세스를 처리하는 경우 프로세스 트리는 다음과 같이 변한다.



(그림 3) 목표들의 병렬 처리에 대한 프로세스 트리

여기서 두개의 자식 프로세스 p1 과 p2가 둘 다 성공한 후에야 b는 실행될 수 있다.

4.1.3 논리절들의 직렬 탐색

다음과 같은 직렬로 연결된 논리절들이 있을때

```
h :- g1 : b1..
h :- g2 : b2..
h :- b.
```

직렬 탐색은 첫번째 논리절의 보호부 g1을 실행하는 자식 프로세스를 생성함으로써 시작된다. 프로세스 상태는 OR가 되고 계속지점은 g1이 실패했을 때 실행되는 g2에 표시된다. 프로세스는 다음 그림과 같이 변

화된다.

g1:b1..g2:b2..b ⇒ OR: g1:b1..g2:b2..b

g1:b1

(그림 4) 논리절들의 직렬 탐색에 대한 프로세스 트리

4.1.4 논리절들의 병렬 탐색

다음과 같은 병렬로 연결된 논리절들이 있을 때,

```
h <- g1 : b1.
h <- g2 : b2..
h <- b.
```

병렬로 연결된 논리절 그룹안에서는 각 논리절의 보호부를 병렬로 처리하기 위한 자식 프로세스가 각각 생성된다. 계속지점은 병렬 그룹의 논리절들의 보호부가 모두 실패했을 때 실행되는 b에 표시된다. 프로세스 트리는 다음과 같이 변화된다.

(g1:b1.g2:b2)..b ⇒ OR:(g1:b1.g2:b2)..b

g1:b1 g2:b2

(그림 5) 논리절들의 병렬 탐색에 대한 프로세스 트리

4.1.5 AND 관계의 성공 처리

목표들의 AND 관계를 처리하기 위해서 생성된 프로세스는 그 AND 관계에 있는 모든 목표들이 성공적으로 처리된 후에야 성공적으로 종료된다. 부모 프로세스는 AND 프로세스 형을 갖는다. 성공한 자식 프로세스가 이웃을 갖지 않을 때는 그 프로세스는 제거되고 부모 프로세스는 저장된 계속지점에서 계속된다. 다음 그림에서 직렬 처리의 경우를 나타냈다.

AND: p & b ⇒ p & b

p

(그림 6) 직렬 AND 관계의 성공 처리에 대한 프로세스 트리

성공한 자식 프로세스가 이웃을 가질 때는 그 자식 프로세스는 제거되고 그 이웃 프로세스들의 성공 여부를 기다린다. 다음 그림에서 병렬 처리의 경우를 나타냈다.

AND:(p1,p2) & b ⇒ AND:(p1,p2) & b

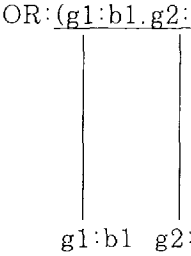
p1 p2 p2

(그림 7) 병렬 AND 관계의 성공 처리에 대한 프로세스 트리

4.1.6 OR 관계의 확정 (commitment)

논리절의 보호부를 처리하기 위해서 생성된 프로세스는 OR 프로세스 형의 부모 노드를 갖는다. 만약 보호부가 성공하고 다른 논리절이 확정되어 있지 않다면, 그 논리절에 확정(commit)된다. 확정된 프로세스는 부모 프로세스를 대체해서 성공한 논리절의 몸체를 실행하게 된다. 확정된 프로세스에 이웃이 있다면 그것들은 전부 제거된다. 다

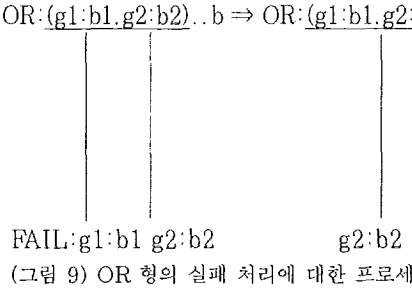
음의 그림에서 그것을 표현하고 있다.



(그림 8) OR 관계의 확정에 대한 프로세스 트리

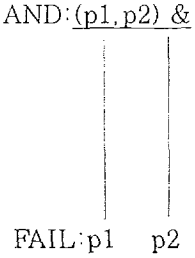
4.1.7 실패 처리

부모 프로세스 형이 OR일 때 자식 프로세스인 한 논리절의 보호부가 실패하고 이웃 보호부 프로세스가 있다면 실패한 보호부 프로세스는 제거된다. 다음 그림에서 그것을 나타낸다.



(그림 9) OR 형의 실패 처리에 대한 프로세스 트리

부모 프로세스 형이 AND일 때 어떤 자식 프로세스가 실패하면 부모 프로세스는 실패한다. 이런 경우 모든 이웃 프로세스는 제거되고 부모 프로세스는 실패로 된다. 다음 그림이 그것을 나타낸다.



