A PROTEIN TYROSINE PHOSPHATASE WAREHOUSE
A Kleisli/CPL Tutorial

Limsoon Wong

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

16 October 1996

Kleisli/CPL Version D Release 3 September 1996
BioKleisli Experimental Version Release 16 October 1996

1 Introduction

Kleisli/CPL [5, 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 "monitor" a combination of databanks for newly arrived data, as in warehouse and electronic "newsletter" applications.

In this tutorial, I will show how BioKleisli can be used to implement such customized warehouses and electronic "newsletters." I will illustrate the task by building a tyrosine phosphatase warehouse suggested to me by Catherine Pallen.[1]

The tyrosine phosphatase warehouse is constructed by regularly checking the amino acid portion of GenBank at NCBI [4] for new tyrosine phosphatase entries. I will not be storing entire entries into my warehouse. I will only be storing the accession number and the amino acid sequence of these entries — illustrating BioKleisli's unique ability in extracting data from complex ASN.1 structures used by GenBank in an extremely simple manner.

I will also take this opportunity to demonstrate the power of BioKleisli by showing several interesting variations of such a warehouse. In particular, I will show how to watch for sequences that exhibit certain motifs, for sequences that contain certain number of domains, performing multiple alignment, and so on.

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 [7] for a detailed description of CPL and [2, 3] for an introduction to the underlying theory.

[1] I would like to thank Catherine Pallen for suggesting the tyrosine phosphatase warehouse and for introducing me to motif in amino acid sequences.

2 Overview

The Tyrosine Phosphatase Warehouse contains three sets of data files derived by monitoring GenBank for tyrosine phosphatase entries. These data sets are then used to answer several interesting queries in tyrosine phosphatase research, mentioned to me by Catherine Pallen. Figure 1 is a picture of the organization of our warehouse.

The file `summaries` contains a set of summary records showing the tyrosine phosphatase entries found in GenBank up to the last warehouse update. The file `delta` contains a set of summary records showing the new tyrosine phosphatase entries found in GenBank during the current warehouse update. These two files have exactly the same structure. The file `sequences` is really a Unix directory of files containing actual amino acid sequence of tyrosine phosphatase entries. In GenBank, each entry has a unique identifier, denoted `uid`. The files in the directory `sequences` are named using the `uid`'s of the sequences they store. For example, the amino acid sequence whose `uid` is 5281 is stored in the file `sequences/5281`.

The file `mk-warehouse` is a script for searching GenBank and updating the warehouse data files. It is a simple Kleisli/CPL program embedded inside a C-script. The file `vw-warehouse` is a script for viewing the warehouse `summaries` and `sequences` files, looking for those having special patterns. The file `vw-update` is a script for view the warehouse `delta` and `sequences` files, looking for new entries having special patterns. Both of these are Kleisli/CPL programs. Other kind of viewing or data mining programs are possible too. One example is the script in the file `vw-align`, which is for performing multiple alignment on the receptor sequences found in the warehouse.

In order to update the warehouse, we simply run `mk-warehouse`. In fact, if you add a line like `source mk-warehouse` to your Unix `.login` script, the warehouse will be updated whenever you login to your machine.

To answer a query, we simply run one of `vw-warehouse`, `vw-update`, or other adhoc programs depending on the query. We normally run a query through a Netscape-like query interface.

In the rest of this tutorial, I will show you what the data files in the warehouse look like. I will also show and explain to you each of the four program files and how you can build a Netscape-like interface to them. Hopefully, by the end of this tutorial, you can figure out for yourself how to do more sophisticated queries.

```
                      +------------+
GenBank  -----------> |mk-warehouse|
                      +------------+
                             |
             +---------------+---------------+
             |               |               |
             V               V               V
          summaries      sequences         delta
             |               |               |
             +---------------+---------------+
                             |
                             V
                       +-------------+
                       |vw-warehouse/|
     requests -------> |  vw-update/ | -----------> results
                       |   vw-align  |
                       +-------------+
```

Figure 1: A simple tyrosine phosphatase warehouse

3 Summaries and Sequences

The `summaries` and `delta` files have the same structure. So I will just show you the first one. It is just a set of records that look like that given below. Each record has three fields. The `uid` field is a number that uniquely identifies the entry in GenBank. The `accession` field is the usual accession number used biologists. The `title` field is a short description title of the entry.

```
{ (#uid: 5281,
   #accession: "SPPYP2 cds1",
   #title: "protein-tyrosine phosphatase"),
. . . }
```

For each record in `summaries`, there will be a file in the `sequences` directory storing the amino acid sequence associated with that record. For example, there is a file `sequences/5281` containing the sequence below, corresponding to the record above.

```
MLHLLSKDEFNSTLKS...YDVVDYLNSGVNQASKPLMT
```

The data in these warehouse files are extracted from GenBank. The structure of these data as shown above is very simple and is thus very easy to use. This simplicity is in total contrast to the data in their original form in GenBank, which is complex beyond description — you will see a partial sample of it soon enough in Example 4.3.

