요약해석을 이용한 버퍼오버런 분석에서 루프 분석결과의 정교화

(Refinement for Loops in Buffer-Overrun Abstract Interpretation)

오 학 주 † 이 광 근 ††
(Hakjoo Oh) (Kwangkeun Yi)

요 약 버퍼오버런 분석기가 루프안에서 발생시키는 허위경보를 간편하고도 효율적으로 줄이는 방법과 경험을 소개한다. 버퍼오버런 분석기는 루프와 배열을 많이 사용하는 프로그램을 분석할 때 많은 허위경보를 발생시킨다. 우리는 먼저 루프를 많이 사용하는 프로그램인 임베디드 프로그램과 암호화 관련 프로그램들에서 발생하는 허위경보를 조사하여 허위경보를 일으키는 루프의 패턴을 조사했다. 그 다음에 그 루프에 특화된 간단하고 효율적인 재분석기를 고안하였다. 우리가 제안하는 재분석기는 분석기가 분석을 끝낸 후 내 놓는 분석결과를 보고 재분석할 목표가 되는 루프만을 찾아서 초벌분석보다 더 정교한 분석을 하여 허위경보를 안전하게 제거한다. 버퍼오버런 분석기인 아이락에 구현하여 실험해본 결과 전체 루프 관련 허위경보 중 32% 가량이 제거되었다.

키워드 : 요약 해석, 허위경보, 정적분석, 정적 오류탐지

Abstract We present a simple and effective method to reduce loop-related false alarms raised by buffer-overrun static program analyzer. Interval domain buffer-overrun analyzer raise many false alarms in analyzing programs that frequently use loops and arrays. Firstly, we classified patterns of loop-related false alarms for loop-intensive programs, such as embedded programs or mathematical libraries. After that we designed a simple and effective false alarm refiner, specialized for the loop-related false alarms we classified. After the normal analysis of program in which alarms considered as false. We implemented this method on our buffer-overrun analyzer with the result that our refinement method decreased the number of false alarms by 32% of total amount the analyzer reported.

Key words : Abstract Interpretation, False alarms, Static analysis, static bug detector

1. 서 론

정적 프로그램 분석(static program analysis)이란 프로그램을 실행시키지 않고 프로그램이 실행중에 가질 성질을 안전하게(sound) 분석하는 것이다. 이 기술을 이용하면 프로그램의 오류를 컴파일 단계에서 모두 자동으로 찾을 수 있다. 이러한 프로그램 분석기의 한 예로 아이락[1]은 C프로그램에 대하여 실행중 발생가능한 버퍼오버런 오류를 실행전에 모두 찾아준다. 정적분석은 프로그램의 무한한 실제 실행성질을 유한하게 요약하기 때문에 필연적으로 허위경보를 가질수 밖에 없다. 즉 실제로는 오류가 아닌데 오류일 수 있다고 경보하는 경우가 있다. 작은 허위경보율(실제오류와 허위경보의 비율)은 분석기가 실제현장에서 사용되기 위해 필수적이다.

버퍼오버런 분석기도 많은 허위경보를 발생시키는데 특히 루프내에서 발생하는 경우가 많다. 루프는 특히 임베디드 프로그램에서 자주 쓰이는데, 때문에 이러한 프로그램을 분석할 때 분석기는 많은 허위경보를 루프내에서 발생하게 된다. 버퍼오버런 분석기인 아이락[1]으로 MPSoC내에 들어가는 상용 임베디드 프로그램을 분석한 결과 총 521개의 알람이 발생했는데 그 중 307개가 루프에서 발생한 것이었다.

이 논문에서 우리는 정적 분석기가 발생시키는 허위경보들 중 루프내에서 발생하는 허위경보를 효율적으로 줄이는 방법을 제시한다.

1.1 문제 정의

구간(interval)을 도메인으로 사용하는 버퍼오버런 분석기는 루프에서 많은 허위경보를 발생시킨다. 이러한 허위경보들은 대부분 루프의 시작지점에서 적용되는 넓히기(widening)[2] 연산 때문에 발생한다. 넓히기 연산은 루프의 분석이 종료함을 보장하기 위해서 꼭 필요한 연산이지만 루프내에서 변화하는 변수들의 값을 그 변수가 루프내에서 가질수 있는 모든 값을 포함하는 값으로 어림잡는다. 예를 들어 다음 C 프로그램을 보자.

