XML을 사용한 C언어 소스 코드 분석 및 제어 흐름 분석

정아랑, 김현수
충남대학교 컴퓨터공학과
e-mail : {arangj, hskim401}@cnu.ac.kr

C language source code analysis and control flow analysis using XML

Arang Jeong, Hyeon Soo Kim
Dept of Computer Engineering, Chungnam National University

요 약

제어 흐름 그래프(CFG : Control Flow Graph)는 제어 흐름상의 오류나 문제점을 찾아내고 흐름에 대해 한눈에 파악할 수 있기 때문에 소프트웨어공학 분야에서 많이 사용되고 있다. 현재 여러 테스팅 분야에서 다양한 제어 흐름 분석 기법들이 연구, 소개되고 있는데 본 논문에서는 XML 문서를 이용하여 CFG를 나타내고자 한다. XML은 트리구조를 가진 문서 모델로 C 언어 소스 코드를 구조적으로 나타냄으로써 좀 더 쉽게 코드를 분석하고, 제어 흐름 요소를 추출하여 제어 흐름 그래프를 나타내는 데에 유용하다. 따라서 중간 분석 파일로 XML을 이용하여 보다 빠르고 쉽게 CFG를 나타내는 기법을 제안한다.

1. 서 론

CFG는 프로그램의 문장들 간의 제어 흐름 정보를 표현하는 방법이다. 제어 흐름 정보는 프로그램 분석과 소프트웨어 공학 분야에서 많이 이용되고 있다. 예를 들어, 자료 흐름 분석(data flow analysis), 제어 종속 분석(control dependence analysis), 예외 분석(exception analysis), 리그레션 테스팅(regression testing), 프로그램 슬라이싱, 테스트 데이터 생성 등을 위해서 제어 흐름 정보를 사용할 수 있다[1].

그 동안 여러 프로그래밍 언어에 대해서 제어 흐름 연구가 이루어져왔고 다양한 방법들이 소개되었다. 특히 C언어의 경우 재사용성이 떨어지고 메모리 오류 가능성이 높다는 단점을 가지고 있음에도 임베디드 분야 및 처리 속도를 중점적으로 생각하는 다양한 프로젝트에서 많이 사용되고 있다. 그러한 C언어의 소스 코드를 분석하여 제어 흐름 분석을 하는 연구들이 많이 이루어졌으나, 기존의 연구에서 C언어에서 CFG로 나타내기 위해 사용되는 중간 분석 파일들이 구조적이지 않은 경우가 대부분이었으며 그 때문에 원하는 CFG 요소들을 추출하기 위해서는 소스 코드의 구조 분석을 선행 작업으로 해야 하므로 더 복잡하고 더 많은 시간을 요구했다.

본 연구에서는 W3C 표준이며 구조적 문서 모델인 XML을 이용하여 C언어 소스 코드 구조를 알아내고 파싱하여 제어 흐름을 분석하는 방법을 제시한다. XML은 트리 구조를 가지는 문서로 CFG를 나타내기 위해 필요한 요소들을 구조적으로 파악할 수 있게 함으로써 더 빠르고 간편한 요소 추출을 하는 데에 유용하다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구로 C언어 소스 코드를 XML로 변환하는데 사용된 C2XML이라는 도구가 무엇인가, 제어 흐름이란 무엇인가 대해 기술한다. 3장에서는 어떻게 XML 파일에서 원하는 정보를 파싱하고 외부 함수에 대해서 처리하는 지에 대해, 4장에서는 결론 및 향후 연구 과제에 대해서 기술한다.

2. 관련 연구

2.1 제어 흐름 분석(Control Flow Analysis)

제어 흐름 분석은 프로그램의 각 문장에 대해서, 다음에 수행될 수 있는 문장들을 결정하는 것이다. 정상 제어 흐름은 기본적으로 프로그램의 쓰여 진 순서대로 한번 씩 수행이 되고, 제어 구조를 바꾸는 구문은 조건문, 반복문, 그리고 함수 호출이 있다[1]. 제어 구조가 바뀌는 부분을 결정 포인트(Decision point)라고 하는데 이 결정 포인트에서 유효한 테스트 케이스를 도출해 낼 수 있기 때문에 소프트웨어 테스팅에서 제어 흐름 분석이 사용된다. 제어 흐름 정보가 정확하면 정확할수록 정확한 분석 또는 테스팅 결과를 얻을 수 있다.

