Call grape

박준형

2019 05 23

1 콜 그래프의 개념과 야크에서 구현한 방법에 대한 설명 (동작 방식)

콜 그래프란 프로그램의 프로 시저 간의 호출 관계를 나타내는 유향 그래프를 뜻한다. 콜 그래프는 인간이 프로그램을 이해하기 쉽도록 도와주는데 어떤 프로그램에 대한 디자인을 파악하여 프로그램의 성능을 향상시키거나 기능을 수정하는 등 분석하는 데에도 많은 도움을 준다. c언어로 된 프로그램을 렉스와 야크를 통해 코드를 파싱하고 코드에서 어떤 함수가 다른 함수를 부르면 부르는 횟수와 몇번째 줄에서 불리는 지의 정보와 함께 그래프로 나타낸다. 그래프로 나타내기 위해 여러과정이 필요하다. 우선 렉스를 통해 토큰화되는 코드들을 야크에서 자료구조에 저장할 수 있도록 야크의 정의절에서 자료구조와 카운트하기 위한 선언, 함수이름을 담기위한 char형 문자열 등을 선언한다. 또한 렉스에서 토큰화하는 과정에서 name과 line에 대한 정보를 extern을 통해 야크에서 초기화한 name과 line에서 사용할 수 있도록 하였다. 모든 선언 후에 yylex()가 호출되면 렉스를 통해 정규표현식으로 코드를 토큰화한다. 토큰화된 코드들을 야크의 규칙절에서 파싱트리로 reduce과정을 통해 함수의 이름이 선언 또는 호출 될 때마다 호출한 함수의 이름과 호출된 함수의 이름이 정의절에서 선언한 링크드리스트(자료구조)에 구조체로 차곡차곡 연결된다. 서브루틴절에서는 3개의 함수를 추가하였는데, 자료구조 리스트에 저장된 노드들을 정리하고 콜 그래프를 만들도록 도와주는 기능을 하는 함수들이다. 이 함수들에 대한 설명은 밑에서 자세히 다루겠다.

2 콜 그래프 구현을 위해 기존 야크 코드 중 추가한 부분의 코드와 설명

기존 과제에서 syntex error가 2, 5번케이스에서 발생하였는데, 이유는 함수가 자료형없이 선언구현될 때에 대한 야크의 코드가 없었기 때문이다. 따라서 function definition에서 declaration specifiers가 빠진 코드를 2줄 추가하였다.

497번 498번 라인 추가

```
494 function_definition
495     : declaration_specifiers declarator declaration_list compound_statement
496     | declaration_specifiers declarator compound_statement
497     | declarator declaration_list compound_statement
498     | declarator compound_statement
499     ;
```

렉스와 야크에서 파싱하는 동안 코드를 거치며 function부분 즉, 함수들의 이름을 자료구조 리스트에 적절히 연결 저장되도록 구현해야하는데, 크게 선언부(declarator)와 표현부(expression)으로 나눌 수 있다. external declaration이 함수의 한 단위라고 생각한다면(예를 들어 main()) function definition을 통해 declarator로 먼저 들어가는데 direct declarator까지 들어가 IDENTIFIER을 만나면 렉스에서 name으로 저장한 문자열을 strcpy를 통해 decl 문자열에 복사한다. 이때, 추가로 direct declarator를 쓸데없이 거쳐 decl에 복사된 문자가 바뀌는 것을 방지하기 위해 c라는 변수를 통해 처음으로 direct declarator로 들어와 IDENTIFIER을 거친 토큰만 저장하도록 하였다.

선언부: 호출하는 함수(caller)를 decl에 복사
c가 0이면 decl에 복사 후 쓰레기값이 복사되지 않도록 c를 1로

```
328 direct_declarator
329         : IDENTIFIER {
330                 if(c==0) strcpy(decl, name);
331                 c = 1;
332         }
```

compount statement에서는 함수가 구현될 때 중괄호 안에 block item list를 통해 여러 statment와 declaration들을 호출하거나 선언할 수 있는데, 이때 호출된 함수(callee)들의 이름들을 선언부에서 저장한 decl과 함께 add함수를 통해 노드를 구성하고 리스트(자료구조)에 추가하는 기능을 하게 된다. 이 역시 쓰레기 값들이 callee로써 노드에 추가되는 것을 방지하기 위해 primary expresstion에 IDENTIFIER를 거칠 때 e가 0일 경우 funct에 변수명을 복사하고 e를 1로 설정한다. 이 때 가장 중요한 곳이 있는데 바로 postfix expression이다. 함수가 아닌 변수들은 funct에 저장되더라도 노드로 자료구조에 담지 않고 오직 괄호() 가 나오는 경우에만 add함수를 통해 이전에 복사했던 decl(caller)와 funct(callee)를 노드에 저장하고 리스트func에 추가한다. 이때 다시 e를 0으로 초기화하여 다른 expression에서 함수가 호출될 때 자료구조에 담길 수 있도록 한다.

