Journal of Korea Multimedia Society Vol. 8, No. 11, November 2005(pp. 1556-1564)

패턴 테이블을 이용한 코드 최적화

윤성림[†], 오세만[††]

요 약

원시 프로그램에 대한 컴파일 과정 중 최적화 단계에서는 프로그램의 실행 속도를 개선시키고 코드 크기를 줄일 수 있는 다양한 최적화 기법을 수행한다[17]. 최적화 패턴 매칭 방법 중 스트링 패턴 매칭 방법은 중간 코드에 대응하는 최적의 패턴을 찾기 위한 방법으로 과다한 최적화 패턴 검색 시간으로 비효율적이다. 트리 패턴 매칭은 패턴 결정시 중복 비교가 발생할 수 있으며, 코드의 트리 구성에 많은 비용이 드는 단점을 가지고 있는 방법들이다[16,18]. 본 논문에서는 기존의 최적화 방법들의 단점을 극복하기 위한 방법으로 DFA (Deterministic Finite Automata) 최적화 테이블을 이용한 코드 최적화기를 제안하려고 한다. 이 방법은 다른 패턴 매칭 기법보다 결정적인 오토마타(Automata)로 구성하기 때문에 비용은 적어지고, 오토마타를 통해 결정적으로 패턴이 확정됨에 따른 패턴 선택 비용이 줄어들며, 최적화 패턴 검색 시간도 빨라지는 효율적인 방법의 최적화기이다.

Code Optimization Using Pattern Table

Sung-Lim Yun[†], Se-Man Oh[††]

ABSTRACT

Various optimization techniques are deployed in the compilation process of a source program for improving the program's execution speed and reducing the size of the source code. Of the optimization pattern matching techniques, the string pattern matching technique involves finding an optimal pattern that corresponds to the intermediate code. However, it is deemed inefficient due to excessive time required for optimized pattern search. The tree matching pattern technique can result in many redundant comparisons for pattern determination, and there is also the disadvantage of high cost involved in constructing a code tree. The objective of this paper is to propose a table-driven code optimizer using the DFA(Deterministic Finite Automata) optimization table to overcome the shortcomings of existing optimization techniques. Unlike other techniques, this is an efficient method of implementing an optimizer that is constructed with the deterministic automata, which determines the final pattern, reducing the pattern selection cost and expediting the pattern search process.

Key words: DFA(결정적인 유한 오토마타), Pattern Table(패턴 테이블), Table Driven Optimizer(테이블 구동 최적화기), Code Optimizer Generator(코드 최적화기 생성기), Pattern Matching(패턴 매칭), Pattern Normalization(패턴 정규화)

1. 서 론

원시 프로그램에 대한 컴파일 과정 중 최적화 단계에서는 프로그램의 실행 속도를 개선시키고 코드 크기를 줄일 수 있는 다양한 최적화 기법을 수행한다[2]. 컴파일러 개발 과정에서 최적화 기법의 수행은

※ 교신저자(Corresponding Author) : 윤성림, 주소 : 서울시 중구 필동 3가 26번지(100-715), 전화 : 02)2260-3342, FAX : 02)2265-8742, E-mail : yslhappy@dgu.edu
접수일 : 2005년 6월 8일, 완료일 : 2005년 7월 6일
[†] 동국대학교 대학원 컴퓨터공학과 박사수료
[††] 정회원, 한국정보처리학회 게임연구회 위원장
(E-mail : smoh@dgu.edu)
※ 본 연구는 한국과학재단 목적기초연구(R01-2002-000-00041-0)지원으로 수행되었음.

목적기계 독립적인 중간코드 최적화(Intermediate Code Optimization)와 목적기계 의존적인 목적코드 최적화(Target Code Optimization)로 구분할 수 있다.

또한 최근의 컴파일러 개발에 대한 관심이 다양한 목적기계에 쉽게 적용할 수 있는 코드를 생성하도록 재목적이 가능한 최적화 컴파일러(Retargetable Optimization Compiler)에 있으므로 목적기계와 독립적인 중간코드에서의 최적화는 더욱 필요하다.

특히, 펍홀 최적화(Peephole Optimization)는 최적화에 드는 비용에 비해 큰 효과를 얻을 수 있으며, 그 기법은 비효율적인 명령어의 순서를 구별해 내고 연속되는 명령어의 순서를 의미적으로는 동등하면서 좀 더 효율적인 코드로 개선하는 방법이다. 펍홀 최적화 기법 중에서 주로 다루는 최적화 기법에는 불필요한 명령어의 제거, 도달될 수 없는 코드의 제거, 대수적 간소화, 효율적인 명령어로의 대치 등을 주로 행한다. 요즘 최적화 알고리즘으로는 패턴 매칭 최적화 기법을 많이 사용한다. 패턴과 대치를 쌍으로 기술하여 입력에서 패턴을 발견하면 그에 해당하는 대치를 생성하는 방법이다[2,13,19].

