

string-case:
Discriminating without asking too many
questions

Paul Khuong

July 23, 2008

<http://discontinuity.info/~pkhuong/string-case.lisp, pdf>

Introduction

Implementation

Conclusion

Problem

```
(defun test (x)
  (case x
    ("foo" 1)
    ("bar" 2)
    ("quux" 3)
    (t      nil)))
```

(test "foo") => NIL !?

Workarounds

- ▶ cond: (**cond** ((**string=** x " ... ") ...) ...)
- ▶ hash table: (**case** (**gethash** x (load-time-value ...)) ...)
- ▶ symbols: (**case** (find-symbol x "PACKAGE") ...)

My solution: *Optimal* search tree

Optimal:

- ▶ $\leq (m + n)$ tests executed
- ▶ $\leq (m \cdot n)$ tests generated
- ▶ no test executed twice

test: (**eq1** [ch] (**aref** × [idx])).

Careful with default case and code duplication.

400 lines, heavily commented (semi-literate, $\approx 50\%$).

<http://discontinuity.info/~pkhuong/string-case.pdf>

Introduction

Implementation

Conclusion

Generating the search tree

1. Discriminate by length
2. Prune possibilities down to a single candidate
3. Confirm candidate

Discriminating by length

```
{possibilities}  
{lengths}  
(typecase × ((array * 3) ...) ...)
```

Pruning possibilities

```
(defun %generate-tree (possibilities)
  (multiple-value-bind (idx ch)
    (find-best-split possibilities)
    '(if (eql ,ch (aref x ,idx))
        ,(%generate-tree (keep possibilities
                           idx ch))
        ,(%generate-tree (keep-if-not possibilities
                           idx ch))))))
```

Finding the best *split*

Good split: as close to 50/50 as possible (balanced binary tree)

$$\max_{(i,c) \in \text{Indices} \times \text{Char}} \min \left\{ \begin{array}{l} \|\{str \in \text{poss} \mid str_i = c\}\| \\ \|\{str \in \text{poss} \mid str_i \neq c\}\| \end{array} \right\}$$

$$\max_{i \in \text{Indices}} \max_{c \in \{str_i \mid str \in \text{poss}\}} \min \left\{ \begin{array}{l} \|\{str \in \text{poss} \mid str_i = c\}\| \\ \|\{str \in \text{poss} \mid str_i \neq c\}\| \end{array} \right\}$$

Confirming the candidate

- ▶ Iterate through all the indices (unrolled `string=` loop)
Keep track of positive information (known indices)
... by tracking set of indices we don't know yet.
- ▶ Coalesce tests via (**logior** (**logxor** \times y) ...) .
Work/misprediction trade-off.
- ▶ Hoists tests earlier.

Introduction

Implementation

Conclusion

Recap.

Want to construct a search tree to discriminate between "foo", "bar" and "quux".

1. Discriminate by length
2. Generate each subtree
3. Confirm the match

Discriminate by length

Possibilities: {"foo", "bar", "quux"}

Lengths: {2, 3}

```
(typecase x
  ((simple-array * 2) [tree for {"bar", "baz"}])
  ((simple-array * 3) [tree for "quux"])
  (t [default]))
```

Generate each subtree

Possibilities: {"foo", "bar"}

"foo"₀ = f ≠ b = "bar"₀

```
(if (eql #\f (aref x 0))  
    [tree for {"foo"}]  
    [tree for {"bar"}])
```

Confirm the match

x_0 is known to be f. Leaves 1, 2 to check (length is known too).

```
(if (and (zerop
         (logior
          (logxor (char-code #\o) (char-code (aref x 1)))
          (logxor (char-code #\o) (char-code (aref x 2))))
      [body for "foo"]
      [default]))
```