· 이 연구는 교육인적자원부 두뇌한국 21사업의 지원을 받았음을 밝힙니다.
· 이 논문은 2007 한국컴퓨터종합학술대회에서 '요약해석을 이용한 버퍼오버런 분석에서 루프 분석결과의 정교화'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 서울대학교 컴퓨터공학과
pronto@ropas.snu.ac.kr
†† 정 회 원 : 서울대학교 컴퓨터공학과 교수
kwang@ropas.snu.ac.kr

논문접수 : 2007년 9월 28일
심사완료 : 2008년 1월 4일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제14권 제1호(2008.2)

```
char p[64];
char d[64];
...
char* pd = &p[0];
for (j = 0; j < 8; j++)
{
    d[0] = (pd[0]+4)>>3;
    d[1] = (pd[1]+4)>>3;
    ...
    d[7] = (pd[7]+4)>>3;
    d += 8;
    pd += 8;
}
```

구간 도메인을 사용하는 버퍼오버런 분석기(이하 분석기)는 위의 루프내의 모든 버퍼 접근에 대하여 경보를 발생한다. 분석기는 루프의 조건식으로부터 변수 j가 루프를 돌때마다 1씩 증가하고 최대 8번까지 변화한다는 것은 알수 있지만 변수 j와 d, j와 pd 사이의 관계(j가 1증가할때마다 d와 pd는 8씩 증가한다는 사실)는 유추할 수 없기 때문이다. 그래서 분석기는 포인터 변수 d와 pd가 루프 내에서 8번 변화한다는 사실을 알 수 없고, 무한번 변화할수 있다고 근사시킨다. 때문에 루프내에서 d와 pd가 가리키는 주소에 접근하는 모든 식들에 대하여 버퍼오버런 가능성을 보고하게 된다. 이로써 위의 예와 같이 분석기는 루프의 실행 횟수(iteration number)를 쉽게 파악할 수 있는 루프에서 조차도 많은 수의 허위경보가 발생한다. 우리의 목표는 이러한 루프에서 발생하는 허위경보를 안전하게 제거하는 것이다.

1.2 루프 관련 허위경보를 줄이는 기존 방법들

기존에 루프에서 발생하는 허위경보를 줄이기 위해 쓰인 기술들은 실제 프로그램 분석에 적용하기에 정확도와 성능면에서 무리가 있었다. 관계분석(relation analysis)[3]은 변수들 사이의 관계 정보를 유지함으로써 루프내에서의 분석 정확도를 높인다. 하지만 이 방법은 모든 변수들 사이의 관계를 추적하기 때문에 분석의 시간과 메모리 사용량이 크게 늘어난다는 단점이 있다. 루프 펼치기(static loop unrolling)[4]는 분석전에 프로그램에서 쓰이는 모든 루프들을 일정횟수 펼치는 방법이다. 이 방법 또한 분석할 프로그램의 크기가 커지기 때문에 분석비용이 크게 증가하여 분석기의 확장성(scalibility)를 떨어뜨린다. 또한 루프펼치기의 정확도는 사용자가 지정한 루프를 펼칠 횟수에 의존하기 때문에 사용자가 분석할 프로그램에서 사용되는 루프의 패턴이 어떠한지 잘 알고 있어야 한다는 부담이 있다. 넓히기 지연시키기(delayed widening)[2]는 루프 펼치기와 달리 프로그램 코드를 복사하지 않으므로 효과적인 방법이지만 루프내에 변수들간의 관계(relation)을 추적하지 못하므로 실제 프로그램에서 허위경보들을 효과적으로 제거하지 못한다.

