제36회 한국정보처리학회 추계학술발표대회 논문집 제18권 제2호 (2011. 11)

자바 프로그램의 분석 및 검증을 위한 제어흐름그래프 시각화

정지웅* 김제민* 박준석* 유원희*
*인하대학교 컴퓨터공학과
jje0615@hanmail.net

Visualization of Control Flow Graph for Analysis and Verification of JAVA Byte Code

Ji Woong Jung* Je Min Kim* Joon Seok Park* Weon Hee Yoo*
*Dept of Computer Science Engineering, INHA University

요 약

프로그램에 대한 검증을 수행하기 위해서는 자료흐름을 알아야 하고, 입력된 프로그램에 대응하는 제어흐름그래프(control flow graph)가 필요하다. 이에 더하여 제어흐름 그래프를 시각화한다면 사용자 입장에서는 더 편한 프로그램이 될 것이다. 본 논문에서는 자바 프로그램의 검증도구에 사용하는 중간 표현 언어 중 하나인 BIRS(Bytecode Intermediate Representation with Specification)에 의해 생성되는 제어흐름그래프를 시각화하는 방법에 대해 제안한다.

1. 서론

우리 생활에서 소프트웨어의 비중은 날이 갈수록 커져가고 있다. 그 중 특히 자바 언어는 다양한 장소와 각종 장치에서 사용됨에 따라 안전한 실행이 보장되는지 검증하는 과정이 필요하다. 이러한 검증을 수행하는데 있어서 단순한 형태의 자료흐름 분석을 용이하게 하고, 정적분석과 검증에 대해서도 가능하게 해준다. 그런데 자료흐름 분석을 수행하기 위해 제어흐름그래프의 필요성이 생긴다.

제어흐름그래프는 분명 검증하는데 있어서 유용하고 효율적인 검증을 가능하게 하지만 사용자에게는 시각적이고 직관적인 형태의 모델이 오류를 수정하고 디버깅을 하는데 도움이 될 것이다.

본 논문은 2장에서 제어흐름그래프의 생성에 관한연구와 제어흐름그래프의 시각화와 관련된 기존연구에 관해 논의하고 3장에서 BIRS[1]에 대한 간략한 소개를 하고 4장에서 CFG의 생성방법을 제안한다.

2. 관련연구

2.1 CFG 생성기

CFG 생성기[2]는 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. 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.

2.2 Visualizer

Visualizer[3] 응용 프로그램은 내부 제어 흐름 그래프뿐만 아니라 레지스터 할당과 컴파일러의 중간 표현을 제공한다. 이것은 이클립스 리치 클라이언트 플랫폼 위에 구현되었고 플러그인 시스템을 사용한다. 그래프의 생성에는 Draw2D 라이브러리가 사용되었다. 이 논문은 제어 흐름 그래프의 시각화에 초점을 맞췄으며 위치 노드와 라우팅 간선에 대한 몇 가지 알고리즘을 제공한다.

3. BIRS 프로그래밍 언어

본 논문에서는 BIRS를 대상으로 한다. BIRS는 자바 프로그램 검증만을 위한 중간 표현 언어로서 자바 바이트 코드를 입력받아 생성한다. 스택 코드로 이루어진 자바 바이트 코드를 Sawja[4] 라이브러리를 이용해 스택리스 코드로 변경하여 검증에 용이하도록 설계되었다.

3.1 BIRS의 문법

BIRS의 문법에 대해 여기서 모두 다루기에는 힘들기 때문에 그 중 일부분을 소개한다. BIRS의 문법 중 최상위 수준은 선언들로 이루어진다. 변수의 선언, 함수의 선언, 클래스의 선언으로 프로그램의 시작을 표현한다.

```
LogicalFunctionDec ::==  logicalfunction Id :
                         (TypeList) → Type
LogicalExpr ::==  T | F | "¬" LogicalExpr
                 |ArithExpr RelOp ArithExpr
                 |ArithExpr BoolOp ArithExpr
                 |LogicalExpr "⇒" LogicalExpr
                 |LogicalExpr "≡" LogicalExpr
                 |PredicateId(ParamList)
                 |"forall"TypeIdList "." LogicalExp
                 |"exists"TypeIdList "." LogicalExpr
                 |PredicateDec
PredicateDec ::==  predicate PredicateId
                   (ParamList) := (LogicalExpr)
PredicateId  ::==  Id
```