패턴 매칭 방법을 이용한 코드 생성 방법은 양질의 코드 생성을 위해 중간 언어에 대한 분석을 수행한 후 하나 이상의 명령어로 구성된 패턴에 대해 목적 코드를 생성하는 방식이다. 따라서, 패턴 매칭을 이용한 코드 생성 방식에서는 먼저 중간언어의 패턴에 대한 변환될 목적 코드간의 관계가 기술되어 실질적인 코드 변환 시에 참조될 수 있도록 해야 한다. 패턴 매칭 방법 중 스트링 패턴 매칭(String Pattern Matching)은 패턴 결정시에 반복적으로 많은 비교 동작이 수행됨으로 최적화 패턴을 찾는데 걸리는 시간이 많아 비효율적이다. 트리 패턴 매칭(Tree Pattern Matching)은 패턴 결정 과정에서 중복 비교가 발생할 수 있고, 코드의 트리 구성에 많은 비용이 드는 단점들이 있다.

본 논문의 내용은 패턴 기술을 입력 받아 코드 최적화기 생성기에서 DFA 최적화 테이블을 생성한다. 코드 최적화기는 테이블을 이용한 최적화기는 DFA 최적화 테이블을 참조하여 DFA 패턴 매칭 알고리즘을 적용하여 중간 코드를 최적화 하는 것이다.

본 논문의 구성은 2장에서는 본 연구의 기반이 되는 스트링 패턴 매칭, 트리 패턴 매칭에 대한 관련 연구를 소개한다. 3장에서는 기존의 패턴 매칭 방법인 스트링 패턴 매칭, 트리 패턴 매칭들의 단점을 보완하기 위한 새로운 매칭 방법으로 DFA 최적화 테이블을 이용한 코드 최적화기를 기술한다. 4장에서는 코드 최적화기 생성기(Code Optimization Generator)를 기술한다. 마지막으로, 5장에서는 본 연구의 결론과 향후 연구 과제에 대해서 기술한다.

2. 관련 연구

2.1 스트링 패턴 매칭 기법

스트링 패턴 매칭 기법은 중간 코드를 입력 받아 패턴으로 기술된 스트링을 찾아 최적화된 스트링 패턴으로 매칭시키는 방법을 말한다. 다시 말하면, 몇 개의 명령어에 대한 윈도(Window)를 가지고 입력을 검색하여 일련의 비효율적인 코드를 좀 더 효율적인 코드로 대치하는 방법이다.

스트링 패턴 매칭 기법 알고리즘[9]을 살펴보면, 텍스트와 패턴의 각 문자열을 비교한다. 각 문자열이 일치하게 되면 다음의 문자열로 이동하여 비교한다. 만약, 패턴의 우측 마지막 문자열까지 일치하면 패턴 검색은 완료된 것으로 간주한다. 그러나 일치하지 않은 문자열이 발견되면 패턴을 문자열만큼 우측으로 이동시켜 다시 패턴의 시작 문자열부터 다시 비교를 시작한다.

스트링 패턴 매칭 기법의 단점은 패턴 결정 시에 반복적으로 많은 비교 동작(5000번의 리스캔)[17]이 이루어지므로 최적화 패턴을 찾는데 걸리는 시간이 많아 비효율적이다. 스트링 패턴 매칭 기법을 이용하여 ACK 중간 코드인 EM 명령어는 그림 1의 6단계의 과정을 거쳐 수행된다.

2.2 트리 패턴 매칭 기법

트리 패턴 매칭 기법은 중간 코드를 입력 받아 패턴으로 기술된 트리 형태로 재구성이 가능하며, 패턴 분석 및 패턴 결정이 용이하다는 장점을 갖는다. 하향식(Top-Down)으로 이루어지는 패턴 결정 과정에서 중복 비교가 발생할 수 있으며, 코드의 트리 구성에 많은 비용이 드는 단점이 있다.