CFG는 실시간 및 자료 유도형 시스템에 적용 가능하며, 논리 및 자료 요구사항들을 텍스트를 좀 더 쉽게 분석할 수 있는 그래픽 흐름으로 변환한다. 대형 프로젝트에 대해, 메인과 서브 루틴간의 계층구조를 나타내는 CFG를 사용함으로써, 이 시스템의 프로그램 제어흐름을 유용하게 이해할 수 있다.

2.2 CFG를 이용한 연구 사례

제어 흐름 정보는 프로그램의 성능 측정, 테스트 데이터 생성, 예외 분석 등 소프트웨어 공학의 여러 분야에서 응용될 수 있다. CFG를 실제로 이용한 연구 사례는 아래와 같다.

내장형 시스템은 일반 PC와는 다르게 메모리 크기, 전력 소비, 신뢰성, 사이즈, 비용 등과 같은 제약사항을 내포하기 때문에 소프트웨어 성능의 필요성이 높다. [3]에서는 현재 가장 널리 사용되고 있는 이진 실행 파일인 ELF(Executable and Linkable Format)파일에서 성능을 측정하기 위한 기본 요소로서 CFG를 사용한다.

바이트코드(bytecode) 수준에서 프로그램 분석과 최적화

를 위한 구조를 서술하기 위해 CFG를 생성하여 사용한 연구 사례도 있다[4]. CFG 작성을 위해 기본 블록을 생성, 블록 간 관계를 이용하여 최적화 과정에서 사용되는 각종 정보를 생성한다. 생성된 CFG는 자바 바이트 코드의 이해와 유지보수를 위해 테스트되고, 데이터 흐름 분석과 의존성 분석과 같은 다른 분석을 위해서 사용된다.

또 다른 예로 통신 프로토콜 동작 기술을 위한 CFG 사용을 들 수 있다. 이종 간 통신을 위한 프로토콜 표준화 과정에서 CFG를 이용한 적합성 시험의 자동화를 위한 기법도 제안되었다[5].

2.3 C2XML

C2XML은 C언어 소스 코드를 분석하여 XML로 변환시켜주는 도구이다. C2XML에서는 크게 Expression, Declaration, Statement로 C언어 구문을 나누고 분석하는데, 정해진 DTD(Document Type De fination)에 따라 XML이 구성된다[2].

C2XML은 코드 개선, 분석 등에 관심이 있는 프로그래머와 연구자들에 의해서 개발되었다. ANSI-C 표준에 맞는 소스 코드는 모두 변환이 가능하다.

위에서 말한 Expression, Declaration, Statement는 각각 여러 요소를 가지고 그 모든 요소들이 서로 중첩되어 구조적으로 사용된다. 따라서 C언어 소스 코드에 따라 변환된 각 XML 파일은 다분히 다른 구조를 가지며, 특정 구조로 고정할 수 없다. 예를 들자면 <expression> 요소의 하위 요소로 <expression_assignment> 가 오고 그 하위 요소로 <expression_function>이 올 수 있는 반면, <expression> 요소의 하위 요소로 바로 <expression_function>이 오는 경우도 있다. 정의된 DTD 내에서 구조는 자유롭게 변경되므로 XML 파일 파싱 시에 고려해야 한다.

C2XML은 전처리 부분과 외부 함수에 대해서 자동적으로 파일 변환을 제공하지 않기 때문에 본 연구에서는 외부 파일에 대한 자동적인 XML 변환을 제공하기 위해 자동적으로 include 부분의 파일명을 얻어와 XML 변환이 되도록 한다.

3. XML 변환을 이용한 제어 흐름 분석 기법

제어 흐름의 구성요소는 분기, 반복, 함수 호출, 일반 연산으로 나눌 수 있는데 이러한 요소들을 노드(node)로 나타고 그 요소들 사이의 연관관계를 에지(edge)로 연결하여 나타낸다. 이러한 요소들이 모여서 하나의 제어흐름 그래프를 만들어 내기 때문에 제어 흐름 그래프 생성을 위해서는 위의 요소를 추출해야 한다.

