CS738: Advanced Compiler Optimizations

Liveness based Garbage Collection

Amey Karkare
karkare@cse.iitk.ac.in

http://www.cse.iitk.ac.in/~karkare/cs738
Department of CSE, IIT Kanpur

Ideal Garbage Collection

. . . garbage collection (GC) is a form of automatic memory management. The garbage collector, or just collector, attempts to reclaim garbage, or memory occupied by objects that are no longer in use by the program. . . .

From Wikipedia

https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)

Real Garbage Collection

. . . All garbage collectors use some efficient approximation to liveness. In tracing garbage collection, the approximation is that an object can't be live unless it is reachable. . . .

From Memory Management Glossary

www.memorymanagement.org/glossary/g.html#term-garbage-collection

Liveness based GC

- During execution, there are significant amounts of heap allocated data that are *reachable but not live.*
  - Current GCs will retain such data.
- Our idea:
  - We do a liveness analysis of *heap data* and provide GC with its result.
  - Modify GC to mark data for retention *only if it is live.*
- Consequences:
  - Fewer cells marked. More garbage collected per collection. Fewer garbage collections.
  - Programs expected to run faster and with smaller heap.

Demand

- Demand (notation: σ) is a description of intended use of the result of an expression.
- We assume the demand on the main expression to be $(0+1)^*$, which we call σ_{all}.
- The demands on each function body, σ_f, have to be computed.

Liveness analysis – The big picture

π_{main}: (**let** z ← . . . **in**
(**let** y ← . . . **in**
π_9: (**let** w ← (**append** y z) **in**
π_{10}: (**let** a ← (**cdr** w) **in**
π_{11}: (**let** b ← (**car** a) **in**
π_{12}: (**return** b)))))))

(**define** (**append** l1 l2)
π_1: (**let** test ← (**null?** l1) **in**
π_2: (**if** test π_3:(**return** l2)
π_4: (**let** tl ← (**cdr** l1) **in**
π_5: (**let** rec ← (**append** tl l2) **in**
π_6: (**let** hd ← (**car** l1) **in**
π_7: (**let** ans ← (**cons** hd rec) **in**
π_8: (**return** ans))))))))

Liveness analysis – The big picture

π_{main}: (**let** z ← . . . **in**
(**let** y ← . . . **in**
π_9: (**let** w ← (**append** y z) **in**
π_{10}: (**let** a ← (**cdr** w) **in**
π_{11}: (**let** b ← (**car** a) **in**
π_{12}: (**return** b)))))))

(**define** (**append** l1 l2)
π_1: (**let** test ← (**null?** l1) **in**
π_2: (**if** test π_3:(**return** l2)
π_4: (**let** tl ← (**cdr** l1) **in**
π_5: (**let** rec ← (**append** tl l2) **in**
π_6: (**let** hd ← (**car** l1) **in**
π_7: (**let** ans ← (**cons** hd rec) **in**
π_8: (**return** ans))))))))

Liveness environments:

$L_1 = \ldots$
$L_2 = \ldots$
$\ldots$
$L_9 = \ldots$
$L_{10} = \ldots$

Liveness analysis

- **GOAL:** Compute Liveness Environment at various program points, statically.

$\mathcal{L}app(a, \sigma)$ – Liveness environment generated by an *application* a, given a demand σ.

$\mathcal{L}exp(e, \sigma)$ – Liveness environment before an *expression* e, given a demand σ.

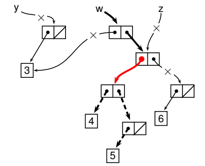
The language analyzed

- First order eager Scheme-like functional language.
- In Administrative Normal Form (ANF).

$$
\begin{array}{lcl}
p \in Prog & ::= & d_1 \ldots d_n \; e_{main} \\
d \in Fdef & ::= & (\textbf{define}\ (f\ x_1\ \ldots\ x_n)\ e) \\
e \in Expr & ::= & \left\{\begin{array}{l}(\textbf{if}\ x\ e_1\ e_2) \\ (\textbf{let}\ x \leftarrow a\ \textbf{in}\ e) \\ (\textbf{return}\ x)\end{array}\right. \\
a \in App & ::= & \left\{\begin{array}{ll}k & \\ (\textbf{cons}\ x_1\ x_2) & \\ (\textbf{car}\ x) & (\textbf{cdr}\ x) \\ (\textbf{null?}\ x) & (+\ x_1\ x_2) \\ (f\ x_1\ \ldots\ x_n) & \end{array}\right.
\end{array}
$$

