Downloading Sequence Reports According To Locus Names
A Kleisli/CPL Tutorial

Wang Jiren

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

10 June 1997

Kleisli/CPL Version D Release 6 April 1997
BioKleisli Experimental Version Release 23 April 1997

1 Introduction

In this tutorial, we show some Kleisli/CPL programs that used to download complete sequence reports given locus names.[1] I was given a list of about 13860 locus names. The objective was to download GenBank sequence reports corresponidng to these loci.

As the number of locus names involved is quite large, a lot of time is needed to download their complete sequence reports from GenBank. The problem of interruption of the data transmission was anticipated to take place with high likelihood. I will show here two sets of programs. The first assumes that data transmission would never be interrupted. The second relaxes this assumption. The simpler first program gives us a chance to focus on the key steps involved, before learning how to handle the engineering problem of unreliable transmission.

2 The First Program

There are four key steps.

Step 1. The locus names are given to us in a file `data.txt`. Each locus name is given on a separate line. To simplfy subsequent manipulation, I read it in to CPL as a set, `data-list`, defined below.

```
primitive data-list ==
  { x | \x <--- process (
                  "/usr/bin/cat",
                  ["data.txt"],
                  "")
                using syscall-line };
```

Step 2. GenBank only allows sequence reports to be downloaded given their unique identifiers (UID) or their accession numbers. So it is necessary for me to obtain these UID first. Fortunately, there is a web page at NCBI that let us search for UIDs given locus names. So in this step, we have to submit the locus names in `data-list` one after the other to get their corresponding UID, which must be extracted from the HTML pages returned by the server at NCBI. Here is the program.[2] The results are saved to the file `map-locus`.

```
!
! load url routines
!
usescript "urlget-12oct96.cpl";
!
! load perl library
!
usescript "perl-25oct96.cpl";
!
```

[1] I thank Meena Sakharkar for suggesting this problem.

[2] This program uses routines from the files `urlget-12oct96.cpl` and `perl-25oct96.cpl` from the BioKleisli libraries. In particular, the `urlget-doit` function returns a HTML page given the corresponding URL as input.

```
! function to form url for getting GenBank UID
!
primitive mk-url == \locus =>
  "http://www2.ncbi.nlm.nih.gov/cgi-bin/submit_form_query?" ^
  "TITLE=GenBank+Text+Retrieval+Output&INPUTS=4&Q1=" ^ locus ^
  "&OP1=AND&Q2=&OP2=AND&Q3=&OP3=AND&Q4=&NDOCS=100&DB" ^
  "=gbupdates+genbank";
!
! function to use the locus to get a html page from GenBank server
! then extract GenBank UID from the resulting html page.
! the UID always appears in the HTML page returned in
! a context like: ...birx_doc?UID"><IMG...
!
primitive get-uid == \data-list =>
  { (#locus: l, #link: L)
  | \l <- data-list,
    \L == perl-cond-extractline-string (
            l.mk-url.urlget-doit,
            "(.*)birx_doc\?+(.*)" ^ string-quote ^ "><IMG(.*)",
            "$2",
            "/") };
!
! get the UID and write them to the file map-locus.
!
write get-uid (data-list) to "map-locus" using stdout;
```

Step 3. The UID in the file `map-locus` can now be used to bring back the desired sequence report by sending a HTTP request like `http://www2.ncbi.nlm.nih.gov/cgi-bin/birx_doc?`UID to the server at NCBI. Here is the program to repeat the above request for all the UID in `map-locus`. We store the resulting HTML pages in the sub-directory `/usr2/jrwang/result` in files having names of the form LocusName-UID.

```
!
! read GenBank UID from the map-locus file
!
readfile address from "map-locus" using stdin;
!
! function to get HTML pages and store them in the files
!
primitive get-report == \address =>
  { force (
      process (
        "/usr2/jrwang/result/" ^ x ^ "-" ^ z,
        urlget-doit ("http://www2.ncbi.nlm.nih.gov/cgi-bin/birx_doc?" ^ z))
      using appendtofile)
  | (#locus: \x, #link: \y) <- address,
    \z <--- y };
!
! go get and store the HTML pages.
!
get-report (address);
```