중간 표현으로써 트리 구조에 기반을 둔 TCOL을 사용하였으며, 목적기계 코드를 생성하기 위해 순환적으로 트리를 순회할 수 있는 알고리즘을 사용하였다. 즉, 트리 패턴의 중간 코드를 입력으로 받아 목적 코드를 생성하는 방법이다[1,8,10].

```
[단계 1]
  ▶EM 코드에 대한 기본 블록 구성
[단계 2]
  ▶연속된 3개의 명령어에 대해 해쉬값 계산
[단계 3]
  ▶첫명령어를 기준으로 패턴 테이블에서 가장 긴 연산자 패턴부터 검색
[단계 4]
  ▶피연산자와 패턴에 대한 피연산자 일치검사
[단계 5]
  ▶제약 조건이 만족되면 대치 패턴으로 변환
[단계 6]
  ▶대치가 수행된 후 매칭 과정이 다시 대치된 첫 문장으로 제어를 옮겨 재수행
```

그림 1. EM 명령어 스트링 패턴 매칭 단계

예를 들어, ACK 중간코드 최적화 알고리즘인 스트링 패턴 매칭 알고리즘 사용으로 인하여 발생되는 과다한 최적화 패턴 검색 시간이 걸리는 문제를 개선하기 위하여 트리 패턴 매칭 방법을 사용하였다.

트리 패턴 매칭의 알고리즘[15]은 중간코드 트리와 최적화 트리 패턴이 구성된 후 트리 패턴 매칭을 시작한다. 중간 코드 트리를 중위운행법으로 운행하면서 최적화 패턴 테이블에서 매칭되는 하위 중간 코드 트리가 존재하는지의 여부를 확인하고 존재할 경우 조건식의 값이 참인지를 확인한다. 이 조건을 만족할 경우에 최적화된 트리로 재구성한다. 그림 2는 이러한 트리 패턴 매칭 알고리즘을 나타낸 것이다.

3. DFA 최적화 테이블을 이용한 코드 최적화기

3.1 시스템 구성도

DFA 최적화 테이블을 이용한 코드 최적화기는 스트링 패턴 매칭의 최적화 패턴 검색 시간이 많이 걸리는 문제와 트리 패턴 매칭의 하향식으로 이루어지는 패턴 결정 과정에서 중복 비교의 발생으로 인한 코드의 트리 구성에 많은 비용이 드는 단점을 보완하기 위한 것이다. DFA를 이용한 코드 최적화는 오토마타를 통해 결정적으로 패턴이 확정됨에 따른 패턴 선택 비용이 줄어들며, 최적화 패턴 검색 시간이 빨라진다.

테이블을 이용한 코드 최적화기 시스템의 전체 구성은 그림 3과 같이 코드 최적화기 생성기(Code Optimizer Generator)와 코드 최적화기(Code Optimizer)로 구성된다. 코드 최적화기 생성기는 중간 코드에 대한 코드 생성 규칙으로 기술한 패턴을 입력 받아 DFA 최적화 테이블을 생성한다. 코드 최적화기 시스템의 구성은 DFA(Deterministic Finite Automata) Optimization Table, Table Driven Optimizer, Read Code, Doubly Linked List, Write Code로 구성

```
match_treepattern(node_ptr ppar, node_ptr pnode, bool flag)
/* pparparent 중간코드node 포인터
pnode매칭할 중간코드node 포인터
flagpnode가 ppar의 자노드/제노드 여부
*/
{
if (pnode != NULL)
  if (매칭되는 하위 tree 존재)
   트리를 재구성.
   pnode를 해당 위치로 변경.
match_treepattern(pnode, pnode->son, true);
match_treepattern(pnode, pnode->brother, flase);
}
```

그림 2. 트리 패턴 매칭 알고리즘

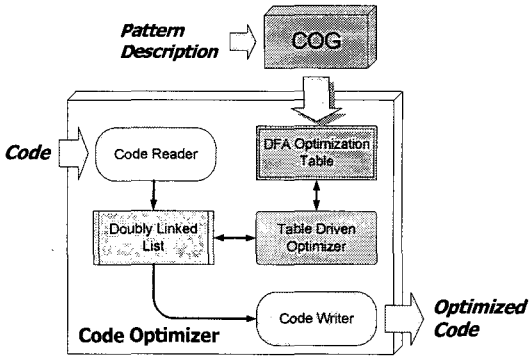

그림 3. 코드 최적화기 구성도

되며, 각 모듈에 대한 내용을 살펴보도록 한다.