An Example

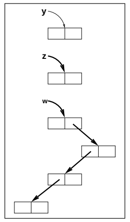
```
(define (append l1 l2)
 (if (null? l1) l2
  (cons (car l1)
        (append (cdr l1) l2))))

(let z ←(cons (cons 4 (cons 5 nil))
         (cons 6 nil)) in
(let y ← (cons 3 nil) in
 (let w ← (append y z) in
     π:(car (cdr w)))))
```



- *Though all cells are reachable at π, a liveness-based GC will retain only the cells pointed by thick arrows.*

Liveness – Basic Concepts and Notations

- *Access paths:* Strings over {**0**, **1**}.
  0 – access **car** field
  1 – access **cdr** field
- Denote traversals over the heap graph
- *Liveness environment:* Maps root variables to set of access paths.



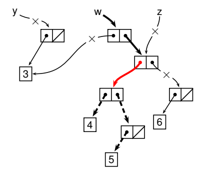
$$
\mathsf{L}_i \;:\; \left\{\begin{array}{l} y \mapsto \emptyset \\ z \mapsto \{\epsilon\} \\ w \mapsto \{\epsilon, 1, 10, 100\}\end{array}\right.
$$

Alternate representation.

$$
\mathsf{L}_i \;:\; \left\{\begin{array}{l} \emptyset \;\cup \\ \{z.\epsilon\} \;\cup \\ \{w.\epsilon, w.1, w.10, w.100\}\end{array}\right.
$$

Demand

(**car** (**cdr** w))



- Demand (notation: σ) is a description of intended use of the result of an expression.

Liveness analysis of Expressions

$\mathcal{L}exp((\textbf{return}\ x), \sigma) = \{x.\sigma\}$

$\mathcal{L}exp((\textbf{if}\ x\ \ e_1\ \ e_2), \sigma) = \{x.\epsilon\} \cup \mathcal{L}exp(e_1, \sigma) \cup \mathcal{L}exp(e_2, \sigma)$

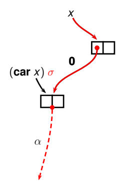
$\mathcal{L}exp((\textbf{let}\ x\ \leftarrow\ s\ \textbf{in}\ e), \sigma) = \mathsf{L} \setminus \{x.*\} \cup \mathcal{L}app(s, \mathsf{L}(x))$
where $\mathsf{L} = \mathcal{L}exp(e, \sigma)$

Notice the similarity with:

$$live_{in}(B) = live_{out}(B) \setminus kill(B) \cup gen(B)$$

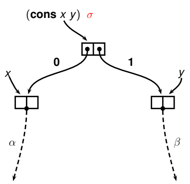
in classical dataflow analysis for imperative languages.

Liveness analysis of Primitive Applications



$\mathcal{L}app((\textbf{car}\ x), \sigma) = \{x.\epsilon,\ x.\mathbf{0}\sigma\}$

Liveness analysis of Primitive Applications

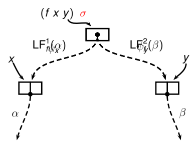


$$
\mathcal{L}app((\textbf{cons}\ x\ y), \sigma) \;=\; \{x.\alpha \mid \mathbf{0}\alpha \in \sigma\} \;\cup\; \{y.\beta \mid \mathbf{1}\beta \in \sigma\}
$$

- $\bar{\mathbf{0}}$ – Removal of a leading **0**
  $\bar{\mathbf{1}}$ – Removal of a leading **1**

$$\mathcal{L}app((\textbf{cons}\ x\ y), \sigma) \;=\; x.\bar{\mathbf{0}}\sigma \cup y.\bar{\mathbf{1}}\sigma$$

Liveness Analysis of Function Applications



$\mathcal{L}app((f\ x\ y), \sigma) = x.\overline{\psi_x}\sigma \cup y.\overline{\psi_y}\sigma$

- We use LF_f: context independent summary of *f*.
- To find $\mathsf{LF}_f^i(\ldots)$:
  - Assume a symbolic demand σ_{sym}.
  - Let e_f be the body of *f*.
  - Set $\mathsf{LF}_f^i(\sigma_{sym})$ to $\mathcal{L}exp(e_f, \sigma_{sym})(x_i)$.
  - How to handle recursive calls? Use LF_f with appropriate demand !!

