20-2-R05

가벼운 모양분석을 통한 메모리 누수 탐지[1]

Memory-Leak Detection by Lightweight Shape Analysis

이욱세·정승철
한양대학교 안산캠퍼스 컴퓨터공학과
{oukseh; scjung}@pllab.hanyang.ac.kr

김태호
한국전자통신연구원
taehokim@etri.re.kr

요 약

최근 독보적인 연구들을 통해서 포인터를 다루는 복잡한 프로그램에 대한 포인터 오류 및 메모리 누수 탐지 기술이 획기적으로 진화하여, 복잡한 포인터 프로그램의 오류를 감지할 수 있는 기술이 마련되었다. 그러나, 복잡한 자료구조를 위해 비용이 비싸게 설계되었는데, 비교적 단순한 메모리 구조만을 사용하고 있는 실제 현장에 적용하기 어려운 점이 있다. 본 논문에서는 트리 정도 단순한 모양의 단순한 자료구조를 사용하는 포인터 프로그램의 메모리 오류 발견 및 누수 탐지를 위한 저렴한 분석기를 제안하고자 한다. 문법기반 모양분석 기술을 단순화하여 트리에 대해서만 정밀하게 분석하는 분석기를 고안하였고, 실제 코드에 적용한 사례를 보아 가격과 성능이 만족스러울 것이라고 예상된다.

1 서론

최근 독보적인 연구들[3, 4, 7, 8, 9]을 통해서 포인터(pointer)를 다루는 복잡한 프로그램에 대한 포인터 오류 및 메모리 누수 (memory leak) 탐지 기술이 획기적으로 진화하였다. 힙 메모리 (heap memory) 분석은 그 복잡성으로 인하여 한동안 분석 기술이 답보되어 있다가 모양 분석(shape analysis)[4, 9]과 분리 로직(separation logic)[3, 7, 8] 같은 독보적인 기술들이 제안됨으로써 분석기술이 다시 주목받고 있다. 이 기술들은 검증하기 어렵다는 Deutsch-Schorr-Waite 트리 검색 알고리즘[10]이 완전하게 옳다는 것(total correctness)을 검증해 내기도 하였다[6, 11]. 문제는 이러한 검증의 비용은 저렴하지는 않다는 것이다.

실제 현장에서 필요한 기술은 복잡하지 않은 자료구조를 사용하는 프로그램 코드에 대해 저렴한 비용으로 검증하는 기술이다. 실제 리눅스(linux) 운영체제에 사용되는 디바이스 드라이버 (device driver) 소스 코드를 검토해 본 결과, 복잡한 재귀적 자료 구조를 사용하는 경우는 적었다. 재귀적이지 않은 자료 구조를 사용하거나, 재귀적 자료 구조라 하더라도 비교적 단순한 리스트(list)와 유사한 형태의 자료 구조를 사용하고 있다. 복잡한 자료구조를 검증해 내는 것도 분명 필요한 기술이지만 현장에서 사용되는 단순한 자료구조에 대해 효율적인 검증 엔진을 고안하는 것 또한 매우 중요한 일이다.

본 논문에서는 트리 (tree) 모양의 단순한 자료구조를 사용하는 포인터 프로그램의 메모리 오류 발견 및 누수 탐지를 위한 저렴한 모양 분석기를 제안하고자 한다. 제안하는 분석

[1]본 연구는 한국전자통신연구원의 지원으로 이루어졌음.

기는 문법기반 모양분석(grammar-based shape analysis)[4]을 트리 모양의 구조만 정밀하게 분석할 수 있도록 단순화한 형태이고, 분리로직의 프로그램 성질 유추엔진[3]을 리스트 이상의 구조를 분석할 수 있도록 확장한 형태이다.

2 분석 대상 언어

분석 대상 언어는 일반적인 포인터 연산을 하는 프로그램이다.

$$
\begin{array}{lll}
\textit{Variable} & x & \\
\textit{Field} & f & \in \{\texttt{1}, \texttt{2}\} \\
\textit{Expr} & e & ::= x\texttt{==}x \mid x\texttt{==0} \mid \texttt{!}e \mid \texttt{true} \\
\textit{Statement} & t & ::= x\texttt{=}x \mid x\texttt{=0} \mid x\texttt{=}x\texttt{.}f \mid x\texttt{.}f\texttt{=}x \mid x\texttt{=new} \mid \texttt{free}\ x \\
 & & \quad \mid\ t\texttt{;}\ t \mid \texttt{if}\ e\ t\ t \mid \texttt{while}\ e\ t
\end{array}
$$

0는 널 포인터(null pointer)를 의미한다. 명령문의 new는 새로운 힙 셀(heap)을 할당(allocation)해 주며, 할당된 힙 셀은 두 개의 필드(field)로 구성되어 있다고 가정한다. 즉, $x\texttt{.1}$는 변수 x가 가리키는 힙 셀의 첫 번째 필드를 의미한다. $\texttt{free}\,x$는 x가 가리키는 힙 셀만을 반환(disposal)하는 명령문이다.

