C 프로그램의 의존성 그래프 생성

김용호° 신승철
동양대학교 컴퓨터학과
yongho_kim°@hanmail.net shin@dyu.ac.kr

Dependence Graph Creation of C Program

YongHo Kim° Seung Cheol Shin
School of Computer Engineering, DongYang University

요 약

의존성 그래프는 컴파일러 최적화 분야에서 도입되어 유용하게 사용되고 있으며 프로그램 자르기(program slicing) 분야 등 활용가치를 높여가고 있는 프로그램 표현 방식이다. 우리는 실제적인 C 프로그램의 의존성 그래프를 생성하는 시스템을 구현하였으며 본 논문에서는 의존성 그래프의 개념과 구현과정을 설명한다.

1. 소 개

의존관계를 이용한 표현 방법은 컴파일러 최적화 분야에서 프로그램의 엄격한 실행순서를 고려하지 않으면서 최적화에 적당한 표현 방법을 찾으면서 나타났다. 이 표현 방법은 프로그램의 특정 지점에서 특정 변수에 대하여 영향을 미치는 문장을 찾아내는 프로그램 자르기[2] 분야에서 유용하게 사용되고 있다.

의존성은 어떤 문장을 실행할 때 다른 문장으로부터 영향을 받는 것을 의미하는 것으로 실행 의존성(control dependence)과 자료 의존성(data dependence)으로 나누어진다. 실행 의존성은 어떤 문장이 조건식에 영향을 받아 실행 여부가 결정될 때를 말하고 자료 의존성은 어떤 문장의 변수의 값이 다른 문장의 변수의 값에 영향을 받아 결정될 때를 말한다.

의존성과 관련된 표현은 1975년 Dennis[1]에 의해 자료흐름계산(data flow computation) 분야에서 처음 언급되었다. 그 후 실행 의존성과 자료 의존성은 각각 컴파일러 최적화 분야에서 응용되었으며 Ottenstein[3] 등에 와서 두 의존성을 하나의 그래프에 표현한 프로그램 의존성 그래프(Program Dependence Graph, PDG)란 용어를 정의하여 사용하게 되었다. PDG는 하나의 프로시저 내에서 즉 함수 호출 관계가 없는 의존 그래프로 표현한 방법이었기 때문에 완전한 프로그램을 표현하기에는 부족한 면이 있었다. Horwitz[4] 등은 함수 호출을 고려한 프로그램 자르기(interprocedural slicing)를 하기 위하여 PDG를 확장하여 시스템 의존성 그래프(System Dependence Graph, SDG)를 정의하였다.

본 논문에서는 실제적인 C 프로그램을 대상으로 하는 SDG 생성기 구현에 대하여 설명한다. 우리는 Harrold[5] 등의 구문에 따른(syntax-directed) 방법을 기반으로 하고, Necula 등이 제안한 C 프로그램의 중간언어(CIL[6])를 이용하였다.

2. 실행 의존성과 자료 의존성

어떤 문장 S가 조건식 C에 참(true) 실행 의존성이 있다는 것은 조건식 C가 참이 될 때 문장 S가 실행된다는 것을 의미한다. 아래의 프로그램 소스에서 2,3,4번 문장이 실행되기 위해서는 1번 조건식이 참이 되어야 한다. 그래서 2,3,4번 문장 각각은 1번 조건식에 참 실행 의존성이 있다. 거짓(false) 실행 의존성은 이와 반대로 조건식이 거짓일 때 실행되는 것이다.

```
1:  if (a>0) {
2:      b = 1 ;
3:      c = 2 ;
4:      d = b + 3 ;
5:  }
```

어떤 문장 S1이 문장 S2에 자료 의존성이 있다는 것은 S1에 포함된 변수의 값이 결정되기 위해서 S2의 문장에 영향을 받는다는 것을 의미한다. 위 소스에서 4번 문장의 d 변수의 값은 2번 문장의 변수 b의 값에 영향을 받는다. 그래서 4번 문장은 2번 문장에 자료 의존성이 있다.

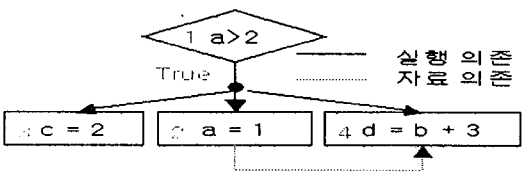

그림 1 의존성 그래프

[그림 1]은 위 소스에 대한 의존 그래프를 나타낸 것으로 1번 노드가 가장 먼저 실행되고 그다음 실행되는 것은 4번을 제외한 2,3번 노드가 실행될 수 있다. 4번 노드가 실행되지 못하는 이유는 2번 노드에 의존하기 때문이다. 이렇게 의존 그래프는 실행 흐름 그래프(Control Flow Graph, CFG)와 달리 프로그램의 의미를 변경시키지 않는 한 실행 순서에 자유롭다.

