EVM SIL에서 C 프로그램 생성을 위한 역컴파일러의 설계 및 구현

김영근°, 권혁주, 이양선
서경대학교 컴퓨터공학과
e-mail:{ykkim°, hjkwon, yslee}@pl.skuniv.ac.kr

Design and Implementation of Decompiler for Generating C Program from EVM SIL

Young-keun Kim°, Hyeok-Ju Kwon, Yang-Sun Lee
Dept. of Computer Engineering, SeoKyeong University

요 약

기존의 ANSI C 프로그램은 각각의 플랫폼에 따른 컴파일러를 통해서 목적기계의 코드로 변환되고, 실행되어 플랫폼에 의존적인 단점이 있다. 이러한 단점을 보완하는 방법으로는 스택기반의 가상기계와 가상기계의 입력형태인 중간코드를 이용하는 기법이 있다. EVM(Embedded Virtual Machine)은 ANSI C 언어와 SUN사의 Java 언어 등을 모두 수용할 수 있는 임베디드 시스템을 위한 가상기계이며, SIL(Standard Intermediate Language)은 EVM에서 실행되는 중간언어로 다양한 프로그래밍 언어를 수용하기 위해서 객체지향 언어와 순차적인 언어를 모두 수용하기 위한 연산 코드 집합을 갖고 있다.

본 논문에서는 SIL 코드가 올바른 수행을 하는 것인지를 검증하고 원시코드의 분석을 용이하게 하기 위해서 생성된 SIL 코드를 어셈블리 형태와 유사한 재 표현된 ANSI C 언어로 바꾸는 역컴파일러 시스템을 설계하고 구현하였다.

1. 서론

범용 프로그래밍 언어에서 널리 쓰이는 ANSI C 언어로 작성된 프로그램은 각각의 플랫폼에 따른 컴파일러를 통해서 목적기계의 코드로 변환되고, 실행되어 플랫폼에 의존적인 단점이 있다. 이러한 단점을 보완하는 방법으로는 스택기반의 가상기계와 가상기계의 입력형태인 중간코드를 이용하는 기법이 있다.

가상기계에는 썬사의 JVM과 마이크로소프트사의 .NET이 있으며, 바이트코드와 MSIL 코드라는 중간코드를 입력으로 받아서 플랫폼에 독립적으로 실행된다. 그리고 모바일 디바이스, 셋톱박스, 디지털 TV와 같은 임베디드 시스템을 위한 EVM이 있다. EVM은 스택기반의 가상기계로 플랫폼에 독립적으로 실행되며, 중간코드로는 SIL 코드가 있다. SIL 코드는 객체지향 언어와 순차적인 언어를 모두 수용하기 위해 설계되었다.

기존의 ANSI C언어로 작성된 프로그램을 ANSI C 컴파일러를 통해서 SIL 코드가 생성되고, EVM에서 실행함으로써 플랫폼에 관계없이 프로그램을 실행할 수 있는 환경을 제공할 수 있다.

이러한 ANSI C 컴파일러의 출력을 EVM 가상기계에서 수행하여 올바르게 결과가 출력됨을 확인하여야 하지만, 현재 EVM이 완성된 상태가 아니라 개발 중에 있기 때문에 확인이 불가능하다. 그래서 SIL 코드를 검증하기 위해서 3-주소 코드의 쿼드러플(quadruple) 형태로 재 표현된 ANSI C 언어로 역컴파일 하여 기존의 Visual C 컴파일러를 통해 실행하는 방법을 선택하였다. 또한 역컴파일러는 실행을 위한 검증만이 아닌 원시코드의 분석을 위한 도구로도 필요한 시스템이다.

본 논문에서는 EVM의 중간코드인 SIL 코드를 재 표현된 ANSI C 언어로 바꾸어 주는 역컴파일러를 설계 및 구현하였다.