DFA 최적화 테이블은 일련의 상태들과 코드에 대한 명령어들의 집합으로 이루어진다. 존재하는 패턴 DFA 형태와 매핑을 시도하여 이에 부합하는 패턴에 대하여 SIL(Standard Intermediate Language)를 최적화된 SIL DFA로 대치하는 것을 결정한다. 테이블을 이용한 최적화기(Table Driven Optimizer)는 DFA 최적화 테이블에서 결정된 패턴 DFA의 모든 정보를 참조한다. DFA의 arc의 이름(name)은 sil 코드명을 사용하며, sil 코드를 보고 트랜잭션(transaction)이 일어나게 된다. 코드 최적화기는 중간코드인 *.sil 코드를 라인 단위로 입력 받아 코드 최적화기 생성기에 의해 생성된 DFA 최적화 테이블 정보를 참조하여 실질적으로 최적화된 *.sil 코드를 생성한다. 각 패턴들의 모든 정보는 오토마타 형태의 정보를 가지고 있으며 *.sil 코드와 더불어 코드 최적화기(Code Optimizer)에 의해 DFA를 이용하여 중간코드를 매칭 할 수 있도록 한다. 입력 코드와 출력코드는 코드 최적화기의 자료구조인 이중 연결 리스트(Doubly Linked List)는 테이블을 이용한 최적화기에 의해 결정된 패턴을 자료구조에 맞게 저장한다. 입력코드를 읽어 들여 연결 리스트 안에 저장된 내용과 비교하여 출력코드를 생성한다.

3.2 패턴 기술

SIL 코드의 특성을 고려하여 작성된 패턴 기술(Pattern Description)을 통해 DFA 최적화 테이블을 생성한다. 패턴 기술은 대치부(Replace Part) ::= 패턴부(Pattern Part)의 쌍으로 구성된다. 대치부와 패턴부를 구분하기 위한 구분자는 '::='로 표시한다. 각각의 SIL 코드를 구분하기 위한 구분자로 '/'을 이용하여 표시한다.

대치부는 패턴에 대해 최적화된 명령어로 변환되는 부분을 나타낸다. 패턴부는 코드 최적화기에 의해 인식될 수 있는 최적화되지 않은 명령어의 열을 나타낸다. 기술된 패턴으로부터 COG를 이용하여 패턴 기술의 내용을 파싱하고 (대치부 ::= 패턴부) 쌍으로 구성된 입력을 받아들여 DFA 최적화 테이블을 출력한다. 그림 4는 최적화에 사용된 SIL 코드 패턴 중에서 상수, 스택, 배정, 제어에 관련된 패턴을 보여주고 있다.

```
// 상수
  ldc.i.m1  ::= ldc.i $11 / neg

// 스택
  ldl dup $21 ::= ldl $22 / ldl $23 {$21 = $22 $23}

// 배정
  inc $31 $32 ::= ldl $33 / ldc.i $34 / add {$31 = $33, $32 = $34}
  dec $41 $42 ::= ldl $43 / ldc.i $44 / sub {$41 = $43, $42 = $44}

// 제어
  bge $51 ::= lt / brf $52 {$51 = $52}
  ble $61 ::= gt / brf $62 {$61 = $62}
```

그림 4. 패턴에 반영된 최적화 내용

3.3 DFA 최적화 테이블의 Action

DFA 최적화 테이블의 Action을 정의하기 위해 테이블에서 사용될 심벌들을 정의하면 그림 5로 나타낸다. 심벌의 구성은 상태(State), 명령코드 집합(Instruction Code-Set), DFA 패턴 테이블로 구분된다.

```
Symbols
  ▶State(S)
    Ss : start state
    Sc : current state
    SF : set of final state

  ▶Instruction Code-Set (I)
    IP : start instruction code-set of pattern
    IC : current instruction code-set
    IR : set of replace instruction code-set

  ▶DFA Table
    DFATBL[S, I] : DFA optimization table
```

그림 5. DFA Action의 Symbols정의

상태의 종류는 시작상태, 현재상태, 종결상태로 구분된다. 명령어의 종류는 패턴의 시작 명령어, 현재 명령어, 대치명령어로 구분된다. 명령코드 집합의 형식은 (instruction, parameter)로 표시된다.

DFA 최적화 테이블의 각 상태들과 명령어들이 일치하는 부분에서 shift, replace, advance 행동에 의해 테이블이 작성된다. 최적화 테이블의 엔트리에서 s는 shift, r은 replace로 나타낸다. Shift 행동은 현재 입력 심벌에 매칭되는 상태이며, DFA를 선택하여 매칭된 상태로 이동하는 것을 의미한다. Replace 행동은 이동한 현재 상태가 종결상태인 경우를 의미한다. 마지막으로, Advance 행동은 현재 입력 명령어에 매칭된 상태가 없는 경우를 의미한다. DFA 최적화 테이블은 정의된 심벌로 3개의 Action을 그림 6으로 정의한다.

```
Initialize
    SC ← SS
    IP ← IC ←I0
Shift
    SC ←DFATBL[SS, IC ]
    IC ← IC +1
Replace
    SC ←SS
    IP ← IC ← IRi
Advance: DFATBL[SS, IC]
    SC ← SS
    IP ← IC ← IP+1
```

