자바 객체의 스택 저장 가능성 판별을 위한 정적 분석 기법

(Escape Analysis for Stack Allocation in Java)

조 은 선 †

(Eun-Sun Cho)

요 약 자바에서 사용이 끝난 객체의 수집(garbage collection)은 프로그래머의 메모리 관리 부담을 덜어준다는 장점이 있다. 그러나, 수행 속도에 영향을 미치게 되므로 정적 분석 기법을 사용하여 이를 극복하는 기법이 제안되어왔다. 이 중 하나인 스택 저장 기법은, 힙메모리 대신 메소드 스택에 저장될 수 있는 객체들을 수행 전에 분석하여 파악해두는 방식을 따른다. 본 논문에서는 메소드의 호출/반환에 대해 직접 분석하여 자신을 생성한 메소드의 반환 후에도 접근될 소지가 있는 객체들을 판별하는 정적 분석 기법을 제시한다. 이로써 객체와 변수의 지정(assignment) 관계 분석을 기준으로 하는 기존의 스택 저장 기법들에서 간과되었던 객체들 중에서도 스택에 저장할 수 있는 대상을 발견할 수 있도록 한다.

키워드 : 정적 분석, 스택 저장, 요약 해석

Abstract Garbage collecting objects in Java makes memory management easier for the programmer, but it is time consuming. Stack allocation may be an alternative which identifies stack-allocatable objects before the execution, without performance degradation. We suggest an escaping analysis recording the interprocedural movement of the method, to detect an object the method of whose creation may have been already deactivated during the access. Our approach is different from prior works, enables us to handle some cases that are missed in the previous variable-oriented approach.

Key words : static analysis, stack allocation, abstract interpretation

1. 서 론

자바에서는 사용이 끝난 객체의 메모리 공간을 자동적으로 수집(garbage collection)하여 재사용하고 있다. 이 방법은 프로그래머의 메모리 관리 부담을 덜어준다는 장점이 있는 반면 프로그램의 수행 속도를 늦추게 되는 단점이 있다.

이에 대한 대안으로 객체를 스택에 저장하는 기법들이 제안되어왔다[1-3]. 즉, **'new'** 명령으로 생성되는 객체들 중 일부를 힙메모리(heap) 대신 해당 객체를 생성한 메소드의 스택에 저장하게 된다. 이 때, 스택에 저장해도 상관없는 객체와 그렇지 않은 객체들은 정적 분석에 의해 수행 전에 선별되게 된다. 이를 위해 사용되는 정적 분석을 '탈출 분석(escape analysis)'라고 한다.

· 본 연구는 한국과학재단에서 지원하였음(과제번호 R04-2002-000-00093-0)

† 종신회원 : 충북대학교 전기전자컴퓨터학부 교수

eschough@cbnu.ac.kr

논문접수 : 2003년 10월 14일

심사완료 : 2004년 3월 16일

현재까지 제안된 대부분의 스택 저장을 위한 정적 분석 방법들에서는, 리턴 값으로 반환되거나 전역변수, 형식 인자 등의 비-지역적 변수(non-local variable)에 지정되는 객체들 모두가 자신들을 생성한 메소드에서 벗어나 사용되는 것으로 간주, 이들의 스택 저장은 불가능하다고 결정하고 있다. 따라서, 다음 예의 ①, ②, ③에서 생성된 객체들은 모두 메소드 someMethod(SomeClass arg)의 스택에 저장될 수 없다고 판명된다.

```
class SomeClass {
  ...
  public SomeClass someMethod (SomeClass arg) {
   arg.someField = new SomeClass();  // ①
   ...
   this.someField = new SomeClass(); // ②
   ...
   return new SomeClass();           // ③
   }
    private SomeClass someField;
}
```

그림 1

```
class DoThingsObject {
      public DoThingsObject() {
            ...
            this.myData = new MyData("a default value");
            // creates a data object
            // but may not be used outside this method (constructor)
      }
      void setData(String s) {
            ...
            if (s_is_valid)
                  this.myData = new MyData(s);
            ...
      }
      void main() {
            (new DoThingsObject()).setData("an actual value");
      }
      MyData myData;
}
```

그림 2 비-지역적 변수에 지정된 후 사용되지 않는 객체의 예

이러한 방법의 단점은, 실제로 메소드 밖에서 사용되지는 않지만 전역 변수나 형식인자에 지정되거나 리턴됨으로써 스택에 저장될 기회를 잃는 객체들이 생긴다는 점이다. 예를 들어 객체의 필드(field)는 여러 메소드가 공유하므로 비-지역적 변수로 볼 수 있으며, 여기에 지정되는 모든 객체들은 특정 메소드의 스택에 저장이 불가능하다고 판별된다. 그러나 필드에 지정이 되었더라도 경우에 따라서는 자신을 생성한 메소드 밖에서 해당 필드값이 사용되지 않는 경우가 있을 수 있다. 예를 들어 그림 1에서는 생성자(constructor)에서 필드 myData에 디폴트 값으로 지정된 객체가 생성자 반환 후에는 사용되지 않고 있음을 알 수 있다.

본 논문에서는 단일-단계(single phase)요약 해석(abstract interpretation)[4,5]을 사용하여, 객체가 지정되는 변수 성격에 의존하지 않고 보다 직접적으로 스택 저장 가능 여부를 판별하는 정적 분석 기법을 제안한다. 여기서는 각 객체가 자신을 생성한 메소드의 반환 이후에도 접근되는지를 직접 파악하며, 소지가 있으면 힙메모리에, 아니면 메소드 스택에 저장하도록 하는데, 이를 위하여 객체마다 해당 메소드가 호출/반환되는 과정을 기록, 관리하게 된다. 즉, 이 기록으로부터 해당 메소드의 예상되어지는 상태를 분석할 수 있으므로, 만일 반환되었을 가능성이 있는 위치에서 객체가 접근되게 되는 부분이 적어도 하나 있다면 힙메모리에 저장하고, 해당 메소드의 반환된 이후에 접근될 가능성이 없다면 객체는 그 메소드의 스택에 저장하게 된다.

다음 절에서는 본 논문에서 사용될 단순화된 자바 언어의 구문과 의미구조를 소개한다. 3절에서는 이러한 언어에 대해 스택 저장 여부를 분석하기 위한 요약 해석기를 제안한다. 4절과 5절에서는 비교 토의 및 결론을 기술한다.

2. 대상 언어

제안되는 분석 기법에서 대상 언어로 사용될 언어는 자바의 핵심 부분을 간추린 단순화된 형태로서 그 구문은 표 1과 같다. 반복문은 순환 구조 (recursion) 로 대치하도록 하고, 생성자도 따로 두지 않으며 메소드와 동일하게 다루도록 하였다. 필드는 하나만 있다고 가정하였으며 각 메소드는 한 개의 인자를 가진다고 가정하였고, 배열은 현재 고려하지 않는다. 지역 변수도 본 분석에 영향을 미치지 않으므로 고려하지 않았으며, 분석 편의를 위해 자신의 각 필드나 메소드 호출 표현 시에 'this'가 생략되지 않는다고 가정하였다. 이러한 제한은 분석 자체의 성질과는 상관이 없으며 쉽게 확장될 수 있는 부분들이다.

표 1 대상 언어의 구문

P ::= C^*	e ::= x
C ::= **class** c **ext** c {**var** x^* M^*}	| $head.x$
M ::= **method** $m(x)$=e	| $x := e$
	| $head.x := e$
c *class name*	| **new** c
m *method name*	| e ; e
x *variable name*	| **this**
	| **if** e **then** e **else** e
	| $e.m(e)$
	$head$ ::= **x** | **this** | $head.x$

표 2 대상 언어의 의미 구조에서 사용되는 도메인

	$l \in$	OId
object reference	$r \in$	$ORef_\perp = OId \times \mathcal{P}_\perp$
heap	$h \in$	$Heap_\perp = ORef \rightarrow ORef_\perp$
environment	$\sigma \in$	$Env_\perp = ORef_\perp \times ORef_\perp$
interprocedural behavior	$p \in$	$\mathcal{P}_\perp = (m^{\oplus} \mid m^{\ominus})^*$
set of methods	$m \in$	M
escaping	$e \in$	$Escape = ORef_\perp \rightarrow \{\top, \perp\}$

$is\_escaped(p, r) =$ *if* $Net(p - trd(r))$ *contains* $m^{\oplus}$ *then* $\top$ *else* $\perp$
where Net() eliminates the matched pairs of activation and deactivation of the same method invocation, and m is the method of creation[6]

대상 언어의 의미 구조에서 사용되는 도메인들은 표 2에 나타나 있는데, 의미 구조 자체를 나타내는 함수들과 $\mathcal{P}_\perp$, $Escape$ 등 메소드의 호출/반환과 관련된 사항들로 구성되어 있다. 먼저, $ORef$는 객체 참조(object rerference) 들을 위한 도메인으로서 $OId \times \mathcal{P}_\perp$ 로 구성되어 있다. 여기서 OId는 객체 식별자를 가리키며, $\mathcal{P}_\perp$는 객체의 생성 시간을 나타내는 타임 스탬프로서 뒤에 자세히 설명된다. 그리고, 특별한 객체 참조로서 'true'나 'false' 등이 추가되어 있는 것으로 가정한다.

