논리 프로그램의 병렬 수행 모형에서 완결성과 효율성의 분석

(The Analysis of Completeness and Soundness in Parallel Execution Models of Logic Programs)

김 도 형 † 이 수 현 †† 최 광 무 †††

(Do-Hyung Kim) (Su-Hyun Lee) (Kwang-Moo Choe)

요 약 논리 프로그램(logic program)의 효율적인 병렬 수행을 위한 후방 수행(backward execution) 알고리즘들이 많이 제안되었다. 그 중 Conery와 Ng-Leung에 의해 제안된 방법들은 마크 집합(mark set)이라고 부르는 자료 구조를 사용하여 AND-병렬 수행 시 발생한 실패 경력(failure history)을 저장하고, 그것에 근거하여 백트랙 리터럴 선택과 재초기화(resetting)를 수행한다. 두 알고리즘은 근본적으로 동일한 생각에 바탕을 두고 설계되었고 그 직관적 설득력 때문에 옳은 것으로 간주되어 왔다. 이 논문에서는 이들 알고리즘을 세밀히 분석하여 그것들이 수행하는 백트랙 리터럴 선택과 재초기화가 모두 틀림을 보인다. 그러한 오류가 어디에서 연유하였는지를 논의하고 또한 마크 집합 같은 자료 구조의 장점과 그 잠재적 가능성을 지적한 뒤, 마크 집합과 유사한 자료 구조를 사용하는 새로운 후방 수행 알고리즘을 제안하고 그 옳음을 증명한다.

Abstract Conery and Ng-Leung have recently proposed two backward execution algorithms for efficient AND-parallel execution of logic programs, respectively. They adopt the same kind of data structure called mark (or marks) set to store information about failure history during AND-parallel evaluation of clauses. This mark-set data structure is used to select the backtrack literal for failure and to determine the literals to be reset. Both of the two algorithms essentially have the same rationale with minor differences and have been considered correct, largely due to their intuitive persuasiveness. We, however, closely re-examine those algorithms and find out that they are incorrect for both backtrack literal selection and resetting. Their incorrectness comes from oversimplifications about general failure situations. In this paper, our analyses focused on both the mark-set-based backtrack literal selection and resetting are presented. We also propose a backtrack literal selection and resetting method based on mark-set-like data structures after discussing merits and potential of the data structures, and prove the correctness of the method.

1. 서 론

논리 프로그래밍(logic programming)[20, 26]이 새로운 프로그래밍 규범(paradigm)으로 제안된 이래, 그 프로그램의 병렬 수행을 위한 잠재적 가능성은 많은 연구자들로부터 주목받아 온 터이다. 논리 프로그램에 내재하는 여러 종류의 병렬성 중에서 가장 뚜렷하고 중요한 것은 AND 병렬성과 OR 병렬성이다[6, 9, 13, 14]. 주지하다시피, OR 병렬성은 하나의 리터럴(literal)을 해결하기 위하여 그 리터럴과 단일화(unification)가 가능한 헤드(head)를 가지는 여러 개의 클로즈(clause)를 동시에 수행해 보는 것이고, AND 병렬성은 하나의 클로즈를 해결하기 위하여 그 클로즈 내에 있는 여러 개의 리터럴들을 동시에 수행하는 것을 일컫는다. 이 논문에서는 AND 병렬성과 관련된 문제를 다루려고 한다.

동일한 클로즈 내의 리터럴들은 공유하는 변수가 있을

· 이 논문은 1994년도 한국학술진흥재단의 공모과제 연구비에 의하여 연구되었음

† 종신회원 성신여자대학교 전산학과 교수
†† 종신회원 · 창원대학교 전자계산학과 교수
††† 종신회원 · 한국과학기술원 전산학과 교수

논문접수 : 1996년 3월 2일
심사완료 : 1996년 10월 5일

수 있기 때문에, 만약 모든 리터럴들을 동시에 수행시킨다면 상이한 값이 공유 변수에 바인딩(binding)되어 리터럴들 간에 서로 상충(conflict)이 발생할 수 있다. 이러한 상황을 방지하기 위해 여러 수행 모형(model)이 제안되었는데, 가장 널리 알려진 것은 Conery의 *AND/OR Process Model*이다[6, 9]. 이 모형에서는 두 종류의 프로세스(process)가 존재한다. AND 프로세스는 클로즈의 해(solution)를 구하는 역할을 맡고 있고, OR 프로세스는 리터럴의 해를 구하는 역할을 한다. 따라서 AND 프로세스는 담당하는 클로즈 내의 각 리터럴을 처리하는 OR 프로세스를 생성하고 그 수행을 조정하여 앞서 언급한 것과 같은 바인딩의 상충이 발생하지 않도록 해야 한다. 그러한 목적을 달성하기 위하여 AND 프로세스는 그 클로즈 내의 각 변수에 대하여 *생산자*(*producer* 혹은 *generator*)의 역할을 하는 리터럴을 하나씩 지정한다. 그 변수를 포함하는 나머지 리터럴들은 자연스럽게 그 변수의 *소비자*(*consumer*)가 된다. 이후 모든 소비자 리터럴들은 생산자 리터럴이 수행을 완료하여 관련 변수를 바인딩시킨 뒤에야 그 바인딩된 값들을 가지고 자신의 수행을 시작할 수 있다. 이렇게 리터럴들의 수행 순서를 제어하기 위한 생산자-소비자 관계(relation)를 나타내는 자료 구조로서 순환이 없는(acyclic) 방향성(directed) 그래프를 AND 프로세스가 유지하는데, 이것을 *자료 종속 그래프*(*data dependency graph*; DDG)라고 부른다.

AND 프로세스의 수행은 크게 보아 *전방 수행*(*forward execution*)과 *후방 수행*(*backward execution*)이라고 불리는 두 단계로 이루어진다. 전방 수행에서는 리터럴들 간의 생산자-소비자 관계를 결정한 뒤 수행 준비가 완료된 리터럴들을 위한 OR 프로세스[1]를 생성하고 수행을 시작하도록 한다. 그런데 리터럴의 수행 도중 *실패*(*failure*), 즉 리터럴이 현재의 고정된 바인딩으로는 단일화가 가능한 클로즈를 발견할 수 없는 경우가 발생할 수 있다. 바로 이 시점부터 후방 수행이 시작된다. 그 수행의 목적은 실패 상태를 해결하는 것이다. 방법은 실패한 리터럴의 상태에 영향을 줄 수 있는 *백트랙 리터럴*(*backtrack literal*)을 하나 정해서 그것을 *재수행*(*redo*)시키는 것이다. 당연히 어떠한 리터럴들을 백트랙 리터럴의 후보로 할 것인가 하는 것이 먼저 문제로 제기되는데, 이것을 *백트랙 리터럴 선택 문제*(*backtrack literal selection problem*)라고 한다. 여기에 더해서 백트랙 리터럴 후보들 중에서 어느 것을 실제 백트랙 리터럴로 선택할 것인가 하는 점도 역시 그 문제에 포함된다. 만약 그 선택을 무작위하게 하는 경우 존재하는 해를 놓칠 가능성이 있다.[2] 그래서 일반적으로 클로즈 내의 리터럴들에 대해서 선형 순서(linear ordering)를 부여하여 백트랙 리터럴 선택 시 후보들 중에서 그 선형 순서로 보아 가장 뒤 혹은 오른쪽에 있는 리터럴을 고르도록 하는 것이 일반적인 방법이다. 이것을 해를 생성하기 위한 *내포 루프 모형*(*nested loop model*)이라고 부른다[6, 9]. 백트랙 리터럴이 재수행되어 그 변수들에 새로운 값을 바인딩시킨 뒤, 앞서 언급한 선형 순서에서 백트랙 리터럴 뒤에 있는 전부 혹은 일부 생성자 리터럴들의 바인딩을 재출발(re-starting) 혹은 *재초기화*(*resetting*)시키는 것이 필요하다. 이 또한 내포 루프 모형 하에서 존재하는 해를 놓치지 않기 위함이다. 이렇게 재초기화시킬 리터럴들을 결정하는 문제를 *재초기화 문제*(*reset problem*)라고 한다. (이후, 재수행이나 재초기화되어 해가 바뀐 리터럴들로부터 바인딩된 값을 넘겨받았던 소비자 리터럴, 즉 DDG에서 보아 후손 리터럴들은 *취소*(*cancel*)되어야 한다.) 지금까지 언급한 후방 수행에서의 두 가지 문제, 즉 백트랙 리터럴 선택 문제와 재초기화 문제는 많은 연구자들의 관심과 주의를 끌어 상당한 연구가 수행되었다[1-5, 7, 8, 10, 12, 15-19, 21, 22, 25, 28-33].

이 논문에서 다루고 있는 문제를 보다 정확하게 인식하기 위해서는 선행 연구들을 살펴봄이 적당할 듯 싶다. 원래 Conery가 제안한 백트랙 리터럴 선택 알고리즘[6, 9]은 오류를 포함하고 있어서 옳은 해를 놓칠 경우가 있었다. 이 문제점은 Lin 등[22, 25]과 Woo와 Choe[32, 33]에 의해서 발견되었다. 그들의 백트랙 리터럴 선택 알고리즘은 원리적으로는 동일하며 그 정확성(correctness)도 증명되었다. 이후에 Conery는 '고정(static)' DDG를 위한 수정된 알고리즘[7, 8]을 제안하였다. 이 알고리즘은 *마크 집합*(*mark set*)[3]이라고 부르는 자료 구조에 기반을 두고 있다(자세한 것은 2절에서 서술할 것이다). Chang과 Despain[2]도 일찌기 고정

1) 이 논문에서는 앞으로 '리터럴'과 "그 리터럴의 해를 구하기 위한 OR 프로세스"를 다른 지장이 없을 경우 편리한대로 혼용하기로 한다.

2) 여기서 해는 클로즈 내에 나타나는 변수들이 취할 수 있는 값의 조합의 형태를 가지게 된다. 따라서 우리가 무작위하게 백트랙 리터럴을 골라 재수행시킴으로 인하여 변수들의 값이 무절서하게 바뀌는 경우, 빠뜨리는 조합이 생길 수 있다는 뜻이다.

3) 이 자료 구조는 Conery의 논문에서는 "마크들의 집합(marks set)"이라고 원래 불렸으나, 다른 논문에서는 "마크 집합"이라고 부르기도 하므로 이 논문에서는 차후 "마크 집합"이라고 일관되게 호칭하겠다.

DDG를 위한 다른 백트랙 리터럴 선택 알고리즘을 제안한 바 있다. 근래에 Ng과 Leung은 Conery의 수정된 알고리즘을 확장하여 '가변(non-static 혹은 dynamic)' DDG에도 적용될 수 있도록 하였다[27, 28](역시 자세한 것은 2절에서 서술할 것이다). Conery는 그의 새로운 알고리즘이 동일한 실패 상황 하에서는 Lin 등이나 Woo-Choe의 알고리즘과 동일한 백트랙 리터럴을 선택할 수 있다고 주장했다[8]. 그러나 Kim 등[18]은 Conery의 새 알고리즘이나 그것을 확장했다고 하는 Ng-Leung의 알고리즘이, Lin 등이나 Woo-Choe의 정확성이 증명된 알고리즘과 백트랙 리터럴 선택 면에서 동일하지 않음을 보였다; Conery-Ng-Leung의 알고리즘[4]은 Conery의 원래 알고리즘과 마찬가지로 어떤 실패 상황에서는 맞는 해를 역시 놓치는 것이다.