그림 6. DFA 최적화 테이블의 Action

3.4 테이블을 이용한 코드 최적화기

패턴 기술을 입력 받아 COG에서 DFA 최적화 테이블을 생성한다. 테이블을 이용한 최적화기는 패턴으로 기술된 DFA 형태의 정보를 참조하여 최적화된 DFA 형태로 결정한다. 테이블을 이용한 최적화기의 DFA 패턴 매칭 알고리즘은 그림 7로 기술한다.

```
Algorithm dfa_pattern matching
begin
    SC ← SS
    IP ← IC ← I0

    for IP ∈ I
      if DFATBL[SC, IC] ∈ S then   // shift
          SC ← DFATBL[SC, IC]
          IC ← IC+1
      else if DFATBL[SC, IC] ∈ SF then  //replace
          SC ← DFATBL[SC, IC]
          IC ← IC+1
          IP ← IC ← IRi
       else     // advance
          SC ← SS
          IP ← IC ← IP+1
       fi
    end for
end.
```

그림 7. DFA 패턴 매칭 알고리즘

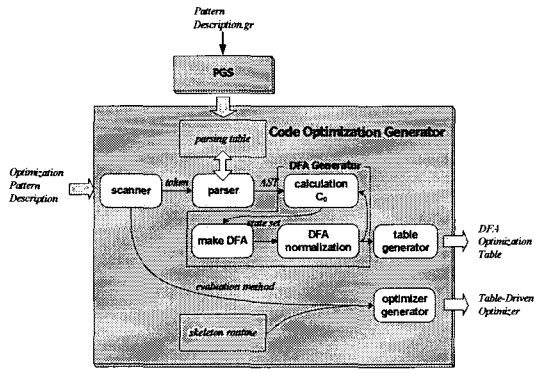

그림 8. 코드 최적화기 생성기 구성도

4. 코드 최적화기 생성기

4.1 시스템 구성도

코드 최적화기 생성기는 최적화 패턴 형식(Pattern.gr)을 입력받아 그림 8과 같은 처리과정을 거쳐 DFA 최적화 테이블로 출력된다. 코드 최적화기 생성기 구조는 스캐너(Scanner), 파서(Parser), DFA 생성기(DFA Generator), 테이블 생성기(Table Generator), 최적화기 생성기(Optimizer Generator)으로 구성된다. DFA 생성기의 구성은 C_0계산(Caculation C_0), DFA 생성(Make DFA), DFA 정규화(DFA Normalization)로 구성된다.

스캐너는 최적화 패턴 형식을 입력받아, 문법적으로 의미 있는 최소단위인 일련의 토큰을 생성하는 일을 한다. 파서는 패턴 문법을 통해 얻어진 파싱 테이블을 이용하여 패턴 형식을 구문 분석하고, 그 결과로 AST를 생성한다. DFA 생성기(DFA Generator)는 생성된 AST를 이용하여 패턴을 DFA 형태로 변환하고, 정규화 과정을 거쳐 DFA 최적화 테이블

에 사용 가능한 상태 집합(State Set)으로 구성한다. 테이블 생성기는 상태 집합을 입력으로 받아 테이블을 이용한 최적화기에서 사용가능한 자료구조(2차원 배열)로 변환한다. 최적화기 생성기는 shift, replace, advance 3개의 Action 정보를 가지고 있는 스켈레톤 루틴과 패턴 기술에서 얻어진 evaluation_method를 결합하여 테이블을 이용한 최적화기를 생성한다.

4.2 패턴 형식(Pattern.gr) 기술

패턴의 기술은 패턴 기술(Pattern Description)의 최적화 패턴 형식(pattern.gr)은 그림 9의 EBNF 형식으로 대치 부분과 패턴 부분으로 구성된다. 대치 부분과 패턴 부분에 대한 길이는 제한이 없으며, 하나의 패턴은 한 라인으로 구성된다.

패턴 기술은 명령어 코드 집합들의 대치 부분과 패턴 부분을 구분하기 위하여 구분자 '::='로 기술한다. 패턴 부분의 명령어 코드 집합과 eval_method로 기술한다. 명령어 코드 집합은 명령어와 파라메터 그리고 구분자 '/'로 기술한다.

4.3 DFA 최적화 테이블

패턴 기술을 입력받아 COG에 의하여 DFA Optimization Table이 생성된다. 이 테이블에서는 기술된 각 패턴의 명령어들에 대한 shift, replace, advance 3가지 Action들로 테이블을 구성한다. 그림 10은 여러 가지 패턴들을 DFA로 상태 전이도를 나타내며, 그 형태를 보고 DFA 최적화 테이블은 표 1로 도시화 할 수 있다.

