김일성종합대학학보
(정보과학)
주체107(2018)년
제64권 제1호

Java프로그람의 슬라이스를 위한 계층적계산모형

김현아, 신춘옥

지금까지 프로그람슬라이스는 프로그람의 오유수정, 병렬화, 프로그람차분화와 통합화, 쏘프트웨어유지, 시험, 역공학, 콤파일러들에 응용되였다.[1] 최근에는 슬라이스의 개념을 쏘프트웨어검증에 받아들이기 위한 연구[2, 3]도 진행되고있다. 그러나 수속형프로그람의 슬라이스생성에 대하여 연구되였지만 객체지향프로그람의 슬라이스생성에 대해서는 연구되지 못하였다.

론문에서는 Java언어의 고유한 계층적특징에 기초하여 Java프로그람에 대한 슬라이스를 생성하기 위하여 새로운 슬라이스계산모형을 제기하였다.

객체위주의 프로그람들은 명백한 계층구조를 가지고있다. 패키쥐는 클라스와 대면부로 이루어져있으며 클라스와 대면부는 method들과 성원변수로, method는 명령문들과 국부변수들로 이루어져있다. 이러한 계층적구조의 프로그람들에 대한 슬라이스도 계층적으로 생성되여야 한다고 본다.

계층적슬라이스생성모형 HSM(Hierarchical Slicing Model)을 3개의 구성부분으로 나누어 볼수 있다.

① 계층적슬라이스기준
② 계층적의존그라프
③ 단계별슬라이스생성알고리듬

계층적슬라이스생성모형을 다음의 그림에 보여주었다.

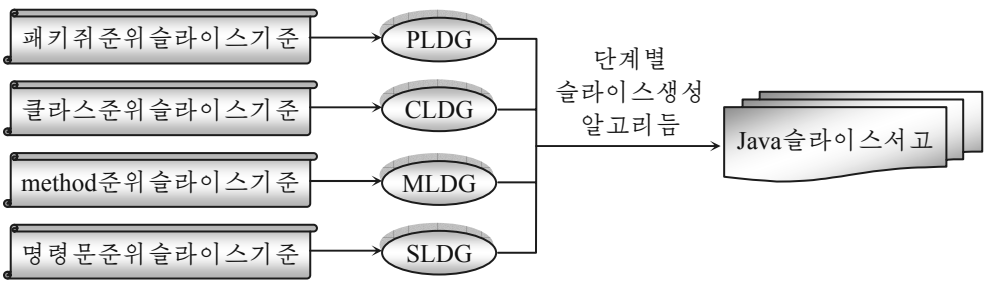

그림. 계층적슬라이스생성모형

1. 계층적슬라이스기준

Java프로그람 J가 주어졌다고 할 때 J에서 정의되였거나 리용된 패키쥐, 클라스/대면부, method, 명령문들의 모임을 각각 P_J, C_J, M_J, S_J라고 하자. 이때 $s \in S_J$에 대하여 명령문 s에서 출현하는 변수들의 모임을 V_S로 표시한다. V_S는 비였거나 단일원소모임일수 있다.

임의의 명령문 $s \in S_J$에 대하여 넘기기 $M(s), C(s), P(s)$에 의해 닫긴 method, 클라스

혹은 대면부와 패키쥐를 얻는다고 하자. 이때 속성 m의 명령문 s에 대해서는 m그자체이다. 변수 v에 영향을 주는 명령문들은 변수선언명령문, 값할당명령문의 왼쪽에 v가 있는 할당명령문 혹은 참고파라메터에 의한 method호출명령문들이 될수 있다. 변수 v에 의해 영향을 받는 명령문들은 v가 1번 이상 출현하는 식들이거나 v를 파라메터로 가지는 method호출명령문들이 될수 있다.

v에 영향을 주는 명령문들의 모임을 $affect(v)$로, v에 의해 영향을 받는 명령문들의 묶음을 $affected-by(v)$로 표시한다.