3 분석 도메인

분석 도메인은 기본적으로 문법기반 모양분석[4]의 것을 단순화한 것이다. 문법기반 모양분석은 힙 메모리를 두 요약 수준으로 요약하는데, 구체적인 부분과, 요약된 부분으로 나누어 요약한다. 구체적인 부분은 실제 힙 메모리를 그대로 표현한 부분이고, 요약된 부분은 힙 메모리의 객체를 문법으로 표현하면서 실제를 뭉뚱그린 부분이다.

문법기반 모양분석에서 단순화한 부분은 문법으로 기술된 요약 부분이다. 문법을 사용하면 다양하고 복잡한 자료구조를 표현할 수 있다. 그러나, 현장에서 사용되는 코드가 그 표현력만큼 복잡한 자료구조를 사용하고 있지 않고, 또한 비용도 저렴하지 않아서, 특정한 형태의 문법 형태만 사용하도록 고안하였다.

사용하는 분석 도메인은 다음과 같다.

$$
\begin{array}{lllll}
\textit{Location} & l & & & \\
\textit{Pointer} & p & ::= & 0 \mid - \mid l & \\
\textit{Cell} & c & ::= & \langle o, o \rangle & \\
\textit{Object} & o & ::= & p \mid c \mid \star p & \\
\textit{Stack} & s & \in & \textit{Variable} \to \textit{Object} & \\
\textit{Heap} & h & \in & \textit{Location} \to \textit{Cell} & \\
\textit{Memory} & m & \in & \textit{Stack} \times \textit{Heap} & \\
\textit{AbstractDomain} & M & \in & \mathcal{P}(\textit{Stack} \times \textit{Heap}) + \{\top\} &
\end{array}
$$

포인터는 주소 또는 널포인터 0, 그리고 끊어진 포인터(dangling pointer) $-$이다. 힙 셀은 두 필드를 위한 객체를 갖고 있는데, 객체는 포인터, 셀, 아니면 요약객체 (abstract object) $\star p$이다. 요약객체 $\star p$는 도달가능한 셀들의 모임인데 반드시 출구는 하나 이하인 것만 가능하다. 출구란 요약객체가 의미하는 셀들의 모임에서 모임 바깥을 가리키는 포인터를 뜻한다. 예를 들어 $\star 0$는 출구가 없는 것을 말하고, $\star l$은 출구가 반드시 하나 있고, 그 출구가 주

소 l을 가리킨다는 뜻이다.[2] 요약 메모리는 두 부분으로 나뉘어 있는데 스택 부분은 변수에서 객체로 가는 맵이고, 힙은 주소에서 셀로 가는 맵이다.[3] 요약 도메인은 메모리의 집합으로 여러 경우를 표현할 수 있도록 고안되었다. 분석이 어려운 경우를 위해 모든 가능성을 의미하는 $\top$을 두었다.

요약객체는 단순하지만 여러 상황을 표현할 수 있다. 예를 들어, x가 트리를 가리키고 있고, y가 트리의 중간을 가리키고 있는 상황을 다음과 같이 기술할 수 있다.

$$s = \{x \mapsto *l, y \mapsto l\}, h = \{l \mapsto \langle *0, *0\rangle\}$$

스택에서 보면 x는 출구가 있는 요약객체를 가리키는데 그 출구가 주소 l을 가리키고, y는 l을 직접 가리키고 있다. 주소 l에는 출구가 없는 요약객체를 가리키고 있는 셀이 있다.

요약 도메인에서의 순서 관계는 $\top$이 가장 높고 나머지는 집합의 포함관계를 따른다. 단지, 같은 모양을 갖고 주소만 다른 두 메모리는 같다고 판정하다.

$$M \sqsubseteq \top$$
$$M_1 \sqsubseteq M_2 \Longleftrightarrow M_1, M_2 \neq \top \wedge \forall m_1 \in M_1.\exists m_2 \in M_2.\exists R.m_1 =_R m_2$$

여기서 $m_1 =_R m_2$는 R이 m_1에 나오는 주소들과 m_2에 나오는 주소들의 일대일 관계(one-to-one relation)이고 m_1의 모든 주소를 관계되는 주소로 변경했을 때 m_2와 동일할 때 성립한다. 예를 들어, $m_1 = (\{x \mapsto 1\}, \{1 \mapsto \langle *0, *0\rangle\})$와 $m_2 = (\{x \mapsto 2\}, \{2 \mapsto \langle *0, *0\rangle\})$에 대해 $m_1 =_{\{(1,2)\}} m_2$가 성립한다.

4 분석기 (Analysis)

분석 엔진은 문법기반 모양분석[4]과 같은 구조를 가지고 있다. 모양분석 엔진의 특징은, 힙 셀에 연산을 수행할 때 초점을 맞추어 요약된 메모리를 구체화하는 것 (focus라 부른다[9]), 그리고, 반복문 등에서 고정점 (fixed-point) 계산이 무한히 반복되는 것을 막기 위해 확장 (widening) 연산[1, 2]을 수행하여 구체화된 메모리를 요약시키는 것이다. 우선, 초점 맞추기 연산과 확장 연산을 설명하고, 실제 분석 엔진을 기술하도록 하겠다.