Liveness analysis – The big picture

π_{main}: (**let** z ← … **in**
 (**let** y ← … **in**
 π_9: (**let** w ← (**append** y z) **in**
 π_{10}: (**let** a ← (**cdr** w) **in**
 π_{11}: (**let** b ← (**car** a) **in**
 π_{12}: (**return** b)))))))

(**define** (**append** l1 l2)
 π_1: (**let** test ← (**null?** l1) **in**
 π_2: (**if** test π_3:(**return** l2)
 π_4: (**let** tl ← (**cdr** l1) **in**
 π_5: (**let** rec ← (**append** tl l2) **in**
 π_6: (**let** hd ← (**car** l1) **in** $\mathsf{LF}^2_{append}(\bar{1}\sigma)$
 π_7: (**let** ans ← (**cons** hd rec) **in** $\bar{1}\sigma$
 π_8: (**return** ans)))))))

Liveness environments:

$\mathsf{L}_1^{l1} = \{\epsilon\} \cup \mathbf{0\bar{0}}\sigma_{append} \cup \mathbf{1}\mathsf{LF}^1_{append}(\bar{1}\sigma_{append})$

$\mathsf{L}_1^{l2} = \sigma \cup \mathsf{LF}^2_{append}(\bar{1}\sigma_{append})$

…

$\mathsf{L}_9^{y} = \mathsf{LF}^1_{append}(\{\epsilon, \mathbf{1}\} \cup \mathbf{10}\sigma_{all})$

Demand summaries:

Function summaries:

$\mathsf{LF}^1_{append}(\sigma) = \{\epsilon\} \cup \mathbf{0\bar{0}}\sigma \cup \mathbf{1}\mathsf{LF}^1_{append}(\bar{1}\sigma)$

$\mathsf{LF}^2_{append}(\sigma) = \sigma \cup \mathsf{LF}^2_{append}(\bar{1}\sigma)$

Liveness analysis – Demand Summary

$\sigma_{main} = \sigma_{all}$ σ_1

$\sigma_{append} = \sigma_1 \cup \ldots \sigma_2$ σ_2

π_{main}: (**let** z ← … **in**
 (**let** y ← … **in**
 π_9: (**let** w ← (**append** y z) **in**
 π_{10}: (**let** a ← (**cdr** w) **in**
 π_{11}: (**let** b ← (**car** a) **in**
 π_{12}: (**return** b)))))))

(**define** (**append** l1 l2)
 π_1: (**let** test ← (**null?** l1) **in**
 π_2: (**if** test π_3:(**return** l2)
 π_4: (**let** tl ← (**cdr** l1) **in**
 π_5: (**let** rec ← (**append** tl l2) **in**
 π_6: (**let** hd ← (**car** l1) **in**
 π_7: (**let** ans ← (**cons** hd rec) **in**
 π_8: (**return** ans)))))))

Liveness environments:

$\mathsf{L}_1^{l1} = \{\epsilon\} \cup \mathbf{0\bar{0}}\sigma_{append} \cup \mathbf{1}\mathsf{LF}^1_{append}(\bar{1}\sigma_{append})$

$\mathsf{L}_1^{l2} = \sigma \cup \mathsf{LF}^2_{append}(\bar{1}\sigma_{append})$

…

$\mathsf{L}_9^{y} = \mathsf{LF}^1_{append}(\{\epsilon, \mathbf{1}\} \cup \mathbf{10}\sigma_{all})$

Demand summaries:

$\sigma_{main} = \sigma_{all}$

$\sigma_{append} = \{\epsilon, \mathbf{1}\} \cup \mathbf{10}\sigma_{all} \cup \bar{1}\sigma_{append}$

Function summaries:

$\mathsf{LF}^1_{append}(\sigma) = \{\epsilon\} \cup \mathbf{0\bar{0}}\sigma \cup \mathbf{1}\mathsf{LF}^1_{append}(\bar{1}\sigma)$

$\mathsf{LF}^2_{append}(\sigma) = \sigma \cup \mathsf{LF}^2_{append}(\bar{1}\sigma)$