PDG는 하나의 함수에 대한 의존성 그래프라 생각할 수 있으며 이 PDG를 모아 프로그램 전체의 의존성 그래프인 SDG를 만들 수 있다.

3. 실행 의존성 그래프 생성

두 가지 의존성 그래프 생성법이 있는데, 하나는 CFG와 포스트 지배자 트리(post-dominator tree)를 이용하여 생성하는 방법[3]이고 나머지 하나는 프로그램의 AST(Abstract Syntax

Tree)로부터 바로 CDG를 생성하는 방법[5]이다. 두 번째 방식이 두 가지 입력 자료를 만들지 않아 효율성이 높다고 할 수 있으며 이 논문에서 사용한 방식이다.

AST를 입력 받아서 CDG를 생성하기 위해서는 문장 구조에 따라 생성되는 CDG의 모양을 알아야 된다.

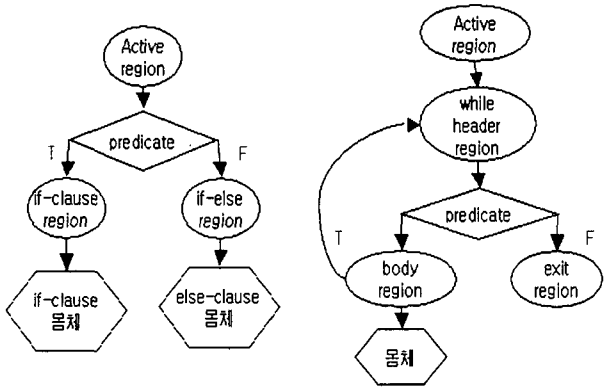

그림 2 if문 CDG 구조와 while문 CDG 구조

[그림 2]의 왼쪽은 입력된 AST가 if문장일 때 생성되는 CDG구조를 나타낸다. 구역(region)은 프로그램 소스의 문장과 직접적으로 대응되지는 않지만 그래프 상에서 다른 노드들을 포함하는 역할과 특정위치를 나타내는 역할을 한다. 활성 구역(active region)은 이 노드 아래에 CDG구조가 생성된다는 것을 나타내는 노드로서 프로그램 처음 시작할 때는 시작노드가 활성 구역이 된다. predicate는 조건식을 의미하고 각 몸체는 선택문의 참/거짓일 때 실행되는 문장들에 대한 CDG 구조가 생성되는 곳을 말한다.

C가 조건식이고 B가 몸체인 "while C B" 형태의 while문장은 "if C {B; while C B}" 형태로 해석될 수 있다. 몸체 B와 "while C B"문장은 조건식 C에 참 실행 의존성을 가지는 것을 알 수 있다. 그래서 [그림 2]의 오른쪽에 나타난 대로 while문의 CDG 구조에 몸체구역(body region)에서 while header 구역으로 연결선을 표시한다.

프로그램은 단순히 순차적으로만 실행되는 것은 아닌데, 즉 문장구조 안에 실행 전이자(break, return, continue)가 포함되어 있기 때문에 특정 문장으로 분기하게 된다. 이것은 실행 의존성 그래프를 생성하기 어렵게 만드는 요인이다.

```
1:  if(a>100)                1:  if(a>100)
2:      return;      ⇒       2:      return;
3:  a=a+1;                   3:  else a=a+1;
```

위 왼쪽의 2번 return문에 의해 실행 의존성은 영향을 받아 프로그램은 오른쪽과 같이 해석되어야 된다. 즉 2번 문장이 단순히 대입문이다면 3번 문장은 1번 조건식으로부터 실행의존성이 없지만 2번 return문 같이 실행전이자일 경우 3번 문장은 1번 조건식으로부터 실행 의존성이 있어야 된다.

이렇게 문장구조 속에 실행 전이자가 포함되어 있는 경우 실행 의존성은 크게 영향을 받는다. 또한 같은 실행 전이자라도 문장구조에 따라 실행 의존성 영향을 달리 준다. while문안에 break나 continue문이 포함되어 있는 경우 실행 전이자는 while문 범위를 벗어나 특정 지점을 실행할 수 없기 때문에 while문 밖의 실행 의존성 변화에는 영향을 끼치지 못하지만, 반면에 if문 속에 실행 전이자는 if문 범위를 벗어나 특정 지점을 실행 전이할 수 있기 때문에 if문 밖의 실행 의존성에 영향을 끼치게 된다.

