Visualizing Graph Data [1]

Eric Bailey

November 20, 2017 [2]

[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

Contents

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.

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

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

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 "\"];")))
                     (cdar 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

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


(defun dot-name (exp)
  (substitute-if #\_ (complement #'alphanumericp) (prin1-to-string exp)))


(defparameter *max-label-length* 30)

(defun dot-label (exp)
  (if exp
      (let ((s (write-to-string exp :pretty nil)))
        (if (> (length s) *max-label-length*)
            (concatenate 'string (subseq s 0 (- *max-label-length* 3)) "...")
            s))
      ""))


(defun nodes->dot (nodes)
  (mapc (lambda (node)
          (fresh-line)
          (princ (dot-name (car node)))
          (princ "[label=\"")
          (princ (dot-label node))
          (princ "\"];"))
        nodes))


(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))
```

```
(defun graph->dot (nodes edges)
  (princ "digraph{")
  (nodes->dot nodes)
  (edges->dot edges)
  (princ "}"))


(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)))


(defun graph->png (fname nodes edges)
  (dot->png fname
            (lambda ()
              (graph->dot nodes edges))))


(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 "\"];")))
                   (cdar 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))))
```

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`.