표현부: 호출되는 함수(callee)를 funct에 복사
e가 0이면 funct에 복사 후 쓰레기 값이 복사되지 않도록 e를 1로

```
74 primary_expression
```

```
75          : IDENTIFIER      {
76                                                  if(e==0)strcpy(funct, name); e=1;
77                            }
```

add함수를 통해 decl(caller)와 funct(callee)를 func리스트에 추가
다음 expression을 통해 리스트에 연결시키기 위해 e를 0으로 초기화

```
83 postfix_expression
84          : primary_expression
85          | postfix_expression '[' expression ']'
86          | postfix_expression '(' ')'
            {add(decl, funct, func); e=0;}
87          | postfix_expression '(' argument_expression_list ')'
            {add(decl, funct, func); e=0;}
88          | postfix_expression '.' IDENTIFIER
89          | postfix_expression PTR_OP IDENTIFIER
90          | postfix_expression INC_OP
91          | postfix_expression DEC_OP
92          | '(' type_name ')' '{' initializer_list '}'
93          | '(' type_name ')' '{' initializer_list ',' '}'
94          ;
```

이 이외에도 쓰레기 값이 decl와 funct를 변경하지 않도록 도와주는 c와 e이 여러번의 expression 또는 statement 그리고 external내에서 function과 declaration에서 c와 e를 초기화 시켜주는 부분이 필요하다.

expression이 여러번 나오는 경우에도 각 expression에서 함수들을 모두 자료구조에 담을 수 있도록 e=0으로 초기화

```
206 expression
207         : assignment_expression {e=0;}
208         | expression ',' assignment_expression {e=0;}
209         ;
```

for(i=함수();;)의 경우에도 함수를 자료구조에 담을 수 있도록 e=0으로 초기화

```
192 assignment_operator
193         : '=' {e=0;}
194         | MUL_ASSIGN {e=0;}
195         | DIV_ASSIGN {e=0;}
196         | MOD_ASSIGN {e=0;}
197         | ADD_ASSIGN {e=0;}
198         | SUB_ASSIGN {e=0;}
199         | LEFT_ASSIGN {e=0;}
200         | RIGHT_ASSIGN {e=0;}
201         | AND_ASSIGN {e=0;}
202         | XOR_ASSIGN {e=0;}
203         | OR_ASSIGN {e=0;}
204         ;
```

external declaration는 main과 같은 호출함수, 전역변수 또는 헤더로 갈라지는데 function과 declaration가 새로 바뀔때마다 c=0으로 초기화 하여 decl를 초기화 시키는 역할

```
488 external_declaration
489         : function_definition {c=0;}
490         | declaration {c=0;}
491         | include
492         ;
```

3 사용한 자료구조와 C 라이브러리 혹은 콜 그래프 출력방법에 대한 설명(reference 포함)

자료구조는 구조체 노드 뒤에 추가되는 노드가 있으면 연결시키는 링크드리스트를 구현하였다. 구조체 Node에는 호출함수 caller와 호출당하는 함수callee의 문자열을 담는 char형 변수들과 caller와 callee의 이름이 겹치는 노드의 갯수를 나타내는 count, 겹치는 노드들의 callee(호출되는 함수)가 몇번째 줄에서 호출되는지를 표현하는 line을 배열로 구성하였다. count와 line은 int가 아닌 short로 자료형을 선택한 이유는 short형으로도 충분히 모든 함수의 갯수와 몇번째 줄이든 표현 할 수 있기 때문이다. 온전히 테스트케이스 10000줄에서 10000개를 초과하는 함수가 없기에 노드도 10000개이상 생기지 않는다고 볼 수 있으며 함수가 10000번째 줄에 호출되더라도 short는 약6만 이상 표현할 수 있기에 충분하다 생각하였고 구조체 Node의 크기를 줄이기 위해 사용하였다. next는 다음 노드로 연결하는 포인터이다. 구조체 list는 모든 노드의 시작을 가리키는 구조체로 자료구조에서 노드를 꺼내거나 담거나 콜그래프를 만들 때 편하게 코딩할 수 있도록 만들었다. // makelist()함수는 구조체 list를 만드는 생성자와 비슷한 역할을 한다. // add()함수는 노드를 (자료구조)리스트에 추가하는 기능을 하는 가장 중요한 함수이다. 바로 caller와 callee에 대한 인자를 받아와 createnode()함수를 통해 노드를 만들어 리스트에 추가하는 기능을 한다. 이렇게 만들어진 자료구조가 lex와 yacc를 통해 리스트에 노드들이 연결되어 완성되면 위에서 살짝 다루었던 3개의 함수를 사용해 main에서 콜그래프를 만든다. //

equalsearch() 함수는 파싱되어 노드들이 연결된 리스트를 순회하여 겹치는 노드들을 합치는 기능을 한다. // selection()함수는 문자열과 리스트를 인자로 넘기면 리스트를 순회하며 호출함수(caller)와 인자로 넘긴 문자열이 같은 노드를 발견하면 그 노드를 리스트에서 제거하고 노드를 반환하도록 하였다.//