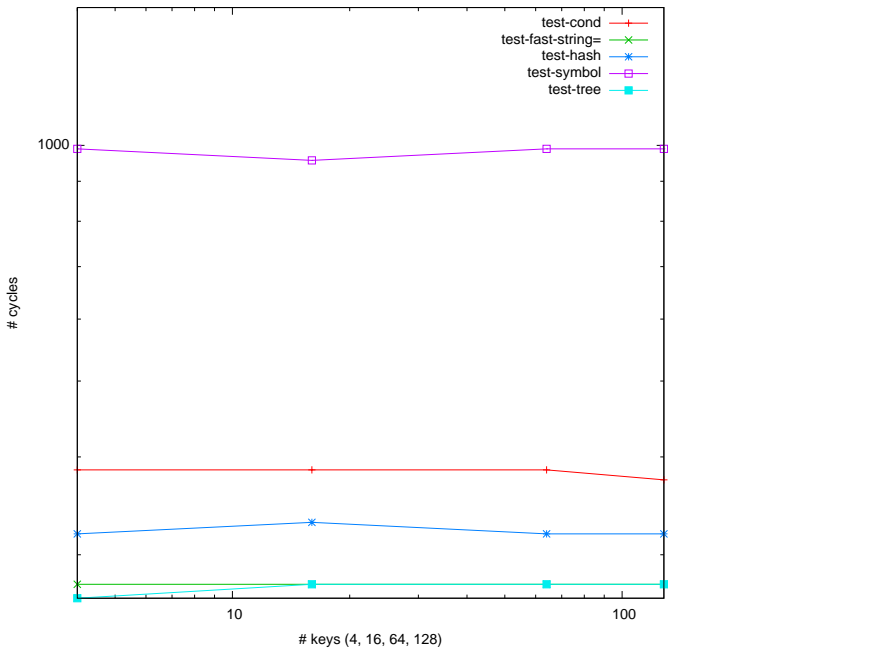
Putting it all together

```
(typecase x
  ((simple-array * 2)
   (if (eql #\f (aref x 0))
       (if (and (zerop
                 (logior
                  (logxor (char-code #\o)
                           (char-code (aref x 1)))
                  (logxor (char-code #\o)
                           (char-code (aref x 2))))))
           [body for "foo"]
           [default])
       [confirm "bar"])))
((simple-array * 3) [tree for "quux"])
(t [default]))
```

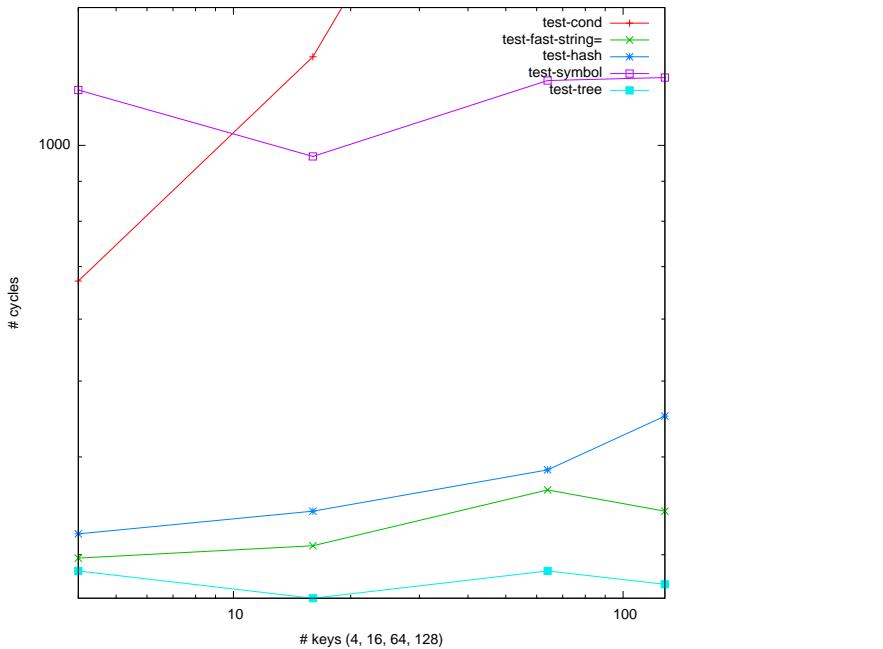
Performance tests (microbenchmark)

- ▶ Tested: `cond/string=`, `cond/fast-string=`, `case/hash`, `case/symbol`, `tree`
- ▶ Possibilities: randomly generated, A-Z, length 3 to 15
- ▶ Workloads: first, last, miss, random
- ▶ Instrumentation overhead: ≈ 264 cycles, report median of 1024
- ▶ All graphs are log – log and the y axis is cut off at 264.

