SIMPLE MANIPULATION OF ENTREZ DATA
A Kleisli/CPL Tutorial

Limsoon Wong

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

23 April 1997

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

Contents

1 Introduction

Kleisli/CPL [8, 4] 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. In this tutorial, I will focus on using Kleisli/CPL for simple extraction and manipulation of Entrez data. Entrez [10] is a very important data source. It includes data from DDBJ [11], EMBL [9], GenBank [7], PIR [2], PDB [3], and so on.

This tutorial primarily concerns the use of a small part 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 [12] for a detailed description of CPL and [5, 6] for an introduction to the underlying theory. (We will help you by putting some footnotes on the syntax and such, whenever we remember.)

2 Preparation

We have a specially prepared BioKleisli library for this tutorial. You need to load it first. To do that, simply type the following at the `cpl >` prompt:

```
usescript' "/home/wlsdemo/cplscripts/util/tutorial-23apr97.cpl";
```

By the way, I have appended the file above to the end of this tutorial. This is to give the more courageous reader a chance to see how some of the BioKleisli library routines are implemented.

We are now ready to try out various ways to extract and manipulate data from Entrez. The rest of this tutorial is organized according to the functions provided by the above library, one section for each.

3 DNA Queries

Let us begin with the functions for getting DNA data out of Entrez.

3.1 dna-get-n-summaries

The function `dna-get-n-summaries` requires a pair (K, M) as its input. K is a selection string and M is a number. It returns any M dna entry summaries in Entrez that satisfy the selection K.

Example 3.1 *Get any 2 cyclin dna entry summaries in Entrez.*

```
dna-get-n-summaries ("word cyclin", 2);
```

The word `word` *in the selection string indicates that* `cyclin` *is to be treated as a word. So any entry whose report contains the word* `cyclin` *is considered to have satisfied the selection condition. This kind of selection condition has good recall perfomance, but some irrelevant entries can slip in.*

The output of the example looks like this

```
{(#accession: "X68321",
  #uid: 9,
  #title: "B.taurus mRNA for cyclin A"),
 (#accession: "X69099",
  #uid: 2576,
  #title: "C.gloeosporioides mRNA for dispensible cyclin")}
```

It has two records, each with three fields. The `#accession` *field*[1] *is the accession number used to reference the record in GenBank. The* `#uid` *field is the unique identifier (uid) assigned to the record by Entrez. The* `#title` *field is a short description of the record. We will be using the uid quite a bit later.* □

Example 3.2 *Get any 3 vassopressin promoter region dna summaries from Entrez.*

```
dna-get-n-summaries ("(feat promoter) and (word vasopressin)", 3);
```

The word `feat` *in the selection string indicates that* `promoter` *is to be treated as a 'feature.' Every Entrez entry has an associated 'feature table' that records interesting features of the sequence entry. By indicating that* `promoter` *is to be treated as a feature, only those entries whose feature table contain the word* `promoter` *can satisfy the selection condition. This kind of selection condition increases precision performance, but may miss some relevant entries.* □

Example 3.3 *Get any 4 non-human protein kinase dna summaries from Entrez.*

[1] Kleisli/CPL syntax: The name of a record field is always marked by a #.

```
dna-get-n-summaries ("(protein kinase) butnot (word human)", 4);
```

The word `protein` *in the selection string indicates that* `kinase` *is to be treated as the name of a protein. So only those Entrez entries that explicitly say their sequences encode protein kinase can qualify. Thus such a selection string has high precision but poor recall. Had we use* `word kinase` *instead, then any entry having kinase activity would qualify, leading to good recall but low precision.* □

3.2 dna-get-summaries

The function `dna-get-summaries` requires a selection string K as its input. It returns all dna entry summaries in Entrez that satisfy K.

