가상실행에 기반을 둔 SW의 이해에 관현 연구

정양재, 박성옥, 이문근
전북대학교 컴퓨터 과학과
e-mail : {yjjung, sopark, mklee}@cs.chonbuk.ac.kr

A Study on SW Understanding based on Virtual Execution

Yang-jae jeong, Sung-og Park, Moon-kun Lee
Dept. of Computer Science, Chonbuk National University

요 약

다양한 실행경로가 존재하는 실시간 시스템을 이해하기 위해 정적 정보와 함께 동적 정보의 이해가 필요하다. 본 논문은 매개언어 SRL을 사용하여 정적 정보와 동적 정보를 모두 표현할 수 있는 방법을 제안했다. SRL을 통해 표현된 정적 정보와 동적 정보는 많은 시스템 이해 도구에 사용될 수 있다. 또한 동적 정보를 얻기 위한 방법으로 실제 시스템이 아닌 가상기계를 통한 가상실행을 수행함으로 안전성이 중요한 시스템을 검증이전에 위험부담 없이 검사할 수 있다.

1. 개요

실시간 시스템을 이해하는 것은 매우 어려운 일이다. 실시간 시스템은 매우 방대하여 수십만에서 수백만 라인으로 이루어져 있으며 그 내부를 살펴보면 수많은 병렬 프로세스들이 통신을 수행하고, 입출력, 예외처리, 시간에 대한 처리 등 많은 복잡한 관계가 성립되어 있다. 이렇게 복잡한 실시간 시스템에 대한 이해를 돕기 위해서 많은 방법론이 나오고 있다. 그러나 대부분의 방법론은 정적인 정보에 기반을 두고 있다. 복잡한 환경 속에서 존재하는 수많은 실행 경로 중에서 실재 실행 경로를 알기 위해서는 동적 정보를 알아야만 한다. 또한 SW 실행 중에서 문제가 발행했을 때 추적이 가능하기 위해서는 동적인 정보가 필요하다.

본 논문에서는 소프트웨어의 정적인 정보와 동적인 정보를 저장하기 위해 매개 언어 SRL(Software Representation Language)을 사용한다. 정적인 정보는 원시언어를 SRL로 변경한 후 SRL을 분석함으로 얻고 동적인 정보는 가상기계를 만든 후 가상실행을 통해 얻는다. 가상 기계를 사용함으로 심각한 오류가 있는 소프트웨어도 안전하게 검사할 수 있는 장점이 있다.

SRL에 포함된 정적인 정보와 동적인 정보는 프로그램 이해 도구를 만드는데 사용된다. 정적 정보 이해 도구는 CFG(Control Flow Graph), CG(Call Graph), DFG(Data Flow Graph), PDG(Procedure Dependency Graph)[4]를 만들고 동적 이해 도구는 동적 CFG, 검사 자료(Test Suite), 순방향·역방향 추적기, 변수 변환 패턴 분석기를 만들 수 있다.

2. 관련 연구

2.1 프로그램의 역실행

일반적인 디버거는 순방향으로의 실행만이 가능하다[2]. 만약 이미 실행된 부분에서 문제가 발생된다면 처음부터 문제 발생 부분까지 재 실행시켜야만 된다. [2]에서는 역방향 실행을 사용해서 이미 실행된 곳을 되돌아 갈 수 있는 방법을 제안했다. 역방향 실행을 위한 가장 쉬운 방법은 모든 실행 정보를 저장하는 것이지만 그렇게 하기 위해서는 방대한 양의 메모리가 필요하다. 역방향 실행문을 사용함으로 정보의 저장이 필요 없게 된다. 그렇지만 모든 실행문에 대해 역방향 실행문이 존재하는 것은 아니다. 역방향 실행문이 존재하지 않는 경우에는 추적파일(trace file)을 사용한다.

2.2 가상 기계를 사용한 컴퓨터의 개념 교육

[1]의 목적은 가상기계를 사용해서 학생들에게 컴퓨터 아키텍처를 가르쳐 주는 것이다. 가상 기계는 실재 하드웨어와 동일한 기능을 수행해야 한다. 학생들이 작성한 어셈블리어는 가상기계를 위한 기계어로 컴파일되고 가상 기계에서 실행된다. [1]에서 개발한 가상 기계는 Unix 기계 위에서 작동한다. 실재 하드웨어는 오직 한사람만이 사용할 수 있지만 가상기계는 한 번에 여러 개를 실행시킬 수 있으므로 여러 사람이 한 번에 사용할 수 있는 장점이 있다.

