공학석사학위논문

이진 탐색을 이용하는 단계 고정점 계산 가속 기법을 통한 C 프로그램 정적 분석 정확도 개선

Improving the Precision of Static Analysis by Using Thresholded Widening with Binary Search in C Programs

2015 년 2 월

서울대학교 대학원

전기 · 컴퓨터 공학부

김 솔

이진 탐색을 이용하는 단계 고정점 계산 가속 기법을 통한 C 프로그램 정적 분석 정확도 개선

Improving the Precision of Static Analysis by Using Thresholded Widening with Binary Search in C Programs

지도교수 이 광 근

이 논문을 공학석사 학위논문으로 제출함

2014 년 12 월

서울대학교 대학원

전기 · 컴퓨터 공학부

김 솔

김솔의 석사학위논문을 인준함

2015 년 1 월

위 원 장	염헌영	(인)
부위원장	이광근	(인)
위 원	엄현상	(인)

요약

이 논문은 구간 도메인(interval domain)을 이용하는 정적 분석에서 보다 정교하고 효율적인 고정점 계산 가속 기법(widening)을 이용하여 분석 정확도를 개선하는 방법을 제시한다. 기본적인 고정점 계산 가속 기법(conventional widening)[1]은 반복문에서 나타나는 변수들을 과도하게 추정하여 계산하는데, 이는 허위 경보의 원인 중 하나이다. 이로 인해 발생하는 허위 경보들은 더 정교하게 계산하는 고정점 계산 가속 기법(widening) 이외의 다른 방법으로는 줄이기 어려우며, 이를 실험으로 보이고자 한다. 단계 고정점 계산 가속 기법(thresholded widening)[2]을 이용하면 기본적인 고정점 계산 가속 기법(conventional widening)으로 인해 발생하는 허위 경보들을 줄일 수 있다. 하지만 단계 고정점 계산 가속 기법(thresholded widening)은 추가 비용이 크며 이를 줄이기 위한 방법으로 이진 탐색을 적용하는 방법을 소개하고자 한다. 이 논문에서는 이진 탐색을 이용하는 단계 고정점 계산 가속 기법(thresholded widening using binary search) 및 분석 디자인을 소개하고, 이로 인해 허위경보가 줄어드는 것을 실험을 통해 보인다.

주요어: 요약해석, 정적분석, 프로그램 분석, 고정점 계산 가속 기법

학번: 2013-20767

목차

그림 목차

표 목차

제 1 장 서론

1.1 동기

이 논문은 구간 도메인(interval domain)을 이용하는 정적 분석에서 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 이용하여 분석 정확도를 개선하는 방법을 제시한다. 기본적인 고정점 계산 가속 기법(conventional widening)[1]은 정적 분석에서 허위 경보를 만들어 내는 주 요인 중 하나이다. 보다 정교한 기법인 단계 고정점 계산 가속 기법(thresholded widening)[2]이 있지만, 너무 큰 비용으로 인해 큰 규모의 프로그램에 사용할 수 없다. 단계 고정점 계산 가속 기법(thresholded widening)은 보다 정확한 분석 결과를 계산해 내기 위해 순차적으로 단계 값들을 탐색한다. 그 결과 단계 값이 많으면 많아질수록 추가 비용이 늘어나게 되며, 그 결과 큰 규모의 프로그램 분석 시간이 크게 늘어난다. 따라서 적은 추가 비용으로 보다 정확한 분석 결과를 계산하기 위해 단계 값을 탐색할 때 이진 탐색을 이용하는 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 제시하고자 한다.

1.2 기본적인 고정점 계산 가속 기법과 문제점

기본적인 고정점 계산 가속 기법(conventional widening)은 반복문을 분석할 때 끝남을 보장하기 위해 흔히 사용하는 기법이다[1]. 예를 들어 그림 1.1과 같이 프로그램만 보아서는 언제 끝나는지 알 수 없는 프로그램이 있다고 하자. 이 경우 끝나기를 기원하며 반복문을 무한히 분석하고 있을 수 없기 때문에, 이를 가속하여 계산하는 고정점 계산 가속 기법(widening)을 이용한다. 프로그램 분석을 위해 구간 도메인(interval domain)을 이용하여 그림 1.1의 프로그램을 분석한다고 하면 `i`의 값은 다음과 같이 계산한다. 우선 최초의 상태 $[0, 0]$을 이용하여 반복문을 한 단계 분석한다. 한 단계 분석한 결과 `i`는 $[0, 1]$이 되고, 두 결과 $[0, 0]$, $[0, 1]$을 살펴보아 불안정한 오

른쪽 범위 값을 가속하여 $[0, \infty)$와 같이 계산한다. 그 결과 값 `i`는 $[0, \infty)$로 계산되어 프로그램의 고정점(fixpoint)에 도달하게 되고, 언제 끝날지 모르는 상태의 반복문에 대해서도 분석을 끝마칠 수 있다.

```
1 int i = 0;
2 while (???)
3   i++;
```

그림 1.1: 고정점 계산 가속 기법(widening)이 필요한 이유와 예시

구간 도메인(interval domain)을 이용하여 버퍼 오버런(buffer overrun)을 검사하는 정적 분석에서 기본적인 고정점 계산 가속 기법(conventional widening)은 많은 허위 경보의 원인이다. 고정점 계산 가속 기법(widening)은 분석의 끝남을 보장하기 위해 꼭 사용해야만 하지만 과도하게 추정하여 계산하기 때문에 이로 인해 허위 경보가 발생한다. 구간 도메인(interval domain)에서 기본적으로 사용하는 고정점 계산 가속 기법(conventional widening)의 정의는 그림 1.2과 같다[1]. 구간 도메인(interval domain)의 왼쪽 범위와 오른쪽 범위에 대해 각각 작아지거나 커지는 상태에 놓인 값을 $-\infty$, ∞와 같이 가속하여 계산한다. 이는 반복문 분석을 빠르게 끝마칠 수 있는 장점이 있지만 과도한 추정으로 인해 정교함이 부족한 분석 결과를 만들어 낸다.

