효율적인 메모리 사용을 위한 free 명령어 삽입 알고리즘*

이욱세O
한국과학기술원 전자전산학과 / 프로그램 분석 시스템 연구단
cookcu@ropas.kaist.ac.kr

An Algorithm to Insert Safe Deallocations for Efficient Memory Usage

Oukseh LeeO
Dept. of EECS / Research On Program Analysis System, KAIST

요 약

메모리 반납 (deallocation) 명령어는 프로그램에게 할당된 힙 셀(heap cell)을 반납하는 명령어로 힙 사용량을 낮추어 주지만, 잘못된 반납으로 인해 심각한 오류를 일으킬 수 있다. 본 논문에서는 재귀적인 자료구조(recursive data structure)를 안전하게 반납하는 명령어를 삽입하는 알고리즘을 제시한다. 메모리의 모양새를 분석하고 나중에 쓰이지 않을 힙 셀들을 추정하여 반납 명령어를 삽입한다. 분석시 요약 수준을 적절히 조절함으로써 빠르면서도 정확하게 분석한다. 또한, 실행시간에 부가적인 정보를 전달하여 일찍 힙 셀을 반납할 수 있도록 한다. 제시한 알고리즘으로 메모리 반납을 하지 않는 프로그램에 반납 명령어를 삽입하여 전체 메모리 할당량의 5.2-98.7%를 반납할 수 있었다.

1 서론

메모리 반납 (*deallocation*) 명령어는 프로그램에게 할당된 힙 셀(heap cell)을 반납하는 명령어로 힙 사용량을 낮추어 줌으로써 프로그램을 효율적으로 수행토록 해준다. 그러나, 잘못된 반납은 심각한 오류를 일으킬 수 있다. 나중에 사용될 힙 셀을 미리 반납해 버리면, 그 힙 셀의 내용을 읽어서 사용하는 부분에서, 메모리 접근 오류 (segmentation fault) 또는 논리적인 오류가 발생할 수 있다.

본 논문에서는 재귀적인 자료구조(recursive data structure)를 안전하게 반납하는 명령어를 삽입하는 알고리즘을 제시한다. 메모리의 모양새를 분석하고 나중에 쓰이지 않을 힙 셀들을 추정하여 반납 명령어를 삽입한다. 분석시 요약 수준을 적절히 조절함으로써 빠르면서도 정확하게 분석한다. 또한, 실행시간에 부가적인 정보를 전달하는 방법을 사용하여, 보다 일찍 힙 셀을 반납할 수 있도록 한다.

예제 1 제시하는 방법의 효과를 극명하게 보여주는 트리 구조(tree)를 복사하는 nML[1]로 작성된 함수를 보자.

```
fun copy t =
  case t of
  | L i       => L i
  | N(t1,t2) => let val p = copy t1
                    val q = copy t2
                in  N(p,q)
                end
```

L은 잎새(leaf)를 위한 데이터 구성자(data constructor)이고 N은 노드(node)를 위한 데이터 구성자이다. 이 예제에 메모리 반납을 삽입하기 어려운 이유는 트리구조가 실제로는 DAG (direct acyclic graph) 구조로 구현될 수도 있기 때문이다.

제시하는 알고리즘으로 변환하면 다음과 같다.

```
fun copy b s t =
  case t of
    L i       => (free t when b; L i)
  | N(t1,t2) => let val p = copy (b && not s) s t1
                    val q = copy b s t2
                in  free t when b;
                    N(p,q)
                end
```

추가로 두 인자(argument)를 받는데, b는 입력 트리가 함수 호출 이후에 사용 안되는지를 의미하는 논리값이고, s는 입력 트리의 내부 힙 셀이 공유되어 있는지를 의미하는 논리값이다. 즉 트리가 이후에 사용되지 않고 내부적으로 공유된 힙 셀이 없으면 트리의 힙 셀 모두 free 명령어에 의해 반납된다. 이후에 사용되지 않으나 내부적으로 공유된 힙 셀이 있으면 루트(root)에서 가장 오른쪽에 있는 잎새까지의 힙 셀들만 반납된다. □

