C++ 컴파일러에서 심벌 테이블의 검증과 분석을 위한 역번역기의 설계 및 구현

손민성*, 권혁주*, 이양선*
*서경대학교 컴퓨터공학과
e-mail : {msson, hjkwon, yslee} @skuniv.ac.kr

Design and Implementation of a Detranslator for Verification and Analysis in C++ Compiler

Min-Sung Son*, Hyeok-Ju Kwon*, Yang-Sun Lee*
*Dept of Computer Engineering, SeoKyeong University

요 약

본 논문에서는 C++ 컴파일러 구현과정에서 객체지향 언어의 속성을 처리하기 위한 역번역기(detranslator)를 설계하고 구현하였다. 구현된 역번역기는 C++ 컴파일러의 선언부 처리 과정에서 심벌 테이블에 입력된 속성들을 본래의 C++ 프로그램으로 역번역 한다. 따라서 C++ 컴파일러 개발 과정에서 설계된 심벌 테이블과 심벌테이블에 입력된 정보가 올바른지 쉽게 검증할 수 있다. 심벌 테이블은 C++ 컴파일러의 어휘 분석과 구문 분석 과정에서 인식되는 명칭(identifier)에 대하여 그 속성(attribute)들을 수집하여 저장하는 자료구조로, 심벌 테이블에 저장된 속성들은 의미분석(semantic analysis) 단계에서 참조된 명칭의 사용이 타당한지 검사하는데 사용 되어 코드 생성(code generation) 단계에서 올바른 코드가 생성 되도록 한다.

본 역번역기를 구현함으로써 심벌 테이블이 올바르게 설계 되었는지 검증할 수 있으며, 컴파일 할 때 심벌 테이블에 필요한 모든 속성이 저장되어 있는지 쉽게 확인 할 수 있게 되었다. 그리고 디버그 정보도 함께 출력되어 객체지향 언어를 위한 컴파일러 개발의 정확성을 기할 수 있다.

1. 서론

C++ 언어는 C 언어의 기능을 확장하여 만든 객체지향 프로그래밍 언어로, C의 특징을 모두 수용하고 있어 시스템 프로그램에 적당할 뿐만 아니라 클래스, 연산자 중복, 상속 등과 같은 객체지향의 특성을 지원하여 여러 응용분야에서 실용적인 언어로 이용되고 있다[1,2].

가상기계는 하드웨어를 대신할 수 있는 소프트웨어이며, 하드웨어-소프트웨어 간에 인터페이스 역할을 하는 일종의 미들웨어로 볼 수 있다. 특히, 임베디드 시스템을 위한 가상기계 기술은 모바일 디바이스, 디지털 TV, 홈 관리 시스템 등에 탑재할 수 있는 핵심 기술로서 다운로드 솔루션에서는 꼭 필요한 소프트웨어 기술이다[3,4,5].

EVM(Embedded Virtual Machine)은 ANSI C/C++ 언어와 SUN사의 Java 언어 등을 모두 수용할 수 있는 임베디드 가상기계이고, SIL(Standard Intermediate Language)은 EVM의 중간언어로 ANSI C/C++ 언어, Java 언어 등으로 작성된 프로그램을 변환한 어셈블리 언어 형태이다. SIL은 다양한 프로그래밍 언어를 수용하기 위해서 기존의 가상기계 어셈블리 언어들의 분석을 토대로 정의 되었으며, 객체지향 언어와 순차적 언어를 모두 수용하기 위한 연산 코드 집합을 갖고 있다[8].

본 연구팀은 EVM을 위한 C++ 컴파일러를 개발하였으며, 컴파일러는 중간언어로 SIL을 생성한다. 개발된 C++ 컴파일러는 어휘 분석과 구문 분석 과정에서 인식되는 명칭에 대해서 그 속성(attribute)들을 심벌 테이블에 저장하였다. 심벌 테이블에 수집된 속성들은 의미 분석(semantic analysis) 단계에서 참조된 명칭의 사용이 타당한지 검사하는데 사용되며, 코드 생성(code generation) 단계에서 올바른 코드를 생성하게 한다.