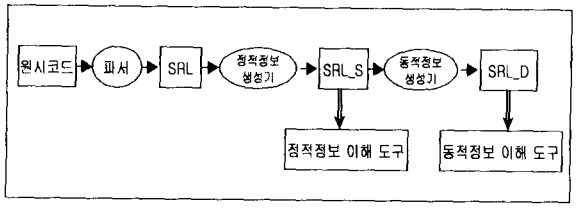
3. SRL을 통한 정·동적 정보 표현

프로그램의 정적 정보와 동적 정보의 추출은 <그림 1>의 단계를 거친다.

첫 번째 단계에서 파서는 원시언어를 파싱하여 SRL로 변경한다. 이 과정에서 원시 언어는 구문 노드와 그에 대한 수식 노드로 표현된다. 아직은 정적인 정보와 동적인 정보는 가지고 있지 않고 프로그램 구조에 관한 튜플만을 가지고 있으며 텍스트 형태로 파일로 관리된다. 두 번째 단계에서 정적정보생성기가 정적 정보를 생성한다. 텍스트 형태의 SRL을 메모리에 로드한 후 정적정보를 찾아낸다. 정적정보가 포함된 SRL(SRL_S, SRL with Static information)은 구문과 구문 사이를 연결하는 튜플과 구문과 구문의 내부에 있는 수식을 연결하는 튜플로 정적인 정보를 표현한다. 이 때 추출된 정적인 정보는 정적정보 이해도구에서 사용된다. 세 번째 단계는 가상기계를 사

본 연구는 한국과학재단 특정기초연구 (과제번호 19999-2-203-003-3) 지원으로 수행되었음.

용해서 SRL의 구문 내부에 연결되는 튜플 안에 존재하는 저장소에 동적인 정보를 저장한다. 이때의 SRL을 SRL_D(SRL with Dynamic iformation)라고 한다. 동적인 정보는 동적 정보 이해도구에서 사용된다.



<그림 1> 정·동적 정보를 추출하기 위한 전체 구조도

3.1 가상기계와 가상 실행

가상 기계는 하드웨어의 기능을 소프트웨어로 구현하는 프로그램으로 SRL을 입력받아서 실행된다. SRL은 실시간 시스템을 대상으로 하기 때문에 가상 기계는 실시간 시스템에 필요한 병렬성, 통신, 동기화, 시간 등의 기능까지도 포함한다. 또한 자원 등의 시스템의 환경요소까지도 지정 가능해야 하며, 모든 하드웨어에서 동일한 역할을 수행해야 한다.

가상 기계를 사용하면 몇 가지 장점을 얻을 수 있다. 첫째, 하드웨어에 대해 독립성을 얻을 수 있다. 모든 하드웨어에 동일한 기능을 수행하는 가상 기계를 사용하면 SRL은 하드웨어에 관계없이 동일한 실행을 보장할 수 있다. 둘째, 안전하게 실시간 시스템을 검사할 수 있다. 실시간 시스템의 경우는 안전성이 크게 요구되는 시스템들이다. 아직 검증되지 않은 시스템을 실제 하드웨어에 실행시키면 큰 사고가 발생될 수 있다. 가상 기계를 사용하면 모든 하드웨어의 기능을 활용할 수 있으면서, 문제점이 발생되더라도 사고의 위험을 피할 수 있다.

가상실행은 원시코드를 컴파일해서 실행시키는 것이 아니라 SRL로 변경한 후 가상기계를 통해서 실행시키는 것을 의미한다. 가상실행을 할 때 어려운 점은 특정언어의 라이브러리 함수의 경우 함수 내부의 문장을 알 수 없기 때문에 SRL로 표현하지 못한다는 것이다. 이 경우 함수 이름이 그대로 전달된다. 이 문제를 해결하기 위해서는 특정언어에만 있는 라이브러리 함수를 처리하기 위한 보관소(Repository)가 필요하다. C언어의 printf()를 살펴보면 printf() 안에는 출력을 하는 기능이 들어있다. 가상기계가 printf()의 기능을 수행하기 위해서는 라이브러리 이전의 원시 코드를 알아야 하지만 SRL을 통해서 가상기계를 통해 가상실행을 수행할 때 printf()라는 함수 이름만을 알 수 있다. 따라서 가상기계는 각 언어의 내장 함수를 처리하기 위한 보관소를 관리하고 SRL은 가상실행 이전의 값과 이후의 값을 저장한다.