3.1 실행 전이자가 포함된 문장의 실행 의존성

먼저 몇 가지 용어를 정의하고 문장 속에 실행 전이자가 포함되어 있을 때 실행 의존성을 구하는 방법을 알아보자.

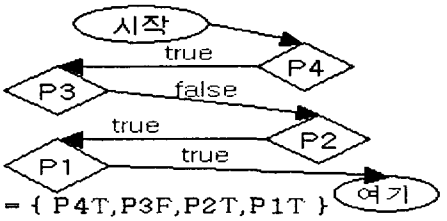

그림 3 조건 경로

□ 조건 경로(predicate path): 시작 구역에서부터 특정 노드까지 거쳐 온 조건식 경로로서 P_i가 (조건식, 진리값)의 쌍을 표시할 때 P_i 의 집합, $\Pi = \{P_n, P_{n-1}, \ldots, P_1\}$

□ ㄱΠ : 특정 노드가 실행되지 못하게 하는 조건 경로의 집합을 구하는 연산, $\Pi = \{P_n, P_{n-1}, \ldots, P_{wp}\}$

$$P_n \wedge P_{n-1} \wedge \ldots \wedge P_{wp-1} \wedge \neg P_{wp}$$
$$\vee\ P_n \wedge P_{n-1} \wedge \ldots \wedge \neg P_{wp-1}$$
$$\ldots$$
$$\vee\ \neg P_n$$

□ $\oplus \Gamma$: $\Gamma = \{\Pi_1, \Pi_2, \ldots, \Pi_m\}$, $\oplus \Gamma = \Pi_1 \vee \Pi_2 \vee \ldots \vee \Pi_m$

□ $\otimes \Gamma$: $\Gamma = \{\Pi_1, \Pi_2, \ldots, \Pi_m\}$, $\otimes \Gamma = \Pi_1 \wedge \Pi_2 \wedge \ldots \wedge \Pi_m$

[그림 3]은 P로 시작하는 노드가 조건식 노드라고 할 때 "여기" 노드까지 조건경로를 나타내고 있다.

아래의 조건에서 실행 의존성을 구해 보자.

- ○ 문장 S안에 실행 전이자 T_i가 m개 있음 ($1 \le i \le m$)
- ○ S의 시작부터 T_i까지의 조건 경로: $\prod_j$ ($1 \le j \le n$, $m \le n$)
- ○ S1은 S의 다음에 오는 첫 문장
- ○ A = {ㄱ$\prod_i$ | $\prod_i$는 T_i의 실행 전이 지점이 S1을 넘어가는 경우의 조건경로}
- ○ B = {$\prod_j$ | $\prod_j$는 T_j의 실행전이 지점이 S1인 경우의 조건경로}

경우 1: 아래의 왼쪽 예제처럼 모든 실행 전이자 T_i의 실행 전이 지점이 S1이 아니면서 S1을 넘어가지 않는 경우 S1은 T_i에 실행 의존성이 없다.

if(P3) // 문장 S, 경우 1	if(P3) // 문장 S, 경우 2
while(P2)	while(P2)
if(P1) continue;	if(P1) return;
else continue;	else continue;
S1	S1

경우 2: 실행 전이자 T_i의 실행 전이 지점이 S1은 아니면서 S1을 넘어가는 경우 S1은 ㉮ A에 실행 의존성을 가진다.

경우 3: 실행 전이자 T_i의 실행 전이 지점이 S1을 넘어가거나 S1인 경우 S1은 ㉮A ∨ ㉯B에 실행 의존성을 가진다.

```
while(P3)  // 문장 S, 경우 3
   if(P2) return;          // P3T, P2T
   else if(P1) break;      // P3T, P2F, P1T
        else return;       // P3T, P2F, P1F
S1
```

위 예제의 실행의존성을 계산해보면:

$\delta A = \{(P3F)\}$

$\delta B = \{(P3T,P2F,P1T)\}$

$\delta A \vee \delta B = \{(P3F),(P3T,P2F,P1T)\}$

이와 같이 나오는데 이것은 S1이 실행될 경우는 P3이 거짓이거나 break가 있는 문장이 실행되는 경우 밖에 없기 때문이다.

4. 자료 의존성 그래프 생성

자료 의존성은 흐름(flow) 의존성, 역(anti) 의존성, 출력(output) 의존성이 있지만 여기에서는 관심 있는 흐름 의존성만을 의미하는 것으로 한다.

도달 정의 분석(reaching definition analysis)법을 이용하여 자료 의존성을 계산하였다. 이 분석법을 사용하기 위해서는 프로그램의 실행 순서를 알아야 되는데 실행의존성 그래프를 생성할 때에 실행흐름 정보를 함께 포함시켜 CFG를 따로 생성하지 않도록 하였다.