술어 B_v, F_v를 $B_v = affect(v), F_v = affected-by(v)$라고 하자.

만일 패키쥐 $p \in P_J$, 클라스(대면부) $c \in C_J$, method $m \in M_J$에 대하여

$$\exists s \in S_J, v \in V \cdot P(s) = p \wedge B_v(s)$$
$$\exists s \in S_J, v \in V \cdot C(s) = c \wedge B_v(s)$$
$$\exists s \in S_J, v \in V \cdot M(s) = m \wedge B_v(s)$$

라면 패키쥐 p, 클라스(대면부) c, method m은 변수모임 V에 영향을 줄수 있다.

만일

$$\exists s \in S_J, v \in V \cdot P(s) = p \wedge F_v(s)$$
$$\exists s \in S_J, v \in V \cdot C(s) = c \wedge F_v(s)$$
$$\exists s \in S_J, v \in V \cdot M(s) = m \wedge F_v(s)$$

라면 패키쥐 $p \in P_J$, 클라스(대면부) $c \in C_J$, method $m \in M_J$는 변수모임 V에 의하여 영향을 받을수 있다.

이로부터 변수모임 V에 영향을 주는 패키쥐는

$$\sum_{P \to V} = \{p \mid \exists s \in S_J, v \in V \cdot P(s) = p \wedge B_v(s)\}$$

에 의하여 결정되며 변수모임 V에 영향을 주는 모든 클라스(대면부)는

$$\sum_{C \to V} = \{c \mid \exists s \in S_J, v \in V \cdot C(s) = c \wedge B_v(s)\}$$

에 의하여 주어진다.

변수모임 V에 영향을 주는 모든 method들은

$$\sum_{M \to V} = \{m \mid \exists s \in S_J, v \in V \cdot M(s) = m \wedge B_v(s)\}$$

에 의하여 결정된다.

변수모임 V에 의하여 영향을 받는 패키쥐는

$$\sum_{P \leftarrow V} = \{p \mid \exists s \in S_J, v \in V \cdot P(s) = p \wedge F_v(s)\}$$

로 결정되고 변수모임 V에 의하여 영향을 받는 모든 클라스(대면부)는

$$\sum_{C \leftarrow V} = \{c \mid \exists s \in S_J, v \in V \cdot C(s) = c \wedge F_v(s)\}$$

에 의하여 주어진다.

변수모임 V에 의하여 영향을 받는 모든 method들은

$$\sum_{M \leftarrow V} = \{m \mid \exists s \in S_J, v \in V \cdot M(s) = m \wedge F_v(s)\}$$

에 의하여 주어진다.

정의 1 (계층적슬라이스기준)

프로그람 J가 주어지면 J와 관련된 슬라이스기준은 $< s, v >$이다. 여기서 $s \in S_J, v \in V(s)$이다. 넘기기 $M(s), C(s), P(s)$에 의하여 method준위, 클라스준위, 패키쥐준위의 슬라이스기준 $< M(s), v >, < C(s), v >, < P(s), v >$를 얻는다.

정의 2 (패키쥐준위슬라이스)

프로그람 J와 슬라이스기준 $< P(s), v >$가 주어지면 $< P(s), v >$와 관련된 J의 패키쥐준위뒤방향슬라이스는 $\sum_{P \leftarrow V}$에 의하여 주어진다. 여기서 $V = \{v\}$이다. $< P(s), v >$와 관련된 J의 패키쥐준위앞방향슬라이스는 $\sum_{P \rightarrow V}$에 의하여 주어진다.

슬라이스기준 $< M(s), v >, < C(s), v >, < s, v >$에 대하여 우와 같은 방법으로 method준위, 클라스준위, 명령문준위의 슬라이스들을 정의한다.