XML 파일을 이용하여 제어 흐름 그래프를 생성하고 분석하기 위해서는 다음의 과정을 거친다.

(1) 메인 소스 코드 파일에서 참조하고 있는 파일명을 분석하여 프로그램에서 어떤 파일이 사용되는 지를 파악한다.

(2) 분석하고자 하는 C언어 프로그램의 메인 파일과 참조 파일을 모두 XML로 변환한다.

(3) XML 파일에서 CFG 구성요소를 추출한다.

(4) 변환한 모든 XML 파일에서 각각 자신이 가지고 있는 함수가 어떤 함수인지 분석해서 함수명을 리스트로 만들어 저장한다. 이 과정은 (3)에서 구성 요소 추출 시에 동시에 이루어지게 된다.

(5) main()을 시작으로 모든 함수에 대해 함수별로 제어 흐름을 생성한다. 이때 연관관계는 고려하지 않는다.

(6) 함수의 호출 관계를 분석하여 함수명과 파일명을 해쉬테이블 구조로 저장한다.

(7) 위에서 분석한 개별 함수의 제어 흐름과 연관관계를 총합하여 종합적인 제어 흐름 그래프를 생성한다.

3.1 참조 파일명 추출

C언어의 소스 코드는 크게 main함수를 포함하고 있는 실행주관 파일과 기타 함수 등을 구현한 외부 참조 파일로 나눌 수 있다.

C2XML에 의해서 XML로 변환된 내용에는 참조된 파일 내용이 포함되지 않기 때문에 include 되는 파일을 소스 코드에서 따로 추출해서 정보를 얻어야 한다. 다음과 같은 과정을 통해 참조 파일들에 대한 정보를 얻는다.

먼저 변환할 메인 소스 코드를 지정하여 읽어 들이며 어떤 파일을 포함하고 있는지를 파악한다. 파일명에 대한 정보는 리스트로 저장된다. - ex) stdio.h, test.h, sample.c, text.txt 등 include될 수 있는 모든 형태.

참조된 파일에서 또 다른 파일을 참조하는 경우도 존재한다. 리스트의 모든 참조파일에 대해서 더 이상 참조하는 파일이 없을 때까지 위의 과정을 재귀적으로 행한다.

일반적으로 사용자 정의 파일의 경우 #include " " 의 형식으로 표현되고, 표준 제공 참조 파일의 경우는 #include < > 의 형식으로 표현된다. 따라서 #include " " 의 형식을 가진 파일명만을 추출해 내도록 한다.

3.2 C 소스 코드 파일의 XML 변환

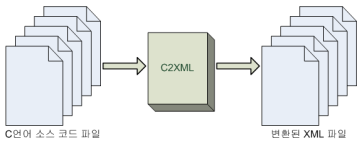

그림 1 C2XML을 이용한 C 소스 코드 파일 변환

메인 파일을 포함하여 모든 참조 파일명을 알아낸 후 그 파일들을 C2XML을 이용해서 전부 XML 파일로 변환한다.

```
.....
printf("~");

for(;i < 10;)
    .....

test_print(sum);
if(age >= 18) {
    .....
}
else {
    .....
}
sum = test_sum(inputNum1, inputNum2);
.....
```

(가) 원본 코드

```
.....
<expression_statement>
    <expression>
        <expression_function>
            <expression_id token="printf"/>
                .....
</expression_statement>
<for_statement>
    <expression_statement>
        <expression>
            <expression_lesser>
                <expression_id token="i"/>
                    <expression_cte token="10"/>
                        .....
</for_statement>
<expression_statement>
    <expression>
        <expression_function>
```

```
                <expression_id token="test_print"/>
                    <expression_id token="sum"/>
</expression_statement>
<if_statement>
    <expression>
        <expression_greater_equal>
            <expression_id token="age"/>
                <expression_cte token="18"/>
                    </expression_greater_equal>
    .....
    <compound_statement>
        .......
</if_statement>
<expression_statement>
    <expression>
        <expression_assignment>
            <expression_id token="sum"/>
                <expression_function>
                    <expression_id token="test_sum"/>
                        <expression_id token="inputNum1"/>
                        <expression_id token="inputNum2"/>
                            .....
</expression_statement>
```

(나) 변환된 XML 문서

그림 2 변환된 샘플 코드

변환된 XML 파일은 위의 그림 2의 (나)와 같은 구조를 가지게 된다.