프로그램에 포인터 변수가 포함되어 있는 경우 좀더 정확한 자료 의존성을 계산하기 위해서는 포인터 분석(pointer analysis)을 해야 되는데 차후에 구현할 예정이다.

5. 의존성 그래프 생성기의 구현

의존성 그래프는 [그림 4]와 같은 절차로 생성되었다. CIL 프로그램을 이용하여 C언어 소스에 대한 CIL AST를 생성하고 이것을 CDG 생성 모듈이 입력 받아 CDG를 생성하였다. CIL은 Frontc[7]라는 어휘검사와 문법 검사를 하는 프로그램을 내부적으로 호출하여 사용한다. 생성된 CDG에는 CFG의 정보가 포함되도록 하였기 때문에 DDG생성모듈에서 별도의 CFG를 이용하지 않고 자료 의존성 그래프를 생성할 수 있다. 구현을 통해 생성된 그래프는 매우 복잡하기 때문에 디버깅이 매우 어렵다. 따라서 우리는 메모리 상에 생성된 그래프를 그림으로 나타내기 위해 그림 그리기 도구[8]를 이용하도록 출력 모듈을 구현하였다. [그림 5]는 [표 1]의 소스에 대한 생성된 그래프의 그림을 보여주는 예이다.

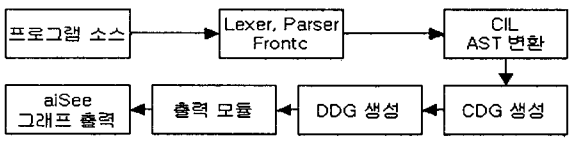

그림 4 생성 절차

```
void main(void) {
   int a=0;
   int b;
   while(a<100){
      if(a>50) break;
      else a++;
      b++;
   }
   return;
}
```

표 1

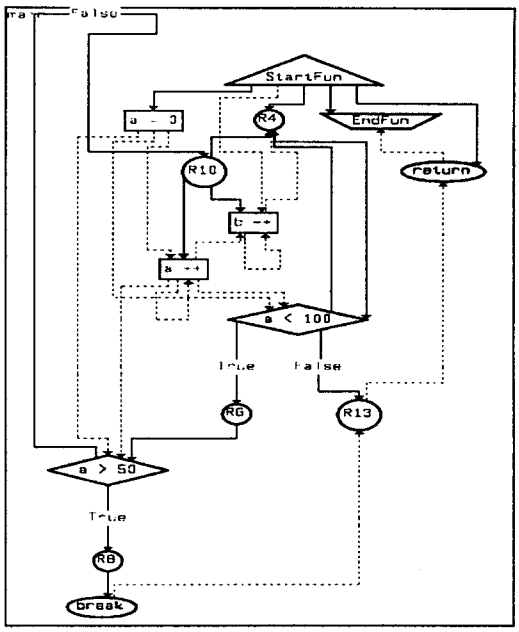

그림 5 생성된 SDG

5. 결 론

본 논문은 C 프로그램의 의존성 그래프 생성기를 구현하였다. 이것을 이용하여 C 프로그램의 프로그램 자르기 도구를 구현 중에 있으며 포인터 등을 포함할 수 있도록 확장하는 작업이 진행 중이다.

참고 문헌

[1] Dennis, J. B., First version of a data flow procedure language, revised Comp. Struc. Group Memo 93(MAC Teck. Memo 61) MIT LCS(May 1975) 21pages.

[2] Weiser, M., Program slicing, IEEE Transactions on Software Engineering SE-10(4) (July 1984) pp. 352-357

[3] Ferrante, J., Ottenstein, K., and Warren, J. The program dependence graph and its use in optimization. ACM Trans. Program. Lang. Syst. 9, 3 (July 1987), 319-349.

[4] Horwitz, S., Reps, T., and Binkley, D. Interprocedural slicing using dependence graphs. In Proceedings of the ACM SIGPLAN 88 Conference on Programming Language Design and Implementation (Atlanta, Ga., June 22-24, 1988). ACM SZGPLAN Not. 23, 7 (July 1988), 35-46.

[5] Mary Jean Harrold and Gregg Rothermel Syntax-Directed Construction of Program Dependence Graphs, The Ohio State University

[6] George C. Necula, Scott McPeak, S.P. Rahul and Westley Weimer, CIL(C Intermediate Language)
http://manju.cs.berkeley.edu/cil/

[7] Hugues Cass, FrontC is an OCAML library providing a C parser and lexer
http://casse.hugues.free.fr/projects/frontc.html

[8] aiSee Graph Layout Software
http://www.aisee.com/