1.3 우리의 방법, 루프 재분석

우리의 방법은 요약해석 정교화 방법의 한 예이다. 이 방법은 2장에서 자세히 설명된다. 아이디어는 간단하다: 기존의 분석기의 기존의 방법대로 구간분석을 수행한 다음, 알람이 많이 발생한 루프만을 찾아서 더 자세한 분석방법으로 재분석 하는 것이다. 즉, 기존의 효율적인 분석기를 이용하여 전체 프로그램을 한번 분석한 후 나온 분석결과를 보고 필요한 부분의 분석결과를 정교하게 다듬는다. 이렇게 하면 초벌 분석은 기존의 분석방법을 그대로 이용하므로 분석기의 확장성(scalibility)을 그대로 유지할 수 있다. 또한 분석정확도 향상이 필요한 프로그램의 부분들이 각각 가지는 특성에 따라 서로 다른 재분석방법을 적용할 수 있어서 효과적으로 허위경보를 제거할 수 있다. 예를 들면 관계분석이 필요한 코드조각에 대해서는 관계분석을 적용하고 넓히기 지연시키기가 적절한 곳에는 그것을 적용하는 식이다.

이 논문에서 우리는 실제 임베디드 프로그램에서 자주 쓰이는 루프 패턴에 대해서만 재분석을 적용한다. 임베디드 프로그램에서 발생하는 허위경보들을 분류한 결과 루프변수의 하한값(lower bound)와 상한값(upper bound)가 루프조건식에 명확지 쓰여져 있어서 루프의 도는 횟수를 유추 가능한 바운드룹(bounded loop)에서 많은 허위경보들이 발생한다는 사실을 알아냈다. 그리고 넓히기 지연시키기와 유사하게 루프의 도는 횟수만큼만 넓히기 없이 분석하면 효율적으로 허위경보를 제거할 수 있다고 판단하였다. 이와 같은 방법을 구현하여 MPSoC 상용 임베디드 프로그램에 대하여 적용해본 결과 총 307개의 루프내에서 발생한 알람중 99개를 제거할 수 있었고 재분석 때문에 17%의 분석시간이 증가했고 메모리 사용량은 변화가 없었다.

2. 루프에 대한 요약해석 정교화(Abstract Interpretation Refinement for Loops)

구간도메인을 이용하는 퍼오버런 분석기는 많은 상용 프로그램에 대하여 유용함이 밝혀졌지만[5] 루프내에서 버퍼에 접근하는 방식이 많은 프로그램에 대해서는 많은 수의 허위경보를 발생시킨다. 일반적으로 이러한 정확도 문제를 해결하는 방법은 더 자세한 방법으로 분석하는 것일 것이다. 하지만 더 자세한 분석방법은 분석비용을 증가시켜서 분석기의 확장성을 떨어뜨린다. 또한 일관되게 자세한 분석을 모든 프로그램의 분석에 적용하는 것은 자세한 분석이 필요없는 특정 프로그램에 대해서도 동일하게 적용되어 분석비용을 증가시키므로 오히려 비효율적일 수 있다.

우리의 목표는 분석기의 전체적인 성능을 크게 떨어뜨리지 않으면서 실제 프로그램에서 자주 나타나는 유형의 허위경보들을 효과적으로 제거하는 것이다. 우리가

제안하는 방법은 요약해석 분석결과를 정교화(abstract interpretation refinement)하는 방법이다. 이는 기존의 가벼운 분석방법으로 프로그램을 분석하여 얻은 분석결과에서 더 자세한 분석이 필요한 부분만을 선택한 후 그 부분을 더 자세하게 다시 분석하는 방법이다. 우리는 정확도 향상이 필요한 프로그램 부분을 기존의 것보다 정확한 요약 실행 함수(abstract semantics function)으로 재분석 하는 방법을 이용하였다.

요약해석 정교화를 실제 사용 프로그램 분석에 적용하려면 최대한 효율성을 높여야 한다. 이를 위해서는 다음 세 가지를 고려해야 한다. 첫째는 정교화를 통하여 정확도를 높이려는 대상이 명확해야 한다. 특정 정교화 방법이 모든 종류의 허위경보를 제거하는 데에 적합한 것은 아니기 때문이다. 우리는 허위경보를 일으키는 루프들 중 바운드 루프를 대상으로 잡았다. 둘째, 정교화를 위해 선택된 프로그램 부분들 중 재분석을 적용했을 때 실제로 정확도가 올라갈 것인지를 미리 예측해야 한다. 우리의 실험결과 정확도가 더 올라갈 여지가 도저히 없는데 다시 분석하는 것은 정교화의 성능을 많이 떨어뜨렸다. 이 부분에 대한 예측은 기존의 분석결과를 보고 어느정도 파악할 수 있다. 셋째, 대상에 적합한 효율적인 재분석 방법을 적용해야 한다. 보통, 정교화를 적용할 대상이 명확히 정해지면 효율적인 재분석 설계가 쉬워진다.

