논·문15-1-3

Mobile C 컴파일러에서의 에러 복구

(Error Recovery in Mobile C Compiler)

오 세 만, 김 정 숙

동국대학교 컴퓨터공학과, 삼육의명대학 인터넷과

smoh@dgu.ac.kr, kimjs@syu.ac.kr

요 약

에러 복구(Error Recovery)란 파싱하는 도중에 에러가 일어났을 때, 계속하여 파싱할 수 있도록 문법적인 에러를 처리하는 작업을 말한다. 이것은 소스 프로그램에 있는 문법적인 에러를 가능한 많이 찾아서 보고하는 것을 목적으로 하며 이와 같은 처리는 컴파일러 제작에 있어서 매우 중요한 일이다. 에러 복구에 관한 연구는 이론적으로는 많은 연구 결과가 있었으나 실질적인 방법이 되지 못하고 적용하기에는 많은 제약점이 있다.

본 연구에서는 LR 파서에서 간단하면서도 실질적으로 적용할 수 있는 에러 복구 방법을 제안하고자 한다. 이 방법은 크게 2가지 경우로 구분하여 처리하며 서로 보완적인 역할을 한다. 첫째, 패닉 방법(panic method)은 입력의 일정 부분을 삭제하고 상태 스택을 조정하는 방법으로 에러가 일어난 모든 경우에 적용할 수 있는 방법이다. 둘째, 수정 방법(modifying method)은 직접 파싱 테이블을 수정하여 해당하는 에러를 개별적으로 처리하는 방법이다.

본 논문에서 제안한 에러 복구 방법은 실질적인 규모의 컴파일러에 쉽게 적용할 수 있을 뿐 아니라 컴파일러 수업 시간에 제작하는 실험용 컴파일러에도 간단하게 적용할 수 있는 방법이다. 또한, Mobile C 프로그램을 GVM(**G**ame **V**irtual **M**achine) 코드[5]로 번역하는 실제적인 규모의 컴파일러에 제안된 방법을 사용하여 실험하였다[3,4].

1. 서론

프로그래밍 속성 상 소스 프로그램에는 에러가 있게 마련이고 컴파일러는 항상 올바른 프로그램만 입력으로 받는 것이 아니고 에러가 있는 프로그램을 만나게 된다. 이와 같은 경우가 발생하면, 컴파일러는 문법의 생성 규칙에 따라 반드시 처리해 주어야 한다. 이와 같이 컴파일 도중에 에러가 일어난 경우에 계속해서 파싱을 할 수 있도록 복구해 주는 작업을 에러 복구라 한다.

컴파일러를 자동적으로 제작하려는 연구가 활발히 진행되고 에러 복구가 컴파일러 작성에 있어서 꼭 필요한 작업이기 때문에 자동적인 파싱 방법(automatic parsing method)에 적용하기 위한 많은 연구가 있었으며 연

구 결과도 상당히 바람직한 수준에 도달하였다. 그러나, 실질적인 효과에 비해 이론이 너무 복잡하고 적용하기에도 많은 제약점을 가지고 있다.

본 연구에서는 LR 파서에서 쉽게 적용할 수 있는 실질적인 에러 복구 방법을 제안하고자 하며 또한, 실질적인 규모의 컴파일러를 제작하는데 이 방법을 적용하였다. 본 연구에서 제안한 방법은 4가지 경우로 나누어 처리하며 서로 보완적인 작업이다.

첫째, 패닉 방법은 선언 구분이나 문장 구분 토큰이 나올 때까지 입력을 삭제한 후 파서의 상태 스택을 조정하여 삭제 이후의 토큰이 타당한 토큰(valid token)이 되도록 처리하는 방법이다. 대부분의 프로그래밍 언어는 선언 부분과 문장 부분으로 구성되어 있으며 상태 스택(state stack)은 현재 파싱하는 부분에 대한 과거 정보를 갖고 있기 때문에 이 정보를 이용하여 상태 스택을 쉽게 조정할 수 있다.

