타입 한정자를 이용한 메모리 해제 오류 검출 기법

김연어○ 주성용* 조장우
동아대학교 컴퓨터공학과
yeoneo@donga.ac.kr, SeongYongJ@gmail.com, jwjo@dau.ac.kr

A Method Detecting Memory Free Errors Using Type Qualifiers

Yeoneo Kim○ Seongyong Joo* Jang-wu Jo
Dept. of Computer Engineering, The Dong-A University

1.서 론

메모리 해제 오류는 C나 C++에서 발생 가능한 오류로서 할당되지 않았거나 이미 해제된 메모리를 다시 해제하려고 시도하는 오류이다. 메모리 해제 오류의 유형으로는 할당되지 않은 메모리 해제, 이미 해제된 메모리에 대한 해제, 함수의 인수로 전달된 포인터에 의한 메모리 중복 해제 그리고 전역변수가 가리키는 메모리 중복 해제 등이 있다. 메모리 해제 오류는 이미 많이 알려져 있는 문제이며 지속적으로 연구되고 있다[1,2]. 또한 메모리 해제 오류는 2009년 SANS에서 발표한 프로그램에서 범하기 쉬운 오류 25가지에도 포함 되었다[3].

메모리 해제 오류는 프로그램 실행 중 예기치 않은 오류를 발생 시킬 수 있다. 그리고 이 같은 동적 메모리와 관련된 오류는 실행시간 오류이기 때문에, 기존 컴파일러는 이를 적절하게 다루지 못한다. 그러므로 안전한 소프트웨어 개발을 위해서 메모리 해제 오류를 검출하기 위한 도구가 요구된다.

본 논문에서는 메모리 해제 오류를 검출하기 위해서 타입 한정자를 이용하는 기법을 제안한다. 타입 한정자는 타입으로 설명하기 힘든 성질을 설명하는 가벼운 주석이다[4]. 타입 한정자는 C언어의 const, register, volatile 과 같이 기존 프로그래밍 언어들에서 사용되고 있기 때문에 친숙하며, 타입과 함께 검사될 수 있는 장점이 있다.

2.본 론

본 논문에서는 메모리 해제 오류를 검출하기위해서 타입 한정자를 이용하는 방법을 제안한다. 제안 기법은 메모리의 할당과 해제와 관련된 함수 선언에 타입 한정자를 부가하고, 이를 기반으로 각 식의 타입 한정자를 추론하기 위한 제약식을 생성한다. 그리고 생성된 제약식이 만족되지 않으면 메모리 해제 오류로 보고하는 방법이다. 제안 기법에서는 메모리 해제 오류를 검출하기 위해서 메모리 상태를 할당된 상태와 할당되지 않은 상태로 구분하며, 이를 표현하기 위해서 타입 한정자 ALLOC과 DEALLOC을 정의한다. ALLOC은 현재 메모리가 할당된 상태를 표현하며, DEALLOC은 메모리가 할당되지 않았거나 해제된 상태를 표현한다.

C 프로그램에서 메모리 할당과 해제는 라이브러리 함수를 통해서 수행되기 때문에 메모리 할당이나 해제와 관련된 라이브러리 함수 사전 선언에서 그 함수의 인수와 반환 타입에 타입 한정자를 추가한다. <그림 1>은 타입 한정자를 함수 선언에 추가한 예이다. malloc()은 메모리를 힙에 할당한 후 힙의 주소를 반환하기 때문에 함수의 반환 타입에 타입 한정자 ALLOC을 부가한다. 그리고 free()는 인수로 전달된 포인터가 가리키는 메모리를 해제한다. 그러므로 free()로 전달되는 인수는 항상 ALLOC이어야 하며, free를 수행하고 난 뒤 인수는 DEALLOC이어야 한다. 그러나 기존의 타입 한정자는 값의 현재 상태만을 표현하기 때문에 이 같은 성질을 표현하지 못한다. 그러므로 이 같은 성질을 표현하기위해 새로운 타입 한정자 매크로 ALLOCTODEALLOC과 DEALLOCTOALLOC을 정의한다. 타입 한정자 매크로 ALLOCTODEALLOC은 제약식에서 좌변식이 ALLOC이어야 하며, 이를 만족하는 경우 좌변식의 속성을 DEALLOC으로 변환함을 의미한다. 마찬가지로 DEALLOCTOALLOC은 제약식의 좌변의 속성이 DEALLOC이어야 하며, 이를 만족하는 경우 좌변식의 속성을 ALLOC으로 변환함을 의미한다.

```
DEALLOCTOALLOC  void *malloc( size_t size )
void free(ALLOCTODEALLOC void* ptr)
```

