스택-기반 코드로부터 분석을 위한 CFG 생성기의 구현

김영국 유원희
인하대학교 컴퓨터 정보공학과
ykkims@pelab.inha.ac.kr

Implementation of the CFG generator for the analysis from The stack-based codes

Young Kook Kim[0] Weon Hee Yoo[O]
[O]Department of Computer Science & Information Inha University

요 약

자바의 문제점은 실행속도의 저하이다. 바이트코드 최적화 방법을 사용하는 CTOC(Class To Optimized Classes)에서 중간코드로 사용하는 3-주소 코드를 스택-기반 코드로 코드 확장 기법으로 변환 시 불필요한 코드가 생성된다.

이러한 불필요한 코드를 제거하기 위한 정보를 필요로 한다. 필요한 정보를 얻기 위한 분석기로 CFG생성기를 설계 및 구현한다.

1. 서론

자바는 많은 장점에도 불구하고 실행속도의 문제점을 갖는다. 이러한 실행속도의 문제점을 해결하기 위한 방법으로 아래와 같은 방법이 연구 중에 있다.

- 클래스 파일을 실행코드로 변환[1,2,3]
- JIT 컴파일 방법[4]
- 바이트코드 최적화[5,6]

위와 같은 방법중 바이트코드 최적화의 장점을 사용하는 CTOC에서 3-주소 코드에서 스택-기반 코드로 변환 시 생기는 불필요한 코드가 생성된다. 생성된 불필요한 코드를 제거하기 위한 정보를 얻기 위해 CFG 생성기를 생성 및 구현한다[7].

본 논문의 구성은 2장에서 코드 확장 기법, 그리고 기본블록에 대해서 서술한다. 3장에서는 CFG생성기에 대해서 논하고, 마지막으로 4장에서는 결론과 향후 연구 방향에 대해서 기술한다.

2. 관련연구

2.1 코드 확장 기법

코드 확장 기법은 코드 추출기, 코드 변환기, 코드 변환 테이블로 구성된다. 각 구성에서 하는 역할은 다음과 같다.

첫째, 코드 추출기는 텍스트 파일을 입력으로 받아 CTOC-T 코드를 추출한다.

둘째, 코드 변환기는 코드 변환 테이블을 참조하여 코드에 해당하는 변환된 코드를 생성한다.

셋째, 코드 변환 테이블은 코드 변환에는 중간코드 변환에 대한 실질적인 정보를 저장하는 부분으로서 3-주소 코드의 특성을 고려하여 유사한 기능을 갖는 명령어 그룹을 구성되어 있다. 명령어 그룹의 간단한 예는 [표 1]과 같다.

[표 1]에서 CTOC-B코드 명령어 그룹에서 표시하는 *t* 는 타입을 표현하는데 사용된다. 명령어 그룹을 이용하여 변환되는 내용은 다음과 같다.

a = b % 10t	load.i b push 10 rem.i store.i a
(a)CTOC-T	(b)CTOC-B

[표 1] 명령어 그룹의 예

result = local % constant	load.*t* local push constant rem.*t* store.*t* result
local := @parameter constant	local := @parameter constant
if local != constant goto label n	load.*t* local push constant ifcmpne.*t* label n label n:
specialinvoke immediate. identifier (type_list): type > (immediate_list);	specialinvoke < identifier (type_list): type >;
(a) CTOC-T	(b) CTOC-B

그러나 코드 확장 기법은 빠르게 변환이 가능하다는 장점을 갖지만 생성된 코드의 질이 효율성이 떨어질 수 있으므로 최적화 동작이 수행 되어야한다는 단점을 갖는다[7].

그래서 본 논문은 코드의 질과 효율성을 보완할 수 있는 정보를 분석이 필요하다. 정보 분석을 위한 기본 자료 분석을 위한 CFG를 구성한다.

2.2 기본 블록

코드 확장 기법을 이용하여 CTOC-T코드를 CTOC-B코드로 변환한다. CTOC-B코드를 이용하여 기본 블록을 구성한다. 기본 블록을 구성하는 알고리즘은 [그림 1]와 같다.

"이 논문은 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임"(R05-2004-000-11694-0)

```
Basicblock_Construction() return set of basicblock
    bblist: set of basicblock
    stat: set of statement
    str: set of statement
    s: statement
    name : String
    change := true: boolean
begin
        repeat
            change := false
            for each s ∈ stat do
                if(change == true)
                    name = methodname(s); change := false;
                    str = {};
                fi
                else if(isBranch(s))
                        change := true; type = branchType(s);
                        bblist.add(newBasic(str, name, type));
                fi esle
                else if(isClass(s))
                        change := true; type = "class"
                        bblist.add(newBasic(str,name,type));
                fi esle
                str.add(s);
            od
        until !change
        return bblist;
end || basicConstruction()
```

[그림 1] 기본 블록 알고리즘

[그림 1]의 알고리즘에서 볼드체로 표시된 부분이 실제 기본 블록을 나누어 주는 부분이다. 여기서 isBranch()는 분기문을 구분해 주는 역할을 한다. 각 분기문의 명령어는 [표 2]와 같다.