우리는 루프 관련 허위경보 제거에 요약해석 정교화 방법을 응용하였다. 일단 프로그램은 기존의 구간 분석을 이용하여 분석된다. 그 후에 분석된 결과를 보고 더 자세한 분석이 필요한 부분만을 판단하고 더 자세한 분석을 적용했을시 정확도가 올라갈 부분만을 다시 추려낸 후 재분석을 진행한다. 이 세가지 단계를 아래의 각 절에서 설명한다.

2.1 대상: 바운드 루프(Bounded Loop)

우리는 먼저 구간분석기인 아이락[1]으로 임베디드 프로그램들을 분석하여 허위경보들이 자주 발생하는 루프의 패턴을 조사했다. 그 결과 가장 많은 허위경보를 발생시킨 루프 패턴들은 다음과 같이 네 가지로 정리할 수 있었다.

Relation 루프내의 변수들의 값들 사이에 연관이 있는 경우이다.

```
int y[10];
int *yptr = &y[0];
for (i=0; i<6; i++) {
        *yptr = 1;
        yptr++;
}
```

Oscillation 변수가 루프를 돌면서 증가하지만 루프내의 조건문에서 그 값이 일정한 값이상 증가하지 않도록 막고 있는 경우이다. 구간연산의 한계상 정확한 계산을 하지 못하여 허위경보가 발생한다.

```
int x[10];
int i, s = 0;
for (i=0; i<128; i++) {
        x[s++] = 1;
        if (s >= 10) s-=10;
}
```

Stride 루프 변수가 루프를 돌때마다 1보다 큰 값으로 증가하는 경우이다. 루프의 종료시 인덱스 변수의 값을 정확히 계산하지 못하여 허위경보가 발생한다.

```
int x[256];
int i, data1, data2;
for (i=0; i<256; i+=2) {
        x[i] = data1;
        x[i+1] = data2;
}
```

Nested 실제 임베디드 프로그램을 조사한 결과 허위경보가 발생하는 다중루프의 종류는 매우 다양했다. 임베디드 프로그램의 분석결과 가장 안쪽루프의 영향을 받는 경우들을 네 가지와 가장 바깥쪽 루프의 영향을 받는 경우들을 세 가지로 분류할 수 있었다. 지면관계상 자세한 설명은 생략한다.

이러한 조사를 바탕으로 우리는 다음과 같이 C프로그래밍 언어의 for 키워드로 생성되며 루프 변수의 하한값과 상한값을 명확히 알 수 있는 루프의 분석결과만을 정교화해도 상당한 정확도 향상이 있을 것이라고 생각했다.

for (i=exp_1; i<exp_2; i+=c)
C;

현재 우리는 이와 같은 루프를 정교화 대상으로 하고 있고 효율성을 더욱 높이기 위해서 추후에는 위의 네가지 형태의 루프에 대해서만 재분석을 적용하도록 할 계획이다.

2.2 실제 정확도 향상을 예측하기

먼저 재분석을 시작하기 전에 실제로 정확도가 높아질 가능성이 높은 루프들만을 분류한다. 먼저 2.1에서 살펴본 바와 같이 우리는 실제 임베디드 프로그램에서 가장 많은 허위경보를 발생시킨 바운드룹에 대해서만 재분석을 적용한다. 그리고 루프내에서 발생한 경보가 허위경보일 가능성이 높을 경우에만 재분석을 적용한다. 현재 이 가능성은 세가지로 판별하고 있다. 먼저 루프내에서 발생한 경보의 버퍼 접근시 인덱스가 넓히기(widening)의 영향을 받았는지에 대한 여부를 살핀다. 발생한 버퍼오버런 경보의 인덱스가 상수(constant)라면 실제 오류일 가능성이 크기 때문이다. 둘째, 재분석 대상인 루프에 대한 입력 메모리 상태를 살핀다. 재분석을