3.2 SRL

SRL은 시스템을 구성하는 소프트웨어, 하드웨어, 통신망, 형상, 요구사항, 설계, 성능 등에 관한 정보를 저장소에 보관 및 관리하고 이를 재·역공학 과정에서 사용자의 특정 목적에 부합하게 표현하고 분석, 이해 및 평가하기 위한 메타언어이다.

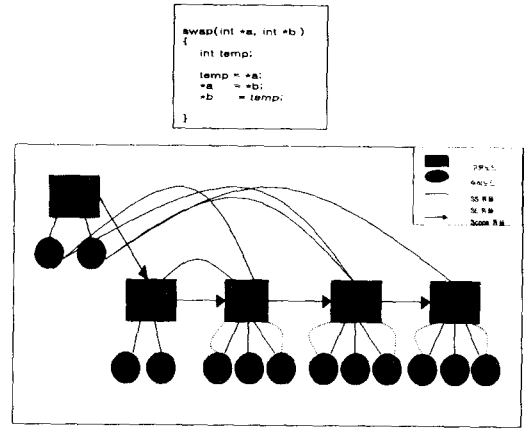
SRL은 ERA(Entity-Relation-Attribute)[4] 모델을 기반으로 실시간 SW를 나타내는 그래픽 언어이며, 문장을 위한 노드(Node)와 문장들간의 관계를 위한 에지(Edge)로 이루어졌다.

SRL에서는 SW의 각 구문을 노드로 표현한다. 노드 n의 스키마는 Sn=(id: 자연수, type: node의 유형, expression: string)이다. 기본 노드 선언문과 실행문이 있으며, 각각 다음과 같이 8개와 9개의 유형이 있다. 그리고 이 외에도 정의되지 않은 내용에 대한 참조를 표현하고 관리하기 위하여 비정의문이 있다. 선언문 노드 형의 종류는 N_{system}, $N_{generic}$, N_{spec}, N_{body}, N_{type}, N_{var}, N_{entry}와 N_{file}이 있다. 실행문 노드형의 종류는 $N_{context}$, $N_{control}$, $N_{condition}$, N_{assign}, N_{call}, N_{io}, N_{loop}, N_{begin}과 $N_{message}$이 있다. 각 노드의 세부 내용은 expression 노드로 표현된다.

SRL에 존재하는 튜플은 크게 두종류가 존재한다. 문장과 문장과 문장간의 관계를 나타내는 SS(Statement to Statement) 튜플과 문장 내부에 관계를 나타내는 SE(Statement to Expression) 튜플이 있다. SS 튜플은 문장의 연결된 두 개의 문장의 종류에 따라 8개의 형태가 있다. 8가지 튜플의 형태는 E_{scope}, E_{type}, $E_{context}$, E_{call}, E_{memory}, E_{io}, $E_{schedule}$과 $E_{message}$이다. 여기에서 E_{scope} 튜플은 다시 4가지 형태로 세분화된다. 4가지 E_{scope}은 $E_{scope.father}$, $E_{scope.brother}$, $E_{scope.then_son}$과 $E_{scope.else_son}$이다. SE 튜플은 1종류만이 존재한다. SE 튜플은 문장에 존재하는 변수와 사용자 자료형등에 대한 정적 정보 및 이들의 실행 값을 표현하는 동적 정보를 가질 수 있다. SE 튜플은 세부적인 정보를 표현하며 SS 튜플은 SE 튜플의 정보를 이용하여 문장들간의 관계를 나타낸다.

3.3 정적 정보

정적인 정보는 정적정보 생성기를 통해 추출한다. 정적 정보 생성기는 SRL을 메모리에 로드한 후 변수, 함수등의 식별자(Identifier)를 사용하여 변수 사용, 함수 호출, 함수 선언과 정의 등의 노드와 노드 사이의 관련성을 찾아내서 튜플로 연결한다. 구문 단위의 관련 정보는 SS 튜플로 표현되고 보다 자세한 정보는 SE튜플로 표현된다. <그림 2>를 보면 'a'와 'b' 두 개의 변수가 매개변수로 선언되었고, '*a=*b' 할당문(N_{assign})이 있다. 변수 'a'수에 대해 선언 노드와 사용 노드간에 메모리 튜플이 생성이 된다. 이 때 문장 단위로 SS튜플이 생성이 되고 사용된 곳을 보다 자세히 보이기 위해 SE튜플이 생성된다.