Obtaining a closed form solution for LF

- Function summaries will always have the form:

$$\mathsf{LF}^i_f(\sigma) = \mathsf{I}^i_f \cup \mathsf{D}^i_f\sigma$$

- Consider the equation for LF^1_{append}

$$\mathsf{LF}^1_{append}(\sigma) = \{\epsilon\} \cup \mathbf{0\bar{0}}\sigma \cup \mathbf{1}\mathsf{LF}^1_{append}(\bar{1}\sigma)$$

- Substitute the assumed form in the equation:

$$\mathsf{I}^1_{append} \cup \mathsf{D}^1_{append}\sigma = \{\epsilon\} \ \cup \ \mathbf{0\bar{0}}\sigma \cup \mathbf{1}(\mathsf{I}^1_{append} \cup \mathsf{D}^1_{append}\bar{1}\sigma)$$

- Equating the terms without and with σ, we get:

$$\mathsf{I}^1_{append} = \{\epsilon\} \ \cup \ \mathbf{1}\mathsf{I}^1_{append}$$
$$\mathsf{D}^1_{append} = \mathbf{0\bar{0}} \cup \mathbf{1}\mathsf{D}^1_{append}\bar{\mathbf{1}}$$

Summary of Analysis Results

Liveness at program points:

$\mathsf{L}_1^{l1} = \{\epsilon\} \cup \mathbf{0\bar{0}}\sigma \cup \mathbf{1}(\mathsf{I}^1_{append} \cup \mathsf{D}^1_{append}\bar{1}\sigma_{append})$

$\mathsf{L}_1^{l2} = \{\epsilon\} \cup \mathsf{I}^2_{append} \cup \mathsf{D}^2_{append}\bar{1}\sigma_{append}$

$\mathsf{L}_5^{l1} = \{\epsilon\} \cup \mathbf{0\bar{0}}\sigma_{append}$

$\mathsf{L}_5^{tl} = \mathsf{I}^1_{append} \cup \mathsf{D}^1_{append}\bar{1}\sigma_{append}$

$\mathsf{L}_5^{l2} = \mathsf{I}^2_{append} \cup \mathsf{D}^2_{append}\bar{1}\sigma_{append}$

…

Demand summaries:

$\sigma_{append} = \{\epsilon, \ \mathbf{1}\} \cup \bar{1}\sigma_{append} \cup \mathbf{10}\sigma_{all}$

Function summaries:

$\mathsf{I}^1_{append} = \{\epsilon\} \ \cup \ \mathbf{1}\mathsf{I}^1_{append}$

$\mathsf{D}^1_{append} = \mathbf{0\bar{0}} \cup \mathbf{1}\mathsf{D}^1_{append}\bar{\mathbf{1}}$

$\mathsf{I}^2_{append} = \mathsf{I}^2_{append}$

$\mathsf{D}^2_{append} = \{\epsilon\} \cup \mathsf{D}^2_{append}\bar{\mathbf{0}}$

Practical $\bar{0}/\bar{1}$ simplification

- The simplification is possible to do on a finite state automaton.
- Over-approximate the CFG by an automaton (Mohri-Nederhoff transformation).
- Perform **0/1** removal on the automaton.

Example

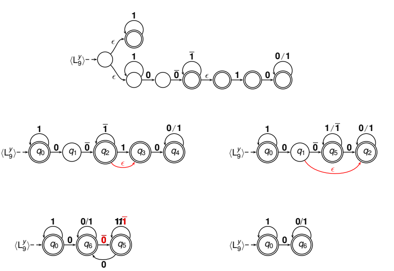
Grammar for L_9^{y}

$$\mathsf{L}_9^y \rightarrow \mathsf{I}^1_{append} \mid \mathsf{D}^1_{append}(\epsilon \mid \mathbf{1} \mid \mathbf{10}\sigma_{all})$$
$$\mathsf{I}^1_{append} \rightarrow \epsilon \mid \mathbf{1}\mathsf{I}^1_{append}$$
$$\mathsf{D}^1_{append} \rightarrow \mathbf{0\bar{0}} \mid \mathbf{1}\mathsf{D}^1_{append}\bar{\mathbf{1}}$$
$$\sigma_{all} \rightarrow \epsilon \mid \mathbf{0}\sigma_{all} \mid \mathbf{1}\sigma_{all}$$

