논•문 16-3-6

컴파일러 수업에서 레고 마인드스톰의 활용
An Application of Lego Mindstorms to Compiler Course

이 수 현
창원대학교 컴퓨터공학과
suhyun@sarim.changwon.ac.kr

요 약

컴파일러 교과목은 인간 중심의 프로그래밍 언어 기술과 기계 중심의 하드웨어 기술이 모두 적용되는 이수 난이도가 높은 교과목이며, 컴파일러 제작 실습이 수반되어야 하는 교과목이다. 컴파일러 제작 실습에 있어 가상기계를 타겟(목적기계)으로 이용하는 방법을 많은 대학에서 채용하고 있으나, 가상기계의 동작을 시뮬레이션으로 보여줄 수밖에 없어서 현실감이 떨어지는 단점이 있다. 레고의 고수준 장난감인 마인드스톰은 컴퓨터를 내장한 로봇 조립용 키트이며, 간단한 명령어만으로 로봇을 동작시킬 수 있다. 로봇을 제어하는 간단한 언어를 설계하고 마인드스톰을 타겟으로 하는 컴파일러를 제작해 본다면 컴파일러의 동작을 직접 눈으로 확인하고 느낄 수 있어 컴파일러 교육의 효과를 높일 수 있다. 본 논문에서는 레고 마인드스톰을 제어하는 언어인 LegoC를 설계하고 LegoC 언어의 컴파일러를 제작한 경험을 기술한다. 또한 마인드스톰을 컴파일러 수업에서 활용할 수 있는 방안을 제시한다.

1. 서론

컴퓨터공학과의 교과과정 중에서 컴파일러 교과목은 상당히 어려운 교과목에 속한다. 컴파일러 교과목이 학생들에게 어렵다고 느끼는 이유 중의 하나는 컴파일러를 이해하기 위해서는 프로그래밍 언어에 관한 지식과 하드웨어의 지식을 모두 갖추고 있어야 한다는 점이다.

컴파일러 제작을 실습하는 경우에 실제의 컴퓨터는 너무 복잡하여 타겟으로 적당하지 않으므로 많은 대학에서 간단한 형태의 가상기계를 타겟으로 사용하고 있다. ([1]에서 가상기계의 한 형태를 볼 수 있다.) 실제의 컴퓨터와 가상기계 중 어느 것을 타겟으로 하

든 제작된 컴파일러의 동작을 보여주는데 현실감이 떨어지므로 학생들의 흥미를 유발시키는데는 미흡하다.

컴파일러의 동작을 흥미롭게 보여주기 위해서는 실제로 움직이는 기계를 타겟으로 하는 것이 바람직하며, 메카트로닉스 분야의 교육에 활용되는 암(arm) 로봇이나 보행 로봇 등이 한가지 방법이 될 수 있다. 그러나 이들 로봇은 동작 자체가 단순하고 출력은 제어할 수 있으나 입력장치가 제한적이라는 단점이 있다.

조립형 장난감을 만드는 레고(Lego)의 마인드스톰(MindStorms)[2]은 컴퓨터를 내장한 로봇 조립용 키트로서 완벽한 로봇의 제작을 간단한 조립과 프로그래밍으로 가능하게 한다. 마인드스톰은 흥미 있는 장난감이면서 하드웨어는 복잡하지 않고 입출력을 이용하여 다양한 형태의 움직임을 만들 수 있으므로 컴파일러의 타겟으로 적절하다. 제작된 컴파일러의 동작은 실제로 움직이는 로봇으로서 확인할 수 있으므로 컴파일러 교육의 효과를 매우 높일 수 있을 것이다.

본 논문에서는 레고 마인드스톰을 제어하는 컴파일러를 제작한 경험을 기술하고, 컴파일러 수업에서 레고 마인드스톰을 활용할 수 있는 방안을 제시한다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 본 연구에서 사용한 타겟인 레고 마인드스톰에 대하여 설명한다. 3장에서는 레고 로봇을 제어하기 위한 언어에 대하여 기술하였으며, 4장에서는 프로토타입 컴파일러에 대하여 기술하였다. 5장에서는 컴퓨터 공학의 몇 개의 교과과정에서 레고 마인드스톰을 활용하는 방안을 제시하였다. 마지막으로 6장에서는 결론 및 향후 연구 방향으로 끝을 맺고자 한다.

2. 관련연구

본 장에서는 레고 마인드스톰과 마인드스톰의 두뇌에 해당하는 RCX, 그리고 레고 프로그래밍과 관련한 내용을 정리하였다.

2.1 레고 마인드스톰과 RCX

조립형 블록 장난감을 생산하는 레고사에서는 1998년 12세 이상을 대상으로 하는 로봇 제품인 마인드스톰을 발표하였다. 마인드스톰이 레고의 다른 제품들과 다른 점은 사용자가 조립한 블록을 프로그래밍을 통하여 직접 제어할 수 있다는 점이다. 이를 위하여 마인드스톰에는 소형 컴퓨터를 내장하고 있으며, 각종 센서와 모터를 연결할 수 있다.