통해 정확도를 향상할 대상인 값이 루프에 진입하기도 전에 이미 부정확한 값을 가지고 있었다면 그 값이 루프 분석 때문이 아닌 이미 이전에 부정확해진 것이기 때문이다. 셋째, 경보가 발생한 루프내에서 정확도를 높이고자 하는 변수가 전역변수이고 루프내에서 함수호출이 있는 경우에도 재분석을 하지 않는다. 왜냐하면 함수호출위치를 구별하지 않는(context insensitive) 분석기인 경우에는 전역변수가 함수호출시 흘러들어가 호출된 함수내의 루프에서 넓히기가 적용될 가능성이 크기 때문이다.

2.3 루프 재분석

재분석할 루프를 분류한 다음 기존의 분석결과에 기록되어 있는 루프의 시작지점의 입력 메모리를 시작으로 해당 루프에 대해서 추정한 도는 회수만큼 넓히기 없이 분석한다. 바운드 루프의 경우 루프의 조건식으로부터 루프가 몇번 돌지 예측할 수 있다. 즉 조건식이

for (i=c_1; i< c_2; 1+=c_3)

인 루프에 대해서 도는 횟수를

$$\lceil (c_2 - c_1) / c_3) \rceil$$

로 예측할 수 있다. 하지만 루프 내에서 루프변수 i의 값을 조작할 수 있으므로 이 값은 실제로 루프가 도는 횟수와 정확히 같지 않을 수 있다. 루프가 실제로 이 값보다 적게 도는 경우와 많이 도는 경우를 고려해야 한다. 예측한 값이 루프가 실제로 도는 횟수보다 더 클 경우를 대비하여 재분석을 통해 예측 한 횟수만큼 분석해 나가면서 매번 루프의 조건식이 성립하는지를 확인해야 한다. 성립하지 않을 경우 루프가 예측한 횟수보다 더 적게 돈다는 것을 의미하므로 그대로 재분석을 종료한다. 그리고 예측 한 만큼 재분석을 마치고 다시한번 루프의 조건식을 체크하여 성립하지 않는 지를 체크한다. 만약 재분석을 다 마친 상태에서도 루프의 조건식이 성립한다면 루프가 실제로는 예측한 횟수보다 더 많이 수행된다는 것을 의미하기 때문이다. 이 경우에는 넓히기 연산을 적용하여 루프를 다시한번 분석한 후 종료한다. 이 두 가지 경우 모두 분석의 안전성(soundness)을 보장한다. 이 방법은 루프를 재분석 할때에 추가적인 메모리를 사용하지 않는다. 루프 펼치기와 달리 루프를 펼치는 횟수만큼 루프의 분석상태를 저장하지 않고 루프를 한 횟수 돌면서 분석할때마다 버퍼오버런 경보의 발생 여부를 체크하고 발생한 경보들을 모은다.

다중루프를 효율적으로 재분석하기 위해서는 루프의 형태에 따라 가장 바깥쪽의 루프와 가장 안쪽의 루프중 어떤 루프를 재분석할지를 결정해야 한다. 우리는 2.1에서 언급된 다중 루프의 경우들에 대해서 가장바깥루프 재분석(outermost refiner)과 가장안쪽루프 재분석(innermost refiner)을 각각 적용한다. 다중 루프의 분석에서 어떤 분석방법을 적용할지 판단할 수 없는 경우에는 두 가지 분석방법을 차례로 적용하고 있다.

3. 실험 결과

우리는 2장에서 설명한 루프 재분석기를 기존의 버퍼오버런 분석기인 아이락[1]에 구현하였다. 아이락은 구간을 도메인으로 사용하며 함수호출을 구별하지 않는(context insensitive) 분석을 수행한다. 실험결과에서 아이락의 분석결과를 A로 나타내며 아이락 뒤에 루프 재분석기를 추가한 분석기를 AR로 나타내기로 한다. AR은 루프 재분석을 제외하면 A와 완전히 똑같다.