본 연구는 한국과학재단 목적기초연구(R01-2002-000-00041-0) 지원으로 수행되었음

2. 관련연구

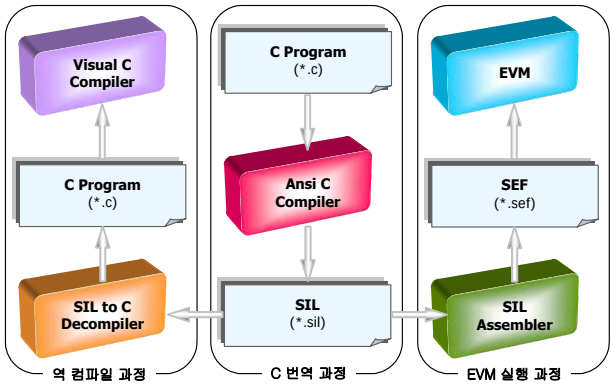
2.1 번역기

하나의 프로그래밍 언어로 작성된 프로그램을 다른 플랫폼에서 실행하기 위한 번역기에 관한 연구로는 마이크로소프트의 자바 프로그램을 C# 프로그램으로 변환하는 JLCA(Java Language Conversion Assistant) 번역기가 있다[8]. 또한, 헬시온소프트는 .NET의 MSIL 코드를 자바 소스 프로그램으로 번역하는 iNET 번역기[9]를 개발 하였으며, 역으로 리모트소프트는 자바 소스 프로그램을 .NET의 MSIL 코드로 변환하는 Java.NET 번역기[10]를 개발하였다.

2.2 EVM

EVM은 모바일 디바이스, 셋톱 박스, 디지털 TV 등에 탑재되어 동적 응용 프로그램을 다운로드하여 실행할 수 있는 가상기계 솔루션이다.

EVM은 크게 컴파일러, 어셈블러, 가상기계의 세 부분으로 구성된다. EVM은 계층적인 구조로 설계되어 리타겟팅 과정의 부담을 최소화 한다[3]. 다음 [그림1]은 EVM의 시스템 구성도 이다.



[그림1] EVM 시스템 구성도

2.3 SIL

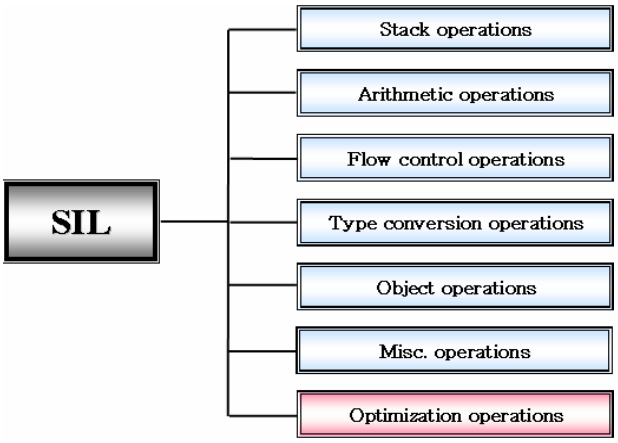
EVM의 가상기계 코드인 SIL은 일반적인 임베디드 시스템을 위한 가상기계 코드의 표준화 모델로 설계되었다.

SIL은 스택 기반의 명령어 집합으로 언어 독립성과 하드웨어 및 플랫폼 독립성을 갖고 있다. SIL은 다양한 프로그래밍 언어를 수용하기 위해서 바이트코드, .NET IL 등 기존의 가상기계 코드들의 분석을 토대로 정의 되었으며, 객체 지향 언어와 순차적 언어를 모두 수용하기 위한 연산 코드 집합을 갖고 있다.

SIL은 클래스 선언 등 특정 작업의 수행을 나타내는 의사 코드와 실제 명령어에 대응되는 연산 코드로 이루어져 있다[3].

연산 코드는 특정 하드웨어나 소스 언어에 종속되지 않는 추상적인 형태를 지니며, 어셈블리 언어 수준의 디버깅을 용이하게 하기 위해 일관성 있는 이름 규칙을 적용하여 가독성 높은 니모닉으로 정의되어있다. 또한 최적화를 위한 short form 연산 코드를 갖고 있다.

SIL은 7개의 카테고리로 분류할 수 있으며 각각의 카테고리는 서브 카테고리를 갖는다. [그림2]은 SIL의 연산코드 카테고리 이다.

