언어 변환을 위한 설계 방법

박주열○ 곽동규 고방원 유재우
숭실대학교 컴퓨터학과
{jypark○, coolman, bwko}@ss.ssu.ac.kr cwyoo@ssu.ac.kr

Design Method for Language Translation

Ju-Yeol Park, Dong-Guy Kawk, Bang-Won Ko, Chae-Woo Yoo
Dept. of Computing, Soongsil University

요 약

현재 컴퓨터 응용프로그램을 개발하기 위해 많은 프로그래밍 언어가 존재한다. 사용자는 사용 목적에 따라 그에 맞는 새로운 프로그래밍 언어를 학습하고 사용한다. 그러나 프로그래밍 언어를 학습하고 사용하기 위해서는 많은 시간과 노력이 필요하기 때문에 사용자의 언어 학습의 비용과 시간을 줄이기 위해 두 언어간에 변환을 가능하게 해주는 방법이 필요하다. 본 논문에서는 기존의 어휘 분석, 구문 분석 등의 여러 단계를 거치는 방법과는 다른 토큰 매핑 테이블과 XML로 기술한 변환 규칙을 통한 새로운 언어 변환을 위한 설계 방법을 제시함으로써 재사용성과 생산성을 높일 수 있는 것으로 기대된다.

1. 서 론

현재 많은 분야에서 컴퓨터 응용프로그램이 운영되고 있으며, 이러한 응용프로그램을 개발하기 위한 수 많은 프로그래밍 언어가 존재한다. 프로그래밍 언어들은 추구하는 목적에 따라서 기능들이 특화되어 있고, 사용자는 그 사용목적에 따라서 프로그래밍 언어를 선택하고 사용한다. 하지만 다른 새로운 프로그래밍 언어를 학습하고 사용하는 과정은 쉽지 않으며 사용자에게 많은 시간과 노력을 요구한다.

언어 변환기(language translator)란 어떤 언어 A로 작성된 프로그램을 다른 언어 B의 형태로 변환시켜주는 프로그램으로 새로운 언어를 학습하고 사용하는데 드는 시간과 노력을 줄여줄 수 있다. 언어 변환기는 추상 구문 트리(abstract syntax tree)를 순회(traversal)하면서 목적 언어에 대응(mapping)하는 코드를 생성(code generation)해준다. 언어 변환기에 의해 생성된 두 프로그램은 의미론적으로 동일하다. 이러한 언어 변환기를 만드는 과정은 복잡한 여러 단계를 거치고 파싱 기법이 들어가 매우 어렵다.

본 논문에서는 새로운 언어 변환 규칙 생성에 대한 방법을 제시한다. DMLT(Design Method for Language Translation)는 기존의 AST를 순회하면서 생성하는 방법에 비해 절차가 간결하고 XML 문서로 기술하기 때문에 사용자는 서로 다른 언어로 기술된 프로그램의 구조를 명확하게 이해 할 수 있다. 또한 기존의 많은 XML 파서와 개발 도구를 사용할 수 있는 장점이 있다. 이전의 변환 방법은 1:1 대응하는 방법으로 AtoB 변환기와 BtoA는 별개의 프로그램으로 처음부터 만들어야 하는 단점이 있다. 그러나 DMLT는 변환 규칙 명세서만 있으면 AtoB와 BtoA 의 변환기를 생성할 수 있으므로 재사용성과 확장성과 생산성이 높은 장점을 가진다. DMLT의 언어 변환과정은 다음과 같다.

첫째, 기존의 방법과는 달리 A와 B 두 언어를 각각 어휘 분석기(lex)[1]를 이용하여 어휘 분석을 하고 분석한 어휘들을 이용하여 토큰(token)과 어휘항목(lexeme)을 가지는 토큰 테이블(token table)을 생성한다.

둘째, 토큰 테이블에 토큰들의 순서를 나타내는 순번(sequential number)과 공통 토큰을 정의, 추가해 테이블을 확장한다. 공통 토큰이란 두 언어 사이에서 공유될 수 있는 토큰을 말하며 미리 작성된 토큰 매핑 테이블에 따라서 작성된다.

셋째, 이렇게 생성된 확장 토큰 테이블을 이용하여 변환 규칙(translation rule)들을 생성한다. 생성된 변환 규칙들은 XML[2]로 기술한다.

넷째, 변환 규칙 문서를 파싱하여 목적 코드를 생성할 수 있는 변환기를 만든다.

다섯째, A언어로 쓰여진 프로그램과 XML로 기술된 변환 규칙 명세서를 변환기의 입력으로 넣으면 변환기는 B언어로 쓰여진 프로그램을 생성해준다.