*본 연구는 과학기술부 창의적 연구진흥사업 추진으로 얻어진 결과임

$$
\begin{array}{rrcl}
\textit{Integer} & i & \textit{Loc}\ l & \textit{Var}\ x \quad \textit{DataConst}\ \kappa \quad \textit{Flag Var}\ \beta \\
\textit{Flag} & p & ::= & \beta \mid \mathtt{T} \mid \mathtt{F} \mid p \vee p \mid p \wedge p \mid \neg p \\
\textit{AtomicExpr} & a & ::= & x \mid i \mid \mathtt{fix}\, x^{\tau}\, \lambda\vec{\beta}.\lambda x.e \mid l \\
\textit{Expr} & e & ::= & a \\
 & & \mid & \kappa(a,a) \\
 & & \mid & \mathtt{case}\ a\ m \cdots m \\
 & & \mid & \mathtt{let}\ x = e\ \mathtt{in}\ e \\
 & & \mid & a\ [\vec{p}]\ a \\
 & & \mid & \mathtt{free}\ a\ \mathtt{when}\ p \\
\textit{Match} & m & ::= & \kappa(x,x) \Rightarrow e \\
\textit{Type} & \tau & ::= & \mathrm{int} \mid \tau \to \tau \mid t \mid (\tau,\tau) \to t \\
\textit{Value} & v & ::= & i \mid l \mid \mathtt{fix}\, x\, \lambda\vec{\beta}.\lambda x.e \\
\textit{HeapCell} & h & ::= & \kappa(v,v) \mid \star_{\kappa} \\
\textit{Heap} & H & \in & \textit{Loc} \xrightarrow{fin} \textit{HeapCell} \\
\textit{Continuation} & K & \in & (\textit{Var} \times \textit{Expr})^{\leq k} \\
\textit{State} & & : & \textit{Expr} \times \textit{Heap} \times \textit{Continuation}
\end{array}
$$

$(\kappa(v_1,v_2),H,K) \to (l, H \cup \{l \mapsto \kappa(v_1,v_2)\}, K)$ where $l \notin \mathrm{dom}(H)$
$(\mathtt{free}\ l\ \mathtt{when}\ p, H \cup \{l \mapsto \kappa(v_1,v_2)\}, K) \to (0, H \cup \{l \mapsto \star_{\kappa}\}, K)$ if $p \equiv \mathtt{T}$
$(\mathtt{free}\ l\ \mathtt{when}\ p, H, K) \to (0, H, K)$ if $p \equiv \mathtt{F}$
$(\mathtt{case}\ l\ m_1 \cdots m_k, H, K) \to (e[v_1/x_1][v_2/x_2], H, K)$
where $H(l) = \kappa(v_1,v_2)$ and $m_j = \kappa(x_1,x_2) \Rightarrow e$
$((\mathtt{fix}\ f\ \lambda\vec{\beta}.\lambda x.e)\ [\vec{p}]\ v, H, K) \to$
$(e[(\mathtt{fix}\ f\ \lambda\vec{\beta}.\lambda x.e)/f][\vec{p}/\vec{\beta}][v/x], H, K)$
$(\mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, H, K) \to (e_1, H, (x, e_2) \cdot K)$
$(v, H, (x,e) \cdot K) \to (e[v/x], H, K)$

그림 1: 언어와 의미구조.

2 언어

알고리즘의 결과 언어는 그림 1과 같고, 입력 언어는 반납 명령어 free와 플래그 (flag) 관련 부분이 없는 것이다. 메모리 반납 명령어 "free a when p"는 플래그 p가 참일 때 a가 가리키는 힙 셀을 반납하라는 뜻이다. 플래그는 특별한 값으로 취급하고 함수 호출시에도 별도로 전달한다. 여기서 T, F는 각각 참, 거짓을 뜻하고, 벡터(vector) 표시는 여러 개를 의미한다.

알고리즘의 기술을 간단히 하기 위해 다음을 가정한다. (1) 프로그램 내의 모든 이름은 다르다. (2) 함수내의 모든 이름은 함수의 인자이거나 함수 내에서 묶인 것이다. (3) 데이타 구성자는 두 개의 인자를 갖는다 (왼쪽과 오른쪽). (4) 함수는 메모리에 저장되지 않는다; 즉, 함수는 데이타 구성자(data constructor)의 인자가 될 수 없다.

3 메모리 타입 (memory type)

안전한 메모리 명령어를 삽입하는 알고리즘은 두 단계로 나뉘어져 있다. (1) 하나는 각각의 프로그램 지점(program point)에서 메모리의 모양새와 어떤 힙 셀이 사용되는지 분석하는 메모리 타입 유추 단계이고, (2) 또 하나는 분석된 사용 정보로부터 안전한

Name	X, Y, Z	*PartId*	π
PartClass	o	$::=$	$\mathrm{E} \mid \mathrm{S} \mid \mathrm{L} \mid \mathrm{R}$
PartMap	Π	$\in$	$\mathit{PartId} \xrightarrow{\text{fin}} \mathit{PartClass}$
PartName	X^{Π}	$\in$	$\mathit{Name} \times \mathit{PartMap}$
Names	D	$\in$	$\wp(\mathit{PartName})$
Constraint	$p \hookrightarrow D$	$\in$	$\mathit{Flag} \times \mathit{Names}$
Constraints	$\mathcal{C}$	$\in$	$\wp(\mathit{CondConstraint})$
CollapsedTy	$\bar{\mu}$	$\in$	$\mathit{DataConst} \xrightarrow{\text{fin}} \mathit{Names} \times \mathit{Bool}$
MemTy	μ	$::=$	$\langle D, \kappa, \mu, \mu \rangle \mid \bar{\mu} \mid \tau \to (\mu, D)$
MemEnv	Δ	$\in$	$\mathit{Var} \xrightarrow{\text{fin}} \mathit{MemTy}$
Substitution	S	$\in$	$\mathit{Name} \xrightarrow{\text{fin}} \mathit{Names}$