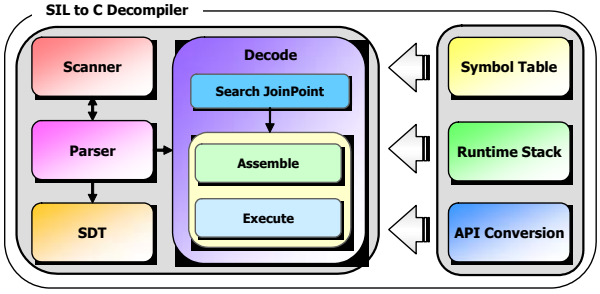


[그림2] SIL의 연산코드 카테고리

3. SIL-to-C 역컴파일러

3.1 시스템 구성도

SIL-to-C 역컴파일러는 SIL 코드와 데이터를 포함한 SAF(Standard Assembly Format)를 입력으로 받아서 어휘 분석과 구문 분석을 한다. 구문 분석이 끝나면 심볼에 대한 정보를 테이블에 저장하고, 심볼의 초기값에 대한 정보는 initial 테이블에 저장한다. 오퍼레이션에 대한 트리를 탐색하면서 가상기계와 같은 방법으로 수행하여 재 표현된 ANSI C 코드를 생성한다. [그림3]는 SIL-to-C 역컴파일러의 시스템 구성도 이다.



[그림3] SIL-to-C 역컴파일러 구성도

3.2 자료구조

SIL-to-C 역컴파일러는 ANSI C 언어의 선언된 변수들과 타입을 위한 심볼 테이블의 자료구조와 연산코드의 실행을 위한 런타임 스택, API 함수를 사용에 따른 헤더 파일 선언을 위해서 API 테이블이 존재한다. 심볼테이블에는 타입의 정보를 유지하기 위해서 symbolTable, concreteTable, abstractTable로 구성되고, 타입 테이블은 typeTable, aggregate Table, memberTable로 구성된다. 런타임 스택은 operatorNumber, operandValue 그리고, operator에 대한 parameter의 정보를 유지하기 위한 필드로 구성된다.

3.3 역컴파일러

역 컴파일 과정은 크게 SearchJoinPoint, Assemble, Execute의 3부분으로 이루어진다.

어셈블 과정에서는 해당 코드에 따른 연산을 수행하고 연산의 결과를 임시변수에 저장한다. 그리고 스택에 관련된 명령어는 실제 런타임 스택에서 스택 연산을 하며, 로드에 관련된 연산자에 대해서는 오퍼런드1(base), 오퍼런드2(offset)를 통해서 심볼테이블을 참조하여 변수의 이름을 런타임 스택의 오퍼런드값에 저장한다.

실행 과정은 해당 연산 코드에 따라 수행하고 해당되는 ANSI C 코드를 생성한다. [표1]는 실행에 관련된 출력 형태를 나타낸 것이다.

[표1] 재 표현된 ANSI C 코드의 출력형태

Store to XXX operations format : operatorName = rhs
Increase/Decrease operations format : tempVarialble = stack[top].operandValue++;

Conditional jump operations
format : tjp - if(operation) goto operand;
fjp - if(!(operation)) goto operand;

Unconditional jump operations format : ujp - goto operand; ret - return; retv.t - return operand;
Function call operations format : tempVariable = operatorName(operation);
Activation record operations format : return_type functionName(arg_type arg_name);

4. 실험 및 결과

다음은 이진 탐색 프로그램으로 ANSI C 컴파일러로 컴파일해서 SIL(*.sil)과 STB(*.stb)를 받아 역컴파일한 것을 보여주는 것이다.

C 프로그램

```
#define _WIN32
#include "C:\Program Files\Microsoft Visual Studio\VC98\Include\stdio.h"
#define N  10
int index_bs(int x, int v[]);
void main() {
        int v[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
        int key = 7;
        printf("array index of %d = %d",key,index_bs(key,v));
}
int index_bs(int x, int v[]) {
        int low, high, mid;
        low = 0;
        high = N - 1;
        while (low <= high) {
        mid = (low + high) / 2;
        if (x < v[mid]) high = mid - 1;
        else if (x > v[mid]) low = mid + 1;
        else return mid;
        }
        return -1;
}
```

Table Information

```
##EXTERNAL
#StringPool
size_twchar_twint_t
...중간생략...
#symbolTable
1 1819 3 0 5 1 8 4 1 312 -1
1 1822 4 0 5 1 12 4 1 313 -1
1 1826 3 0 5 1 16 4 1 314 -1
#concreteTable
4 0 3 -1 0 0 0 0 0 0 0 0 0 0 -1 0 0
4 0 3 -1 0 0 0 0 0 0 0 0 0 0 -1 0 0
4 0 3 -1 0 0 0 0 0 0 0 0 0 0 -1 0 0
#abstractTable
#typeTable
#aggregateTable
#memberTable
```