두 가지 종류의 프로그램에 대해서 실험하였다. 하나는 임베디드 프로그램이고 다른 하나는 암호화 관련 프로그램들이다. 임베디드 프로그램들로서 divx-player, openavs-1.0.2와 tmndec-3.2.0을 선택했다. 그리고 암호화 프로그램으로는 blowfish, vncdec와 ice를 선택했다.

모든 실험은 펜티엄4 3.2GH와 4GB 메인 메모리위의 리눅스 2.6 시스템위에서 수행하였다. Time은 파일입출력을 포함한 분석기의 전과정에 쓰인 CPU시간이며 #alarms는 알람의 개수이다.

3.1 루프 재분석의 성능

표 1, 2는 임베디드 프로그램과 암호화 프로그램에 대하여 A와 AR의 성능을 비교한 것이다. 임베디드 프로그램에 대하여 AR은 A에서 발생한 알람의 24%인 195개를 줄였다. 이에 따른 분석시간의 증가량은 33%였다. 유난히 분석시간이 크게 늘어난 두 파일 MADStream15.c와 vlc.c를 제외하면 분석시간은 521초에서 536초로 증가하여 3%의 증가량을 보였다.

표 1 임베디드 프로그램에 대한 루프 재분석 실험결과

Software		A		AR	
		time(s)	#alarms	time(s)	#alarms
divx player	H263FRDivxI3.c	260.92	43	271.96	34
	MADStreamI5.c	2750.15	478	3382.82	388
openavs -1.0.2	pred.c	61.82	32	62.36	16
	macroblock.c	15.00	20	15.29	8
	vlc.c	96.21	36	564.07	13
	decode.c	10.38	5	10.76	0
	stream.c	9.24	5	9.62	0
	global.c	22.24	6	22.78	0
tmndec -3.2.0	getblk.c	7.31	33	7.63	33
	getpic.c	117.77	74	118.69	74
	store.c	0.57	2	0.57	2
	display.c	1.51	1	1.78	0
	getvlc.c	6.65	17	6.62	17
	idctref.c	0.03	4	0.06	0
	recon.c	4.60	16	4.70	0
	ring_buf.c	0.06	17	0.06	17
	dither.c	0.16	2	0.30	0
	tmndec.c	2.87	5	2.87	2
Total		3367.49	796	4482.94	604

표 2 암호화 프로그램에 대한 루프 재분석 실험결과

	A		AR	
Program	time(s)	#alarms	time(s)	#alarms
blowfish.c	0.55	2	0.56	0
blowfish2.c	0.12	5	0.30	0
vncdec.c	0.68	32	0.81	18
ice.c	0.29	2	0.31	1
Total	1.64	43	1.98	19

이때 허위경보의 수는 28%가량 줄었다. 두 파일에서 유난히 많은 시간이 필요했던 이유는 프로그램 내의 거의 모든 루프에서 허위경보일 가능성이 있는 알람이 발생해서 거의 모든 루프를 재분석하였기 때문이다. getblk.c와 getpic.c와 같이 허위경보가 줄어들지 않은 프로그램들에 대해서는 분석시간 역시 거의 늘어나지 않음을 볼 수 있다. 암호화 프로그램에 대해서는 평균 적으로 전체 허위경보의 56%를 줄일 수 있었으며 이 때 분석시간은 21% 증가했다. 두 종류의 프로그램에 대해서, 전체적으로는 평균 25%가량의 허위경보를 제거할 수 있었으며, 분석시간은 32%가량 증가했다. 루프에서 발생한 알람들만을 고려한 결과를 살펴보면, divx player의 경우, 루프에서 발생한 알람 307개중 99개를 제거(32%)했고 분석시간 증가량은 17%였다.

3.2 루프펼치기, 넓히기 지연시키기와의 비교

표 3과 표 4는 루프 재분석 방법과 기존의 방법들인 루프펼치기와 넓히기 지연시키기와의 성능 비교이다.

