제26회 한국정보처리학회 추계학술발표대회 논문집 제13권 제2호 (2006. 11)

객체지향 컴파일러의 심벌 테이블 검증을 위한 선언문 복원 기법

손민성*, 권혁주**, 김영근**, 이양선*
*서경대학교 컴퓨터공학과
**(주) 넥솔텔레콤 연구소
e-mail:{msson, yslee}@skuniv.ac.kr
{sputnik, carople}@nexoltelecom.com

The Declarations Reconstruction Technique for the Symbol Table Verification of the Object-oriented Compiler

Min-Sung Son*, Hyeok-Ju Kwon**, Young-Keun Kim**, Yang-Sun Lee*
*Dept of Computer Engineering, SeoKyeong University
**Research Institute, Nexol Telecom

요 약

본 연구팀은 유비쿼터스 게임 플랫폼을 위한 Embedded C++ 컴파일러를 개발하였으며, 컴파일러 개발 과정에서 객체지향언어인 C++과 Java 언어를 모두 수용할 수 있는 심벌 테이블을 설계하였다. 심벌 테이블은 컴파일러의 어휘 분석과 구문 분석 과정을 거친 후 SDT(syntax-directed translation)에 의해 생성된 AST(Abstract Syntax Tree)를 분석하여 인식된 명칭(identifier)과 그 속성(attribute)들을 수집하여 저장하는 자료구조로써, 심벌 테이블에 저장된 속성들은 의미 분석(semantic analysis) 단계에서 수집된 속성과 참조된 명칭의 사용이 타당한지를 검사하고, 코드 생성(code generation) 단계에서 올바른 코드가 생성되도록 하는 중요한 요소이다. 따라서 심벌 테이블의 설계가 올바른지와 입력된 속성이 정확한지에 대한 검증과 분석은 필수 불가결하다. 본 논문에서는 컴파일러 개발과정에서 설계한 심벌 테이블을 검증하고 분석하기 위한 목적으로써, 심벌 테이블을 이용하여 선언문을 복원시키는 역번역기(detranslator)에 대하여 기술한다. 구현된 역번역기는 C++ 컴파일러와 Java 컴파일러의 선언문 처리 과정에서 심벌 테이블에 입력된 속성들을 본래의 입력 프로그램으로 역번역한다. 따라서 역번역기를 통하여 심벌 테이블의 완전성과 심벌 테이블에 입력된 속성 정보의 정확성을 쉽게 검증할 수 있으며, 역번역과 함께 출력되는 디버그 정보를 이용하여 효율적으로 컴파일러의 개발과 수정을 할 수 있다.

1. 서론

최근 프로그램 개발자의 70~80%가 고급언어인 C/C++ 그리고 Java 언어를 프로그래밍 언어로 사용하고 있다. 특히 C++ 언어는 C언어의 기능을 확장하여 만든 객체 지향형 언어로, C의 특징을 모두 수용하고 있어 시스템 프로그램에 적당할 뿐만 아니라 클래스, 함수 오버로딩, 상속, 다형성, 정보 은닉 등과 같은 객체지향의 특성을 지원하여 여러 응용분야에서 실용적으로 사용되고 있다[1].

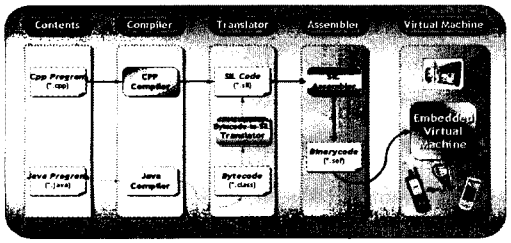
하지만, 기존의 컴파일러들은 C/C++ 언어로 작성된 프로그램을 목적 코드로 번역하여 실행하는 구조를 지니고 있다. 따라서 C/C++ 언어로 작성된 프로그램을 해당 플랫폼에 적합하게 수정해야 하고 플랫폼 마다 수행 가능한 목적 코드로 번역하는 컴파일러가 각각 존재해야 한다. 또한, 실행되는 플랫폼마다 목적코드가 다르기 때문에 코드의 재활용성과 이식성이 떨어진다[5,6,7].

본 연구팀은 이러한 문제점들을 해결하기 위하여 컴파일러 제작 기술과 가상기계 기술을 기반으로 플랫폼이 변경되더라도 작성된 프로그램을 수정 없이 실행시킬 수 있는 유비쿼터스 게임 플랫폼을 위한 C++ 컴파일러와 가상기계 EVM을 개발하였다. 본 논문에서는 C++ 컴파일러 개발과정에서 설계된 심벌 테이블을 검증하고 분석하기 위하여 심벌 테이블에 입력된 속성들을 본래의 프로그램으로 다시 복원 시켜주는 역번역기에 대하여 기술한다.

