2000년 한국정보처리학회 추계 학술논문발표집 제7권 제2호

데이터 토큰에 기반한 정보 흐름 그래프

박병기*, 최완규**, 이성주*
*조선대학교 전자계산학과
**광주대학교 컴퓨터전자통신공학부
e-mail:windwar@mina.chosun.ac.kr

The Information Flow Graph based on The Data Tokens

Byeoung-Gee Park*, Wan-kyoo Choi**, Sung-Joo Lee*
*Dept. of Computer Science, Cho-Sun University
**Div. of Computer Electronic & Communication Engineering, Kwang-Ju University

요약

본 논문은 프로시저 내에서 데이터 토큰들간의 정보흐름을 모델링 하기 위해서 정보 흐름 그래프(IFG : Information Flow Graph)라고 하는 프로그램 표현 도구를 개발하였다. IFG는 기존 제어 흐름 그래프와는 달리 프로그램 내에서 관심 있는 변수 값의 변화 과정을 쉽게 탐색할 수 있고, 정보 흐름의 미세 변화를 잘 표현할 수 있으므로, 프로그램에 대한 이해를 증진시킬 수 있다.

1. 서론

프로그램의 이해에 지대한 영향을 미치는 것은 프로그램의 표면적인 특징들이 아니라 프로그램의 정보 흐름(information flow)의 특징이다[6].

프로그램에서 정보 흐름을 표현하려는 기존의 연구들은 주로 제어 흐름 그래프(control flow graph)에 기초하고 있으며, 이러한 제어 흐름 그래프는 문장간의 제어 흐름(control flow) 즉, 각 문들의 실행 순서 및 제어 절차에 그 바탕으로 두고 있다[5]. 그러나, 이런 제어 흐름 그래프는 프로그래머가 이해하고자 하는 변수들 즉, 프로그램의 가장 작은 단위의 데이터 토큰들간의 정보 흐름에 대한 정보를 가지고 제공하지 않을 뿐만 아니라, 순환에 대한 정보 흐름을 알 수 없다. 따라서 본 연구에서는 정보 흐름에 따른 변수들의 미세 변화(elementary change)를 잘 표현 할 수 있도록 프로시저 내의 데이터 토큰들에서의 사용관계를 통해 프로그램의 정보 흐름을 모델링 할 수 있고[1], 순환에 대한 데이터 토큰들간의 정보 흐름을 표현하는 데이터 토큰 기반 정보 흐름 그래프(information flow graph : IFG)를 개발한다. 정보 흐름 그래프는 프로시저 내에서 데이터 토큰들간의 정보의 흐름을 분명하게 보여주므로, 관심 있는 변수 값의 변화 과정을 쉽게 탐색할 수 있고, 프로그램의 이해를 증진시킬 수 있다.

2. 정보 흐름 그래프(Information Flow Graph)

프로그램 문장 내에서 정의된 변수와 상수 정의를 의미하는 데이터 토큰[2, 3, 4]에 근거하여 프로그램에서의 정보의 흐름을 모델링하기 위해 본 연구에서는 정보 흐름 그래프(information flow graph: IFG)라고 하는 프로그램 표현 방법을 제안한다.

프로시저 P에 대한 IFG는 방향성 그래프, IFG(P) = <N, E>이다. 여기서 N은 프로시저 P에 포함되는 데이터 토큰들의 집합이고, E는 데이터 토큰들간의 정보 흐름을 나타내는 간선들의 집합이다.

프로시저 P에 대한 IFG(P)는 다음과 같은 단계를 거쳐서 구현한다.

단계 1: 문장 내에서 사용되는 데이터 토큰들간의 "정보 흐름" 관계에 따라서 프로시저 내의 모든 문

장들을 정보흐름 간선들의 집합으로 표현한다.

d_i, $d_j \in N$이 하나의 문장에서 나타나는 데이터 토큰이라 하자. 문장 내에서 d_i를 사용하는 연산의 결과가 d_j에 배정되면 ***d_j는 d_i를 직접 사용한다고*** 하고, $d_i \rightarrow d_j$로 표현한다. 반면에, 비교연산 등이 사용되는 경우는 ***dj는 di를 간접 사용한다고*** 하고, $d_i \dashrightarrow d_j$로 표현한다. d_j가 d_i를 직접 또는 간접 사용하면 ***d_i에서 d_j로 직접 또는 간접 정보가 흐른다***라고 한다[1, 2].

$N(S_k)$를 프로시저 내의 문장 S_k에 나타나는 모든 데이터 토큰들의 집합이라고 하자. 이때, 문장 내에서 정의된 데이터 토큰들간의 정보흐름 간선들의 집합은 다음과 같다.

$E(S_k) = \{d_i \rightarrow d_j \text{ or } d_i \dashrightarrow d_j \mid d_i, d_j \in N(S_k), i \neq j\}$

단계 2: 동일 변수에 속하는 데이터 토큰들간의 "정보 흐름"관계에 따라서 프로시저 내의 임의의 변수들에 관한 정보 흐름 간선들의 집합을 구한다.