2. 계층적의존그라프

Java원천프로그람의 문장을 문장론해석준위에서 분석하여 코드정보나무(CIT)와 대역적기호표(GST)를 만든다.[1]

계층적슬라이스모형에서는 4가지 종류의 의존그라프 즉 패키쥐준위의존그라프(PLDG), 클라스준위의존그라프(CLDG), method준위의존그라프(MLDG), 명령문준위의존그라프(SLDG)들을 사용한다. 이러한 의존그라프를 CIT와 GST에 기초하여 구축하며 이것들은 내장형의 전문콤파일러인 JAST에 의해 얻을수 있다.

의존그라프생성알고리듬은 다음과 같다.

알고리듬 1 PLDG생성

PLDG의 자료구조는 int *PackageHierarchyDependence이다. PLDG는 정방형의 행렬 PA이며 그것의 위수는 기호표 SymbolTable의 PackageCount의 길이이다. PA의 첨수들은 원천프로그람에 있는 모든 패키쥐 id를 표현한다. $PA[i, j] = 1$이라는것은 패키쥐 i가 패키쥐 j를 반입한다는것을 의미하며 $PA[i, j] = 2$라는것은 패키쥐 i가 패키쥐 j의 부분패키쥐라는것을 의미한다.

입력: CIT

출력: PLDG

수속: PLDG구성(CIT *aProgram)

begin

1. for(int i=0; i<PackageCount; i++)
2. for SymbolTable에서 PackageDefine pi에 대하여
3. if (pi가 부분패키쥐) then

4. 부분패키쥐의존관계를 창조한다.
5. 정방행렬 PA에서 대응하는 위치에 2를 추가한다.
6. else
7. pi.ImportPackages로부터 pi의 모든 반입된 패키쥐의 이름을 얻는다.
8. SymbolTable로부터 반입된 매개 패키쥐의 id를 얻는다.
9. 패키쥐와 그것의 반입된 패키쥐사이에 반입의존관계를 만든다.
10. 정방행렬 PA에서 대응하는 위치에 1을 추가한다.

end

알고리듬 2 CLDG의 생성

CLDG의 자료구조는 int *ClassHierarchyDependence이다.

CLDG는 정값행렬 CA이며 행렬 CA의 위수는 원천프로그람에 있는 클라스와 대면부의 수 즉 ClassCount이다. CA의 첨수는 원천프로그람에 있는 매개 클라스 혹은 대면부의 id를 표현한다.

$CA[i, j]=1$은 클라스 혹은 대면부 j가 클라스 혹은 대면부 i를 계승한다는것을 의미하며 $CA[i, j]=2$는 클라스 j가 대면부 i를 실현한다는것을 의미한다.

$CA[i, j]=3$은 클라스 혹은 대면부 j가 클라스 혹은 대면부 i의 성원변수라는것을 의미하며 $CA[i, j]=4$는 클라스 j가 클라스 i를 창조한다는것을 의미한다.

CLDG구축알고리듬은 다음과 같다.

입력: 패키쥐의 CIT

출력: CLDG

수속: CLDG(CIT *aPackage)를 구축

begin

1. for(int i=0; i<클라스 Count; i++)
2. for SymbolTable에 있는 매 ClassDefine ci
3. 확장명령문에서 ci에 의해 계승된 모든 클라스의 이름을 찾는다.
4. 확장명령문에서 ci에 의해 계승된 모든 대면부들의 이름을 찾는다.
5. SymbolTable에서 매개 부모객체 및 부모대면부의 id를 찾는다.
6. 《계승의존》관계를 창조한다.
7. 정방형의 행렬 CA에 있는 해당한 위치에 1을 추가한다.
8. ci에 의해 실행된 매개 대면부의 이름을 찾는다.
9. SymbolTable에 있는 실행된 매개 대면부의 id를 찾는다.
10. 《실현의존》관계를 창조한다.
11. 정방형행렬 CA의 대응되는 위치에 2를 추가한다.
12. for ci에 있는 매개 성원method 혹은 성원변수
13. ci에 있는 성분변수 혹은 성원method의 정보 inf를 얻는다.
14. if(inf는 클라스형이거나 대면부형과 함께 성원변수) then
15. SymbolTable에 있는 성분변수의 클라스 혹은 대면부 id를 찾는다.
16. ci와 var의 클라스 혹은 대면부사이에 《집합의존》을 창조한다.
17. 정방형의 행렬 CA에 있는 해당한 위치에 4를 추가한다.