$\sqsubseteq$ for o, X^{Π}, D, (D, b), μ, and (μ, D)

$\mathrm{L} \sqsubseteq \mathrm{S} \quad \mathrm{R} \sqsubseteq \mathrm{S} \quad o \sqsubseteq o$

$X^{\Pi_1} \sqsubseteq X^{\Pi_2}$ iff $\forall \pi \in \mathrm{dom}(\Pi_2).\ \pi \in \mathrm{dom}(\Pi_1) \wedge \Pi_1(\pi) \sqsubseteq \Pi_2(\pi)$

$D_1 \sqsubseteq D_2$ iff $\forall X^{\Pi_1} \in D_1.\ \exists X^{\Pi_2} \in D_2.\ X^{\Pi_1} \sqsubseteq X^{\Pi_2}$

$(D_1, b_1) \sqsubseteq (D_2, b_2)$ iff $D_1 \sqsubseteq D_2 \wedge (b_1 \Rightarrow b_2)$

$\emptyset \sqsubseteq \mu$

$\bar{\mu} \sqsubseteq \bar{\mu}'$ iff $\mathrm{dom}(\bar{\mu}) \subseteq \mathrm{dom}(\bar{\mu}') \wedge \forall \kappa \in \mathrm{dom}(\bar{\mu}).\ \bar{\mu}(\kappa) \sqsubseteq \bar{\mu}'(\kappa)$

$\langle D_1, \kappa, \mu_1, \mu_1' \rangle \sqsubseteq \langle D_2, \kappa, \mu_2, \mu_2' \rangle$ iff $D_1 \sqsubseteq D_2 \wedge \mu_1 \sqsubseteq \mu_2 \wedge \mu_1' \sqsubseteq \mu_2'$

$\tau \to (\mu, D) \sqsubseteq \tau \to (\mu', D')$ iff $(\mu, D) \sqsubseteq (\mu', D')$

$\mu \sqsubseteq \alpha(\mu)$ if $\alpha(\mu)$ is defined

$(\mu_1, D_1) \sqsubseteq (\mu_2, D_2)$ iff $\mu_1 \sqsubseteq \mu_2 \wedge D_1 \sqsubseteq D_2$

그림 2: 메모리 타입.

메모리 반납 명령어를 삽입하는 단계이다.

메모리 타입은 분석되는 메모리 객체(memory object)의 모양새이다. 요약 수준에 따라 구조 갖춘 (structured) 타입, 무너진 (collapsed) 타입으로 표현된다. 구조 갖춘 타입 $\langle D, \kappa, \mu_1, \mu_2 \rangle$는 입구 셀이 D에 속하고 그 구성자가 κ이고 왼쪽 객체의 타입이 μ_1, 오른쪽 객체의 타입이 μ_2이다. 무너진 타입은 구조는 알 수 없고 단지 도달 가능한 셀들을 구성자 별로 분류해 둔 것이다. 함수 타입은 인자의 일반 타입과 결과의 메모리 타입, 그리고 함수 내부(body)에서 사용하는 힙 셀들로 구성되어 있다. 여기서 결과 타입은 무너진 타입이거나 함수 타입으로 제한한다.

구조 갖춘 타입을 무너뜨릴 때 (α) 공유 정보를 유지한다. 무너뜨리는 것은 타입에 나온 모든 힙 셀의 이름들을 모아서 구성자별로 분류하는 것인데, 이 때 모아지는 이름들중에 공통 부분이 있으면 공유되었다고 한다. 즉, 특정 힙 셀이 입구 셀로부터 도달 가능한 경로가 두 개 이상인 경우를 말한다. "$\vdash D * D'$"는 이름들 D와 D'에 공통 부분이 없음을 의미한다. "rconsts(τ)"는 τ 타입의 메모리 객체가 가질 수 있는 모든 데이타 구성자를 뜻한다.