Example 3.4 *Get all protein kinase dna summaries from Entrez.*

```
dna-get-summaries "protein kinase";
```

The above only produces a small number of summaries, even though there are a lot of other entries that contain kinase. Why? and how do you get the other kinase entries out of Entrez? □

Example 3.5 *Get all promoter dna summaries created during March 1997 from Entrez.*

```
dna-get-summaries "(feat promoter) and (date '1997/03/...')";
```

The word `date` *indicates that* `1997/03/...` *is to be treated as a date. The* `...` *indicates partial match is okay. Any record created or updated during the specified date can satisfy this condtion.* □

3.3 dna-get-summaries-by-uid

The function `dna-get-summaries-by-uid` requires a uid U as input. It returns (most of the time!) the Entrez dna entry summary uniquely identified by U.

Try it out using some uids from the output of the previous examples. Here is one:

Example 3.6 *Get the Entrez dna entry summary whose uid is 1429197.*

```
dna-get-summaries-by-uid 1429197;
```

□

3.4 dna-get-reports-by-uid

The function `dna-get-reports-by-uid` requires a uid U as input. It returns the Entrez dna entry report uniquely identified by U.

Sometimes the uid may not uniquely identify a single report. This happens when two or more existing entries were discovered to be part of the same sequence (eg. one is the first exon and the other is the second exon) and a new entry is made to chain them together.

Example 3.7 *Get the Entrez dna entry report whose uid is 1429197.*

```
dna-get-reports-by-uid 1429197;
```

The Entrez dna entry report has a very complicated structure. I will just show you a few lines from the output to illustrate.[2]

```
{(#id: {<#embl: (#name: <#some: "ACMNPVPNK">,
                 #accession: <#some: "X99292">,
                 #release: <#some: "48">,
                 #version: <#some: 1>)>,
        <#general: (#db: "NID",
                    #tag: <#str: "e997514">)>,
        <#gi: 1429197>},
  #descr: {<#user: (#class: <#some: "gbfix">,
                    #type: <#id: 1>,
                    #data: [])>,
           <#title: "Autographa californica nuclear polyhedrosis ...">,
           <#embl: (#class: "standard",
                    #div: <#some: "vrl">,
                    #creation_date: <#std: (#year: 1996,
...
```

There are a lot more lines after the above, showing publication records, feature tables, dna sequence, and so on. We will come to getting information out of this complicated report later—it is in general necessary to know the structure of Entrez report to correctly extract information. □

So far, we have seen simple retrieval. It is time to introduce something a bit more sophisticated. We will do a little real programming to get the reports of dna entries identified by some selection conditions, as opposed to uid. For this, we just need to combine what we saw in previous subsections with `dna-get-reports-by-uid`.

Example 3.8 *Get reports on any 2 Entrez dna entries about cyclin.*

[2] Kleisli/CPL syntax: Elements of a sets are enclosed by { and } brackets. Records are enclosed by `(` and `)` brackets and record fields are written as name-value pairs like `#f : v`, where `#f` is the field name and `v` is the field value. Variants are enclosed by `<` and `>` brackets and the value of a variant is tagged by its kind like `#k: v`, where `#k` indicates the kind of the variant and `v` is the value of that variant.

```
{ (#summary: x,
   #report: y)
| \x <- dna-get-n-summaries ("word cyclin", 2),
  \y <- dna-get-reports-by-uid (x.#uid) };
```

The above expression[3] *returns a set of records* `(#summary: x, #report: y)` *such that each* `x` *is an Entrez dna entry summary about cyclin and each* `y` *is the report identified by* `x`*'s uid.* □

Example 3.9 *Get reports on all non-human protein kinase dna entries in Entrez.*

```
{ (#summary: x,
   #report: y)
| \x <- dna-get-summaries "(protein kinase) butnot (word human)",
  \y <- dna-get-reports-by-uid (x.#uid) };
```

□

It is also boring to see the whole report every time. You can easily select any component of it. We do a simple one below.