의미 구조 함수인 $\mathcal{E}_I$와 $\mathcal{F}_I$은 표 3에서 보이는 것과 같이 각 표현식에 대하여 의미를 계산하고 있다. 그 타입은 그림 3과 같다.

지역적 수행 환경(local environment)을 나타내는 $Env_\perp$: $ORef_\perp \times ORef_\perp$ 의 원소는 특별 변수인 'this'와 지역 인자인 'x'의 값을 나타내는 쌍으로 되어있다. 따라서 주

$$
\begin{array}{l}
E_I \in Expr \rightarrow (Env_\perp \times Heap_\perp \times P_\perp \times Escape_\perp) \rightarrow \\
\quad (ORef_\perp \times Env_\perp \times Heap_\perp \times P_\perp \times Escape_\perp) \\
F_I \in (Expr \rightarrow (Env_\perp \times Heap_\perp \times P_\perp \times Escape_\perp) \rightarrow \\
\quad (ORef_\perp \times Env_\perp \times Heap_\perp \times P_\perp \times Escape_\perp)) \rightarrow \\
\quad (Expr \rightarrow (Env_\perp \times Heap_\perp \times P_\perp \times Escape_\perp) \rightarrow \\
\quad (ORef_\perp \times Env_\perp \times Heap_\perp \times P_\perp \times Escape_\perp))
\end{array}
$$

그림 3

어진 쌍 $\sigma \in Env_\perp$의 첫 번째 원소와 두 번째 원소의 내용은 각각 σ(this), σ(x)와 같이 나타내기로 한다. 프로그램에서 그 값들은 x, **this**와 같은 표현식에 의해 접근되며, x:=e와 같은 표현식에 의해 변경되는데, **this**의 값을 변경하는 것은 불가능하다. 정의 중 나타난 기호 f [b/a]는 $f\,a$의 값을 b로 가지는 것을 제외하고는 f 와 동일한 함수를, f [b//a]는 f [$b \sqcup (f\ a)$ /a]을 각각 의미한다. 힙메모리를 위한 $Heap_\perp$: $ORef \rightarrow ORef_\perp$의 원소는 각 객체를 주고 필드값을 돌려주는 함수인데, 이 때의 필드값은 또한 객체이다. 프로그램에서 힙메모리는 *head.x* 표현식에 의해 접근되며, *head.x* := e와 **new** C가 변경시킨다. 객체 생성시 필드의 초기값은 매 클래스 c마다 가지고 있는 특수한 값인 *InitField(c)*이다.

메소드가 호출되면 대상 객체와 인자에 의해 구성되는 새로운 수행 환경을 가지고 수행된다. 기존의 수행 환경은 저장되었다가 메소드 반환시 복귀 된다. 메소드 간 호출/반환 양상을 나타내는 $\mathcal{P}_\perp$는 메소드 식별자에 d, u 표시를 윗첨자로 한 것의 연결된 형태이다. d는 해당 메소드의 호출, u는 반환을 각각 표현한다. 예를 들어 주어진 메소드 식별자 m_1과 m_2가 프로그램 시작 이후로 m_1 - m_1 - m_2 순으로 반환 없이 호출되었다면 $m_1{}^d m_1{}^d m_2{}^d$으로 나타낸다. 만일 이 때 메소드 m_2의 수행이 끝나 반환하게 된다면, $m_2{}^u$를 마지막에 붙여 $m_1{}^d m_1{}^d m_2{}^d m_2{}^u$가 된다. 이러한 문자열의 원래 목적은 현재 메소드 스택의 상태를 보여주는 것이다. 하지만 또한 프로그램의 주어진 시간까지 진행 정도를 추상적으로 나타낼 수 있으므로, 추상적인 경과 시간을 나타내는 개념으로 사용된다. 특히 $ORef_\perp$의 정의에서 볼 수 있듯이 객체가 생성될 때 객체 참조 내에 포함되는데, 해당 객체가 생성되었을 때까지 프로그램에서 이루어져온 메소드 호출/반환 기록을 가진다. 결국 이것은 해당 객체가 언제 생성되었는지를 설명해주므로 해당 객체의 '생일(birth-date)'이라고 명명한다. 표 3에서 사용된 + 연산자는 두 문자열을 순서대로 이어 붙인 결과를 나타낸다.

*Escape*은 각 객체를 {⊥,⊤}에 대응시키는 것으로, ⊤는 자신을 생성한 메소드를 벗어나서 사용됨을, ⊥는 그 밖의 경우를 나타낸다. 수행을 시작할 때는 모든 객체 참조에 대해 ⊥으로 설정되어 있다가 객체가 자신을 생성한 메소드가 반환된 후 사용될 때 ⊤로 변경한다. 본 논문에서는 문자열 p로 나타내어지는 특정 시간에 객체 r이 자신을 생성한 메소드에서 벗어나 사용되는가를 '*is_escape(p,r)*' 함수 값으로 표시하기로 한다. 이를 판단하기 위해서는 객체 r의 생일과 주어진 현재 시간 p를 비교하여, 객체의 생성으로부터 주어진 시간까지 일어난 메소드 호출, 반환 기록을 검사한다. 만일 그 사이에 자신을 생성한 메소드 m에 대해 m^u가 발생되었다면 r이 p 시점에서 m을 벗어나 사용되었음을 알 수 있다.

예를 들어, m_1과 m_2가 순서대로 호출된 후 생성된 객

표 3 대상 언어의 의미 구조(concrete interpreter)

$$
\begin{array}{lcll}
\mathcal{F}_I\mathcal{E}_I\,[\![\ x\ ]\!]\langle\,\sigma,h,p,e\,\rangle & = & & \langle\,\sigma(x),\sigma,h,p,e\,\rangle \\
\mathcal{F}_I\mathcal{E}_I\,[\![\ \mathbf{this}\ ]\!]\langle\,\sigma,h,p,e\,\rangle & = & & \langle\,\sigma(this),\sigma,h,p,e\,\rangle \\
\mathcal{F}_I\mathcal{E}_I\,[\![\ head.x\ ]\!]\langle\,\sigma,h,p,e\,\rangle & = & let & \langle\,r,\sigma,h,p,e_1\,\rangle = \mathcal{E}_I\,[\![\ head\ ]\!]\langle\,\sigma,h,p,e\,\rangle \\
 & & in & \langle\,h(r),\sigma,h,p,e_1[is\_escaped(p,r)//r]\,\rangle \\
\mathcal{F}_I\mathcal{E}_I\,[\![\ x{:=}e\ ]\!]\langle\,\sigma,h,p,e\,\rangle & = & let & \langle\,r_1,\sigma_1,h_1,p_1,e_1\,\rangle = \mathcal{E}_I\,[\![\ e\ ]\!]\langle\,\sigma,h,p,e\,\rangle \\
 & & in & \langle\,r_1,\sigma_1[r_1/x],h_1,p_1,e_1\,\rangle \\
\mathcal{F}_I\mathcal{E}_I\,[\![\ head.x{:=}e\ ]\!]\langle\,\sigma,h,p,e\,\rangle & = & let & \langle\,r_1,\sigma,h,p,e_1\,\rangle = \mathcal{E}_I\,[\![\ head\ ]\!]\langle\,\sigma,h,p,e\,\rangle \\
 & & & \langle\,r_2,\sigma_2,h_2,p_2,e_2\,\rangle = \mathcal{E}_I\,[\![\ e\ ]\!]\langle\,\sigma,h,p,e_1\,\rangle \\
 & & in & \langle\,r_2,\sigma_2,h_2[r_2/r_1],p_2,e_2[is\_escaped(p_2,r_1)//r_1]\,\rangle \\
\mathcal{F}_I\mathcal{E}_I\,[\![\ \mathbf{new}\ c\ ]\!]\langle\,\sigma,h,p,e\,\rangle & = & let & r = \langle\,l,p\,\rangle\ new\ l \\
 & & in & \langle\,r,\sigma,h[InitField(c)/r],p,e[\perp/r]\,\rangle \\
\mathcal{F}_I\mathcal{E}_I\,[\![\ e_1.m(e_2)\ ]\!]\langle\,\sigma,h,p,e\,\rangle & = & let & \langle\,r_1,\sigma_1,h_1,p_1,e_1\,\rangle = \mathcal{E}_I\,[\![\ e_1\ ]\!]\langle\,\sigma,h,p,e\,\rangle \\
 & & & \langle\,r_2,\sigma_2,h_2,p_2,e_2\,\rangle = \mathcal{E}_I\,[\![\ e_2\ ]\!]\langle\,\sigma_1,h_1,p_1,e_1\,\rangle \\
 & & & \lambda_m x.e = Method(snd(r_1),m) \\
 & & & \langle\,r_3,\sigma_3,h_3,p_3,e_3\,\rangle = \mathcal{E}_I\,[\![\ e\ ]\!]\langle\,\langle r_1,r_2\rangle,h_2,p_2+m^d,e_2\,\rangle \\
 & & in & \langle\,r_3,\sigma_2,h_3,p_3+m^u,e_3[is\_escaped(r_1,p_1)//r_1][is\_escaped(r_2,p_2)//r_2]\,\rangle \\
\mathcal{F}_I\mathcal{E}_I\,[\![\ e_1;\ e_2\ ]\!]\langle\,\sigma,h,p,e\,\rangle & = & let & \langle\,r_1,\sigma_1,h_1,p_1,e_1\,\rangle = \mathcal{E}_I\,[\![\ e_1\ ]\!]\langle\,\sigma,h,p,e\,\rangle \\
 & & in & \mathcal{E}_I\,[\![\ e_2\ ]\!]\langle\,\sigma_1,h_1,p_1,e_1\,\rangle \\
\mathcal{F}_I\mathcal{E}_I\,[\![\ \mathbf{if}\ e_1\ e_2\ e_3\ ]\!]\langle\,\sigma,h,p,e\,\rangle & = & let & \langle\,r_1,\sigma_1,h_1,p_1,e_1\,\rangle = \mathcal{E}_I\,[\![\ e_1\ ]\!]\langle\,\sigma,h,p,e\,\rangle \\
 & & in & \mathcal{E}_I\,[\![\ e_2\ ]\!]\langle\,\sigma_1,h_1,p_1,e_1[is\_escaped(p_1,r_1)//r_1]\,\rangle \text{ if } r_1 \text{ is true} \\
 & & & \mathcal{E}_I\,[\![\ e_3\ ]\!]\langle\,\sigma_1,h_1,p_1,e_1[is\_escaped(p_1,r_1)//r_1]\,\rangle \text{ otherwise}
\end{array}
$$