<그림 2> 정적 정보를 포함한 SRL

정적정보로는 CFG, CG, DFG, PDG등이 있다. CFG는 프로그램의 실행순서를 그래프로 표현한다. CFG를 그리기 위해서는 먼저 컨디션이 들어있지 않은 기본 블록을 구성해야 한다. SRL은 중첩에 따라 블록으로 구성되어 있기 때문에 아주 간단히 기본 블록을 만들 수 있다. 다음에는 실행순서에 따라 아크(arc)를 그린다. If, select 등의 조건문은 조건에 따라 실행해야할 부분을 하위 블록으로 소유하고 있고 그밖에 goto 명령에 대해서도 튜플이 존재한다. CFG에서 블록과 실행 흐름을 쉽게

알 수 있기 때문에 CFG를 간단히 생성할 수 있다. CG의 경우는 모든 함수의 호출 관계가 표현된 E_{call} 튜플을 통해 만들 수 있다. DFG의 경우는 변수의 선언과 변수의 사용 관계가 표현된 E_{memory} 튜플을 통해 추출된다. PDG의 경우 함수 사이의 관계에서 E_{call} 튜플을 통해 얻을 수 있다.

3.4 동적 정보

가상기계를 사용하면 가상 기계를 바탕으로 가상실행을 수행함으로서 정적인 정보가 포함된 SRL_S를 동적인 정보까지 포함된 SRL_D로 변경할 수 있다. 동적인 정보는 SE 튜플 안에 존재하는 저장소에 저장된다. 이 때 단순히 변수의 변화 값만을 저장하는 것이 아니라 프로그램의 상태 정보까지도 같이 저장한다.

동적인 정보는 SE튜플 안에 있는 버킷 안에 저장된다. 동적 정보를 표현할 때 실행 값과 함께 여러 가지 상태 정보를 같이 입력해야된다. SW의 이해를 돕기 위해서는 동적 정보의 변화가 발생한 상황도 필요하기 때문이다. 동적 정보의 저장형태는 블록, 중첩, 함수, 반복, 선택, 위치정보 등의 문제를 고려해서 결정해야 한다.

1) 블록 : 프로그램은 블록으로 구성되고 블록은 또 다른 블록과 구문으로 구성된다. 구문마다 구문 ID가 존재하지만 프로그램의 이해를 돕기 위해 블록으로 구별할 필요가 있다. 블록 구별하기 위해서 각 블록에 ID를 부여한다.
2) 중첩 : 블록 안의 블록을 나타내기 위해 블록 인덱스의 리스트로 표현한다.
3) 함수 : 함수의 경우 여러 번 호출될 수도 있다. 따라서 SE 튜플 안에는 여러 실행 정보가 함께 저장된다. 이 실행 정보를 각 함수 호출별로 구분하기 위해서 함수 호출을 구별할 필요가 있다. 이를 위해 매개변수 리스트로 이루어진 정보가 필요하다. 이를 본 논문에서는 parameter-context라고 명했다. 또 라이브러리 함수와 같은 경우는 가상기계를 통해서 초기값과 최종값을 알 수 있으므로 두 가지 값만을 저장한다.
4) 반복 : 반복의 경우에도 함수와 같이 문장이 여러 번 실행된다. 동적 정보가 반복 안에서 몇 번째의 실행인지를 구별하기 위해 반복횟수를 구별할 수 있는 인덱스가 필요하다. 만약 반복이 중첩되었다면 블록 인덱스로 구별한다.
5) 선택 : 여러 선택이 존재할 때 실행 경로를 추적하기 위한 방법으로 카운터를 사용한다. 카운터는 문장노드를 실행하면서 1씩 증가하게 된다. 만약 여러 선택이 반복으로 인해 여러 개의 실행 경로가 존재한다면 카운터를 검사함으로 실행 순서를 알 수 있다.
6) 실행 위치 정보 : 정적인 위치 정보는 블록인덱스와 구문노드의 ID로 알 수 있다. 동적인 위치, 즉 실행 과정의 위치 정보를 알기 위해서 카운터가 필요하다. 실행 값의 위치 정보를 알기 위해 전체 프로그램 안에서 카운터를 적용할 경우 카운터가 수백만 수천만이 넘는 숫자를 저장할 수 있도록 해야한다. 이런 낭비를 막기 위해서 위치 정보는 블록 ID 와 함수 내부의 순서만을 나타내는 지역 카운터의 조합으로 사용한다.