4.1 초점 맞추기 (Focus)

초점 맞추기는 사용할 메모리를 구체적으로 변환하여 메모리에 대한 코드 수행에 대한 분석이 정확하게 이루어질 수 있도록 준비하는 연산이다. 포인터 값을 읽거나 메모리 내용의 변경은 반드시 구체적인 부분에서 이루어져야 완전히 이루어질 수 있다 (strong update). 본 분석기에서는 두 가지 경우의 초점 맞추기가 있는데 하나는 변수에 접근할 때 초점을 맞추는 것과, 셀의 필드에 접근할 때 초점을 맞추는 것이다. 스택 또는 힙에서 연산을 취하는 점을 제외하고는 비슷하게 정의되었다.

초점 맞추기 연산은 그림 1에 있다: 변수 초점 맞추기(Focus_x)와 필드 초점 맞추기($\mathsf{Focus}_{x.f}$). 두 연산 모두 초점 대상을 보고 세 가지 경우로 나뉘는 데, 초점 대상이 포인터이면 초점이 이미 맞추어진 것이고, 초점 대상이 셀인 경우 셀을 독립시키게 되고, 초점 대상이 요약객체인 경우 요약객체를 한 번 풀어서 경우를 구체화시킨다. 요약객체 $\star p$를 풀 때는 세 가지 경우가 생기는데, 첫 번째 경우는 요약객체가 셀 없이 바로 출구인 경우 포인터 p로 치환하고, 두번째 경우는 요약객체의 입구 셀을 꺼내는 데 출구가 첫 번째 필드쪽

[2] 문법기반 모양분석의 수식을 빌리면 $\star 0$는 "$\alpha \rightarrow \mathrm{nil} \mid \langle \alpha, \alpha\rangle$" 문법의 α가 의미하는 객체이고, $\star l$은 "$\beta(n) \rightarrow n \mid \langle \beta(n), \alpha\rangle \mid \langle \alpha, \beta(n)\rangle$" 문법의 $\beta(l)$이 의미하는 객체이다.

[3] 요약 메모리의 스택, 힙은 실제 의미구조의 스택, 힙과 정확히 일치하지 않는다.

$\mathsf{Focus}_x M = M \neq \top$이면 $\bigcup \{\mathsf{focus}_x\, m \mid m \in M\}$, 그 외의 경우, $\top$

$$\mathsf{focus}_x\,(s,h) = \begin{cases} \{(s,h)\}, & s(x) = p\text{인 경우} \\ \{(s[l/x], h[c/l])\},\ \text{새로운 주소 } l, & s(x) = c\text{인 경우} \\ \bigcup \{\mathsf{focus}_x\,(s[p'/x], h \cup h') \mid (p',h') \in \mathsf{unfold}\,\star p\}, & s(x) = \star p\text{인 경우} \end{cases}$$

$$\mathsf{unfold}\,\star p = \{(p,\emptyset), (l, \{l \mapsto \langle \star p, \star 0\rangle\}), (l, \{l \mapsto \langle \star 0, \star p\rangle\})\},\ \text{새로운 주소 } l$$

$\mathsf{Focus}_{x.1} M = M' \neq \top$이며, $\forall m \in M'.\mathsf{focus}_{x.1}\, m$이 정의되면 (여기서 $M' = \mathsf{Focus}_x M$)
$\bigcup \{\mathsf{focus}_{x.1}\, m \mid m \in M'\}$, 그 외의 경우, $\top$
($\mathsf{Focus}_{x.2} M$도 유사하게 정의)

$$\mathsf{focus}_{x.1}\,(s,h) = \begin{cases} \{(s,h)\} & s(x) = l,\ h(l) = \langle p, o\rangle\ \text{인 경우} \\ \{(s, h[\langle l', o\rangle / l, c/l'])\},\ \text{새로운 주소 } l' & s(x) = l,\ h(l) = \langle c, o\rangle\ \text{인 경우} \\ \bigcup \{\mathsf{focus}_{x.1}\,(s, h[\langle p', o\rangle] \cup h') \mid (p',h') \in \mathsf{unfold}\,\star p\} & s(x) = l,\ h(l) = \langle \star p, o\rangle\ \text{인 경우} \\ \text{정의 안됨}, & \text{그외의 경우} \end{cases}$$

[그림 1] 초점 맞추기 연산 (focus operators).

에 있는 경우, 마지막 경우는 출구가 두 번째 필드쪽에 있는 경우이다. 요약객체의 구체화는 풀기 연산(unfold)을 통해 수행된다.

예를 살펴보면 다음과 같다. 앞서 소개했던 예제, 즉, x가 트리를 가리키고 있고, y가 트리의 중간을 가리키고 있는 상황에서 시작하겠다.

$$s = \{x \mapsto *l, y \mapsto l\}, h = \{l \mapsto \langle *0, *0\rangle\}$$

x에 대해 연산을 하기 위해 초점을 맞추어 보면, x가 요약객체를 가리키고 있으므로, 풀어서 세가지 상황이 생긴다.

$$s = \{x \mapsto l, y \mapsto l\}, h = \{l \mapsto \langle *0, *0\rangle\}$$
$$s = \{x \mapsto l', y \mapsto l\}, h = \{l' \mapsto \langle *l, *0\rangle, l \mapsto \langle *0, *0\rangle\}$$
$$s = \{x \mapsto l', y \mapsto l\}, h = \{l' \mapsto \langle *0, *l\rangle, l \mapsto \langle *0, *0\rangle\}$$

첫번째 경우는 x, y가 동일 셀을 가리키는 경우, 두번째 경우는 x의 첫번째 필드를 통해 공유하는 경우, 마지막 경우는 x의 두번째 필드를 통해 공유하는 경우이다.