체 r의 생일은 $m_1^d m_2^d$이며, m_2가 그 객체 r을 생성시킨 메소드이다. 그런데 만일 r이 $m_1^d m_2^d m_1^d m_1^u m_2^u m_1^d$ 와 같이 표현되는 시점 p에 프로그램에서 접근되었다면, 생일 이후에 벌어진 메소드 호출/반환 기록은, p에서 r의 생일만큼을 제외한 이후의 나머지 부분 즉 $m_1^d m_1^u m_2^u m_1^d$ 가 된다. 이 부분이 m_2^u를 포함하고 있으므로 p시점에서 m_2는 이미 반환되어 r을 생성했던 메소드 스택이 이미 팝업되어 존재하지 않음을 의미한다. 주의할 것은 $m_1^d m_1^u$와 같이 동일 메소드에 대한 호출/반환의 쌍을 이루는 부문자열(substring)은 현재의 스택 상태를 나타내는 데에 영향을 주지 않으므로[6], 결과에 지장을 주지 않고 생략될 수 있다.

3. 스택 저장을 위한 요약 해석기(Abstract Interpreter)

본 절에서는 앞서 소개된 대상 언어에 의미 구조를 요약 해석(abstract interpretation)하여 객체의 스택 저장 가능 여부를 판별하는 정적 분석을 제안한다. 의미 구조의 도메인들은 표 4와 같이 추상화 된다. 여기서는 나머지 $ORef_\perp$, $\mathcal{P}_\perp$, $Escape_\perp$의 추상화 방식을 중심으로 기술한다. $Heap_\perp$과 $Env_\perp$는 각각의 원소를 추상화 하는 방법으로 추상화 될 수 있다.

3.1 객체 참조와 생일

객체 참조의 추상화된 형태는 '프로그램의 어느 위치'에서 '언제' 생성되었는가로 나타낸다. 즉, 해당 객체를 생성한 **new**-표현식 $\beta \in B$와 추상화된 생일 $p \in \mathcal{P}_\perp^\#$의 쌍으로 표현된다. **new**-표현식을 객체 참조에 포함하는 것은 기본적으로 프로그램의 각 객체 생성 위치마다 서로 다른 객체가 생성되는 것으로 인식함을 의미하며, $\mathcal{P}_\perp^\#$을 포함하는 것은 같은 표현식에 의해 생성되는 객체들 중에서도 생성 시점(또는 생성시 스택 상태)에 따라 다른 객체로 인식하고 있음을 뜻한다. 실제로 동일한 **new**-표현식에 의해 다른 객체가 생겨나는 경우는 수행할 메소드가 여러 위치에서 호출되었거나 재귀적으로 호출될 때 등이 있다.

생일을 나타내는 $\mathcal{P}_\perp^\#$는 다음과 같은 절차에 의해 $\mathcal{P}_\perp$를 추상화시킨 결과이다.

첫째, 생일의 문자열 중, 객체를 생성한 메소드 m의 호출/반환 기록만 의미가 있으므로 기타 메소드들에 관한 기록을 제거한다. 그리고 이들은 m^u 또는 m^d로 이루어진 문자열이므로 m을 생략해도 무방하다. 따라서 u와 d만의 연속인 문자열로 나타낼 수 있다.

둘째, 생일 자체를 객체에 기록하지 말고 객체의 생성 이후에 발생된 m의 호출 반환 기록으로 대신한다. 이것은 객체가 생성될 때부터 주어진 현재 시점 p까지 메소드 m의 호출 반환 기록 즉, 나이(age)가 된다. 원래 생일을 기록하는 목적이 후에 주어진 현재 시점 p와의 차이를 비교하여 m^u의 존재를 검사하는 데에 있으므로, 처음부터 객체 생성 시점과 현재 시점과의 차이만을 기록해나가는 것으로 바꾸는 것은 무리가 없다.

표 4 요약 해석을 위한 도메인

labels for new expressions	$\beta \in B$
interprocedural behavior	$\hat{p} \in \mathcal{P}^\# = \{\epsilon, u, dd^+, uu^+, ud, ud^+\} \cup \perp$
object reference	$\hat{r} \in ORef_\perp^\# = B_\perp \times \mathcal{P}_\perp^\#$
heap	$\hat{h} \in Heap_\perp^\# = ORef_\perp^\# \to 2^{ORef_\perp^\#}$
environment	$\hat{o} \in Env_\perp^\# = 2^{ORef_\perp^\#} \times 2^{ORef_\perp^\#}$
escaping	$\hat{e} \in Escape_\perp^\# = B_\perp \to \{\top, \perp\}$

셋째, u,d로 이루어진 임의의 길이의 문자열 대신 $\mathcal{P}_\perp^\# = \{\perp, \boldsymbol{\varepsilon}, \boldsymbol{u}, \boldsymbol{uu^+}, \boldsymbol{d^+}, \boldsymbol{ud}, \boldsymbol{ud^+}\}$ 중 한 원소로 대치한다. 이들은 주어진 객체 o를 생성시킨 메소드 m에 대해 o의 생성 이후 발생된 사건들을 그룹화하여 나타낸 것이다. 각 원소들은 다음과 같은 상태들을 의미한다. 주의할 것은, m이 이미 호출되어 수행되는 동안 o를 생성하였으므로, 아래에 나열된 상태들은 모두 적어도 한번 이상 m의 호출이 발생된 후라는 점이다.

- $\perp$: 아직 분석되지 않음
- $\boldsymbol{\varepsilon}$: m에 대해 아무런 호출/반환도 일어나지 않은 상태
- $\boldsymbol{u}$: o를 생성시킨 m이 반환된 상태
- $\boldsymbol{uu^+}$: m이 두 번 이상 반환된 상태(o를 생성시킨 m을 포함, 객체 생성 이전에 m이 두 번 이상 호출된 경우)
- $\boldsymbol{d^+}$: o를 생성시킨 m을 반환 하지 않은 채로 다시 한 번 이상 m이 호출된 상태
- $\boldsymbol{ud}$: o를 생성시킨 m을 반환한 후 다시 m을 한 번 호출한 상태
- $\boldsymbol{ud^+}$: o를 생성시킨 m을 반환한 후 계속해서 m을 임의의 횟수(0포함)로 반복해서 반환한 후 다시 m을 여러 번 호출한 상태

위의 상태들을 정의하는데에는 du 즉, 호출후 즉시 반환함을 나타내는 문자열은 스택구조에 무관하여 o를 생성한 m이 반환되었는지 여부에 영향을 미치지 않으므로 배제하였다. 위의 경우들 중에 객체를 생성한 메소드가 반환되었을 경우를 나타내는 것은 u 문자가 들어있는 $\boldsymbol{u}$, $\boldsymbol{uu^+}$, $\boldsymbol{ud}$, $\boldsymbol{ud^+}$ 이다. 따라서 Abs_p는 다음과 같이 정의할 수 있다.

