

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_\(computer_science\)](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.

The language analyzed

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

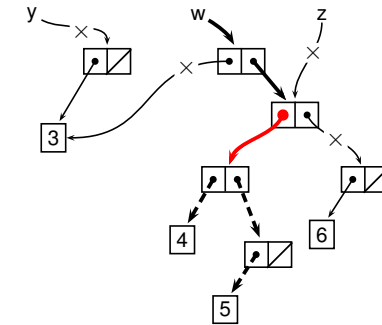
$$\begin{aligned}
 p \in Prog &::= d_1 \dots d_n e_{\text{main}} \\
 d \in Fdef &::= (\text{define } (f \ x_1 \dots x_n) \ e) \\
 e \in Expr &::= \begin{cases} (\text{if } x \ e_1 \ e_2) \\ (\text{let } x \leftarrow a \text{ in } e) \\ (\text{return } x) \end{cases} \\
 a \in App &::= \begin{cases} k \\ (\text{cons } x_1 \ x_2) & (\text{cdr } x) \\ (\text{car } x) & (+ \ x_1 \ x_2) \\ (\text{null? } x) & (f \ x_1 \dots x_n) \end{cases}
 \end{aligned}$$

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
       $\pi$ :(car (cdr w)))))
  
```



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

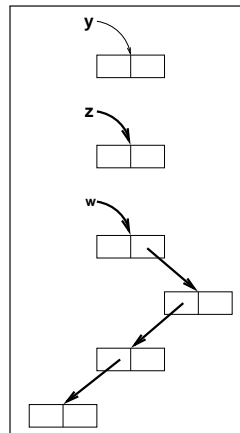
Liveness – Basic Concepts and Notations

- 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.

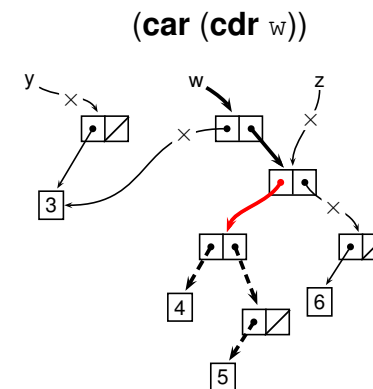
$$L_i : \begin{cases} y \mapsto \emptyset \\ z \mapsto \{\epsilon\} \\ w \mapsto \{\epsilon, 1, 10, 100\} \end{cases}$$

Alternate representation.

$$L_i : \begin{cases} \emptyset \cup \\ \{z.\epsilon\} \cup \\ \{w.\epsilon, w.1, w.10, w.100\} \end{cases}$$



Demand



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

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

```
 $\pi_{main}$ : (let z  $\leftarrow$  ... in
  (let y  $\leftarrow$  ... in
     $\pi_9$ : (let w  $\leftarrow$  (append y z) in
       $\pi_{10}$ : (let a  $\leftarrow$  (cdr w) in
         $\pi_{11}$ : (let b  $\leftarrow$  (car a) in
           $\pi_{12}$ : (return b))))))
  (define (append l1 l2)
     $\pi_1$ : (let test  $\leftarrow$  (null? l1) in
       $\pi_2$ : (if test  $\pi_3$ : (return l2)
         $\pi_4$ : (let t1  $\leftarrow$  (cdr l1) in
           $\pi_5$ : (let rec  $\leftarrow$  (append t1 l2) in
             $\pi_6$ : (let hd  $\leftarrow$  (car l1) in
               $\pi_7$ : (let ans  $\leftarrow$  (cons hd rec) in
                 $\pi_8$ : (return ans)))))))))
```

Liveness analysis – The big picture