<그림 1> 타입 한정자를 추가한 함수 선언

제안한 기법을 대상 프로그램에 적용하기 위해서 malloc()이나 free()를 내부적으로 호출하는 사용자 정의함수는 사용자가 직접 반환 타입과 인수의 선언에 적절한 타입 한정자를 부가해야 한다. 그리고 본 논문에서는 메모리가 할당되지 않은 변수의 기본 속성을 DEALLOC으로 정의한다.

<그림 2>는 타입 한정자를 부가한 소스코드와 제약식 생성의 예이다. 1행의 제약식에서 malloc_q는 malloc()의 반환값에 대한 타입 한정자 변수이며, 2행과 3행의 free_arg1_q와 remove_arg1_q는 각각 그 함수의 첫 번째 인수에 대한 타입 한정자 변수이다. buf_q는 int* 변수 buf에 대한 타입 한정자 변수이다. 그리고 3행의 remove는 함수 내부에서 free() 함수를 호출해서 첫 번째 인수로 전달된 포인터가 가리키는 메모리를 해제하는 함수이다. 5행에서 buf는 초기화 되지 않았기 때문에 buf_q의 타입 한정자는 DEALLOC이 된다. 6행의 제약식에서 buf_q의 타입 한정자는 malloc_q의 값과 일치해야 한다. 그렇지만 malloc_q의 값이 DEALLOCTOALLOC이기 때문에 buf_q의 값은 DEALLOC이어야하며, 비교 후 buf_q의 값은 ALLOC으로 변환된다. 마찬가지로 8행에서 buf_q의 값은 ALLOC이어야 하며, 비교 후 buf_q의 값은 DEALLOC으로 변환된다. 9행에서는 buf_q의 값이 ALLOC이어야 하나, 현재 buf_q의 값은 DEALLOC이기 때문에 오류가 발생한다. 이것은 메모리 해제 오류가 발생했음을 의미한다.

타입 한정자를 부가한 소스코드		제약식
1:	DEALLOCTOALLOC void * malloc(int size);	malloc_q := DEALLOCTOALLOC
2:	void free(ALLOCTODEALLOC void *pt);	free_arg1_q := ALLOCTODEALLOC
3:	void remove(ALLOCTODEALLOC int *pt);	remove_arg1_q := ALLOCTODEALLOC
4:	...	
5:	int *buf;	buf_q := DEALLOC
6:	buf = malloc(n * sizeof(int));	buf_q = malloc_q
7:	...	
8:	remove(buf);	buf_q = remove_arg1_q
9:	free(buf);	buf_q = free_arg1_q
10:	...	

<그림 2> 타입 한정자를 부가한 소스코드와 제약식 생성 예

3.결 론

본 논문에서는 타입 한정자를 이용한 메모리 해제 오류 검출 기법을 제안하였다. 타입 한정자를 이용한 메모리 해제오류 검출 기법은 malloc()과 free()를 포함하는 사용자 정의 함수에 대해서만 타입 한정자를 부가하면 되기 때문에 사용자는 적은 노력으로 메모리 해제 오류를 검출 할 수 있으며, 대상 프로그램이 이진 파일 형태로 제공되는 라이브러리를 사용한 경우도 라이브러리의 함수의 원형에 타입 한정자를 부가하면 메모리 오류 검사가 가능하다.

참고문헌

[1] Nurit Dor, Detecting Memory Errors via Static Pointer Analysis, M.Sc. Thesis, School of Computer Science The Raymon and Beverly Sackler Faculty of Exact Sciences Tel-Aviv University, May, 1999.

[2] D. Dhurjati, S. Kowshik, and V. Adve, SAFECode: Enforcing alias analysis for weakly typed languages, In ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 2006.

[3] SANS TOP 25 Programming Errors, http://www.sans.org/top25-programmingerrors/.

[4] Jeffrey Scott Foster, Type Qualifiers Lightweight Specifications to Improve Software Quality. Ph.D. thesis, University of California, Berkeley, 2002.