무너진 타입을 다시 재건축할 때 (γ) 이름을 분할한다. 무너진 타입의 입구 부분만을 재건축하는데, 이 때 입구(entry) 셀에 해당하는 부분(E)과 왼쪽(L), 오른쪽(R)에 해당하는 부분으로 분할한다. 분할 정보는 이름에 붙는데, 예를 들어, $X^{\{\pi \mapsto \mathrm{E}, \pi' \mapsto \mathrm{R}\}}$은 X가 π 시점에 분할된 것 중 입구에 해당하고 π' 시점에 분할된 것 중 오른쪽에 해당하는 영역에 있는 힙 셀들을 의미한다. 분할 시 내부적으로 공유되어 있으면 왼쪽과 오른쪽으로 깨끗하게 나누어지지 않으므로 왼쪽과 오른쪽 대신 입구가 아닌 부분(S)으로 분할한다. 이름의 치환에서도 공유정보를 안전하게 추정하도록 정의되었다.

4 1단계: 메모리 타입 유추

메모리 타입 유추 알고리즘은 그림 4에 있다. "$\Delta \rhd e : (\mu, D)$"는 Δ 환경아래 프로그램 e의 메모리 타입을 유추하면 결과의 메모리 타입은 μ이고 e를 수행하면서 D에 해당하는 힙 셀을 사용한다는 뜻이다.

데이타 생성시에는 (udata) 새로 할당된 힙 셀에 새 이름을 준다. `let`-문에서는 (ulet) e_1을 먼저 유추하고 그 결과를 이용하여 e_2를 유추한다. 사용한 이름들은 e_1과 e_2에서 사용한 것들을 합한 것이되, 현재 환경 Δ와 결과 μ_2에 나오지 않는 이름들은 불필요하므로 제거한다. 여기서 names는 함수 타입에서 나오는 것을 제외하고 나오는 이름들을 주는 함수이다.

선택문에서는 (ucase) x의 메모리 타입을 각각의 구성자별로 재건축한다. 입구에 해당하는 셀(D_i)를 사용하고, 재건축된 메모리 타입으로 각각을 유추하여 결과를 합한다. $\sqcup$은 $\sqsubseteq$의 순서에 의한 최소상한값(least upper bound)를 주는 연산자이다.

$\vdash A * A$ for o, X^{Π}, D, and (D, b)

$$\frac{\vdash \mathrm{E} * \mathrm{S} \quad \vdash \mathrm{E} * \mathrm{L} \quad \vdash \mathrm{E} * \mathrm{R} \quad \vdash \mathrm{L} * \mathrm{R}}{\vdash o_2 * o_1} \qquad \frac{}{\vdash o_1 * o_2}$$

$$\frac{X \neq Y}{\vdash X^{\Pi_1} * Y^{\Pi_2}} \qquad \frac{\exists \pi \in \mathrm{dom}(\Pi_1) \cap \mathrm{dom}(\Pi_2).\ \vdash \Pi_1(\pi) * \Pi_2(\pi)}{\vdash X^{\Pi_1} * X^{\Pi_2}}$$

$$\frac{\forall X^{\Pi_1} \in D_1. \forall X^{\Pi_2} \in D_2.\ \vdash X^{\Pi_1} * X^{\Pi_2}}{\vdash D_1 * D_2} \qquad \frac{\vdash D_1 * D_2}{\vdash (D_1, b_1) * (D_2, b_2)}$$

$\alpha(\mu)$

$$\alpha(\bar{\mu}) = \bar{\mu}$$
$$\alpha(\langle D, \kappa, \mu_1, \mu_2 \rangle) = \{\kappa \mapsto (D, \mathrm{F})\} \uplus \alpha(\mu_1) \uplus \alpha(\mu_2)$$
$$(D_1, b_1) \uplus (D_2, b_2) = (D_1 \cup D_2, b_1 \vee b_2 \vee \neg(\vdash D_1 * D_2))$$
$$\bar{\mu} \uplus \bar{\mu}' = \{\kappa \mapsto \bar{\mu}(\kappa) \uplus \bar{\mu}'(\kappa) \mid \kappa \in \mathrm{dom}(\bar{\mu}) \cup \mathrm{dom}(\bar{\mu}')\}$$

$\gamma_\kappa(\mu)$

$$\gamma_\kappa(\bar{\mu}) = \left\langle D \downarrow \{\pi \mapsto \mathrm{E}\}, \kappa, (\bar{\mu}|_{K_1}) \downarrow \{\pi \mapsto \mathrm{L}\}, (\bar{\mu}|_{K_2}) \downarrow \{\pi \mapsto \mathrm{R}\} \right\rangle$$
where $(D, b) = \bar{\mu}(\kappa)$, $\mathrm{type}(\kappa) = (\tau_1, \tau_2) \to t$,
$K_i = \mathrm{rconsts}(\tau_i)$, and new π
$$\gamma_\kappa(\langle D, \kappa, \mu_1, \mu_2 \rangle) = \langle D, \kappa, \mu_1, \mu_2 \rangle$$
$$D \downarrow \Pi = \cup_{X^{\Pi'} \in D} \{X^{\Pi \cup \Pi'}\}$$
$$(D, \mathrm{F}) \downarrow \Pi = (D \downarrow \Pi, \mathrm{F})$$
$$(D, \mathrm{T}) \downarrow \Pi = (D \downarrow (\Pi[\mathrm{S/L}, \mathrm{S/R}]), \mathrm{T})$$
$$\bar{\mu} \downarrow \Pi = \{\kappa \mapsto \bar{\mu}(\kappa) \downarrow \Pi \mid \kappa \in \mathrm{dom}(\bar{\mu})\}$$