Example 3.10 *Get dates of update on all all non-human protein kinase dna entries in Entrez.*

```
{ (#summary: x,
   #update-date: d)
| \x <- dna-get-summaries "(protein kinase) butnot (word human)",
  \y <-  dna-get-reports-by-uid (x.#uid),
  <#update_date: \d> <- y.#descr };
```

□

The above[4] requires explicit knowledge of where dates of update can be found in a dna entry report. This is not very good. So our BioKleisli library also provides some functions to extract useful data from such reports. These are described next.

[3]Kleisli/CPL syntax: `r.#f` means the value of the `#f` field of the record `r`. `f (e)` means apply the function `f` to the argument `e`; the brackets can be omitted in certain places if you are a lazy fellow like me. `{ A | \x <- B, C }` is the set such that an object `A` is in it if and only if we can find an object `x` in the set `B` so that the condition `C` is satisfied. In general, the value of `A` and `C` depends on the value of `x`. Lastly, I should mention that the layout of an expression is not important; so I generally put lots of white space and newlines and so on to make programs easier to read.

[4]Kleisli/CPL syntax: We have our first use of pattern matching here. `<#update_date: \d> <- y.#descr` finds those variants in `y`'s `#descr` field tagged as having the kind `#update_date` and uses the variable `d` to refer to the value of these variants.

3.5 dna-get-seq-from-report

The function `dna-get-seq-from-report` requires an Entrez dna entry report as input. It returns the dna sequence contained in that report.

Example 3.11 *Get sequences of all non-human protein kinase dna entries in Entrez.*

```
{ (#summary: x,
   #sequence: s)
| \x <- dna-get-summaries "(protein kinase) butnot (word human)",
  \y <- dna-get-reports-by-uid (x.#uid),
  \s <- dna-get-seq-from-report (y) };
```

The above produces a set of pairs `(#summary: x, #sequence: s)` *such that* `x` *is a non-human protein kinase dna entry summary from Entrez,* `y` *is a dna entry report corresponding to* `x`*, and* `s` *is the sequence found in* `y`*.* □

Example 3.12 *Get sequences of all non-human protein kinase dna entries in Entrez. Return only just the summaries of those that are greater than 2000 na in length.*

```
{ x | \x <- dna-get-summaries "(protein kinase) butnot (word human)",
      \y <- dna-get-reports-by-uid (x.#uid),
      \s <- dna-get-seq-from-report (y),
      string-length (s) > 2000 };
```

In the above `string-length` *is a function provided in the standard libraries of Kleisli/CPL that return the length of its input string.* □

Example 3.13 *Get sequences of all non-human protein kinase dna entries in Entrez. Return only just the summaries of those that are greater than 2000 na in length and contain a stretch of 7 T's.*

```
{ x | \x <- dna-get-summaries "(protein kinase) butnot (word human)",
      \y <- dna-get-reports-by-uid (x.#uid),
      \s <- dna-get-seq-from-report (y),
      string-length (s) > 2000,
      string-issubstring (s, 0, "TTTTTTT") };
```

In the above `string-issubstring` *is a function provided in the standard libraries of Kleisli/CPL. It expects a triple* (S, P, T) *as input. It returns true if and only if the string* T *appears somewhere as a substring of the string* S *after ignoring the first* P *characters of* S*.* □

3.6 dna-get-annotations-from-report

The function `dna-get-annotations-from-report` requires an Entrez dna entry report as input. It returns a set containing all the descriptive annotations in the report.

Example 3.14 *Get descriptive information about any 2 promoters submitted to Entrez during March 1997.*

```
{ (#summary: x,
   #annotations: a)
| \x <- dna-get-n-summaries ("(feat promoter) and (date '1997/03/...')", 2),
  \y <- dna-get-reports-by-uid (x.#uid),
  \a == dna-get-annotations-from-report (y) };
```

If you run the above program[5]*, you will notice that all the annotations found in the same report are grouped together. Why?* □

3.7 dna-get-muids-from-report

The function `dna-get-muids-from-report` requires an Entrez dna entry report as input. It returns a set of numbers representing all the MedLine identifiers (muid) found in the report. These muids can be used later to bring back the associated MedLine abstracts.