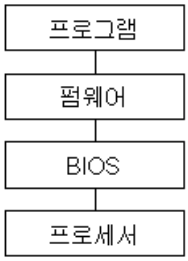
마인드스톰의 두뇌에 해당하는 RCX는 프로세서로 8-Bit Hitachi H8/3292 16MHz를 내장하고 있으며, (그림 1)에서 보는 바와 같이 3개의 입력단자와 3개의 출력단자, 액정 디스플레이, 적외선 단자 등으로 이루어져 있다.

(그림 1) 마인드스톰의 두뇌 RCX

A, B, C라는 이름이 붙어 있는 출력단자에는 주로 모터가 연결되며, 램프도 연결 가능하다. 1, 2, 3이라는 이름이 붙어 있는 입력단자에는 다양한 종류의 센서가 연결돼 주위 환경에 반응하는 로봇을 만들 수 있다. 센서의 종류로는 빛의 세기를 감지하는 빛 센서, 누름 스위치와 같은 기능을 할 수 있는 접촉 센서, 회전의 양을 측정하는 회전 센서, 온도를 감지하는 온도 센서 등이 있다.

적외선 단자는 PC 또는 다른 RCX와의 통신에 사용되며, 프로그램을 다운로드 받거나 메시지를 교환할 수 있다. 적외선 통신을 위해서 PC에는 시리얼 포트 또는 USB 포트에 연결되는 적외선 장치(IR tower)가 설치되어야 한다.

소프트웨어의 측면에서 RCX는 (그림 2)에서 보는 바와 같이 4개의 계층으로 되어 있다[3].



(그림 2) RCX의 논리 구조

RCX의 운영체제는 두 부분으로 나누어져 있는데, 마이크로프로세서에 내장된 16K 크기의 ROM에 포함된 BIOS와 RAM에 저장되어 있는 펌웨어(firmware)로 구성되어 있다. 초기상태의 RCX에는 펌웨어가 존재하지 않으며, 펌웨어는 PC로부터 다운로드 되어 사용된다. RCX에서 작동되는 프로그램은 프로세스에 의해 해석되는 기계어가 아닌 펌웨어가 해석할 바이트코드(Bytecode)로 쓰여져 있다. 펌웨어는 사용자 프로그램을 해석하여 기계어 코드로 번역하여 실행한다.

RCX의 RAM은 32KB의 크기이며, 16KB는 펌웨어가 사용하며, 사용자 프로그램은 6KB, 나머지 공간은 바이트코드 해석이나 프로그램의 실행시의 데이터 영역으로 사용된다. 6KB의 사용자 프로그램 영역에는 최대 5개의 프로그램을 저장할 수 있으며, 하나의 프로그램에는 동시에 실행될 수 있는 10개의 태스크(task)와 8개의 서브루틴을 포함할 수 있다.

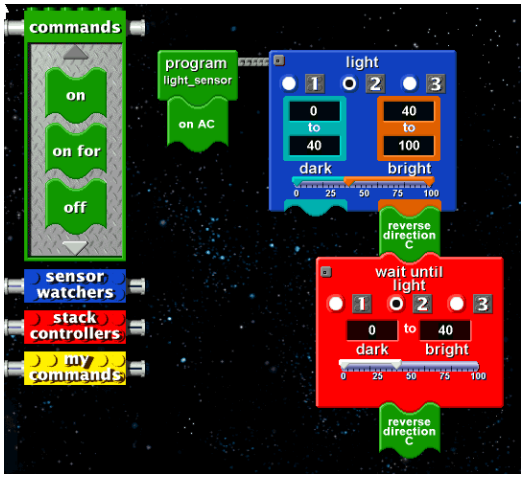
표준 펌웨어 환경에서 서브루틴은 값을 반환할 수 없고, 다른 서브루틴을 호출하지 못하며 지역변수를 가질 수 없는 제약을 있다[4]. 변수는 16비트이며 32개까지 사용할 수 있는데 전역변수로만 사용된다. 즉, 32개의 변수는 모든 프로그램과 모든 태스크에서 공유되며 명시적인 명령이 없으면 프로그램의 시작 등에 의해서 값이 변경되지 않는다.

2.2 RCX 프로그래밍

레고 마인드스톰으로 로봇을 제작하는 단계는 첫째, 로봇의 동작을 구상하고 형태와 구조를 설계한 다음, 둘째, 블록을 이용하여 로봇을 조립하고, 셋째, 로봇의 동작을 제어하는 프로그램을 PC에서 작성하여 RCX로 다운로드 하여, 넷째, 로봇을 실제로 동작시키는 단계로 이루어진다. 이번 절에서는 세 번째 단계인 프로그래밍을 위한 도구들에 대해서 살펴본다.

(1) RCX Code

마인드스톰과 함께 제공되는 RCX Code는 비주얼 프로그래밍 환경으로 어린이들이나 프로그래밍에 익숙하지 않은 사람도 쉽게 사용할 수 있도록 모든 구문을 블록의 모양으로 만들었다. 프로그래밍은 (그림 3)에서 보는 바와 같이 레고 블록을 끼워 맞추듯이 RCX Code에서 구문 블록을 끼워 맞추면 된다. RCX Code는 다양한 구문과 제어 구조를 제공하고 있기는 하지만, 프로그램의 길이가 길어지면 가독성에 문제가 있으며 RCX의 모든 기능을 사용할 수 없다는 단점이 있다. 또한 마이크로소프트 윈도우즈 운영체제에서만 실행할 수 있다.