4.2 확장 (Widening)

확장 연산은 구체적인 객체를 요약함으로써 분석이 무한히 끝나지 않음을 막아준다 [1, 2]. 제안하는 분석기의 확장 연산은 다음과 같이 정의할 수 있다. 문법기반 모양분석[4]의 경우와 매우 유사하지만, 각 연산이 보다 단순하고 문법을 정리하는 연산이 없다.

$$\mathsf{Widen} = \mathsf{Unify} \circ \mathsf{Bound}_k \circ \mathsf{Fold} \circ \mathsf{Checkleak}$$

첫 번째 연산 Checkleak은 메모리 누수가 있는 경우 분석 결과를 $\top$으로 해 준다. 두 번째 연산 Fold는 구체적인 셀들을 접어서 메모리 객체를 그룹화한다. 세 번째 연산 Bound_k는 그래도 해결할 수 없는 복잡한 자료구조인 경우 깊이를 제한해 준다. 마지막 연산 Unify는 비슷한 경우를 합쳐주어 경우의 수를 줄인다. 확장 연산의 부속 연산자들은 Checkleak을 제외하고 그림 2에 있다.

– 누수 검사 (Checkleak): 메모리 (s,h)에 대해 주소 l이 도달가능하다는 것은 어떤 변수 $x \in \mathrm{dom}(s)$에 대해 $s(x)$에 l이 나타나거나, 도달가능한 주소 $l' \in \mathrm{dom}(h)$에 대해 $h(l')$에

$$\mathsf{Fold}\ M = M \neq \top \text{이면 } \{\mathsf{fold}\ m \mid m \in M\},\ \text{그 외의 경우, } \top$$

$$\mathsf{fold}\ (s,h) = \begin{cases} \mathsf{fold}\ (s\{c/l\}, h'\{c/l\}), & \text{어떤 } l \in \mathrm{dom}(h)\text{에 대해, } h = h' \uplus \{l \mapsto c\} \text{ 이고,} \\ & l\text{이 한 번 } (s,h')\text{에 나타나며 } \mathrm{exit}(c)\text{가 정의된 경우} \\ (s,h), & \text{그외의 경우} \end{cases}$$

$$\mathrm{exit}(o) = \begin{cases} p, & o = p \text{ 또는 } o = \star p\text{인 경우} \\ \mathrm{exit}(o_2), & o = \langle o_1, o_2\rangle \text{ 이고 } \mathrm{exit}(o_1) = 0\text{인 경우} \\ \mathrm{exit}(o_1), & o = \langle o_1, o_2\rangle \text{ 이고 } \mathrm{exit}(o_2) = 0\text{인 경우} \\ \text{정의 안됨}, & \text{그외의 경우} \end{cases}$$

$$o\{c/l\} = \begin{cases} \langle o_1\{c/l\}, o_2\{c/l\}\rangle, & o = \langle o_1, o_2\rangle \text{ 인 경우} \\ c, & o = l\text{인 경우} \\ \star p, & o = \star l\text{이고 } \mathrm{exit}(c) = p\text{인 경우} \\ o, & \text{그 외의 경우} \end{cases}$$

$$(s,h)\{c/l\} = (\{x \mapsto (s(x))\{c/l\} \mid x \in \mathrm{dom}(s)\}, \{l \mapsto (h(l))\{c/l\} \mid l \in \mathrm{dom}(h)\})$$

$$\mathsf{Bound}_k\ M = M \neq \top \text{이면 } \{\mathsf{bound}_k\ m \mid m \in M\},\ \text{그 외의 경우, } \top$$

$$\mathsf{bound}_k\ (s,h) = (s,h')\{-/l_1\}\cdots\{-/l_n\}$$

$$\text{단, } l_i \in \left\{ l \in \mathrm{dom}(h) \quad \mathrm{depth}_{(s,h)}(l) > k \right\} \text{이고 } h = h' \uplus \{l_1 \mapsto c_1, \cdots, l_n \mapsto c_n\}$$

$\mathrm{depth}_{(s,h)}(l)$ = 변수에서 l까지의 경로 중 가장 짧은 것의 길이
경로는 $\rightsquigarrow$로 연결된 것의 나열: $s(x)$에 l이 있으면 $x \rightsquigarrow l$, $h(l)$에 l'이 있으면 $l \rightsquigarrow l'$

$$\mathsf{Unify}\ M = \begin{cases} (M \setminus \{m_1, m_2\}) \cup \{m_1 \oplus_S m_2\}, & M \neq \top\text{이고, } M\text{에 } m_1 \sim_R m_2\text{인 } m_1, m_2, R\text{이 있으면} \\ M & \text{그 외의 경우} \end{cases}$$

