프로그램 선언부를 위한 심벌테이블에서 C 프로그램 역번역기의 설계 및 구현

권혁주*, 김영근*, 이양선*, 오세만**
*서경대학교 컴퓨터공학과
**동국대학교 컴퓨터공학과
*e-mail : {hjkwon, ykkim, yslee} @pl.skuniv.ac.kr
**e-mail : smoh@dgu.ac.kr

Design and Implementation of C Program Detranslator from Symbol Table for Program Declaration Part

Hyeok-Ju Kwon*, Young-Koun Kim*, Yang-Sun Lee*, Se-Man Oh**
*Dept of Computer Engineering, SeoKyeong University
**Dept of Computer Engineering, DongGuk University

요 약

ANSI C 언어는 UNIX 시스템에서 뿐만 아니라 DOS 환경에서 수행되는 C 컴파일러와 각종 지원 도구가 개발되어 보급됨으로써 오늘날 널리 사용되는 범용 프로그래밍 언어 중 하나이다. EVM(Embedded Virtual Machine)은 ANSI C 언어와 SUN사의 Java 언어 등을 모두 수용할 수 있는 임베디드 시스템을 위한 가상 기계이며, SIL(Standard Intermediate Language)은 EVM에서 실행되는 중간언어로 다양한 프로그래밍 언어를 수용하기 위해서 객체지향 언어와 순차적 언어를 모두 수용하기 위한 연산 코드 집합을 갖고 있다. EVM을 위한 ANSI C 컴파일러는 ANSI C 언어를 받아 들여 EVM의 중간 언어인 SIL 코드를 출력한다.

ANSI C 컴파일러에서 어휘 분석과 구문 분석 과정에서 인식되는 명칭에 대해서 그 속성들을 수집하고 이용한다. 이 속성들은 명칭이 명시적으로 혹은 묵시적으로 정의되는 곳에서 심벌 테이블에 수집된다. 본 논문에서는 수집된 정보가 올바르게 되었는지 확인하기 위하여 심벌 테이블에 있는 정보를 다시 ANSI C 언어로 복원시키는 역변역기(detranslator)를 구현하였다.

1. 서론

ANSI C 언어는 이식성이 매우 높아 한 시스템에서 다른 시스템으로 별다른 수정 없이 쉽게 이식될 수 있으며 효율성 위주로 설계되어 프로그래머의 생산성을 향상 시킨다. EVM(Embedded Virtual Machine)은 ANSI C 언어와 SUN사의 Java 언어 등을 모두 수용할 수 있는 임베디드 가상기계이고, SIL(Standard Intermediate Language)은 EVM의 중간언어이다. ANSI C 언어, Java 언어 등으로 작성된 프로그램을 어셈블리 언어 형태인 *.sil 파일로 변환한 후에, EVM에서 실행 시킨다[5,6].

EVM을 위한 ANSI C 컴파일러는 어휘 분석과 구문 분석 과정에서 인식되는 명칭에 대해서 그 속성(attribute)들을 수집하고 이용할 필요성을 갖는다. 대개 이 속성들은 명칭이 명시적으로 혹은 묵시적으로 정의되는 곳에서 심벌 테이블에 수집되며, 의미 분석(semantic analysis) 단계에서는 테이블에 수집된 속성과 참조된 명칭의 사용이 타당한지를 검사하고, 코드 생성(code generation) 단계에서는 속성을 이용하여 올바른 코드를 생성하도록 하는 역할을 한다.

본 논문에서는 심벌 테이블 구조가 올바른지에 대한 판별과 심벌 테이블에 삽입한 모든 정보들이 올바른지를 판별하기 위해 심벌 테이블에 있는 정보를 다시 ANSI C 언어의 선언부로 변환하는 역변역기(detranslator)를 구현하였다.