C언어 소스 코드를 구조적으로 한 번에 보고 판단하여 파싱할 수 있기 때문에 다른 중간 매개체를 이용하는 것보다 훨씬 간편하게 소스 분석을 할 수 있다. 표현된 요소와 찾기를 원하는 제어 요소를 비교하여 찾아 낸 후 구조적으로 하위 요소에 접근이 가능하기 때문에 비구조적인 문서를 바탕으로 파싱하는 것보다 시간적, 경제적으로 더 적인 비용으로 제어 흐름 그래프 생성 시 필요한 정보를 얻어낼 수 있다. 이런 장점 때문에 본 연구에서는 XML을 중간 매개체로 이용하여 제어 흐름 분석을 제안한다.

3.3 XML 파일에서의 요소 추출

CFG는 노드와 에지로 구성되는데, 기본 블록(Basic Block)이 노드가 되고. 블록들 사이의 제어 흐름을 에지로 나타낸다. 기본 블록은 연속된 명령문의 나열로 블록 중간에 멈추거나 종료되지 않는, 시작과 종료 지점을 하나씩만 가지는 블록을 말한다.

명령문은 크게 분기문(Jump Statement), 반복문(Iteration Statement), 선택문(Selection Statement), 순차적 실행 구문(선언문, 할당문 등)으로 나눌 수 있다. C언어에서는 ANSI-C 표준으로 이런 명령문을 제공하고 그에 따라 제공되는 명령문들은 C2XML에 의해 각 함수를 나타내는 요소로 변환된다. 순차적 실행 구문들은 <expression> 요소의 하위 구조를 이용하여 나타내고 표준 함수의 경우 아래의 표와 같이 요소명이 지정되어 있다.

C2XML의 XML 구조 상 제어 문장을 나타내는 요소들은 다음과 같다. J는 분기문, I는 반복문, S는 선택문을 나타낸다.

요소명	종류	설명
label_statement	S	goto의 label 제어문에 대해 나타내기 위한 요소. 자식 요소로 관련 정보를 나타낸다.
case_statement	S	case 제어문을 나타내기 위한 element. 자식 요소로 관련 정보를 나타낸다.
default_statement	S	label, case, goto 등의 제어문에서 사용되는 default 제어문을 나타내기 위한 요소. 자식 요소로 관련 정보를 나타낸다.
if_statement	S	if 제어문을 나타내기 위한 element. 자식 요소로 관련 정보를 나타낸다.
switch_statement	S	switch 제어문을 나타내기 위한 요소. 자식 요소로 관련정보를 나타낸다.
while_statement	I	while 제어문을 나타내기 위한 요소. 자식 요소로 관련정보를 나타낸다.
do_statement	I	do 제어문을 나타내기 위한 요소. 자식 요소로 관련정보를 나타낸다.
for_statement	I	for 제어문을 나타내기 위한 요소. 자식 요소로 관련정보를 나타낸다.
goto_statement	J	goto 제어문을 나타내기 위한 요소. 속성으로 이동할 label 정보를 나타낸다.
continue_statement	J	continue 제어문을 나타내기 위한 요소. 빈 요소로 자식 요소도, 속성정보도 가지지 않는다.
break_statement	J	break 제어문을 나타내기 위한 요소. 빈 요소로 자식 요소도, 속성 정보도 가지지 않는다.
return_statement	J	return 제어문을 나타내기 위한 요소. 여러 expression 요소를 자식 요소로 가진다.

표 1 제어문을 나타내는 요소들

위의 표 1에서 나타내는 내용들을 XML 파일에서 추출해 낸다면 제어 흐름에 필요한 정보를 얻어낼 수 있다. 전체 XML 파일을 navigator를 통해 검색해서 표 1의 요소들과 동일한 요소명을 발견하면 그 요소를 하나의 노드로 판별한다. XML에서는 요소의 시작과 끝을 함께 나타내므로 언제 그 요소가 종료 되는지를 알 수 있다. 요소 a의 종료 후 나타나는 요소 b를 판별함으로써 둘 사이를 연결하는 에지 또한 알 수 있다. 이때 함수로 판단되는 요소는 그 함수의 이름과 파일명을 따로 해쉬테이블에 저장한다.