2. 관련 연구

2.1 EVM (Embedded Virtual Machine)

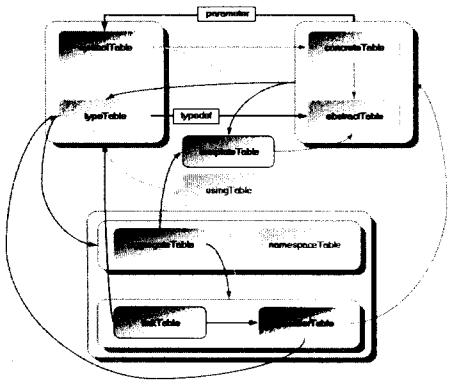
유비쿼터스 게임 플랫폼을 위한 EVM은 모바일 디바이스, 셋톱박스, 디지털 TV등에 탑재되어 동적인 응용 프로그램을 다운로드 하여 실행할 수 있는 가상 기계이다. 또한 콘텐츠 개발을 용이하게 하기 위해서 다양한 언어를 지원하고 언어 간의 통합이 가능하다[6,7].



(그림 1) 유비쿼터스 게임 플랫폼의 시스템 구성도

2.2 Symbol Table

심벌 테이블이란 심벌(또는 명칭)들에 대한 바인딩 정보와 영향 범위를 관리하고 운영하는데 사용되는 자료구조이다. 컴파일러에서는 어휘 분석과 구문 분석을 거친 후 SDT에 의해 생성된 추상 구문 트리를 분석하여 인식되는 명칭에 대해서 그 속성들을 수집하고 참조하기 위해서 사용한다. 이 속성들은 명칭이 정의되는 곳에서 심벌 테이블에 수집되며, 의미 분석단계에서는 테이블에 수집된 속성과 참조된 명칭의 사용이 타당한지를 검사하고, 코드 생성단계에서는 속성을 이용하여 올바른 코드를 생성하도록 하는 역할을 한다. 다음 (그림2)는 구성된 심벌 테이블과 테이블간의 연관 관계를 도식한 것이다[2,3,4].



(그림 2) 심벌 테이블 관계도

Symbol Table은 일반 변수와 함수 선언문들에 대한 이름과 영향 범위 그리고 상대 주소를 저장하기 위한 테이블이며, Type Table은 클래스, 구조체, 공용체 등과 같은 사용자 정의 타입에 대한 이름과 크기, 타입의 종류들을 저장하기 위한 테이블이다.

Storage Table은 Concrete Table과 Abstract Table로 구성되어있다. 이들은 주로 변수의 타입이나 함수의 리턴 타입 및 파라미터 정보를 저장하기 위한 테이블이며 Concrete Table은 Abstract Table과는 달리 초기 값을 저장하기 위한 추가의 필드가 할당 되어 있다.

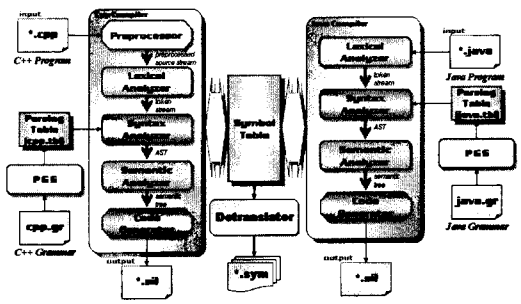
사용자 정의 타입 테이블은 Aggregate Table과 Member Table, Link Table로 구성되어 있다. 이들 테이블에는 상속 정보, 멤버의 개수, 멤버 변수와 멤버 함수의 이름, 멤버 변수의 상대주소 그리고 접근 권한 등의 정보가 저장된다.

Template Table은 C++의 함수 템플릿과 클래스 템플릿에 사용된 템플릿 매개변수를 저장하기 위한 공간으로, 각 매개 변수에 대한 정보와 초기 값 및 특수화(Specialization)과정에 필요한 정보가 저장된다.

3. 역번역기 시스템

3.1 역번역기 시스템의 개요

역번역기 시스템은 C++/Java로 작성된 파일(*.cpp, *.java)로부터 선언문들을 심벌 테이블에 삽입하고 심벌 테이블에 삽입된 정보를 이용하여 본래의 프로그램 선언문으로 다시 복원한다.



