

Pudding Eater ¹

Eric Bailey

November 20, 2017 ²

Converting Node Identifiers

First, create a string representation of `exp`, with escape characters written where appropriate, via `prin1-to-string`.

Then replace each character that *(is not alphanumeric 1c)* with *(an underscore 1d)*.

```
1e (<* 1a>)+≡  
  (defun dot-name (exp)  
    (substitute-if <an underscore 1d> <is not alphanumeric 1c> <exp as a string 1b>)))
```

Defines:

`dot-name`, used in chunks 1, 2, 4, and 7.

Adding Labels to Graph Nodes

```
1h (<* 1a>)+≡  
  (defparameter *max-label-length* 30)  
  
  (defun dot-label (exp)  
    (if exp  
      (let ((s <create a string representation of exp 1f>))  
        <Truncate s if it's too long. 1k>))  
      <otherwise return the empty string 1g>)))
```

Defines:

`*max-label-length*`, used in chunk 1.
`dot-label`, used in chunks 2 and 4.

If *<s is too long 1i>*, i.e. more than `*max-label-length*` long, *<truncate s 1j>* and append "...".

```
1k <Truncate s if it's too long. 1k>≡  
  (if <s is too long 1i>  
    (concatenate 'string <truncate s 1j> "...")  
    s)
```

This code is used in chunk 1h.

¹

Conrad Barski. *Land of Lisp: Learn to Program in Lisp, One Game at a Time!*, chapter 7, pages 107–127. No Starch Press, 2010. ISBN 9781593273491. URL <http://landoflisp.com>

² Last updated November 21, 2017

`src/graphviz.lisp`:

```
1a (<* 1a>)+≡  
  (in-package :cl-user)  
  (defpackage lol.graphviz  
    (:use :cl :prove)  
    (:export dot-name))  
  (in-package :lol.graphviz)
```

This definition is continued in chunks 1–4.

Root chunk (not used in this document).

Defines:

`lol.graphviz`, used in chunk 7.

Uses `dot-name` 1e.

```
1b <exp as a string 1b>≡  
  (prin1-to-string exp)
```

This code is used in chunk 1e.

```
1c <is not alphanumeric 1c>≡  
  (complement #'alphanumericp)
```

This code is used in chunk 1e.

```
1d <an underscore 1d>≡  
  #\_
```

This code is used in chunk 1e.

```
1f <create a string representation of exp 1f>≡  
  (write-to-string exp :pretty nil)
```

This code is used in chunk 1h.

```
1g <otherwise return the empty string 1g>≡  
  ""
```

This code is used in chunk 1h.

```
1i <s is too long 1i>≡  
  (> (length s) *max-label-length*)
```

This code is used in chunk 1k.

Uses `*max-label-length*` 1h.

```
1j <truncate s 1j>≡  
  (subseq s 0 (- *max-label-length* 3))
```

This code is used in chunk 1k.

Uses `*max-label-length*` 1h.

Generating the DOT Information for the Nodes

```

2a  (* 1a)+≡
      (defun nodes→dot (nodes)
        (mapc (lambda (node)
                  (fresh-line)
                  (princ (dot-name (car node)))
                  (princ "[label=\")
                  (princ (dot-label node))
                  (princ "\"];"))
              nodes))

```

Defines:

nodes→dot, used in chunks 3a and 4.
 Uses dot-label 1h and dot-name 1e.

Converting Edges into DOT Format

```

2b  (* 1a)+≡
      (defun edges→dot (edges)
        (mapc (lambda (node)
                  (mapc (lambda (edge)
                          (fresh-line)
                          (princ (dot-name (car node)))
                          (princ "→")
                          (princ (dot-name (car edge)))
                          (princ "[label=\")
                          (princ (dot-label (cdr edge)))
                          (princ "\"];"))
                      (cdr node)))
              edges))

```

Defines:

edges→dot, used in chunks 3a and 4.
 Uses dot-label 1h and dot-name 1e.

Generating All the DOT Data

```

3a  (* 1a)+≡
    (defun graph→dot (nodes edges)
      (princ "digraph{")
      (nodes→dot nodes)
      (edges→dot edges)
      (princ "}"))

```

Defines:

graph→dot, used in chunk 3c.

Uses edges→dot 2b and nodes→dot 2a.

Turning the DOT File into a Picture

```

3b  (* 1a)+≡
    (defun dot→png (fname thunk)
      (with-open-file (*standard-output*
                      fname
                      :direction :output
                      :if-exists :supersede)
        (funcall thunk))
      (uiop:run-program (concatenate 'string "dot -Tpng -O " fname)))

```

Defines:

dot→png, used in chunks 3c and 4.

Creating a Picture of Our Graph

```

3c  (* 1a)+≡
    (defun graph→png (fname nodes edges)
      (dot→png fname
        (lambda ()
          (graph→dot nodes edges))))

```

Defines:

graph→png, never used.