(그림 3) RCX Code 프로그램

(2) NQC

NQC(Not Quite C)[5]는 Dave Baum에 의해 개발된 RCX 전용 프로그래밍 언어이다. NQC의 구문은 C와 유사하여 프로그래머에게 친숙한 장점이 있으며 NQC 프로그램은 RCX의 기능을 완전히 이용할 수 있어서 RCX 프로그래머에게 가장 널리 사용되는 언어이다. NQC는 다양한 운영체제를 지원하므로, Unix/Linux나 Mac 등의 환경에서 프로그램을 작성할 때 유용하다.

NQC는 RCX 표준 펌웨어의 바이트코드를 타겟으로 하기 때문에 RCX 펌웨어의 제한요소가 그대로 적용된다. 자료형(data type)은 16-비트 정수형만을 허용하며, C에서와 같이 정수 값 0은 참(true)을 나타내기도 한다. 사용자가 정의한 함수는 반환 값은 가질 수 없으며, inline 함수로 간주되어 호출된 위치에 함수 본체가 삽입된다.

RCX의 태스크와 서브루틴은 프로그램에서 task 키워드와 sub 키워드를 사용하여, 사용자 함수와 구별된다. <표 1>에 이들의 구문을 비교하였다.

<표 1> 태스크, 함수 및 서브루틴의 구문

종 류	구 문
태스크	task *name*() { *body*; }
함수	void *name*(*args*) { *body*; }
서브루틴	sub *name*() { *body*; }

(3) 자바

자바 프로그램을 RCX에 실행하기 위해서 자바가상기계(JVM)를 RCX에 포팅하려는 노력들이 있어 왔다. 이런 노력은 표준 펌웨어를 대체하는 새로운 펌웨어의 개발을 가져왔고, 2.3절에서 소개하는 TinyVM과 LeJOS가 대표적이다. 이러한 펌웨어 상에서 자바 바이트코드가 실행될 수 있다.

2.3 RCX 펌웨어

레고 마인드스톰의 표준 펌웨어는 여러 가

지 제약점이 많은 관계로 펌웨어를 향상시키려는 시도가 있어 왔다 본 절에서는 이러한 시도들에 대하여 간략히 알아본다.

(1) 표준 펌웨어

2001년 레고 마인드스톰의 버전이 2.0으로 올라가면서 표준 펌웨어도 2.0으로 향상되었다. 펌웨어 2.0의 향상된 기능 중에서 프로그래밍과 관련한 중요한 것은 다음과 같다.

- 배열(array)의 지원
- 태스크 당 16개의 지역변수 사용 가능
- 다양한 LCD 디스플레이의 지원

(2) LegOS

LegOS[6]는 Markus Noga에 의해 개발된 RCX용 운영체제이다. LegOS의 목적은 프로그램을 Hitachi 프로세서의 기계어로 직접 실행하는 것이다. LegOS에 의하여 하드웨어의 기능을 완전히 사용할 수 있을 뿐만 아니라 프로그램을 표준 C/C++로 작성할 수 있는 장점이 있다.

다중작업(multitasking), 부동 소수점, 네트워크, 쓰레드 등의 기능을 지원하고 있어 RCX 펌웨어 중에서 가장 강력하다.

(3) pbForth

Forth는 내장형 시스템을 위한 프로그래밍 환경으로 로보틱스와 자동화 응용에 많이 사용되고 있다. pbForth(programmable brick Forth)[7]는 Ralph Hempel에 의해서 개발된 표준 펌웨어를 대체하는 Forth 해석기이다. pbForth는 대화형(interactive)으로 작동하여 터미널에서 입력한 Forth 명령이 바로 실행되어 RCX에서 결과로 나타난다.

(4) TinyVM과 LeJOS

TinyVM[8]은 Jose Solorzano에 의해서 개발된 표준 펌웨어를 대체하는 약 10KB 정도 크기의 자바가상기계이다. PC에서 작성된 자바 프로그램은 컴파일되어 RCX에 적재된 후 실행된다. TinyVM은 표준 자바 라이브러리의 부분집합, 센서와 모터 조정을 위한 라이브러리 등을 지원한다.

LeJOS[9]는 TinyVM을 확장한 것으로 여러 개의 프로그램을 RCX에 적재할 수 있는 약 17KB의 크기를 가지는 운영체제이다. LeJOS는 부동 소수점 연산, 수학 함수, 쓰레드, 스트링 등 자바 언어의 완전한 기능을 지원한다.

3. LegoC 언어

LegoC 언어는 표준 펌웨어가 탑재된 RCX를 제어하는 프로그램을 작성하기 위한 언어이다. 현재의 프로토타입 구현에서 LegoC는 RCX를 제어하기 위한 최소한의 기능만을 포함하고 있다.

3.1 언어 설계

LegoC 언어는 C와 유사한 문법구조를 가지고 있다. 제어구조로는 배정문, if-문, while-문, 복합문, 함수호출 등을 제공하고 있다. 그러나 사용자가 함수를 정의할 수는 없기 때문에, 함수호출은 내장함수를 호출하는 데만 사용된다. if-문과 while-문에서 사용되는 조건식은 단순 크기 비교만을 허용하

며, and, or, not과 같은 논리연산자는 허용되지 않는다.