```
 $\pi_{main}$ : (let z  $\leftarrow$  ... in
  (let y  $\leftarrow$  ... in
     $\pi_9$ : (let w  $\leftarrow$  (append y z) in
       $\pi_{10}$ : (let a  $\leftarrow$  (cdr w) in
         $\pi_{11}$ : (let b  $\leftarrow$  (car a) in
           $\pi_{12}$ : (return b))))))
  (define (append l1 l2)
     $\pi_1$ : (let test  $\leftarrow$  (null? l1) in
       $\pi_2$ : (if test  $\pi_3$ : (return l2)
         $\pi_4$ : (let t1  $\leftarrow$  (cdr l1) in
           $\pi_5$ : (let rec  $\leftarrow$  (append t1 l2) in
             $\pi_6$ : (let hd  $\leftarrow$  (car l1) in
               $\pi_7$ : (let ans  $\leftarrow$  (cons hd rec) in
                 $\pi_8$ : (return ans)))))))))
```

Liveness environments:

```
L1  = ...
L2  = ...
...
L9  = ...
L10 = ...
```

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 .

Liveness analysis of Expressions

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

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

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

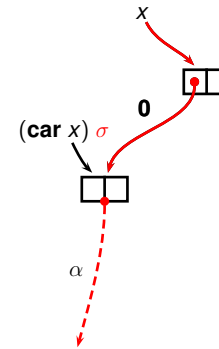
where $L = \mathcal{L}exp(e, \sigma)$

Notice the similarity with:

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

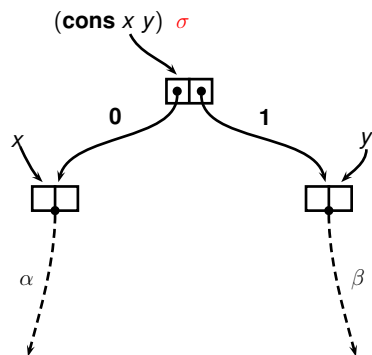
in classical dataflow analysis for imperative languages.

Liveness analysis of Primitive Applications



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

Liveness analysis of Primitive Applications

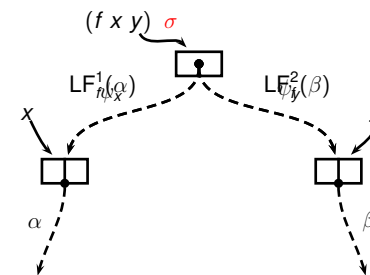


$$\mathcal{L}app((\text{cons } x \text{ } y), \sigma) = \{x.\alpha \mid 0\alpha \in \sigma\} \cup \{y.\beta \mid 1\beta \in \sigma\}$$

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

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

Liveness Analysis of Function Applications



$$\mathcal{L}app((f \ x \ y), \sigma) = x.\bar{\psi}_x\sigma \cup y.\bar{\psi}_y\sigma$$

- We use LF_f : context independent summary of f .
- To find $LF_f^i(\dots)$:
 - Assume a symbolic demand σ_{sym} .
 - Let e_f be the body of f .
 - Set $LF_f^i(\sigma_{sym})$ 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

```

πmain: (let z ← ... in
  (let y ← ... in
    π9: (let w ← (append y z) in
      π10: (let a ← (cdr w) in
        π11: (let b ← (car a) in
          π12: (return b))))))

(define (append l1 l2)
  π1: (let test ← (null? l1) in
    π2: (if test π3: (return l2)
    π4: (let t1 ← (cdr l1) in
      π5: (let rec ← (append t1 l2) in
        π6: (let hd ← (car l1) in
          π7: (let ans ← (cons hd rec) in
            π8: (return ans))))))

```

LF²_{append}($\bar{1}\sigma$)

Liveness environments:

$$\begin{aligned}
 L_1^{11} &= \{\epsilon\} \cup 00\bar{\sigma}_{\text{append}} \cup 1LF_{\text{append}}^1(\bar{1}\sigma_{\text{append}}) \\
 L_1^{12} &= \sigma \cup LF_{\text{append}}^2(\bar{1}\sigma_{\text{append}}) \\
 \vdots \\
 L_9^{\bar{y}} &= LF_{\text{append}}^1(\{\epsilon, 1\} \cup 10\sigma_{\text{all}})
 \end{aligned}$$

Demand summaries:

Function summaries:

$$\begin{aligned}
 LF_{\text{append}}^1(\sigma) &= \{\epsilon\} \cup 00\bar{\sigma} \cup 1LF_{\text{append}}^1(\bar{1}\sigma) \\
 LF_{\text{append}}^2(\sigma) &= \sigma \cup LF_{\text{append}}^2(\bar{1}\sigma)
 \end{aligned}$$

Liveness analysis – Demand Summary