둘째, 삽입 방법은 하나의 토큰이 빠졌을 때, 그 토큰을 삽입해 줌으로써 올바른 프로그램이 되도록 처리해 주는 방법이다. 그러나, 상태에 따라 어떤 종류의 토큰을 삽입해 주어야 하는가가 문제이나 과거 정보와 현재 토큰의 종류에 따라 결정하였다. 현재 삽입 방법으로 처리한 에러 경우의 수는 11가지이며 27가지 행동으로 처리하였다.

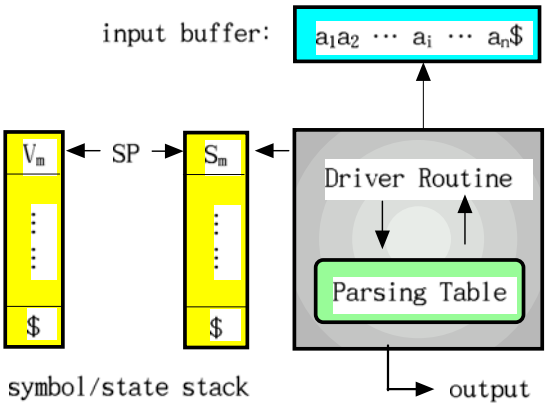
셋째, 삭제 방법은 불필요한 토큰이 발견되었을 때, 그 토큰을 삭제하여 처리하는 방법이다. 예를 들어, if (a > b)) max = a; 에서 불필요한)를 삭제하여 파싱을 계속하는 방법이다.

넷째, 대치 방법은 에러가 된 토큰이 발견되었을 때, 그 상태에서 올바른 토큰으로 대치하는 방법을 말한다. 예를 들어, i = i + j: 과 같이 ;이 나와야 하는데 :이 나온 경우 : 을 ;으로 대치하여 파싱을 계속하도록 처리한다.

이와 같은 방법을 적용하여 문법적인 에러를 처리한 결과 대부분의 경우 한번의 파싱 과정을 통하여 프로그램에 있는 에러를 모두 검출할 수 있었다.

2. LR 파서

LR 파서란 주어진 입력 프로그램을 왼쪽에서 오른쪽으로 읽어 나가면서 bottom-up 방법을 사용하여 결정적으로 파싱하는 구문 분석기를 말하며 스택과 파싱 테이블을 이용하여 구문 분석을 수행한다.



[그림 1] LR 파서

파싱 스택은 에러 복구를 원활히 수행하기 위해 병렬로 운영되는 두 개의 스택으로 구성되어 있으며 심벌 스택에는 일련의 문법 심벌이 있으며 상태 스택에는 파싱 과정에 필요한 상태들을 유지한다. LR 파서의 주요 행동은 shift, reduce, accept, error이며 한 상태에서 입력 심벌에 따라 파싱 테이블을 참조하여 결정된다.

shift 행동이란 입력 버퍼에 있는 현재의 입력 심벌을 스택 top으로 이동하는 것을 의

미하며 reduce 행동이란 생성 규칙의 왼쪽을 오른쪽으로 대치하는 작업이다. accept는 주어진 입력이 올바른 스트링이라는 의미이며 파싱을 끝낸다. 그리고 error 행동이란 주어진 입력 스트링이 틀린 스트링이라는 의미이며 계속해서 파싱을 하기 위해선 반드시 에러 복구 루틴을 불러 일어난 에러를 처리해야만 한다.

파서가 에러 행동을 한다는 의미는 주어진 입력 프로그램에 에러가 있다는 의미이며 이 경우에 파싱을 계속하기 위해선 반드시 일어난 에러를 바르게 고쳐주어야 한다. 본 논문에서 제안한 에러 복구 방법은 파서의 에러 행동에 적용할 수 있는 방법이다.

파싱 테이블의 구조[2]는 2차원 배열로 행은 terminal과 nonterminal 심벌들이며, 열은 상태수가 된다. 구문 분석기의 행동은 열과 행이 만나는 곳에 정의되는데 **shift**되는 상태는 양수로, **reduce**되는 생성 규칙 번호는 음수로 나타낸다. 그리고 **accept**는 추가된 생성규칙(augmented production)으로 reduce 될 때이며, **error**는 0으로 표시한다.