$$[a, b] \ \triangledown \ [c, d] \ = \begin{cases} [a, b] & if \ [c, d] \ = \ \bot \\ [c, d] & if \ [a, b] \ = \ \bot \\ [(c < a \ ? \ -\infty : a), (b < d \ ? \ \infty : b)] & o.w \end{cases}$$

그림 1.2: 구간 도메인(interval domain)에서의 기본적인 고정점 계산 가속 기법(conventional widening)의 정의[1]

그림 1.3의 예제들이 전형적인 기본적인 고정점 계산 가속 기법(conventional widening)으로 인해 허위 경보가 발생하는 예제이다. 그림 1.3a 예제의 경우 기본적인 고정점 계산 가속 기법(conventional widening)을 이용하면 i값을 $[0,0] \rightarrow [0,1] \overset{\triangledown}{\rightarrow} [0,+\infty) \rightarrow [21,+\infty)$ 순서로 계산한다. 또한 그림 1.3b 예제의 경우 i값을 $[0,0] \rightarrow [0,1] \overset{\triangledown}{\rightarrow} [0,+\infty)$ 순서로 계산한다. 즉, 기본적인 고정점 계산 가속 기법(conventional widening)의 과도한 추정으로 인해 실제로는 안전한 버퍼 접근에도 경보를 낸다.

기본적인 고정점 계산 가속 기법(conventional widening)으로 인해 발생하는 허위 경보들은 값들을 더 정교한 고정점 계산 가속 기법(widening)을 이용하지 않고서는 없애기 힘들다. 그림 1.3의 두 예제 대해 비용을 더 들여 정확도를 높이는 흐름에 민감한(flow-sensitive) 분석, 문맥에 민감한(context-sensitive) 분석을 이용해도 허위 경보는 줄어들지 않는다. 기본적인 고정점 계산 가속 기법(conventional widening)과 더 정교한 고정점 계산 기속 기법(widening)을 이용하는 두 분석에 모두 흐름에 민감한 분석(flow-sensitive)과 선택적으로 문맥에 민감한(selective context-sensitive)[3] 분석을 적용한 결과를 이용하여 이를 실험적으로 보이고자 한다. 위의 두 실험에서 줄어드는 허위 경보는 흐름에 민감한(flow-sensitive) 분석과 선택적으로 문맥에 민감한(selective context-sensitive) 분석을 적용해도 사라지지 않으며, 더 정교한 고정점 계산 가속 기법(widening)을 적용함으로 없어지는 것을 보일 수 있다.

1.3 단계 고정점 계산 가속 기법에서 이진 탐색 이용의 필요성

단계 고정점 계산 가속 기법(thresholded widening)은 반복문을 분석할 때 불안정한 값들에 대해 단계 값들을 순차적으로 거쳐가며 분석하는 기법이다. 기본적인 방법의 경우 불안정한 값을 $-\infty, \infty$와 같이 추정하지만, 단계 고정점 계산 가속 기법(thresholded widening)은 가지고 있는 단계 값들 만큼씩 추정해 나가는 기법이다. 예를 들어 단계 값이 $\{0,5,25,50\}$과 같이 있다면, 그림 1.1에 대해 i의 값을 분석할 때 $0,5,25,50$을 거쳐가며 분석한다. 단계적으로 추정해 가며 분석할 경우 더 정교한 분석 결과를 얻을 수 있는 기회가 생기게 된다. 예를 들어 그림 1.3의 두 예제

```
1 int i = 0, buf[30];
2 while (???)
3   if (i > 20)
4     break;
5   i++;
6 buf[i];   /* false alarm */
```

(a) 조건에 따라 반복문이 끝나고, 버퍼 접근이 안전해 지는 경우

```
1 int x, i = 0, buf[9] = {1,2,3,1,2,3,1,2,3};
2 while (x < 5)
3   i = buf[i];   /* false alarm */
4   x++;
5 buf[i];   /* false alarm */
```

(b) 반복문을 여러번 수행해도 버퍼에 접근하는 갑이 특정 범위 밖으로 나가지 않는 경우

그림 1.3: 기본적인 고정점 계산 가속 기법(conventional widening)을 이용할 때, 안전한 버퍼 접근에도 경보를 발생하는 경우들

에서 단계 값으로 $\{0, 5, 25, 50\}$을 사용하는 단계 고정점 계산 가속 기법(thresholded widening)을 이용하면 다음과 같은 결과를 얻을 수 있다. 그림 1.3a 예제의 경우는 i의 값이 $[0,0] \rightarrow [0,1] \xrightarrow{\nabla_t} [0,5] \rightarrow [0,6] \xrightarrow{\nabla_t} [0,25] \rightarrow [21,25]$의 순서로 계산되며 버퍼 접근이 안전하다는 것을 알 수 있다. 그림 1.3b 예제의 경우는 i의 값이 $[0,0] \rightarrow [0,1] \xrightarrow{\nabla_t} [0,5]$의 순서로 계산되며, 마찬가지로 버퍼 접근이 안전하다는 것을 알 수 있다. 이와같이 더 정교한 결과로 인해 허위 경보를 없앨 수 있다. 또한 여전히 경보가 발생하지만 정보가 더 정교해 지는 효과를 결과를 얻을 수 있다. 예를 들어 기본적인 고정점 계산 가속 기법(conventional widening)을 이용하면 경보의 결과가 [0, ∞)와 같이 나오지만, 단계 고정점 계산 가속 기법(thresholded widening)을 이용하면 [0, 100]과 같이 나오는 등 더 정확한 정보를 제공할 여지가 생긴다.