$$\begin{aligned} o \sim_R o' \quad &\Longleftrightarrow\ o = \langle o_1, o_2\rangle \wedge o' = \langle o_1', o_2'\rangle \wedge o_1 \sim_R o_1' \wedge o_2 \sim_R o_2' \\ &\quad\ \vee\ (o \notin \mathit{Cell} \vee o' \notin \mathit{Cell}) \wedge (\mathrm{exit}(o), \mathrm{exit}(o')) \in R \\ (s,h) \sim_R (s',h') \ &\Longleftrightarrow\ \mathrm{dom}(s) = \mathrm{dom}(s') \wedge R\text{이 } \mathrm{dom}(h) \times \mathrm{dom}(h')\text{에 대한 일대일 관계} \\ &\quad\ \wedge\ \forall x \in \mathrm{dom}(s).s(x) \sim_R s'(x) \wedge \forall (l,l') \in R.h(l) \sim_R h'(l') \end{aligned}$$

$$o \oplus o' = \begin{cases} \langle o_1 \oplus o_1', o_2 \oplus o_2'\rangle, & o = \langle o_1, o_2\rangle,\ o' = \langle o_1', o_2'\rangle \text{ 인 경우} \\ o, & o, o' \in \mathit{Pointer}\text{인 경우} \\ \star(\mathrm{exit}(o)), & \text{그 외의 경우} \end{cases}$$

$$(s,h) \oplus_R (s',h') = (\{x \mapsto s(x) \oplus s(x') \mid x \in \mathrm{dom}(s)\}, \{l \mapsto h(l) \oplus h'(l') \mid (l,l') \in R\})$$

[그림 2] 확장(widening)을 위한 연산자: Fold, Bound_k, Unify.

l이 나타나는 경우이다. 메모리 (s,h)에 누수가 없다는 것은 모든 주소 $l \in \mathrm{dom}(h)$가 도달가능하다는 것이다. 요약도메인 M에 대해 누수가 없다는 것은 M이 $\top$이 아니고, M에 속한 모든 메모리 (s,h)가 누수가 없다는 것이다.

$$\mathsf{Checkleak}(M) = \begin{cases} M, & M\text{에 누수가 없을 때} \\ \top, & M\text{에 누수가 있을 때} \end{cases}$$

– 접기 (Fold): 접기 대상은 매우 한정적인 객체에 대해서만 행해진다. 자신을 가리키는 포인터가 하나이고, 객체가 가리키는 포인터(출구)가 하나 이하일 때에만 접는다. 함수 exit를 통해서 객체의 출구를 찾는데, 출구가 하나 일 때는 그 포인터, 출구가 없을 때는 널포인터(0)를 결과로 하고, 출구가 여러 개일 때는 정의되지 않는다. 접는 방법은 자신을 가리키는 포인터를 자기자신으로 치환하는 것이다. 치환 과정에서 요약객체의 출구에 치환될 때 객체는 요약되어 치환된다. 즉, 객체의 구체적인 구조는 무시되고 탈출 포인터만 남게된다. 예를 들어, x는 출구가 l인 요약객체를 가리키고, 주소 l에는 첫번째 필드가 0이고, 두 번째 필드가 포인터 l'이면, 그 셀은 x의 요약객체와 통합되어, 결과적으

로 x는 출구가 l'인 요약객체를 가리키게 되는 것이다.

접는다고 모든 셀이 요약되지는 않는다. 문법기반 모양분석에서는 모든 접어진 셀은 문법으로 변환되어 요약될 가능성이 높아진다. 그러나, 본 논문에서 제시한 접기는 요약객체의 출구로 접을 때를 제외하고는 바로 요약되지는 않는다. 단지 주소만 사라지고 그 셀의 모양이 그대로 객체 내에서 유지되게 된다. 예를 들어, x가 길이 3인 리스트를 가리키고 있다고 하자. 이런 경우 문법기반 모양분석에서는 접기 과정을 통해 3개의 셀이 문법으로 변환되어, 마지막 문법 요약 과정에서 x가 임의의 길이의 리스트를 가리키고 있다고 요약되게 된다. 그러나, 본 접기 방법은 셀들의 주소만 사라지고 x가 길이 3의 리스트를 가리키고 있다는 정보가 남게 된다. 특별히 x가 길이가 다른 리스트를 가리키는 추가의 경우가 없을 때는 이후에도 요약되지 않는다.

– 깊이 제한 (Bound_k): 요약 메모리를 접어서 단순화하더라도 분석 도메인 자체가 단순한 트리 모양 구조에만 적합하기 때문에 요약 메모리의 크기가 무한히 증가하는 경우가 발생한다. 이 경우에는 어쩔 수 없이 깊이 제한을 할 수 밖에 없다. 예를 들어, 프로그램이 필드 1, 2가 같은 셀을 가리키도록 셀을 생성하고, 다시 셀을 생성하되 필드 1, 2가 그 전에 생성된 셀을 가르키도록 하기를 반복하면, 전혀 접어지지 않아 무한한 그래프를 만들어 낼 수 있다. 이런 최악의 경우에는 깊이를 재서 깊이가 너무 깊은 것(k 초과)은 끊어진 포인터로 치환해 준다.