정의 1. $Abs_p : P \to M \to \mathcal{P}_\perp^\#$ 는 다음과 같이 정의된다.

$$
Abs_p = \lambda p.\lambda m \begin{cases} \varnothing, & \text{if } p \text{ or } m = \perp \\ \text{let } Net(Trace\ p\ m)) = m^{x_1} \ldots m^{x_n} & \text{otherwise} \\ \quad \text{in case } x_1 x_2 \ldots x_n \in & \\ \qquad \varepsilon : \boldsymbol{\varepsilon} & \\ \qquad d^+ : \boldsymbol{d^+} & \\ \qquad u : \boldsymbol{u} & \\ \qquad uu^+ : \boldsymbol{uu^+} & \\ \qquad ud : \boldsymbol{ud} & \\ \qquad uu^+d^+ | u^+d^+d : \boldsymbol{ud^+} & \end{cases}
$$

단 d^+, u, uu^+, ud, $uu^+d^+ | u^+d^+d$ 등은 정규식(regular

expression)이다.

다음 정리는 위와 같이 그룹화 된 상태들로써 가능한 객체의 나이들이 모두 정의될 수 있다는 것을 의미한다.

정리 1. $\mathcal{P}_\perp^{\#}$의 원소 **ε, u, uu+, d+, ud, ud+** 은 모든 가능한, 객체의 나이들을 배타적(disjoint)으로 포함한다.

증명. *객체의 나이가 'u' 또는 'd'의 단일 문자의 열로 이루어지는 경우라면 정규식(regular expression)으로 표현하였을 때 'u⁺'거나 'd⁺'로 나타내어질 것이다. 이 중에서 'd⁺'는 $\mathcal{P}_\perp^{\#}$의 원소 **d+**를 그대로 나타내며, 'u⁺'는 u가 한번 발생한 u와 두 번이상 발생한 'uu⁺'로 다시 정밀하게 나뉘어질 수 있다. 이것은 $\mathcal{P}_\perp^{\#}$ 의 원소 **u**와 **uu+**를 의미한다.*

*객체의 나이에 'u'와 'd'의 두 문자가 섞여있는 문자열인 경우는, [6]에 따르면 모든 객체의 나이는 ud-bitonic ('u'만의 부 문자열 하나와 'd'만의 부 문자열 하나가 이어져 전체 문자열을 구성)하여 정규식 'u⁺d⁺'로 나타내어지므로, 'u'와 'd'가 각각 한번 발생할 때(정규식 'ud')와 그 밖의 경우로 나누어 볼 수 있다. 따라서 이 각각은 $\mathcal{P}_\perp^{\#}$의 원소 **ud**와 **ud+**를 의미한다.*

m에 대해 아무런 호출 반환이 일어나지 않은 경우는 ε이다. □

추상적으로 표현된 객체 참조는 자신의 내부에 포함되어 있는 나이의 변화에 따라 다른 값을 가지게된다. 예를 들어 한 메소드가 호출되었을 때 이 메소드로부터 이전에 생성되었던 객체들은 현재 나이에 d가 덧붙여지게되고, 반환되면 다시 u가 덧붙여지게 된다. 표 5의 $\oplus$는 u와 d를 각 추상적인 나이에 덧붙였을 때의 결과 상태를 의미하며, 연산자 $\ominus$는 $\hat{p} \ominus k = \{x \mid x \oplus k \ni \hat{p}\}$로 정의되는, $\oplus$의 역함수의 의미이다(단, k는 u 또는 d).

표 5 추상적인 메소드 호출/반환에 따른 나이의 변화

	ϵ	u	d^+	uu^+	ud	ud^+	$\perp$
$\oplus d$	$\{d^+\}$	$\{ud\}$	$\{d^+\}$	$\{ud^+\}$	$\{ud^+\}$	$\{ud^+\}$	ϕ
$\oplus u$	$\{u\}$	$\{uu^+\}$	$\{\epsilon, d^+\}$	$\{uu^+\}$	$\{u\}$	$\{ud, uu^+, ud^+\}$	ϕ
$\ominus d$	ϕ	ϕ	$\{\epsilon, d^+\}$	ϕ	$\{u\}$	$\{uu^+, ud^+\}$	ϕ
$\ominus u$	ϕ	$\{\epsilon, ud\}$	ϕ	$\{u, uu^+\}$	ϕ	ϕ	ϕ

정리 2. k는 u 또는 d일 때, $\mathrm{Abs}_p\ p\ m_k \sqsubseteq \mathrm{Abs}_p\ p\ m \oplus k$이다.

증명. *만일 p가 ⊥이면 +가 정의되지 않으므로 좌변 = ⊥이고, $\mathrm{Abs}_p\ p\ m$ 또한 ⊥이므로 ∅ 즉, ⊥이다. $p \neq \perp$ 이면, $Net(\mathrm{Trace}\ p\ m) = m^{x_1} \ldots m^{x_n}$일 때, $x_1\ x_2 \ldots\ x_n$가 **ε, u, uu+, d+, ud, ud+**인 각각의 경우에 대해 직관적인 추론에 의해 증명한다. 자세한 증명은 생략한다.* □

3.2 추상화

추상화 함수를 정의할 때 주목할 것은 추상화 후 나이 정보가 생일을 대신하게 되므로 현재 시점 p 에 따라 추상화 결과가 달라지게 된다는 점이다. 따라서 각 도메인의 추상화 함수 자체도 주어진 시점 p에 의한 함수의 결과로 정의된다. 다음과 같은 함수들이 추상화를 위해 사용된다. 각각의 올바름에 대한 증명은 지면관계상 본 논문에서는 생략한다.

정의 2. $Abs_r : \mathcal{P} \rightarrow ORef_\perp \rightarrow ORef_\perp^{\#}$는 다음과 같이 정의된다.

$$Abs_r = \lambda p.\lambda r.\begin{cases} \perp, & \text{if } p \text{ or } r = \perp \\ \langle \beta, p' - p \rangle, & otherwise \end{cases}$$

단 $r = \langle o, p' \rangle$, β는 객체 o를 생성시킨 **new**-표현식을 나타냄.

정의 3. $Abs_\sigma : \mathcal{P} \rightarrow Env_\perp \rightarrow Env^{\#}{}_\perp$ 는 다음과 같이 정의된다.

$$Abs_\sigma = \lambda p.\lambda \sigma.\begin{cases} \perp, & \text{if } p \text{ or } \sigma = \perp \\ \langle Abs_{sr}\ p\ \sigma(x), Abs_{sr}\ p\ \sigma(this) \rangle, & otherwise \end{cases}$$

정의 4. $Abs_H : \mathcal{P} \rightarrow Heap_\perp \rightarrow Heap^{\#}{}_\perp$ 는 다음과 같이 정의된다.

$$Abs_H = \lambda p.\lambda h.\lambda \hat{r}.\begin{cases} \perp, & \text{if } p, h, \text{ or } \hat{r} = \perp \\ \bigsqcup_{\forall r. Abs_r\ p\ r = \hat{r}} Abs_r\ p\ h(r), & otherwise \end{cases}$$

정의 5. $Abs_e : \mathcal{P} \rightarrow Escape_\perp \rightarrow Escape^{\#}{}_\perp$는 다음과 같이 정의된다.

$$Abs_e = \lambda p.\lambda e.\lambda \beta.\begin{cases} \perp, & \text{if } p, e, \text{or } \beta = \perp \\ \bigsqcup_{\forall r. \exists p'. \langle \beta, p' \rangle \in Abs_r\ p\ r} e(r), & otherwise \end{cases}$$

의미 구조 함수의 적용 전/후 상태에 대한 추상화는 이러한 단위 도메인의 추상화 함수를 이용하여 다음과 같이 정의될 수 있다.

정의 6. 의미 구조 함수 적용 전 상태에 대한 추상화를 나타내는 $Abs_{pre\text{-}state} : Env_\perp \times Heap_\perp \times \mathcal{P}_\perp \times Escape_\perp \rightarrow Env^{\#}{}_\perp \times Heap^{\#}{}_\perp \times Escape^{\#}{}_\perp$는 다음과 같이 정의된다.

$$Abs_{pre\text{-}state} = \lambda \sigma.\lambda h.\lambda p.\lambda e.\langle Abs_\sigma\ p\ \sigma,\ Abs_H\ p\ h,\ Abs_e\ p\ e \rangle$$

정의 7. 의미 구조 함수 적용 후 상태에 대한 추상화를 나타내는 $Abs_{post\text{-}state} : ORef_\perp \times Env_\perp \times Heap_\perp \times \mathcal{P}_\perp \times Escape_\perp \rightarrow ORef_\perp^{\#} \times Env^{\#}{}_\perp \times Heap^{\#}{}_\perp \times Escape^{\#}{}_\perp$는 다음과 같이 정의된다.

$$Abs_{post\text{-}state} = \lambda r.\lambda \sigma.\lambda h.\lambda p.\lambda e.\langle \{Abs_r\ p\ r\},\ Abs_\sigma\ p\ \sigma,\ Abs_H\ p\ h,\ Abs_e\ p\ e \rangle$$