파싱 테이블의 entry가 0일 때, 패닉 방법이 적용되며 ;을 만날 때까지 입력을 삭제하고 상태 스택을 조정한다. 그리고 수정 방법은 에러 상황과 입력 심벌에 대해 고유의 에러 번호를 배정하고 이 번호를 직접 파싱 테이블에 수정한다. 고유의 에러 번호는 shift 행동에서 구분되도록 상태 수보다 큰 수로 정한다. 따라서, entry가 상태 수보다 크면 그에 해당하는 에러 처리 루틴을 불러 처리하게 된다. 다음은 에러 처리가 포함된 파서의 행동을 알고리즘으로 나타낸 것이다.

```
while (true) {
 current_state = stateStack[sp];
 entry = pt[current_state][token.number];
 if (entry > 0) {
  if (entry > NO_STATES) {
     // error recovery action
     syntaxError(entry);
     continue;
  }
  else {  // shift action
   sp++;
   symbolStack[sp] = token.number;
   stateStack[sp] = entry;
  }
 else if (entry < 0) { // reduce action
  ruleNumber = -entry;
  if (ruleNumber == GOAL_RULE)
  // accept action: announce acceptance
  // ... reduction
 else // error action
     errorRecovery();
}
```

[그림 2] Parsing Algorithm with Error Recovery

여기서, syntaxError() 함수는 삽입 방법과 삭제 방법 그리고 대치 방법을 처리하는 작업을 행하며 errorRecovery() 함수는 패닉 방법을 처리하는 함수이다.

3. 패닉 방법

하나의 소스 프로그램은 일련의 선언과 문장들로 구성되어 있다. 그리고 각각의 선언과 문장은 ;(semicolon)으로 구분되어 있다. 따라서, 에러가 일어났을 때 먼저 ;을 만날 때까지 입력을 삭제하고 상태 스택을 조정하여 다음 선언이나 문장에 대한 파싱을 계속할 수 있도록 에러를 처리하는 방법이다.

패닉 모드를 구현하기 위해서는 먼저, 파서 생성기(PGS: Parser Generating System)의 출력인 리스트 파일을 조사하여 선언의 시작 상태와 문장의 시작 상태 등을 알아야 한다. 그리고 선언 부분을 파싱하는 도중에

일어난 경우와 문장 부분을 파싱하는 도중에 일어난 경우로 나누어 처리해야 한다.

3.1. 선언 부분에서의 패닉처리

선언 부분을 파싱하는 도중에 에러가 일어나면, ;을 만날 때까지 입력을 스킵하여 현재의 선언문에 대한 파싱을 끝내고 다음 선언문에 대한 파싱을 시작할 수 있는 상태로 조정한다. 선언의 시작 상태는 LR(0) 아이템에서 . declaration이 있는 상태이다. 그런데 문법에서 . declaration이 있는 상태는 소스 프로그램의 파싱 위치에 따라 여러 곳이 존재한다.

외부 선언을 파싱하는 도중에는 첫 번째 선언문을 파싱하는 경우에 에러가 일어난 상태와 두 번째 선언문 이후에 에러가 일어난 상태가 다르다. 또한, 함수 내에 있는 선언부분을 파싱할 때 에러가 일어난 경우도 선언의 시작 상태는 다르다. 따라서, 상태 스택을 찾아 경우에 따라 에러 처리 후의 상태를 다르게 세팅해야 한다.

현재, 설계한 Mobile C 문법[4]에서는 다음 4가지 경우에 . declaration이 있는 상태가 존재한다.

▶ 0번 상태 : <첫 번째 외부 선언인 경우>
translation_unit -> . external_dcl
translation_unit -> . translation_unit external_dcl
external_dcl -> . declaration

▶ 4번 상태 : <외부 선언이 반복되는 경우>
translation_unit -> translation_unit . external_dcl
external_dcl -> . declaration
상태 스택 : 0 4 8 43

▶ 43번 상태 : <첫 번째 내부 선언>
declaration_list -> . declaration
declaration_list -> . declaration_list declaration

▶ 60번 상태 : <내부 선언이 반복되는 경우>
declaration_list -> declaration_list . declaration
상태 스택 : 0 4 8 43 60 62 48 66 64

3.2. 문장 부분에서의 패닉처리

문장 부분을 파싱하는 도중에 에러가 일어나면, 문장의 끝을 나타내는 토큰인 ;을 만날 때까지 입력을 삭제하고 다음 문장을 파싱할 수 있는 상태로 조정한다. 문장의 시작 상태는 LR(0) 아이템에서 . statement가 있는 상태이다.