```

πmain: (let z ← ... in
  (let y ← ... in
    π9: (let w ← (append y z) in
      π10: (let a ← (cdr w) in
        π11: (let b ← (car a) in
          π12: (return b))))))

(define (append l1 l2)
  π1: (let test ← (null? l1) in
    π2: (if test π3: (return l2)
    π4: (let t1 ← (cdr l1) in
      π5: (let rec ← (append t1 l2) in
        π6: (let hd ← (car l1) in
          π7: (let ans ← (cons hd rec) in
            π8: (return ans))))))

```

σ_{main} = σ_{all} σ_{append} = σ₁ ∪ ... σ₂

Liveness environments:

$$\begin{aligned}
 L_1^{11} &= \{\epsilon\} \cup 00\bar{\sigma}_{\text{append}} \cup 1LF_{\text{append}}^1(\bar{1}\sigma_{\text{append}}) \\
 L_1^{12} &= \sigma \cup LF_{\text{append}}^2(\bar{1}\sigma_{\text{append}}) \\
 \vdots \\
 L_9^{\bar{y}} &= LF_{\text{append}}^1(\{\epsilon, 1\} \cup 10\sigma_{\text{all}})
 \end{aligned}$$

Demand summaries:

$$\begin{aligned}
 \sigma_{\text{main}} &= \sigma_{\text{all}} \\
 \sigma_{\text{append}} &= \{\epsilon, 1\} \cup 10\sigma_{\text{all}} \cup \bar{1}\sigma_{\text{append}}
 \end{aligned}$$

Function summaries:

$$\begin{aligned}
 LF_{\text{append}}^1(\sigma) &= \{\epsilon\} \cup 00\bar{\sigma} \cup 1LF_{\text{append}}^1(\bar{1}\sigma) \\
 LF_{\text{append}}^2(\sigma) &= \sigma \cup LF_{\text{append}}^2(\bar{1}\sigma)
 \end{aligned}$$

Obtaining a closed form solution for LF

- Function summaries will always have the form:

$$LF_f^i(\sigma) = I_f^i \cup D_f^i \sigma$$

- Consider the equation for LF_{append}^1

$$LF_{\text{append}}^1(\sigma) = \{\epsilon\} \cup 00\bar{\sigma} \cup 1LF_{\text{append}}^1(\bar{1}\sigma)$$

- Substitute the assumed form in the equation:

$$I_{\text{append}}^1 \cup D_{\text{append}}^1 \sigma = \{\epsilon\} \cup 00\bar{\sigma} \cup 1(I_{\text{append}}^1 \cup D_{\text{append}}^1 \bar{1}\sigma)$$

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

$$\begin{aligned}
 I_{\text{append}}^1 &= \{\epsilon\} \cup 1I_{\text{append}}^1 \\
 D_{\text{append}}^1 &= 00 \cup 1D_{\text{append}}^1 \bar{1}
 \end{aligned}$$

Summary of Analysis Results

Liveness at program points:

$$\begin{aligned}
 L_1^{11} &= \{\epsilon\} \cup 00\bar{\sigma} \cup 1(I_{\text{append}}^1 \cup D_{\text{append}}^1 \bar{1}\sigma_{\text{append}}) \\