다만 단계 고정점 계산 가속 기법(thresholded widening)은 많은 추가 비용이 필요하기 때문에, 이진 탐색을 이용하여 추가 비용을 줄이고자 한다. 단계 고정점 계산 가속 기법(thresholded widening)은 기본적인 고정점 계산 가속 기법(conventional widening)에 비해 반복문을 더 여러번 분석을 하기 때문에 분석에 많은 추가 비용이 든다. 반복문의 복잡도가 높아질 수록, 단계 값들이 많을수록 추가 비용은 더욱 커진다. 이진 탐색을 이용하면 반복문을 분석하는 횟수를 이론적으로 단계 고정점 계산 가속 기법(thresholded widening)에 비해 로그 단위로 줄일 수 있다. 따라서 이진 탐색을 단계 고정점 계산 가속 기법(thresholded widening)에 적용하면 적은 추가 비용으로 보다 정교한 분석 결과를 얻을 수 있으며, 이를 보이고자 한다.

1.4 논문의 구성

이 논문은, 총 5장으로 구성되어있다. 2장에서는 구간 도메인(interval domain)에서 기본적인 고정점 계산 가속 기법(conventional widening)의 문제 상황과, 어떻게 개선할지를 설명한다. 3장에서는 개선책에 대한 자세한 구현과 실험 결과를 다룬다. 4장에서는 제시하는 개선책의 한계점과 이를 보완하기 위한 아이디어를 다루고, 5장에서 결론을 내린다.

제 2 장 이진 탐색 단계 고정점 계산 가속 기법

요약 해석[1, 4, 5]에 기반한 스파스 분석(sparse analysis) 프레임워크[6]를 이용하는 분석기에 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 적용하여 효율적으로 허위 경보를 줄일 수 있는 것을 보이고자 한다. 구간 도메인을 이용하여 버퍼 오버런(buffer overrun)을 검사하는 정적 분석기에서 기본적인 고정점 계산 가속 기법(conventional widening)을 이용할 경우 과도한 추정으로 인해 허위 경보가 발생할 수 있다. 이 장에서는 이와 같이 발생하는 허위 경보를 줄이기 위한 기법으로 단계 고정점 계산 가속 기법(thresholded widening)을 소개하고, 한계점을 알아본다. 또한 단계 고정점 계산 가속 기법(thresholded widening)에 이진 탐색을 적용하여 효율적으로 단계 값들을 탐색하는 기법을 소개하고자 한다.

2.1 단계 고정점 계산 가속 기법과 한계점

단계 고정점 계산 가속 기법(thresholded widening)은, 기본적인 방법과는 다르게 반복문을 분석할 때 단계 단계 확장해 나가며 분석을 시도하는 방법이다. 앞서 설명한 바와 같이 예제 1.1에서 기본적인 고정점 계산 가속 기법(conventional widening)은 `i`의 값을 $[0,0] \to [0,1] \xrightarrow{\triangledown} [0,\infty)$와 같이 분석을 한다. 반면 단계 고정점 계산 가속 기법(thresholded widening)은 어떤 단계값 `x`, `y` (단, 1 < x < x+1 < y)가 있다면, `i`의 값을 $[0,0] \to [0,1] \xrightarrow{\triangledown_t} [0,x] \to [0,x+1] \xrightarrow{\triangledown_t} [0,y] \to [0,y+1] \xrightarrow{\triangledown_t} [0,\infty)$와 같이 분석을 한다.

단계 고정점 계산 가속 기법(thresholded widening)은 기본적인 방식보다 반복문을 정교하게 분석하며, 그 정의는 그림 2.1과 같다[2]. 단계 고정점 계산 가속 기법(thresholded widening)에서는 준비된 정수 집합 t에 속해있는 값 중 가장 근접한 값부터 차례대로 추정해 나간다. 이로 인해 불안정한 값들을 $-\infty$ 또는 ∞로 추정하는 기본적인 고정점 계산 가속 기법(conventional widening)에 비해 정교한 분석이 가

능하다. 단, 그림 2.1의 정의에도 나와 있듯이 단계적으로 추정해 나가는 값은 구간 도메인(interval domain)의 오른쪽 영역에 한정한다. 그 이유는 구간 도메인(interval domain)을 이용하여 버퍼 오버런(buffer overrun)을 탐지할 경우 허위 경보는 오른쪽 구간 값이 과도하게 추정되어 발생하는 경우가 대부분이기 때문이다. 특히 고정점 계산 가속 기법(widening)으로 인해 발생하는 대부분의 허위 경보는 오른쪽 구간의 과도한 추정과 관계가 있다. 따라서 오른쪽 구간에 대해서만 단계 고정점 계산 가속 기법(thresholded widening)을 적용한다.

$$[a,b] \ \nabla_t \ [c,d] \ =$$

$$\begin{cases} [a, b] & if \ [c, d] = \bot \\ [c, d] & if \ [a, b] = \bot \\ [(c < a \ ? \ -\infty : a), (b < d \ ? \ \min(\{x \mid x > d \wedge x \in t\}) : b)] & o.w \end{cases}$$

그림 2.1: 구간 도메인(interval domain)에서의 단계 고정점 계산 가속 기법의 정의[2]

표 2.1는 기본적인 고정점 계산 가속 기법(conventional widening)과 단계 고정점 계산 가속 기법(thresholded widening) 사이의 정확도, 분석 시간 차이를 보여주고 있다. 또한 모든 실험은 흐름에 민감한 분석(flow-sensitive)과 선택적으로 문맥에 민감한 분석(selective context-sensitive)을 사용하였으며, 줄어든 허위 경보들은 단계 고정점 계산 가속 기법(thresholded widening)을 적용해야만 줄어드는 것임을 뜻하는 것을 알 수 있다. 표 2.1을 살펴보면, 단계 고정점 계산 가속 기법(thresholded widening)을 적용했을 때 평균적으로 5.71%정도의 경보가 줄어들고, 5.15%의 경보가 더욱 정교해 진다. 하지만 약 46.38%의 추가비용이 필요하다. 단계 값들은 반복문별로 모으고, 이를 다시 반복문의 변수들이 독립적으로 사용하도록 구성하였다. 반복문 별로 흐름에 민감하지 않은(flow-insensitive) 분석 결과값과, 반복문에 등장하는 상수값을 모아서 단계 값으로 이용한다. 이진 탐색을 이용하는 방법 역시도 같은

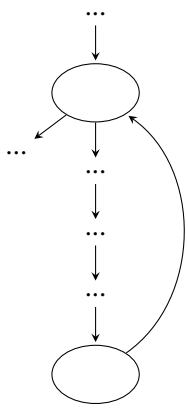
방법으로 단계 값을 모아서 사용하며, 단계 값을 모으는 것에 대한 자세한 내용은 3.1장에서 다룬다.