현재, 설계한 Mobile C 문법에서는 다음과 같은 2가지 상태에 . statement가 존재한다.

▶ 59번 상태 : <<첫 번째 문장인 경우>>
statement_list -> . statement_list statement
statement_list -> . statement

▶ 78번 상태 : <<두 번째 문장 이후>>
statement_list -> statement_list . statement
상태 스택 : 0 4 8 39 59 78 84 179 257 297

Mobile C 문법에서 문장 부분은 함수의 몸체 부분이나 복합문(compound statement)이 될 수 있는데 두 부분의 처리는 모두 동일하다.

3.3. 패닉처리를 위한 알고리즘

파싱하는 과정에서 에러가 일어난 시점까지 처리했던 히스토리 정보가 모두 상태 스

택에 있기 때문에 현재 어느 부분을 파싱하고 있는가를 알아내는 것은 매우 쉬운 일이다. 따라서, 상태 스택을 역순으로 찾아 현재의 상태를 세팅하면 되는 것이다. 이때 찾고자하는 상태는 먼저 . statement가 있는 상태이며 그 다음에 . declaration이 있는 상태이다.

패닉 처리를 위한 알고리즘은 먼저, ;을 만날 때까지 입력을 스킵하고 상태 스택을 역순으로 찾아 현재의 상태를 세팅하면 되는 것이다. 이와 같은 과정을 알고리즘으로 표현하면 다음과 같다.

```
void errorRecovery()
{
 struct tokenType tok;
 int parenthesisCount, braceCount;
 int i;

 // step 1: skip to the semicolon
 parenthesisCount = braceCount = 0;
 while (1) {
  tok = scanner();
  if (tok.number == teof) exit(1);
  if (tok.number == tlparen)
      parenthesisCount++;
    else if (tok.number == trparen)
      parenthesisCount--;
  if (tok.number==tlbrace) braceCount++;
    else if (tok.number == trbrace)
                  braceCount--;
  if ((tok.number==tsemicolon) &&
  (parenthesisCount<=0)&&(braceCount<=0))
     break;
 }

 // step 2: adjust state stack
 for (i=sp; i>=0; i--) {
 // statement part
  if (stateStack[i] == 78) break;
  if (stateStack[i] == 59) break;

  // declaration part
  if (stateStack[i] == 60) break;
  if (stateStack[i] == 43) break;
  if (stateStack[i] == 4) break;
  if (stateStack[i] == 0) break;
 }
 sp = i;
 token = scanner();
}
```

[그림 3] Algorithm for Handling Panic Mode

세미콜론을 만날 때까지 스킵하는 도중에 여는 괄호를 만나면 반드시 닫는 괄호 이후에까지 삭제가 이루어져야지 그 후에 그것으로 인하여 새로운 에러가 발생하지 않는다.

4. 수정 방법

수정 방법(modifying method)이란 직접 파싱 테이블을 수정하여 해당하는 에러를 개별적으로 처리하는 방법이다. 각각의 에러 상황에 대해서 고유의 에러 번호를 배정하고 이 번호를 파싱 테이블에 반영하는 것이다. 수정 방법에는 삽입 방법, 삭제 방법, 그리고 대치 방법 등이 있다.

삽입 방법(insertion method)은 프로그래머가 흔히 실수를 하는 콤마나 세미콜론 등을 빠뜨렸을 때 하나의 토큰을 삽입하여 처리하는 방법이며, 삭제 방법(deletion method)은 불필요한 하나의 토큰을 제거하는 방법이다. 그리고 대치 방법(replacement method)은 에러가 일어난 토큰을 제거하고 올바른 토큰으

로 대치하는 방법이다.

4.1. 삽입 방법

삽입 모드로 처리하는 방법은 문장의 끝을 나타내는 ;이나 명칭 리스트에서 ,가 빠졌을 경우 해당하는 토큰을 삽입하여 처리한다. 즉, 하나의 토큰이 빠진 경우 에러 상황을 분석하여 올바른 토큰을 삽입하는 방법이다.