요약 해석이 수행되는 동안 추상적인 객체 참조가 변화함에 따라 추상적인 힙메모리 상태와 수행 환경, 스택 저장 여부 등이 모두 변경된다. 표 6의 의미 구조는 이러한 과정들을 반영하는 요약 해석을 위해 정의되고 있다. 이 때 기술을 간단히 하기 위해 몇 가지 함수로 표

표 6 스택 저장 분석을 위한 추상(요약) 해석기(일부)

$$
\begin{array}{lcl}
\mathcal{F}^{\#}\mathcal{E}^{\#}[\![\ \mathtt{this}\ ]\!]\langle\hat{\sigma},\hat{h},\hat{e}\rangle & = & \langle\hat{\sigma}(\mathtt{this}),\hat{\sigma},\hat{h},\hat{e}\rangle \\
\mathcal{F}^{\#}\mathcal{E}^{\#}[\![\ head.x:=e\ ]\!]\langle\hat{\sigma},\hat{h},\hat{e}\rangle & = & \mathit{let}\ \langle r\hat{s}_1,\hat{\sigma},\hat{h},\hat{e_1}\rangle=\mathcal{E}^{\#}[\![\ head\ ]\!]\langle\hat{\sigma},\hat{h},\hat{e}\rangle \\
 & & \quad\ \langle r\hat{s}_2,\hat{\sigma_2},\hat{h_2},\hat{e_2}\rangle=\mathcal{E}^{\#}[\![\ e\ ]\!]\langle\hat{\sigma},\hat{h},\hat{e_1}\rangle \\
 & & \mathit{in}\ \ \langle r\hat{s}_2,\hat{\sigma_2},WriteH\ \hat{h_2}\ r\hat{s}_1\ r\hat{s}_2,changeE\ \hat{e_2}\ r\hat{s}_1\rangle \\
\mathcal{F}^{\#}\mathcal{E}^{\#}[\![\ \mathtt{new}_{\beta}\ c\ ]\!]\langle\hat{\sigma},\hat{h},\hat{e}\rangle & = & \mathit{let}\ \hat{r}=\langle\beta,\epsilon\rangle\ \ r\hat{s}=\{\hat{r}\} \\
 & & \mathit{in}\ \ \langle r\hat{s},\sigma,WriteH\ \hat{h}\ r\hat{s}\ \{InitField(c)\},\hat{e}\rangle \\
\mathcal{F}^{\#}\mathcal{E}^{\#}[\![\ e_1.m(e_2)\ ]\!]\langle\hat{\sigma},\hat{h},\hat{e}\rangle & = & \mathit{let}\ \langle r\hat{s}_1,\hat{\sigma_1},\hat{h_1},\hat{e_1}\rangle=\mathcal{E}^{\#}[\![\ e_1\ ]\!]\langle\hat{\sigma},\hat{h},\hat{e}\rangle \\
 & & \quad\ \langle r\hat{s}_2,\hat{\sigma_2},\hat{h_2},\hat{e_2}\rangle=\mathcal{E}^{\#}[\![\ e_2\ ]\!]\langle\hat{\sigma_1},\hat{h_1},\hat{e_1}\rangle \\
 & & \mathit{in}\ \ \begin{cases}\langle r\hat{s}_1,\hat{\sigma},\hat{h},\hat{e}\rangle & \mathit{if}\ r\hat{s}_1=\bot \\ \bigsqcup_{\mathbf{pairwise}}\{\langle\hat{UpRs}\ r\hat{s}_3\ m,\sigma_2,\hat{UpH}\ \hat{h_3}\ m,\hat{e_3}\rangle\mid \\ \quad\langle r\hat{s}_3,\hat{\sigma_3},\hat{h_3},\hat{e_3}\rangle=\mathcal{E}^{\#}[\![\ e\ ]\!] \\ \quad\ \langle\hat{DownEnv}\ \langle\{\hat{r}\},r\hat{s}_2\rangle\ m,\hat{DownH}\ \hat{h_2}\ m, \\ \quad\ changeE\ (changeE\ \hat{e_2}\ r\hat{s}_2)\{\hat{r}\}\rangle \\ \quad\wedge\lambda_m x.e=\hat{Method}(snd(\hat{r}),m)\wedge\hat{r}\in r\hat{s}_1\} & \mathit{otherwise}\end{cases} \\
\mathcal{F}^{\#}\mathcal{E}^{\#}[\![\ \mathtt{if}\ e_1\ e_2\ e_3\ ]\!]\langle\hat{\sigma},\hat{h},\hat{e}\rangle & = & \mathit{let}\ \ \langle r\hat{s}_1,\hat{\sigma_1},\hat{h_1},\hat{e_1}\rangle=\mathcal{E}^{\#}[\![\ e_1\ ]\!]\langle\hat{\sigma},\hat{h},\hat{e}\rangle \\
 & & \mathit{in}\ \ \mathcal{E}^{\#}[\![\ e_2\ ]\!]\langle\hat{\sigma_1},\hat{h_1},changeE\ \hat{e_1}\ r\hat{s}_1\rangle \\
 & & \quad\bigsqcup\mathcal{E}^{\#}[\![\ e_3\ ]\!]\langle\hat{\sigma_1},\hat{h_1},changeE\ \hat{e_1}\ r\hat{s}_1\rangle
\end{array}
$$

표 7 의미 구조에 사용된 보조 함수들(일부)

$$
DownR=\lambda\langle\beta,\hat{p}\rangle.\lambda m.\begin{cases}\bot=\phi & \mathit{if}\ \langle\beta,\hat{p}\rangle=\bot \\ \{\langle\beta,\hat{p'}\rangle\mid\hat{p'}\in\hat{p}\oplus d\} & \mathit{if}\ birthmethod(\beta)=m \\ \{\langle\beta,\hat{p}\rangle\} & \end{cases}
$$

$$
DownH=\lambda\hat{h}.\lambda m.\lambda\langle\beta,\hat{p}\rangle.\begin{cases}\bot & \mathit{if}\ \hat{h}=\bot=\phi \\ \bigsqcup\{DownR\ \hat{h}\langle\beta,\hat{p'}\rangle\ m\mid\hat{p'}\in\hat{p}\ominus d\} & \mathit{if}\ birthmethod(\beta)=m \\ DownR\ \hat{h}\ \langle\beta,\hat{p}\rangle\ m & \mathit{otherwise}\end{cases}
$$

$$
ReadH=\lambda\hat{h}.\lambda r\hat{s}.\bigsqcup_{\hat{r}\in r\hat{s}}\hat{h}(\hat{r}),\quad WriteH=\lambda\hat{h}.\lambda r\hat{s}_1.\lambda r\hat{s}_2.\lambda\hat{r}.\begin{cases}\bot & \mathit{if}\ \hat{r}=\bot \\ \hat{h}(\hat{r}) & \mathit{if}\ \hat{r}\notin r\hat{s}_1 \\ \hat{h}(\hat{r})\sqcup r\hat{s}_2 & \mathit{otherwise}\end{cases}
$$

$$
changeE=\lambda\hat{e}.\lambda r\hat{s}\begin{cases}\bot & \mathit{if}\ \hat{e}=\bot \\ \lambda\beta.\begin{cases}\hat{e}(\beta)\sqcup\top & \mathit{if}\ \hat{p}\in\{u,uu^{+},ud,ud^{+}\} \\ & \mathit{for\ some}\ \langle\beta,\hat{p}\rangle\in r\hat{s} \\ \hat{e}(\beta) & \mathit{otherwise}\end{cases} & \mathit{otherwise}\end{cases}
$$

현하였으며 그 정의의 일부는 표 7에 나타나 있다. 의미 구조 함수 $\mathcal{E}^{\#}$와 $\mathcal{F}^{\#}$는 대상 언어의 요약된 의미를 나타내며, 그림 4와 같은 타입을 가진다.

제안되는 분석 기법의 올바름은 요약 해석이 대상 언어의 의미구조에 대한 올바른 요약인지를 보임으로서 증명된다. 이를 위해 구체적인 의미 구조 전이 함수인 $\mathcal{F}_1$와, 요약 해석의 의미 구조 전이 함수인 $\mathcal{F}^{\#}$의 고정점 간의 관계가 올바른 요약 관계인지 증명한다. 다음 정리

$E^{\#}\in Expr\rightarrow(Env_{\perp}^{\#}\times Heap_{\perp}^{\#}\times Escape_{\perp}^{\#})\rightarrow(2^{ORef^{\#}}\times Env_{\perp}^{\#}\times Heap_{\perp}^{\#}\times Escape_{\perp}^{\#})$
$F^{\#}\in(Expr\rightarrow(Env_{\perp}^{\#}\times Heap_{\perp}^{\#}\times Escape_{\perp}^{\#})\rightarrow(2^{ORef^{\#}}\times Env_{\perp}^{\#}\times Heap_{\perp}^{\#}\times Escape_{\perp}^{\#}))\rightarrow(Expr\rightarrow(Env_{\perp}^{\#}\times Heap_{\perp}^{\#}\times Escape_{\perp}^{\#})\rightarrow(2^{ORef^{\#}}\times Env_{\perp}^{\#}\times Heap_{\perp}^{\#}\times Escape_{\perp}^{\#}))$

그림 4

에서 $fix\ \mathcal{F}_1[expr]$는 의미 구조 자체를 나타내게 되고, $fix\ \mathcal{F}^{\#}[expr]$는 요약 해석의 의미 구조가 된다. 부록에 증명의 일부가 소개되어 있다.