Substitution SA for X, D, (D, b), and μ

$$SX = \begin{cases} (D, b), & \text{if } (X \mapsto (D, b)) \in S \\ (\{X^{\emptyset}\}, \mathrm{F}), & \text{otherwise} \end{cases}$$

$$SD = \begin{cases} \cup_{X^{\Pi} \in D} D' \downarrow \Pi[\mathrm{S/L}, \mathrm{S/R}] \text{ where } SX = (D', b), & \text{if } \mathrm{share}(S, D) \\ \cup_{X^{\Pi} \in D} D' \downarrow \Pi \text{ where } SX = (D', b), & \text{otherwise} \end{cases}$$

$$S(D, b) = (SD, b \vee \mathrm{share}(S, D))$$
$$S\bar{\mu} = \{\kappa \mapsto S(\bar{\mu}(\kappa)) \mid \kappa \in \mathrm{dom}(\bar{\mu})\}$$
$$S\langle D, \kappa, \mu_1, \mu_2 \rangle = \langle SD, \kappa, S\mu_1, S\mu_2 \rangle$$
$$S(\tau \to (\mu, D)) = \tau \to (\mu, D)$$
$$\mathrm{share}(S, D) = \left(\exists X^{\Pi} \in D.\ SX = (D', \mathrm{T})\right) \vee \left(\exists X^{\Pi_1}, Y^{\Pi_2} \in D \text{ s.t. } \vdash X^{\Pi_1} * Y^{\Pi_2}.\ \neg(\vdash (SX) \downarrow \Pi_1 * (SY) \downarrow \Pi_2)\right)$$

그림 3: 메모리 타입의 연산: 공통 부분 없음, 무너뜨리기, 재건축, 이름 치환.

함수 호출은 (uapp) 치환을 통해 해결한다. 함수의 메모리 타입에는 오직 A_κ와 R_κ의 특수한 이름만 사용되는데, A_κ는 인자에 해당하고 R_κ는 함수 내부에서 새로 만들어진 힙 셀 중 결과에 포함되는 것들을 의미한다. 그러므로 함수 호출시에는 A_κ를 실제 인자 이름들로, R_κ를 새 이름으로 치환하면 된다.

함수의 타입은 고정점 계산(fixed-point iteration)을 통해 얻어진다. 함수의 일반 타입이 $\tau \to \tau'$일 경우, 함수의 메모리 타입을 $\tau \to (\emptyset, \emptyset)$으로 가정하고 함수의 내용(body)를 유추한다. 여기서 인자의 메모리 타입은 $[\![\cdot]\!]$을 통해 얻어진다. 인자가 함수 타입일 경우는 최악의 메모리 타입을 가정한다. 유추한 결과는 정리한 후 (simp) 함수의 결과 타입으로 가정하고 다시 함수 내용을 분석한다. 이 작업은 함수 타입이 변하지 않을때까지 한다. 유추된 결과 (μ, D)를 정리하는 방법은, 결과 타입을 무너뜨리고 (α) 분할정보를 모두 떼어낸 후 ($\lfloor \cdot \rfloor$), 새로운 이름들을 통합한다 (ψ). 여기서 치환 ψ는 앞서 정의된 이름 치환(S)과는 달리 공유정보에 영향을 주거나 받지 않고 이름만 바꾼다.

5 2단계: 명령어 삽입

삽입 알고리즘은 그림 5에 있다. "$\mathcal{C} \rhd e \Rightarrow e' : \mathcal{C}'$"는 $\mathcal{C}$ 조건 아래에서 e를 e'으로 변환하고 추가적으로 $\mathcal{C}'$ 조건이 나온다는 뜻이다. 여기서 $\mathcal{C}$는 변환도중 이미 반납한 이름과 이후에 사용될 이름들이고, $\mathcal{C}'$은 e를 변환하면서 반납한 이름들이다. 삽입 알고리즘의 입력 프로그램에는 메모리 타입 유추의 결과($\Delta, (\mu, D)$ 또는 Δ, μ)가 붙어 있다고 가정하고 필요할때만 표기하였다.

알고리즘은 다음 전략으로 고안되었다. (1) 함수는 두 개의 플래그 인자를 가진다. β는 인자가 나중에 사용되지 않을 것인가이고 β_s는 인자가 내부적으로 공유되어 있는가이다. (2) 반납 명령어는 할당 직전에만 삽입한다. 이유는 반납과 할당은 재사용으로 간주할 수 있어 효율적으로 수행 가능하기 때문이다. (3) 함수의 결과는 함수 내부(body)에서 반납하지 않는다.