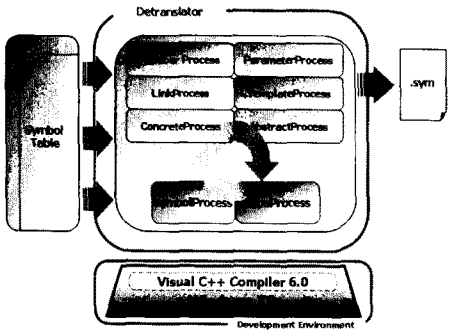
(그림 3) 선언문 역번역기 개요도

3.2 역번역기 시스템의 구성

역번역기 시스템은 크게 심벌 테이블(Symbol Table)과 심벌 테이블의 정보를 바탕으로 하여 역번역을 수행하는 심벌 프로세스(Symbol Process)와

타입 프로세스(Type Process)로 구성되어 있다.

역번역기의 심벌 프로세스와 타입 프로세스는 선언된 변수 또는 타입에 따라 서브 프로세스(sub-process)인 멤버 프로세스(Member Process). 파라미터 프로세스(Parameter Process). 링크 프로세스(Link Process). 템플릿 프로세스(Template Process), 컨크리트 프로세스(Concrete Process), 앱스트랙트 프로세스(Abstract Process)를 호출하여 역번역을 수행 한다. 다음 (그림 4)는 역번역기 시스템의 구성도 이다.

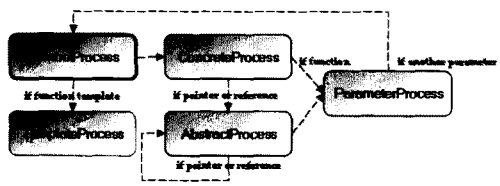


(그림 4) 역번역기 구성도

3.3 역번역기 시스템의 구현

본 역번역기는 Window XP 환경 하에 Visual C++ 6.0을 이용하여 구현하였다.

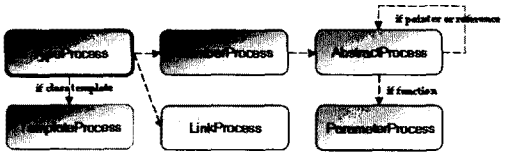
심벌 프로세스는 심벌 테이블에서 외부 변수와 함수에 해당하는 모든 정보를 추출하여 역번역하는 모듈이다. 추출된 정보는 컨크리트 프로세스에서 변수의 자료형(data type)을 명세화하고 배열인지 함수인지 일반 변수인지를 판별하여 역번역을 하게 된다. 만약 변수의 자료형이 포인터 또는 레퍼런스 타입일 경우 앱스트랙트 프로세스를 호출 하여 타겟 타입에 대한 역번역을 수행 한다. 그리고 파라미터를 가지는 함수의 경우 파라미터 프로세스를 추가로 수행하여 파라미터 부분을 역번역한다. 다음 (그림 5)는 심벌 프로세스의 흐름도이다.



(그림 5) 심벌 프로세스의 흐름도

타입 프로세스는 함수나 변수 등을 처리하는 심벌 프로세스와는 달리 사용자 정의 타입에 대한 역번역을 수행하는 모듈이다. 타입 프로세스는 타입 테이블에서 자료형이 사용자 정의 타입인지 *typedef* 타입인지를 먼저 구별한다.

그리고 자료형이 사용자 정의 타입일 경우 *class, struct, union, enum*중 어떤 것인지를 판단한 후 상속 여부를 확인한다. 상속을 받았을 경우 링크 프로세스를 호출하여 상속에 관련된 부분을 역번역한 후에 멤버 프로세스를 호출하여 타입들이 지니고 있는 각각의 자신들의 멤버 변수와 멤버 함수에 대한 역번역을 수행하게 된다. *typedef* 타입일 경우에는 바로 앱스트랙트 프로세스를 호출하여 역번역을 수행한다. 다음 (그림 6)은 타입 프로세스의 흐름도이다.



(그림 6) 타입 프로세스의 흐름도

템플릿 프로세스는 C++의 함수 템플릿과 클래스 템플릿을 역번역하는 부분으로 함수 템플릿의 경우에는 심벌 프로세스에서 호출이 되고 클래스 템플릿일 경우 타입 프로세스에서 각각 호출되어 템플릿 매개 변수 부분을 역번역한다.

