TRAVERSING THE WORLD WIDE WEB
A Kleisli/CPL Tutorial

Limsoon Wong

BioInformatics Centre &
Institute of Systems Science
Heng Mui Keng Terrace, Singapore 119597
`email: limsoon@iss.nus.sg`

4 Janaury 1997

Kleisli/CPL Version D Release 31 December 1996
BioKleisli Experimental Version Release 31 December 1996

1 Introduction

Kleisli/CPL [4, 1] is a powerful query system for broad-scale integration of heterogeneous distributed databanks. It is particularly suitable for dealing with the complexity of the many existing biological databanks. I have begun encapsulating many frequently used routines for accessing various biological databanks into a library suite that I called **BioKleisli**.

BioKleisli is designed for implementing adhoc queries and for performing on-the-fly integration. It offers a lot of flexibility and power. As a result, many queries can be implemented very easily. One particularly interesting type of queries are those that traverses the World Wide Web to gather interesting web pages.

In this tutorial, I will show how BioKleisli can be used to implement such a "web crawler" in a simple way. I will illustrate the task by writing both a depth-first and a breadth-first traversal routine.[1]

This tutorial primarily concerns the use of the BioKleisli library suite. You will need to have some background understanding of CPL, the Collection Programming Language supported by Kleisli, to fully appreciate this tutorial. You should consult [6] for a detailed description of CPL and [2, 3] for an introduction to the underlying theory.

[1]I would like to thank Lim Teck Sin for suggesting this demo to me.

2 Overview

Our web crawler is a simple one, as shown in Figure 1. It consists of a program `traverse`, which traverses the WWW to gather web pages that are reachable within a specified number of links. It generates a list of URL's that are referenced and reached during the traversal; this is stored in the file `good.html`. It also fecthes all pages corresponding to these good URL's and caches them in the directory `cache`. It names these cached files `1.html`, `2.html`, etc. Finally, it generates a list of URL's that are referenced during the traversal but cannot be fetched for some reason (eg: the page does not exist any more); this is stored in the file `bad.html`.

In the rest of this tutorial, I will first show how you can implement `traverse` in a depth first manner. Then I will show you the breadth-first version.

```
             +------------+
WWW -------->|  traverse  |<--------- control parameters
             +------------+
                    |
      +-------------+-----------+
      |             |           |
      V             V           V
   good.html     bad.html     cache
```

Figure 1: A simple web crawler.

3 Depth-First Traversal

Having implemented all the above, we can now easily gather various interesting web pages as we crawl around the web.

Example 3.1 *Suppose we want to traverse the web in a depth-first manner for two levels starting from the BioInformatics Centre's home page and gather the first 20 pages we see. Furthermore, we do not want to see any pages ending with ".Z," ".gz,", or ".ps," and we are only interested in those pages having ".iss." somewhere in their paths. We can do this by:*

```
gather-dtraverse (
  [(#url: "//corona.iss.nus.sg:8080", #depth: 2)],
  {},
  20,
  gather-filter ("%.Z %.gz %.ps", "%.iss.%"),
  gather-process)
```

```
primitive gather-dtraverse == fix (
  \traverse => (\urls, \acc, \count, \omit, \proc) =>
    if urls = []
    then ()
    else if count < 0
    then ()
    else if urls.list-head.#url.omit
    then traverse (urls.list-tail, acc, count, omit, proc)
    else if urls.list-head.#depth < 0
    then traverse (urls.list-tail, acc, count, omit, proc)
    else if urls.list-head.#url set-member acc
    then traverse (urls.list-tail, acc, count, omit, proc)
    else [ (#url: x, #depth: urls.list-head.#depth - 1) |
              \x <--- proc (
                   urls.list-head.#url,
                   "-good.html",
                   "-bad.html",
                   count.num-stringify ^ ".html") ] before (\more =>
         traverse (more [+] urls, urls.list-head.#url +} acc,
           count - 1, omit, proc)));
```

Figure 2: The gather-dtraverse program.

```
primitive gather-process == (\url, \good, \bad, \file) =>
 urlget-doit ("http:" ^ url) before (\raw =>
 if raw = ""
 then appendtofile (bad,
         "<a href=http:" ^ url ^ ">" ^ url ^ "</a><br>" ^ string-newline)
      before ( _ =>
      [])
 else appendtofile ( good,
         url ^
         " <a href=http:" ^ file ^ ">cached copy</a>" ^
         " <a href=http:" ^ url ^ ">original site</a><br>" ^ string-newline)
      before ( _ =>
      appendtofile (file,
         "<base href=http:" ^ url ^ "></base>" ^ string-newline ^ raw)
      before (\page =>
      deep-get-http-url (page)
      before (\urls =>
      deep-split-url (url, "", "")
      before ((\root, \curr, _) =>
      [ deep-format-url (u, root, curr) | \u <--- urls ])))));
```