한편 재초기화 문제의 경우, 원래의 AND/OR Process Model에서는 백트랙 리터럴 뒤에 있는 모든 생산자 리터럴들이 재초기화되었다. Lin 등[25]과 Woo-Choe[32, 33]는 Conery의 이러한 보수적이지만 안전한 재초기화 방식을 따랐다. 후에 여러 연구자들이 Conery의 방식이 지나치게 보수적이며, 재수행되는 리터럴 뒤에 있더라도 어떤 생산자 리터럴들은 초기화될 필요가 없음을 지적했다. 즉 *선택적 리터럴 재초기화(literal-level selective resetting)*가 어떤 형태의 분석을 통해 가능하다는 것이다. Chang 등[3]이 처음 선택적 리터럴 재초기화의 가능성을 언급했다. Choe 등[4]과 Park 등[29]은 Chang 등과 같은 생각에 근거하여 선택적 리터럴 초기화 방법들을 독자적으로 개발했다. Conery[7, 8]도 앞의 두 방법과 비슷한, 고정 DDG를 위한 선택적 리터럴 재초기화 방법을 제안했다. 뒤에 Ng과 Leung[27, 28]은 Conery의 방법을 수정하여 역시 가변 DDG에 적용할 수 있도록 하였다. 앞 단락에서 살펴본 백트랙 리터럴 선택의 경우와 마찬가지로, Conery-Ng-Leung의 선택적 재초기화 방법들도 역시 틀렸음을 이 논문에서 증명할 것이다; 이 방법들은 해가 존재함에도 불구하고 어떤 상황 하에서는 잘못된 리터럴 재초기화로 인하여 그 해를 놓칠 수 있는 것이다(자세한 것은 3절에서 서술할 것이다). 그 외 Lin과 Kumar[22], Kim과 Choe[15]도 가변 DDG를 위한 선택적 리터럴 재초기화를 제안하였다. 우리가 이 논문에서 서술할 Conery-Ng-Leung의 알고리즘의 수정된 것(4절 참조)은 Lin-Kumar이나 Kim-Choe 방법의 구체적이고 효율적인 구현의 한 형태로 생각할 수도 있다. 한편 Winsborough[31]와 Kim 등[17, 19]은 *선택적 해 재초기화(solution-level selective resetting)*라고 부를 수 있는 기법들에 대한 연구를 수행했다. 그러나 이 논문에서는 선택적 리터럴 재초기화만 다루기로 하겠다.

논문의 구성은 다음과 같다. 다음 절에서는 Conery-Ng-Leung의 알고리즘들을 자세하게 살펴보고 그들 사이의 관계를 조사한다. 그 알고리즘들을 수정한, 정확성을 갖춘 새로운 방법들과 그 정확성의 증명은 4절에서 기술한다.

2. 마크 집합에 기반을 둔 후방 수행 알고리즘들의 고찰과 비교

이 절에서는 Conery의 수정된 알고리즘과 Ng-Leung의 확장된 알고리즘을 자세히 살펴 보겠다. 이것은 3절에서의 서술과 토의를 위해 필요하다.

2.1 Conery의 수정된 알고리즘

Conery[7, 8]는 그의 원래 알고리즘에 오류가 있다는 지적에 대응하여 수정된 후방 수행 알고리즘을 제안하였다. 그의 새로운 방법은 전 방법에 존재하던 오류의 수정에 더해서, 병렬 수행 모형의 효율적인 구현에 대한 문제를 좀 더 자세히 다루었다.

이 새 방법은 DDG가 컴파일 시에 만들어진 뒤 수행 중에 바뀌지 않는다는 가정을 하고 있다. 즉, 앞에서 언급했다시피 고정 DDG에만 적용할 수 있는 것이다.[5] 그 알고리즘은 여러 연산의 효율을 위하여 여러가지 비트 집합(bit set) 자료 구조를 운영한다: 그것들 중에서 가장 핵심적인 자료 구조는 MARKS(Ng-Leung의 알고리즘에서는 MARK)로 표기하는 *마크(mark)*들의 집합이다. 이 자료 구조는 각 생산자 리터럴에 대해서 만들어진다. 만약 리터럴 j가 리터럴 i 실패의 잠재적인 원인이라면, i는 MARKS[j]의 한 원소가 된다. 리터럴 i가 실패할 때마다, 그것의 인식자 i(Ng-Leung의 알고리즘에서는 m_i)를 리터럴 i의 DDG 상의 모든 조상들의 마크 집합에 추가시킨다. 왜냐하면 전술한 바와 같이 각 조상은 후손의 실패에 대해 잠재적인 원인이 될 수 있기 때문이다. 위와 같은 상황에서의 백트랙 리터럴은 i나 i의 후손을 마크 집합에 포함하고 있는 리터럴들 중에서 선형 순서로 보아 가장 마지막 것이 된다. i 뿐만 아니라 i의 후손들을 고려해야 하는 이유는 *제2형 백트래킹(type II backtracking)*[2, 3] 중의 어떤 경우를 처리하

4) "Conery의 새로운 알고리즘"과 "Ng-Leung의 확장된 알고리즘"을 줄여서 이렇게 부르기로 한다.

5) 그러나 이러한 제한이 있느냐 없느냐 하는 것은 본 논문의 요지와는 상관없는 것이다.

기 위함이다. Conery의 처음 백트랙 리터럴 선택 알고리즘은 제2형 백트래킹을 일으킬 수 있는 *실패 정력*(*failure history*)을 제대로 유지하지 못했다[22, 25, 32, 33]. Conery의 새로운 백트랙 리터럴 선택 알고리즘(그 표기 양식은 Conery의 저술[7, 8]에 있는 것과 약간 차이가 있음)은 다음과 같다.

알고리즘 1: Conery의 마크 집합에 기반을 둔 백트랙 리터럴 선택 알고리즘

입력: 실패한 리터럴 i

출력: i를 위한 백트랙 리터럴 b

방법: /* 제일 처음에는 모든 MARKS 집합은 공집합이다. */

(1) **for** $\forall\ j \in$ ANCESTORS(i) **do** MARKS[j] := MARKS[j] $\cup$ {i} **od**

/*ANCESTORS, DESCENDANTS, PARENTS, CHILDREN 등의 자료 구조는 이 논문에서 DDG 상의 영단어 뜻 그대로 쓰인다. */

(2) J := { j | MARKS[j] $\cap$ ({i} $\cup$ DESCENDANTS(i)) $\neq \emptyset$이고 $j <_l i$ }

/* '$<_l$'[27, 28]은 리터럴들 상에 부여된 선형 순서에 의해 정의되는 선행자-후행자(predecessor-successor) 관계를 나타낸다. */

(3) b := RIGHTMOST(J) ($<_l$)

/* RIGHTMOST는 $<_l$ 상에서 영단어 뜻 그대로 나타낸다. 우리는 내포 루프 모형에 따라 J에 저장되어 있는 후보들 중에서 선형 순서로 보아 가장 오른쪽의 리터럴을 선택한다. */ □

Conery의 새로운 재초기화 알고리즘은 백트랙 리터럴 b 뒤에 있는 모든 생산자 리터럴들을 초기화시키는 것이 아니라, b와 함께 b의 후손의 해(혹은 성공 또는 실패)에 기여하는 리터럴들만 초기화시킨다. 이 리터럴들은 그 CANDIDATES 집합에 b를 가지고 있는 것들이 된다(CANDIDATES 집합에 대해서는 아래에 나와 있다).

알고리즘 2: Conery의 마크 집합에 기반을 둔 재초기화 알고리즘

입력: 백트랙 리터럴 b

방법: /* 리터럴 i에 대한 CANDIDATES 집합은 다음과 같이 정의된다:

CANDIDATES[i] = ANCESTORS(i) $\cup$ $\bigcup_{x \in DESCENDANTS(i)}$ ANCESTORS(x) - {i} */

(1) R := { ι | $i \in$ CANDIDATES[b], $b <_l i$, 그리고 i는 생산자 리터럴 }

/* R은 초기화시킬 후보 리터럴들의 집합이다. */

(2) **while** $R \neq \emptyset$ **do**

$r \in R$이라고 하자; $R := R - \{r\}$

if not (r이 재초기화되거나 취소되었음) **then** R을 재초기화시킨다 **fi**

od □

2.2 Ng과 Leung의 Competition Model

Ng과 Leung[27, 28]은 Lin 등의 전방 수행 알고리즘과 Conery의 수정된 후방 수행 알고리즘에 근거하는 또 하나의 AND-병렬 수행 모형을 제안했다. Ng과 Leung의 후방 수행 알고리즘은 가변 DDG도 다룰 수 있다는 뜻에서 Conery의 방법을 확장한 것으로 간주할 수 있다. 또한 Conery의 방법에서 제2형 백트래킹을 처리하기 위해 실패한 리터럴 L_f의 모든 후손을 사용하는 것과는 달리 그것들의 일부, 즉 L_f의 실패에 기여한 후손들만 사용한다. Ng과 Leung의 백트랙 리터럴 선택과 재초기화를 위한 알고리즘은 다음과 같다.

알고리즘 3: Ng과 Leung의 마크 집합에 기반을 둔 백트랙 리터럴 선택 알고리즘

입력: 실패한 리터럴 L_f

출력: L_f를 위한 백트랙 리터럴 L_b

방법: /* 제일 처음에는 모든 MARK 집합은 공집합이다. */

(1) **for** $\forall\ L \in$ ANCESTORS(L_f) **do** MARK(L) := MARK(L) $\cup$ {m_{Lf}} **od**

(2) L_B := { L | MARK(L) $\cap$ (MARK(L_f) $\cup$ {m_{Lf}}) $\neq \emptyset$ 그리고 $L <_l L_f$ }

/* Conery의 수정된 알고리즘에서 DESCENDANTS(L_f) 대신에, MARK(L_f)가 가변 DDG를 지원하기 위하여 사용된다. */

(3) L_b := RIGHTMOST(L_B) ($<_l$) □

알고리즘 4: Ng과 Leung의 마크 집합에 기반을 둔 재초기화 알고리즘

입력: 백트랙 리터럴 L_b

방법: (1) L_R := { L | MARK(L) ∩ MARK(L_b) ≠ ∅ 그리고 $L_b <_l L$ }

/* L_R은 동적으로 변하는, 초기화될 리터럴들의 집합이다. */

(2) **while** $L_R \neq \emptyset$ **do**

$L_r \in L_R$이라고 하자; $L_R := L_R - \{L_r\}$

if not (L_r이 재초기화되거나 취소되었음) **then**

L_r을 재초기화시킨다

$L_R := L_R \cup \{ L \mid \text{MARK}(L) \cap \text{MARK}(L_r) \neq \emptyset$ 그리고 $L_r <_l L \}$

fi

od □

2.3 두 방법의 비교

Conery의 수정된 백트랙 리터럴 선택 알고리즘(알고리즘 1)과 Ng-Leung이 확장시킨 것(알고리즘 3)은 고정 DDG의 경우에는 동등하다. 이것은 직관적으로도 명백해 보이긴 하지만, 다음 정리는 그것을 명확히 서술하고 있다.