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

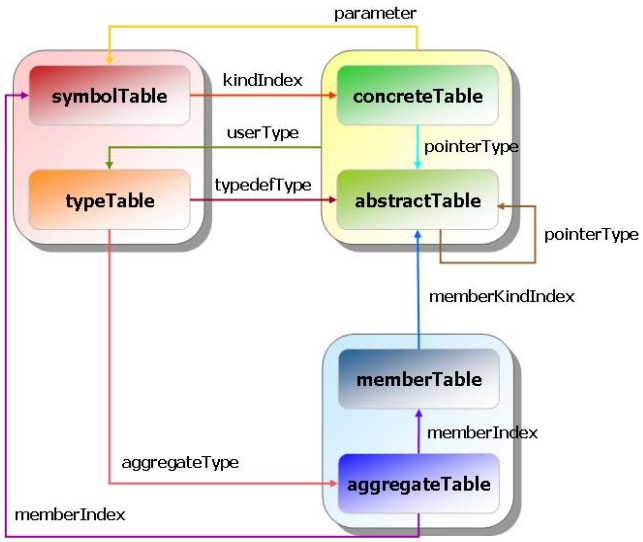
2. 심벌 테이블

2.1 테이블 관계도

심벌 테이블은 Symbol Table, Concrete Table,

Abstract Table, Type Table, Aggregate Table, Member Table로 구성되어 있다. 선언부에서 변수가 선언되면 symbol Table에 변수의 이름, 레벨, 상대 주소, concrete Table의 인덱스 등의 속성이 삽입되고, concrete Table에는 변수의 타입, 초기값 , 파라미터, abstract Table의 인덱스 등의 속성이 삽입된다. abstract Table에는 concrete Table과 같은 내용이 삽입되지만 초기값에 대해서는 삽입하지 않는다. 선언부에서 타입이 선언되면 type Table에 타입의 이름, 상대 주소, 길이 등의 속성이 삽입된다. type Table에서 typedef로 선언한 타입의 경우에는 abstract Table의 인덱스를 가지게 되며, 구조 타입일 경우에 대해서는 aggregate Table의 인덱스를 가지게 된다. aggregate Table에는 멤버의 개수와 member Table의 인덱스를 삽입한다. member Table에는 멤버의 이름과 상대 주소, 이름, abstract Table의 인덱스를 삽입한다.

[그림 1]은 테이블의 관계도를 나타낸 것이다.



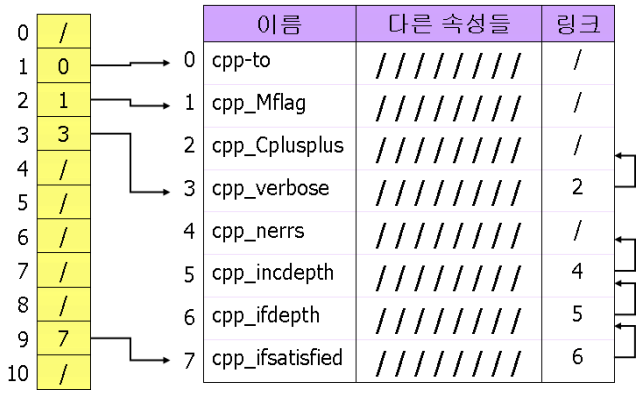
[그림 1] 테이블의 관계도

2.2 해시 심벌 테이블

심벌 테이블의 기본적인 기능은 테이블에 심벌을 삽입하고 검색하는 일이며, 이런 조작이 컴파일하는 시간의 많은 양을 차지하기 때문에 효과적인 테이블 구성은 중요한 문제가 된다. 이러한 문제를 해결하기 위하여 해시 심벌 테이블을 이용한다. 해시 심벌 테이블은 해시 버켓(hash bucket)과 심벌 테이블로 구성되어 있으며 해시 버켓의 내용은 심벌 테이블의 인덱스이다. 해시 함수(hash function)는 심벌에 대한 주소를 직접 계산하기 위한 함수이다. 해시 심벌 테이블은 지금까지 알려진 테이블 구조 중 가장 좋은 검색법이다. 본 논문에서는 해시 함수 중 제산법을 이용하여 심벌을 수로 표현하여 버켓의 크기로 나누고 그 나머지를 해시 값으로 취하는 방법을 이용한다.