프로그램	라인	기본적인 고정점 계산 가속 기법		단계 고정점 계산 가속 기법		
		경보	시간 (초)	경보 ↓	정보 ↑	시간 (초) ↑
archimedes	7K	1273	15.71	168	34	3.85
gnuchess	11K	1294	46.23	62	60	39.27
bc	13K	555	144.32	8	3	133.25
tar	20K	954	78.96	13	0	159.23
make	27K	1847	224.28	28	35	432.28
grep	28K	936	26.88	8	65	21.71
wget	35K	760	118.47	33	52	88.60
a2ps	64K	2029	2073.74	261	263	416.41
fftw	184K	526	111.63	0	12	22.8

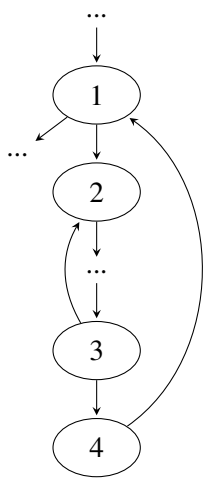
표 2.1: 단계 고정점 계산 가속 기법(thresholded widening)을 적용 한 결과. 경보는 평균 5.71%정도 줄었으며, 경보의 정보가 더 정교해 지는 경우는 평균 5.15%정도이다. 분석 시간은 평균 46.38%정도 더 걸린다.

반복문을 더 정확하게 계산하기 위해 반복문과 관련 있는 변수들을 단계 단계 거쳐가며 여러 차례 분석을 수행을 하기 때문에 많은 추가 비용이 발생한다. 특히 큰 규모의 프로그램 같은 경우 반복문의 규모도 크기 때문에 반복문을 추가로 수행하는 횟수가 늘어날 수록 추가 비용은 기하급수적으로 늘어난다. 정확도를 얻는 대신 비용을 더 들이는 것은 자연스럽지만, 얻을 수 있는 정확도에 비해 너무 많은 추가 비용이 든다. 그림 2.2는 추가 비용의 주 원인인 큰 규모의 반복문, 중첩된 반복문을 보여주고 있다. 우선 그림 2.2a와 같이 큰 규모의 반복문의 경우, 단계 고정점 계산 가속 기법(thresholded widening)을 이용하면 매 단계마다 반복문과 관련된 노

드를 매번 새로이 분석해야 한다. 반복문 내에 복잡한 함수 호출 등이 있을 경우 마찬가지로 새로이 분석해야 하며, 이로 인해 추가 비용이 발생한다. 그림 2.2b의 경우는 프로그램 노드 2∽3번의 반복문을 단계 고정점 계산 가속 기법(thresholded widening)을 이용해 분석을 수행하고, 노드 1∽4번의 반복문을 분석한 결과로 인해 다시 노드 2∽3번의 반복문 분석을 수행해야 하는 경우가 생길 수 있다. 이런 예 들로 인해 단계 고정점 계산 가속 기법(thresholded widening)을 이용 할 경우 많은 추가 비용이 발생한다.



(a) 아주 큰 규모의 반복문으로 인해 많은 추가 비용이 발생한다.



(b) 반목문이 중첩되어 여러번 분석을 다시 수행함으로 인해 많은 추가 비용이 발생한다.

그림 2.2: 큰 규모의 반복문이나, 중첩된 반복문으로 인해 단계 고정점 계산 가속 기법(thresholded widening)을 이용 할 경우 추가 비용이 많이 발생하게 된다.

2.2 이진 탐색을 이용하는 단계 고정점 계산 가속 기법

이진 탐색을 이용하는 단계 고정점 계산 가속 기법(thresholded widening using binary search)은 단계 값들을 순차적으로 탐색하지 않고 이진 탐색을 이용하여 탐색해 반복문의 고정점(fixpoint)을 빠르게 찾는 것이 목표이다. 단계 고정점 계산 가속

기법(thresholded widening)의 경우 단계 값이 $\{5, 10, 20, 30, 50, 80, 100, 120\}$와 같이 있다면, 최악의 경우 5부터 120까지 모든 단계 값을 거쳐가며 분석을 해야 한다. 같은 조건에 이진 탐색을 적용하면 $30, 80, 100, 120$ 네 번의 탐색만으로도 단계 값들의 탐색을 끝마칠 수 있다. 이진 탐색을 적용하면 단계 고정점 계산 가속 기법(thresholded widening)에 비해 로그 단위만큼만 단계 값을 탐색해도 분석을 끝마칠 수 있다. 즉 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 통해 많은 단계 값들을 빠르게 탐색하여 기본적인 고정점 계산 가속 기법(conventional widening)에 비해 정교한 분석 결과를 계산하는 것이 가능하게 된다.

이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)의 정의는 다음과 같으며, 구간 도메인(interval domain)을 이용하는 분석과 함께 디자인[7]할 수 있다.

프로그램 분석할 대상 프로그램을 흐름 그래프(control flow graph)로 표현할 수 있다.

$$(\mathbb{C}, \hookrightarrow)$$

$\mathbb{C}$ 는 노드들의 유한한 집합이며, $(\hookrightarrow) \subseteq \mathbb{C} \times \mathbb{C}$ 는 두 노드 사이의 흐름을 뜻한다.