Example 3.15 *Get MedLine identifiers from 'small-scale' Entrez dna entry reports.*

```
{ (#summary: x,
   #muids: m)
| \x <- dna-get-summaries "word small-scale",
  \y <- dna-get-reports-by-uid (x.#uid),
  \m ==  dna-get-muids-from-report (y) };
```

□

Example 3.16 *Get protein kinase entries in Entrez that has more than 1 MedLine citation.*

```
{ (#summary: x,
   #muids: m)
| \x <- dna-get-summaries "protein kinase",
  \y <- dna-get-reports-by-uid (x.#uid),
  \m ==  dna-get-muids-from-report (y),
  set-numitems (m) > 1 };
```

Here `set-numitems` *is a function in the standard libraries of Kleisli/CPL that returns the cardinality of its input set.* □

[5] Kleisli/CPL syntax: `\a == e` is a shorthand for `\a <- { e }`. It is thus just a way to say that the name `a` can now be used to refer to the object `e`, within the scope of this expression. You can also read it informally to mean `a` "equals" `e`.

4 MedLine Queries

Let us now look at some functions for obtaining MedLine data.

4.1 ml-get-reports-by-muid

The function `ml-get-reports-by-muid` requires a MedLine identifier (muid) as input. It returns the set of MedLine reports identified by that muid.

Example 4.1 *Get the MedLine report identified by muid 93360826.*

```
ml-get-reports-by-muid 93360826;
```

□

Example 4.2 *Get MedLine reports cited in protein kinase dna entries in Entrez.*

```
{ (#summary: x,
   #medline: m)
| \x <- dna-get-summaries "protein kinase",
  \y <- dna-get-reports-by-uid (x.#uid),
  \m == { r | \muid <- dna-get-muids-from-report (y),
              \r <- ml-get-reports-by-muid (muid) } };
```

□

4.2 ml-get-abstracts-from-report

The function `ml-get-abstracts-from-report` requires a MedLine report as input. It returns the set of abstracts contained in the report.

Example 4.3 *Get MedLine abstracts identified by muid 93360826.*

```
{ y | \x <- ml-get-reports-by-muid 93360826,
      \y <- ml-get-abstracts-from-report (x) };
```

□

Example 4.4 *Get MedLine abstracts cited in protein kinase dna entries in Entrez.*

```
{ (#summary: x,
   #medline: m)
| \x <- dna-get-summaries "protein kinase",
  \y <- dna-get-reports-by-uid (x.#uid),
  \m == { a | \muid <- dna-get-muids-from-report (y),
              \r <- ml-get-reports-by-muid (muid),
              \a <- ml-get-abstracts-from-report (r) } };
```

□

5 Amino Acid Queries

We move on to amino acid data. The functions provided for them are very similar to those provided for dna data. So I will be kind of brief `:-)`

5.1 aa-get-n-summaries

The function `aa-get-n-summaries` requires a pair (K, M) as its input. K is a selection string and M is a number. It returns any M amino acid entry summaries in Entrez that satisfy the selection K.

Example 5.1 *Get any 5 soybean dwarf virus amino acid entry summaries from Entrez.*

```
aa-get-n-summaries ("(word soybean) and (word dwarf) and (word virus)", 5);
```

Amino acid entry summaries have exactly the same structure as dna entry summaries. In particular, each one has a `#uid` *field that uniquely identifies the entry.* □

5.2 aa-get-summaries

The function `aa-get-summaries` requires a selection string K as its input. It returns all amino acid entry summaries in Entrez that satisfy K.

Example 5.2 *Get all soybean dwarf virus amino acid entry summaries from Entrez.*

```
aa-get-summaries "(word soybean) and (word dwarf) and (word virus)";
```

□

5.3 aa-get-summaries-by-uid

The function `aa-get-summaries-by-uid` requires a uid U as input. It returns the set of Entrez amino acid entry summaries identified by U. Usually this set contains a single summary.