4. 실험 결과 및 분석

다음은 C++ 선언문에 대해서 심벌 테이블에 속성들을 삽입한 후 역번역기를 통하여 C++ 프로그램으로 복원하는 예제이다.

```
char ch;
int count;
const double pi;
extern int error_number;
struct Date { int d,m,y; };
int day(Date*p);
template<class T> T abs(T a);
enum Beer { Carlsberg, Tuborg, Thor };
namespace NS { int a; }
template<class T>
class D_Day : public Date
{
        T* day;
        public:
        T& operator+(T*);
        T& get_D_day(int) const;
}
```

```
// ================================================
//                  SYMBOL TABLE DUMP              =
// ================================================
/* index: #1, base: #0 */
char ch;
/* index: #2, base: #0 */
int count;
/* index: #3, base: #0 */
const double pi;
/* index: #4, base: #0 */
extern int error_number;
/* index: #6, base: #0 */
int day(Date* ##arg5);
/* index: #8, base: #0 */
template <class I>
I abs(I ##arg7);
// ================================================
//                   TYPE TABLE DUMP               =
// ================================================
/* index: #31, base: #0 */
struct Date {
        int d;  // memberID:1
        int m;  // memberID:2
        int y;  // memberID:3
};
/* index: #32, base: #0 */
enum Beer {
        Carlsberg-0, Tuborg-1, Thor-2
};
/* index: #33, base: #0 */
namespace NS {
        int a ; // memberID:0
};
/* index: #34, base: #0 */
template <class T>
class D_Day : public Date {
        T* day; // memberID:4
public:
        T& operator +(T* ##P001);       // memberID:1
        T& get_D_day(int ##P002) const; // memberID:2
};
```

[예제 1] 입력된 C++ 선언문과 복원된 C++ 선언문

다음은 Java 선언문에 대해서 심벌 테이블에 속성들을 삽입한 후 역번역기를 통하여 Java 프로그램으로 복원하는 예제이다.

```
class AnsiC{
        int declarations;
        int operators;
        int statements;
        void functions(){ }
}
class Java extends AnsiC {
        int classes;
        int exceptions;
        int threads;
        void functionsOverloadings(){ }
}
public class Languages {
    public static void main(String[] args) { }
}
```

```
/* index: #2, base: #0 */
class AnsiC {
        int declarations;
        int operators;
        int statements;
        void functions(){
        }

/* index: #3, base: #0 */
class Java extends AnsiC {
        int classes;
        int exceptions;
        int threads;
        void functionsOverloadings(){
        }

/* index: #4, base: #0 */
public class Languages {
        public static void main(String[] args){
        }
```

[예제 2] 입력된 Java 선언문과 복원된 Java 선언문

5. 결론 및 향후 연구

본 논문에서는 C++ 컴파일러 개발 과정에서 설계된 심벌 테이블의 검증과 분석을 위한 역번역기를 설계하고 구현하였다. 역번역기는 심벌 테이블에 있는 정보만을 가지고 다시 본래의 프로그램으로 복원시키는 작업을 한다. 따라서 설계된 심벌 테이블의 완전성과 심벌 테이블에 저장된 명칭의 속성 정보가 올바른지 쉽게 검증하고 분석할 수 있다. 그리고 검증된 심벌 테이블을 기반으로 심벌 테이블에 저장된 명칭의 속성과 참조된 명칭의 사용이 올바른지를 검사하고 코드 생성 단계에서 올바른 코드를 생성할 수 있게 되었다. 또한 역번역기는 프로그램 역번역뿐만 아니라 디버그 정보를 출력함으로써 컴파일러의 수정을 용이하게 해준다.

향후 역번역기가 현재 일부를 수용하고 있는 자바 언어를 모두 수용할 수 있도록 계속해서 확장할 것이며, 심벌테이블의 정보를 이용하여 더 많은 디버그 정보를 출력할 예정이다.

참고문헌

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

[2] Dewhurst, Stephen C., "FLEXIBLE SYMBOL TABLE STRUCTURES FOR COMPILING C plus plus.", Software - Practice and Experience , Volume 17, Is-sue 8, August 1987, Pages 503-512

[3] Gallego-Carrillo, M, Gortázar-Bellas, F., Urquiza-Fuentes,J. and Velázquez-Iturbide, JÁ. "SOTA: a visualization tool for symbol tables." ACM SIGCSE Bul-letin, 37, 3(Sept, 2005)385.

[4] 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

[5] 김영근·권혁주·이양선, "EVM SIL에서 C 프로그램 생성을 위한 역컴파일러의 설계 및 구현", 한국정보처리학회 2005 춘계 학술 발표 대회 논문집, Vol.12, No.1, pp.549-552 ,Mar. 2005.

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

[7] 이 양선·박 진기, "Translation Java Bytecode to EVM SIL Code for Embedded Virtual Machine", 한국멀티미디어학회 논문지, Vol8, No.12, pp.827 - 835, Dec 2005.