$N(v)(N(v) \in N)$를 프로시저 내에서 사용되는 임의의 변수 v에 대한 데이터 토큰들의 집합이라고 하자. $d_i, d_j \in N(v)$일 때, $d_i \in N(S_p)$이고, $d_j \in N(S_q)$이고, $p < q$이면 ***d_j는 d_i를 직접 사용한다***(즉, ***d_i에서 d_j로 직접 정보가 흐른다***)라고 한다.

이때, 임의의 변수 v에 대한 정보 흐름 간선들의 집합은 다음과 같다.

$E(v) = \{d_i \rightarrow d_j \mid d_i, d_j \in N(v), i \neq j\}$

단계 3: $E(S_k)$와 $E(v)$를 결합하여 하나의 프로시저에 대한 흐름 그래프를 생성한 후, 추이 관계를 만족하는 데이터 토큰 과 간선들의 집합을 삭제한다. 즉, $\{d_i \rightarrow d_j, d_i \rightarrow d_k, d_j \rightarrow d_k\}$이면, $\{d_i \rightarrow d_j \rightarrow d_k\}$ 이다.

단계 4: 프로시저 내에 반복문이 있는 경우, 반복문 내에서 사용되는 동일 변수에 관한 데이터 토큰들간의 정보흐름을 추가한다.

$IFG_{loop}(v)$를 반복문 구조 내에서 위치한 문장들에서 사용되는 변수 v에 관한 데이터 토큰들로부터 생성되는 IFG(P)의 부분그래프(sub-graph))라 할 때, $IFG_{loop}(v)$에서 종단 노드 d_i에서 시작노드 d_j로의 직접사용 간선(즉, $d_i \rightarrow d_j$)을 추가한다.

$IFGloop(v) = \{ d_i \rightarrow d_j \mid d_i, d_j \in v, d_i < d_j\}$

```
Procedure SumAndProduct(
S1 : N : integer;
S2 : var SumN,
S3 : ProdN : integer)
S4 : ver I : integer;
begin
        S5 : SumN : = 0;
        S6 : ProdN : = 1;
        S7 : I : = 1 to S8 : N do begin
                S9 : SumN : = SumN + I;
                S10 : ProdN : = ProdN + I;
        S 11 : end
S 12 : end
```

그림 1. SumAndProduct procedure

예를 들어서 그림 1의 프로시저에 대해서, 각 문장들에 대한 노드들의 집합은 다음과 같다.

$N(S_1) = \{N_1\}$
$N(S_2) = \{SumN_1\}$
$N(S_3) = \{ProdN_1\}$
$N(S_4) = \{I_1\}$
$N(S_5) = \{SumN_2, 0_1\}$
$N(S_6) = \{ProdN_2, 1_1\}$
$N(S_7) = \{I_2, 1_2\}$
$N(S_8) = \{N_2\}$
$N(S_9) = \{SumN_3, SumN_4, I_3\}$
$N(S_{10}) = \{ProdN_3, ProdN_4, I_4\}$

위의 S_1, S_2, ... S_9, S_{10}은 각 문장의 수행 순서를 나타내며, 변수에 쓰여진 아래첨자는 각각의 변수와 상수에 대한 출현 순서를 나타낸다.

또한, 동일 변수에 속하는 노드들의 집합은 다음과 같다.

$N(N) = \{N_1, N_2\}$
$N(I) = \{I_1, I_2, I_3, I_4\}$
$N(SumN) = \{SumN_1, SumN_2, SumN_3, SumN_4\}$
$N(ProdN) = \{ProdN_1, ProdN_2, ProdN_3, ProdN_4\}$

문장내의 데이터 토큰의 "정보 흐름" 관계에 따라서 구해진 각 문장 내에서의 정보 흐름 간선들의 집합은 다음과 같다.

$E(S_5) = \{0_1 \rightarrow SumN_2\}$
$E(S_6) = \{1_1 \rightarrow ProdN_2\}$

$E(S_7) = \{1_2 \rightarrow I_2\}$
$E(S_8) = \{N_2 \rightarrow I_2\}$
$E(S_9) = \{SumN_4 \rightarrow SumN_3, I_3 \rightarrow SumN_3\}$
$E(S_{10}) = \{ProdN_4 \rightarrow ProdN_3, I_4 \rightarrow ProdN_3\}$

프로시저 내에서 사용되는 동일 변수들에 대한 "정보 흐름" 간선들의 집합은 다음과 같다.

$E(N) = \{N_1 \rightarrow N_2\}$
$E(I) = \{I_1 \rightarrow I_2, I_2 \rightarrow I_3, I_3 \rightarrow I_4\}$
$E(1) = \{1_1, 1_2\}$
$E(SumN) = \{SumN_1 \rightarrow SumN_2, SumN_2 \rightarrow SumN_3, SumN_2 \rightarrow SumN_4\}$
$E(ProdN) = \{ProdN_1 \rightarrow ProdN_2, ProdN_2 \rightarrow ProdN_3, ProdN_2 \rightarrow ProdN_4\}$

위의 노드들의 집합(N)과 간선들의 집합(E)으로 생성된 IFG에서 추이관계인 $SumN_2 \rightarrow SumN_3$, $ProdN_2 \rightarrow ProdN_3$을 삭제하고, 반복문 내에 있는 동일 변수의 I, SumN과 ProdN에 대한 간선들 $I_4 \rightarrow I_2$, $SumN_3 \rightarrow Sum_4$, $ProdN_3 \rightarrow ProdN_4$의 간선을 추가하면 그림 1에 대한 IFG(P)는 그림 2와 같다.