– 유사 메모리 통합 (Unify): 유사 메모리를 통합하여 경우의 수를 줄여 준다. 유사하다는 것은 각 스택의 도메인이 같고 힙의 주소간에 일대일대응 관계가 있을 때, 스택의 대응되는 객체끼리 유사하고, 힙의 대응되는 객체끼리 유사하다는 것이다. 두 객체가 유사하다는 것은, 둘 다 셀인 경우 각 필드들이 끼리끼리 유사하여야 하고, 모두 셀이 아닌 경우는 출구가 대응되는 주소로 같아야 하는 것이다. 예를 들면, 포인터 p와 출구가 q인 요약객체는 p, q가 대응되는 주소이면 유사하다.

두 유사한 메모리의 통합은 유사함을 판단하는 것과 비슷하게 통합된다. 두 셀은 각 필드 별로 통합되고, 두 포인터는 포인터로 통합되는데, 그 외의 경우는 모두 요약객체로 통합된다. 즉, 출구 정보만 남기고 모두 요약되는 것이다. 예를 들어, 유사한 메모리에 한쪽은 x가 널 포인터 0를 갖고 있고, 다른 쪽은 x가 모든 필드가 0인 셀을 갖고 있다면, 통합되면서 x는 요약객체를 갖게 되고, 그 출구는 공통된 출구인 0가 되는 것이다. 결국 크기가 다른 객체들은 유사 메모리 통합 과정을 통해 임의의 크기를 가지는 요약객체가 되는 것이다.

4.3 분석 엔진

분석 엔진 또한 문법기반 모양분석의 것[4]과 유사하게 그림 3에 정의되었다. 메모리에 대한 연산을 수행할 때 해당 변수 또는 해당 필드에 대해 초점을 맞춘 후 수행한다. 초점이 맞추어진 부분은 항상 포인터 값을 가지게 된다는 것을 기억하기 바란다. 메모리를 반환하는 `free` 명령문에 대해서는 양쪽 필드 모두에 대해 초점을 맞춘 후 수행한다. 이유는 `free` 명령문은 셀이 소멸되는 것으로 단지 그 변수에 대해서만 메모리 명령을 수행하는 것이 아니라 필드에 있는 포인터들도 동반 소멸되는 것이기 때문이다. 역시 반복문인 `while` 문에서는 확장연산을 수행하여 무한히 반복하지 않도록 한다.

조건문이나 반복문의 조건계산식에 대해서는 가능한 경우를 줄여서 수행한다 (context pruning). 이 때도 역시 조건 대상이 되는 변수에 대해 초점을 맞춘 후 조건을 만족하는지 파악하다. 초점을 맞추고 나면 대상이 모두 포인터가 되기 때문에 조건 만족을 파악하기 용이하다. 끊어진 포인터($-$)는 어떤 것과도 같을 수도 있고 다를 수도 있다. 그 외의 경우는 같고 다름이 실제 의미구조와 일치한다. 즉, 요약 도메인에서의 같은 주소는 실제 의미구조에서도 같은 주소이고, 서로 다른 주소는 실제 의미구조에서도 서로 다른 주소를 의미한다.

$\mathcal{B} : Expr \to AbstractDomain \to AbstractDomain$

$$
\begin{array}{lcl}
\mathcal{B}[\![e]\!]M_0 & = & M_n \neq \top\text{인 경우 } B[\![e]\!]M_n \\
 & & \text{여기서, } \{x_1, \cdots, x_n\} = V(e)\text{이고 } M_i = \mathsf{Focus}_{x_i} M_{i-1} \text{ for } 1 \leq i \leq n \\
 & & V(x\texttt{==}0) = \{x\},\ V(x\texttt{==}y) = \{x, y\},\ V(\texttt{!}e) = V(e) \\
\mathcal{B}[\![e]\!]M & = & \top\ (\text{그 외의 경우}) \\
B[\![x\texttt{==}0]\!]M & = & \{(s,h) \in M \mid s(x) = 0 \vee s(x) = -\} \\
B[\![\texttt{!}(x\texttt{==}0)]\!]M & = & \{(s,h) \in M \mid s(x) \neq 0\} \\
B[\![x\texttt{==}y]\!]M & = & \{(s,h) \in M \mid s(x) = s(y) \vee s(x) = - \vee s(y) = -\} \\
B[\![\texttt{!}(x\texttt{==}y)]\!]M & = & \{(s,h) \in M \mid s(x) \neq s(y) \vee s(x) = - \vee s(y) = -\} \\
B[\![\texttt{!!}e]\!]M & = & B[\![e]\!]M
\end{array}
$$