Figure 3: The gather-process program.

```
primitive gather-filter == (\forb, \comp) => \url =>
  (url util-has-some-pattern forb) orelse
  ((not (comp = "")) andalso (url util-has-no-pattern comp));
```

Figure 4: The gather-filter program.

4 Breadth-First Traversal

Breadth-first traversal means that we can visit a page at level $i + 1$ only after we finish visiting *all* the pages at level i. It turns out that we can implement this mode of traversal in the program `gather-btraverse` shown in Figure 5. It differs from the `gather-dtraverse` program in Line 19, where we have

```
19a.    traverse (urls [+] more, urls.list-head.#url +} acc,
```

as oppose to

```
19.     traverse (more [+] urls, urls.list-head.#url +} acc,
```

Recall that in these programs, `urls` is the list of URL's that is yet to be traversed and `more` is the list of additional URL's found in the current page. Line 19a says that we should visit the pages listed in `urls` before those in `more`. Since those in `more` is one level below the current page, this implies that the next level would not be visited unless we have finished all the pages found on the current level (they are listed in `urls`). This of course gives us the desired breadth-first traversal behaviour,

Example 4.1 *Suppose we want to traverse the web in a breadth-first manner for two levels starting from the BioInformatics Centre's home page and gather the first 20 pages we see. Furthermore, we do not want to see any pages ending with ".Z," ".gz,", or ".ps," and we are only interested in those pages having ".iss." somewhere in their paths. We can do this by:*

```
gather-btraverse (
  [(#url: "//corona.iss.nus.sg:8080", #depth: 2)],
  {},
  20,
  gather-filter ("%.Z %.gz %.ps", "%.iss.%"),
  gather-process)
```

```
primitive gather-btraverse == fix (
 \traverse => (\urls, \acc, \count, \omit, \proc) =>
    if urls = []
    then ()
    else if count < 0
    then ()
    else if urls.list-head.#url.omit
    then traverse (urls.list-tail, acc, count, omit, proc)
    else if urls.list-head.#depth < 0
    then traverse (urls.list-tail, acc, count, omit, proc)
    else if urls.list-head.#url set-member acc
   then traverse (urls.list-tail, acc, count, omit, proc)
    else [ (#url: x, #depth: urls.list-head.#depth - 1) |
              \x <--- proc (
                   urls.list-head.#url,
                   "-good.html",
                   "-bad.html",
                   count.num-stringify ^ ".html") ] before (\more =>
          traverse (urls [+] more, urls.list-head.#url +} acc,
            count - 1, omit, proc)));
```

Figure 5: The gather-btraverse program.

References

[1] Peter Buneman, Susan Davidson, Kyle Hart, Chris Overton, and Limsoon Wong. A data transformation system for biological data sources. In *Proceedings of 21st International Conference on Very Large Data Bases*, pages 158–169, Zurich, Switzerland, August 1995.

[2] Peter Buneman, Leonid Libkin, Dan Suciu, Val Tannen, and Limsoon Wong. Comprehension syntax. *SIGMOD Record*, 23(1):87–96, March 1994.

[3] Peter Buneman, Shamim Naqvi, Val Tannen, and Limsoon Wong. Principles of programming with complex objects and collection types. *Theoretical Computer Science*, 149(1):3–48, September 1995.

[4] Susan Davidson, Christian Overton, Val Tannen, and Limsoon Wong. BioKleisli: A digital library for biomedical researchers. Technical Report 96-215-0, Institute of Systems Science, Heng Mui Keng Terrace, Singapore 119597, August 1996.

[5] Carl A. Gunter. *Semantics of Programming Languages: Structures and Techniques*. Foundations of Computing. MIT Press, 1992.

[6] Limsoon Wong. *The Collection Programming Language Reference Manual*. Institute of Systems Science, Heng Mui Keng Terrace, Singapore 119597, October 1995. Available at `http://sdmc.iss.nus.sg/kleisli/psZ/cpl-defn.ps`.