정리 3. 대상 언어의 모든 $expr$에 대해 다음 식이 성립한다.

$$Abs_{post-state}\circ fix\ \mathcal{F}_1[expr]\sqsubseteq fix\ \mathcal{F}^{\#}[expr]\circ Abs_{pre-state}$$

3.3 판별 방식

$\hat{e}\in Escape_{\perp}^{\#}$는 분석의 결과를 나타내는 것으로서 각 **new**-표현식을 $\{\perp,\top\}$ 중 하나로 대응시키고 있다. $\top$는 해당 표현식이 나타내는 모든 객체들이 자신을 생성한 메소드를 벗어나 사용될 수 있다는 의미이며 $\perp$는 그러한 가능성이 없다는 의미이다. 분석이 시작될 때에는 모든 **new**-표현식마다 $\perp$를 설정한다. 요약 해석 도중 해당 new-표현식에 의해 생성된 객체가 자신을 생성한 메소드를 벗어나 사용될 가능성을 발견하는 즉시 $\top$로 변경된다. 결과적으로 계속 $\perp$값을 갖게 되는 **new**-표현

식들은 일반적으로 번역되는 자바 가상기계 언어 '**invoke_special**'[7] 대신 다른 명령으로 번역되어 가상머신으로 하여금 해당 메소드 스택에 저장할 수 있도록 한다.

제안하고 있는 분석 기법은 기존의 방법과 달리 객체가 어떠한 변수에 저장하고 있는지 여부와는 무관하게 진행되므로 더 많은 스택 저장 가능한 객체를 찾아줄 수 있다. 그리고 다음 정리는 제안된 분석 방식이 우선 기존의 방식으로 스택 저장 가능성이 보장되는 객체들을 모두 찾아낼 수 있음을 의미한다.

정리 4. 정적 분석을 통해 비-지역적(non-local) 변수에 저장될 가능성이 없는 것으로 판명되는 모든 **new**-표현식 β에 대하여, 표 6의 요약 해석을 통해 도출되는 $\hat{e}(\beta)$값은 $\perp$이다.

증명. 만일 생성하는 모든 객체가 비-지역적 변수에 저장될 가능성이 없는 **new**-표현식 중, $\hat{e}$ 값이 $\top$인 것이 하나 이상 존재한다고 가정하고 이를 β로 두자. 이러한 β에서 생성하는 모든 객체는, 자신을 생성한 메소드가 반환된 후 사용될 수 있는 가능성이 존재하지 않는다[1]. 따라서 이 경우 요약 해석을 통해 β의 $\hat{e}$ 값이 변경되는 과정을 한번도 거치지 않게되며, 결국 $\hat{e}(\beta)$값은 초기값인 $\perp$를 분석 끝까지 유지하게 된다. 따라서, $\hat{e}$ 값이 $\top$인 β의 존재는 모순이다. □

4. 관련 연구

주어진 자료가 자신을 생성한 환경(context)를 벗어나서 사용되는지를 판단하는 탈출 분석[8,9]이나, 자료의 생존 기간을 사전에 분석하여 병렬성을 추출하거나 메모리 재할당을 하는데에 이용하는 생존기간 분석(*life-time analysis*)[6,10] 등은 함수 언어 프로그램을 대상으로 비교적 오래전부터 시도되어 왔다. 그러나, 함수 언어 프로그램에서는 변화되지 않는 '값(value)'을 중심으로 하므로, 객체가 부수 효과(side effect)를 남기는 객체 지향 프로그램에 대해 같은 분석이 그대로 적용되기 어렵다. 객체가 필드 형태로 다른 객체에 대한 참조를 자유롭게 가질 수 있는 객체지향 프로그램에서는 분석 대상 및 입체적인 망형 구조의 힙메모리에 대한 다른 정의가 필요하다.

Blanchet[1]는 타입 제약을 이용하여 인자로 전달되는 객체에 대해 보다 정교하게 계산하는 기법을 추가하여 본 논문의 제안과는 다른 방향으로 객체지향 프로그램의 정교한 탈출 분석을 도모하였다. 따라서 본 논문의 방식과 함께 적절히 결합하여 사용한다면 보다 정확한 분석을 얻을 수 있을 것이다. [2,3]등에서 제안된 기법들은 탈출 분석 자체를 모듈화 해서 수행 시에 결합할 수 있도록 하여, 정적 분석의 효과를 반감시킬 수 있는 자바의 동적 연결에 응용할 수 있도록 분석 기법을 확장하고 있다. 그러나, 이러한 기법들은 모두 객체가 비-지역적 변수에 저장되는 순간 스택에 저장하는 것을 포기하게 된다[1-3]. 앞서 언급되었듯이 본 연구에서는 어느 변수에 저장 되는지와 무관하게 메소드 종료 후에 사용되지 않는 객체들을 직접 파악하여 스택 저장을 가능하게 하는 데에 초점을 두었다.

본 정적 분석에서 메소드의 호출/반환 기록에 생일 또는 나이 개념을 부여한 것은, 함수 언어의 스택 저장 분석에 대한 연구[6]에 이론적인 기반을 두고 있다. 특히, 객체의 추상적 나이 집합 $\mathcal{P}_I^{\#}$이 가지는 7 가지 원소는, 기존의 Harrison의 함수 호출/반환 기록(inter-procedural movement record)에 대한 카테고리 집합에서 불필요한 것을 없애고 보다 세분화 시킨 것으로 볼 수 있다. 예를 들어, 순환 메소드가 생성하는 객체를 분석하는 도중에 임시로 도입되는 d 카테고리[6]는, 고정점(fixed point) 계산 중에 언제나 $dd+$로 수렴하게 되므로 본 논문에서는 $d+$ 하나로 병합하여 사용하였다. 그리고 상대적으로 정교한 분석을 위해 $ud+$ [6] 대신 ud와 $ud+$로 세분화하였다.

본 정적 분석은 요약 해석에 기반을 두어 모듈별로 분석된 것을 합치는 방식이 아니라 전체 모듈의 수행을 요약하고 있다. 따라서 다른 일반적인 요약 해석을 통한 분석 방법들과 마찬가지로 동적 연결에 적용하는 데에 한계가 있다. 그러나, Cousot[11]의 연구 등에서 제안된 전역 요약 해석 방식을 적용하여 본 요약 해석 결과들을 모듈별로 결합시키는 방안을 찾을 수 있을 것으로 기대된다.

5. 결 론

본 논문에서는 객체가 스택에 저장될 수 있는지 여부를 판단하는 정적 분석을 위한 요약 해석을 제안하고 관련되는 성질을 소개하였다. 기존의 방법은 비-지역적 변수에 저장되는 순간 스택에 저장하는 것을 포기하는 방법을 사용하는데 반해, 본 논문에서는 어느 변수에 저장 되는지 여부와는 상관 없이 메소드 종료 후에 사용되지 않는 객체들을 파악하여 원천적으로 스택 저장 여부를 판별한다. 결과적으로 기존의 기법에 비해 보다 많은 스택 저장 객체를 발견할 수 있다는 장점을 가진다. 이를 위하여 메소드의 호출/반환에 대한 기록을 객체마다 저장해두고 객체가 사용될 때 확인해 봄으로써 객체가 메소드의 반환 이후에도 사용되는지를 판명한다. 본 논문은 동 저자에 의해 동일 주제로 학회에 축약 발표된 것에 관한 확장 기술로서 주요 정리와 정의, 관련 연구가 보다 심도있게 포함된 것이다[12]. 향후 본 연구를 계속 확장하여 스택 저장 외에도 다른 메모리 및 자원 관리 관련 정적 분석을 수행할 계획에 있다.

참 고 문 헌

[1] B. Blanchet. "Escape analysis for object oriented languages application to java," In Proc. of the ACM OOPSLA Conf., 1999.

[2] J-D. Choi et. al, "Escape Analysis for Java," In Proc. of the ACM OOPSLA Conf., 1999.

[3] J. Whaley and M. Rinard. "Compositional Pointer and Escape Analysis for Java Programs," In Proc. of the ACM OOPSLA Conf., 1999.

[4] P. Cousot and R. Cousot. "Abstract interpretation : a unified lattice model for static analysis of programs by construction of approximation," In Proc. of SIGPLAN Conf. on Principle of Programming Languages, 1977.

[5] P. Cousot and R. Cousot. "Abstract interpretation frameworks," Journal of Logic and Computation, 2(4), 1992.

[6] W. L. Harrison, "The Interprocedural Analysis and Automatic Parallelization of scheme Programs," Lisp and Symbolic Computation, 2(3):179-396, 1989.

[7] T. Lindholm and F. Yelline, "The JavaTM Virtual Machine Specification-2nd edition," Addison-Wesley, 1999.