프로그램은 task 키워드를 가지는 하나의 함수로 구성되어 있으며, 변수는 task 함수의 바깥에 전역변수로만 정의될 수 있다. 자료형은 정수형만을 지원하며, 수식은 변수와 상수를 포함하는 사칙연산과 괄호를 포함할 수 있으나, 함수를 사용할 수는 없다. (그림 4)에 LegoC 언어의 문법을 나타내었다.

```
<LegoControl> ::= <VarDecl> task <Name>
                  <CompStmt>
<VarDecl> ::= <DclList> | ε
<DclList> ::= <DclList> <Decl> | <Decl>
<Decl>    ::= int <VarList> ;
<VarList> ::= <VarList> , <Var> | <Var>
<Stmt>    ::= <AsgnStmt> | <IfStmt> | <CallStmt>
            | <WhileStmt> | <CompStmt>
<AsgnStmt> ::= <Var> = <Exp> ;
<IfStmt>   ::= if ( <Cond> ) <Stmt>
             | if ( <Cond> ) <Stmt> else <Stmt>
<WhileStmt> ::= while ( <Cond> ) <Stmt>
<CallStmt>  ::= <Name> ( <ParamList> ) ;
<ParamList> ::= <ExpList> | ε
<ExpList>   ::= <ExpList> , <Exp> | <Exp>
<CompStmt> ::= { <StmtList> }
<StmtList>  ::= <StmtList> <Stmt> | <Stmt>
<Cond>::= <Exp> <= <Exp> | <Exp> >= <Exp>
        | <Exp> != <Exp> | <Exp> = <Exp>
<Exp> ::= <Exp> + <T> | <Exp> - <T> | <T>
<T> ::= <T> * <F> | <T> / <F> | <F>
<F> ::= <Var> | <Number> | - <F> | ( <Exp> )
```

(그림 4) LegoC 언어의 문법

LegoC 언어의 어휘는 다음과 같이 정의하였다.

- 키워드: task, int, if, else, while는 특정한 의미를 가지고 있으므로 프로그램에서 변수나 함수의 이름으로 사용될 수 없다. 프로그램 전체에서 대소문자는 구별하지 아니한다.
- 연산자와 구분자는 다음과 같다:
  = <= >= != + - * / ; ( ) { } ,
- 이름: 변수와 함수의 이름은 영문자와 숫자 및 밑줄문자(_)로 구성되어 있으며, 첫 글자는 영문자이어야 한다.
- 숫자: 수식에서 사용할 수 있는 숫자는 정수의 10진수 또는 16진수 표기이다. 16진수 숫자는 0x 또는 0X로 시작한다.
- 주석: 프로그램 내에서 //가 발견되면, // 부터 해당 줄의 끝까지 모든 문자열을 주석으로 간주되어, 어휘분석기에 의하여 제거된다.

3.2 내장 함수

RCX에서 입출력은 주로 센서와 모터를 통하여 일어난다. 이에 따라 RCX에는 센서와 모터를 제어하기 위한 명령어들이 포함되어 있으며, LegoC 언어의 입출력은 이들 명령어로 번역된다.

LegoC에서 입출력은 내장 함수의 형태로 지원되며 모든 함수는 값을 반환하지 않는다. 현재의 구현에서 RCX의 일부의 기능만을 지원하며 함수이름은 NQC를 참고하였다.

(1) 센서

- 센서 값 읽기

  SensorValue(*sensor*, *n*)

  *sensor*로 지정된 센서의 현재 값을 읽어서 변수 *n*에 저장한다. 이 함수는 부대효과(side effect)를 유발하여 함수호출에 의하여 변수 *n*의 값이 바뀌게 된

다. *sensor* 인자에는 각각 RCX의 1, 2, 3번 포트에 연결된 센서를 나타내는 SENSOR_1, SENSOR_2, SENSOR_3과 같은 기호상수를 사용할 수 있다.

- 센서 종류 지정

  SetSensor(*sensor*, *config*)

  *sensor*로 지정된 센서의 종류를 *config*로 지정한다. *config* 인자에는 접촉 센서를 나타내는 SENSOR_TOUCH와 빛 센서를 나타내는 SENSOR_LIGHT 기호상수를 사용할 수 있다.

(2) 모터

- 모터 동작/정지

  On(*motors*),

  Off(*motors*)

  *motors*로 지정된 모터들을 켜거나 정지시킬 때 사용한다. *motors* 인자에는 각각 RCX의 A, B, C번 포트에 연결된 모터를 나타내는 OUT_A, OUT_B, OUT_C와 같은 기호상수를 사용할 수 있다. 모터 지정할 때, + 연산자를 사용하여 여러 개의 모터를 한꺼번에 지정할 수도 있다. 예를 들어, A 포트와 C 포트에 연결된 두 개의 모터를 동시에 지정하려면 OUT_A+OUT_C와 같이 쓴다.

- 모터 회전방향 지정

  SetDirection(*motors*, *dir*)

  *motors*로 지정된 모터의 회전방향을 *dir*로 지정한다. *dir* 인자에는 각각 정방향, 역방향, 방향전환을 나타내는 OUT_FWD, OUT_REV, OUT_TOGGLE와 같은 기호 상수를 사용할 수 있다.