정리 1: 마크 집합에 기반을 둔 백트랙 리터럴 선택 알고리즘들의 동등성

L_f가 실패한 때에는 다음 등식이 성립한다:

$$\{ L \mid \text{MARKS}[L] \cap ((L_f \cup \text{DESCENDANTS}(L_f)) \neq \emptyset \} = \{ L \mid \text{MARK}(L) \cap (\text{MARK}(L_f) \cup \{m_{Lf}\}) \neq \emptyset \}.$$

증명: DESCENDANTS(L_f)에 포함된 리터럴들 중에서 $L_{f1}, L_{f2}, \cdots, L_{fn}$을 실패한 것이라고 하자(그 결과 자신들의 마크를 선조의 마크 집합에 추가시켰다). 그리고 L_S는 DESCENDANTS(L_f)의 리터럴들 중에서 아직까지 실패하지 않은 것들의 집합이라고 하겠다. 즉, L_S = DESCENDANTS(L_f) − { $L_{f1}, L_{f2}, \cdots, L_{fn}$ }. 그러면 식의 좌변인,

$\{ L \mid \text{MARKS}[L] \cap ((L_f \cup \text{DESCENDANTS}(L_f)) \neq \emptyset \}$

$= \{ L \mid \text{MARKS}[L] \cap \{L_f\} \neq \emptyset \} \cup \{ L \mid \text{MARKS}[L] \cap \text{DESCENDANTS}(L_f) \neq \emptyset \}$

(집합 연산의 성질에 의해서)

$= \{ L \mid \text{MARKS}[L] \cap \{L_f\} \neq \emptyset \} \cup \{ L \mid \text{MARKS}[L] \cap (\{ L_{f1}, L_{f2}, \cdots, L_{fn} \} \cup L_S) \neq \emptyset \}$

(L_S의 정의에 의해서)

$= \{ L \mid \text{MARKS}[L] \cap \{L_f\} \neq \emptyset \} \cup \{ L \mid \text{MARKS}[L] \cap \{ L_{f1}, L_{f2}, \cdots L_{fn} \} \neq \emptyset \} \cup \{ L \mid \text{MARKS}[L] \cap L_S \neq \emptyset \}$

(집합 연산의 성질에 의해서)

$= \{ L \mid \text{MARKS}[L] \cap \{L_f\} \neq \emptyset \} \cup \{ L \mid \text{MARKS}[L] \cap \{ L_{f1}, L_{f2}, \cdots, L_{fn} \} \neq \emptyset \} \cup \emptyset$

(L_S에 있는 리터럴은 아직 실패하지 않았기 때문에, 그것의 마크는 어떤 마크 집합에도 포함되어 있지 않으므로)

$= \{ L \mid \text{MARK}(L) \cap \{m_{Lf}\} \neq \emptyset \} \cup \{ L \mid \text{MARK}(L) \cap \{ m_{Lf1}, m_{Lf2}, \cdots, m_{Lfn} \} \neq \emptyset \}$

$= \{ L \mid \text{MARK}(L) \cap \{m_{Lf}\} \neq \emptyset \} \cup \{ L \mid \text{MARK}(L) \cap \text{MARK}(L_f) \neq \emptyset \}$

($L_f \in \text{ANCESTORS}(L_{fi})$이므로 각 $L_{fi}(i = 1, 2, \cdots, n)$는 자신의 마크를 MARK($L_f$)에 추가했기 때문이다. L_S에 있는 리터럴들은 아직 실패하지 않았음을 다시 환기시키고 싶다.)

$= \{ L \mid \text{MARK}(L) \cap (\text{MARK}(L_f) \cup \{m_{Lf}\}) \neq \emptyset \}$

(집합 연산의 성질에 의해서).

이것은 정리 1에 있는 식의 우변이다. □

3절에서 서술할 논지의 핵심은 DDG가 수행 도중에 변화하고 안하고에 관계가 없으므로, 우리는 앞으로 백트랙 리터럴 선택 문제에 대해 논할 때는 Ng-Leung의 알고리즘을 기준으로 하겠다. 그렇게 하더라도 그 논의는 Conery의 방법에도 적용될 수 있는 일반성을 상실하지 않기 때문이다.

한편 재초기화 알고리즘 측면에서 보면, Ng-Leung의 알고리즘으로 구해지는 재초기화될 리터럴들의 집합이 Conery의 방법으로 구해지는 집합에 항상 포함된다. (물론 이것은 의미있는 비교를 위해 고정 DDG를 가정하고 하는 얘기다.) 다음 정리는 앞의 진술을 증명하고 있다.

정리 2: 마크 집합에 기반을 둔 두 재초기화 알고리즘들의 관계

L_b를 백트랙 리터럴이라고 하자. 그리고 L_R^N을 { L | MARK(L) ∩ MARK(L_b) ≠ ∅이고 $L_b <_l L$ }, L_R^C를 { L | $L \in$ CANDIDATES[L_b], $L_b <_l L$, 그리고 L은 생산자} 집합을 각각 나타낸다고 한다. 여기서 CANDIDATES[L] = ANCESTORS(L) ∪ $\bigcup_{x \in \text{DESCENDANTS}(L)}$ ANCESTORS(x) − {L}로 정의된다(L_R^N과 L_R^C는 각각 Ng-Leung과 Conery의 알고리즘에 의해 구해지는 재초기화할 리터럴들의 집합을 표시한다). 그러면, $L_R^N \subseteq L_R^C$이다.

증명: DESCENDANTS(L_b)에 포함된 리터럴들 중에서 $L_{f1}, L_{f2}, \cdots, L_{fn}$을 실패한 것이라고 하자(그 결과 자신들의 마크를 선조의 마크 집합에 추가시켰다). 리터럴 L이 L_R^N의 원소라고 가정하자. 곧, MARK(L) ∩ { $m_{Lf1}, m_{Lf2}, \cdots, m_{Lfn}$ } ≠ ∅이고 $L_b <_l L$이라는 것이다 (MARK(L_b) = { $m_{Lf1}, m_{Lf2}, \cdots, m_{Lfn}$ }임을 주의하라).

리터럴은 실패했을 때 자신의 마크를 선조들의 마크 집합에 추가시키므로, MARK(L)이 m_{1f1}, m_{1f2}, … , m_{1fn} 중의 적어도 하나를 포함하고 있다는 사실은 L이 L_{fi}(i = 1, 2, … , n)의 선조들 중 하나임을 의미한다. 그리고 가정에 의해서 L_{f1}, L_{f2}, … , L_{fn}은 DESCENDANTS(L_b)의 원소들이다.

따라서 $L \in \bigcup_{x \in \mathrm{DESCENDANTS}(i)} \mathrm{ANCESTORS}(x) - \{L_b\}$

($L_b <_l L$임을 주의하라)

$\subseteq$ CANDIDATATES[L_b]. 게다가 MARK(L)이 공집합이 아니라는 사실은 L이 생산자임을 뜻한다. 그러므로 $L \in L_R{}^C$. □

따라서 우리는 재초기화 문제의 '완결성'을 토의하기 위해서는 Conery의 알고리즘을 사용할 것이고, '건전성'을 토론할 때는 Ng-Leung의 확장을 이용할 것이다. 그렇게 하면 각각의 방법에 대해 완결성과 건전성을 모두 검토할 필요가 없을 것이기 때문이다. (완결성과 건전성의 정확한 정의는 다음 절에서 서술하겠다).

3. 정확성—완결성과 건전성 양 측면—의 자세한 분석

이 절에서 우리는 2절에서 서술한 마크 집합에 근거한 백트랙 리터럴 선택 및 재초기화 알고리즘들이 바르지 않다—완결되지도 건전하지도—는 것을 반례(counter-example)를 통해 보일 것이다.

우선 후방 수행 알고리즘에 대한 '완결성'과 '건전성'이라는 개념들의 정의부터 하겠다. 백트랙 리터럴 선택 문제의 경우, 실패가 발생했을 때 어떤 백트랙 리터럴 선택 알고리즘에 의해 구해진 후보 백트랙 리터럴들의 집합이 그 실패를 고칠 가능성이 '전혀 없는' 리터럴은 포함하고 있지 않는 것을 *건전성*(*soundness*)이라고 한다. 만약 그 집합이 실패를 차유할 일말의 가능성을 가지고 있는 모든 리터럴들을 빠짐없이 포함하고 있으면 우리는 그 알고리즘이 *완결성*(*completeness*)을 가지고 있다고 말한다.[6]

비슷한 정의를 재초기화 알고리즘의 완결성과 건전성에 대해서도 할 수 있다. 한 리터러이 재수행될 때, (재초기화 알고리즘의) 건전성이란 어떤 재초기화 알고리즘에 의해 구해진 재초기화될 리터럴들의 집합이 질의(query)에 대한 해(solution)에 잠재적으로 참여할 수 있는 부분(*partial*) 해를 만들지 않은 리터럴은 포함하지 않음을 뜻한다. 한편 재초기화 알고리즘의 완결성이란, 전술한 가능성이 있는 모든 리터럴은 빠짐없이 그 집합에 포함되도록 하는가의 여부를 말한다.

따라서 달리 표현하자면, 건전성은 알고리즘의 효율(혹은 지능(intelligence))과 관련이 있고(즉, 쓸모없는 재수행이나 재초기화 혹은 병렬 수행 가능한 일들의 순차화 같은 것을 피하는 일), 반면에 완결성은 존재하는 모든 적법한 해를 빠뜨리지 않고 발견하는 것과 관련이 있다. 재론의 여지없이 완결성이 건전성보다 훨씬 중요하다.

이제 위의 정의 하에, 마크 집합에 근거한 백트랙 리터럴 선택/재초기화 알고리즘들의 건전성과 완결성을 토의하기 위해서는 먼저 "참조 집합(reference set)", 즉 알고리즘에 의해 구해지는 집합과 비교의 기준이 되는 "후보 백트랙/재초기화 리터럴들의 정확한 집합"을 정의해야 한다.

아래의 정리 3과 4는 각각 Lin 등의 논문[22, 25]에 나오는 정리 1과 2를 본 논문에서 사용하는 표기법을 이용해 약간 변형시킨 것이다.

정리 3[22, 25]: 백트랙 리터럴 선택을 위한 참조 집합

L_f가 실패했을 때 어떤 리터럴 l_b를 재수행하는 것이 그 실패를 고쳤다면, $l_b \in \{ L \mid L \in \mathrm{ANCESTORS}(L_f) \cup \mathrm{ANCESTORS}(M)$이고, M의 실패가 L_f의 재수행을 야기했으며[7] $L <_l L_f \}$이다. □

정리 3에 의해 정의된 백트랙 리터럴 선택을 위한 참조 집합이 내포 루프 모형 하에서는 완결성을 유지하면서 보다 축소될 수 있음을 다음 정리는 말하고 있다.

정리 4[22, 25]: 내포 루프 모형 하에서 백트랙 리터럴 선택을 위한 축소된 참조 집합

$\mathrm{RIGHTMOST}_{<_l}\{ L \mid L \in \mathrm{ANCESTORS}(L_f) \cup \mathrm{ANCESTORS}(M)$이고, M의 실패가 L_f의 재수행을 야기했으며 $L <_l L_f \} = \mathrm{RIGHTMOST}_{<_l}\{ L \mid L \in \mathrm{PARENTS}(L_f) \cup \mathrm{PARENTS}(M)$이고, M의 실패가 L_f의 재수행을 야기했으며 $L <_l L_f \}$. □

6) 사실 완결성과 건전성의 이러한 정의는 수학 분야에서 보편적으로 사용하고 있는 이 용어들의 뜻과 거의 같다. 단지 그것을 후방 수행 알고리즘에 적용한 것 뿐이라고 할 수 있다.