[8] A. Deutsch, "On the Complexity of Escape Analysis," In Proc. of the ACM symposium on principles of Programming Languages(POPL), 1997.

[9] B. Blanchet, "Escape Analysis: Correctness Proof, Implementation and Experimental Results," In Proc. of the ACM symposium on principles of Programming Languages (POPL), 1998.

[10] A. Deutsch, "On Determining lifetime and aliasing of dynamically allocated data in higher-order functional specifications," In Proc. of the ACM symposium on principles of Programming Languages (POPL), 1990.

[11] P. Cousot and R. Cousot, "Modular Static Program Analysis," in Proc. of the Eleventh International Conference on Compiler Construction (CC 2002), 2002.

[12] E.-S. Cho, K. Yi, "Escape Analysis for Stack Allocation in Java," ECOOP workshop on Formal Techniques for Java Programs, 2000.

부록 : 정리 3의 증명

먼저 추상화 함수의 올바름에 관한 증명을 위해 필요한 보조정리를 소개한다. 증명은 생략한다.

보조정리 1. *fix $\mathcal{F}_1$를 적용했을 때 모든 expr에 대해 입력의 세 번째 원소(p)와 출력의 세 번째 원소(p)가 동일하다. 즉 fix $\mathcal{F}_1$를 적용했을 때 p에 대해 변동이 없다.*

보조정리 2. *$Abs_\tau\ p\ h(r) \sqsubseteq Abs_H\ p\ h\ (Abs_\tau\ p\ r)$ 이다. 이것은 Abs_H 함수의 추상화의 올바름에 대한 성질을 나타낸다[4,10].*

다음은 본문의 정리 3의 증명이다. 지면 관계상 일부만을 보인다.

정리 3. *대상 언어의 모든 expr에 대해 다음 식이 성립한다.*

$$Abs_{post\text{-}state} \circ fix\ \mathcal{F}_1[expr] \sqsubseteq fix\ \mathcal{F}^\#[expr] \circ Abs_{pre\text{-}state}$$

증명. *모든 expr에 대해 위의 식을 만족함을 보이기 위해 고정점 귀납법(fixed point induction)을 사용한다. 지면 관계상 일부만 개략적으로 소개하면 다음과 같다.*

(i) 재귀적으로 나타난 $\mathcal{E}_1$와 $\mathcal{E}^\#$가 $\perp$일 때 : $\perp \sqsubseteq \perp$ 성질에 의해

$$Abs_{post\text{-}state} \circ \mathcal{E}_1[expr] \sqsubseteq \mathcal{E}^\#[expr] \circ Abs_{pre\text{-}state}$$

는 어떤 expr에 대해서도 늘 성립한다.

(ii) 재귀적으로 나타난 $\mathcal{E}_1$와 $\mathcal{E}^\#$에 대해

$$Abs_{post\text{-}state} \circ \mathcal{E}_1[expr] \sqsubseteq \mathcal{E}^\#[expr] \circ Abs_{pre\text{-}state}$$

가 성립된다고 가정할 때

$$Abs_{post\text{-}state} \circ \mathcal{F}_1\ \mathcal{E}_1\ [expr] \sqsubseteq \mathcal{F}^\#\mathcal{E}^\#\ [expr] \circ Abs_{pre\text{-}state}$$

임을 보인다.

각 expr에 대해 성립함을 보이는 방식으로 한다. 여기서는 expr = 'head.x := e'인 경우의 일부만 소개한다(다른 expr의 경우도 비슷한 증명 과정을 적용할 수 있다.). 즉, 귀납 가정이 성립한다면, 주어진 어떠한 상태 $<\sigma, h, p, e>$에 대해서도,

$$Abs_{post\text{-}state} \circ \mathcal{F}_1\ \mathcal{E}_1\ [head.x := e]\ <\sigma, h, p, e>$$
$$\sqsubseteq \mathcal{F}^\#\mathcal{E}^\#[head.x:=e] \circ Abs_{pre\text{-}state}\ <\sigma, h, p, e>$$

임을 보인다. 이는 앞서 의미 구조들의 'head.x := e' 항목의 정의에 의해 다음과 같이 바꾸어 쓸 수 있다.

$$<\{Abs_\tau\ p\ r_2\}, Abs_\sigma\ p\ \sigma_2, Abs_H\ p\ h_2[r_2/r_1],$$
$$Abs_e\ p\ e[is\_escaped(p_2,r_1)//r_1]\ >$$
$$\sqsubseteq <r\hat{s}_2, \hat{\sigma}_2, WriteH\ \hat{h}_2\ r\hat{s}_1\ r\hat{s}_2, changeE\ \hat{e}_2\ r\hat{s}_1\ >$$

단, $<r_1, \sigma, h, p, e_1> = \mathcal{E}_1\ [head]\ <\sigma, h, p, e>$, $<r_2, \sigma_2, h_2, p_2, e_2> = \mathcal{E}_1\ [e]\ <\sigma, h, p, e_1>$이고, $<r\hat{s}_1, Abs_\sigma\ p\ \sigma, Abs_H\ p\ h, \hat{e}_1> = \mathcal{E}^\#[head]\ <Abs_\sigma\ p\ \sigma, Abs_H\ p\ h, Abs_e\ p\ e>$, $<r\hat{s}_2, \hat{\sigma}_2, \hat{h}_2, \hat{e}_2> = \mathcal{E}^\#[e]\ <Abs_\sigma\ p\ \sigma, Abs_H\ p\ h, \hat{e}_1>$인 경우이다.

① $\{Abs_\tau\ p\ r_2\} \sqsubseteq r\hat{s}_2$ 와 $Abs_\sigma\ p\ \sigma_2 \sqsubseteq \hat{\sigma}_2$ 임을 증명 : $\mathcal{E}^\#[head]$에 관한 귀납 가정에서,

$$Abs_{post\text{-}state}\ \mathcal{E}_1\ [head]\ <\sigma, h, p, e>$$
$$= <\{Abs_\tau\ p\ r_1\}, Abs_\sigma\ p\ \sigma, Abs_H\ p\ h, Abs_e\ p\ e_1>$$
$$\sqsubseteq <r\hat{s}_1, Abs_\sigma\ p\ \sigma, Abs_H\ p\ h, \hat{e}_1>$$

이다. 따라서 $Abs_e\ p\ e_1 \sqsubseteq \hat{e}_1$임을 알 수 있다.

그런데, $Abs_{post\text{-}state}\ \mathcal{E}_1\ [e]\ <\sigma, h, p, e_1>$

$= Abs_{post\text{-}state}\ <r_2, \sigma_2, h_2, p_2, e_2>$

$= <\{Abs_\tau\ p_2\ r_2\}, Abs_\sigma\ p_2\ \sigma_2, Abs_H\ p_2\ h_2, Abs_e\ p_2\ e_2>$ ($Abs_{post\text{-}state}$ 정의에서)

$\sqsubseteq \mathcal{E}^\#[e]\ <Abs_\sigma\ p\ \sigma, Abs_H\ p\ h, Abs_e\ p\ e_1>$ ($\mathcal{E}^\#[e]$에 관한 귀납 가정에서)

$\sqsubseteq \mathcal{E}^\#[e]\ <Abs_\sigma\ p\ \sigma, Abs_H\ p\ h, \hat{e}_1>$ ($Abs_e\ p\ e_1 \sqsubseteq \hat{e}_1$와 $\mathcal{E}^\#$의 단조 증가 성질-여기서는 증명 생략-에 의해)

$= \langle r\hat{s}_2,\ \hat{\sigma}_2,\ \hat{h}_2,\ \hat{e}_2\rangle$

이다.

즉, $\langle\{Abs_r\ p_2\ r_2\}, Abs_\sigma\ p_2\ \sigma_2\ ...\rangle \sqsubseteq \langle r\hat{s}_2,\ \hat{\sigma}_2, ...\rangle$ 이므로 $\{Abs_r\ p_2\ r_2\} \sqsubseteq r\hat{s}_2$, $Abs_\sigma\ p_2\ \sigma_2 \sqsubseteq \hat{\sigma}_2$ 임을 알 수 있다. 그런데, 보조정리 1에서 $p_2 = p$이므로, 따라서

$\{Abs_r\ p\ r_2\} \sqsubseteq r\hat{s}_2$ 이고, $Abs_\sigma\ p\ \sigma_2 \sqsubseteq \hat{\sigma}_2$

이다.

② $Abs_H\ p\ h_2[r_2/r_1] \sqsubseteq WriteH\ \hat{h}_2\ r\hat{s}_1\ r\hat{s}_2$ 임을 증명 :

$\mathcal{E}^\#[head]$에 관한 귀납 가정에서,

$Abs_{post\text{-}state}\ \mathcal{E}_1[head] \langle\sigma, h, p, e\rangle$

$= \langle\{Abs_r\ p\ r_1\}, Abs_\sigma\ p\ \sigma, Abs_H\ p\ h, Abs_e\ p\ e_1\rangle$

$\sqsubseteq \langle r\hat{s}_1, Abs_\sigma\ p\ \sigma, Abs_H\ p\ h,\ \hat{e}_1\rangle$

이다. 따라서 $\{Abs_r\ p\ r_1\} \sqsubseteq r\hat{s}_1$ 임을 알 수 있다.