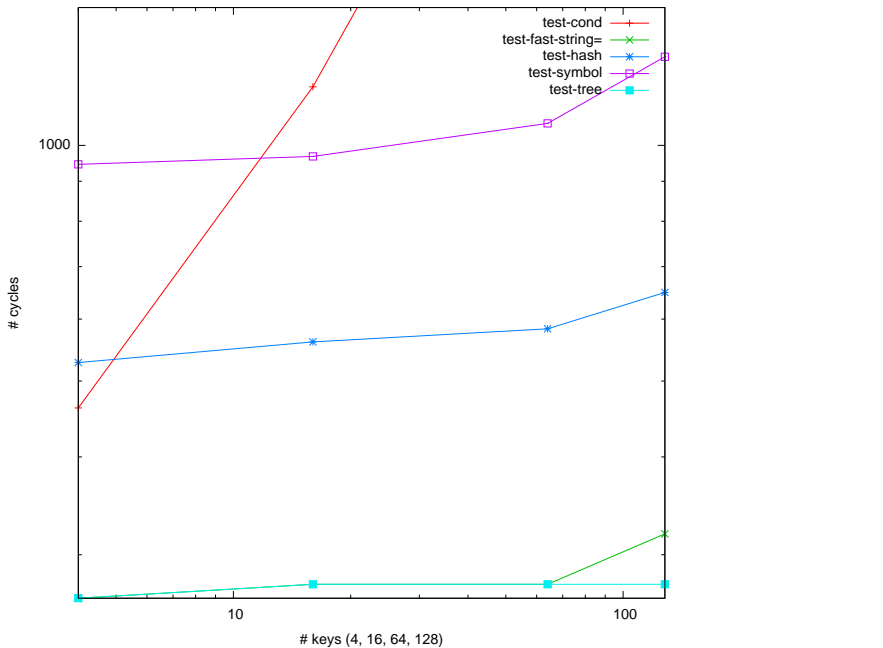
test-head



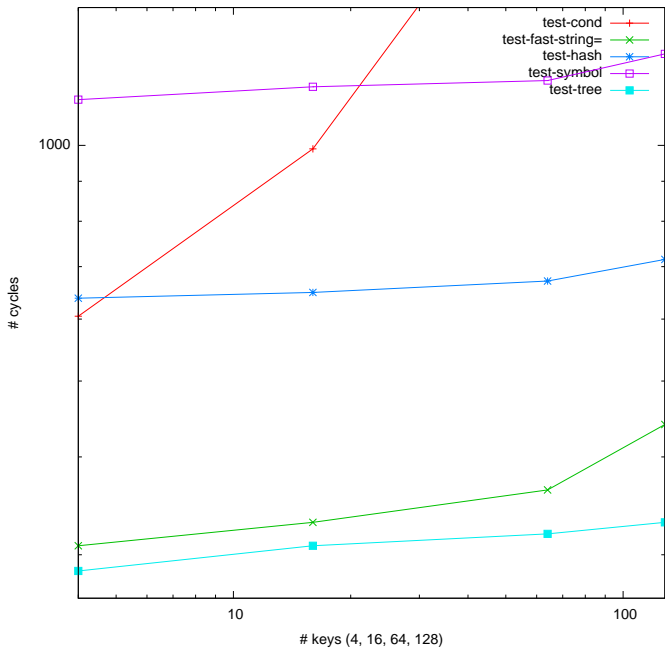
test-tail



test-fail



test-random



What's next?

- ▶ Generation time? $O(m \cdot n^2)$
- ▶ Branch prediction, probabilities.
- ▶ $O(1)$ jump tables (switch/case)
- ▶ Wildcards? Other string matching/processing tasks (non-regex)...

<http://discontinuity.info/~pkhuong/string-case.lisp>
(or .pdf)