It is therefore very remarkable that the Kleisli/CPL program, `mk-warehouse`, which is responsible for constructing these warehouse files is extremely simple and is only 20 lines in length! An exposition of this program is the subject of the next section.

4 Mk-Warehouse

The script file `mk-warehouse` is shown in Figure 2. We have numbered its lines for easy referencing, as we attempt to explain what the script is doing. Let us first briefly explain the script line by line.

- Line 1. It says this file is a C-shell script.
- Line 2. It invokes Kleisli/CPL and passes it everything between Line 3 and Line 24 for execution.
- Line 3. It asks Kleisli/CPL to load the definitions in the file `load-entrez.cpl`. This is the first BioKleisli library that is used in our tutorial. This library is used because we need two of its routines to access GenBank. The two routines are
  - `enrez-get-aa-summary` and
  - `entrez-get-aa-by-uid`.

  The former is for extracting GenBank amino acid summaries given some selection criteria. The latter is for extracting GenBank amino acid full records given their GenBank identification number.
- Lines 4-10. Summaries of tyrosine phosphatase entries are retrieved from GenBank (Lines 8, 9). The `uid`, `title`, and `caption` fields of each summaries are extracted (Lines 5, 6, 7). These fields are written to a file `new-summaries` (Line 10). Note that we have renamed the `caption` field to `accession` to better reflect its meaning (Line 7).
- Lines 11-14. The summary entries from Line 10 are read in and assigned the name `new-summaries` (Line 11). The summary entries upto the last warehouse update are read in and assigned the name `old-summaries` (Line 12). The difference between these two sets is computed (Line 13) and written to the file `delta` (Line 14). That is, `delta` contains the "new" entries found during this warehouse update.
- Lines 15-23. The `delta` entries are read in (Line 15). For each of these new entries `x` (Line 17), `entrez-get-aa-by-uid` is invoked on the uid to obtain the corresponding full GenBank report `y` (Line 18). The amino acid sequence is extracted from the report and assigned the name `s` (Lines 20–23). It is written to the `sequences` directory (Line 16).

It is worth going into Lines 5-9 and 16-24 more carefully, as they illustrate the use of our two BioKleisli library routines. We will explain them using a sequence of examples. I assume that we have already gotten the Kleisli/CPL system started up and the `load-entrez.cpl` library script loaded.

```
#!/usr/bin/csh
cpl <<END
usescript  "load-entrez.cpl";
writefile
  {(#title: x.#title,
    #uid: x.#uid,
    #accession: x.#caption) |
    \x <- entrez-get-aa-summary
              "(word tyrosine) and (word phosphatase)" }
  to "new-summaries" using stdout;
readfile newsummaries from "new-summaries" using stdin;
readfile oldsummaries from "summaries" using stdin;
writefile newsummaries set-diff oldsummaries
  to "delta" using stdout;
readfile delta from "delta" using stdin;
{ appendtofile ("sequences/" ^ (num-stringify (x.#uid)), s) |
    \x <- delta,
    <#seq: \y> <- entrez-get-aa-by-uid (x.#uid),
    \s <- case y.#inst.#seq_data
          of <#ncbieaa: \s> => {s}
          or <#iupacaa: \s> => {s}
          otherwise { }  };
END
mv new-summaries.val summaries.val
```

Figure 2: The mk-warehouse program.

Example 4.1 *The function* `entrez-get-aa-summary` *expects as its input a selection condition that consists of a Boolean combination of index-value pairs and returns a set of GenBank amino acid summaries that satisfy the selection condition. The valid Boolean operators are* `and`, `or`, *and* `butnot`. *The valid indices are* `word`, `keyword`, `accession`, `feature`, `mesh`, `author`, `journal`, `organism`, `gene`, `protein`, *and* `ecno`. *These indices mirror that of Entrez and have the corresponding meaning. The set of records returned have many fields, amongst them is a field named* `#uid` *which uniquely identifies a record in GenBank. Thus in the Kleisli/CPL program below, summaries of amino acid entries in GenBank containing the word tyrosine and phosphatase are returned. We have shown only the first record in the result set.*

```
entrez-get-aa-summary "(word tyrosine) and (word phosphatase)";

Result: { (#uid: 5281, #weight: 0, #no_abstract: false,
           #translated_title: false, #no_authors: false,
           #caption: "SPPYP2 cds1", #title: "protein-tyrosine phosphatase"),
         . . . }
```