DFA 최적화 테이블은 한 상태와 입력 심벌(명령어)을 보고 다른 상태로 이동하는 것을 나타낸다. 상태 0에서 코드 ldl을 보고 상태 1로 shift한다. 다시

```
pattern_description ::= instruction_code_sets '::=' instruction_code_sets '%%'
                        '%eval_methods' '%%' ₁
instruction_code_sets ::= '%instruction' parameters '/' ₁
parameters ::= '%param' ₁
```

그림 9. 패턴 형식

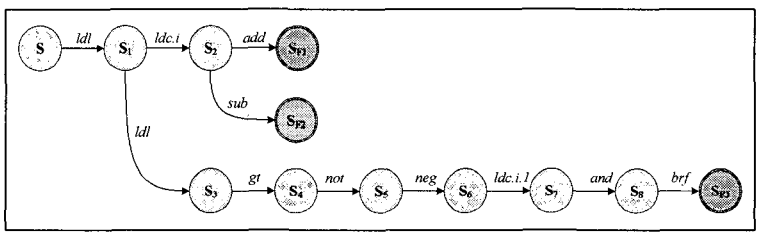

그림 10. 패턴 예제

표 1. DFA 최적화 테이블

codes / states	ldl	ldc.i	add	sub	ldl	gt	not	neg	ldc.i.1	and	brf
0	s1										
1		s2			s3						
2			r1	r2							
3						s4					
4							s5				
5								s6			
6									s7		
7										s8	
8											r3

상태 1에서 코드 ldl을 보고 상태 3로 shift한다. 이와 같은 방법을 반복하여 상태 8에서코드 brf를 보고 replace를 한다.

4.4 패턴 정규화(Pattern Normalization)

패턴 정규화는 DFA를 이용한 패턴 검색시 발생하는 중복 비교를 제거하기 위한 과정으로, 패턴 형태의 단순화 및 구조를 개선하고자 하는 일련의 과정을 의미한다. 정규화 이유는 DFA 패턴 검색시 처리가 간결하고 해당 패턴을 빠르게 찾는데 있다.

패턴 정규화 DFA 단위는 상태들은 원으로 표시하고, 상태전이는 arc로 구성된다. DFA 단위는 그림 11을 순서대로 설명하면, 원 안에 S가 있는 경우는 시작 상태이며, S가 없는 경우는 정상 상태, 종결 상태, 조건적인 종결 상태로 구분된다. 원 안에 실선인 상태는 종결 상태이고, 점선으로 표시된 상태는 조건적인 종결 상태로 구분한다. 화살표는 arc로 표시한다.

정규화의 조건은 1개의 가지(branch)에는 반드시 종결 상태가 1개만 존재하며, 종결상태는 항상 가지의 마지막 상태인 조건을 가진다. 그림 12는 패턴의 정규화된 형태이다.

정규화를 하는 목적은 그림 13에 나타낸 형식처럼 정규화 조건에 맞지 않는 패턴을 정규화된 형식으로 만들기 위한 것이다. 다시 말하면, 그림 12처럼 각 패턴은 1개의 가지에 반드시 종결 상태가 1개만 존재하며, 종결 상태는 항상 가지(Branch)의 마지막 상태로 표현된다.

패턴 정규화를 위한 방법으로 첫 번째 패턴 분할,

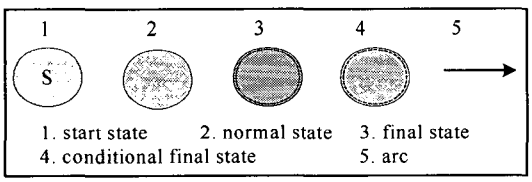

그림 11. 패턴 DFA의 단위

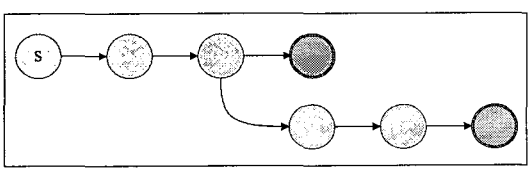

그림 12. 패턴의 정규화된 형태

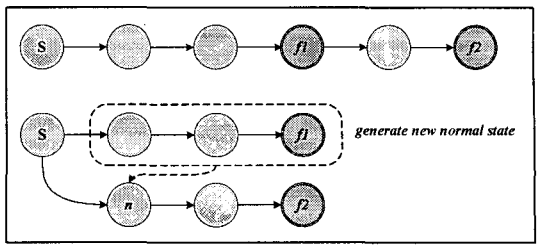

그림 13. 패턴 분할 메소드

두 번째 조건에 따라 종결 상태 분할, 마지막으로 종결상태의 의존 여부에 따라 분할로 체크한다.