7) 이 논문에서 "재수행 야기"란 말을 쓸 때는 묵시적으로 "직접(directly) 혹은 간접적으로(indirectly) 재수행을 야기"했음을 의미하는 것으로 사용한다[17, 22, 25].

정리 4에 따르면 우리가 내포 루프 모형을 채택한 경우(즉, 후보 백트랙 리터럴 중에서 가장 오른쪽 것을 항상 선택하는 경우), 집합 $\{ L \mid L \in \mathrm{PARENTS}(L_f) \cup \mathrm{PARENTS}(M)$이고, M의 실패가 L_f의 재수행을 야기했으며 $L <_l L_f \}$를 백트랙 리터럴 선택의 완결성 증명을 위한 참조 집합으로 사용할 수 있는 것이다. $\mathrm{PARENTS}(L)$은 항상 $\mathrm{ANCESTORS}(L)$의 부분집합이므로, 이러한 축소된 집합의 경우 건전성은 저절로 성립되게 된다. 우리는 단일화 실패의 원인을 항별로(term by term) 분석[1, 5, 10, 30]하지 않기 때문에, 정리 4에서 정의된 집합이 참조 집합의 역할을 할 수 있다.

다음은 재초기화를 위한 참조 집합을 정의한다.

정리 5: 재초기화를 위한 참조 집합

L_m의 현재 바인딩이 (재수행이나 재초기화에 의해) 바뀌었을 때, L_r이 재초기화되어야 한다면 다음 조건들이 성립한다:

(1) $L_f \in \mathrm{CHILDREN}(L_m)$이고 L_f의 실패가 L_r의 재수행을 야기한, 그러한 L_f가 존재한다.

(2) $L_m <_l L_r$.

(3) $L_r \notin (\mathrm{DESCENDANTS}(L_m) \cup \mathrm{DESCENDANTS}(L_c))$. 여기서 L_c는 재초기화되거나 취소되는 리터럴을 나타낸다.

증명: 조건 (2)와 (3)이 성립해야만 한다는 것은, (i) 우리가 값들의 조합을 생성하기 위해 내포 루프 모형을 채택한다는 것과, (ii) 어떤 리터럴의 재초기화와 취소가 동시에 필요하다면 그 리터럴에 전달되는 값들이 이제는 의미없다는 뜻이므로 취소가 재초기화보다 우선해야 한다[4, 16, 22]는 사실로부터 자명하다. 따라서 조건 (1)에만 초점을 맞추겠다. 대우(contraposition)를 이용해 증명한다. 조건 (1)이 성립하지 않는다고 가정하자. L_r에 대해서는 다음 두 가지 경우가 있을 수 있다:

경우 1.

L_r은 여태까지 재수행된 적이 없다. 즉, 그것의 첫 번째 바인딩이 어떤 리터럴에 의해서도 거부되지 않은 것이다. 이런 경우 L_r이 재초기화될 필요가 없음은 명백하다.

경우 2.

L_r은 재수행된 적이 있다. 즉, L_r의 어떤 바인딩이 L_m의 자식이 아닌 어떤 실패한 리터럴에 의해 거부된 적이 있다. 이런 경우, L_m의 바인딩을 바꾸는 것은 실패한 리터럴이 L_m의 자식이 아니기 때문에 그 실패를 치유할 수가 없다. 결과적으로 L_r이 재수행되는 것을 막을 수도 없다. 달리 말하자면, L_m의 이전 바인딩은 설령 재초기화에 의해 다시 사용할 수 있게 될지라도 과거와 꼭같은 이유로 실패한 리터럴에 의해 다시 거부될 것이다. 한마디로 L_r은 재초기화할 필요가 없는 것이다.

그러므로 위의 두 경우에 의해서 L_r을 재초기화해야 할 때에는 조건 (1)이 성립한다. □

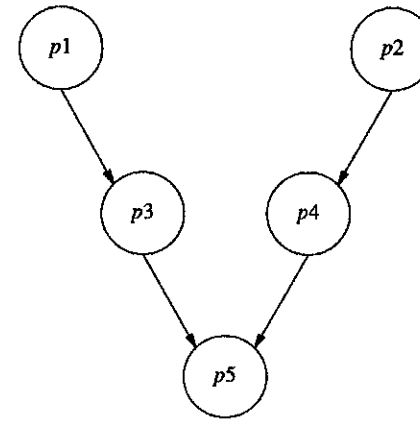

그림 1 재초기화의 참조 집합을 설명하기 위한 DDG

어쩌면 L_m의 바인딩을 바꾸는 것이 설사 실패한 리터럴이 L_m의 자식은 아니라 하더라도 L_m의 후손이면 그 실패를 고칠 수 있지 않을까 하고 생각할지 모르겠다. 그림 1의 DDG를 생각해 보자. $p5$가 실패하고 $p2$는 $p5$에 의해 재수행되었다고 가정한다. 이 경우 $p1$의 바인딩을 바꾸는 것은 분명히 $p5$의 실패를 고칠 수도 있다. 그러나 $p5$가 $p2$의 재수행을 일으키기 위해서는 먼저 $p4$, 다음 $p3$의 재수행이 내포 루프 모형에서는 $p2$보다 선행해야 한다. 왜냐하면 $p3$과 $p4$가 백트랙 리터럴 선택을 위한 참조 집합에 포함되기 때문이다(정리 3과 4를 보라). 만약 $p3$를 재수행한 뒤에도 $p2$의 재수행이 추가로 필요하다면, 이제 $p3$($p1$의 자식이다)이 $p2$의 재수행을 야기하는 것이다. 이것은 실패한 리터럴이 L_m(즉 $p1$)의 자식이 아닌 후손이라는 가정에 모순이다.

우리는 역시 단일화 실패의 원인을 분석하거나 선택적해 재초기화를 수행하지 않으므로, 정리 5에서 정의된 집합은 재초기화 알고리즘의 정확성 증명을 위한 참조 집합으로 사용될 수 있다.

3.1 건전성에 대하여

다음 정리는 2절의 두 백트랙 리터럴 선택 알고리즘

(알고리즘 1과 3)의 불건전성(unsoundness)에 대해 말하고 있다.

정리 6[18]: 마크 집합에 기반을 둔 백트랙 리터럴 선택 알고리즘들의 불건전성

L_f가 실패했을 때, 다음과 같은 l이 존재할 수 있다:

$l \in \{ L \mid \text{MARK}(L) \cap (\text{MARK}(L_f) \cup \{m_{Lf}\}) \neq \emptyset$ 그리고 $L <_l L_f \}$이고

$l \notin \{ L \mid L \in \text{PARENTS}(L_f) \cup \text{PARENTS}(M)$이고, M의 실패가 L_f의 재수행을 야기했으며 $L <_l L_f \}$.

즉, 마크 집합에 근거한 2절에 있는 백트랙 리터럴 선택 알고리즘들은 건전하지 않다는 것이다. □

정리 6에 의하면, 마크 집합에 근거한 백트랙 리터럴 선택 알고리즘들은 실패를 해소하는 데 전혀 도움이 되지 않음이 명백한 재수행도 할 수 있다는 것이다. 한마디로, 이 알고리즘들은 어떤 상황에서는 바른 알고리즘들[17-19, 21, 22, 25, 31-33]보다 보수적이다(결과적으로 덜 지능적이고 덜 효율적일 가능성이 있다).

다음 정리는 2절의 두 재초기화 알고리즘들(알고리즘 2와 4)의 불건전성을 얘기하고 있다.

정리 7: 마크 집합에 기반을 둔 재초기화 알고리즘들의 불건전성

L_b와 $L_R{}^N$을 정리 2에서 정의된 대로 하자. 또한 $L_R{}^R$을 $\{ L \mid L_f \in \text{CHILDREN}(L_b)$이고 L_f의 실패가 L의 재수행을 야기하는 L_f가 존재하고, $L_b <_l L \}$이라는 집합을 나타낸다고 하자. (이 집합은 재초기화를 위한 참조 집합의 조건(정리 5)에서 세번째 조건을 삭제하고 만들어진 것이다. 왜냐하면 $L_R{}^N$과 $L_R{}^C$도 또한 그 조건이 없이 정의되어 있기 때문이다.) 그럴 경우, $l \in L_R{}^N$이고 $l \notin L_R{}^R$인 l이 존재한다. 즉 2절에 있는 마크 집합에 근거한 재초기화 알고리즘들은 건전하지 않다.

증명: 반례에 의한다. 그림 2와 같은 DDG가 현재 수행 중인 클로즈를 위해 구성되었다고 한다. 그리고 다음과 같은 일련의 진행이 일어났다고 한다(실제로 아래와 같은 순서로 실패가 발생하는 경우, Ng-Leung의 알고리즘은 정확히 이러한 진행을 보인다):

(1) $p7$이 실패한다.

(2) $p5$가 재수행되고 성공한다.

(3) $p6$가 실패한다.

(4) $p4$가 재수행되고 실패한다.

(5) $p2$가 재수행되고 성공한다.

$p2$가 재수행될 때, 집합 $\{ L \mid \text{MARK}(L) \cap \text{MARK}(p2) \neq \emptyset$이고 $p2 <_l L \}$은 $\{ p3, p4, p5 \}$가 된다(MARK($p2$) = $\{ m_{p4}, m_{p6}, m_{p7} \}$). 반면에 집합 $\{ L \mid L_f \in \text{CHILDREN}(p2)$이고 L_f의 실패가 L의 재수행을 야기하는 L_f가 존재하고, $p2 <_l L \}$은 $\{p5\}$이다(L_f = $p7$). □

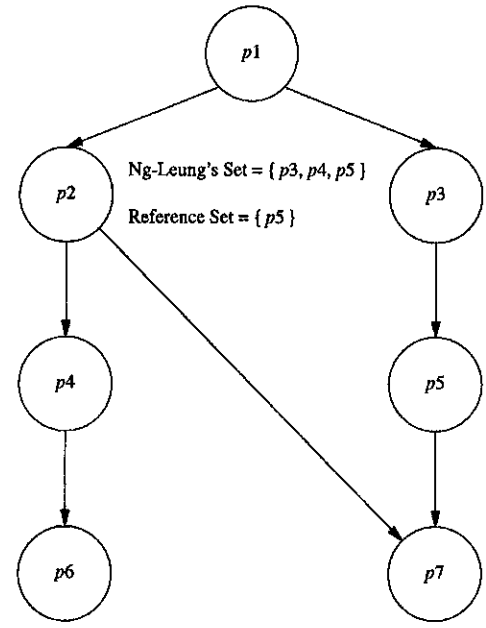

그림 2 재초기화의 불건전성을 보여주는 DDG

그림 2에서 볼 수 있듯이 $p3$나 $p4$를 재초기화하는 것은 쓸데없는 일이다. 그러한 재초기화가 질의에 대한 해 튜플(solution tuple)에 추가시키는 것이 없기 때문이다.

3.2 완결성에 대하여

더더욱 심각한 것은, 마크 집합에 근거한 백트랙 리터럴 선택과 재초기화 알고리즘들이 완결성도 결여되어 있다는 것이다. 그 결과 어떤 경우에는 적법한 해를 놓칠 수도 있다. 다음 정리는 2장에 있는 두 백트랙 리터럴 선택 알고리즘들(알고리즘 1과 3)의 불완결성(incompleteness)을 말하고 있다.