 L_1^{12} &= \{\epsilon\} \cup I_{\text{append}}^2 \cup D_{\text{append}}^2 \bar{1}\sigma_{\text{append}} \\
 L_5^{11} &= \{\epsilon\} \cup 00\bar{\sigma}_{\text{append}} \\
 L_5^{12} &= I_{\text{append}}^2 \cup D_{\text{append}}^2 \bar{1}\sigma_{\text{append}} \\
 \dots
 \end{aligned}$$

Demand summaries:

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

Function summaries:

$$\begin{aligned}
 I_{\text{append}}^1 &= \{\epsilon\} \cup 1I_{\text{append}}^1 \\
 D_{\text{append}}^1 &= 00 \cup 1D_{\text{append}}^1 \bar{1} \\
 I_{\text{append}}^2 &= I_{\text{append}}^2 \\
 D_{\text{append}}^2 &= \{\epsilon\} \cup D_{\text{append}}^2 \bar{0}
 \end{aligned}$$

Solution of the equations

View the equations as grammar rules:

$$\begin{aligned} L_1^{11} &\rightarrow \epsilon \mid \mathbf{00}\sigma \mid \mathbf{1}(l_{\text{append}}^1 \mid D_{\text{append}}^1 \bar{\mathbf{1}}\sigma_{\text{append}}) \\ l_{\text{append}}^1 &\rightarrow \epsilon \mid \mathbf{1}l_{\text{append}}^1 \\ D_{\text{append}}^1 &\rightarrow \mathbf{00} \mid \mathbf{1}D_{\text{append}}^1 \bar{\mathbf{1}} \end{aligned}$$

The solution of L_1^{11} is the language $\mathcal{L}(L_1^{11})$ 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_π^x :
 - ▶ Does $\alpha \in \mathcal{L}(L_\pi^x)$?
 - ▶ If yes, then mark the current cell
- ▶ Note that α is a *forward-only* access path
 - ▶ consisting only of edges $\mathbf{0}$ and $\mathbf{1}$, but not $\bar{\mathbf{0}}$ or $\bar{\mathbf{1}}$
 - ▶ But $\mathcal{L}(L_\pi^x)$ has access paths marked with $\bar{\mathbf{0}}\bar{\mathbf{1}}$ for $\mathbf{0}/\mathbf{1}$ removal arising from the **cons** rule.

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

- ▶ $\mathbf{0}$ removal from a set of access paths:

$$\alpha_1 \bar{\mathbf{00}} \alpha_2 \hookrightarrow \alpha_1 \alpha_2$$

$$\alpha_1 \bar{\mathbf{01}} \alpha_2 \hookrightarrow \text{drop } \alpha_1 \bar{\mathbf{01}} \alpha_2 \text{ from the set}$$

- ▶ $\mathbf{1}$ removal from a set of access paths:

$$\alpha_1 \bar{\mathbf{11}} \alpha_2 \hookrightarrow \alpha_1 \alpha_2$$

$$\alpha_1 \bar{\mathbf{10}} \alpha_2 \hookrightarrow \text{drop } \alpha_1 \bar{\mathbf{10}} \alpha_2 \text{ from the set}$$