몇가지 주요 선언들을 보면 LogicalFunction는 함수의 이름과 파라미터의 타입, 리턴타입을 알려주는 기능을 수행한다. LogicalExpr는 자바 바이트 코드에서 발생할 수 있는 논리적 표현에 관해 정의한 것으로 검증 조건을 생성하는데 적합한 형태로 표현되어있다.

```
ComputationFunctionDec ::==
        computationfunction Id :
        (ParamList)
        →Type{ [require LogicalExpr]
        [read (x,....,x)] [write (y,....,y)]
        Body [ensure LogicalExpr]}
ParamList ::==  Param [ "," ParamList ]
Param     ::==  Type VariableId
```

ComputationFunction는 함수에서 수행하는 모든 명령어에 대한 정보를 표현한다.

```
ClassId       ::==  Id
SuperClassId  ::==  ClassId
SubClassId    ::==  ClassId
InterfaceId   ::==  ClassId
ClassDec      ::==  Class ClassId :
        (SuperClassIdList) (SubClassIdList)
        (InterfaceIdList) (LogicalFuncDecList)
        { ComputationFuncDecList } Variable
```

ClassDec에서는 클래스의 이름과 클래스의 계층적 구조를 슈퍼클래스, 서브클래스, 인터페이스를 정의하여 표현 하고, Command는 자바코드의 모든 명령어를 Index를 기준으로 표현 할 수 있다.

4. BIRS의 CFG 시각화

4.1 시각화 방법과 예시

제어흐름그래프를 생성하기 위해서는 블록을 나누어야 하는데 블록을 나누는 기준은 점프를 하지 않고 순차적으로 실행되는 바이트 코드의 그룹이다. 이렇게 블록단위로 나뉜 것을 "BasicBlock"이라고 한다. 그림 1은 예제코드를 나타내고 그림 2는 BIRS로 변경된 예제 코드이다.

```
public class Example{
        int x;
        public void A(int y){
                if(y>0)
                        x+=y;
        }
}
```

그림 1 예제 코드

```
variable int x
predicate P(int y) := (y>0)
logicalfunction A:=( int ) → unit
computationfunction A:=( int y ) → unit{
read ( y ) write ( x )
block 0 : 0 : ifd y>0 1
assert P(y)
block 1 : 1 : check
          2 : check
          3 : affectField this.x : Example.int :=
              y+x
block 4 : 4 : return
from 0 to 1 when y>0
from 0 to 4 when ¬ y>0
from 1 to 4
returnblock 4 }
```

그림 2 BIRS 변경 코드

블록과 블록 사이를 잇는 표시를 "edge"라고 하며 제어 가능한 흐름을 나타내준다. 그림 3은 예제 코드의 제어흐름그래프를 보여준다. 각각의 블록은 고유번호를 가지고 있고 어떠한 점프도 포함하지 않는다. 블록에서 블록으로 이동하는 것은 간선 시작 블록의 마지막 명령 후 간선 끝 블록의 첫 번째 명령으로 점프를 실행할 때 이다. 예제에서는 B0의 마지막 명령어가 if이고, 실행하게 되면 비교의 결과 값에 따라서 B1이나 B4로 향하게 될 것이다.

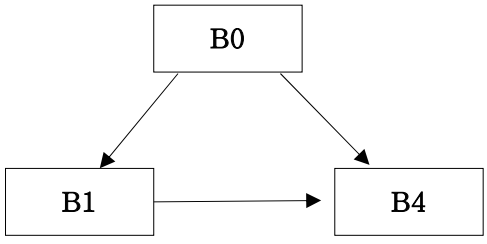

그림 3 예제 코드의 CFG 다이어그램

4.2 시각화

BasicBlock 개체들은 그래프에서 노드들로 연결된다. 이때 Draw2D가 블록을 그리는데 사용된다. 그렇게 만들어진 블록들은 Loop Positioning Algorithm을 사용하여 트리모형으로 만든다. 그림 4는 루프 알고리즘을 사용하여 배치하게 되는 모습이다.