SIL 코드

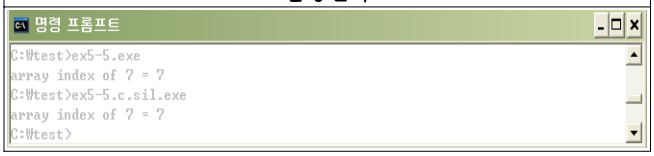
```
%%Data Section
%Structure Table
...중간생략...
%%Code Section
%File : ex5-5.c
main: proc44        1       1
.sym        v       i       0       0       0
            10      40      0x00000000
...중간생략...
%Line 9 : int key = 7;
            ldc.i   7
```

```
        str.i   1   40
%Line   11  :   printf("array index of %d = %d",key,index_bs(key,v));
        ldp
        ldc.p   @0x0000
        lod.i   1   40
        ldp
        lod.i   1   40
        lda     1   0
        call    index_bs
        call    printf
...중간생략...
$$3:    nop
        ujp     $$0
$$1:    nop
        ldc.i   -1
        retv.i
```

Decompile Output

```
#include "ex5-5.c.sil.h"
/* global sym dcl */
// C SourceFile : ex5-5.c
void main() {
/* local sym dcl */
int silSym_0[10]={0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
int silSym_1;
        // C SourceLine(9) : int key = 7;
        silSym_1 = 7;
        // C SourceLine(11) : printf("array index of %d = %d",key,index_bs(key,v));
        tmpInt_0 = silSym_1;
        tmpInt_1 = silSym_1;
        tmpInt_2 = index_bs(tmpInt_1, silSym_0);
        tmpInt_3 = printf("array index of %d = %d", tmpInt_0, tmpInt_2);
}
...중간생략...
$$3:{/* label not operation */}
        goto $$0;
$$1:{/* label not operation */}
        return -1;
}
```

실행결과



5. 결론

EVM은 ANSI C 언어를 멀티 플랫폼에 쉽게 탑재 가능하다. 본 논문은 가상기계 기반의 EVM의 중간언어 SIL 코드를 검증하기 위해서 재 표현된 ANSI C언어로 생성하여 Visual C 컴파일러를 이용해서 실행했다. 따라서 기존의 ANSI C 언어가 SIL 코드로 올바르게 생성되었음을 확인할 수 있었다. 또한, SIL 코드가 어셈블리 형태를 지니므로 코드에 대한 분석에 어려움이 있었다. 하지만, 본 논문에서 제시한 역컴파일러를 통해 생성된 ANSI C 코드는 C 언어 레벨의 분석이 가능하다.

차후로 최적화 코드에 대한 역 컴파일과정을 추가할 예정이다. 그리고 EVM이 순차적인 언어와 객체지향 언어를 수용하는 가상기계이므로 SIL 코드의 객체 지향 관련 코드에 대한 역 컴파일 과정도 추가할 예정이다.

참고문헌

[1] Christopher Fraser & David Hanson, "A Retargetable C Compiler : Design and Implementation", Addison-Wesley, 1995.

[2] Brian W.Kernighan & Dennis M.Ritchie, "The C Programming Language", Prentice Hall, 1988.

[3] 남동근・윤성림・오세만, "가상기계를 위한 어셈블리 언어", 정보처리학회 2003 춘계 학술 발표 논문집, Vol.10, No.1, pp.783-786, Mar. 2003.

[4] 최성규・박진기・이양선, ".NET 언어를 위한 중간 언어 번역기", 멀티미디어학회 2003 추계 학술 발표 대회 논문집, Vol.6, No.2, pp.533-536, Nov. 2003.

[5] 정지훈・박진기・이양선, "자바 언어를 위한 중간 언어 번역기", 멀티미디어학회 2003 추계 학술 발표 대회 논문집, Vol.6, No.2, pp.537-540, Nov. 2003.

[6] 김영근・권혁주・이양선, "구문 트리를 이용한 자바 바이트코드에서 SIL로의 번역기", 정보처리학회 2004 춘계 학술 발표 대회 논문집, Vol.11, No.1, pp.519-522, Mar. 2004.

[7] 권혁주・김영근・이양선, "재목적 Oolong-to-SIL 중간언어 번역기", 멀티미디어학회 2004 춘계 학술 발표 대회 논문집, Vol.7, No.1, pp.310-313, Mar. 2004.

[8] JLCA; Java-Language-to-C# Conversion Assistant, http://www.microsoft.com/korea/press/pressroom/2002/02/02.htm

[9] Halcyon Soft, iNET, http://www.halcyonsoft.com/

[10] Remotesoft, Java.NET, http://www.remotesoft.com/