(그림 10) AND 형의 실패 처리에 대한 프로세스 트리

4.1.8 유보(suspension)

각 프로세스는 일련의 명령들을 수행한다. 이런 명령들 중에 프로세스의 유보(suspension)와 재실행(reactivation)에 관련된 명령으로써 data와 bind 명령이 있다. data(V) 명령은 변수 V가 상수배정되었는지를 시험한다. 만약 V가 상수배정되어 있지 않다면 data(V) 명령을 수행하는 프로세스는 변수 V가 다른 프로세스에 의해서 상수배정될 때까지 유보된다. 변수 V가 상수배정될수 있는 방법은 bind(V,t)라는 변수 V를 항 t로 상수배정하는 명령을 수행하는 다른 프로세스에 의해서이다. 어떤 변수를 소비하는 소비자 프로세스는 여러 개있을 수 있기 때문에 하나의 bind 명령은 여러개의 프로세스를 재실행시킬 수 있다. 프로세스 유보를 구현하는 데는 적어도 두가지 방법이 있다. 하나는 실행대기(busy-waiting) 방법인데 변수 V에 대해서 유보되어 있는 프로세스는 계속적으로 변수 V가 상수배정되었는지를 시험한다. 만약 변수 V가 상수배정된 것을 발견하면 프로세스는 진행되어서 다음의 명령을 수행하게 된다. 다른 하나는 비실행대기(non-busy-waiting)방법인데 만약 어떤 프로세스가 data(V) 명령을 수행하게 되어서 유보된다면 그 프로세스는 비실행화되고 변수 V와 관련된 리스트에 들어간다. 만약 V가 상수배정된다면 그 프로세스는 실행화된다. 변수 V에 관한 유보 리스트(suspens-ion list)는 변수 V에 의해서 유보된 각 프로세스의 식별자(identifier)를 갖고 있다. 한 개의 리스트에는 여러 개의 프로세스가 있을 수 있다. 그러나 각 프로세스는 오직 한 개의 리스트에만 들어갈 수 있다. 왜냐하면 각 프로세스는 일련의 명령들을 수행하다가 어떤 시점에서 한개의 변수 때문에 유보되

었기 때문이다. 예를 들면 다음과 같은 병렬적으로 처리되는 두개의 data명령이 있을 때,

data(U),data(V)

각 data명령은 서로 다른 프로세스에서 기동되었다고 하자. 한 프로세스는 변수 U에 의해서 유보될 수가 있고, 다른 프로세스는 변수 V에 의해서 유보될 수가 있다. 만약 bind(V,t)라는 명령이 수행되면 변수 V에 의해서 유보된 리스트에 있는 모든 프로세스가 재실행하게 된다.

5. 결 론

병렬 논리 프로그래밍 시스템의 구현을 위한 전반적인 구비 요건에 대해서 서술했다. 특히 흐름 AND 병렬성과 확정 선택 비결정성(committed choice non-determinism)에 대해서 상세하게 기술했다. 보호부의 병렬 처리에 관한 여러 기법중에서 두가지의 새로운 기법을 제시했다. 시스템의 구현을 위한 가상의 전산모델로서 AND/OR 프로세스 트리 모델을 제시했다.

앞으로의 연구과제로서는 병렬 논리 프로그래밍 언어의 정규 의미론에 관한 철저한 연구가 필요하고, 병렬 논리 프로그램밍 시스템의 구현을 위한 새롭고 효율적인 기법의 개발이 필요하다. 특히 실행 종료(termination)와 교착상태(deadlock)의 해결 기법과 보호부(guard)의 안전성 문제를 해결하기 위한 효율적인 알고리즘의 개발이 필요하다. 그리고 가상의 전산모델을 실제의 전산기 구조로 변환하기 위해서는 VLSI를 이용한 전산기 설계 기법에 대한 연구가 필요하다. 즉 병렬 논리 프로그래밍 시스템의 구현을 위한 특수 전산기 시스템의 개발에 대한 연구가 수행되어야 한다.

6. 참고문헌

(1) K. L. Clark, "Negation as Failure", In Logic and Databases edited by H. Gallaire and J. Minker, pp. 293-322, Plenum Press, New York, 1978.

(2) K. L. Clark and S. Gregory, "A Relational Language for Parallel Programming", In Proc. of ACM Conf. on Functional Programming Languages and Computer Architecture, pp. 171-178, New York, 1981.

(3) K. L. Clark and S. Gregory, PARLOG : Parallel Programming in Logic, Research Report DOC 84/4, Dept. of Computing, Imperial College of Science and Technology, London, 1984.

(4) S. Gregory, Parallel Logic Programming in PARLOG : The Language and its Implementation, Addison-Wesley, Wokingham England, 1987.

(5) R. A. Kowalski, "Predicate Logic as Programming", In Proc. of IFIP Congress 74, pp. 569-574, Elsevier/North-Holland, Amsterdam, 1974.

(6) J. A. Robinson, "A Machine-oriented Logic Based on the Resolution Principle", J. ACM, Vol.12, No.1, pp. 23-41, Jan. 1965.

(7) P. Roussel, PROLOG : manual de reference dt d'utillisation, Groupe d'Intelligence Artificielle, Universite d'Aix-Marseille, Luminy, Sep. 1975.

(8) E. Y. Shapiro, A Subset of Concurrent Prolog and its Interpreter, Technical Report TR-003, ICOT, Tokyo, Feb. 1983.

(9) A. Takeuchi and K. Furukawa, "Parallel Logic Programming Language", In Proc. of the 3rd Inter. Logic Programming Conf., pp. 242-254, London, July 1986.

(10) K. Ueda, Guarded Horn Clauses : A Parallel Logic Programming Language with the Concept of a Guard, Technical Report TR-208, ICOT, Tokyo, Oct. 1986.