본 논문에서는 언어 변환의 한가지 예로 C프로그램을 PL0프로그램으로 변환하는 과정을 기술한다. C언어는 가장 대표적인 구조적 프로그래밍 언어로 예제 언어로 쓰기에 부족함이 없고 PL0는 교육용 언어로 문법이 간결하고 명확하여 두 언어간 변환 방법을 본 논문에서 기술한다.

2. 관련 연구

2.1 C2JNI[3]

자바 언어는 플랫폼 독립성이라는 장점을 바탕으로 많은 영역에서 사용되고 있다. 자바에서 플랫폼에 종속적

인 기능을 사용하기 위해서는 JNI 명세에 따라 C 혹은 C++ 언어를 이용해서 프로그램을 작성해야 한다. 그러나 JNI명세에 따르는 프로그램을 작성하는 것은 상당히 복잡하고, 번거롭다는 문제점이 있다. C2JNI는 주어진 C 언어 프로그램을 파싱하여 AST를 만들고 생성된 AST는 트리 변환 단계에서 자바 클래스의 멤버에 접근하는 노드를 JNI 명세에 맞는 함수 호출 노드로 변환한다. 변환된 AST는 코드 생성 단계를 거치면서 JNI 명세에 맞는 C언어 코드를 생성한다. 이와 같은 방식은 AST를 순회하면서 코드를 생성하기 때문에 C언어와 JNI 코드간에 1:1 대응되어 있다. 이러한 방법은 원시언어나 목적언어가 바뀌게 되면 새로 작성해야 하는 단점이 있다.

2.2 패턴 매칭 기법을 이용한 자바 바이트코드 변환기의 설계 및 구현[4]

자바 바이트 코드 변환기는, 바이트코드로부터 직접 네이티브 코드를 생성하기 위해 자바의 중간 언어인 class 파일을 입력으로 받아 레지스터 기반의 중간 언어로 변환한다. 이를 위해 첫째, 바이트코드로부터 레지스터 기반 언어로의 효율적인 변환을 위한 패턴을 정형화된 방법으로 기술한다. 둘째 정형화된 패턴으로 입력받아 패턴 매칭에 적합한 테이블 정보를 생성한다. 셋째, 코드 변환 테이블을 참조하여 스트링 패턴 매칭 기법을 적용하여 레지스터 기반 중간 언어인 gasm 코드를 생성하는 패턴 매칭기를 구현한다.

이러한 변환 패턴 방법은 파서 생성기인 bison의 입력에 적합하도록 작성이 되어 두 프로그램간의 구조를 분석하기 어렵다는 단점이 있으므로 이를 구조적으로 표현하기 위한 방법이 필요하다.

3. 확장 토큰 테이블(Extend Token Table) 생성

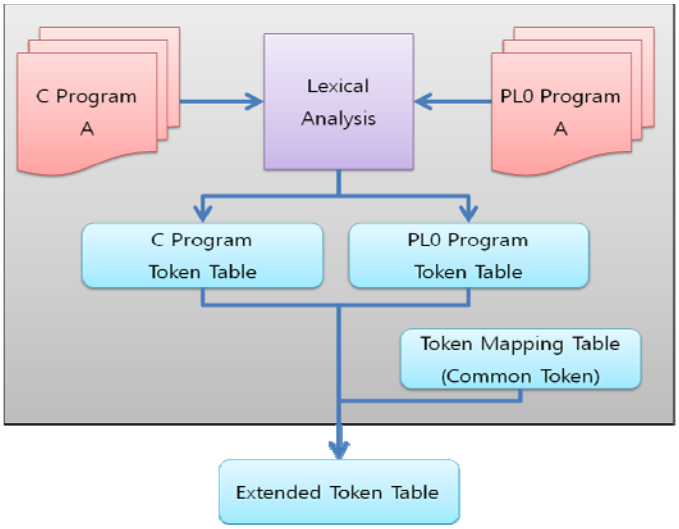

그림 1. 확장 토큰 테이블 생성과정

본 장에서는 DMLT의 변환 규칙을 설계하기 위해 필요한 확장 토큰 테이블 생성과정을 기술한다. 확장 토큰 테이블을 생성하는 과정은 그림 1과 같으며 확장 토큰 테이블은 기존의 토큰 테이블과 비슷한 구조를 가지고 있으나, 추가적으로 변환을 위한 정보를 포함하고 있다.

표 1은 1부터 10까지 더하는 프로그램으로 각각 C와 PL0로 작성되어있다. 두 프로그램의 문맥적 의미는 동일하다.