C2XML을 이용해 변환된 XML 파일의 최상위 요소(root element)는 <translation_unit>이며 하위 요소는 특정 레벨에서만 나타내는 것이 아니라 중첩되고 반복적인 구조를 보인다. 따라서 똑같은 <expression_statement>라는 이름을 가진 요소라고 할지라도, 그 위치와 레벨은 상당한 차이를 보일 수 있다.

그렇기 때문에 일괄적인 위치를 찾아서 원하는 요소 정보 및 속성(attribute)정보를 얻어내는 것은 쉽지 않다. 따라서, C2XML에서 제공하는 DTD에 벗어나지 않는 형태에서 가능한 모든 상황에 대해 검색할 수 있어야 C2XML상의 원하는 정보를 얻을 수 있다. navigator는 원하는 정보의 위치를 알 수 없기 때문에 XML 파일의 처음부터 끝까지 모든 노드를 검색하게 된다.

각 XML 파일은 XPathNavigator 객체를 이용하여 검색을 하며, XPathDocument 클래스는 XML 문서를 읽어 DOM 트리를 형성한다. 실질적으로 XPathNavigator가 하고 있는 것은 아니며, XML.NET에서는 빠른 검색을 위한 XPathNodeIteration 클래스를 제공한다.

생성한 navigator로 XML 파일을 처음부터 끝까지 읽으며 CFG로 나타낼 부분에 대한 정보를 얻는다.

그림2-(나)의 XML 파일을 검색하며 위 표1에 의해 어떤 제어문을 가지고 있는 지 찾아낸다.

3.4 각 XML 파일에서 함수 이름 요소 추출

만들어진 XML 파일명은 소스 코드 파일명을 따라 “참조파일명.xml” 의 형식이므로 각 XML 파일을 읽어 들여 각 파일에 구현되거나 정의된 함수명을 3.3의 추출 기법에 따라 분석해서 리스트로 저장한다.

사용자 함수는 <expression_function> 노드로 표현이 되고, 함수 이름, 함수의 인자 등은 그 하위 노드로 표현이 된다. 3.3에서 XML 파일 분석 시 <expression_function> 요소를 찾고 하위의 <expression_id> 요소를 찾아 그 속성을 알아내면 사용자 함수의 이름을 알 수 있다.

각 XML 파일별로 3.3의 기법으로 분석하며 사용자 함수명을 알아내어 리스트에 저장한다.

3.5 함수 호출 관계 분석

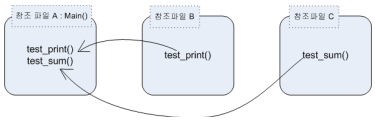

그림 3 함수 간의 호출 관계

파일 A : main의 해쉬테이블	
키(함수명)	값(파일명)
test_print()	파일B
test_sum()	파일C

표 2 main의 함수 관계에 대한 해쉬테이블 구조

표 2의 소스 코드를 함수간의 호출관계를 표시하여 나타내면 위의 그림 2와 같이 나타낼 수 있다.

모든 프로그램의 시작은 main()에서 이루어지기 때문에 main()을 포함하고 있는, 메인 XML 파일에서부터 함수 연관 관계를 파악한다.

XML의 요소 중에 함수를 나타내는 요소가 있다면 그 함수의 이름을 파싱하여 얻어내고 모든 함수 정보가 들어 있는 리스트들과 비교하여 어느 파일에 속하는 함수인지를 알아낸다. 이 연관관계 정보는 또 다른 저장 구조에 저장되게 된다. 해쉬테이블 구조를 이용하여 저장한다. 해쉬테이블은 빠른 검색을 목적으로 최적화된 사전 구조로 되어 있다. 또한 사전식 검색뿐만 아니라 각각의 요소에의 접근 및 나열에도 아주 편리한 기능을 제공한다. 각 함수별 외부에서 참조하는 함수 정보를 표 3과 같이 해쉬테이블로 가지게 된다.