그러나 해시 함수들이 갖추어야 할 요건은 모든 심벌에 대해 고유의 위치를 갖도록 분산시켜야 하며, 심벌에 대한 함수를 계산하는 시간이 적게 걸려야 한다. 이 요건은 완전히 갖추어질 수 없기 때문에 심벌들에 대한 해시 값이 고유의 위치를 갖지 못하는 현상, 즉 서로 다른 심벌이 같은 해시 값을 갖게 되는 충돌(colllision) 현상을 야기하게 된다. ANSI 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.

[그림 2]는 후방법을 이용한 해시 심벌 테이블이다.



[그림 2] 해시 심벌 테이블

3. 역번역기 시스템

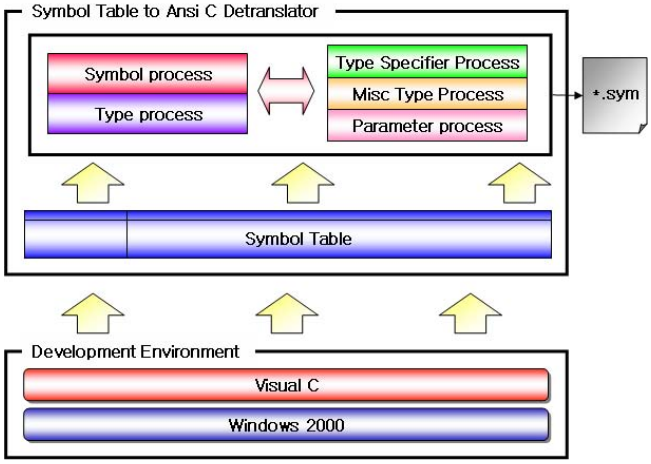
심벌 테이블에서 C 프로그램 역번역기 시스템은 모든 ANSI C 언어의 선언부를 입력으로 받아, 심벌 테이블에 삽입한 후, 삽입한 심벌 테이블의 정보를 이용하여 다시 ANSI C 언어의 선언부로 변환하는 역번역기이다.

3.1 역번역기 시스템의 구조

시스템 구현 환경은 Windows 2000에서 Visual C를 사용하였다. 역번역기 시스템의 구성은 심벌 테이블(Symbol Table), 심벌 프로세스(Symbol Process), 타입 프로세스(Type Process), TSP(Type

Specifier Process), MTP(Misc Type Process), 파라미터 프로세스(Parameter Process)로 구성되어 있다.

[그림 3]은 역번역기 프로그램의 구성도이다.



[그림 3] 역번역기 시스템 구성도

역번역기 시스템은 심벌 테이블의 정보들 중에서 선언부에 해당하는 정보만을 추출하여 역번역하게 된다. 심벌 프로세스는 심벌 테이블에서 외부 변수에 해당하는 모든 정보만을 검색하여 TSP를 통하여 변수의 타입을 구분하고 MTP를 통해 배열인지, 함수인지, 일반 변수인지를 판별하여 ANSI C 언어의 표현으로 역변역하여 출력한다. 이때 함수의 파라미터가 있는 경우에는 파라미터 프로세스를 통해 출력한다. 타입 프로세스는 타입의 종류를 구별하여 TSP와 MTP 그리고 파라미터 프로세스를 이용하여 타입에 대해 출력한다.