구간 도메인(interval domain) 분석 프로그램의 고정점(fixpoint) 을 계산하는 요약 의미 함수(abstract semantic function) $\mathrm{F} \in (\mathbb{C} \to \mathbb{S}) \to (\mathbb{C} \to \mathbb{S})$ 는 다음과 같다.

$$\mathrm{F}(\mathrm{X}) = \lambda c \in \mathbb{C}.f_c(\bigsqcup_{c' \hookrightarrow c} \mathrm{X}(c'))$$

구간 도메인(interval domain)을 이용한 분석이므로 프로그램의 상태 $\mathbb{S}$ 는 다음과 같이 디자인 한다.

$$\begin{aligned} \mathbb{S} &\in \mathbb{L} \to \mathbb{I} \\ \mathbb{L} &\in Var \\ \mathbb{I} &\in \mathbb{Z} \times \mathbb{Z} \end{aligned}$$

이진 탐색 단계 고정점 계산 가속 기법의 정의 단계 고정점 계산 가속 기법(thresholded widening)과 이진 탐색을 위해 각각 $\mathrm{T} \in \mathbb{L} \to \mathcal{P}(\mathbb{Z})$, $\mathrm{B} \in \mathbb{L} \to \mathbb{Z} \times \mathbb{Z}$ 를 두고 이용한다. T는 각 변수에 대한 단계 값 정수 집합 테이블이며, B는 각 변수에 대한 이진 탐색이 진행중인 범위를 나타내는 테이블이다. T는 어떤 방식으로든 미리 준비된 단계 값들을 이용한다. B는 초기값으로 $\lambda l.(0, \mathrm{len}(\mathrm{T}(l)))$ 를 가지고 시작하며 왼쪽 값은 현재까지 찾은 가장 큰 불안정한 값을, 오른쪽 값은 가장 작은 안정한 값을 뜻한다. 예를 들어 어떤 l에 대해 (2, 6) 이라는 이진 탐색 상태가 있다면 $\mathrm{T}(l)[2]$ 는 찾은 값들 중 가장 큰 불안정한 값을, $\mathrm{T}(l)[6]$ 은 찾은 값들 중 가장 작은 안정한 값을 뜻한다. 이를 이용해 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search) 연산자를 다음과 같이 정의한다.

$$s_1 \ \triangledown_{\mathrm{T,B}} \ s_2 \ = \ \lambda l.s_1(l) \ \triangledown_{\mathrm{T}(l),\mathrm{B}(l)} \ s_2(l), \ \textit{where} \ s_1, \ s_2 \in \mathbb{S}$$

$$[a,b] \ \triangledown_{\mathrm{T}(l),\mathrm{B}(l)} \ [c,d] \ =$$

$$\begin{cases} [a, b] & \textit{if} \ [c, d] \ = \ \bot \\ [c, d] & \textit{if} \ [a, b] \ = \ \bot \\ [(c < a \ ? \ -\infty : a), (b < d \ ? \ \mathrm{T}(l)[\mathrm{mid}(\mathrm{B}(l))] : b)] & o.w \end{cases}$$

고정점 계산 알고리즘 앞서 디자인한 도메인과 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search) 연산자를 통해 프로그램의 고정점(fixpoint)을 구하는 알고리즘을 그림 2.3와 같이 디자인 한다.

```
1  W ∈ Worklist = 𝒫(ℂ)
2  X ∈ ℂ → 𝕊
3  f_c ∈ 𝕊 → 𝕊
4  /* Initialized finite set of loc to threshold set */
5  T ∈ 𝕃 → 𝒫(ℤ)
6  B := λl.(0, len(T(l)))
7  W := ℂ
8  X := λc.⊥
9  repeat
10   c := choose(W)
11   W := W - {c}
12   repeat
13     s_out := f_c(⨆_{c'↪c} (X(c')))
14     s_out := X_c ▽_{T,B} s_out
15     stables, unstables =
16       {l | l ∈ s_out ∧ s_out(l) ⊑ X_c(l)},
17       {l | l ∈ s_out ∧ s_out(l) ⋢ X_c(l)}
18     if {l|l ∈stables ∧ mid(B(l))!=B(l).l} != ∅
19       B{l↦(mid(B(l)), B(l).r) | l ∈unstables}
20       B{l↦(B(l).l, mid(B(l)) | l ∈stables}
21     else if {l | l ∈ unstables} != ∅
22       B{l ↦(mid(B(l)), B(l).r) | l ∈unstables}
23     else break
24   until break
25   if s_out ⋢ X_c
26     X_c := s_out
27     W := W ∪ {c' | c ↪ c'}
28 until W = ∅
```

그림 2.3: 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 이용하는 할 일만 하기 고정점 계산(worklist-based fixpoint computation algorithm)[7] 알고리즘

제 3 장 구현 및 실험

이 장에서는 이진 탐색을 이용한 단계 고정점 계산 가속 기법(thresholded widening using binary search)의 구체적인 구현에 대해 설명하고, 실험 결과를 소개한다. 분석기는 요약 해석에 기반하여[1, 4, 5] 스파스 분석(sparse analysis) 프레임워크[6]를 이용하는 Sparrow[8]를 기반으로 2.2장에서 소개한 디자인에 충실하게 구현하였다. 다만 앞서 자세히 소개하지 못했던 단계 값들을 모으는 방법 등 세부 사항을 소개하고자 한다. 또한 이진 탐색을 이용하는 단계 고정점 계산 가속 기법(thresholded widening using binary search)의 실험 결과를 소개한다.