위의 여러 가지 사항을 고려할 경우 SE튜플 안의 버킷에는 다음과 같은 형태로 동적 정보가 저장된다.

value + block-index-list + parameter-context + counter

<그림 2>에서 a가 3이고 b가 4일 때 swap()함수가 호출되고 블록 ID가 B311일 때, '*a=*b'의 'a'에 연결된 SE튜플의 버킷에는 {3, B311, (3,4), 4}가 들어간다. a가 8이고 b가 3일 때 호출된다면 다음 버킷에 {8, B311 (8,3), 4}가 들어간다.

동적인 정보는 수많은 SE 튜플에 분산되어 저장되어 있기 때문에 그대로 동적 정보를 활용하기 어렵다. 동적 정보를 활용하기 쉽게 하기 위해서 두 가지 단계를 거쳐야 한다. 첫 번째 단계는 블록 단위로 동적 정보를 캡슐화한다. 블록에 들어오기 전의 변수 값과 블록을 빠져나갈 때의 변수 값을 저장한다. 이 방법을 사용함으로 블록의 추상화를 이룰 수 있기 때문에 프로그램의 이해를 도울 수 있다. 처음에는 블록을 하나의 단위로 이해하고 블록 내부를 선택하면 보다 자세한 정보를 알 수 있다. 두 번째 단계는 여기 저기 흩어져 있는 동적 정보를 하나의 저장소에 모으는 단계다. 동적 정보를 사용하는 도구를 만들기 위해서 저장소만을 사용하면 된다.

3.4 동적 정보 이용

동적 정보는 여러 도구에서 사용 가능하다. 시스템이 실행 도중에 문제가 발생했을 경우 문제점을 순방향·역방향으로 추적할 수 있다. 이때 모든 실행 정보가 저장되어 있기 때문에 튜플만을 이용하여 실행방향으로도 추적이 가능하고 순·역방향으로도 실행이 가능하다. 카운터를 사용하면 실행순서를 알 수 있다. 또한 루프의 경우 인덱스 정보가 있기 때문에 몇 번째 실행값을 요구할 때도 쉽게 대답할 수 있다.

정적 CFG의 경우는 여러 선택 방향이 존재할 경우 비결정적인 그래프가 생성되기 때문에 실제로 어떤 경로를 통해서 실행되었는지 알 수 없다. 그러나 동적 정보에서는 비결정적인 CFG에서 실행된 경로를 알 수 있으므로 오직 하나의 경로를 갖는 결정적인 동적 CFG를 생성할 수 있다. 또 중요한 변수의 변화 패턴을 살펴볼 수 있다. 변수를 선택한 후 튜플만을 살펴보면 그 변수의 모든 변화 내용을 알 수 있다. 함수의 경우 입력 값과 출력 값의 범위를 알 수 있는데 이는 후에 시스템을 검사하기 위한 검사 자료로 사용 가능하다.

4. 결론 및 향후 연구과제

매개언어 SRL은 정적인 정보뿐만 아니라 동적인 정보도 저장할 수 있다. 동적인 정보를 얻을 때 실제 하드웨어에 의한 실행이 아니라 가상기계에 의한 가상 실행을 수행함으로 여러 장점을 얻을 수 있다. 지금까지의 SRL은 제어에 관한 상태정보는 표현할 수 있지만 다양한 환경에 관한 정보는 표현할 수 없다. 따라서 이에 대한 확장이 필요하다. 또한 동적 정보를 저장하기 위해서는 방대한 양의 메모리가 필요하기 때문에 메모리를 관리하는 방법과 압축기법이 필요하다.

[참고문헌]

[1] Ola Agren, *"Teaching Computer Concepts Using Virtual Machine."* SIGCSE bulletin June, Vol. 3. No. 2, pp. 84-85, 1999.
[2] P. P. Chen. *"The Entity-Relationship Model-Toward a Unified View of Data,"* ACM Transactions on Database Systems, March, pp. 9-36, 1976.
[3] R.Mall, *"Revere Execution of Programs,"* ACM SIGPLAN Notices, April, pp. 61-69, 1999.
[4] Spencer Rugaber, "Program Comprehension," 1995.