본 논문에서는 객체지향 언어를 위한 컴파일러 개

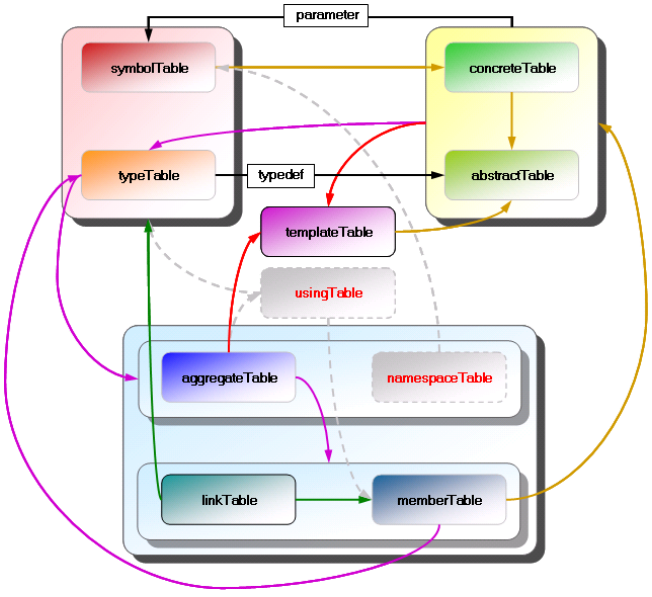
발과정에서 심벌 테이블의 설계와 심벌 테이블에 삽입된 정보들이 올바른지를 검사하기 위해 심벌 테이블에 저장되어 있는 정보를 이용하여 C++ 프로그램으로 복원하는 역번역기를 구현하였다.

2. 심벌 테이블

심벌 테이블이란 심벌(또는 명칭)들에 관한 정보를 관리하고 운영하는 데 사용되는 자료구조이다. 컴파일러에서는 어휘 분석과 구문 분석 과정에서 인식되는 명칭에 대해서 그 속성들을 수집하여 심벌 테이블에 저장하며, 의미 분석(semantic analysis) 단계에서는 테이블에 수집된 속성과 참조된 명칭의 사용이 타당한지를 검사하고, 코드 생성(code generation) 단계에서는 속성을 이용하여 올바른 코드를 생성 하도록 하는 역할을 한다.[6,7]

2.1 테이블 구성

심벌 테이블은 Symbol Table, Concrete Table, Abstract Table, Type Table, Aggregate Table, Member Table, Link Table, Template Table, Namespace Table, Using Table 로 구성되어 있다. [그림 1]은 구성된 테이블의 관계도 이다.



[그림 1] 테이블 관계도

선언부에서 변수가 선언되면 일반 변수는 symbol table에 삽입된다. symbol table은 기본적으로 변수의 이름과 스코프 레벨, 상대주소를 가지게 되며 concrete table을 참조한다. 이때 파라미터를 가진 함수의 경우 다시 symbol table을 참조한다.

사용자 정의 타입이 선언 된 경우 type table에 삽입되고 type table은 타입의 이름과 상대주소, 크기 등을 가지게 된다. type table에 입력된 타입이 class, struct, union, enum, namespace 인 경우 aggregate table을 참조하게 되며 typedef인 경우 abstract table을 참조 한다.

aggregate table은 타입의 종류와 타입에 속한 멤버의 member table 인덱스와 멤버의 개수 그리고 상속 받은 클래스가 있을 경우 link table 인덱스와 상속받은 클래스의 개수를 가진다.

member table은 멤버 변수의 이름과 상대 주소, 접근 권한 정보를 가지게 되며, 이 멤버가 namespace에 속해 있을 경우 concrete table을, 다른 타입에 속해 있을 경우 abstract table을 참조한다.

concrete table에는 변수의 타입, 초기값, 파라미터, abstract table의 인덱스 등의 속성이 삽입되고, abstract table에는 concrete table과 같은 내용이 삽입되지만 초기값에 대한 저장 공간은 할당 되어있지 않다.

template table은 템플릿의 종류와 템플릿 파라미터의 속성 등을 삽입하고 abstract table을 참조한다.

using table은 using 타입과 선언 시점의 스코프가 삽입 되며 선언된 타입에 따라 symbol table, type table, member table 인덱스를 참조한다.