마지막 printNode()를 설명하기 앞서 콜 그래프 출력방법에 대한 설명을 하겠다. 먼저 콜 그래프는 graphviz의 방식을 선택하였다. file pointer에서 fopen을 통해 dot파일을 생성하고 w으로 dot파일에 입력할 수 있게 한다. dot파일에 노드와 노드를 연결 시키는 문법이 존재하는데, https://narusas.github.io/2019/01/25/Graphviz.html 를 참고하였다.

밑에 코드는 graphviz의 일부분이다.

```
digraph B511074 {
node[shape = ellipse] 노드의 모양을 타원으로
    main -> func[label= " 2 times 15,16"]
    func -> a[label= " 2 times 6,8"]
}
```

이렇게 caller -> callee 를 나타내고 label로 방향 선이 count와 line의 정보와 함께 나타날 수 있도록 하면 dot -Tjpg B511074.dot -o B511074.jpg 의 명령어로 jpg파일로 그래프를 출력해준다.// 과정을 다시 설명하자면, 렉스와 야크를 통해 파싱 후 생긴 자료구조를 메인에서 적절히 노드를 빼서 graphviz 문법에 맞게 digraph.dot 파일에 출력하도록 한 뒤, 이 dot파일을 그래프를 출력하는 jpg파일로 변환하면 되는 것이다.// 다시 printNode()함수 설명으로 돌아가면, 노드를 인자로 받아 그 노드의 caller와 callee, count, line의 정보를 dot파일에 입력한다. callee에 대한 정보를 second문자열에 복사해놓고 노드가 차지하고 있는 데이터를 free함수를 통해

해제한다. callee이 caller가 되는 함수들을 while문으로 뽑고 printNode()함수를 재귀하여 리스트에 main으로부터 호출할 수 있는 모든 함수들을 줄줄히 dot파일에 입력한다. dot파일로 출력을 끝내고 파일을 닫고 바로 dot파일로 jpg파일으로 변환 시키도록 system(dot -Tjpg B511074.dot -o B511074.jpg)함수를 추가하였다.// 마지막으로 밑에 사진은 임의로 만든 테스트케이스에 대한 콜 그래프이다.

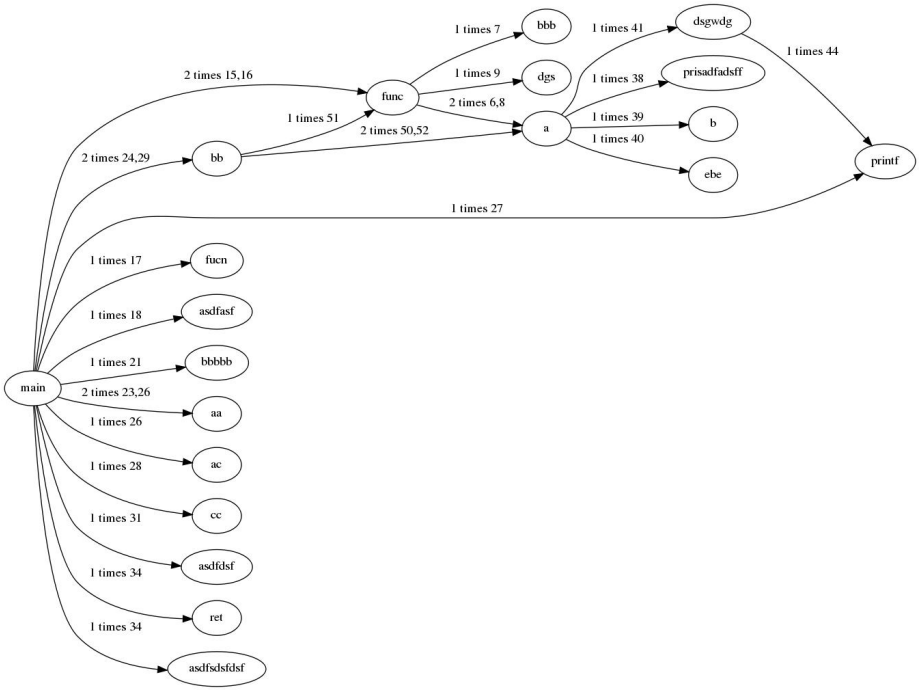

Figure 1: 임의로 만든 예제에 대한 콜 그래프