  Fwd(*motors*),

  Rev(*motors*),

  Toggle(*motors*)

  각각 SetDirection(*motors*, OUT_FWD), SetDirection(*motors*, OUT_REV), SetDirection(*motors*, OUT_TOGGLE)과 같은 기능을 수행한다.

- 모터 회전속도 지정

  SetPower(*motors*, *power*)

  *motors*로 지정된 모터의 회전속도를 *power*로 지정한다. *power* 인자에는 각각 최소 속도, 중간 속도, 최대 속도를 나타내는 OUT_LOW, OUT_HALF, OUT_FULL와 같은 기호상수를 사용할 수 있다.

3.3 기존 언어와의 비교

2.2절에서 소개한 NQC는 RCX 제어용 언어들 중에서 가장 널리 사용되고 있으며 RCX를 완전히 제어할 수 있는 기능을 지원하므로, 새로운 RCX 제어용 언어와 컴파일러를 설계할 때 좋은 참고가 된다. LegoC 언어는 NQC의 기능 중의 일부를 구현한 언어이다. LegoC 언어의 확장이나 새로운 제어 언어의 설계를 위해서 NQC에서는 지원되나 LegoC에는 없는 기능을 간략히 소개하면 다음과 같다.

- 여러 태스크를 정의 및 태스크의 기동(start)와 정지
- inline 함수의 정의와 '참조전달(call by reference)'에 의한 매개변수 전달
- RCX 서브루틴의 지원
- 배열(array)

- for-문, repeat-문, switch-문, until-문, break, continue
- 논리 연산자(and, or, not)
- ++, --, %, <<, >> 등과 같은 연산자
- RCX를 정교하게 제어하기 위한 내장 함수들
- 다양한 컴파일러 명령행(command line) 옵션

NQC에서는 지원되지 않으나 LegoC에서 확장을 고려해 볼만한 기능은 다음과 같다.

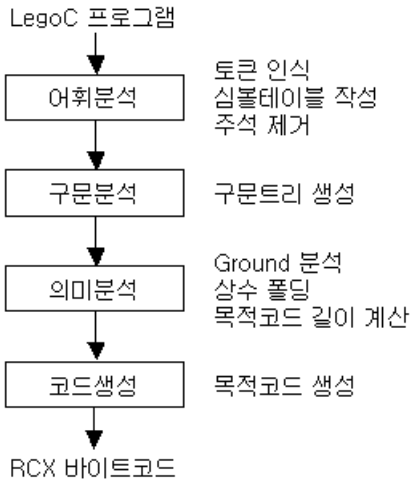
- 부울(boolean) 자료형의 지원
- 반환값을 가지는 함수

4. 프로토타입 컴파일러의 구현

LegoC 언어의 프로토타입 컴파일러를 Linux Red Hat 7.3 환경에서 구현하였다. 도구로는 gcc, lex, yacc 등을 사용하였다. 컴파일된 프로그램을 RCX로 다운로드 하기 위하여 Kekoa Proudfoot의 send 프로그램[10]을 이용하였다.

4.1 컴파일러의 구조

프로토타입 컴파일러 제작의 가장 중요한 목적은 마인드스톰을 교육적으로 활용할 수 있음을 보이고, 컴파일러 수업에서 제작할 컴파일러의 원형을 만드는데 있다. 따라서 구현된 컴파일러는 가장 기본적인 기능만을 수행하도록 설계되었다.



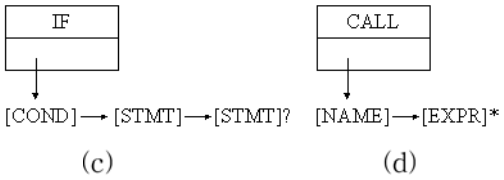
(그림 5) LegoC 컴파일러의 구조

4.2 구문트리

구문분석의 결과로 하나의 구문트리(abstract syntax tree)를 생성한다. 구문트리의 노드의 종류는 다음과 같으며, 몇 개의 노드에 대하여 부분 트리의 구조를 (그림 6)에 제시하였다.

PLUS, MINUS, TIMES, NEG, DIVIDE, EQ, NE, GE, LE, NAME, NUMBER, ASSIGN, IF, WHILE, CALL, COMP, MAIN, VARDECL, DECL, INT

(a) (b)



(그림 6) LegoC 구문트리의 일부

(그림 6)의 (a)에는 덧셈을 의미하는 PLUS 노드가 두 개의 자식 EXPR 노드를 가짐을 나타내고, (그림 6)의 (b)에는 배정문을 의미하는 ASSIGN 노드가 두 개의 자식을 가지는데 첫 번째는 NAME 노드이며, 두 번째는 EXPR 노드임을 나타낸다. (그림 6)의 (c)에는 if-문을 의미하는 IF 노드가 두 개 또는 세 개의 자식 노드를 가짐을 나타낸다. 그림에서 "?" 기호는 해당 노드가 있을 수도 있고 없을 수도 있음을 나타내는데 이는 if-문에 "else"의 유무에 의하여 결정된다. (그림 6)의 (d)에는 함수호출을 의미하는 CALL 노드가 한 개 이상의 자식 노드를 가짐을 나타낸다. 그림에서 "*" 기호는 해당 노드가 0번 이상 나타날 수 있다는 표시인데, 이는 함수호출의 인자로 식이 0개 이상 주어질 수 있음에 해당한다.