정리 8: 마크 집합에 기반을 둔 백트랙 리터럴 선택 알고리즘들의 불완결성

L_f가 실패했을 때, 다음과 같은 l이 존재할 수 있다:

$l \in \{ L \mid L \in \text{PARENTS}(L_f) \cup \text{PARENTS}(M)$이고, M의 실패가 L_f의 재수행을 야기했으며 $L <_l L_f \}$이고

$l \notin \{ L \mid \text{MARK}(L) \cap (\text{MARK}(L_f) \cup \{m_{Lf}\}) \neq \emptyset$ 그리고 $L <_l L_f \}$.

즉, 마크 집합에 근거한 2절에 있는 백트랙 리터럴 선택 알고리즘들은 완결성이 없다는 것이다.

증명: 반례에 의한다. 그림 3과 같은 DDG가 현재 수행 중인 클로즈를 위해 구성되었다고 한다. 그리고 다음과 같은 일련의 진행이 일어났다고 한다:

(1) $p8$이 실패한다.
(2) $p4$가 재수행되고 성공한다.
(3) $p7$이 실패한다.
(4) $p4$가 재수행되고 실패한다.
(5) $p3$이 재수행되고 실패한다.

$p3$이 재수행될 때, 집합 $\{ L \mid \text{MARK}(L) \cap (\text{MARK}(p3) \cup \{m_{p3}\}) \neq \emptyset$ 그리고 $L <_l p3 \}$은 $\{p1\}$이다. 반면에 집합 $\{ L \mid L \in \text{PARENTS}(p3) \cup \text{PARENTS}(M)$이고, M의 실패가 $p3$의 재수행을 야기했으며 $L <_l p3 \}$은 $\{ p1, p2 \}$이다($M = p4$ 혹은 $p7$ 혹은 $p8$). □

다음 정리는 2절에 있는 두 재초기화 알고리즘들(알고리즘 2와 4)이 완결성이 없음을 말하고 있다

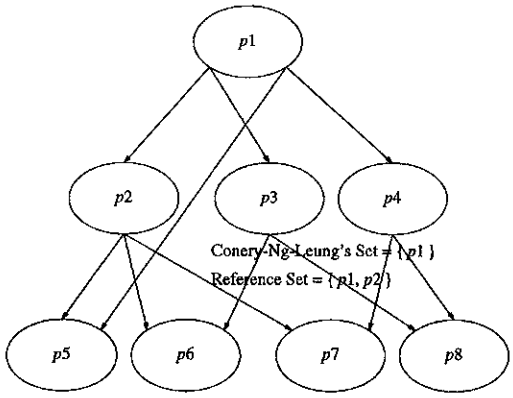

그림 3 백트랙 리터럴 선택의 불완결성을 보여주는 DDG

정리 9: 마크 집합에 기반을 둔 재초기화 알고리즘들의 불완결성

L_b, $L_R{}^C$, 그리고 $L_R{}^R$을 정리 2와 7에서 정의된 바와 같다고 한다. 그럴 경우, $l \in L_R{}^R$이고 $l \notin L_R{}^C$인 l이 존재한다.

증명: 반례에 의한다. 그림 4와 같은 DDG가 현재 수행 중인 클로즈를 위해 구성되었다고 한다. 그리고 다음과 같은 일련의 진행이 일어났다고 한다:

(1) $p8$이 실패한다.
(2) $p4$가 재수행되고 성공한다
(3) $p7$이 실패한다.
(4) $p4$가 재수행되고 실패한다.
(5) $p3$이 재수행되고 성공한다.
(6) $p4$가 재초기화된다.
(7) $p5$가 실패한다.
(8) $p2$가 재수행되고 성공한다.

$p2$가 재수행될 때, 집합 $\{ i \mid i \in \text{CANDIDATES}[p2], p2 <_l i$, 그리고 i는 생산자 $\}$는 $\{p4\}$이다. 반면에 집합 $\{ L \mid L_f \in \text{CHILDREN}(p2)$이고 L_f의 실패가 L의 재수행을 야기하는 L_f가 존재하고, $p2 <_l L \}$은 $\{ p3, p4 \}$이다($L_f = p7$). □

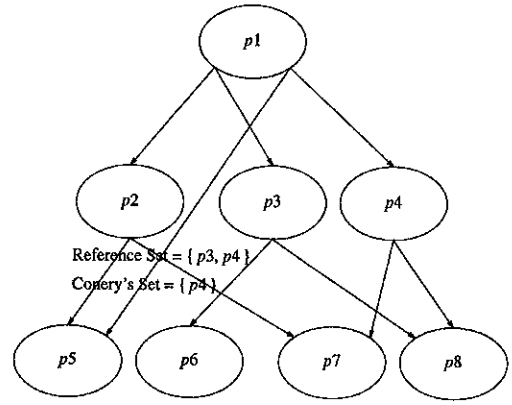

그림 4 재초기화의 불완결성을 보여주는 DDG

4. 올바른 후방 수행을 위한 수정 제안

이 절에서 우리는 마크 집합에 근거한 수정된 후방 수행(백트랙 리터럴 선택과 재초기화) 알고리즘을 제안하고 그 정확성을 증명한다.

4.1 왜 마크 집합과 유사한 자료 구조가 필요한가?

완결성을 갖추고 있다고 알려진 백트랙 리터럴 선택 알고리즘들[2, 3, 17-19, 22, 25, 31-33]과 재초기화 알고리즘들[4, 15-17, 19, 22, 29, 31]이 이미 여럿 존재한다. (물론 개중에는 다른 것들보다 좀 더 보수적이거나, 고정 DDG에만 적용된다거나, 분명한 정확성 증명이 없이 제안된 것들도 섞여 있긴 하다.) 그렇다면 왜 마크 집합에 근거한 새로운 알고리즘이 필요한지에 대한 의문이 제기될 수 있다.

2절에서 설명했다시피, Conery는 후방 수행 알고리즘의 효율적인 구현을 생각하면서 그의 수정된 알고리즘을 제안하였다[7, 8]. 그 알고리즘의 모든 자료 구조는 비

트 집합으로 표현될 수 있고 관련된 모든 연산은 빠른 비트 연산으로 처리된다. 그러기에 효율적인 구체적(low-level) 구현이 가능하게 된다. 따라서 Conery가 언급했듯이, 우리는 마크 집합에 근거한 알고리즘을 채택함으로써, 논리 프로그램의 병렬 수행을 위한 그럴듯한 가상 기계(virtual machine) 혹은 그 이상을 구성할 수 있게 된다[21].

여기에 더해서, 비록 *Conery-Ng-Leung*은 그러한 측면을 살피지 않았지만, 우리는 마크 집합과 같은 비트 집합이라는 자료 구조와 그 사용 방식이 튜플 생성을 위한 내포 루프 모형의 대안을 제시할 잠재적 가능성이 있다고 생각한다. 많은 연구자들이 내포 루프 모형의 엄격한 '선형' 순서에 불편한 느낌을 가졌으리라고 우리는 믿는다. 물론 우리도 예외가 아니다. 그림 5의 DDG를 생각해 보자.

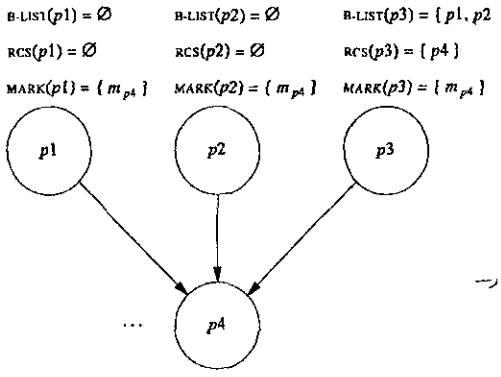

그림 5 마크 집합의 잠재적 유용성을 보이기 위한 DDG

*p4*가 실패했을 때, 원칙적으로는 { *p1*, *p2*, *p3* } 중 어느 리터럴이든지 백트랙 리터럴이 될 수 있다. 그러나 내포 루프 모형에서는 그 DDG가 변하지 않는 한, *p3* (부여한 순서에 따라 *p1* 혹은 *p2*일 수도 있다)이 항상 백트랙 리터럴로 선택되게 되어 있다. 원칙적인 면에서 뿐만이 아니라 실질적인 관점에서도 이러한 상황은 바람직하지 않다. 다음과 같은 경우를 생각해 보자: (1) *p3*의 해를 구하는 일은 매우 어렵거나 매우 느리다, (2) *p3*의 해 중에는 질의에 대한 답에 참여할 수 있는 것들이 별로 없다, (3) *p1*이나 *p2*의 해를 구하는 일은 쉽고 빠르다, 그리고 (4) 질의의 답에 참여할 수 있는 *p1*이나 *p2*의 해는 매우 많이 존재한다. 이러한 경우, *p1*이나 *p2*를 백트랙 리터럴로 선택해 재수행하는 것이 *p3*를 재수행하는 것 보다 훨씬 나은 일이 됨은 명백하다. (프로그램의 정적 분석(static analysis)이 이러한 결정을 내리는 데 도움이 될 수 있으나, 컴파일 시 분석이라는 한계 때문에 만족스럽지 못할 가능성이 높다.)

그렇다면 일반적으로 말해서, 실패한 리터럴을 고치기 위해서 후보 백트랙 리터럴들이 경쟁하면서 해를 구하도록 할 수는 없는가 하는 의문을 제기할 수 있겠다. 그러나 이러한 시도는 마크 집합에 근거한 것을 제외한 기존의 후방 수행 알고리즘들에서는 달성되기가 애초부터 불가능해 보인다. 앞서의 예를 계속 생각해 보자. 그림 5에 이 예에 대한 세 개의 백트랙 리터럴 선택 알고리즘들의 핵심 자료 구조의 갱신 내용을 추가로 보였다. 즉, Lin 등의 B-LIST[22, 25], Woo-Choe의 *재수행 야기 집합* (*redo cause set*; RCS)[32, 33], Ng-Leung의 마크 집합(MARK)들이다. 그림 5에서 볼 수 있듯이, Ng-Leung의 (마크 집합에 기반을 둔) 알고리즘은 백트랙 후보들간의 '대칭성(symmetricity)'을 유지하면서 실패 경력을 저장한다. 반면에 다른 두 알고리즘은 그러한 정보를 백트랙 리터럴에만 부착하고, 결과적으로 그 순간부터 각 리터럴들의 구체적 구현이 얼마나 효율적일 수 있는가[23, 24] 하는 점은 고려치 않고 리터럴들의 순서를 고정시켜 버린다(*p3*가 { *p1*, *p2*, *p3* } 중 가장 오른쪽 혹은 가장 후위의 리터럴이 되는 것이다).

그러므로 우리는 (1) 마크 집합과 유사한 자료 구조를 사용하고 싶고, 또 (2) 완결성과 건전성을 갖추고 있는 알고리즘을 만들고 싶은 것이다. (사실, 마크 집합에 근거한 알고리즘들에 대한 우리들의 수정(4.3절 참조)은 앞의 두 목표는 달성하지만 원래의 알고리즘들이 가지고 있던 대칭성은 깨뜨린다. 우리는 현재 우리들의 수정된 방법에서도 그러한 대칭성을 유지할 가능성을 조사하고 있다. 그래서 진정한 뜻에서의 또다른 Competition Model[27, 28]을 만들려고 한다.)

4.2 근본 원리

무엇보다도 먼저 마크 집합에 기반을 둔 알고리즘들의 오류가 발생하는 원인을 분석하는 것이 바른 순서라고 생각된다.