첫 번째 방법으로 패턴 분할은 그림 13으로 나타내며, 종결 상태 f1과 종결 상태 f2를 두개의 패턴으로 분할한다. 종결 상태 f1까지 패턴을 새로운 패턴 n 상태인 정상 상태로 생성한다.

두 번째 방법으로 조건에 따라 종결 상태 분할은 그림 14로 나타낸다. 조건에 따라 종결 상태가 결정되며, 그 조건에 맞는 evaluation method를 적용한다. 조건 종결 상태 cf1이 종결상태인 경우와 종결 상태가 아닌 경우로 결정하여 패턴을 분할한다. 점선으로 표시된 상태들을 새로운 상태 n으로 대치하여 새로운 패턴을 만든다. 즉, 하나의 패턴 중에 조건 상태가 있는 경우는 그 조건 결정에 따라 패턴 2개로 분할된다.

마지막 방법으로는 종결상태의 의존 여부에 따라 분할로 체크는 그림 15로 나타낸다.

그림 15에서 샘플 1은 종결 상태 f2가 종결 상태 f1에 의존하는 경우는 패턴을 분할하여 새로운 상태 n을 생성한다. 종결 상태 f3은 의존하고 있지 않기 때문에 원래 패턴을 적용한다. 샘플 2는 종결상태 f2와 f3가 종결 상태 f1에 모두 의존하는 경우는 종결 상태 f1까지 패턴을 분할해서 새로운 normal 상태 n을 생성하여 패턴을 재정의한다.

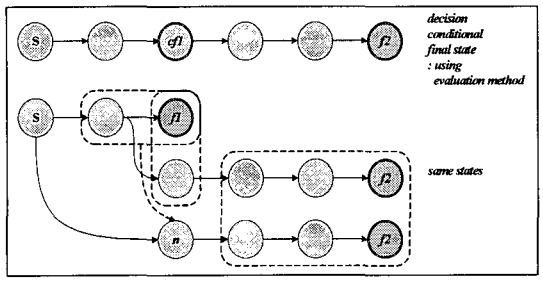

그림 14. 조건에 따라 종결 상태 분할

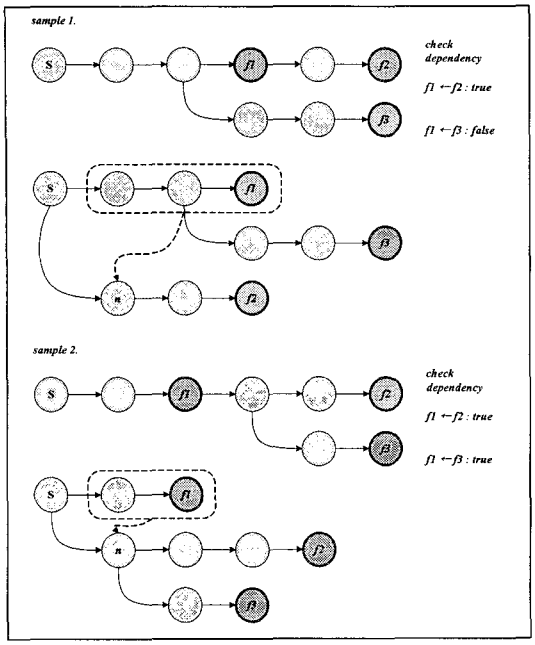

그림 15. 종결상태의 의존 여부에 따라 분할

5. 결론 및 향후 연구

본 논문에서는 기존의 최적화 기법들을 분석하고 기법들의 단점을 해결하기 위한 방안으로 DFA를 이용한 패턴 매칭 방법으로 테이블을 이용한 최적화기를 제안하였다.

테이블을 이용한 최적화기는 패턴 기술을 입력 받아 COG에 의해 기존 패턴을 새로운 DFA 패턴 형태로 변경해 주는 테이블을 생성한다. 테이블을 이용한 최적화기는 DFA 최적화 테이블의 내용을 참조하여 코드 최적화한다.

최적화기에 적용된 알고리즘은 중간코드를 오토마타로 구성하기 때문에 다른 자료구조로의 변환이 불필요하다. 다시 말해, 다른 패턴 매칭 기법보다 구성 비용이 적어지고, 오토마타를 통해 결정적으로 패턴이 확정됨에 따른 패턴 선택 비용이 절감될 것으로 생각된다. 또한, 패턴 정규화는 발생하는 중복 비교를 제거하여 패턴 검색시 해당 패턴을 빠르게 찾을 수 있다.

향후 연구 과제로는 기존 패턴 매칭 기법들의 문제점을 보완하기 위한 방법으로 제안한 테이블을 이용한 최적화기의 구현과 실제로 각 기법을 비교 분석할 수 있는 성능평가가 있어야 하며, 코드 최적화에 대한 지속적인 연구가 필요하겠다.