그리고 ①에서 $\{Abs_r\ p\ r_2\} \sqsubseteq r\hat{s}_2$ 이므로,

$WriteH\ \hat{h}_2\ \{Abs_r\ p\ r_1\}\ \{Abs_r\ p\ r_2\} \sqsubseteq WriteH\ \hat{h}_2\ r\hat{s}_1\ r\hat{s}_2$

이다. '$\sqsubseteq$'가 전이성(transitivity)을 가지므로, 여기서는

$Abs_H\ p\ h_2[r_2/r_1] \sqsubseteq WriteH\ \hat{h}_2\ \{Abs_r\ p\ r_1\}\ \{Abs_r\ p\ r_2\}$

임을 보임으로써, 궁극적으로 $Abs_H\ p\ h_2[r_2/r_1] \sqsubseteq WriteH\ \hat{h}_2\ r\hat{s}_1\ r\hat{s}_2$ 임을 보인다. 여기서는 이 식의 좌변과 우변을 각각 식 ⓐ와 식 ⓑ으로 놓고, '식 ⓐ $\sqsubseteq$ 식 ⓑ'임을 보인다.

그런데, 식 ⓐ =

$$Abs_H\ p\ h_2[r_2/r_1] = \lambda\hat{r}.\begin{cases}\bot, & \text{if } \hat{r}=\bot\\ \bigsqcup_{\forall r.Abs_r\ p\ r=\hat{r}} Abs_r\ p\ h_2[r_2/r_1](r), & otherwise\end{cases}$$

$$=\lambda\hat{r}.\begin{cases}\bot, & \text{if } \hat{r}=\bot\\ \bigsqcup_{\forall r.Abs_r\ p\ r=\hat{r}\wedge r\neq r_1} Abs_r\ p\ h_2[r_2/r_1](r), & \\ \bigsqcup \bigsqcup_{\forall r.Abs_r\ p\ r=\hat{r}\wedge r=r_1} Abs_r\ p\ h_2[r_2/r_1](r), & otherwise\end{cases}$$

$$=\lambda\hat{r}.\begin{cases}\bot, & \text{if } \hat{r}=\bot\\ \bigsqcup_{\forall r.Abs_r\ p\ r=\hat{r}\wedge r\neq r_1} Abs_r\ p\ h_2(r), & \\ \bigsqcup \bigsqcup_{\forall r.Abs_r\ p\ r=\hat{r}\wedge r=r_1} Abs_r\ p\ h_2, & \text{if } \hat{r}\neq\bot\end{cases}$$

$$=\lambda\hat{r}.\begin{cases}\bot, & \text{if } \hat{r}=\bot\\ \bigsqcup_{\forall r.Abs_r\ p\ r=\hat{r}\wedge r\neq r_1} Abs_r\ p\ h_2(r) \sqcup Abs_r\ p\ r_2, & \\ & \text{if } \hat{r}\neq\bot\wedge Abs_r\ p\ r_1=\hat{r}\\ \bigsqcup_{\forall r.Abs_r\ p\ r=\hat{r}\wedge r=r_1} Abs_r\ p\ h_2(r), & otherwise\end{cases}$$

또한, 위 ①의 증명 과정 중

$Abs_{post\text{-}state}\ \mathcal{E}_1[e] \langle\sigma, h, p, e_1\rangle$

$= \langle\{Abs_r\ p_2\ r_2\}, Abs_\sigma\ p_2\ \sigma_2, Abs_H\ p_2\ h_2, Abs_e\ p_2\ e_2\rangle$

$\sqsubseteq \langle r\hat{s}_2,\ \hat{\sigma}_2,\ \hat{h}_2,\ \hat{e}_2\rangle$

으로 부터 $Abs_H\ p_2\ h_2 \sqsubseteq \hat{h}_2$를 얻을 수 있으므로 식 ⓑ는 다음과 같이 정리할 수 있다.

식 ⓑ = $WriteH\ \hat{h}_2\ \{Abs_r\ p\ r_1\}\ \{Abs_r\ p\ r_2\}$

$\sqsupseteq WriteH\ Abs_H\ p\ h_2\ \{Abs_r\ p\ r_1\}\ \{Abs_r\ p\ r_2\}$

$$= \lambda\hat{r}.\begin{cases}\bot, & \text{if } \hat{r}=\bot\\ Abs_H\ p\ h_2(\hat{r}), & \text{if } \hat{r}\notin Abs_r\ p\ r_1\\ Abs_H\ p\ h_2(\hat{r}) \sqcup Abs_r\ p\ r_2, & \text{if } \hat{r}\in Abs_r\ p\ r_1\end{cases}$$

$$= \lambda\hat{r}.\begin{cases}\bot, & \text{if } \hat{r}=\bot\\ Abs_H\ p\ h_2(\hat{r}), & \text{if } \hat{r}\neq Abs_r\ p\ r_1\\ Abs_H\ p\ h_2(\hat{r}) \sqcup Abs_r\ p\ r_2, & \text{if } \hat{r}=Abs_r\ p\ r_1\end{cases}$$

따라서, 다음 각 ②-i~②-iii의 3가지 경우에 대하여 식 ⓐ $\sqsubseteq$ 식 ⓑ임을 보인다.

②-i) $\hat{r} = \bot$일 때, $\bot \sqsubseteq \bot$이므로 식 ⓐ $\sqsubseteq$ 식 ⓑ가 자명

②-ii) *if* $\hat{r} \neq \bot \wedge Abs_r\ p\ r_1 = r\hat{}$ 일 때, Abs_H의 정의에서

$(Abs_H\ p\ h_2)\ (\hat{r}) = (Abs_H\ p\ h_2)\ (Abs_r\ p\ r_1)$

$= \bigsqcup_{\forall r.Abs_r\ p\ r=Abs_r\ p\ r_1} Abs_r\ p\ h_2(r)$

따라서,

식 ⓑ $\sqsupseteq Abs_H\ p\ h_2\ (\hat{r}) \sqcup Abs_r\ p\ r_2$

$= \bigsqcup_{\forall r.Abs_r\ p\ r=Abs_r\ p\ r_1} Abs_r\ p\ h_2(r) \sqcup Abs_r\ p\ r_2$

$\sqsupseteq \bigsqcup_{\forall r.Abs_r\ p\ r=Abs_r\ p\ r_1\wedge r\neq r_1} Abs_r\ p\ h_2(r) \sqcup Abs_r\ p\ r_2$

= 식 ⓐ

②-iv) *if* $\hat{r}$, $Abs_r\ p\ r_1 \neq \bot \wedge \hat{r} \neq Abs_r\ p\ r_1$

식 ⓐ = $\bigsqcup_{\forall r.Abs_r\ p\ r=\hat{r}\wedge r\neq\hat{r}} Abs_r\ p\ h_2\ (r)$

$\sqsubseteq \bigsqcup_{\forall r.Abs_r\ p\ r=\hat{r}} Abs_r\ p\ h_2\ (r)$

$= Abs_H\ p\ h_2\ (\hat{r})$ (Abs_H의 정의에 의해)

$\sqsubseteq$ 식 ⓑ

③ $Abs_e\ p\ e[is_escaped(p_2,r_1)//r_1] \sqsubseteq changeE\ \hat{e}_2\ r\hat{s}_1$에 대해서도 비슷한 방식으로 증명이 가능하다.

이로써 ①~③으로부터 '$expr = head.x := e$'일 때, $Abs_{post\text{-}state} \circ \mathcal{E}_1[expr] \sqsubseteq \mathcal{E}^\#_1[expr] \circ Abs_{pre\text{-}state}$가 성립된다고 가정할 때 $Abs_{post\text{-}state} \circ \mathcal{F}_1\ \mathcal{E}_1\ [expr] \sqsubseteq \mathcal{F}^\#\mathcal{E}^\#[expr] \circ Abs_{pre\text{-}state}$이다. 이것은 다른 모든 $expr$에 대해서도 비슷한 방식으로 증명이 가능하다.

따라서 위 (i)과 (ii)로부터 귀납법에 의해 모든 $expr$에 대하여 $Abs_{post\text{-}state} \circ fix\ \mathcal{F}_1[expr] \sqsubseteq fix\ \mathcal{F}^\#[expr] \circ Abs_{pre\text{-}state}$ 가 성립한다. □

조 은 선

1991년 서울대학교 자연과학대학 계산통계학과 졸업. 1993년 서울대학교 자연과학대학 전산과학 석사. 1998년 8월 서울대학교 자연과학대학 전산과학 박사. 1999년~2000년 4월 한국과학기술원. 2000년 5월~2002년 2월 아주대학교 정보통신전문대학원 조교수대우. 2002년~현재 충북대학교 전기전자컴퓨터학부 조교수