[15-17]에서 지적되었다시피, 백트랙 리터럴 선택과 재초기화는 다음과 같은 뜻에서 서로 '이원적(dual)' 동작이다: (1) 실패한 리터럴에 영향을 끼친 리터럴들이 후보 백트랙 리터럴이 되고, (2) 백트랙 리터럴에 영향을 받은 리터럴들은 재초기화되거나 취소되어야 한다. 그러기에 2절에서 살펴본 마크 집합에 근거한 알고리즘들은 백트랙 리터럴 선택과 재초기화 두 가지 일을 위해 단지 마크 집합 자료 구조만 사용하는 것이다. 이것은 다시 이 자료 구조의 처리를 적절하게 하지 못하면 잘못된 백트랙 리터럴 선택과 재초기화 모두를 초래할 수도

있음을 암시한다. 실제로 Conery-Ng-Leung 알고리즘들의 오류가 바로 이 경우이다. 따라서 재초기화는 재수행에 의해 야기되므로, 백트랙 리터럴 선택 시 발생하는 그 자료 구조의 부적절한 갱신을 분석하는 것만으로 충분하리라 본다.

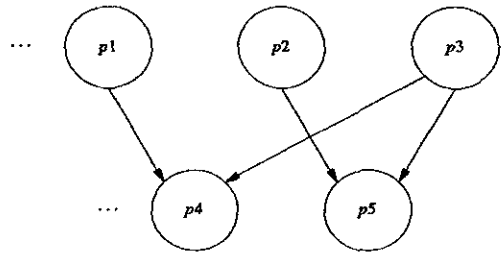

그림 6 마크 집합에 근거한 알고리즘들의 오류 원인을 보이기 위한 DDG

그림 6에 있는 DDG를 생각해 보자. 그리고 다음과 같은 일련의 진행이 일어났다고 가정하자:

(1) $p5$가 실패한다.
(2) $p3$이 재수행되고 성공한다.
(3) $p4$가 실패한다.
(4) $p3$이 재수행되고 실패한다.

이 때 제2형 백트래킹이 $p2$로 일어나야 하나, Conery의 원래 알고리즘[6, 9]은 $p2$ 대신 $p1$으로 잘못 백트래킹을 했었다. 한편으로, $p5$보다 $p4$의 실패가 먼저 발생했다면 그 알고리즘은 $p2$라는 바른 백트랙 리터럴을 선택했을 것이다. 즉, 그 알고리즘의 결함은 가장 최근에 일어난 실패를 제외하고는 실패 경력을 잊어 버린다는 것이다. 그 이유는 백트랙 리터럴의 결정을 *재수행 리스트(redo list)*라고 부르는 융통성없는(inflexible) 자료 구조에 의해 하기 때문이다(자세한 것은 [6, 9]를 참조하라). 이러한 정보의 상실은 당연히 위의 예에서 보듯이 어떤 상황에서는 잘못된 동작을 유발하는 것이다. 그 예에서는 $p3$이 $p5$가 실패했었다는 사실을 잊은 것이다.

달리 표현하자면, Conery의 원래 알고리즘은 제2형 백트래킹의 일반적으로 일어날 수 있는 상황을 지나치게 단순화시켰다고 할 수 있다; 그 알고리즘은 두 개 이상의 (자식) 리터럴에 의해 야기되는 제2형 백트래킹의 경우에 주의를 기울이지 않은 것이다.

Conery의 새 알고리즘과 Ng-Leung의 확장된 것은 마크 집합이라고 부르는 자료 구조에 실패한 (후손) 리터럴들의 '마크'를 저장함으로써, 위에서 언급한 결함을 교정했다. 앞의 예에서 이 새 알고리즘들은 $p2$로 바르게 백트래킹한다. 그럼에도 불구하고 이 알고리즘들은 우리가 3절에서 보았듯이 여전히 적법한 해를 놓칠 수 있다. 앞의 예에서 계속하여 $p2$가 재수행되고 실패했다고 추가로 가정해 보자. $p1$의 마크 집합(= $\{m_{p4}\}$)과 $p2$의 마크 집합(=$\{m_{p5}\}$)의 교집합이 없기 때문에 이들 알고리즘은 $p1$으로 백트랙하지 않는다. 사실, 이러한 현재의 마크 집합들은 Conery-Ng-Leung이 생각하는 마크 집합의 정확한 뜻, 즉 "리터럴 P가 다른 리터럴 Q의 마크를 가지고 있다는 것은 P가 Q의 실패에 책임이 있을 수 있음을 의미한다"[27, 28]는 것에 비춰봐도 틀린 것이다. 다음과 같은 이유에서이다. 그들의 알고리즘에서는 $p3$의 실패에 의해 $p2$로 백트랙하는데 이것은 $p2$가 $p3$의 실패에 책임이 있을 수 있음을 뜻한다. 따라서 $p2$는 재수행된 뒤, $p3$의 마크와 $p3$가 포함하고 있던 마크들(= $\{m_{p4}, m_{p5}\}$)을 역시 포함해야 하는 것이다. 그러나 그들의 알고리즘은 실패한 리터럴이 가지고 있던 마크들을 백트랙 리터럴의 마크 집합에 추가시키지 않고 다만 실패한 리터럴의 마크를 그 선조들에게 더할 뿐이다. 그러므로 제2형 백트래킹이 연속적으로 발생하는 경우 잘못된 백트랙 리터럴을 선택할 수 있다. (참고삼아 얘기하자면, 사실 그들의 알고리즘은 $p2$로 백트래킹할 때 이미 잘못된 길로 들어선 것이다. $p1$과 $p2$는 모두 $p3$의 실패로 인한 제2형 백트래킹을 할 때의 후보 백트랙 리터럴들이었다. 따라서 $p2$로 백트래킹할 때는 $p3$에 의해 재수행이 야기된 $p2$가 다시 실패할 경우를 대비하여 ($p2$의 대안인) $p1$이 존재함을 기억해야만 하는 것이다.)

한번 더 달리 표현하자면, 그들의 알고리즘도 역시 제2형 백트래킹이 일어나는 일반적인 상황을 지나치게 단순하게 생각한 것이다; 제2형 백트래킹은 두 리터럴(하나는 실패한 리터럴이고 다른 것은 백트랙 리터럴)이 DDG 상에서 공통의 후손을 가질 때($p2$와 $p3$는 공통의 후손 $p5$를 가진다)뿐만이 아니라, 그들의 후손들이 공통의 선조를 가질 때($p2$의 후손 $p5$와 $p1$의 후손 $p4$는 공통의 선조 $p3$를 가진다)에도 발생할 수 있다. Conery-Ng-Leung의 알고리즘은 전자의 경우는 고려하고 있으나 후자의 경우는 무시하고 있는 것이다.

게다가 Conery-Ng-Leung의 알고리즘에서는 마크 집합의 종류가 단 한 가지 밖에 없으므로, 실패한 리터럴의 선조들을 서로 구별할 수가 없다. 예를 들어, 그림 6에서 $p5$가 실패했을 때 그것의 마크는 $p2$와 $p3$의 마크 집합에 더해진다. 비록 $p3$는 백트랙 리터럴이고 $p2$는 아님에도 불구하고, 이 두 리터럴들의 마크 집합은 똑같아 보인다. 그 결과, 마크 집합만 보고는 $p5$에 의해 실제로 ($p2$와 $p3$ 중) 어느 리터럴이 재수행되었는지를 알 수가

없다. 당연히 우리는 *p5*가 *p2*와 *p3* 모두의 재수행을 야기했다고 생각할 수 밖에 없다. 이렇게 리터럴들간의 필요한 구별이 안됨으로써 그들의 알고리즘은 불건전, 즉 보다 보수적이 되는 것이다. 비슷한 이유로, 부모뿐만 아니라 다른 선조들에게까지 마크를 추가시키는 것도 불건전성의 다른 원인이다(정리 4도 참조하라).

4.3 세분 마크 집합 방법

4.2절에서 서술한 관찰을 통해 우리는 *세분 마크 집합 방법(Refined Mark Set Method)*이라고 부를 수 있는, 정확성-완결성과 건전성 모두-을 갖추고 마크 집합에 바탕을 둔 새로운 후방 수행 방법을 제안한다.

4.2절에서 토의한 원리에 의해 우리 알고리즘에서는 두 가지 종류의 마크 집합이 존재한다; '후보' 백트랙 리터럴들을 위한 마크 집합(C-MARK)과 '실제' 백트랙 리터럴들을 위한 마크 집합(B-MARK)이 그것이다. 우리는 알고리즘을 건전하게 만들기 위해서 이렇게 마크 집합을 '세분'한 것이다. 만약 Q가 P의 실패에 대해 '당연한(trivial)' 후보 백트랙 리터럴, 즉 P의 부모 중 하나라고 한다면 $m_P \in$ C-MARK(Q)이다. m_P가 B-MARK(Q)에 포함되어 있으면, P가 실제로 Q의 재수행을 야기한 것이다. 그리고 우리 알고리즘이 완결성을 갖도록 하기 위해서, P가 실패해서 Q로의 백트래킹이 일어났다면 B-MARK(P)에 포함되어 있는 마크들과 m_P를 B-MARK(Q)에 더해 준다.

백트랙 리터럴 선택과 재초기화를 위한 우리들의 알고리즘은 다음과 같다.

알고리즘 5: 세분 마크 집합 방법의 백트랙 리터럴 선택

입력: 실패한 리터럴 L_f

출력: L_f를 위한 백트랙 리터럴 L_b

방법: /* 제일 처음에는 모든 C-MARK, B-MARK 집합은 공집합이다. */

(1) **for** $\forall L \in$ PARENTS(L_f) **do** C-MARK(L) := C-MARK(L) $\cup$ {m_{Lf}} **od**

(2) L_B := { L | C-MARK(L) $\cap$ (B-MARK(L_f) $\cup$ {m_{Lf}}) $\neq \emptyset$ 그리고 $L <_l L_f$ }

(3) L_b := RIGHTMOST$_{<_l}$(L_B)

(4) B-MARK(L_b) := B-MARK(L_b) $\cup$ {m_{Lf}} $\cup$ B-MARK(L_f) □

알고리즘 6: 세분 마크 집합 방법의 재초기화

입력: 백트랙 리터럴 L_b

방법: (1) L_R := { L | C-MARK(L_b) $\cap$ B-MARK(L) $\neq \emptyset$ 그리고 $L_b <_l L$ }; C-MARK(L_b) := $\emptyset$

/* C-MARK(L_b)는 그 역할을 다했으므로 다시 초기값으로 돌아간다. */

(2) **while** $L_R \neq \emptyset$ **do**

$L_r \in L_R$이라고 하자; $L_R := L_R - \{L_r\}$

if not (L_r이 재초기화되거나 취소되었음) **then**

L_r을 재초기화시킨다; B-MARK(L_r) := $\emptyset$

$L_R := L_R \cup$ { L | C-MARK(L_r) $\cap$ B-MARK(L) $\neq \emptyset$ 그리고 $L_r <_l L$} C-MARK(L_r) := $\emptyset$

/* L_r은 재초기화(곧, 처음부터 다시 시작)했기 때문에 그 마크 집합도 다시 초기화된다. */

fi

od □

4.4 정확성 증명

이 절에서는 우리의 수정된 알고리즘의 정확성을 증명하겠다. 다음 정리는 우리 알고리즘이 나중의 백트랙 리터럴 선택과 재초기화를 정확히 하기 위하여, AND-병렬 수행 동안의 실패 경력을 바르게 유지함을 말하고 있다. 이 정리는 세분 마크 집합 방법의 정확성 증명을 위한 기초가 된다.