18. else(inf가 성원method이면)
19. CreatClassName에 있는 창조된 매개 클라스의 이름을 찾는다.
20. SymbolTable에 있는 매개 클라스의 id를 찾는다.
21. ci와 클라스 및 대면부사이에《창조의존》을 창조한다.
22. 정값행렬 CA의 해당한 위치에 4를 추가한다.

end

알고리듬 3 MLDG의 생성

MLDG의 자료구조는 int *MemberDependence이다. MLDG는 정방형행렬 MA로 표현된다. 행렬 MA의 위수는 모든 성분변수들과 성원method들의 개수 즉 MemberCount이다. MA의 첨수들은 클라스에 있는 매개 성분변수 혹은 성원method의 id를 표시한다.

$MA[i, j]=1$이라는것은 성원method j가 성분변수 i를 사용한다는것을, $MA[i, j]=2$라는것은 성원method j가 성분변수 i를 호출한다는것을 의미한다.

MLDG구축알고리듬은 다음과 같다.

입력: 클라스 CIT

출력: MLDG

수속: MLDG구축(CIT *a클라스)

Begin

1. for(int i=0; i<MemberCount; i++)
2. SymbolTable에 있는 for each method Define mi
3. if(mi가 성원method) then
4. MemberDataRefID에 있는 mi에 있는 참고된 매개 성분변수의 id를 찾는다.
5. MemberDataDefID에서 mi에 있는 참고된 매개 성분변수의 id를 찾는다.
6. MemberDataDefID에서 mi에 있는 정의된 매개 성분변수의 id를 찾는다.
7. MemberDataDefID에서 mi에 있는 정의된 매개 성분변수의 id를 찾는다.
8. mi와 이전의 성원method들사이에《MSY의존》을 창조한다.
9. 행렬 MA의 해당한 위치에 1을 추가한다.
10. Member method UseID에서 mi의 참고된 매개 성원method의 id를 찾는다.
11. mi와 이전의 성원method들사이에《호출의존》을 창조한다.
12. 행렬 MA의 해당한 위치에 2를 추가한다.

End

주의 1 SLDG의 생성알고리듬은 전통적인 알고리듬과 같다.[1]

주의 2 단계별슬라이스계산알고리듬은 의존그라프우에서의 뒤방향슬라이스계산알고리듬에 따른다.[3]

맺 는 말

Java프로그람의 계층성을 고려하여 슬라이스계산을 위한 계층모형을 제안하였다.

프로그람에 대한 계층적슬라이스계산은 패키쥐준위로부터 클라스준위, method준위, 명령문준위까지 단계별로 진행한다. 단계별슬라이스기준을 정의하고 단계별슬라이스계산

알고리듬을 제안하였다.

패키쥐준위, 클라스준위, method준위, 명령문준위의 의존그라프를 구축하는 방법과 구축된 의존그라프에 대하여 도달가능성을 분석하는 알고리듬을 제안하였다.

참 고 문 헌

[1] Xiaoguang Mao et al.; The Journal of System and Software, 89, 2, 51, 2014.

[2] Sara Van Langenhove and Albert Hoogewijs; WADT, 44, 2, 142, 2006.

[3] Ingo Bruckner, Heike Wehrheim; ICFEM, 6, 13, 2, 2005.

주체106(2017)년 11월 5일 원고접수

A Model for Slicing JAVA Programs Hierarchically

Kim Hyon A, *Sin Chun Ok*

Program slicing can be effectively used in debug, test, analyze, understand and maintain object—oriented software.

In this paper, a new slicing model is proposed to slice Java programs based on their inherent hierarchical feature.

Key words: software engineering, hierarchic model, program slice