

Visualizing Graph Data ¹

Eric Bailey

November 20, 2017 ²

Contents

<i>Converting Node Identifiers</i>	2
<i>Adding Labels to Graph Nodes</i>	2
<i>Generating the DOT Information for the Nodes</i>	3
<i>Converting Edges into DOT Format</i>	3
<i>Generating All the DOT Data</i>	4
<i>Turning the DOT File into a Picture</i>	4
<i>Creating a Picture of Our Graph</i>	4
<i>Creating Undirected Graphs</i>	5
<i>Full Listing</i>	6
<i>Tests</i>	8

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:

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

This definition is continued in
chunks 2–5.

Root chunk (not used in this document).

Defines:

lol.graphviz, used in chunk 8.

Uses dot-name 2d.

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* `2b` with *an underscore* `2c`.

`2a` `<exp as a string 2a>≡`
`(prin1-to-string exp)`
 This code is used in chunk `2d`.

`2b` `<is not alphanumeric 2b>≡`
`(complement #'alphanumericp)`
 This code is used in chunk `2d`.

`2c` `<an underscore 2c>≡`
`#_`
 This code is used in chunk `2d`.

`2d` `<* 1>+≡`
`(defun dot-name (exp)`
`(substitute-if <an underscore 2c> <is not alphanumeric 2b> <exp as a string 2a>))`

Defines:

`dot-name`, used in chunks `1`, `3`, `5`, and `8`.

Adding Labels to Graph Nodes

`2e` `<create a string representation of exp 2e>≡`
`(write-to-string exp :pretty nil)`
 This code is used in chunk `2g`.

`2f` `<otherwise return the empty string 2f>≡`
`""`
 This code is used in chunk `2g`.

`2g` `<* 1>+≡`
`(defparameter *max-label-length* 30)`

`(defun dot-label (exp)`
`(if exp`
`(let ((s <create a string representation of exp 2e>))`
`<Truncate s if it's too long. 2j>)`
`<otherwise return the empty string 2f>))`

Defines:

`*max-label-length*`, used in chunk `2`.
`dot-label`, used in chunks `3` and `5`.

If `<s is too long 2h>`, i.e. more than `*max-label-length*` long, `<truncate s 2i>` and append `"..."`.

`2h` `<s is too long 2h>≡`
`(> (length s) *max-label-length*)`
 This code is used in chunk `2j`.
 Uses `*max-label-length* 2g`.

`2i` `<truncate s 2i>≡`
`(subseq s 0 (- *max-label-length* 3))`
 This code is used in chunk `2j`.
 Uses `*max-label-length* 2g`.

`2j` `<Truncate s if it's too long. 2j>≡`
`(if <s is too long 2h>`
`(concatenate 'string <truncate s 2i> "...")`
`s)`

This code is used in chunk `2g`.

Generating the DOT Information for the Nodes

```

3a  (* 1)+≡
      (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 4a and 5.

Uses dot-label 2g and dot-name 2d.

Converting Edges into DOT Format

```

3b  (* 1)+≡
      (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 4a and 5.

Uses dot-label 2g and dot-name 2d.

Generating All the DOT Data

```

4a  (* 1)+≡
      (defun graph→dot (nodes edges)
        (princ "digraph{")
        (nodes→dot nodes)
        (edges→dot edges)
        (princ "}"))

```

Defines:

graph→dot, used in chunk 4c.

Uses edges→dot 3b and nodes→dot 3a.

Turning the DOT File into a Picture

```

4b  (* 1)+≡
      (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 4c and 5.

Creating a Picture of Our Graph

```

4c  (* 1)+≡
      (defun graph→png (fname nodes edges)
        (dot→png fname
          (lambda ()
            (graph→dot nodes edges))))

```

Defines:

graph→png, never used.

Uses dot→png 4b and graph→dot 4a.

Creating Undirected Graphs

```

5  < * 1 > + ≡
    (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 4b, dot-label 2g, dot-name 2d, edges→dot 3b, and nodes→dot 3a.

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

```
8 <test/graphviz.lisp 8>≡  
  (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 2d and lol.graphviz 1.
```


Glossary

object any Lisp datum. 9

`prin1-to-string` acts like `write-to-string` with `:escape t`, that is, escape characters are written where appropriate. 2, 9

`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. 9

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>.