3.6 함수내의 제어 흐름 그래프 출력

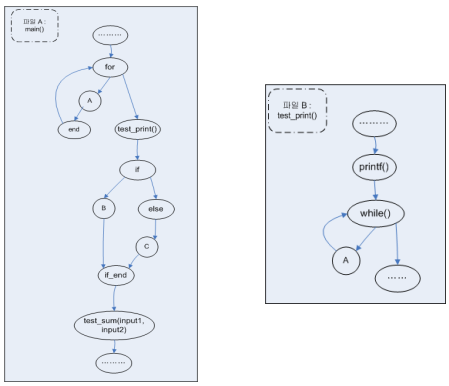

그림 4 각 함수별 제어 흐름 그래프

3.4과 3.5에서 분석한 정보를 기반으로 각 함수 단위로 제어 흐름 그래프를 출력하도록 한다.

C언어의 흐름은 main()이 주도하기 때문에 main() 함수의 제어 흐름부터 출력하도록 한다. 이 단계에서는 각 함수간의 관계는 나타내지 않고, 외부로부터 참조되는 함수가 있다는 정도만 나타낸다.

3.7 전체 시스템의 제어 흐름 분석

3.4~3.6의 정보를 모두 종합하여 그림 4와 같이 전체 C언어 프로그램의 제어 흐름을 나타낸다. 3.6에서 나타낸 각 함수의 제어 흐름 그래프에 에지를 추가함으로써 어떤 함수 호출 관계가 있는 지를 나타낼 수 있다. 구현은 되었으나 실제로 호출되지 않은 함수의 경우, 그래프는 나타나지만 연결되는 에지가 없이 홀로 떨어져 존재하게 된다.

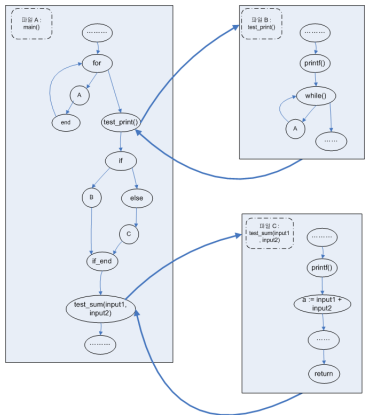

그림 5 호출 관계를 포함한 전체 제어 흐름 그래프

4. 결론 및 향후 과제

본 연구에서는 XML 이라는 구조적인 문서를 이용하여 C언어 소스 코드를 분석하여 CFG를 나타내는 데 필요한 요소들을 추출하고, 그 요소를 이용하여 CFG를 나타내는 방법에 대해서 제안하였다.

C언어 소스 코드 전체를 포괄하여 제어 흐름을 분석하는 것을 중점으로 했으며 ANSI-C 기반의 소스 코드는 C2XML을 통해 거의 모든 부분에서 변환, 파싱 가능하였다.

향후 연구에서는 크기가 크고 외부 레퍼런스 파일이 많은 소스 코드의 경우에 얼마나 빨리 XML로 변환하고 로드하고 파싱하느냐가 성능에 영향을 끼치기 때문에 효율성을 높이기 위해 좀 더 간결한 알고리즘을 사용하도록 보완 할 것이다. 파일이 많아지고 커질 때의 경우에 CFG에서 함수와 파일, 제어 문장 간의 상호 관계가 명확하게 나타나도록 가시화를 해야 되기 때문에 가시화에 대한 연구도 더불어 진행할 것이다.

5. 참고 문헌

[1] 조장우, 이정수, “Java의 예외 제어 흐름을 포함한 제어 흐름 그래프 생성”, 한국정보과학회, Vol. 29, No.2, pp. 649-651, 2002

[2] http://www.plutospin.com/c2xml.html

[3] 황요섭, 안성용, 이정아, 심재홍, “실시간 내장형 S/W의 성능분석을 위한 Control Flow Graph 추출”, 한국정보과학회, Vol.30, No.1, pp.217-219, 2003

[4] 김기태, 유원희, “CTOC에서 자바 바이트코드를 이용한 제어 흐름 분석에 관한 연구”, 한국콘텐츠학회논문지, Vol.6, No.1, pp.160-168, 2006

[5] 김경희, 박재홍, 이재용, 정일영, “적합성 시험을 위한 CFG Generator에 관한 연구”, 한국정보과학회, Vol.21, No.2, pp.493-496, 1994