Example 4.2 *As you can see, the records returned previously have 7 fields. Only three of them are useful for our purpose. They are the* `uid` *field, which is the unique GenBank identifier of the record; the* `caption` *field, which is the accession number of the record, and the* `title` *field, which a short desriptive title of the record. So it would be nice if we can get the system to automatically discard the unneeded fields. It would also be nice if we can rename the* `caption` *field to* `accession`. *The Kleisli/CPL program below does just that by using a* **set comprehension**. *We have shown only the first record of the result set.*

```
{ (#title: x.#title, #uid: x.#uid, #accession: x.#caption) |
      \x <- entrez-get-aa-summary "(word tyrosine) and (word phosphatase)" }

Result: { (#uid: 5281, #accession: "SPPYP2 cds1",
           #title: "protein-tyrosine phosphatase"),
         . . . }
```

Example 4.3 *The function* `entrez-get-aa-by-uid` *expects as its input a* `uid` *and returns as output a singleton set containing the complete GenBank record thus uniquely identified. In the Kleisli/CPL program below, the full GenBank report of the amino acid entry in GenBank uniquely identified by the* `uid` *5281 is returned. We show only parts of the result.*

```
entrez-get-aa-by-uid (5281);

Result: { <#seq: (#id: ..., #descr: { <#update_date: ...>, <#pub: ...>, <#pub:
...>, <#source: ...>, <#name: "SPPYP2 cds1">, <#title: "protein- tyrosine
phosphatase">}, #inst: ( #repr: "raw", #mol: "aa", #length: <#some: 711>,
#fuzz: ..., #topology: "linear", #strand: <#none:()>, #seq_data: <#iupacaa:
"MLHLLSKDEFNSTLK...VDYLNSGVNQASKPLMT">, #ext: ..., #hist: ..., #annot: { (#id:
{}, #db: ..., #name: ..., #desc: {}, #data: <#ftable: { ( #id: ..., #data:
<#cdregion: ( #orf: ..., #frame: "one", #conflict: ..., #gaps: ..., #mismatch:
..., #code: { <#id: 1>}, #code_break: [], #stops: ...)>, #partial: ...,
#except: ..., #comment: ..., #product: <#whole: <#gi: 5281>>, #location: <#int:
...>, #qual: [], #title: ..., #ext: ..., #cit: ..., #exp_ev: ..., #xref:{ ...},
```

```
#dbxref: { ( #db: "SWISS-PROT", #tag: <#str: "P32586">)}), ( #id: ..., #data:
<#prot: (#name: {"protein-tyrosine phosphatase"}, #desc: ..., #ec: {"3.1.3.48"},
#activity: {}, #db:{}, #processed: "not-set")>, #partial: ..., #except: ...,
#comment: ..., #product: ..., #location: <#whole: <#gi: 5281>>, #qual: [],
#title: ..., #ext: ..., #cit: ..., #exp_ev: ..., #xref: {}, #dbxref: {})}>)})>}
```

Example 4.4 *As you can see, the record returned above is very complex. We actually do not wish to see the whole record. Rather, we would only like to extract the actual amino acid sequence itself and discard the rest of the record. The program below does this by doing* **structural pattern matching**. *The "..." in the program is a syntax to tell Kleisli/CPL to ignore record fields that are not mentioned in the pattern.*

```
{ s | <#seq: (#inst: (#seq_data: <#iupacaa: \s>, ...), ...)> <-
                entrez-get-aa-by-uid 5281 };

Result: { "MLHLLSKDEFNSTLKS...YDVVDYLNSGVNQASKPLMT" }
```

Example 4.5 *We can also do the above over all the records in the warehouse, by substituting the* uid*'s found there for 5281. This is easily done by first reading the warehouse and then performing a set comprehension over it, as shown below. I have shown only the first two sequences in the output.*

```
{ s | \x <- newsummaries,
      <#seq: (#inst: (#seq_data: <#iupacaa: \s>, ...), ...)> <-
                entrez-get-aa-by-uid (x.#uid) };

Result: { "MLHLLSKDEFNSTLKS...YDVVDYLNSGVNQASKPLMT",
          "MDSWFILVLLGSGLIC...FCYKVVQEYIDAFSDYANFK",
         ... }
```