그러나, 상태에 따라 어떤 종류의 토큰을 삽입해 주어야 하는가가 문제이나 상태 스택 정보와 현재 토큰의 종류에 따라 결정하였다. 예를 들어, 선언 int i에서와 같이 ;이 빠진 경우에 상태 53번에서 에러가 일어나며 현재 토큰이 다른 선언의 시작이나 문장의 시작이 오는 경우에 ;을 삽입하는 행동을 파싱 테이블에 반영하는 것이다.

Mobile C 컴파일러에서 고려한 토큰의 종류는 ;, „ (,), {, }, =, 연산자, int 등이며 이들의 조합으로 이루어진 경우도 취급되었다. 또한, ;인 경우에도 8가지 상태에서 다루어졌으며 전체적으로 현재 삽입 방법으로 처리한 에러 경우의 수는 11가지이며 27가지 행동으로 처리하였다.

삽입 처리 방법에서 각 에러 경우에 따른 에러 번호를 1001번부터 고유 번호를 배정했으며 그에 따른 처리 방법을 알고리즘으로 기술하면 다음과 같다.

```
struct tokenType syntaxError(int errno)
{
  struct tokenType token;
  int previousState;
  switch (errno) {
  case 1001:            // semicolon
     errmsg(" ';' expected");
     token.number = tsemicolon;
     break;
  case 1002:            // comma
     errmsg(" ',' expected");
     token.number = tcomma;
     break;
     // ... ... handle each case
  }
  return token;
}
```

[그림 4] Algorithm for Insertion Method

4.2. 삭제 방법

삭제 방법이란 불필요한 토큰이 발견되었을 때, 그 토큰을 삭제하여 처리하는 방법이다. 예를 들어, if (a > b)) max = a;와 같이 괄호가 두개 겹쳐 에러가 일어난 경우에 하나를 삭제하여 파싱을 계속하는 방법이다.

Mobile C 컴파일러에서 고려한 토큰의 종류는)(parenthesis) ,(comma) }(brace) 등이며 현재 삭제 방법으로 처리한 에러 경우의 수는 3가지이며 4가지 행동으로 처리하였다.

삭제 처리 방법에서 각 에러 상황에 따른 에러 번호를 2000번부터 고유 번호를 배정했으며 그에 따른 처리 방법을 알고리즘으로 기술하면 다음과 같다.

```
  case 2000: // discard right parenthesis
     errmsg(" ')' is not necessary");
     token = scanner();
     saveToken = scanner();
     break;
  case 2001: // discard comma
     errmsg(" ',' is not necessary");
     token = scanner();
     saveToken = scanner();
     break;
```

```
case 2002: // skip to next function def
    errmsg(" '}' is not necessary");
    token = skipToFuncDef();
    saveToken = scanner();
    break;
```

[그림 5] Program Segment for Deletion Method

4.3. 대치 방법

대치 방법이란 한 상태에서 틀린 토큰을 찾아내어 분석한 후 그 상태에서 바른 토큰으로 대치해 주는 방법이다. 예를 들어, 문장의 끝을 나타내는 ;(semicolon)을 :(colon)으로 잘못 타이핑한 경우에 :을 ;으로 대치하여 처리하는 방법이다.

대치 방법으로 Mobile C 컴파일러에서 고려한 경우는 ;이 나와야 하는데 :이 나온 경우나 또는 .이 나온 경우이다.

대치 처리 방법에서 각 에러 경우에 따른 에러 번호를 3000번부터 고유 번호를 배정했으며 그에 따른 처리 방법을 알고리즘으로 기술하면 [그림 6]과 같다. 앞으로 좀더 많은 대치 상황을 고려해서 처리하는 행동을 추가해야 할 것이다.

```
case 3000:  // replace : with ;
    errmsg(" ':' is replaced ';'");
    token.number = tsemicolon;
    saveToken = scanner();
    break;
case 3001:  // replace . with ;
    errmsg(" '.' is replaced ';'");
    token.number = tsemicolon;
    saveToken = scanner();
    break;
```

[그림 6] Program Segment for Replacement Method

5. 구현 결과

본 논문에서 설명한 에러 처리 방법을 모두 Mobile C에 적용하여 실험하였다. 대부분의 프로그램의 경우에 에러를 정확히 지적해 줄뿐만 아니라 한번의 컴파일로 전체 에러를 찾아 낼 수 있었다.