2.2 해시 심벌 테이블

심벌 테이블의 기본적인 기능은 테이블에 심벌을 삽입하고 검색하는 일이며, 이런 조작이 컴파일 하는 시간의 많은 양을 차지하기 때문에 효과적인 테이블 구성은 중요한 문제가 된다.

본 연구팀은 심벌 테이블에 삽입된 심벌들의 검색 시간을 최소화시키기 위하여 해시 심벌 테이블을 사용하였다. 해시 심벌테이블은 해시 버켓(hash bucket)과 심벌 테이블로 구성되어 있으며 해시 버켓의 내용은 심벌 테이블의 인덱스이다. 해시 함수(hash function)은 심벌에 대한 주소를 직접 계산하기 위한 함수이다. 본 논문에서는 해시 함수 중 제산법을 이용하여 심벌을 수로 표현하여 버켓의 크기로 나누고 그 나머지를 해시 값으로 취하는 방법을 이용한다.

그러나 해시 함수들은 모든 심벌에 대해 고유의 위치를 갖도록 분산시켜야 하며, 심벌에 대한 함수를 계산하는 시간이 적게 걸려야 하지만 이러한 요건은 완전히 갖추어지기 어렵다. 따라서 서로 다른 심벌이 같은 해시 값을 갖게 되는 충돌(collision)현상을 야기하게 된다. C++ 컴파일러에서는 이와 같은

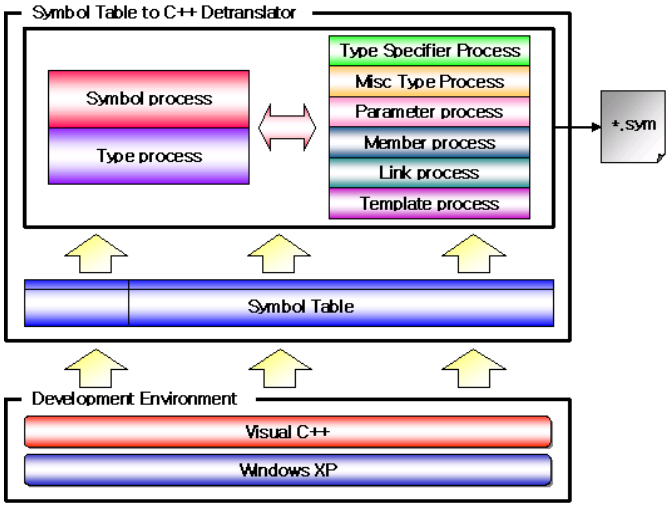
현상을 연쇄법을 사용하여 해결한다. 연쇄법 중 후방법(backward method)은 해시 버켓의 값이 새로 삽입된 명칭 레코드를 가리키게 하고 새로운 레코드의 포인터가 기존의 해시 버켓이 가리키고 있던 레코드를 가리키게 하는 방법이다.

3. 역번역기 시스템

심벌 테이블 역번역기 시스템은 모든 C++ 언어의 선언부를 입력 받아 심벌 테이블에 삽입한 후, 삽입된 심벌 테이블의 정보를 이용하여 C++ 프로그램으로 복원하는 역번역기이다.

3.1 역번역기 시스템 구성

역번역기 시스템의 구성은 심벌 테이블(Symbol Table)과 이를 이용해 C++ 프로그램으로 역번역을 수행하는 심벌 프로세스(Symbol Process)와 타입 프로세스(Type Process)로 구성되어 있다. 그리고 각각의 심벌 프로세스와 타입 프로세스는 멤버 프로세스(Member Process), 파라미터 프로세스(Parameter Process), 링크 프로세스(Link Process), 템플릿 프로세스(Template Process), Type Specifier Process(이하 TSP), Misc Type Process(이하 MTP)등의 서브 프로세스로 구성된다. [그림 2]는 역번역기 시스템의 구성도이다.