Example 4.6 *Don't you find the above output a bit confusing? It is hard to tell which sequence comes from which entry. Let me show you how to do it better, by returning both the sequence and its* uid.

```
{ (#uid: x.#uid, #sequence: s) |
           \x <- newsummaries,
           <#seq: (#inst: (#seq_data: <#iupacaa: \s>, ...), ...)> <-
                entrez-get-aa-by-uid (x.#uid) };

Result: { (#uid: 5281, #sequence: "MLHLLSKDEFNSTLKS...YDVVDYLNSGVNQASKPLMT"),
          (#uid: 32067, #sequence: "MDSWFILVLLGSGLIC...FCYKVVQEYIDAFSDYANFK",
         ... }
```

Example 4.7 *Let us put all these sequences into a warehouse. Recall that we want to store each new sequence into a file* sequences/xxx *where* xxx *is the* uid *of that sequence. The Kleisli/CPL program below shows how this can be done.*

```
{ appendtofile ("sequences/" ^ (num-stringify (x.#uid)), s) |
          \x <- delta,
           <#seq: (#inst: (#seq_data: <#iupacaa: \s>, ...), ...)> <-
                entrez-get-aa-by-uid (x.#uid) };
```

5 Vw-Update

The script file `vw-update` is shown in Figure 3. In this script, we have a Kleisli/CPL program that provides a simple facility for inspecting tyrosine phosphatase updates to GenBank since the last time we loaded up our warehouse. Let us briefly explain the script line by line.

- Line 1. It instructs the Kleisli/CPL system to load the BioKleisli library `util-17aug96.cpl`. This library contains several useful routines for substring testing. We will be using the following two:

  - `util-all-words-substring` and
  - `util-no-word-substring`.

  The semantics of `util-all-words-substring`(X, Y) is to test whether all words in X appear as substrings of Y. The semantics of `util-no-word-substring`(X, Y) is to test whether no word in X appears as substring of Y. Both functions are case insensitive.

- Line 2. It tells Kleisli/CPL to load the BioKleisli library `perl-25oct96.cpl`. This library contains routines for interfacing to Perl [6] to perform string matching and manipulation. We use only one routine here:

  - `perl-match-file`

  The semantics of `perl-match-file`(F, P, D) is like this: F is the name of a file, P is a pattern specified using regular expression in the normal Perl way, and D is a delimiter character that does not occur in P. The result returned is a string containing all lines in F that has pattern P.

  For example, `perl-match-file("abc", "AA[EF]", "/")` has the same effect as the Perl program `perl -ne 'print if m/AA[EF]/' abc`, which prints all the lines in the file `abc` containing a substring `AAE` or `AAF`.

- Line 3. Reads the contents of the file `delta` into the variable `DB`. Recall that `delta` stores the difference between the current state of our warehouse and its previous state; that is, the most recent updates.

- Lines 4-11. It defines our main function `lastupdate-find`. This function expects as its input a pattern, a string of compulsory words, and a string of forbidden words (Line 4). It looks at all the entries `x` in `DB` (Line 7). It checks that all the compulsory words appear in the `title` field of `x` (Line 8) and that none of the forbidden words appears in the `title` field of `x` (Line 9). It also checks that the specified pattern appears in `x`'s sequence (Line 11). For each `x` that satisfies all these checks, it returns its `title`, `uid`, `accession`, and amino acid sequence (Lines 5, 6). For convenience, we have defined `f` to be the name of the file that stores the sequence of `x` (Line 10).

```
usescript' "util-17aug96.cpl";
usescript' "perl-25oct96.cpl";
readfile DB from "summaries" using stdin;
primitive lastupdate-find == (\motif, \compulsory, \forbidden) =>
  { (#title: x.#title, #accession: x.#accession, #uid: x.#uid,
     #sequence: file-stringify (f)) |
       \x <- DB,
       util-all-words-substring (compulsory, x.#title),
       util-no-word-substring (forbidden, x.#title),
       \f == "sequences/" ^ (x.#uid.num-stringify),
       not ("" = perl-match-file (f, motif, "#")) };
```

Figure 3: The vw-update program.

We illustrate the use of the `lastupdate-find` function by several examples.