Mobile C[4]란 휴대용 단말기(HHD: Hand Held Device)에서 실행될 수 있는 다양한 응용 프로그램에 적합하도록 고안된 언어를 말하며 표준 C 언어를 기초로 하였다. 특히, Mobile C 언어는 WAP 게임을 쉽게 제작할 수 있도록 설계되었다. 현재, Mobile C 언어는 많은 CP(Contents Provider) 업체에 의해서 사용되고 있으며 에러 처리 방법에 대해서 대체로 만족하고 있는 상태이다.

다음은 에러가 있는 프로그램을 입력으로 하여 에러 처리를 수행하여 그에 해당하는 에러 메시지를 출력한 리스트 파일이다. 프로그램의 의미는 없으며 단지 에러 복구 방법을 확인하기 위한 것이다.

```
(error.c, 1): int v[10] = { 1, 2, ,3};
 *** error(error.c, 1) ===> ',' is not necessary !!!

(error.c, 2): int x
 *** error(error.c, 2) ===> ';' expected !!!

(error.c, 4): void main()
(error.c, 5): {
(error.c, 6):   int i=0,j=1;
(error.c, 8):   if ((i==0) || (i==1)))
                  i = 100;
 *** error(error.c, 8) ===> ')' is not necessary !!!

(error.c, 10):  switch (i) {
(error.c, 11):  case 1: if (j==1) j=100;
(error.c, 12):  }
```

```
(error.c, 14):  i = i + j;
(error.c, 16):  DrawStr(10, 10, "i = ");
(error.c, 17):  println(i);
(error.c, 18): }
(error.c, 20): int f()
(error.c, 21): {
(error.c, 22):  return v[x]
 *** error(error.c, 22) ===> error in source(panic mode) !!!
```

[그림 7] Example for Error List

6. 결론

본 논문에서 제안한 에러 복구 방법은 이론적으로 간단하며 실질적으로 컴파일러의 에러 처리 과정에 쉽게 적용할 수 있는 방법이다. 또한, 실제적인 규모의 컴파일러(practical-scaled compiler)에 적용할 수 있을 뿐 아니라 컴파일러 수업 시간에 텀 프로젝트로 제작하는 실험용 컴파일러에도 간단하게 적용할 수 있는 방법이다.

제안된 방법은 Mobile C 언어의 컴파일러에 적용하여 쉽게 활용할 수 있는 기법임을 증명하였으며 대부분의 경우에 한번의 컴파일링으로 프로그램에 있는 모든 에러를 검출할 수 있었다.

앞으로 실제적인 컴파일러의 에러 처리 방법으로 활용하기 위해서는 더 많은 경우의 에러 상황을 고려하여 대부분의 문법적인 에러를 패닉 에러로 처리하지 않고 수정 에러로 처리할 수 있도록 해야 할 것이다.

참고 문헌

[1] M. Burke and G.A. Fisher, "*A Practical Method for Syntactic Error Diagnosis and Recovery*," Proceeding of the SIGPLAN Symposium on Compiler Construction, pp.67-78, 1982.

[2] Seman Oh, *Introduction to Compiler Construction*, Jungik Publishing Co., 2000.3.

[3] Seman Oh et al., *Production of a Compiler for the Handheld Device's Game*, Project Report, Research Center for Information Communication, Dongguk University, 2000.8.

[4] Seman Oh and Jungsook Kim, *Extension of SG Compiler*, Project Report, Research Center for Information Communication, Dongguk University, 2001.2.

[5] Sinjisoft, *http://www.gvmclub.com*

오 세만

'93.03-'99.02 동국대학교 컴퓨터공학과 대학원 학과장

'85.03-현재 동국대학교 컴퓨터 공학과 교수

관심분야 : 컴파일러, 프로그래밍 언어, XML, 모바일 언어.

김 정숙

1999 동국대학교 대학원 컴퓨터 공학과(공학박사)

2001-현재 삼육의명대학 인터넷과 조교수

관심분야 : 컴파일러, 프로그래밍 언어, XML, 멀티미디어 언어.