After Mohri-Nederhoff transformation

$$\mathsf{L}_9^y \rightarrow \mathsf{I}^1_{append} \mid \mathsf{D}^1_{append}(\epsilon \mid \mathbf{1} \mid \mathbf{10}\sigma_{all})$$
$$\mathsf{I}^1_{append} \rightarrow \epsilon \mid \mathbf{1}\mathsf{I}^1_{append}$$
$$\mathsf{D}^1_{append} \rightarrow \mathbf{0\bar{0}}\widehat{\mathsf{D}}^1_{append} \mid \mathbf{1}\mathsf{D}^1_{append}$$
$$\widehat{\mathsf{D}}^1_{append} \rightarrow \bar{\mathbf{1}}\widehat{\mathsf{D}}^1_{append} \mid \epsilon$$
$$\sigma_{all} \rightarrow \epsilon \mid \mathbf{0}\sigma_{all} \mid \mathbf{1}\sigma_{all}$$

Automaton for L_9^y



Experimental Setup

- Built a prototype consisting of:
  - An ANF-scheme interpreter
  - Liveness analyzer
  - A single-generation copying collector.
- The collector optionally uses liveness
  - Marks a link during GC only if it is live.
- Benchmark programs are mostly from the no-fib suite.

Solution of the equations

View the equations as grammar rules:

$$\mathsf{L}_1^{l1} \rightarrow \epsilon \mid \mathbf{0\bar{0}}\sigma \mid \mathbf{1}(\mathsf{I}^1_{append} \mid \mathsf{D}^1_{append}\bar{1}\sigma_{append})$$
$$\mathsf{I}^1_{append} \rightarrow \epsilon \mid \mathbf{1}\mathsf{I}^1_{append}$$
$$\mathsf{D}^1_{append} \rightarrow \mathbf{0\bar{0}} \mid \mathbf{1}\mathsf{D}^1_{append}\bar{\mathbf{1}}$$

The solution of L_1^{l1} is the language $\mathscr{L}(\mathsf{L}_1^{l1})$ generated by it.

Working of Liveness-based GC (Mark phase)

- GC invoked at a program point π
- GC traverses a path α starting from a root variable x.
- GC consults L^x_π:
  - Does $\alpha \in \mathscr{L}(\mathsf{L}^x_\pi)$?
  - If yes, then mark the current cell
- Note that α is a *forward*-only access path
  - consisting only of edges **0** and **1**, but not $\bar{\mathbf{0}}$ or $\bar{\mathbf{1}}$
  - But $\mathscr{L}(\mathsf{L}^x_\pi)$ has access paths marked with $\bar{\mathbf{0}}/\bar{\mathbf{1}}$ for **0/1** removal arising from the **cons** rule.

$\bar{0}/\bar{1}$ handling

- **0** removal from a set of access paths:

$$\alpha_1\mathbf{0\bar{0}}\alpha_2 \hookrightarrow \alpha_1\alpha_2$$
$$\alpha_1\bar{\mathbf{0}}\mathbf{1}\alpha_2 \hookrightarrow \text{ drop } \alpha_1\bar{\mathbf{0}}\mathbf{1}\alpha_2 \text{ from the set}$$

- **1** removal from a set of access paths:

$$\alpha_1\mathbf{1\bar{1}}\alpha_2 \hookrightarrow \alpha_1\alpha_2$$
$$\alpha_1\bar{\mathbf{1}}\mathbf{0}\alpha_2 \hookrightarrow \text{ drop } \alpha_1\bar{\mathbf{1}}\mathbf{0}\alpha_2 \text{ from the set}$$

GC decision problem

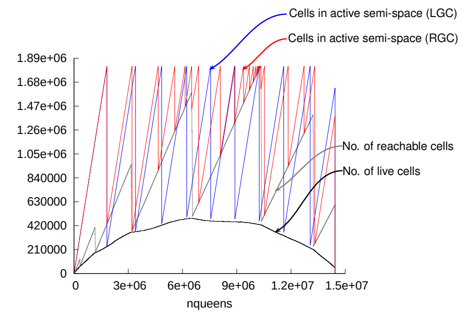
- Deciding the membership in a CFG augmented with a fixed set of unrestricted productions.

$$\mathbf{0\bar{0}} \rightarrow \epsilon$$
$$\mathbf{1\bar{1}} \rightarrow \epsilon$$

- The problem shown to be undecidable[1].
  - Reduction from Halting problem.