[표 2] 분기문의 종류

조건 분기문	ifcmpeq, ifcmpne, ifcmpge, ifcmpgt, ifcmple, ifcmpgt
반환 분기문	return, ret
호출 분기문	virtualinvoke, staticinvoke, specialinvoke, interfaceinvoke
무조건 분기문	goto

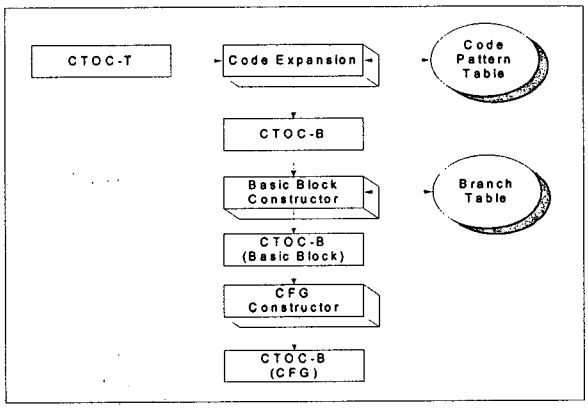
branchType()의 경우는 현재 명령어의 분기 타입을 알려주는 역할을 한다. 각 분기 타입은 return, sequence, condition, goto, invoke등으로 나누어진다. 이렇게 생성된 기본 블록에는 다음과 같은 정보 필드를 포함한다.

```
public class BasicBlock {
   public int first,end;       //연속된 노드 번호
   public int basicnum;        //기본 블록 번호
   public String label;        //레이블
   public String type;                 //분기 타입
   public String gotolabel;    //분기 레이블
   public String methodname;           //메소드 이름
   public Statement st;        //문장 노드 리스트
   public int[] predecessor;//선행 기본 블록의 리스트
   public int[] successor;     //후행 기본 블록의 리스트
   public HashSet visitor;     //preorder list
   public HashSet dominator;   //dominator list
}
```

기본 블록을 생성할 때 생성하는 정보는 문장 노드리스트까지의 정보를 생성하고, 나머지 정보는 CFG구성 이후 정보를 생성하게 된다.

3. CFG 생성기의 구현

CFG 생성기는 코드 확장기, 코드 패턴 테이블, 기본 블록 구성기, 분기 테이블, CFG 구성기로 이루어진다. 코드 확장기는 CTOC-T파일을 입력으로 받아서 코드 패턴 테이블에서 정의한 명령어 그룹을 이용하여 기본 블록을 구성한다. CFG 구성기는 기본 블록을 이용하여 CFG를 구성한다. 전체 구조도는 [그림 2]와 같다.



[그림 2] CFG 생성기의 구조도

[그림 2]의 구성 중 CFG 구성기는 그래프의 노드와 간선을 필요로 한다. 노드는 기본 블록을 이용한다. 그리고 간선은 현재 블록의 후행 기본 블록의 리스트와 선행 기본 블록의 리스트를 이용해서 나타낸다.

그러나 간선을 나타내는 선행 기본 블록 리스트를 먼저 계산하지 않는 이유는 이전 기본 블록의 리스트를 계산하기 힘들기 때문이다. 따라서 우선 이후 기본 블록 리스트를 구하기 위해서는 기본 블록에서 분기 타입 정보가 필요하다. 분기 타입 정보는 기본 블록의 현재 기본 블록의 이전 기본 블록의 개수는 임의로 증가하므로 바로 구할 수 없다. 따라서 이전 블록의 리스트를 구하기 위한 방법으로 기본 블록의 분기 타입에 따라 계산이 가능한 현재 블록의 이후 블록 리스트를 구한 후, 이후 기본 블록 리스트를 이용하여 이전 기본 블록 리스트를 구성할 수 있다.

CFG 구성 방식을 이용해서 전체 파일에 대한 CFG를 구성할 수 있다. 대부분의 CFG는 메소드 단위로 생성된다. 그러나 본 CFG 생성기는 전체 파일에 대해서 CFG를 생성한다. 생성된 CFG는 메소드 단위로 만들어지는 CFG에 비하여 메소드간에 정보를 추가로 얻어낼 수 있다. 이러한 CFG 생성기의 알고리즘은 [그림 3]와 같다.