[그림 2] 역번역기 시스템 구성도

역번역기 시스템은 심벌 테이블의 정보들 중에서 선언부에 해당하는 정보만을 추출하여 역번역하게 되는데, 크게 심벌 프로세스와 타입 프로세스로 이루어진다. 심벌 프로세스는 파라미터 프로세스, TSP, MTP로 구성되어, 심벌 테이블에서 외부 변수에 해당하는 모든 정보만을 검색하여 역번역하는 부분이다. 검색된 정보를 TSP를 통하여 변수의 타입을 명세화하고 MTP를 통해 배열인지, 함수인지, 일반 변수인지를 판별하여 C++ 프로그램으로 역변역하여 출력한다. 이때 함수의 파라미터가 있다면 파라미터 프로세스를 추가로 수행하여 파라미터 부분을 역번역하여 출력한다. 타입 프로세스는 멤버 프로세스, 링크 프로세스, TSP, MTP로 구성되어, 타입 테이블에서 타입 정보를 검색하여 class, struct, union, enum, namespace, typedef 타입을 각각 구분하여 역번역하게 된다. 멤버 프로세스는 타입이 class, struct, union, namespace일 경우 수행하게 되는 과정으로 해당 타입에 속해있는 멤버 변수와 함수들을 심벌프로세스와 마찬가지로 TSP, MTP, 파라미터 프로세스를 통해 C++ 프로그램으로 역번역한다. 또한 멤버에 중첩 클래스가 존재할 경우 멤버 프로세스는 이를 역번역 하기위해 타입 프로세스를 재귀호출 하게 된다. 링크 프로세스는 타입 정보가 클래스와 구조체일 경우 가지게 되는 특성인 상속과 friend를 역번역하여 출력한다. 템플릿 프로세스는 C++ 언어의 중요한 특징 중 하나인 템플릿을 역번역하기 위한 부분으로, 템플릿을 사용한 함수는 심벌 프로세스에서 템플릿을 사용한 클래스는 타입 프로세스에서 템플릿 프로세스를 호출하여 역번역한 후 출력한다.

3.2 역번역기 시스템 구현

역번역기는 심벌 테이블과 타입 테이블을 순차적으로 탐색하면서 테이블의 정보를 이용하여 C++ 프로그램으로 역번역한다. [그림 3]은 역번역기 시스템을 구성하는 알고리즘의 개요이다.

```
void dumpSymbol(const char * name, std::string &returnString)
{
  while(symbolTableIndex > index)
  {
    name lookup

    if(PARAMETER or NULL_SYMBOL)
      skip this index
    else
      // detranslate the declaration part
        for external variable and function
      dumpSymbolProcess(index, outputBuffer);
  }
}

void dumpType(const char * name, std::string &returnString)
{
  while(typeTableIndex > index)
  {
    name lookup

    if(system define typeName or NULL_TYPE)
      skip this index
    else
      // detranslate the declaration part for extenal type
      dumpTypeProcess(index, outputBuffer);
  }
}
```

[그림 3] 역번역기 시스템 알고리즘의 개요

4. 실행 결과 및 분석

다음은 C++ 언어의 선언문에 대해서 심벌 테이블에 속성을 삽입한 후 역번역기를 통하여 C++ 프로그램으로 복원시킨 예제이다. [예제 1]은 입력으로 받을 C++ 프로그램의 선언부이다.

```
extern int a;                       // declares a
extern const int c;                 // declares c
int f(int);                         // declares f
typedef int Int;                    // declares Int
enum p{ up, down };
class tnode {
public:
        unsigned char tag: 1;
        char tword[20];
        int count;
        tnode *left, *right;
        void set(char*, tnode* l, tnode* r);
        tnode();
        ~tnode();
};
template<class T> class Array
 : public tnode
{
        T* v;
        int sz;
public:
        T& operator[](int);
        T& elem(int i) { }

};
```