$\mathcal{S} : Statement \to AbstractDomain \to AbstractDomain$

$$
\begin{array}{lcl}
\mathcal{S}[\![t]\!]M_0 & = & t\text{가 } x\texttt{=}y,\ x\texttt{=}0,\ x\texttt{=}y\texttt{.}f,\ x\texttt{.}f\texttt{=}y,\ x\texttt{=new}\text{에 한해} \\
 & & M_n \neq \top\text{인 경우 } \bigcup\{S[\![t]\!]m \mid m \in M_n\} \\
 & & \text{여기서, } \{a_1, \cdots, a_n\} = E(e)\text{이고 } M_i = \mathsf{Focus}_{a_i} M_{i-1} \text{ for } 1 \leq i \leq n \\
 & & E(x\texttt{=}0) = E(x\texttt{=new}) = \{x\},\ \text{그외의 경우, } E(a\texttt{=}a') = \{a, a'\} \\
\mathcal{S}[\![\texttt{if } e\ t_1\ t_2]\!]M & = & \mathcal{S}[\![t_1]\!](\mathcal{B}[\![e]\!]M) \sqcup \mathcal{S}[\![t_2]\!](\mathcal{B}[\![\texttt{!}e]\!]M) \\
\mathcal{S}[\![t_1\texttt{;}t_2]\!]M & = & \mathcal{S}[\![t_2]\!](\mathcal{S}[\![t_1]\!]M) \\
\mathcal{S}[\![\texttt{while } e\ t]\!]M & = & \mathcal{B}[\![\texttt{!}e]\!](\text{fix } \lambda M'.\mathsf{Widen}(M \sqcup \mathcal{S}[\![t]\!](\mathcal{B}[\![e]\!]M'))) \\
\mathcal{S}[\![t]\!]M & = & \top\ (\text{그 외의 경우})
\end{array}
$$

$$
\begin{array}{lcl}
S[\![x\texttt{=}y]\!](s,h) & = & (s[s(y)/x], h) \\
S[\![x\texttt{=}0]\!](s,h) & = & (s[0/x], h) \\
S[\![x\texttt{=}y\texttt{.}i]\!](s,h) & = & (s[p_i/x], h) \text{ where } h(s(y)) = \langle p_1, p_2 \rangle \\
S[\![x\texttt{.1=}y]\!](s,h) & = & (s, h[\langle s(y), o \rangle / s(x)]) \text{ where } h(s(y)) = \langle p, o \rangle \\
S[\![x\texttt{.2=}y]\!](s,h) & = & (s, h[\langle o, s(y) \rangle / s(x)]) \text{ where } h(s(y)) = \langle o, p \rangle \\
S[\![x\texttt{=new}]\!](s,h) & = & (s[\langle -, - \rangle / x], h) \\
S[\![\texttt{free } x]\!](s,h) & = & (s[-/x], h') \text{ where } h = h' \uplus \{s(x) \mapsto \langle p_1, p_2 \rangle\}
\end{array}
$$

[그림 3] 분석기.

5 구현 및 분석 사례

분석기의 프로토타입(prototype)은 구현이 되었다. Objective Caml 언어[5]로 구현되었고, 입력으로 받는 언어는 C를 단순화한 유사한 중간 언어이다. 향후 C 프로그램을 직접 입력받을 수 있는 분석기로 확장 구현할 예정이다. 구조체의 필드 수가 여러 개인 경우로 확장하였고, 스택 셀의 주소도 사용할 수 있도록 확장하였다. C 프로그램의 메모리 연산들 중 현재 지원하지 않는 기능은, 메모리 필드의 주소를 사용하는 경우, 배열을 사용하는 경우, 공용체(union)을 사용하는 경우, 포인터 정수 연산(pointer arithmetic)을 사용하는 경우 등이다.

프로시저 호출(procedure call)에 대해서는 지역 계산(local reasoning)의 원리[7]를 따라 구현되었다. 기본적으로는 문맥둔감 (context-insensitive) 분석을 수행한다. 그러나, 호출 당시의 모든 메모리를 모두 모아 분석하지는 않는다. 호출된 프로시저가 사용할 메모리 부분만 골라내서 모아 프로시저를 분석하고 결과와 나머지 메모리를 결합하여 호출 후를 분석한다. 이 때 호출된 프로시저가 사용할 메모리는, 전역 변수에서 도달 가능한 메모리와 인자를 통해 전달되는 값에서 도달 가능한 메모리를 말한다. 이 분리 과정에서 분리가 깔끔

하게 처리되지 않는 경우, 즉, 프로시저에 전달할 메모리의 중간 부분을 전달되지 않는 메모리가 가리키고 있는 경우는 정확하게 분석하지 못한다.

디바이스 드라이버 소스 코드 중 필립스 웹 캠에 대한 드라이버 소스 코드에 대해 분석해 본 결과를 사례로 살펴 보기로 한다. 해당 드라이버 소스의 `pwc_handle_frame`이라는 프로시저는 `pdev` 타입의 구조체를 전달 받는다. 이 구조체에는 네 개의 리스트 구조체 포인터 타입의 필드가 있다: `full_frame`, `read_frame`, `empty_frames`, `empty_frames_tail`. 메인 함수를 작성해서 `pwc_handle_frame`을 호출하도록 하였는데 이 때 네 개의 필드를 독립적인 길이 0이상의 리스트를 가리키도록 초기화하였다. 분석 결과 널포인터 접근 오류가 발생하였다. 문제의 소스는 다음과 같다.

```
1:    if (pdev->empty_frames == NULL) {
2:       pdev->empty_frames = pdev->read_frame;
3:       pdev->empty_frames_tail = pdev->empty_frames;
4:   }
5:   else {
6:       pdev->empty_frames_tail->next = pdev->read_frame;
7:       pdev->empty_frames_tail = pdev->read_frame;
8:   }
```