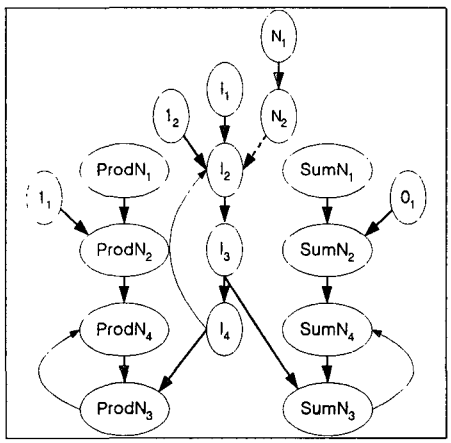

그림 2. 프로시저 SumAndProduct에 대한 IFG(P)

3. 비교 분석

그림 3은 그림 1에 대한 제어 흐름 그래프(CFG)를 보여준다.

제어 흐름 그래프(CFG)는 소스 프로그램의 수식의 결과에 따라 실행되는 문에 노드를 부여함으로써 프로그램의 구조를 분해하고, 노드들간의 정보 흐름을 그래프로 표현한 것이다. 이러한 제어 흐름 그래프는 문들 사이에서 데이터 단위의 제어 흐름 정보를 알 수 없고, 단순히 문들간의 정보 흐름만을 제시한다.

그러나, 본 연구에서 제안한 정보 흐름 그래프(IFG)는 주어진 소스에서 문장 단위로의 분해한 후, 다시 데이터 단위로 분해되어지는 변수들에 노드를 부여하는 과정을 통하여 각각의 변수들에 대한 정보 흐름까지 표현할 수 있다.

이와 같은 분해는 CFG에서 볼 수 없는 데이터 토큰 단위의 제어 흐름과 정보 흐름을 포함하고 있다.

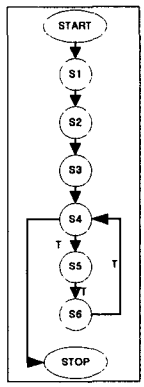

그림 3. 프로시저 SumAndProduct에 대한 CFG(P)

4. 결론

본 논문에서 제시한 정보 흐름 그래프(IFG)는 소스 프로그램에서 사용되는 변수들의 집합(N)과 그 변수들의 사용관계를 표현하는 간선들의 집합(E)을 가지고 있는 방향성 그래프이다. 즉, 소스 프로그램에서 각 문장에 사용된 변수들과 동일 변수에 속하는 변수들을 데이터 토큰단위로 추출한 후, 프로그램내의 각 문장과 변수들을 데이터 토큰들간의 정보 흐름 관계로 표현한다.

구현된 정보 흐름 그래프(IFG)는 각각의 변수들에 대한 정확한 정보 흐름을 가지고 있으며, 이들의 수행순서를 이해하기 쉽게 보여준다. 또한, 반복문내의 변수들의 정보 흐름 관계를 분명하게 표현해줌으로써, 프로그램에서 어떠한 변수들이 순환을 하는지를 알 수 있다. 이는 곧 프로그램 이해에 지대한 영향을 미치는 변수들의 미세 변화를 관찰하는 것이다.

현재까지의 작업은 단일 프로시저에서의 데이터 토큰에 기반한 정보 흐름 그래프의 구현에 집중하였

지만, 프로시저간의 정보흐름 뿐만 아니라, 객체 지향 패러다임에서의 변수간의 정보 흐름 그래프의 구현에 관한 연구가 필요하다.

참고 문헌

[1] 문유미, 최완규, 나영남, 이성주 "VFG에서의 슬라이스 복잡도", *99'정보처리학회 추계학술발표논문집(CD)*, 1999.

[2] J. M. Bieman, L. M. Otto, "Measuring functional cohesion", *IEEE Transaction On Software Engineering*, vol. 20, no. 2, pp. 111-124, 1994.

[3] J. M. Bieman, B. K. Kang, "Cohesion and reuse in an object-oriented system", *Proceeding ACM Symposium on Software Reusability (SSR'95)*, pp. 259-262, April, 1995.

[4] J. M. Bieman, B. K. Kang, "Measuring design-level cohesion", *IEEE Transaction On Software Engineering*, vol. 24, no. 2, pp. 111-124, 1998.

[5] 이광모, 한상영, "프로그램 종속성 그래프를 이용한 프로시듀어간의 종속성 표현과 병렬성 탐지" *정보과학회논문지*, v. 18, n. 4 pp.375-387 July 1991.

[6] Karl J. Ottensteion, Linda M. Ottensteion, "The program dependence graph in a software development environment", *Proceeding of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environment, ACM SIGPLAN Notices*, vol. 19, no. 5, pp. 177-184, 1984.