4.3 의미분석

구문분석에 의하여 생성된 구문트리에 대하여 두 가지 의미 분석을 수행하였다. 첫번째로, groundness 분석으로 식이 상수를 가지는지를 판단하여 최적화와 코드 생성에 이용하였다. RCX의 "비교 후 점프" 명령어에서 두번째 피연산자(operand)는 상수가 될 수 없다는 제한점이 있는데, groundness 분석에 의하여 두 개의 피연산자가 모두 상수라면 두번째 피연산자를 임시 변수에 저장한 다음 "비교 후 점프" 명령어에서는 이 임시 변수를 두번째 피연산자로 사용해야 하며, 첫번째 피연산자는 상수가 아니면서 두번째 피연산자가 상수라면 두 개의 피연산자를 맞바꾸는 것이 필요하다.

또한 groundness 분석에 의하여 식이 상수들의 계산으로 이루어져 있다고 판단되면 식을 컴파일-시간에 계산하는 상수-계산(constant-folding) 최적화를 수행할 수 있다.

두번째로, 번역될 코드의 길이를 목적코드의 생성 이전에 계산함으로써 점프 명령어에서의 상대주소를 정확히 하는 분석을 구현하였다. LegoC의 각 실행문에 의해서 생성되는 목적코드의 길이는 다음의 표와 같다.

<표 2> 각 실행문의 목적코드 길이

실행문	목적코드의 길이 (byte)
배정문	5
If-then	[then 파트의 길이] + 11
If-then-else	[then 파트의 길이] + [else 파트의 길이] + 14
While-문	[실행 파트의 길이] + 14
복합문	모든 문장의 길이의 합
함수호출	함수 종류에 따라 다름 (2~5)

4.4 코드생성

코드생성 단계에서는 구문분석에 의해서 생성되고 의미분석에 의하여 수정된 구문트리를 순회하면서 RCX 바이트코드를 생성한다. 현재의 구현에서는 식에 대한 코드는 의미분석의 상수계산 단계에서 처리되므로 코드생성은 실행문(statement)에 대해서만 실행된다.

배정문은 RCX 바이트코드[11, 12]의 "set variable(0x14)" 명령어로 번역된다. If-문에

대한 코드는 다음과 같은 의사코드(pseudo code)의 절차에 의해 생성된다.

```
1. "cond" 부분에 대한 코드 생성
2. 아래 4번의 주소로 조건부 점프 코드
3. 아래 7번의 주소로 무조건 점프 코드
4. "then" 부분에 대한 코드 생성
5. "else" 부분이 존재하면 6-7을 수행
6. 아래 8번의 주소로 무조건 점프 코드
7. "else" 부분에 대한 코드 생성
8. <다음 문장>
```

위의 절차에 의하여 생성된 코드가 실행될 때, "cond"의 값이 참이면 4번과 6번의 코드가 실행되고, 거짓이면 3번과 7번의 코드가 실행된다.

While-문에 대한 코드는 다음과 같은 의사코드의 절차에 의해 생성된다.

```
1. "cond" 부분에 대한 코드 생성
2. 아래 4번의 주소로 조건부 점프 코드
3. 아래 6번의 주소로 무조건 점프 코드
4. "stmt" 부분에 대한 코드 생성
5. 위 1번의 주소로 무조건 점프 코드
6. <다음 문장>
```

위에서 제시한 if-문과 while-문을 위한 코드생성 절차에서 2번의 조건부 점프 코드는 1번의 "cond"가 참인 경우에 점프를 실행한다. 만약에 "cond"가 거짓인 경우에 점프를 하거나 또는 "cond"의 역(not)이 참인 경우에 점프하도록 코드를 생성한다면, 2번과 3번의 두 개의 점프 코드를 하나의 점프 코드로 줄일 수 있다. 그러나 RCX 표준 펌웨어에서는 조건이 거짓인 경우에 점프를 실행하는 명령어는 존재하지 않는다. 또한 조건을 생성하기 위하여 식을 비교하는 경우에 '크거나 같음'과 '작거나 같음'은 사용할 수 있지만 '더 크다'와 '더 작다'를 비교하는 방법은 제공되지 않는다. 즉, (a >= b)를 (not (a < b))로 동등하게 변환하는 것이 쉽지 않다.

구현한 LegoC 언어에서 사용자가 함수를 정의할 수 없고 내장 함수만을 호출해야 한다는 제약이 있으므로, 함수호출 문장은 내장 함수에 따른 RCX 바이트코드를 직접 생성한다. 각 함수에 따른 코드 생성의 규칙은 다음의 표와 같다.

<표 3> 내장함수와 RCX 명령어와의 관계

내장 함수	RCX code
SensorValue	set variable (0x14)
SetSensor	set sensor mode (0x42)
On, Off	set motor on/off (0x21)
SetDirection, Fwd, Rev, Toggle	set motor direction (0xe1)
SetPower	set motor power (0x13)

4.5 구현 결과

제작된 컴파일러의 동작을 검증하기 위하여 '선추적(line tracing)' 로봇을 조립하여 동작시켜 보았다. 선추적 로봇은 빛 센서로부터 바닥의 색을 감지하여 검은색과 흰색의 경계선을 따라간다.