정리 10: 세분 마크 집합 방법에서의 실패 경력의 바른 유지

L_f가 실패한 때에는 다음 등식이 성립한다:

{ L | C-MARK(L) $\cap$ B-MARK(L_f) $\neq \emptyset$ 그리고 $L <_l L_f$ } = { L | $L \in$ PARENTS(M)이고, M의 실패가 L_f의 재수행을 야기했으며 $L <_l L_f$ }.

증명: 증명은 귀납법을 사용하며, 다음과 같이 정의되는 *실패 수준(failure level*; FL)[17, 22, 25, 31, 33]을 이용한다:

(1) 만약 L_f를 위한 OR 프로세스의 가장 최근 활성화 이후 L_f의 재수행이 없었다고 한다면 FL(L_f)는 0이다.

(2) 위의 경우가 아니라면 FL(L_f)는 $k+1$이다. 여기서 k = MAX { FL(L) | L_f를 위한 OR 프로세스의 가장 최근 활성화 이후, L의 실패가 L_f의 재수행을 '직접'[22, 25] 야기했다 }.

(i) 귀납 기초(induction basis)

FL(L_f)가 0이라고 하자. 그러면 정리 10에 있는 식의 좌변인,

{L | C-MARK(L) $\cap$ B-MARK(L_f) $\neq \emptyset$ 그리고 $L <_l L_f$}

= ∅(B-MARK(L_f) = ∅이므로)

= { L | L ∈ PARENTS(M)이고, M의 실패가 L_f의 재수행을 야기했으며 $L <_l L_f$ }

(그러한 M은 아직 없으므로).

이것은 정리 10에 있는 식의 우변이다. 따라서 기초는 성립되었다.

(ii) 귀납 가정(induction hypothesis)

정리 10이 FL(L_f)가 k 이하인 모든 L_f에 대해 성립한다고 가정하자.

(iii) 귀납 단계(induction step)

FL(L_f)가 k+1이라고 하자. 또한 이러한 L_f를 위한 일반화된 실패 상황을 다음과 같이 가정하자: L_{f1}, L_{f2}, … , L_{fn}이 실패하여 L_f가 백트랙 리터럴로 선택되었다. 따라서 FL(L_{fi})(i = 1, 2, … , n)는 k보다 작거나 같게 된다. 그러면 정리 10에 있는 식의 좌변인,

{L | C-MARK(L) ∩ B-MARK(L_f) ≠ ∅ 그리고 $L <_l L_f$ } = { L | C-MARK(L) ∩ $\bigcup_{i=1}^{n}$ ({m_{Lfi}} ∪ B-MARK(L_{fi})) ≠ ∅ 그리고 $L <_l L_f$}(실패 상황과 우리 백트랙 리터럴 선택 알고리즘의 정의에 의해. 알고리즘 5의 단계 (5)를 참조하라.)

= $\bigcup_{i=1}^{n}$ {L|C-MARK(L) ∩ ({m_{Lfi}} ∪ B-MARK(L_{fi})) ≠ ∅ 그리고 $L <_l L_f$ }(집합 연산의 성질에 의해서)

= $\bigcup_{i=1}^{n}$ ({ L | C-MARK(L) ∩ (B-MARK(L_{fi}) ∪ {m_{Lfi}}) ≠ ∅ 그리고 $L <_l L_f$ } − {L_f})(실패상황의 정의에 의해. L_f가 L_{fi}(i = 1, 2, … , n)를 위한 백트랙 리터럴임을 주의하라. 즉, L_f는 실패한 L_{fi}를 위한 백트랙 후보들 중에서 가장 오른쪽의 리터럴인 것이다.)

= $\bigcup_{i=1}^{n}$ ({ L | L ∈ PARENTS(L_{fi}) ∪ PARENTS(M) 이고, M의 실패가 L_{fi}의 재수행을 야기했으며 $L <_l L_{fi}$ } − {L_f})(귀납 가정에 의해서)

= $\bigcup_{i=1}^{n}$ { L | L ∈ PARENTS(L_{fi}) ∪ PARENTS(M) 이고, M의 실패가 L_{fi}의 재수행을 야기했으며 $L <_l L_{fi}$ }(실패 상황의 정의에 의해서)

={L | L ∈ $\bigcup_{i=1}^{n}$ (PARENTS(L_{fi}) ∪ PARENTS(M) 이고, M의 실패가 L_{fi}의 재수행을 야기했으며 $L <_l L_f$ }(집합 연산의 성질에 의해서)

={L | L ∈ PARENTS(M)이고, M의 실패가 L_f의 재수행을 야기했으며 $L <_l L_f$ }(실패 상황의 정의에 의해).

이것은 정리 10에 있는 식의 우변이다. 따라서 정리 10은 FL(L_f)가 k+1인 L_f에 대해서도 성립한다. □

이제 가장 주요한 정리들을 서술하겠다. 다음 정리는 우리 알고리즘(알고리즘 5)에 의해 구해진 후보 백트랙 리터럴들의 집합이 정리 4에서 정의된 참조 집합과 정확히 같음을 말하고 있다.

정리 11: 세분 마크 집합 방법의 백트랙 리터럴 선택 알고리즘의 정확성

L_f가 실패한 뒤에는 다음 등식이 성립한다:

{L | C-MARK(L) ∩ (B-MARK(L_f) ∪ {m_{Lf}}) ≠ ∅ 그리고 $L <_l L_f$ }

={ L | L ∈ PARENTS(L_f) ∪ PARENTS(M)이고, M의 실패가 L_f의 재수행을 야기했으며 $L <_l L_f$ }.

증명: 정리에 있는 식의 좌변인,

{L | C-MARK(L) ∩ (B-MARK(L_f) ∪ {m_{Lf}}) ≠ ∅ 그리고 $L <_l L_f$}

={L | C-MARK(L) ∩ B-MARK(L_f) ≠ ∅ 그리고 $L <_l L_f$ } ∪ { L | C-MARK(L) ∩ {m_{Lf}} 그리고 $L <_l L_f$ }(집합 연산의 성질에 의해서)

={L | L ∈ PARENTS(M)이고, M의 실패가 L_f의 재수행을 야기했으며 $L <_l L_f$ } ∪ { L | L ∈ PARENTS(L_f)이고 $L <_l L_f$ }(정리 10과 우리의 백트랙 리터럴 선택 알고리즘에 의해서. B-MARK(L_f)가 바뀌지 않았으므로 정리 10이 적용 가능함을 주의하라. 또한 알고리즘 5의 단계 (1)을 참조하라.)

={L | L ∈ PARENTS(L_f) ∪ PARENTS(M)이고, M의 실패가 L_f의 재수행을 야기했으며 $L <_l L_f$ }(집합 연산의 성질에 의해서)

이것은 정리에 있는 식의 우변이다. □

다음 정리는 제안된 알고리즘(알고리즘 6)에 의해 계산된 재초기화될 리터럴들의 집합이 정리 5에 있는 참조 집합과 또한 정확히 일치함을 말하고 있다.

정리 12: 세분 마크 집합의 재초기화 알고리즘의 정확성

L_b와 $L_R{}^R$을 정리 7에서 정의된 바와 같다고 한다. 또한 $L_R{}^K$를 { L | C-MARK(L_b) ∩ B-MARK(L) ≠ ∅ 그리고 $L_b <_l L$ }이라는 집합을 표시한다고 하자. (이 집합은 우리 알고리즘에 의해 구해진 재초기화될 리터럴들의 집합을 나타낸다.) 그러면 L_b가 재수행될 시점에,

$L_R{}^K = L_R{}^R$.

증명: 정리 10의 증명으로부터, 임의의 L_1과 L_2에 대해 "C-MARK(L_1) ∩ B-MARK(L_2) ≠ ∅ 그리고 $L_1 <_l L_2$"라는 진술과 "$L_1 \in$ PARENTS(M)이고, M의 실패가 L_2의 재수행을 야기했으며 $L_1 <_l L_2$"라는 진술은 동치임을 쉽게 알 수 있다. (사실, 이것은 정리 10을 다시 서술한 것일 뿐이다.) 따라서, 임의의 L_b와 L에 대해 "C-MARK(L_b) ∩ B-MARK(L) ≠ ∅ 그리고 $L_b <_l L$"이라는 진술과 "$L_b \in$ PARENTS(M)이고, M의 실패가 L의 재수행을 야기했으며 $L_b <_l L$"이라는 진술은 동치이다(L_1과 L_2를 각각 L_b와 L로 대치한 것이다). 그러면 정리에 있는 식의 좌변인,

{L | C-MARK(L_b) ∩ B-MARK(L) ≠ ∅ 그리고 $L_b <_l L$}

={L | $L_b \in$ PARENTS(M)이고, M의 실패가 L의 재수행을 야기했으며 $L_b <_l L$ }

(앞의 토의에 의해서)

={L | $M \in$ CHILDREN(L_b)이고 M의 실패가 L의 재수행을 야기하는 M이 존재하고, $L_b <_l L$ }

("$L_b \in$ PARENTS(M)"이라는 것은 곧 "$M \in$ CHILDREN(L_b)"임을 고려하여 재서술).

이것은 정리에 있는 식의 (L_b를 M으로 대치한) 우변이다. □

5. 결 론

이 논문에서 우리는 근래에 제안된 마크 집합에 근거한 후방 수행-백트랙 리터럴 선택과 재초기화-알고리즘들, 예컨대 Conery의 새 알고리즘이나 Ng-Leung의 Competition Model 등이 직관적인 설득력이 있음에도 불구하고 틀렸음을 보였다. 또한 그 알고리즘들이 가지고 있는 오류의 원인을 분석하고, 마크 집합과 같은 자료 구조의 잠재적 가능성을 토론한 뒤 유사 자료 구조를 채택한 수정된 백트랙 리터럴 선택 및 재초기화 알고리즘을 제안했다. 제안된 알고리즘의 정확성도 증명했다.

Conery는 자신의 새 알고리즘은 모든 자료 구조가 비트 집합으로 표현되고 관련 연산들이 신속한 비트 연산으로 구현될 수 있기 때문에 효율적임을 주장했다. 간단히 말해서 효율적인 구체적 구현이 가능하다는 것이다. 우리는 제안한 알고리즘이 기존의 마크 집합에 근거한 알고리즘들의 이러한 장점을 모두 가지고 있다는 사실을 지적하고 싶다. 단지 아주 조금의 기억 장소가 더 필요할 뿐이다(Conery-Ng-Leung의 알고리즘에서는 각 생산자 리터럴이 하나의 마크 집합을 가지나 우리 알고리즘에서는 두 개씩의 마크 집합을 가진다). 그러나 마크 집합이 비트 집합임을 생각하면 이 부담은 무시할 만하다고 본다.

Lin과 Kumar[23, 24]은 자신들이 기왕에 제안했던 알고리즘[22, 25]의 비트 벡터(bit vector)를 사용한 구현을 기술한 바 있다. 전방 수행을 위한 그들의 구현 방식은 효율적이라고 여겨진다. 이 논문에서는 전방 수행을 다루지 않았기 때문에, 따라서 그들의 방식을 우리의 후방 수행 방법과 결합시킬 수도 있을 것이다. 후방 수행에 대해서는, 비록 우리들의 방법이 Lin-Kumar의 방법보다 (Conery-Ng-Leung의 방법과의 비교에서와 마찬가지로) 비트 집합을 더 필요로 하나, 두 방법 모두 동일한 생각에 바탕을 두고 있다. 그러나 4.1절에서 언급한 문제로서, Lin-Kumar의 알고리즘도 역시 애시당초 후보 백트랙 리터럴들간의 대칭성을 배제하고 있다. 또한 Lin-Kumar의 알고리즘은 선택적 재초기화도 수행하지 않는다([22]의 부록에서 간단하게 언급하고 있기는 하다). 세분 마크 집합 방법이 선택적 해 재초기화를 지원하게 하거나 재초기화의 선택 행위를 좀더 정교하게 하고 싶은 경우, 자료 구조의 추가는 불가피하다[17]. 따라서 선택적 재초기화를 전혀 수행하지 않는 Lin-Kumar의 방법이 우리 방법보다 적은 기억 장소를 사용하는 것은 일면 당연한 것이다.