루프 펼치기를 적용한 분석결과를 AL, 넓히기 지연시키기를 적용한 결과를 AW라고 하자. AL의 경우 프로그램 내의 모든 루프를 각각 10번과 20번 펼친 후 분석한 결과이다. 루프 펼치기의 경우 AR이 AL보다 더 많은 수의 허위경보를 제거했으며 분석시간 또한 더 짧게 걸렸음을 알 수 있다. 예를 들어 vncdec.c의 경우 AR은 약 20%의 분석시간의 증가와 함께 절반에 가까운 허위경보를 줄인 반면 AL은 알람을 하나도 줄이지 못한 채 분석시간만 1860% 증가했다. 넓히기 지연시키기는 AR과 비슷한 속도이거나 더 빠른 속도를 보였지만 허위경보를 만족할만큼 없애지 못했다.

표 3 루프 펼치기와의 비교

	A		AR		AL(ub=10)		AL(ub=20)	
PROGRAM	time(s)	#alarms	time(s)	#alarms	time(s)	#alarms	time(s)	#alarms
vncdec.c	0.68	32	0.81	18	15.1	32	204.98	32
blowfish.c	0.56	2	0.56	0	20.36	0	225.64	0
blowfish2.c	0.12	5	0.30	0	43.42	1	611.09	1
ice.c	0.29	2	0.31	1	28.83	2	277.41	2
tmndec.c	2.87	5	2.87	2	250.39	5	3119.03	5
recon.c	4.60	16	4.70	0	111.96	8	303.37	8
vlc.c	96.21	36	564.07	13	3732.54	35	out of memory	
getblk.c	7.31	33	7.63	33	out of memory		out of memory	

표 4 넓히기 지연시키기와의 비교

	A		AR		AW	
PROGRAM	time(s)	#alarm	time(s)	#alarm	time(s)	#alarm
macroblock.c	15.00	20	15.29	8	9.56	14
pred.c	61.82	32	62.36	16	61.66	8
vncdec.c	0.68	32	0.81	18	0.68	32
getpic.c	117.77	74	118.69	74	119.65	74
H263FRDivxl3.c	260.92	43	271.96	34	261.48	43

4. 결 론

우리는 구간 도메인의 정적 버퍼오버런 분석기에 루프 재분석 과정을 추가함으로써 루프 분석에 대한 정확도를 비교적 적은 비용으로 효과적으로 높였다. 재분석을 실제 프로그램 분석에 효율적으로 적용하려면 정확도 향상이 필요한 부분을 경험적으로 혹은 통계적으로 분류하는 작업이 필요하다. 실험결과 우리의 방법은 루프 펼치기와 넓히기 지연시키기에 비해 효과적이었다. 루프 펼치기가 펼치는 횟수에 따라 코드를 복사함으로써 분석에 필요한 시간과 메모리 사용량이 급격히 증가하는데 비해 우리 방법은 추가적인 메모리를 전혀 요구하지 않는다. 넓히기 지연시키기 또한 추가적인 메모리 소모가 없지만 우리 방법에 비해 허위경보를 많이 제거하지 못한다.

참 고 문 헌

[1] Jaeho Shin, Jaehwang Kim, Hakjoo Oh, Yikwon Hwang, and Kwangkeun Yi. Airac5: Array index range analyzer for c. http://ropas.snu.ac.kr/2005/airac5

[2] Patrick Cousot and Radhia Cousot. Abstract Interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In POPL, pages 238-252, January 1977.

[3] Mine, A. A new numerical abstract domain based on difference-bound matrices. In PADO II, vol. 2053 of LNCS, Springer-Verlag, pp. 155-172.

[4] Bruno Blanchet, Patrick Cousot, Radia Cousot, Jerome Feret, Laurent Mauborgne, Antoine Mine, David Monniaux, and Xavier Rival. A static analyzer for large safety-critical software. In PLDI '03, pages 196-207, New York, NY, USA, 2003. ACM Press.

[5] Yungbum Jung, Jaehwang Kim, Jaeho Shin, and Kwnagkeun Yi. Taming false alarms from a domain-unaware C analyzer by a Bayesian statistical post analysis. In SAS 2005, volume 3672 of Lecture Notes in Computer Science, pages 203-217. Springer 2005.