Example 5.1 *Catherine Pallen is interested in tyrosine phosphatases that have an active site defined by the motif:* `[IV]HC..G.GR[ST]GT`*. Here each* `.` *stands for any single letter and* `[IV]` *stands for either the letter* `I` *or the letter* `V`*, similarly for* `[ST]`*. We can retrieve these entries for her easily.*[2]

```
lastupdate-find ("[IV]HC..G.GR[ST]GT", "", "");

Result: { (#title: "tyrosine phosphatase precursor",
           #accession: "HSHEPTP cds1",
           #uid: 32067,
           #sequence: "MDSWFILVLLGSGLICVSAN...YKVVQEYIDAFSDYANFK"),
           (#title: "protein-tyrosine phosphatase",
            #accession: "HSHRPTPU cds1",
            #uid: 32456,
            #sequence: "MRTLGTCLATLAGLLLTAA...YKFCYEVALEYLNSG"),
          ... }
```

Example 5.2 *We can also easily restrict the results above to those that are not precursor.*

```
lastupdate-find ("[IV]HC..G.GR[ST]GT", "", "precursor");

Result: {  (#title: "protein-tyrosine phosphatase",
            #accession: "HSHRPTPU cds1",
            #uid: 32456,
            #sequence: "MRTLGTCLATLAGLLLTAA...YKFCYEVALEYLNSG"),
          ... }
```

Example 5.3 *It is just as easy to restrict the results to those that are precursor.*

```
lastupdate-find  ("[IV]HC..G.GR[ST]GT", "precursor", "");

Result: { (#title: "tyrosine phosphatase precursor",
           #accession: "HSHEPTP cds1",
           #uid: 32067,
           #sequence: "MDSWFILVLLGSGLICVSAN...YKVVQEYIDAFSDYANFK"),
         ...}
```

Example 5.4 *Naturally, we can use all these parameters in combination. As an example, we can ask for all receptors that are not precursor. Notice that we have included* `non-receptor` *as a forbidden term, this is to illustrate the subtle fact that the presence of the word* `receptor` *in the title of an entry does not imply that it is for sure a receptor!*

[2] The file `delta` is usually empty since tyrosine phosphatase does not turn up that often in updates to GenBank. In our examples, we run our queries on the file `summaries` instead.

```
lastupdate-find  ("[IV]HC..G.GR[ST]GT", "receptor", "non-receptor precursor");

Result: {(#title: "receptor-linked protein tyrosine phosphatase",
          #accession: "DRODPTP99A cds1",
          #uid: 157298,
          #sequence: "MPRPQHHALLRAML...ETSGSGWESLETEFNLATTVQNLDIVG"),
         (#title: "receptor-linked protein tyrosine phosphatase",
          #accession: "DROTP10D cds1",
          #uid: 158645,
          #sequence: "MLYQLSKATTRIR...GKENIVGPAREMHDNEGYEDDEGIAESGM"),
         ...}
```

6 Vw-Warehouse

Catherine Pallen wants to identify all the tyrosine phosphatase sequences that contain the motif `[IV]HC..G.GR[ST]GT`. Furthermore, she wants only those that appears within the context of some domain has the pattern `E[FY]..[LI]` at its starting point and has the pattern `Q.....Q...Q` at most 10 base pairs from its ending point. As a last requirement, she wants to extract only these domains, as opposed to entire sequences.

The script file `vw-warehouse` is shown in Figure 4. In this script, we have a Kleisli/CPL program that implements Catherine's query is a simple way. Let me describe it in some detail below.

- Line 1. Sets `DB` to be the contents of `summaries`, our warehouse file of tyrosine phosphatase summary records.
- Line 2. Loads the BioKleisli library `util-17aug96.cpl` for the two useful substring testing routines, `util-all-words-substring` and `util-no-word-substring` described earlier.
- Line 3. Loads the BioKleisli library `perl-25oct96.cpl` for four sophisticated regular expression matching routines:
  - `perl-match-file`,
  - `perl-match-string`,
  - `perl-extract-string`, and
  - `perl-extractline-file-global`.

  You have already seen the first one. So I will explain only the remaining three. The function `perl-match-string` is similar to `perl-match-file`. That is, `perl-match-string`(S, P, D) assumes S is a long string to be search, P is the pattern to search for, and D is a delimiter character which does not appear in P. It returns those lines in S that has pattern P.

  The semantics of `perl-extract-string`(S, P, R, D) is like this: S is a string to be searched, P is the pattern to search for, R is a replacement string, and D is a delimiter letter that does not appear in P and R. It returns a new string obtained by substituting the first occurrence of P with R, if any. The pattern P and replacement R follows the normal Perl standard. For example, `perl-extract-string ("stupid computer!", (.*)(com)(.*), $1Windows95, "#")` returns the string `"stupid Windows95"`.