이 논문에서 우리가 한 작업을 정리하면 다음과 같다:

(1) 백트랙 리터럴 선택 알고리즘뿐만 아니라[18], 마크 집합에 근거한 재초기화 알고리즘들도 틀렸음을 밝혔다.

(2) 재초기화 알고리즘의 정확성 증명을 위한 참조 집합을 정의했다(정리 5 참조). 우리가 알고 있는 한, 이것과 유사한 정의는 Lin과 Kumar[22]의 논문 외에는 나타난 적이 없다. 그러나 그들의 정의는 증명이 없어 비정형적(informal)이고, 게다가 틀린 것이다. 예를 들어서 정리 9에 있는 반례를 생각해 보자. Lin-Kumar의 정의를 따르자면, $p2$가 재수행되어 성공했을 때는 집합 { L | $p2 <_l L$ 그리고 $p2$의 재수행을 야기한 실패로 인해 L의 어떤 후손이 취소되었다 }에 포함되어 있는 리터럴들을 재초기화해야 한다[22; p. 403; ll. 21-26]. 이 집합에는 $p3$가 포함되지 않는다(왜냐하면 "$p2$의 재수행을 야기한 실패, 즉 $p5$의 실패에 의해 취소된 $p3$의 후손이 없기" 때문이다). 정의가 이러함에도 불구하고, 그들의 선택적 재초기화 알고리즘[22; p.420]은 $p3$를 재초기화시킨다. 이 사실은 그들의 알고리즘이 자신들의 정의에 의해 엄격하게 작성되지 않았음을 뜻한다.

(3) 후방 수행 알고리즘의 완결성과 건전성을 분석하기

위한 체계적 방법론을 제안했다. 만약 이 분야의 연구자들이 Conery-Ng-Leung의 알고리즘을 주의깊게 조사했다면, 다음과 같은 점은 그리 어렵지 않게 발견할 수 있었을 것이다: (i) 그들의 알고리즘은 실패한 리터럴의 정보를 모든 선조들에게 너무 일찍 전달해 버리기 때문에 불건전하다, 그리고 (ii) 그들의 알고리즘은 실패의 간접적 원인을 기록하지 않기 때문에 불완결하다. 따라서 직관에 너무 의존하는 대신에, 후방 수행 알고리즘을 개발한 사람은 누구든지 3절에서 정의된 참조 집합들(정리 4와 5를 참조하라)에 따라 자신의 알고리즘을 점검해 보는 것이 바람직할 것이다.

(4) 마크 집합에 근거한 후방 수행 알고리즘의 오류를 수정하고 그 정확성을 증명했다. 특히, 재초기화가 백트랙 리터럴 선택과 서로 이원적이라는 사실(4.2절 참조)을 반영하는, 우리의 재초기화 알고리즘에 대한 증명(정리 12)은 이런 종류의 증명, 즉 재초기화 알고리즘에 대한 증명 중에서는 우리가 아는 범위 내에서 가장 명쾌한 것이라고 생각한다. 최근까지 논리 프로그램의 병렬 수행을 위한 많은 알고리즘과 모형들이 제안되었다. 앞서 Conery-Ng-Leung의 경우에서 보았듯이, 그 알고리즘이나 모형이 스스로의 잠재적 효율을 주장하고자 한다면 먼저 정확성 증명이 필수적이라고 우리는 생각한다.

앞으로의 연구 과제로는, 4.1절에서 언급했다시피 세분 마크 집합 방법에 바탕을 둔, 내포 루프 모형의 대안을 현재 찾고 있다. 또한 마크와 해를 묶음으로써 선택적 해 재초기화를 수행할 수 있도록 제안된 방법을 확장하려고 한다[19]. 마지막으로는, 이렇게 확장된 모형을 실제 다중처리기(multiprocessor)에서 구현하여 실험적 결과를 획득하는 것이다[21].

참 고 문 헌

[1] M. Bruynooghe and L. M. Pereira, "Deduction Revision by Intelligent Backtracking," *Implementations of Prolog*, J. A. Campbell (Ed.), pp. 194-215, Ellis Horwood, 1984.

[2] J.-H. Chang and A. M. Despain, "Semi-Intelligent Backtracking of Prolog Based on Static Data Dependency Analysis," *Proceedings of the 2nd SLP*, pp. 10-21, 1985.

[3] J.-H. Chang, A. M. Despain, and D. DeGroot, "AND Parallelism of Logic Programs Based on a Static Data Dependency Analysis," *Proceedings of the CompCon Spring*, pp. 218-225, 1985.

[4] K.-M. Choe, M. J. Lee, and N. S. Woo, "Multiple Backtracking in the AND Process of Logic Programming," *Proceedings of TENCON 87 - IEEE Region 10 Conference*, pp. 826-829, 1987.

[5] C. Codognet and P. Codognet, "Non-Deterministic Stream AND-Parallelism Based on Intelligent Backtracking," *Proceedings of the 6th ICLP*, pp. 63-79, 1989.

[6] J. S. Conery, "The AND/OR Process Model for Parallel Interpretation of Logic Programs," *Ph.D. Dissertation*, Department of Information and Computer Science, University of California at Irvine, 1983.

[7] J. S. Conery, *Parallel Execution of Logic Programs*, Kluwer Academic Publishers, 1987.

[8] J. S. Conery, "Implementing Backward Execution in Nondeterministic AND-Parallel Systems," *Proceedings of the 4th ICLP*, pp. 633-653, 1987.

[9] J. S. Conery and D. F Kibler, "AND Parallelism and Nondeterminism in Logic Programs," *New Generation Computing*, Vol. 3, pp. 43-70, 1985.

[10] P. T. Cox and T. Pietrzykowski, "Deduction Plans: A Basis for Intelligent Backtracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-3, pp. 52-65, 1981.

[11] D. DeGroot, "Restricted AND-Parallelism," *Proceedings of the International Conference on Fifth Generation Computer Systems*, pp. 471-482, 1984.

[12] M. V. Hermenegildo and R. I. Nasr, "Efficient Management of Backtracking in AND-Parallelism," *Proceedings of the 3rd ICLP*, pp 40-54, 1986.

[13] L. V. Kale, "The REDUCE-OR Process Model for Parallel Evaluation of Logic Programs," *Proceedings of the 4th ICLP*, pp. 616-632, 1987.

[14] S. Kasif and J. Minker, "The Intelligent Channel: A Scheme for AND/OR Parallelism in Logic Programs," *Technical Report CS-TR-1414*, Department of Computer Science, University of Maryland, 1984.

[15] D.-H. Kim and K.-M. Choe, "Yet Another Efficient Backward Execution Algorithm in the AND/OR Process Model," *Information Processing Letters*, Vol. 40, No. 4, pp. 201-211, 1991

[16] D.-H. Kim and K.-M. Choe, "The AND Process Configuration. A Unified Framework for AND-Parallel Evaluation of Logic Programs," Unpublished manuscript, 1992

[17] D.-H. Kim, K.-M. Choe, B.-M. Chang, and S.-H. Lee, "The Scorer Method: A Scheme for Selective Resetting without Termwise Unification Failure Cause Analysis in the AND-Parallel Evaluation of Logic Programs," *Technical Report CS-TR-90-49*, Department of Computer Science, KAIST, 1990.

[18] D.-H. Kim, K.-M. Choe, and T. Han, "Refined

Mark(s)-Set-Based Backtrack Literal Selection for AND Parallelism in Logic Programs," *Parallel Processing Letters*, Vol. 2, No. 1, pp. 61-69, 1992.
[19] D.-H. Kim, S.-H. Lee, and K.-M. Choe, "Extending the Refined Mark Set Method - Part I: Model," In preparation.
[20] R. Kowalski, *Logic for Problem Solving*, p287, Elsevier North Holland, New York, 1979.
[21] S.-H. Lee, D.-H. Kim, and K.-M. Choe, "Extending the Refined Mark Set Method - Part II: Empirical Study," In preparation.
[22] Y.-J. Lin and V. Kumar, "An Execution Model for Exploiting AND-Parallelism in Logic Programs," *New Generation Computing*, Vol. 5, pp. 393-425, 1988.
[23] Y.-J. Lin and V. Kumar, "AND-Parallel Execution of Logic Programs on a Shared Memory Multiprocessor: A Summary of Results," *Proceedings of the 5th ICLP*, pp. 1123-1141, 1988.
[24] Y.-J. Lin and V. Kumar, "Parallel Execution of Logic Programs on a Shared-Memory Multiprocessor," *Journal of Logic Programming*, Vol. 20, No. 2, pp. 155-178, 1991.
[25] Y.-J. Lin, V. Kumar, and C. Leung, "An Intelligent Backtracking Algorithm for Parallel Execution of Logic Programs," *Proceedings of the 3rd ICLP*, pp. 55-68, 1986.
[26] J. W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, 1986.
[27] K. W. Ng and H. F. Leung, "The Competition Model for Parallel Execution of Logic Programs," *Proceedings of the 5th ICLP*, pp. 1180-1187, 1988.
[28] K. W. Ng and H. F. Leung, "Competition: A Model of AND-Parallel Execution of Logic Programs," *The Computer Journal*, Vol. 33, No. 3, pp. 215-218, 1990.
[29] C.-I. Park, K.-H. Park, and M. Kim, "Efficient Backward Execution in AND/OR Process Model," *Information Processing Letters*, Vol. 29, No. 4, pp. 191-198, 1988.
[30] L. W. Pereira and A. Porto, "Selective Backtracking," *Logic Programming*, K. L. Clark and S.-A. Tarnlund (Eds.), pp. 107-114, Academic Press, 1982.
[31] W. Winsborough, "Semantically Transparent Selective Reset for AND Parallel Interpreters Based on the Origin of Failures," *Proceedings of the 4th SLP*, pp. 134-152, 1987.
[32] N. S. Woo and K.-M. Choe, "Selecting the Backtrack Literal in the AND/OR Process Model," *Proceedings of the 3rd SLP*, pp. 200-210, 1986.
[33] N. S. Woo and K.-M. Choe, "An Intelligent Backtrack Literal Selection in an AND-Parallel Execution of Logic Programs," Unpublished manuscript, 1987.

김 도 형
제 23 권 제 3 호(B) 참조

이 수 현
1987년 광운대학교 전자계산학과 졸업(이학사). 1989년 한국과학기술원 전산학과 졸업(공학석사). 1994년 한국과학기술원 전산학과 졸업(공학박사). 1994년 ~ 1996년 한국전자통신연구소 연수연구원 및 선임연구원. 1996년 ~ 현재 창원대학교 전자계산학과 조교수. 관심분야는 논리 프로그래밍, 컴파일러, WWW의 교육적 활용 등.

최 광 무
제 23 권 제 3 호(B) 참조