3.2 역번역기 알고리즘

다음은 심벌 테이블의 정보를 다시 C 언어의 표현으로 역번역하기 위한 알고리즘의 개요이다. 역번역기는 Symbol Table을 통해 일반 변수, enum 변수와 label을 구분하여 프로세싱을 하며, Type Table을 통해 구조 타입인지, typedef로 선언된 타입인지를 구분한다. 구조 타입일 경우 aggregate Table을 통해 struct, union, enum 타입을 구분하여 처리한다.

```
void processSymbol(int symbolIndex) {
 name lookup
 if(kind==CONCRETE_KIND){
    detranslate the declaration part for general variable
 }else if(kind==ENUM_KIND){
    detranslate the declaration part for enum variable
 }else if(kind==LABEL_KIND){
    detranslate the declaration part for label
 }
}
void processType(int typeIndex) {
 name lookup
 if(typeKind==AGGREGATE_TYPE){
  switch(aggregateType){
    case STRUCT_SPECIFIER:
      detranslate the declaration part for struct type
    case UNION_SPECIFIER:
      detranslate the declaration part for union type
    case ENUM_SPECIFIER:
      detranslate the declaration part for enum type
  }
 }else if(typeKind==TYPEDEF_TYPE){
   detranslate the declaration part for typedef
 }
}
```

[알고리즘 1] 역번역기 시스템 알고리즘

4. 실행 결과 및 분석

다음은 ANSI C 언어의 선언문에 대해서 심벌 테이블에 속성을 넣은 다음, 역 변환기를 통하여 다시 ANSI C 언어의 선언부로 복원하는 예제이다.

[예제 1]은 입력으로 받을 ANSI C 언어의 선언부이다.

```
int (*daytab)[13];
char (*(*x[3])())[5];
void qsort(void *, int left[], int right);
typedef struct tnode{
      unsigned int is_keyword : 1;
      char *word;
      int count;
      struct tnode *left;
      struct tnode *reght;
      union u_tag{
            int ival;
            float fval;
            char *sval;
      }u;
}NODE;
```

[예제 1] ANSI C 프로그램의 선언부

[예제 2]은 심벌 테이블의 속성을 역번역기를 통해 생성된 ANSI C 선언부와 디버그 정보를 함께 나타낸 것이다.

```
************************************************
*              SYMBOL TABLE DUMP               *
************************************************
int (*daytab)[13];
// test.c,          // 1      // 6
(base = 0, offset = 0, width = 4, isInitial = 0)
char (*(*x[3])())[5];
// test.c,          // 2      // 9
(base = 0, offset = 4, width = 12, isInitial = 0)
void qsort(void *##001, int left[], int right);
// test.c,          // 3      // 5
(base = 0, offset = 16, width = 0, isInitial = 0)
************************************************
*              TYPE TABLE DUMP                 *
************************************************
struct tnode {
        unsigned int is_keyword:1;
        char *word;
        int count;
        struct tnode*left;
        struct tnode*reght;
        union u_tag u;
};
// test.c,// 4,       // 14
(size = 20)
union u_tag {
        int ival;
        float fval;
        char *sval;
};
// test.c,// 10,      // 9
(size = 0)
typedef struct tnode NODE;
// test.c,// 15,      // 0
(size = 20)
```

[예제 2] 역번역기에 의해 생성된 ANSI C 선언부

5. 결론 및 향후연구

EVM을 위한 ANSI C 컴파일러는 ANSI C 언어를 받아 들여 EVM의 중간 언어인 SIL 코드를 출력한다. ANSI C 컴파일러에서 어휘 분석과 구문 분석 과정에서 인식되는 명칭에 대해서 그 속성들을 수집하고 이용한다. 이 속성들은 명칭이 명시적으로 혹은 묵시적으로 정의 되는 곳에서 심벌 테이블에 수집된다. 본 논문에서는 수집된 정보가 올바르며, 사용하기에 불편한 점이 없는지 확인하기 위하여 심벌 테이블에 있는 정보만을 가지고 다시 ANSI C 언어의 표현으로 복원시키는 역변역기를 구현하여 ANSI C 컴파일러의 수정이 더욱 용이해졌다.

앞으로 보다 입력된 소스에 가깝게 표현할 수 있도록 역번역기를 수정할 것이며, 수집된 정보가 올바른 것인지만 표현하는 것이 아니라 더 많은 디버그 정보를 출력하여, 출력한 정보를 이용하여 ANSI C 컴파일러의 수정을 용이하도록 역번역기를 보안할 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, 1998.

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

[4] 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.

[5] 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.

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

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