전략 (2)에 의해 데이타를 생성할 때만 반납 명령어를 삽입한

$\boxed{\Delta \rhd a : \mu}$ $\dfrac{}{\Delta \rhd x : \Delta(x)}$ (uvar) $\dfrac{}{\Delta \rhd i : \emptyset}$ (uint)

$$\frac{(\mu, D) \stackrel{\text{let}}{=} \text{fix}_{(\emptyset,\emptyset)}^{\sqsubseteq}\Big(\begin{array}{l}\lambda(\mu, D).\text{simp}(\mu', D') \text{ where} \\ \{f \mapsto \tau \to (\mu, D),\ x \mapsto [\![\tau]\!]\} \rhd e : (\mu', D')\end{array}\Big)}{\Delta \rhd \texttt{fix}\, f^{\tau\to\tau'}\, \lambda x.e : \tau \to (\mu, D)} \text{ (ufun)}$$

$\boxed{\Delta \rhd e : (\mu, D)}$ $\dfrac{\Delta \rhd a : \mu}{\Delta \rhd a : (\mu, \emptyset)}$ (uatom)

$$\frac{\Delta \rhd a_1 : \mu_1 \quad \Delta \rhd a_2 : \mu_2 \quad \text{new } X}{\Delta \rhd \kappa(a_1, a_2) : \left(\left\langle \{X^\emptyset\}, \kappa, \mu_1, \mu_2 \right\rangle, \emptyset\right)} \text{ (udata)}$$

$$\frac{\begin{array}{c}\Delta \rhd e_1 : (\mu_1, D_1) \\ \Delta \cup \{x \mapsto \mu_1\} \rhd e_2 : (\mu_2, D_2)\end{array}}{\Delta \rhd \texttt{let } x = e_1 \texttt{ in } e_2 : (\mu_2, (D_1 \cup D_2)|_{\text{names}(\Delta)\cup\text{names}(\mu)})} \text{ (ulet)}$$

$$\frac{\begin{array}{l}\forall m_i = \kappa_i\,(x_i, y_i) \texttt{ => } e_i \text{ s.t. } \gamma_{\kappa_i}(\Delta(x)) = \left\langle D_i, \kappa_i, \mu_i, \mu_i' \right\rangle : \\ \Delta \cup \{x \mapsto \left\langle D_i, \kappa_i, \mu_i, \mu_i' \right\rangle, x_i \mapsto \mu_i, y_i \mapsto \mu_i'\} \rhd e_i : (\mu_i, D_i')\end{array}}{\Delta \rhd \texttt{case } x\ m_1 \cdots m_n : (\sqcup_i \mu_i,\ \cup_i (D_i' \cup D_i))} \text{ (ucase)}$$

$$\frac{\begin{array}{c}\Delta(x) = \tau \to (\mu, D) \quad \Delta \rhd a : \mu' \quad \text{new } X_\kappa\text{'s} \\ S \stackrel{\text{let}}{=} \left\{ A_\kappa \mapsto \alpha(\mu')(\kappa) \;\middle|\; \begin{array}{l}\kappa \in \text{rconsts}(\tau), \\ \mu' \text{ is not a function type}\end{array} \right\} \\ \cup \left\{ R_\kappa \mapsto (\{X_\kappa^\emptyset\}, \texttt{F}) \mid \kappa \in \text{dom}(\mu),\ \mu \text{ is collapsed} \right\}\end{array}}{\Delta \rhd x\, a : (S\mu, SD)} \text{ (uapp)}$$

$\boxed{[\![\tau]\!]}$

$[\![\tau]\!] = \{\kappa \mapsto A_\kappa \mid \kappa \in \text{rconsts}(\tau)\}$ if τ is not a function type

$[\![\tau_1 \to \tau_2]\!] = \tau_1 \to (\mu, D)$ where

$K_1 = \text{rconsts}(\tau_1)$, $K_2 = \text{rconsts}(\tau_2)$

$\mu =$ if τ_2 is a function type, then $[\![\tau_2]\!]$, otherwise,

$\left\{\kappa \mapsto (\{A_\kappa^\emptyset, R_\kappa^\emptyset\}, \texttt{T}) \mid \kappa \in K_1 \cap K_2\right\} \cup \left\{\kappa \mapsto (\{R_\kappa^\emptyset\}, \texttt{T}) \mid \kappa \in K_2 \backslash K_1\right\}$

$D = \{\{A_\kappa \mid \kappa \in K_1\} \cup \{R_\kappa \mid \kappa \in K_2\}\}$

$\boxed{\text{simp}(\mu, D)}$

$\text{simp}(\tau \to (\mu, D), D') = (\tau \to (\mu, D), |D'|)$

$\text{simp}(\mu, D) = \psi|(\alpha(\mu), D)|$

where $\psi = \left\{ R_\kappa / X \;\middle|\; (\kappa \mapsto (D', b)) \in \alpha(\mu),\ X^\emptyset \in D',\ X \neq A_\kappa \right\}$

그림 4: 메모리 타입 유추 알고리즘.