[예제 1] C++ 프로그램의 선언부

[예제 2]는 심벌 테이블의 속성을 이용해 역번역기로 생성한 C++ 선언부와 디버그 정보를 함께 나타낸 것이다.

```
// *                            SYMBOL TABLE DUMP
// ************************************************************
/* index: #1, base: #0 */
extern int a;
/* index: #2, base: #0 */
extern const int c;
/* index: #4, base: #0 */
int f(int ##P001);


// ************************************************************
// *                            TYPE TABLE DUMP
// ************************************************************
/* index: #31, base: #0 */
typedef int Int;


/* index: #32, base: #0 */
enum p {
        up=0, down=1
};


/* index: #33, base: #0 */
class tnode {
public:
        unsigned char tag:1;    // memberID:1
        char tword[20]; // memberID:2
        int count;      // memberID:3
        tnode* left;    // memberID:4
        tnode* right;   // memberID:5
        void set(char* ##P002, tnode* ##arg6, tnode* ##arg7);
        tnode();        // memberID:2
        ~tnode();       // memberID:3
};


/* index: #34, base: #0 */
template <class T>
class Array : public tnode {
        T* v;   // memberID:6
        int sz; // memberID:7
public:
        T& operator [](int ##P003);     // memberID:4
        T& elem(int ##arg9);    // memberID:5
};
```

[예제 2] 역번역기에 의해 복원된 C++ 프로그램

5. 결론 및 향후연구

본 논문에서는 심벌 테이블에 저장된 명칭의 속성과 참조된 명칭의 사용이 올바른지를 검사하고, 코드 생성 단계에서 심벌 테이블의 정보를 이용하여 올바른 코드를 생성할 수 있는지 알아보기 위하여 심벌 테이블에 있는 정보만을 가지고 다시 C++ 프로그램으로 복원시키는 역번역기를 구현하였다. 본 역번역기를 구현함으로써 심벌 테이블이 올바르게 설계 되었는지 검증할 수 있으며, 컴파일 할 때 필요한 속성이 모두 저장되어 있는지 쉽게 확인 할 수 있다. 또한 디버그 정보도 함께 출력하여 C++ 컴파일러의 수정이 더욱 용이해졌다.

앞으로 심벌 테이블에 저장된 정보가 더 많은 디버그 정보를 출력할 수 있게 하고, 이 정보를 이용하여 EVM을 위한 C++ 컴파일러의 수정을 용이하게 하도록 역번역기를 보완할 예정이다.

[참 고 문 헌]

[1] Bjarne Stroustrup, "The C++ Programming Language", Addison-Wesley, 2000

[2] INTERNATIONAL STANDARD ISO/IEC 14882:1998(E) "Programming Language - C++", ISO/IEC, 1998

[3] 최성규・정지훈・이양선, "임베디드 시스템을 위한 C# MSIL 코드의 Oolong 코드 번역에 관한 연구",한국정보처리학회 춘계학술발표논문집, Vol.10, No.1 pp.983-986, Mar. 2003

[4] 정지훈・박진기・이양선, "자바 언어를 위한 중간 언어 번역기", 한국멀티미디어학회 2003 추계학술발표논문집, Vol.6, No.2, pp.537-540, Nov. 2003

[5] 김영근・권혁주・이양선, "자바 바이트코드의 EVM SIL 코드 매핑", 한국멀티미디어학회 2004 추계학술발표논문집, Vol.7, No.2, pp.915-918, Nov. 2004

[6] 권혁주・김영근・박진기・이양선, "ANSI C 컴파일러를 위한 심벌 테이블로부터 C 프로그램 역번역기의 개발", 한국멀티미디어학회 2005 춘계학술발표논문집, Vol.8, No.1, pp.69, May. 2005

[7] 손민성・배성균・이양선, "C++ 컴파일러를 위한 심벌 테이블 역번역기의 설계 및 구현", 한국멀티미디어학회 2005 추계학술발표논문집, Vol.8, No2, pp.181, Nov. 2005

[8] 김영근・권혁주・박진기・이양선, "임베디드 가상기계를 위한 번역기 시스템", 한국멀티미디어학회 2005 추계학술발표논문집, Vol.8, No.2, pp180, Nov. 2005