3.1 단계 값들 모으기

이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)에서 사용할 값들을 선정하기 위해 흐름을 고려하지 않은(flow-insensitive) 예비 분석 결과를 이용한다. 스파스 분석(sparse analysis)을 수행하기 위해서는 데이터 사이의 의존관계가 필요하고, 이를 위해 비교적 싼 비용의 예비 분석을 수행한다[6]. 예비 분석의 결과로 각 변수별 분석 결과 값을 얻을 수 있다. 여기서 분석 결과들 중 오른쪽 구간의 정수들만 모아서 단계 값으로 사용한다. 이 때, 구간 도메인(interval domain)의 오른쪽 값만을 모으는 이유는, 구간 도메인(interval domain)에서 허위 경보는 주로 오른쪽 구간이 부정확하게 계산되는 경우가 많이 때문에 이와 관련된 값만을 모으기 위함이다. 그 결과로 하나의 반복문은 정수 집합을 하나씩 가지게 되며, 각 반복문의 변수들은 같은 단계 값을 가지며 이를 독립적으로 이용한다.

예를 들어 그림 3.1과 같은 프로그램의 예비 분석 결과가 표 3.1과 있을 경우, 단계 값은 다음과 같이 모은다. 우선 프로그램에 등장하는 반복문에 관련된 값들을 모두 모으며, 예제에서는 {a, b, c, d}이 된다. 프로그램의 예비 분석 결과를 살펴보아 이 값들의 오른쪽 범위 값 중 사용할 수 있는 값들을 모으며, 예제에서는 {7, 55,

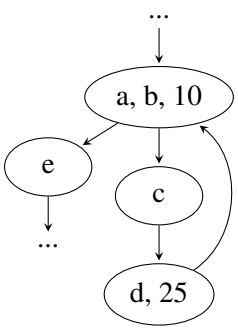


그림 3.1: 프로그램의 흐름 그래프(control flow graph)로, 노드 안의 알파벳은 변수를, 숫자는 프로그램에 등장하는 상수를 듯한다.

100}이 된다. 프로그램에 등장하는 상수값들을 모아서 앞서 모아둔 값들과 합쳐 단계 값으로 사용한다. 예제에서는 {7, 10, 25, 55, 100}이 된다. 이렇게 얻게 된 단계 값의 집합을 다시 반복문과 관련된 변수에 하나씩 할당을 한다. 즉, T(a) = T(b) = T(c) = T(d) = {7, 10, 25, 55, 100}와 같이 단계 값의 집합을 각 변수별로 독립적으로 이용하게 된다.

변수	결과
a	[-∞, 100]
b	[-10, ∞]
c	[0, 55]
d	[-∞, 7]
e	[16, 76]
...	...

표 3.1: 흐름 그래프인 그림 3.1의 예비 분석 결과 예시

예비 분석의 결과 값들을 이용하면 추가 비용 없이 값들을 모을 수 있다는 이점이 있다. 반복문과 관련된 변수를 모으는 일은 예비 분석 도중에 만들어 내는 변수 의존 관계 그래프(def-use graph)를 이용하여[6] 쉽게 수행할 수 있다. 또한 예비 분석

결과에서부터 해당 변수들이 분석 결과 어떤 값들로 계산되었는지 찾고, 프로그램의 상수 값을 모으는 일 역시도 큰 비용이 들지 않는다. 따라서 단계 값들을 모으기 위해 별도의 계산을 할 필요가 없기 때문에 추가 비용이 거의 없다는 장점이 있다.

3.2 실험 결과 및 분석

스파스 분석(sparse analysis) 프레임워크를 이용하는 Sparrow를 기반으로 이진 탐색을 이용한 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 구현해서 실험을 수행했다. 실험 결과는 표 3.2에서 알 수 있다. 이진 탐색을 이용하는 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 적용하여 허위 경보를 평균 4.24%만큼 줄었고, 약 5.28%의 경보는 정보의 질이 좋아졌다. 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)으로 인하여 발생한 추가 비용은 10.32%이다. 단계 고정점 계산 가속 기법(thresholded widening)의 46.38%에 비해 적은 추가 비용으로 비슷한 정확도 향상을 얻을 수 있음을 알 수 있다.

```
1 int i = 0, buf[15];
2 while (true)
3   if (i == 10)
4     break;
5   i++;
6 buf[i];    /* false alarm */
```

그림 3.2: 이진 탐색을 이용함으로 인해, 단계 고정점 계산 가속 기법(thresholded widening)보다 정확도가 나빠질 수 있음을 보여주는 예제.

이진 탐색 단계 고정점 계산 기법(thresholded widening using binary search)은 단계 고정점 계산 기법(thresholded widening) 보다 정확도가 나빠지는 성질이 있다.

이는 불안정한 값들 사이에 안정한 값이 존재할 수 있기 때문이다. 예를 들어 그림 3.2와 같은 예제에서 단계 값이 $\{0, 10, 25, 50, 150\}$과 같이 있다면, `i`값을 $[0,0] \rightarrow [0,1] \xrightarrow{\triangledown_{T,B}} [0,25] \rightarrow [0,26] \xrightarrow{\triangledown_{T,B}} [0,50] \rightarrow [0,51] \xrightarrow{\triangledown_{T,B}} [0,150]$ ... $[0,\infty)$로 계산해 나간다. 결국 6번째 줄의 버퍼 접근이 안전함에도 경보가 발생한다. 즉, 정확히 어떤 값을 거쳐가지 않으면 없애는게 불가능한 허위 경보들이 있으며, 이진 탐색을 이용하면 이를 놓치는 경우가 생길 수 있다. 이로 인해 단계 고정점 계산 가속 기법(thresholded widening)보다 정확도가 낮은 특징을 가지게 된다.