$\boxed{\rhd a \Rightarrow a}$ $\dfrac{}{\rhd x \Rightarrow x}$ (ivar) $\dfrac{}{\rhd i \Rightarrow i}$ (iint)

$$\frac{\{\texttt{T} \hookrightarrow \text{namcs}(\mu)\} \cup \{\neg\beta \hookrightarrow \text{namcs}([\![\tau]\!])\} \rhd e \Rightarrow e' : \mathcal{C}}{\rhd \texttt{fix}\, f^{\tau\to\tau'}\, \lambda x.(e^{\Delta,(\mu,D)}) \Rightarrow \texttt{fix}\, f^{\tau\to\tau'}\, \lambda\beta.\lambda\beta_s.\lambda x.e'} \text{ (ifun)}$$

$\boxed{\mathcal{C} \rhd e \Rightarrow e : \mathcal{C}}$ $\dfrac{\rhd a \Rightarrow a'}{\mathcal{C} \rhd a \Rightarrow a' : \emptyset}$ (iatom)

$$\frac{\begin{array}{c}\rhd a \Rightarrow a' \quad D \stackrel{\text{let}}{=} \text{namcs}(\mu) \quad p = \text{wcfree}(D, \mathcal{C}) \\ q = \exists(\kappa \mapsto (D', \texttt{T})) \in \alpha(\mu) \vee (\beta_s \wedge (\exists A_\kappa \in D))\end{array}}{\mathcal{C} \rhd x\,(a^{\Delta,\mu}) \Rightarrow x\,[p, q]\,a' : \{p \hookrightarrow D\}} \text{ (iapp)}$$

$$\frac{\mathcal{C} \cup \{\texttt{T} \hookrightarrow D\} \rhd e_1 \Rightarrow e_1' : \mathcal{C}_1 \quad \mathcal{C} \cup \mathcal{C}_1 \rhd e_2 \Rightarrow e_2' : \mathcal{C}_2}{\mathcal{C} \rhd \texttt{let } x = e_1 \texttt{ in } (e_2^{\Delta,(\mu,D)}) \Rightarrow \texttt{let } x = e_1' \texttt{ in } e_2' : \mathcal{C}_1 \cup \mathcal{C}_2} \text{ (ilet)}$$

$$\frac{\forall m_i = \kappa_i(x_i, y_i) \texttt{ => } e_i.\mathcal{C} \rhd e_i \Rightarrow e_i' : \mathcal{C}_i \wedge m_i' \stackrel{\text{let}}{=} \kappa_i(x_i, y_i) \texttt{ => } e_i'}{\mathcal{C} \rhd \texttt{case } x\ m_1 \cdots m_n \Rightarrow \texttt{case } x\ m_1' \cdots m_n' : \cup_i \mathcal{C}_i} \text{ (icase)}$$

$$\frac{\beta x.\Delta(x) = \langle D, \kappa', \mu_1, \mu_2\rangle \quad \rhd a_1 \Rightarrow a_1' \quad \rhd a_2 \Rightarrow a_2'}{\mathcal{C} \rhd (\kappa(a_1, a_2))^{\Delta,(\mu,D')} \Rightarrow \kappa(a_1', a_2') : \emptyset} \text{ (inofree)}$$

$$\frac{\begin{array}{c}\exists x.\Delta(x) = \langle D, \kappa', \mu_1, \mu_2\rangle \quad p \stackrel{\text{let}}{=} \text{wcfree}(D, \mathcal{C}) \\ \rhd a_1 \Rightarrow a_1' \quad \rhd a_2 \Rightarrow a_2'\end{array}}{\mathcal{C} \rhd (\kappa(a_1, a_2))^{\Delta,(\mu,D')} \Rightarrow \texttt{free } x \texttt{ when } p;\ \kappa(a_1', a_2') : \{p \hookrightarrow D\}} \text{ (idata)}$$

$\boxed{\text{wcfree}(D, \mathcal{C})}$

$\text{wcfree}(D, \mathcal{C}) = (p_1 \wedge \neg\beta_s) \vee (p_2 \wedge \beta_s)$ where

$p_1 = \text{wc}(D, \mathcal{C})$, $p_2 = \text{wc}(SD, S\mathcal{C})$, and

$S = \{A_\kappa \mapsto (\{A_\kappa\}, \texttt{T}) \mid A_\kappa \text{ occurs in } \mathcal{C} \text{ or } D\}$

$\text{wc}(D, \mathcal{C}) = \bigwedge_{(p \hookrightarrow D') \in \mathcal{C} \text{ s.t. } \neg(\vdash D * D')} \neg p$

그림 5: 삽입 알고리즘.