선추적 로봇을 제어하는 LegoC 프로그램을 (그림 7)에 제시하였다. 빛 센서로부터 읽은 값(light_value)이 특정 값 left_threshold 보다 작으면(즉, 검은색이면) 왼쪽 바퀴를 정지하고 오른쪽 바퀴만을 움직여서 로봇을 왼쪽으로 회전하게 하고(첫번째 if-문), 빛 센서의 흰색을 감지한다면 로봇을 오른쪽으로 움직인다(두번째 if-문).

```
int left_threshold, right_threshold;
int light_value;

task main
{
    left_threshold = 44;            // 어두움
    right_threshold = 53;           // 밝음
    SetSensor(SENSOR_2, SENSOR_LIGHT);
    SetPower(OUT_A+OUT_C, OUT_HALF);
    On(OUT_A+OUT_C);

    SensorValue(SENSOR_2, light_value);
    while (left_threshold <= 50) { // 무한 루프
      if (light_value <= left_threshold) {
          Off(OUT_A);
          On(OUT_C);
      }
      else if (light_value >= right_threshold) {
          Off(OUT_C);
          On(OUT_A);
      }
      else {
          On(OUT_A+OUT_C);
      }
      SensorValue(SENSOR_2, light_value);
    }
}
```

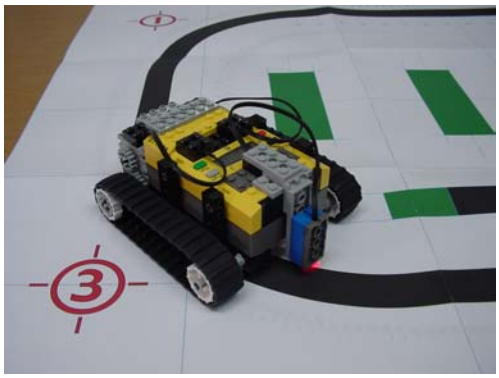
(그림 7) LegoC로 작성한 선추적 프로그램

(그림 8)은 (그림 7)의 LegoC 프로그램을 컴파일하여 생성된 RCX 바이트코드를 보여준다. 이 코드는 로봇으로 전송되어 로봇을 제어한다.

```
91 04
25 00 00 00 57 00
45 00 00 57 00 13 07 02 07 e1 87 14 00 02
2c 00 14 01 02 35 00 32 01 03 13 05 02 03
21 85 14 02 09 01 00 95 42 00 32 00 00 05
00 72 30 00 95 00 00 02 00 00 05 00 72 09
00 21 41 21 84 72 16 00 95 40 00 02 00 01
05 00 72 09 00 21 44 21 81 72 04 00 21 85
14 02 09 01 00 72 b8 00 00
```

(그림 8) 생성된 바이트코드

(그림 9)는 (그림 8)에서 생성된 코드에 의하여 움직이는 로봇을 찍은 사진이다.



(그림 9) 선추적 로봇의 동작

제작된 프로토타입 컴파일러의 소스는 http://pl.changwon.ac.kr/legoc에서 구할 수 있다. 구현된 컴파일러에 대한 간략한 설명과 컴파일러 제작과 관련한 참고자료와 하이퍼링크 역시 위의 사이트에서 얻을 수 있다.

5. 컴파일러 수업에의 활용

레고 마인드스톰과 구현된 프로토타입 컴

파일러를 컴파일러 수업에 활용하는데 있어서 가능한 몇 개의 프로젝트를 본 장에서 제시한다.

(1) 언어 확장 및 설계

- <Exp>에 함수호출 허용
  식에 함수호출을 허용하게 되면 함수는 반환값을 돌려주도록 변경되어야 한다. 예를 들어,

  SensorValue(SENSOR_2, n);

  와 같은 문장은

  n = SensorValue(SENSOR_2);

  와 같이 쓸 수 있다. 아울러 사용자 정의 함수를 제공할 것을 고려해야 하며, NQC의 경우와 같이 inline 함수도 고려해야 한다. 또한 RCX에서 제공되는 8개의 서브루틴 역시 지원하여야 한다.
- 자료형 확장
  부울(boolean)형과 배열형을 지원하는 프로젝트도 흥미롭다. 부울형은 if-문이나 while-문의 조건식에서 사용될 수 있으며, 배열형은 표준 펌웨어 2.0에서 지원되고 있다.
- 새로운 컴파일러 개발
  LegoC 언어가 아닌 다른 언어에 대한 컴파일러를 개발할 수도 있다. 일례로 자바 프로그램을 표준 펌웨어에서 실행하는 과제도 가능하다.

(2) 타겟 확장

- 표준 펌웨어 2.0을 지원하는 LegoC 컴파일러를 제작할 수 있다. 또한 2.3절에서 소개한 여러 펌웨어를 위한 컴파일러의 제작도 재미있는 과제가 될 것이다.