Uses dot→png 3b and graph→dot 3a.

Creating Undirected Graphs

```

4  < * 1a > +≡
    (defun uedges→dot (edges)
      (maplist (lambda (lst)
        (mapc (lambda (edge)
          (unless (assoc (car edge) (cdr lst))
            (fresh-line)
            (princ (dot-name (caar lst)))
            (princ "-")
            (princ (dot-name (car edge)))
            (princ "[label=\"")
            (princ (dot-label (cdr edge)))
            (princ "\"];"))
          (cdr lst)))
        edges))

    (defun ugraph→dot (nodes edges)
      (princ "graph{")
      (nodes→dot nodes)
      (edges→dot edges)
      (princ "}"))

    (defun ugraph→png (fname nodes edges)
      (dot→png fname
        (lambda ()
          (ugraph→dot nodes edges))))

```

Defines:

uedges→dot, never used.

ugraph→dot, never used.

ugraph→png, never used.

Uses dot→png 3b, dot-label 1h, dot-name 1e, edges→dot 2b, and nodes→dot 2a.

Full Listing

```

1  (in-package :cl-user)
2  (defpackage lol.graphviz
3    (:use :cl :prove)
4    (:export dot-name))
5  (in-package :lol.graphviz)
6
7
8  (defun dot-name (exp)
9    (substitute-if #\_- (complement #'alphanumericp) (prin1-to-string exp)))
10
11
12  (defparameter *max-label-length* 30)
13
14  (defun dot-label (exp)
15    (if exp
16        (let ((s (write-to-string exp :pretty nil)))
17          (if (> (length s) *max-label-length*)
18              (concatenate 'string (subseq s 0 (- *max-label-length* 3)) "...")
19              s))
20        ""))
21
22
23  (defun nodes->dot (nodes)
24    (mapc (lambda (node)
25            (fresh-line)
26            (princ (dot-name (car node)))
27            (princ "[label=\\"))
28            (princ (dot-label node))
29            (princ "\\];"))
30          nodes))
31
32
33  (defun edges->dot (edges)
34    (mapc (lambda (node)
35            (mapc (lambda (edge)
36                    (fresh-line)
37                    (princ (dot-name (car node)))
38                    (princ "→")
39                    (princ (dot-name (car edge)))
40                    (princ "[label=\\"))
41                    (princ (dot-label (cdr edge)))
42                    (princ "\\];"))
43                  (cdr node)))
44          edges))

```

```

47 (defun graph→dot (nodes edges)
48   (princ "digraph{")
49   (nodes→dot nodes)
50   (edges→dot edges)
51   (princ "}"))
52
53
54 (defun dot→png (fname thunk)
55   (with-open-file (*standard-output*
56                   fname
57                   :direction :output
58                   :if-exists :supersede)
59     (funcall thunk))
60   (uiop:run-program (concatenate 'string "dot -Tpng -O " fname)))
61
62
63 (defun graph→png (fname nodes edges)
64   (dot→png fname
65     (lambda ()
66       (graph→dot nodes edges))))
67
68
69 (defun uedges→dot (edges)
70   (maplist (lambda (lst)
71             (mapc (lambda (edge)
72                     (unless (assoc (car edge) (cdr lst))
73                       (fresh-line)
74                       (princ (dot-name (caar lst)))
75                       (princ "--")
76                       (princ (dot-name (car edge)))
77                       (princ "[label=\\")
78                       (princ (dot-label (cdr edge)))
79                       (princ "\\"];"))
80                     (cдар lst)))
81             edges))
82
83
84 (defun ugraph→dot (nodes edges)
85   (princ "graph{")
86   (nodes→dot nodes)
87   (edges→dot edges)
88   (princ "}"))
89
90
91 (defun ugraph→png (fname nodes edges)
92   (dot→png fname
93     (lambda ()
94       (ugraph→dot nodes edges))))

```

Tests

```
7 <test/graphviz.lisp 7>≡  
  (in-package :lol.graphviz)  
  
  (plan 1)  
  
  (subtest "Converting Node Identifiers"  
    (is (dot-name 'living-room)  
        "LIVING_ROOM")  
    (is (dot-name 'foo!)  
        "FOO_")  
    (is (dot-name '24)  
        "24"))
```

```
(finalize)  
Root chunk (not used in this document).  
Uses dot-name 1e and lol.graphviz 1a.
```

Glossary

object any Lisp datum. 8

`prin1-to-string` acts like `write-to-string` with `:escape t`, that is, escape characters are written where appropriate. 1, 8

`write-to-string` `prin1-to-string` and `princ-to-string` effectively print an *object* as if by `write`, `prin1`, or `princ`, respectively, and the characters that would be output are made into a string. 8

References

Conrad Barski. *Land of Lisp: Learn to Program in Lisp, One Game at a Time!*, chapter 7, pages 107–127. No Starch Press, 2010. ISBN 9781593273491. URL <http://landoflisp.com>.