  The semantics of `perl-extractline-file-global`(F, P, R, D) is like this: F is the name of a file, P is a pattern to be matched, R is a replacement string, and D is a delimiter letter not in P and R. It returns a list of lines, each coming from the file F, but with all occurrences of P replaced by R. Again, P and R follows the normal Perl standard.
- Lines 4-7. This makes the name `viewwarehouse-motif` a shorthand for the pattern that defines Catherine's domain. Note the use of extremely powerful Perl-style regular expression for specifying the motif pattern.

- Lines 8-11. This makes `viewwarehouse-getdomains` a function for extracting Catherine's domains from a file `f` containing amino acid sequence (Line 1). It first breaks the sequence in `f` into lines such that each line `x` begins with the pattern `E[FY]..[LI]` (Line 10). It checks that line `x` contains the desired motif (Line 11) and returns the desired domain (Line 9).

  It is worth pointing out that we use **list comprehension** in the above function, as indicated by the square brackets `[` and `]`. As a result, the relative positions of the extracted domains are preserved.

- Lines 12-19. It defines a function `viewwarehouse` that accepts a string of compulsory words and a string of forbidden words (Line 12). It looks up each entry `x` in `DB` (line 15). It checks that its `title` contains all the compulsory words (Line 16) and none of the forbidden words (Line 17). Then its sets `f` to the name of the file that contains `x`'s amino acid sequence (Line 18). It tests to make sure that the sequence contains Catherine's motif (Line 19). Then it returns the `title`, `uid`, `accession`, and the domains of `x`.

Example 6.1 *The implementation of the function* `viewwarehouse-find` *is more general then required by Catherine. By specifying various compulsory and forbidden words, we will be able to extract domains from specific tyrosine phosphatase entries in the warehouse. For example, we can obtained the domains of patented tyrosine phosphatases as follow:*

```
viewwarehouse-find ("patent", "");

Result: {(#title: "Sequence 2 from patent US 5536636",
          #uid: 1611272,
          #accession: "US 5536636",
          #domains: ["EFESLQKQE...VHCSAGIGRTGTIIVID...QRSGMVQTEAQYKFIYVAIAQ"]),
         (#title: "Sequence 6 from patent US 5536636",
          #uid: 1611274,
          #accession: "US 5536636",
          #domains: ["EFESLQKQE...VHCSAGIGRTGTIIVID...QRSGMVQTEAQYKFIYVAIAQ"]),
        ...}
```

```
readfile DB from "summaries" using stdin;
usescript' "util-17aug96.cpl";
usescript' "perl-25oct96.cpl";
primitive viewwarehouse-motif ==
   "(.*)(E[FY]..[LI])" ^
   "(.*)([IV]HC..G.GR[ST]GT)" ^
   "(.*)(Q.....Q...Q)(.{0,10})(.*)";
primitive viewwarehouse-getdomains == \f =>
  [ perl-extract-string (x, viewwarehouse-motif, "$2$3$4$5$6$7", "#") |
      \x <--- perl-extractline-file-global(f, "(E[FY]..[LI])", "\n$1", "#"),
      not ("" = perl-match-string (x, viewwarehouse-motif, "#")) ];
primitive viewwarehouse-find == (\compulsory, \forbidden) =>
  { (#title: x.#title, #uid : x.#uid, #accession: x.#accession,
     #domains: viewwarehouse-getdomains (f)) |
        \x <- DB,
        util-all-words-substring (compulsory, x.#title),
        util-no-word-substring (forbidden, x.#title),
        \f == "sequences/" ^ (x.#uid.num-stringify),
        not ("" = perl-match-file (f, viewwarehouse-motif, "#")) };
```

Figure 4: The vw-warehouse program.

7 Vw-Align

In the last section, we see that it is easy to break the selected tyrosine phosphatase sequences into their domains. In this section, I show you how to do more with these domains — we can run ClustalW multiple alignment on them! The script `vw-align` shown in Figure 5 is a Kleisli/CPL that automate the whole process of selecting tyrosine phosphatase sequences, chopping them into domains, and aligning them.

- Line 1. Loads the BioKleisli `util-17aug96.cpl` library for the two substring testing routines already described: `util-all-words-substring` and `util-no-word-substring`.
- Line 2. Loads the BioKleisli `perl-25oct96.cpl` library for the four Perl-like regular expression routines described in the last section: `perl-extract-string`, `perl-extractline-file-global`, `perl-match-file`, and `perl-match-string`.
- Line 3. Loads the BioKleisli `clustalw-23oct96.cpl` library. This library contains a function that interfaces Kleisli/CPL to the popular ClustalW program for multiple alignment:

  - `clustalw-align-domains`.