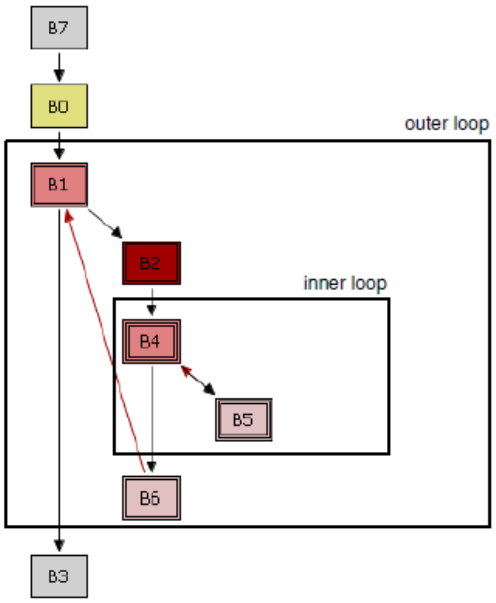

그림 4 루프의 그룹화

트리 모형이 완성되면 블록과 블록을 잇는 간선을 그려야 하는데 이 때 Bezier Routing Algorithm[5]을 사용한다. Bezier Routing은 사람이 선을 그리듯 표현해 주는데 사용자의 편의성을 높일 수 있다. 먼저 블록을 통과하는 간선에 대해서 회피 가능한 점으로 선을 옮겨서 블록에 닿지 않는 곡선형태의 간선을 완성시킬 수 있다. 그림 5는 회피 가능한 점을 나타내고, 그림 6은 선이 통과할 때의 예시이다.

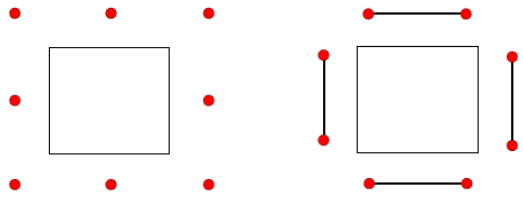

그림 5 회피 가능한 점

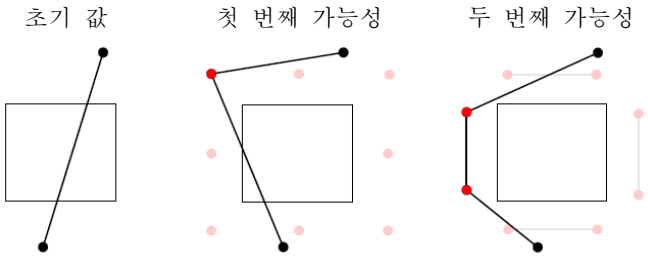

그림 6 선에 따른 두 가지 예시

위의 방법으로 프로그램 전체의 제어흐름을 보여주는 그래프를 시각화하여 검증에 필요한 정보들을 직관적으로 볼 수 있도록 한다. 사용된 알고리즘은 사용자가 판독하기 쉽고 제어흐름그래프를 표현하기 적합한 형태로 자바 프로그램을 분석하고 검증하는데 많은 도움이 될 것이다.

5. 결론 및 향후 연구

본 논문에서는 검증 과정에서의 자료 분석을 직관적으로 표현하기 위해 제어흐름그래프를 시각화는 방법에 대해 제안해 보았다.

향후에 본 논문을 사용하여 시각화 도구를 설계하고 구현해 볼 것이다. 또한 시각화에 사용되는 다른 알고리즘들과 비교하는 기회를 가져보고 적합한 방법에 대한 논의를 지속할 것이다.

논문 사사

이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임 (No. 2011-0026495).

참고문헌

[1] 김선태, 김제민, 박준석, 유원희, “자바 프로그램 검증을 위한 스택 리스 중간 표현 언어 생성기 구현”, 한국정보기술학회, 2011년 9월 30일.

[2] 김영국, 유원희, “스택-기반 코드로부터 분석을 위한 CFG생성기의 구현”, 한국정보기술학회 추계학술발표회 논문집, 2009년 11월.

[3] Thomas Würthinger, “Visualization of Java Control Flow Graphs”, Institute for System Software Johannes Kepler University Linz, October 2006.

[4] Laurent Hubert, Nicolas Barré, Frédéric Besson, Delphine Demange, Thomas Jensen, Vincent Monfort, David Pichardie, Tiphaine Turpin, “Sawja: Static analysis workshop for java”, Formal Verification of Object-Oriented Software (FoVeOOS), June 2010.

[5] Wm. Edwin Ellis, “PostScript ®, B6zler Curves, and Chinese Characters“, CSC '89 Proceedings of the 17th conference on ACM Annual Computer Science Conference, 1989.