표 1. 예제 프로그램

C	PL0

int main()
{
plus();
return 0;
}
void plus()
{
int x ;
x = 1;
while(x < 10)
{
x = x+ 1;
}
}
	PROCEDURE plus
	VAR x;
	BEGIN
	x := 1;
	WHILE x < 10 DO
	BEGIN
	x := x + 1;
	END
	END;
	BEGIN
	CALL plus;
	END.

3.1 토큰 테이블 생성

토큰 테이블은 심볼 테이블의 축약된 형태의 자료 구조로서 토큰과 어휘항목을 가진다. 토큰 테이블 생성을 위해서 기본적으로 입력인 프로그램 소스 코드를 의미있는 문자열로 구분하기 위한 작업이 필요하다. 이러한 작업은 어휘 분석 단계에서 이루어지며, 프로그래밍 언어의 터미널 심볼에 맞게 정규표현(regular expression)식을 정의하고, 어휘분석기에 입력하여 토큰을 생성한다. 이렇게 생성된 토큰들과 각 토큰에 해당하는 실제 어휘항목을 테이블에 넣어 저장한다.

3.2 토큰 매핑 테이블

토큰 매핑 테이블은 두 언어간 비교를 위해 토큰을 공통적으로 표현한 테이블이다. 즉 각 언어에서 같은 심볼을 서로 다른 토큰명(token name)으로 나타낼 수 있기 때문에 이러한 부분을 서로 대응시키기 위해 만든 테이블이다. 예를 들어 C언어에서의 IDENTIFIER를 PL0에서는 ID로 표현 한다면, 두 토큰명은 틀리지만 동일하게 식별자를 나타낸다. 이렇게 같은 의미를 지닌 토큰은 동

일하게 표현한다. 즉 IDENTIFIER로 통일 했다면 PL0의 ID 토큰을 IDENTIFIER 변환한다. 또한 토큰명과 어휘항목(lexeme)도 틀리지만 역시 의미가 같으면 동일한 토큰으로 변환한다. 예를 들어 '{' 은 C언어에서 블록의 시작을 나타낸다. 이와 동일하게 PL0에서 블록의 시작을 나타내는 어휘는 BEGIN이 있다. 이 둘은 토큰명과 어휘항목 모두 다르지만 의미적가 같은 토큰이므로 이 둘을 동일한 토큰으로 표현해야 한다.

3.3 확장 토큰 테이블(Extended Token Table) 생성

확장 토큰 테이블은 생성한 토큰 테이블에 토큰의 순서를 나타내는 순번(sequencial number)과 공통 토큰을 추가한 테이블을 말한다. 이 확장 심볼 테이블 생성 방법은 다음과 같다.

첫째, 기존의 토큰 테이블에 토큰들의 순번을 생성해 넣어준다.

둘째, 매핑 테이블을 이용하여 토큰 테이블의 각 토큰에 해당하는 공통 토큰을 찾아서 넣어준다.

셋째, 확장 테이블에서 프로그램의 시작 블록이나 함수는 토큰에 마크심볼을 더해준다.

위와 같은 순서로 표 1의 예제 프로그램을 수행하여 나온 확장 토큰 테이블의 형태는 표 2와 표 3과 같다.

표 2. C언어 확장 토큰 테이블

C			
어휘항목	토큰	순번	공통 토큰
int	INT	0x0001	TYPE_M
main	ID	0x0002	ID_M
….			
}	RBRACE	0x000B	END_M
void	TYPE	0x000C	TYPE
test	ID	0x000D	ID
(	LPAREN	0x000E	LPAREN
…			

표 3. PL0언어 확장 토큰 테이블

PL0			
어휘항목	토큰	순번	공통 토큰
PROCEDURE	PROCEDURE	0x0001	TYPE
test	ID	0x0002	ID
VAR	VAR	0x0003	TYPE
x	ID	0x0004	ID
…			
BEGIN	BEGIN	0x0016	BEGIN_M
CALL	CALL	0x0017	CALL_M
…			

4. 변환 규칙(translation rule) 생성

확장 토큰 테이블을 이용해 변환 규칙 생성 알고리즘을 정의하고 이를 적용하면 변환 규칙을 만들 수 있다. 이 알고리즘을 이용하면 간단하게 CtoPL0 또는 PL0toC 변환 규칙을 생성할 수 있다. 이렇게 생성된 변환 규칙은 XML로 기술하여 프로그램의 구조를 명확히 하고 재사용성과 생산성을 높일 수 있다. 또한 간단하게 XML Parser를 이용하여 변환 규칙을 파싱하면 변환기를 생성할 수 있다.