  The function `clustalw-align-domains` expects three inputs: a directory name D, a file name F, and a set of domains S to be aligned. The set of domains to be aligned must be given as a set of records, each record having two fields `id` and `domain`, where `domain` is a string of amino or nucleic acid sequence and `id` is a short unique name identifying the sequence. The function works by first creating the file F in directory D by writing out the domains S in a format accepted by ClustalW. Then it runs ClustalW on the file. After that is completed, it returns the output of ClustalW as a string.
- Line 4. Sets `DB` to be the contents of the `summaries` file in our warehouse.
- Lines 5–6. Defines `align-motif` as a shorthand for Catherine Pallen's domains.
- Lines 7–10. Defines the function `align-getdomains`. This function receives the file `f` storing an amino acid sequence (Line 7). It splits the sequence into segments so that each segment `x` starts with `E[FY]..[LI]` (Line 9). It then eliminates those segment that does not have Catherine's pattern (Line 10). For those that passes the test, it return the associated domain (Line 8).
- Lines 11–18. Defines the function `align-find`. This function receives a string of compulsory words and a string of forbidden words (Line 11). It looks at each `x` in `DB` (Line 13). It rejects those that do not contain some of the compulsory words (Line 14). It also rejects those that contain any of the forbidden words (Line 15). For each that passes the two tests, it sets `f` to the file that stores its amino acid sequence (line 16). It then eliminates those that do not exhibit Catherine's pattern (Line 17). For each that remains, it extract the domains associated with Catherine's pattern (Line 18). Finally, the `title`, `uid`, `accession`, and domains are returned (Line 12).

- Lines 19–24. Defines the function `align-serialize` that assigns unique identifiers to the domains. This is necessary before the domains can be submitted to ClustalW for multiple alignment. The function accepts a list of domains (Line 19) in the format as produced by `align-find`. It generates the unique identifiers by prefixing an internal number to the `uid` of each domain.
- Lines 25–29. Defines the main function `align-doit`. It takes in a directory name, a file name, a string of compulsory words, and a strin gof forbidden words (line 25). It calls `align-find` to get all the domains (line 26). It uses `align-serialize` to assign unique identifiers to these domains (Line 27). Then it runs ClustalW to perform multiple alignment on them (Line 28). When that is completed, it returns the domains and the alignment results (Line 29).

The function `align-getdomains` is same as the function `vewwarehouse-getdomains` given in the previous section. But it is worth going to more details for the other functions defined here. We do this by means of examples.

Example 7.1 *The function* `align-find` *is almost the same as* `viewwarehouse-find`*. In fact, the former just unnests the field* `domains` *of the latter. That is, the* `domains` *field of the latter is a list of domains. The* `domain` *field of the former is a single domain. For comparison, let me invoke* `align-find` *using the same parameter as in Example 6.1.*

```
align-find ("patent", "");

Result: {(#title: "Sequence 2 from patent US 5536636",
          #uid: 1611272,
          #accession: "US 5536636",
          #domain: "EFESLQKQE...VHCSAGIGRTGTIIVID...QRSGMVQTEAQYKFIYVAIAQ"),
         (#title: "Sequence 6 from patent US 5536636",
          #uid: 1611274,
          #accession: "US 5536636",
          #domain: "EFESLQKQE...VHCSAGIGRTGTIIVID...QRSGMVQTEAQYKFIYVAIAQ"),
        ...}
```