프로그램	라인	기본적인 고정점 계산 가속 기법		단계 고정점 계산 가속 기법			이진 탐색 단계 고정점 계산 가속 기법		
		경보	시간 (초)	경보 ↓	정보 ↑	시간 (초) ↑	경보 ↓	정보 ↑	시간 (초) ↑
archimedes	7K	1273	15.71	168	34	3.85	160	36	1.15
gnuchess	11K	1294	46.23	62	60	39.27	43	11	17.80
bc	13K	555	144.32	8	3	133.25	4	7	10.29
tar	20K	954	78.96	13	0	159.23	10	3	27.10
make	27K	1847	224.28	28	35	432.28	22	35	85.56
grep	28K	676	26.88	8	65	21.71	10	68	4.01
wget	35K	760	118.47	33	52	88.60	0	53	57.19
a2ps	64K	2029	2073.74	261	263	416.41	182	312	88.29
fftw	184K	526	111.63	0	12	22.8	0	12	6.72

표 3.2: 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 적용 한 결과. 경보는 평균 4.24%정도 줄었으며, 경보의 정보가 더 정교해지는 경우는 평균 5.28%정도이다. 분석 시간은 평균 10.32%정도 더 걸린다.

제 4 장 한계 및 보완사항

이진 탐색을 이용한 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 이용해서 성능과 정확도를 높이고자 할 때 한계점과 개선 방향에 대해 소개하고자 한다. 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)의 정확도 상승 정도는 순차적인 단계 고정점 계산 가속 기법(thresholded widening)의 정확도 상승 정도에 비해 작거나 같은 성질이 있다. 어떤 요인으로 인해 이와 같은 문제가 발생하는지를 소개하고 보완책을 다룬다. 정확도를 높일 수 있는 여지를 가진 단계 값을 모으는 방안들을 살펴보고자 한다. 또한 정확도는 유지한 채 추가 비용을 줄이기 위한 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 반복문에 선별적으로 적용하는 방안을 소개하고자 한다.

4.1 단순 단계 고정점 계산 가속 기법에 비해 이진 탐색 기법의 낮은 정확도

3.2장에서 소개한 바와 같이 이진 탐색을 이용함으로 인해 단계 고정점 계산 가속 기법(thresholded widening)에 비해 정확도가 낮아지는 경우가 생긴다. 이는 정확히 어떤 값을 거쳐가야만 허위 경보를 줄일 수 있는 경우들로 인해 발생하는 문제이다. 이를 해결하는 방법으로는 단계 값 집합을 나누어 단계 고정점 계산 가속 기법(thresholded widening)과, 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)를 혼합하여 분석을 수행하는 방법이 있다. 혼합하여 사용하면 이진 탐색 단계 고정점이 가진 정확도 손실을 막을 수 있고, 단계 고정점 계산 가속 기법이 야기하는 추가 비용을 줄일 수 있다. 이는 다음과 같이 정의할 수 있다.

$$a \nabla_{\text{hybrid}} b \;=\; a \nabla b \;\sqcap\; a \nabla_{t} b \;\sqcap\; a \nabla_{T,B} b$$

혼합하여 사용하는 고정점 계산 가속 기법(hybrid widening)의 목적은 표 3.2의 단계 고정점 계산 가속 기법(thresholded widening)과, 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)의 분석 결과의 중간에 해당하는 결과를 얻을 수 있도록 하는 것이다. 예를 들어 3.1장에서 소개한 단계 값을 모으는 방법을 이용하여 얻은 단계 값들을 단계 고정점 계산 가속 기법(thresholded widening)과 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)이 각각 절반씩 나누어 가진다고 하자. 단계 값을 위와 같이 절반씩 나누어서 혼합된 고정점 계산 가속 기법(hybrid widening)을 이용한다면, 표 3.2에서의 두 기법의 중간 결과인 28.35%의 추가 비용을 들여 4.975%의 허위 경보 감소와, 5.125%의 경보의 질이 좋아지는 결과를 얻을 수 있을 것이다.

4.2 이진 탐색 단계 고정점 계산 가속 기법 선별적 적용

이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 이용함으로 인해 더 정교해지는 분석 결과는 대체로 특정 반복문에 몰려있는 경향이 있다. 표 4.1는 여러 프로그램에 대해 전체 반복문 중 일부 반복문에만 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 적용한 실험 결과이다. 각 실험 결과는 소수의 반복문을 무작위로 뽑아서 분석 수행하고, 이를 여러번 반복하여 허위 경보가 가장 많이 사라지는 실험들을 골라 나열하였다. grep-2.5의 경우, 전체 반복문 723개 중 25개의 반복문에만 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 적용하여도 전체 반복문에 적용한 것과 같은 결과를 얻을 수 있다. 이외에도 archimedes-0.7, make-3.76.1, a2ps-4.14의 경우 전체 반복문 중 50개에만 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 적용해도 전체 반복문에 적용한 결과의 90%에 해당하는 정확도 향상을 얻을 수 있다. 이와 같은 관찰 결과로 미루어 볼 때 정확도 향상은 몇몇 반복문에서 집중적으로 이루어 지는 것을 알 수 있다.

이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 이용했을 때 효과를 볼 반복문에 대해서만 선별적으로 적용하면 추가 비용을 줄

프로그램	라인	고른 반복문 수 / 전체 반복문 수	줄어든 경보 / 줄일 수 있는 경보
archimedes	7K	25 / 1094	0 / 160
archimedes	7K	50 / 1094	157 / 160
make	27K	25 / 1737	18 / 22
make	27K	50 / 1737	18 / 22
grep	28K	25 / 723	10 / 10
grep	28K	50 / 723	10 / 10
a2ps	64K	25 / 3751	48 / 182
a2ps	64K	50 / 3751	137 / 182

표 4.1: 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 선별적으로 적용 한 결과.

일 수 있다. 특히 큰 규모의 프로그램의 경우 반복문이 중첩되어 있거나, 하나의 반복문이 아주 큰 규모를 이루고 있어 이진 탐색을 이용하더라도 많은 추가 비용이 발생한다. 다만 정확도 향상은 몇몇 소수의 반복문에 집중되어 있음을 실험으로 알게 되었으므로 이를 이용하여 정확도 향상과 관련있는 반복문에만 비용이 많이 드는 고정점 계산 가속 기법(widening)을 적용하는 방식을 생각해볼 수 있다. 즉 정확도 향상과 관련있는 몇몇 반복문만 잘 골라낼 수 있다면, 골라낸 반복문에만 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 적용하면 적은 추가 비용으로 같은 정확도를 얻을 수 있을 것이다.