GC decision problem

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

$$\bar{\mathbf{00}} \rightarrow \epsilon$$

$$\bar{\mathbf{11}} \rightarrow \epsilon$$

- ▶ The problem shown to be undecidable¹.
 - ▶ Reduction from Halting problem.

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

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 $\bar{0}/\bar{1}$ removal on the automaton.

Example

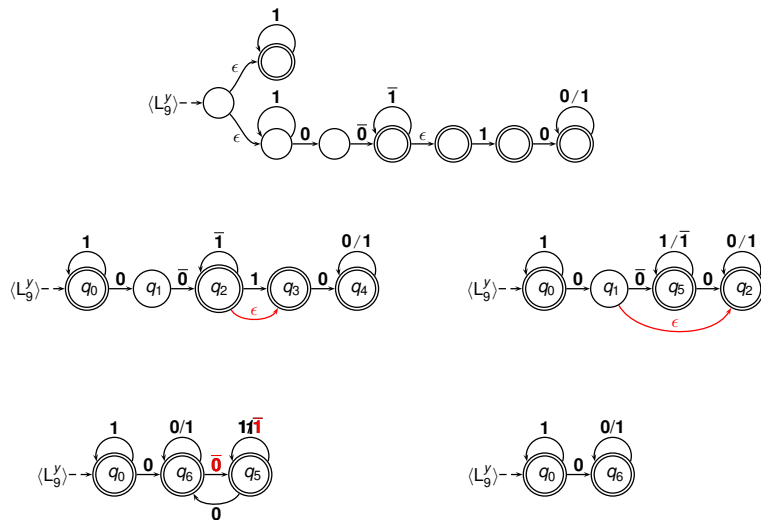
Grammar for L_9^Y

$$\begin{aligned} L_9^Y &\rightarrow I_{\text{append}}^1 \mid D_{\text{append}}^1 (\epsilon \mid \mathbf{1} \mid \mathbf{10}\sigma_{\text{all}}) \\ I_{\text{append}}^1 &\rightarrow \epsilon \mid \mathbf{1}I_{\text{append}}^1 \\ D_{\text{append}}^1 &\rightarrow \mathbf{00} \mid \mathbf{1}D_{\text{append}}^1\bar{\mathbf{1}} \\ \sigma_{\text{all}} &\rightarrow \epsilon \mid \mathbf{0}\sigma_{\text{all}} \mid \mathbf{1}\sigma_{\text{all}} \end{aligned}$$

After Mohri-Nederhoff transformation

$$\begin{aligned} L_9^Y &\rightarrow I_{\text{append}}^1 \mid D_{\text{append}}^1 (\epsilon \mid \mathbf{1} \mid \mathbf{10}\sigma_{\text{all}}) \\ I_{\text{append}}^1 &\rightarrow \epsilon \mid \mathbf{1}I_{\text{append}}^1 \\ D_{\text{append}}^1 &\rightarrow \mathbf{00}\widehat{D}_{\text{append}}^1 \mid \mathbf{1}D_{\text{append}}^1 \\ \widehat{D}_{\text{append}}^1 &\rightarrow \bar{\mathbf{1}}\widehat{D}_{\text{append}}^1 \mid \epsilon \\ \sigma_{\text{all}} &\rightarrow \epsilon \mid \mathbf{0}\sigma_{\text{all}} \mid \mathbf{1}\sigma_{\text{all}} \end{aligned}$$

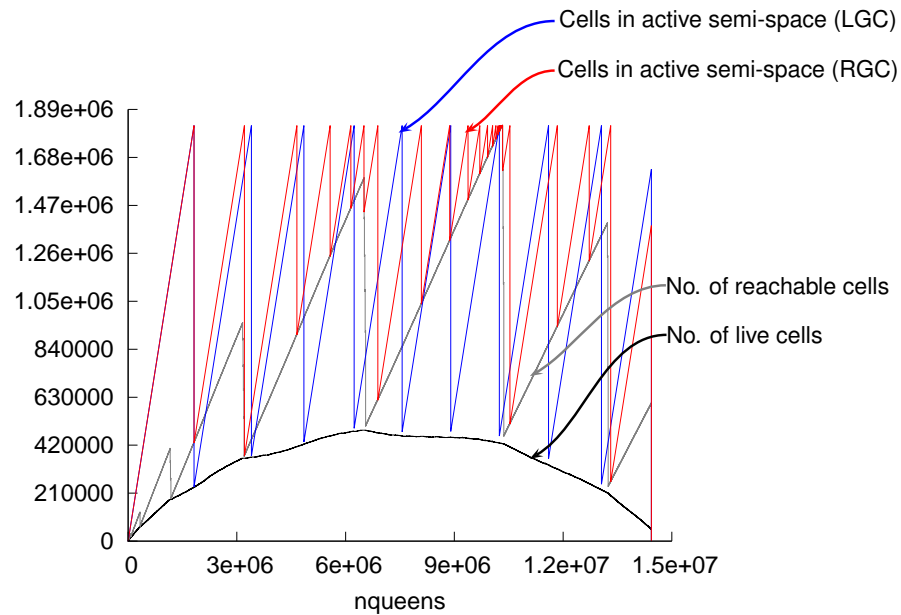
Automaton for L_9^Y



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.

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		#GCs		MinHeap (#cells)		GC time (sec)	
	RGC	LGC	RGC	LGC	RGC	LGC	RGC	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 $(\mathbf{car} \ x)$ is not evaluated at all, it does not generate any liveness for x
- ▶ Require a new terminal **2** with following semantics

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

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

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.