(3) 기능 확장

- 최적화 구현
  RCX에서 사용할 수 있는 변수의 개수는 매우 적으므로, 변수를 최대한 효율적으로 사용하기 위한 “변수 할당”최적화가 필요하다. 또한 표준 펌웨어의 경우에 사용자 프로그램은 6KB를 넘을 수 없으므로, 목적코드의 길이를 줄이는 최적화는 상당한 효과가 있을 것이다.
- 통신 기능의 확장
  PC와 RCX와의 통신 또는 RCX 끼리의 통신이 가능하도록 LegoC 언어를 확장할 수 있다.
- 전처리기와 연동
  매크로, 조건부 컴파일 등을 처리할 수 있도록 C의 전처리기(preprocessor)를 컴파일러 내부에서 호출할 필요가 있다.

(4) 보조도구의 개발

- 어셈블러의 개발
  구현된 프로토타입 컴파일러는 목적코드를 직접 생성하고 있다. 그러나 컴파일러에서 어셈블리 코드를 생성하고 목적코드의 생성은 어셈블러가 담당하는 것이 일반적인 컴파일러의 구조이다. 만약 LegoC를 위한 어셈블러가 존재한다면 컴파일러의 복잡도가 훨씬 낮아질 것이다.
- 정적분석기의 개발
  RCX의 전역변수는 모든 프로그램과

태스크에서 공유되기 때문에, 프로그램에서 사용하는 변수가 초기화가 되었는지를 검사하는 것이 필요할 것이며, 이는 정적분석으로 가능하다. 또한 LegoC 프로그램에 대한 통계산출이라든지 자동 들여쓰기(indentation) 등도 유용한 도구가 될 것이다.

- 웹 인터페이스
  웹 인터페이스를 통하여 원격으로 레고 로봇을 제어하도록 만들 수 있다[13]. 이를 통하여 레고 로봇을 가지지 않은 경우에도 로봇 실험이라든지 로봇 프로그래밍을 가능하게 한다. 이 프로젝트에는 웹 서버 기술과 웹 프로그래밍 기술이 중요한 역할을 할 것이다.

6. 결론

본 논문에서는 레고 마인드스톰을 제어하기 위한 LegoC 언어를 설계하였고 그의 컴파일러를 구현하였으며, 제작된 컴파일러가 올바르게 동작함을 실제의 레고 로봇을 통하여 확인하였다.

장난감을 실습장비로 사용하는데 대한 거부감을 극복한다면, 레고 마인드스톰은 학생들에게 흥미와 동기를 유발할 수 있는 훌륭한 도구가 될 것이며, 컴파일러 교육에 큰 효과가 있을 것으로 기대된다. 이에 본 논문에서는 컴파일러 수업에서 레고 마인드스톰을 활용할 수 있는 몇 가지 방안을 제시하였다.

레고 마인드스톰은 범용 컴퓨터라기보다는 일종의 내장형 시스템(embedded system)이므로, 컴파일러 수업에서 이를 활용한 경험은 내장형 시스템을 이해하는데 도움을 줄 것이다.

참고문헌

[1] Programming Languages Lab., "The P-machine Simulator," Internet Resource, 1997. (http://pl.changwon.ac.kr/compiler/cg/vmSim.html)

[2] Lego, "MindStroms," Internet Resource. (http://www.legomindstorms.com/)

[3] G. Ferrari, et. al., *Programming Lego Mindstroms with Java*, 2002, Syngress.

[4] Lego, *LEGO MindStorms RCX 2.0 Firmware Command Overview*, 2000.

[5] D. Baum, “Not Quite C,” Internet Document. (http://www.enteract.com/~dbaum/lego/nqc)

[6] M. Noga, "legOS," Internet Document. (http://www.noga.de/legOS)

[7] R. Hempel, "pbForth Weblog Page," Internet Document, 2002. (http://www.hempeldesigngroup.com/lego/pbForth/)

[8] J. Solorzano, "TinyVM - Java VM for Lego Mindstorms RCX," Internet Document, 2000. (http://tinyvm.sourceforge.net)

[9] J. Solorzano, "Java Based Lego OS for RCX," Internet Document, 2000. (http://lejos.sourceforge.net)

[10] K. Proudfoot, "RCX Tools," Internet Document, 1999. (http://graphics.stanford.edu/~kekoa/rcx/tools.html)

[11] K. Proudfoot, "RCX Opcode Reference," Internet Document, 1999. (http://graphics.stanford.edu/~kekoa/rcx/opcodes.html)

[12] K. Proudfoot, "RCX Internals," Internet Document, 1999. (http://graphics.stanford.edu/~kekoa/rcx/)

[13] S. Ireland, "WebRCX: Control a LEGO RCX brick from the web," Internet Document, 2001. (http://www.sckans.edu/~sireland/webcam/webrcx/)

이수현

1987년 2월 광운대학교 전자계산학과 졸업(학사)
1989년 2월 한국과학기술원 전산학과 졸업(석사)
1994년 8월 한국과학기술원 전산학과 졸업(박사)
1994년 9월 ~ 1996년 2월 한국전자통신연구원 선임연구원
1996년 3월 ~ 현재 창원대학교 컴퓨터공학과 부교수
관심분야 : 프로그래밍언어, 제한프로그래밍, 생명정보학 등.