	줄수	분석시간 (ms)	총할당량 (words)	반납 (words)	비율
`sieve`	18	4	161112	15000	9.3%
`quick-sort`	24	6	137247	135450	98.7%
`merge-sort`	61	18	440433	40002	9.1%
`queens`	66	16	118224	6168	5.2%
`life`	169	214	989430	104970	10.6%
`kb`	557	604	2747397	235458	8.6%
`nucleic`	3230	19478	1616487	293808	18.2%

그림 6: 실험 결과.

다 (idata). 현재 메모리 환경 Δ에서 하나를 골라 반납할 수 있는 조건을 찾아 낸다. $\text{wcfree}(D, \mathcal{C})$는 $\mathcal{C}$ 조건 아래 D를 반납할 수 있는 조건(flag)을 찾아내는 것으로, 함수의 인자가 내부적으로 공유되어 있을 때(β_s)와 아닐 때($\neg\beta_s$)를 고려하여 조건을 찾아낸다. 여기서 ;은 차례로 수행한다는 뜻이다.

함수에 반납 명령어를 삽입할 때는 (ifun) 전략 (3)에 의해 결과가 나중에 사용된다고 조건에 넣는다. 또한 인자는 전략 (1)에 의해 β가 참이 아닐 때 나중에 사용된다고 조건에 넣는다. 함수 호출시에는 (iapp) 전략 (1)에 의해 두 가지 플래그를 계산한다. p는 호출된 함수 내에서 인자를 반납해도 되는가이고, q는 인자가 내부적으로 공유되어 있는가이다.

6 안전성

원래 프로그램이 제대로 수행되었다면, 제시한 알고리즘으로 반납 명령어를 삽입하고 나서도 제대로 수행된다.

정리 1 (안전성) *타입이 맞는 프로그램 $\wp_0$에 대해, $(\wp_0, \emptyset, \epsilon) \to (v, H, \epsilon)$이고, $\wp_0$에 메모리 타입 유추 알고리즘ㅢ 결과를 붙인 것이 $\wp_1$이고, 삽입 알고리즘에 ㅢ해 $\emptyset \rhd \wp_1 \Rightarrow \wp_2 : \mathcal{C}$이던, $(\wp_2, \emptyset, \epsilon) \to (v, H', \epsilon)$이다.*

7 실험 결과 및 결론

벤치마크 프로그램에 재사용으로 사용될 수 있는 반납만 삽입하여 5.2-98.7%의 재사용 비율을 얻었다 (그림 6). 삽입의 전략은 본 논문에서 제시한 것과 동일하나 플래그를 좀 더 세분화하였다.

재귀적 자료구조에 대해 안전한 반납 명령어를 삽입하는 알고리즘을 제시하였고, 향후 성능의 향상과 언어의 확장이 필요하다. 실험을 통해 `sieve` 프로그램의 경우 85.7%의 안전한 재사용 비율이 가능함을 확인하였는데 본 논문에서 제시한 방법으로는 여기에 미치지 못한다. 좀 더 세분화한 이름 붙이기가 필요하다. 또한 여러 연구결과들[2, 3, 4, 5]을 이용해 힙 셀의 내용을 바꾸는 명령어(destructive update)도 처리해야만 실제 컴파일러에 적용될 수 있을 것이다.

감사의 글. 본 연구 내용에 조언을 아끼지 않으시고 많은 도움을 주신 이광근 박사님께 깊이 감사드립니다.

참고 자료

[1] 이광근, 이욱세, 어현준, 김정택, 최웅식, 류석영, 강현구, 서선애, 장성순, 김범식. nML 컴파일러 시스템 (status report). *정보과학ㅢ 가을 학술ㅢㅢ*, 28(2):340-342, 2001년 10월. `http://ropas.kaist.ac.kr/n`

[2] M. Sagiv, T. Reps, and R. Wilhelm. Solving shape-analysis problems in languages with destructive updating. *ACM Transactions on Programming Languages and Systems*, 20(1):1–50, January 1998.

[3] Samin Ishtiaq and Peter O'Hearn. BI as an assertion language for mutable data structures. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, January 2001.

[4] John C. Reynolds. Separation logic: a logic for shared mutable data structures. In *Proceedings of the Annual IEEE Symposium on Logic in Computer Science*, July 2002.

[5] David Walker and Greg Morrisett. Alias types for recursive data structures. In *Workshop on Types in Compilation*, volume 2071 of *Lecture Notes in Computer Science*, pages 177–206, September 2000.