분석해 보면, `empty_frames`가 널이 아니라고 해서 `empty_frames_tail`가 널이 아닌 것은 아니므로, 6번 줄에서 오류가 발생하였다. 두 번째 분석에서는 `empty_frames_tail`을 길이 1이상의 리스트로 초기화하였다. 결과는 6번 줄에서 메모리 누수가 발생할 수 있다고 보고하였다. 이유는 `empty_frames_tail->next`가 가리키고 있는 셀 들이 도달 불가능해 질 수 있기 때문이다. 소스 코드의 초기화 코드를 보고 다음과 같이 제대로 초기화 해 주었다. 두 경우가 있는데 `empty_frames`, `empty_frames_tail` 두 필드가 모두 널인 경우, 또 하나는 다음과 같은 경우이다.

$$l \mapsto \langle \star l' \rangle, l' \mapsto \langle 0 \rangle$$

여기서 l은 `empty_frames`의 주소이고, l'은 `empty_frames_tail`의 주소이다. 즉, l은 리스트를 가리키는 데 그 끝 셀을 l'이 가리킨 다는 것이다. 제대로 초기화한 경우 메모리 오류 및 누수가 없다고 검증해 주었다.

6 향후 연구과제

향후 연구과제로는 실제 사용되는 소스 코드에 대해 분석을 적용하여 분석기가 실용적임을 입증할 필요가 있다. 현재 구현된 프로토타입은 제약이 있지만 많은 기능에 대해 확장되어 있어 일반적인 C 프로그램에 대해 분석할 준비가 되었다. C 전단부와 중간 언어로의 변환기를 구현하여 실제 C 프로그램을 분석할 수 있는 분석기를 구현하고, 실제 성능과 정확도를 파악하는 일이 필요하다. 그 이후에 분석 설계의 경중을 조정할 필요가 있다.

향후 이론적으로 해결할 과제는 양방향 리스트(doubly-linked list) 등 실제 사용되는 조금 더 복잡한 자료 구조에 대한 분석 확장과 프로시저 호출 부분의 개선이다. 트리 모양의 자료 구조외에 자주 사용하는 양방향 리스트까지는 분석할 수 있는 것이 실용적인 메모리 누수 탐지기를 위해 필요한 일이다. 프로시저 호출 부분에는 인자로 전달되는 메모리와 남는 메모리 간에 간섭이 있는 경우 정확도가 떨어지는 요인이 있는데 이를 개선하는 일이 필요하다.

참고문헌

[1] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 238–252, January 1977.

[2] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.

[3] D. Distefano, P. O'Hearn, and H. Yang. A local shape analysis based on separation logic. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 3920 of *Lecture Notes in Computer Science*, pages 287–302, April 2006.

[4] O. Lee, H. Yang, and K. Yi. Automatic verification of pointer programs using grammar-based shape analysis. In *Proceedings of the European Symposium on Programming*, volume 3444 of *Lecture Notes in Computer Science*, pages 124–140, April 2005.

[5] X. Leroy, D. Doligez, J. Garrigue, D. Rémy, and J. Vouillon. The Objective Caml system release 3.09. Institut National de Recherche en Informatique et en Automatique, October 2005. `http://caml.inria.fr`.

[6] A. Loginov, T. Reps, and M. Sagiv. Automated verification of the Deutsch-Schorr-Waite tree-traversal algorithm. In *Proceedings of the International Symposium on Static Analysis*, volume 4134 of *Lecture Notes in Computer Science*, pages 261–279, 2006.

[7] P. O'Hearn, J. C. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In *Proceedings of the CSL*, Lecture Notes in Computer Science, pages 1–19, 2001.

[8] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of the IEEE Symposium on Logic in Computer Science*, pages 55–74, 2002.

[9] M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Transactions on Programming Languages and Systems*, 24(3):217–298, 2002.

[10] H. Schorr and W. Waite. An efficient machine independent procedure for garbage collection in various list structures. *Communication of the ACM*, 10(8):501–506, August 1967.

[11] H. Yang. *Local Reasoning for Stateful Programs.* PhD thesis, Dept. of Computer Science, University of Illinois, Urbana-Champaign, June 2001.

이욱세

– 1991–1995 한국과학기술원 전산학과 학사
– 1995–1997 한국과학기술원 전산학과 석사
– 1997–2003 한국과학기술원 전산학과 박사
– 2003–2004 서울대학교 전자컴퓨터공학부 박사후연구원
– 2004–현재 한양대학교 컴퓨터공학과 교수

<관심분야> 프로그램 분석 및 검증, 타입 시스템, 컴파일러

정승철

– 2002–2006 한양대학교 전자컴퓨터공학부 학사
– 2006–현재 한양대학교 컴퓨터공학과 석사과정

<관심분야> 프로그램 분석 및 검증, 메모리 분석

김태호

– 1991–1995 성균관대학교 정보공학과 학사
– 1995–1997 한국과학기술원 전산학과 석사
– 1997–2005 한국과학기술원 전산학과 박사
– 2005–현재 한국전자통신연구원 임베디드SW연구단 선임연구원

<관심분야> 정형 명세 및 검증, 임베디드 소프트웨어