[그림 3]의 알고리즘에서 볼드체로 isType()은 이용하여 각 블록의 타입을 구분해 낸다. 식별된 타입을 이용하여 분기의 방법이 다르게 적용된다. goto는 하나의

후행 기본 블록 리스트만을 갖는다. 따라서 분기 레이블을 가지고 있는 기본 블록의 위치를 searchLabel() 함수를 이용하여 찾아낸다. 기본 블록의 위치를 후행 기본 블록 리스트에 추가함으로 후행 기본 블록 리스트를 마친다.

```
CFG_Construction() return set of basicblock
    bblist : set of basicblock
    bb : basicblock
    pre : basicblock

begin
    //Successor Construction
    for each bb ∈ bblist do
        if(isType(bb) == "return") continue; fi
        else if(isType(bb) == "goto")
        bb.addSuccessor(isType(bb),searchLabel(bb, bblist));
        fi esle
        else if(isType(bb) == "invoke"|isType(bb)
=="sequence")
            bb.addSuccessor(isType(bb),searchFirst(bb,
bblist));
        fi esle
        else if(isType(bb) == "condition")
            bb.addSuccessor(isType(bb),searchLabel(bb,
bblist));
            bb.addSuccessor(isType(bb),searchFirst(bb,
bblist));
        fi esle
    od

    //Predecessor Construction
    for each bb ∈ bblist do
        for pre ∈ searchSuccessor(bb, bblist) do
        bb.addPredecessor(pre)
        od
    od
end
```

[그림 3] CFG 구성 알고리즘

4. 결론 및 향후 연구

자바는 실행속도 저하라는 문제점의 해결 방법으로 클래스 파일을 실행코드로 변환, JIT 컴파일 방법, 바이트코드 최적화의 방법이 연구 중에 있다. 이러한 방법 중 CTOC라는 바이트코드 최적화 프레임워크가 존재한다.

그러나 CTOC는 CTOC-T에서 CTOC-B로 변환 시 코드 확장 기법을 사용한다. 코드 확장 기법은 빠르게 코드 변환을 할 수 있지만 생성된 코드의 질이 떨어진다. 따라서 코드 질 향상을 위해서 최적화하기 위한 정보 분석이 요구된다. 정보 분석을 위한 CFG 생성기를 구현한다.

CFG 생성기를 구현하기 위해서는 기본 블록을 구성할 수 있는 기본 블록 구성기와 분기 테이블을 필요로 한다. 기본 블록 구성기는 분기 테이블에서 분기 명령어와 분기 타입에 대한 정보를 이용하여 각 기본 블록에 정보 추가한다. 추가된 정보는 그래프의 노드인 기본 블록 간의 관계를 나타내는 간선을 구하는데 사용하게 된다. 간선은 선행 기본 블록 리스트와 후행 기본 블록 리스트를 통해서 표현한다. 이러한 간선을 나타내는 이전/이후 기본 블록 리스트를 구하기 위해서 각 기본 블록의 분기 타입을 이용한다. 분기 타입을 이용해서 전체 파일에 대한 기본 블록간의 관계를 나타내는 CFG를 생성해낸다.

향후 연구 과제로는 CFG가 그래프이기 때문에 그래프의 노드를 한번만 방문하고 순서를 정하기 위해서 DFST를 구성한다. DFST를 구성 후 지배자 트리와 지배자 경계를 계산하고 최적화를 적용하는 것이다.

참고문헌

[1] Ronald Veldema, "JCC, a native Java compiler ", Technical report, 1998

[2] Todd A. Proebsting, Greg Townsend, Patrick Bridges, "Toba: Java For Applications A Way Ahead of Time(WAT) Compiler", COOTS97, pp. 41-53, 1997

[3] A. Krall and R. Grafl, "CACAO - A 64 bit Java VM Just-in-time Compiler", Appeared at PPoPP'97 Workshop on Java for Science and Engineering Computation, 1997

[4] John Meyer, Troy Downing, "Java Virtual Machine", O' RELLAY, 1997

[5] preEmptive Solutions, "DashO Whitepaper", http://preemptive.com/downloads/documentation.html, 2002-2004

[6] Geoff Cohen (Duke/IBM), Jeff Chase (Duke), David Kaminsky (IBM),"Automatic Program Transformation with JOIE", in Proceedings of the 1998 USENIX Annual Technical Symposium, 1998

[7] 김영국, 김경수, 김기태, 조선문, 유원희,"바이트코드 최적화 프레임워크의 설계", 제21회 한국정보처리학회 춘계학술발표대회 논문집 제11권 제1호, pp. 297 - 300, 2004