참 고 문 헌

[1] Alfred V. Aho, Mahadevan Ganapathi, and Steven W. K. Tjiang, "Code Generation Using Tree Matching and Dynamic Programming," *ACM TOPLAS*, Vol. 11, No. 4., pp. 491-516, Oct. 1989.

[2] Andrew S. Tanenbaum, Hans van Staveren, and Johan W. Stevenson, "Using Peephole Optimization Intermediate Code," *ACM TOPLAS*, Vol. 4, No. 1, pp. 21-36, Jan. 1982.

[3] Christoph M. Hoffmann and Michael J. O'Donnell, "Pattern Matching in Trees," *Journal of the ACM*, Vol. 29, No. 1, pp. 68-95, Jan. 1982.

[4] Christopher W. Fraser and Todd A. Proebsting, "Finite-State Code Generation," *ACM SIGPLAN*, Vol. 34, No. 5, pp. 270-280, May 1999.

[5] Fabrice Le Fessant and Luc Maranget, "Optimizing Pattern Matching," *ACM SIGPLAN*, Vol. 36, No. 10, pp. 26-37, Oct. 2001.

[6] Han Kesong, Wang Yongchen, and Chen Guilin, "Research on a faster algorithm for pattern matching," *Proceedings of the fifth international workshop on Information retrieval with Asian languages*, pp. 119-124, Nov. 2000.

[7] Patrick A. V. Hall and Geoff R. Dowling, "A fast string searching algorithm," *ACM Computing Surveys (CSUR)*, Vol. 12, No. 4, pp. 762-772, Oct. 1977.

[8] R.G.G. Cattell, "Automatic Derivation of Code Generators from Machine Descriptions," *ACM TOPLAS*, Vol. 2, No. 2, pp. 171-190, Apr. 1980.

[9] Richard Cole and Ramesh Hariharan, "Approximate string matching: a simpler faster algorithm," *ACM-SIAM*, pp. 463-472, Jan. 1998.

[10] Susan L. Graham, "Table-Driven Code Generation," *IEEE Computer*, Vol. 13, No. 8, pp.

25-34, Aug. 1980.

[11] Todd A. Proebsting, "BURS automata generation," *ACM TOPLAS*, Vol. 17, No. 3, pp. 461-486, May 1995.

[12] U. Vishkin, "Deterministic sampling－a new technique for fast pattern matching," *Annual ACM Symposium on Theory of Computing*, pp. 170-180, May 1990.

[13] W. M. Mckeeman, "Peephole Optimization," *Communication of the CACM*, Vol. 8, No. 7, pp. 434-444, July 1965.

[14] Zvi Galil, "On improving the worst case running time of the Boyer-Moore string matching algorithm," Vol. 22, No. 9, pp. 505-508, 1979.

[15] 고광만, 김정숙, 오세만, "트리 패턴 매칭 기법을 이용한 코드 생성 알고리즘의 개발," 한국정보과학회 봄 학술발표논문집, 제22권 제1호, pp. 992-996, 1995.

[16] 김정숙, 고광만, 김성철, 한용희, 오세만, "Tree Pattern Matching을 이용한 펍홀 최적화기의 설계 및 구현," 한국정보과학회 프로그래밍언어연구회지 제8권 제2호, pp. 5-15, 1997.

[17] 김정숙, 트리패턴매칭기법을 이용한 중간코드 최적화 시스템의 설계 및 구현, 동국대학교 박사학위 논문, 1998.

[18] 김정숙, 오세만, "트리 패턴 매칭 최적화," 동국논총 34집, 자연과학편, pp. 253-278, 동국대학교, 1995.

[19] 오세만, 컴파일러 입문 개정판, 정익사, 서울, 2004.

[20] 윤성림, 오세만, "DFA를 이용한 코드 최적화," 한국정보처리학회 춘계학술발표논문집, 제12권, 제1호, pp. 523-526, 2005.

윤 성 림

1998년 한국방송통신대학교 전자계산학과(학사)
2000년 동국대학교 교육대학원 컴퓨터교육(교육학석사)
2002년～현재 동국대학교 일반대학원 컴퓨터공학과 박사수료
관심분야 : 컴파일러, 프로그래밍 언어

오 세 만

1985년 3월～현재 동국대학교 컴퓨터공학과 교수
1993년 3월～1999년 2월 동국대학교 컴퓨터공학과 대학원 학과장
2002년 11월～2003년 11월 한국정보과학회 프로그래밍 언어연구회 위원장
2004년 11월～현재 한국정보처리학회 게임연구회 위원장
관심분야 : 컴파일러, 프로그래밍 언어, 모바일 컴퓨팅