Example 7.2 *The* `align-serialize` *function uses* **structural recursion** to assign unique identifier in a non-deterministic way. For more information on structural recursion, consult [3]. Here we use the following example to illustrate its effect. We first get all the patents' domains and then serialize them. Notice that we have just non-deterministically prefixed `uid`'s with some unique numbers.

```
align-serialize (align-find ("patent", ""));

Result: {(#id: "3:1611277",
          #domain: "EFETLQQQEC...VHCSAGIGRTGTFIVID...QRSGMVQTEAQYRFIYMAVQH"),
         (#id: "2:1611275",
          #domain: "EFESLQKQEV...VHCSAGIGRTGTIIVID...QRSGMVQTEAQYKFIYVAIAQ"),
         (#id: "1:1611274",
          #domain: "EFESLQKQEV...VHCSAGIGRTGTIIVID...QRSGMVQTEAQYKFIYVAIAQ"),
         (#id: "0:1611272",
          #domain: "EFESLQKQEV...VHCSAGIGRTGTIIVID...QRSGMVQTEAQYKFIYVAIAQ")}
```

```
usescript' "util-17aug96.cpl";
usescript' "perl-25oct96.cpl";
usescript' "clustalw-23oct96.cpl";
readfile DB from "summaries" using stdin;
primitive align-motif ==
  "(.*)(E[FY]..[LI])(.*)([IV]HC..G.GR[ST]GT)(.*)(Q.....Q...Q)(.{0,10})(.*)";
primitive align-getdomains == \f =>
  [ perl-extract-string (x, align-motif, "$2$3$4$5$6$7", "-") |
      \x <--- perl-extractline-file-global(f, "(E[FY]..[LI])", "\n$1", "-"),
      not ("" = perl-match-string (x, align-motif, "#"))];
primitive align-find == (\compulsory, \forbidden) =>
  { (#title: x.#title, #uid : x.#uid, #accession: x.#accession, #domain: y) |
        \x <- DB,
        util-all-words-substring (compulsory, x.#title),
        util-no-word-substring (forbidden, x.#title),
        \f == "sequences/" ^ (x.#uid.num-stringify),
        not ("" = perl-match-file (f, align-motif, "#")),
        \y <--- align-getdomains (f) };
primitive align-serialize == \domains =>
  let \I == (\x, (\y, \z)) => (y + 1,
        (#id: (y.num-stringify) ^ ":" ^ (x.#uid.num-stringify),
         #domain: x.#domain) +} z) in
  let \E == (0, {})
  in (set-sri @ (I, E) @ domains).#2;
primitive align-doit == (\dir, \file, \compulsory, \forbidden) =>
  align-find (compulsory, forbidden)          before (\matches =>
  align-serialize (matches)                   before (\domains =>
  clustalw-align-domains (dir, file, domains) before (\alignment =>
  (#domains: matches, #alignment: alignment))));
```

Figure 5: The vw-align program.

Example 7.3 *Having gotten these domains, let us test out ClustalW. We just stick the above into the input of* `clustalw-align-domains`*! Notice that we need to supply a scratch space "/tmp/foo" for ClustalW to make use of.*

```
clustalw-align-domains ("/tmp", "foo", align-serialize (align-find ("patent", "")));

Result: "CLUSTAL W(1.60) multiple sequence alignment

3_1611277       EFETLQQQECKLLYSRKEGQRQENKNKNRYKNILPFDHTRVVLHDGDPNEPVSDYINANI
2_1611275       EFESLQKQEVKNLHQRLEGQRPENKGKNRYKNILPFDHSRVILQGRDSNIPGSDYINANY
1_1611274       EFESLQKQEVKNLHQRLEGQRPENKGKNRYKNILPFDHSRVILQGRDSNIPGSDYINANY
0_1611272       EFESLQKQEVKNLHQRLEGQRPENKSKNRYKNILPFDHSRVILQGRDSNIPGSDYINANY
                ***.**.** * *  * **** *** ************.**.*.  * * * *******
..."
```

Example 7.4 *For completeness, let us run the whole thing.*

```
align-doit ("/tmp", "foo", "patent", "");

Result:
(#domains: {(#title: "Sequence 2 from patent US 5536636",
             #uid: 1611272,
             #accession: "US 5536636",
             #domain: "EFESLQKQEVKNLHQRLEGQRPENKSK..."),
            (#title: "Sequence 6 from patent US 5536636",
             #uid: 1611274,
             #accession: "US 5536636",
             #domain: "EFESLQKQEVKNLHQRLEGQRPENKGK..."),
           ...},
 #alignment: "CLUSTAL W(1.60) multiple sequence alignment

3_1611277       EFETLQQQECKLLYSRKEGQRQENKNKNRYKNILPFDHTRVVLHDGDPNEPVSDYINANI
2_1611275       EFESLQKQEVKNLHQRLEGQRPENKGKNRYKNILPFDHSRVILQGRDSNIPGSDYINANY
1_1611274       EFESLQKQEVKNLHQRLEGQRPENKGKNRYKNILPFDHSRVILQGRDSNIPGSDYINANY
0_1611272       EFESLQKQEVKNLHQRLEGQRPENKSKNRYKNILPFDHSRVILQGRDSNIPGSDYINANY
                ***.**.** * *  * **** *** ************.**.*.  * * * *******

...")
```

8 Going on the Web

9 Conclusion

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] C. Burks, M.J. Cinkosky, W.M. Fischer, P. Gilna, J.E. Hayden, G.M. Keen, M. Kelly, D. Kristofferson, and J.D. Lawrence. GenBank. *Nucleic Acids Research*, 20 Supplement:2065–9, 1992.

[5] 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.

[6] Ellie Quigley. *Perl by Example*. Prentice Hall, New Jersey, 1994.

[7] 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`.