그림 2는 확장 토큰 테이블을 이용하여 변환규칙을 적용하는 전체적인 과정이다.

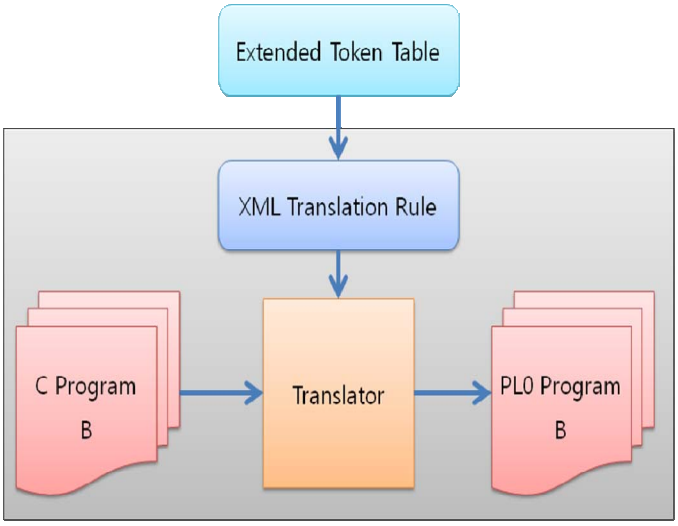

그림 2. 변환 규칙을 이용한 변환 과정

4.1 변환 규칙

생성된 확장 토큰 테이블로부터 정보를 수집해서 변환 규칙을 설계해야 한다. 변환 규칙을 생성하기 위해서 필요한 데이터는 토큰의 순번(Sequencial Number)과, 공통토큰(Common Token)이다. 토큰의 순번을 *SN*, 공통 토큰을 *CT*라 하고 이 두 데이터를 표현하는 새로운 데이터 타입을 *T* 로 표현한다.

$T = (SN, CT)$

이렇게 수집한 정보를 이용해 변환 규칙을 생성하기 위한 알고리즘은 다음과 같다.

첫째, 확장 토큰 테이블에서 두 언어의 모든 *T* 정보를 수집해 각각 리스트 *L1, L2* 에 입력한다.

둘째, *L1* 에서 가장 낮은 *SN* 값을 가진 *T* 부터 비교를 시작한다. (기본값 $SN = 1$) *L1* 과 *L2* 에서 *CT* 가 같은 항목을 찾는다. 같은 항목을 찾으면 *L2* 의 *T* 를 *L1* 의 *T* 로 대체하고 *L1* 의 *SN* 을 1 증가시킨다.

셋째, 현재 *L2* 의 *SN* (*L2* current location)부터 테이블 끝까지 찾지 못하면 다시 L2리스트 처음으로 돌아가 찾는다. 만약 *L2* 의 원래의 위치(*L2* current location)로

돌아오면 통과한다.
넷째, *L1* 의 끝에 도달하면 종료한다.

다음의 표 4는 변환 규칙 알고리즘을 Java 코드로 표현한 예제이다.

표4. Java로 표현한 알고리즘

```
T t;
int sn2 = 1;    //list2의 sequencial number
bool find(T t1, int loc_sn) {
    while(loc_sn < list2.size) {
        t2 = list2.get(loc_sn);    //list2에서 T를 얻는다.
        if(t1 == t2) {
            sn2 = loc_sn + 1;    //list2에서 검색 할 위치를 저장
            return true;
        }
        loc_sn++;
        if(sn2 == loc_sn) {    //리스트 전체를 검색해도
                               // 없는 경우 skip
            sn2++;
            return false;
        }
    }
    return find(t1, 1);      //중간지점 loc_sn 다음에 없으므로
                             //처음부터 loc_sn까지 다시 찾는다.
}
public static void int main(String[] args) {
    for(int sn1 = 1; sn < list1.size; sn1++) {
        t = list1.get(sn1);    // sequencial num에
                               // 해당하는 객체를 받아온다.
        if(find(t, sn2)) {
            list2.get(sn2).set(t);    //T를 교체
        }
    }
}
```

4.2 변환 규칙 기술

XML은 구조적인 데이터를 표현 하는데 적합한 언어이다. 이러한 XML을 이용하여 변환규칙을 기술하면 언어의 구조를 표현하기 좋고 기존의 유용한 XML 툴을 이용하여 개발할 수 있는 장점이 있다.
변환 규칙 생성 알고리즘을 적용하여 나온 변환 규칙을 XML로 표현하면 다음의 표 5와 같다.