[1] Prasanna, Sanyal, and Karkare. *Liveness-Based Garbage Collection for Lazy Languages*, ISMM 2016.

GC behavior as a graph



Results as Tables

Analysis Performance:

Program	sudoku	lcss	gc_bench	knightstour	treejoin	nqueens	lambda
Time (msec)	120.95	2.19	0.32	3.05	2.61	0.71	20.51
DFA size	4251	726	258	922	737	241	732
Precision(%)	**87.5**	**98.8**	**99.9**	**94.3**	**99.6**	**98.8**	**83.8**

Garbage collection performance

Program	# Collected cells per GC RGC	# Collected cells per GC LGC	#GCs RGC	#GCs LGC	MinHeap (#cells) RGC	MinHeap (#cells) LGC	GC time (sec) RGC	GC time (sec) LGC
sudoku	**490**	**1306**	22	9	1704	589	.028	.122
lcss	**46522**	**51101**	8	7	52301	1701	.045	.144
gc_bench	**129179**	**131067**	9	9	131071	6	.086	.075
nperm	**47586**	**174478**	14	4	202597	37507	1.406	.9
fibheap	**249502**	**251525**	1	1	254520	13558	.006	.014
knightstour	**2593**	**314564**	1161	10	508225	307092	464.902	14.124
treejoin	**288666**	**519943**	2	1	525488	7150	.356	.217

Lazy evaluation

- An evaluation strategy in which evaluation of an expression is postponed until its value is needed
  - Binding of a variable to an expression **does not force evaluation** of the expression
- Every expression is evaluated at most once

Laziness: Example

(**define** (**length** l)
 (**if** (**null?** l)
 return 0
 return (+ 1 (**length** (**cdr** l)))))

(**define** (main)
 (**let** a ← (a BIG closure) **in**
 (**let** b ← (+ a **1**) **in**
 (**let** c ← (**cons** b **nil**) **in**
 (**let** w ← (**length** c) **in**
 (**return** w))))))

Handling lazy semantics: Challenges

- Laziness complicates liveness analysis itself.
  - Data is made live by evaluation of closures
  - In lazy languages, the place in the program where this evaluation takes place cannot be statically determined
- Liveness-based garbage collector significantly more complicated than that for an eager language.
  - Need to track liveness of closures
  - But a closure can escape the scope in which it was created
  - Solution: carry the liveness information in the closure itself
  - For precision: need to update the liveness information as execution progresses

Handling possible non-evaluation

- Liveness no longer remains independent of demand σ
  - If (**car** x) is not evaluated at all, it does not generate any liveness for x
- Require a new terminal **2** with following semantics

$$\mathbf{2}\sigma \hookrightarrow \begin{cases} \emptyset & \text{if } \sigma = \emptyset \\ \{\epsilon\} & \text{otherwise} \end{cases}$$

$$\mathcal{L}app((\mathbf{car}\ \mathrm{x}), \sigma) = \mathrm{x}.\{\mathbf{2}, \mathbf{0}\}\sigma$$

Scope for future work

- Reducing GC-time.
  - Reducing re-visits to heap nodes.
  - Basing the implementation on full Scheme, not ANF-Scheme
- Increasing the scope of the method.
  - Lazy languages. (ISMM 2016)
  - Higher order functions.
    - Specialize all higher order functions (Firstification)
    - Analysis on the firstified program
    - For partial applications, carry information about the *base* function
- Using the notion of *demand* for other analysis.
  - Program Slicing (Under Review as of September 2016)
  - Strictness Analysis
    - All path problem, requires doing intersection of demands
    - $\Rightarrow$ intersection of CFGs $\Rightarrow$ under-approximation

Conclusions

- Proposed a liveness-based GC scheme.
- Not covered in this talk:
  - The soundness of liveness analysis.
  - Details of undecidability proof.
  - Details of handling lazy languages.
- A prototype implementation to demonstrate:
  - the precision of the analysis.
  - reduced heap requirement.
  - reduced GC time for a majority of programs.
- Unfinished agenda:
  - Improving GC time for a larger fraction of programs.
  - Extending scope of the method.