4.3 정확도와 관련된 단계 값들을 모으기

정확도를 향상을 위해서는 어떤 값을 단계 값으로 사용하느냐가 중요하다. 앞서 보인 바와 같이 이진 탐색을 단계 고정점 계산 가속 기법(thresholded widening using binary search)에 적용하면 많은 단계 값들을 빠르게 탐색할 수 있다. 하지만 정확도 향상과 직접적인 연관이 있는 것은 어떤 값들을 단계 값으로 이용하느냐이다. 아무리 단계값이 많다고 하더라도 정확도 향상과 전혀 상관없는 값들로 이루어져 있다면 이진 탐색 단계 고정점 계산 가속 기법(thresholded widening using binary search)을 적용해 봐도 정확도 향상을 기대하긴 힘들다.

이 논문에서의 실험들은 경험적인 방법을 이용하여 단계 값들을 모았지만, 머신 러닝이나 정교한 추론 등을 이용하여 단계 값을 잘 선정한다면 더 정확한 분석 결과를 얻을 수 있다. 각 반복문별로 반복문과 관련있는 예비 분석의 결과와 상수 값을 모아서 단계 값으로 이용하는 방법은 경험적인 방법으로 충분히 더 좋은 단계 값 선정 방법들을 생각해볼 수 있다. 모종의 추론 과정을 통해 반복문의 특징과 변수의 특징 들을 살펴보고 정확도에 영향을 줄 것 같은 단계 값들을 고르는 방법이 있을 것이다. 또는 머신 러닝을 이용하여 학습한 정보를 토대로 프로그램의 특징을 살펴보고 정확도 향상과 관련있는 값들을 고르는 방법이 있을 것이다. 사용자에게 반복문의 종료 조건과 관련있는 상수를 입력 받아 단계 값으로 사용하는 방법 역시도 충분히 고려해볼 방법이다. 정확도 향상과 직접적으로 관련 있는 단계 값들을

모으는 방법을 찾아내어 이용하면 더 효율적이고 정확한 분석기를 만들 수 있다.

제 5 장 결론

이 논문은 정적 분석에서 이진 탐색을 이용한 단계적인 고정점 계산 가속 기법(thresholded widening using binary search)을 이용해 C 프로그램의 분석 결과를 개선하는 방법을 제시하였다. 기본적인 고정점 계산 가속 기법(conventional widening)을 이용할 경우 과도하게 추정하여 계산한 값들로 인해 허위 경보가 발생하는 것을 소개했다. 허위 경보를 줄이고, 경보의 질을 높일 수 있는 이진 탐색을 이용하는 단계적인 고정점 계산 가속 기법(thresholded widening using binary search)을 소개하고, 적용하는 방법을 설명했다. 이진 탐색을 이용하는 단계적인 고정점 계산 가속 기법(thresholded widening using binary search)의 자세한 구현과, 한계점 및 보완사항에 대한 내용들을 다루었다. 또한 실험을 통해 10.32%의 추가 비용으로 평균 4.24% 정도의 허위 경보를 줄이고, 평균 5.28%정도의 경보의 질이 좋아짐을 보였다.

참고문헌

[1] Patrick Cousot and Radhia Cousot, “Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approx- imation of Fixpoints,” in *Proceedings of ACM Symposium on Principles of Programming Languages*, January 1977. pp. 238-252.

[2] Lies Lakhdar-Chaouch, Bertrand Jeannet, and Alain Girault, “Widening with Thresholds for Programs with Complex Control Graphs“ in *Springer, Heidelberg (LNCS, vol. 6996)*, 2011. pp. 492–502.

[3] Hakjoo Oh, Wonchan Lee, Kihong Heo, Hongseok Yang, and Kwangkeun Yi, “Selective context-sensitivity guided by impact pre-analysis.” in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2014. p. 49.

[4] Patrick Cousot and Radhia Cousot, “Systematic design of program analysis frameworks,” in *Proceedings of ACM Symposium on Principles of Programming Languages*, 1979. pp. 269-282.

[5] Patrick Cousot and Radhia Cousot, “Abstract Interpretation Frameworks,” in *Journal of Logic and Computation*, 1992. pp. 511-547

[6] Hakjoo Oh, Kihong Heo, Wonchan Lee, Woosuk Lee, and Kwangkeun Yi, “Design and Implementation of Sparse Global Analyses for C-like Languages,” in *ACM SIGPLAN Conference on Programming Language Design and Implementation (ACM SIGPLAN Notices Volume 47 Issue 6)*, June 2012. pp. 229-238.

[7] Hakjoo Oh, and Kwangkeun Yi. “Access-based abstract memory localization in static analysis.” in *Science of Computer Programming*, 2013, 78.9: 1701-1727.

[8] Sparrow `http://ropas.snu.ac.kr/sparrow`

Abstract

Improving the Precision of Static Analysis by Using Thresholded Widening with Binary Search in C Programs

Sol Kim
Department of Electrical Engineering
and Computer Science
College of Engineering
The Graduate School
Seoul National University

This paper presents a technique to improve precision of interval static analysis by using thresholded widening with binary search in C programs. The conventional widening over-approximates variables in loops, and it causes false alarms. The false alarms are hard to reduce unless widening that analyses more precisely than conventional widening is used. Thresholded widening can remove those false alarms, but it requires huge additional analysis time. To reduce the time, this paper proposes to use binary search in thresholded widening. This paper describes how to remove the false alarms using thresholded widening with binary search, and proves soundness of analysis. Experimental result shows that the analysis time increased by 10.32%, false alarms decreased by 4.24%, and 5.28% alarms becomes more informative.

Keywords: abstract interpretation, static analysis, program analysis, widening
Student Number: 2013-20767