Step 4. The HTML pages I downloaded contain some HTML tags that should be removed. Also some symbols such as <, >, and " are coded respectively as <, >, and "; these should be decoded as well. Here is the program to perform these "editing." We stored the resulting files in the directory /usr2/jrwang/final.

```
!
! here is the list of files to be ''edited.''
! recall these files are those we stored in /usr2/jrwang/result.
!
primitive to-be-edited ==
  process (
    "/usr/bin/ls",
    ["/usr2/jrwang/result"],
    "")
  using syscall-line;
!
! function to filter out the lines starting
!  "<TITLE>", "<BODY", and "</PRE>"; and to
! decode &quot;, &lt;, and &gt;.
!
primitive edit == \file =>
  string-implode
    [ y ^ string-newline
    | \x <--- process (
                "/usr/bin/cat",
                ["/home/BIC/tmp/result/" ^ file],
                "")
              using syscall-line,
      not (x string-islike "<TITLE>%"),
      not (x string-islike "<BODY%"),
      not (x string-islike "</PRE>%"),
      \y == perl-extract-string (
              perl-extract-string (
                perl-extract-string (
                  x,
                  "&gt;", ">", "/"),
                "&lt;", "<", "/"),
              "&quot;", string-quote, "/")];
!
! apply the edit function to all the HTML files.
!
{ process ("/usr2/jrwang/final/" ^ file, edit (file)) using appendtofile
| \file <--- to-be-edited };
```

The desired GenBank sequence reports corresponding to the input locus names would then be available in the directory /usr2/jrwang/final.

3 The Second Program

The first program assumes the network is reliable. Unfortunately, given that we have over 13000 locus names and thus over 13000 GenBank sequence reports to download, network failure is inevitable. So we must modifiy the first program to handle such a situation in a way that reduced the amount of work that needs to be repeated. Steps 2 and 3 are the two steps that could fail due to network problem. A general strategy for dealing with such a problem is to use delta files. We illustrate it on Step 2.

Step 2 (Recovery). Recall that Step 2 was downloading UID given locus names. Suppose the set of locus names to use is $S_1 = \{L_1, \ldots, L_n\}$. Suppose we downloaded UID corresponding to locus names in the set $S_2 = \{L_1, \ldots, L_m\}$ with $n > m$. Then the remaining locus names to be processed are given by the set $S_1 - S_2 = \{L_{m+1}, L_{m+2}, ..., L_n\}$. Here is how to set things up in general.

We know what are in set S_1, which is simply the contents of the set `data-list`. To figure out what are in set S_2, we need to find out what we have already downloaded in the `map-locus` file. So we read the file back in[3] and extract the locus names from it. The set `delta` can now be defined as the difference between S_1 and S_2 in the straightforward declaration below.

```
readfile current from "map-locus" using stdin;
primitive delta == data-list set-diff { x.#locus | \x <- current };
```

Now, to produce a file `map-locus2` containing what we already downloaded (ie. those in the file `map-locus`) and those that we still have to download, we simply do the following:

```
writefile current {+} (get-uid (delta))
to "map-locus2" using stdout;
```

Let the above run until it completes or until there is another network failure. At which point, we rename the resulting `map-locus2` file to `map-locus`. If the above completes without network failure, we proceed to Step 3. If there is another network failure, we simply repeat the recovery procedure as described above.

Step 3 (Recovery). Recall that Step 3 was downloading GenBank sequence reports given UID. The problem of transmission interruption should be considered as the total number of the reports to be downloaded is about 14740[4] We use the incremental compuation strategy as in Step 2 (Recovery). Here is the detail.

First, we need to figure out what we have already downloaded. Since we store all the downloaded GenBank sequence reports in the directory `/usr2/jrwang/result` and use file names of the form LocusName-UID, it is easy to extract the locus names we already processed.[5] We remove duplicates also. Here is the script.

[3] We have to edit by hand the `map-locus` file before reading, because a network failure always leaves a closing curly bracket unprinted in this file.

[4] There was 13860 locus names but 14740 UID, because one locus name may correspond to more than one GenBank UID.

[5] If the network failed when we were processing a UID of locus name l, we must first delete all the files of the form l-UID from directory `/usr2/jrwang/result`.

```
primitive done ==
  list-uniquesort
    [ perl-extract-string (x,"^(.*)-(.*)","$1","\")
    | \x <--- process ("/usr/bin/ls", ["/usr2/jrwang/result"], "")
              using syscall-line ];
```

Next we need to figure out what locus names remain to be processed. These are simply those locus names in the set `address` but not in the list `done` above. Thus,

```
primitive not-yet-done ==
  { x | \x <- address,
        not (list-member (x.#locus, done)) };
```

Finally, to download the `not-yet-done` GenBank reports, we simply do

```
get-report (not-yet-done);
```

If everything goes fine, we are done at this point. If network fails again, simply repeat the above procedure.

4 Conclusion

The introduced programs successfully completed the work for downloading complete sequence reports according to locus names. It takes about one day to obtain all Genbank UID and about one and half days to download all reports due to large amount of data transmission and network busy during the process of extracting GenBank UID and downloading complete sequence reports. These programs can be applied to similar downloading tasks.