표 5. C언어를 PL0언어로 변환하는 변환 규칙 명세

```
<DMLT>
    <PROCEDURE>
        <TYPE mapping = "12">PROCEDURE</TYPE>
        <ID mapping = "13" lexeme = "test">test</ID>
        <TYPE mapping = "17">VAR</TYPE>
        <ID mapping = "18" lexeme = "x">x</ID>
        <BEGIN mapping = "16"'>BEGIN</BEGIN>
        <ID mapping = "x">x</ID>
                        ..
                        ..
    </PROCEDURE>
    <MAIN>
        <BEGIN_M mapping = "5">BEGIN</BEGIN_M>
        <CALL_M mapping = null>CALL</CALL_M>
        <ID_M mapping = "6">test</ID_M>
                        ..
                        ..
    </MAIN>
</DMLT>
```

최상위 태그는 DMLT로 main과 function 엘리먼트들의 집합으로 이루어진다. 각 태그는 mapping속성을 가지고 있는데 이는 C언어 확장 토큰 테이블의 순번과 매핑되는 PL0 언어의 확장 토큰 테이블의 순번이다. 해당하는 순번이 없는 경우는 null을 표시한다. 그리고 PCDATA는 실제 C언어의 어휘항목을 나타낸다.

4.3 변환기 생성

변환규칙 명세서를 위해 새로 파서를 만들 필요없이, 기존의 SAX나 DOM과 같은 XML 파서를 이용하여 변환규칙 명세서를 파싱해서 변환기를 만든다.
변환기를 만드는 방법은 다음과 같다.
첫째, 태그 속성 mapping의 값에 대응하는 PL0 확장 토큰 테이블의 순번을 찾는다.
둘째, 순번에 해당하는 어휘항목을 목적 코드로 출력한다.
이처럼 변환 규칙 명세서만 있으면 기존의 XML 파서를 이용하여 간단하게 변환기를 만들 수 있다.

5. 결론 및 향후 연구과제

현재 많은 프로그래밍 언어가 응용프로그램 개발에 사용되고 있다. 일부 프로그래밍 언어는 일반적인 프로그래밍 언어와는 달리 특정 기능(모니터링, 상태체크 등)을 위해 특화된 언어도 있다. 이런 프로그래밍 언어는 범용의 프로그래밍 언어와 다른 형태를 가지고 있어 개발과 유지 보수의 어려움이 있다. 이와 같이 다른 프로그래밍 언어를 학습하고 능숙하게 사용하는 것은 많은 시간과 노력을 필요로 한다.
본 논문에서 제시한 언어 변환기는 이러한 시간과 노력을 줄일 수 있다. 기존의 언어 변환기는 어휘 분석과

구문 분석 후 AST를 순회하는 복잡한 과정을 거치기 때문에 언어 변환기의 생성이 어렵다. 본 논문에서 제안한 DMLT는 이전의 복잡한 과정과는 달리 확장 토큰 테이블을 이용하여 변환하기 때문에 변환과정이 단순하다. 또한 변환 규칙을 XML로 기술하여 재사용성과 생산성을 높이고 프로그램의 구조를 쉽게 이해할 수 있는 장점이 있다. 그리고 변환 규칙 명세서만 있으면 언어 A와 언어 B의 변환을 AtoB와 BtoA로 변환할 수 있는 장점도 있다.

향후 연구 과제로서 두 언어간 표현범위가 다른 언어를 변환하는 방법이 필요하다. 예를 들어 C의 포인터 변수를 Java나 Ruby 등의 언어에서 어떻게 변환할 것인지에 대한 방법이 필요하고, 객체지향 언어를 구조적 프로그래밍 언어로 변환할 때 생길 수 있는 프로그램간 구조를 변환하는 방법의 연구가 필요하다.

5. 참고문헌

[1] M. E. Lesk and E. Schmidt, “Lex – A Lexical Analyzer Generator”, Bell Laboratories, Murray Hill, New Jersey 07974.

[2] http://www.w3.org/TR/2006/REC-xml-20060816/

[3] 유재우, 최종명, 김영철, “C2JNI : 내장 C언어에서 JNI 코드를 생성하는 변환기”, 한국정보과학회논문지: 소프트웨어 및 응용, 제 31권 제 11호, pp.1551~1559, 2004

[4] 고광만, “패턴 매칭 기법을 이용한 자바 바이트 코드 변환기의 설계 및 구현”, 대한전자공학회 논문지, 제39권, 제4호, pp1~9, 2002