Example 5.3 *Get the Entrez amino acid entry summary identified by the uid 1545979.*

```
aa-get-summaries-by-uid 1545979;
```

□

5.4 aa-get-reports-by-uid

The function `aa-get-reports-by-uid` requires a uid U as input. It returns the set iof Entrez amino acid entry reports identified by U. Usually this set contains a single report. The structure of such reports is analogous to those of dna entry reports.

Example 5.4 *Get the Entrez amino acid entry report identified by the uid 1545979.*

```
aa-get-reports-by-uid 1545979;
```

□

We can of course manipulate amino acid reports the same way we manipulated dna entry reports. So let us play with them a bit.

Example 5.5 *Get reports on any 2 Entrez amino acid entries about cyclin.*

```
{ (#summary: x,
   #report: y)
| \x <- aa-get-n-summaries ("word cyclin", 2),
  \y <- aa-get-reports-by-uid (x.#uid) };
```

□

The next one is a bit more sophisticated.

Example 5.6 *Get dates of update on any 2 Entrez amino acid entries about cyclin.*

```
{ (#summary: x,
   #update-date: d)
| \x <- aa-get-n-summaries ("word cyclin", 2),
  \y <- aa-get-reports-by-uid (x.#uid),
  <#update_date: \d> <- y.#descr };
```

□

5.5 aa-get-seq-from-report

The function `aa-get-seq-from-report` requires an Entrez amino acid entry report as input. It returns the amino acid sequence contained in that report.

Example 5.7 *Get reports on all three 'small-scale' proteins back from Entrez. Show the amino acid sequence only.*

```
{ (#summary: x,
   #sequence: s)
| \x <- aa-get-summaries "word small-scale",
  \y <- aa-get-reports-by-uid (x.#uid),
  \s <- aa-get-seq-from-report (y) };
```

□

The three sequences returned by the above example look the same to me. Let's test this out.

Example 5.8 *Eliminate duplicated sequences from the previous example.*

```
set-general-elimination
  @ #sequence
  @ { (#summary: x, #sequence: s)
    | \x <- aa-get-summaries "word small-scale",
      \y <- aa-get-reports-by-uid (x.#uid),
      \s <- aa-get-seq-from-report (y) };
```

Here `set-general-elimination` *is a function provided by the BioKleisli library of this tutorial. It works like this:*[6] `set-general-elimination @` P `@` S *returns a maximal subset* S' *of* S *so that every* x *and* y *are distinct in their* P *image. Thus* `set-general-elimination @ #sequence @` S *has the effect of eliminating duplicates from* S *based on the value of the* `#sequence` *field. So the program in this example first builds a set of summary-sequence pairs and gets rid of duplicates from this set based on sequence values.* □

5.6 aa-get-annotations-from-report

The function `aa-get-annotations-from-report` requires an entrez amino acid entry report as input. It returns a set containing all the descriptive annotations in the report.

Example 5.9 *Get descriptive information about any 2 PTP receptor amino acid entries in Entrez.*

```
{ (#summary: x,
   #annotations: a)
| \x <- aa-get-n-summaries ("(word PTP) and (word receptor)", 2),
  \y <- aa-get-reports-by-uid (x.#uid),
  \a == aa-get-annotations-from-report (y)};
```

□

[6] Kleisli/CPL syntax. `f @ e` is nother way of applying a function `f` to the argument `e`. In fact, `f @ e` is the standard syntax used in Kleisli/CPL. The alternative syntax `f (e)` is treated as a shorthand for `f @ e`. Similarly, `e.f` is treated as a shorthand for `f @ e`.

5.7 aa-get-muids-from-report

The function `aa-get-muids-from-report` requires an Entrez amino acid entry report as its input. It returns the set of MedLine identifiers (muid) contained in that report.

Example 5.10 *Get MedLine identifiers of 'small-scale' Entrez amino acid entries.*

```
{ (#summary: x,
   #muids: m)
| \x <- aa-get-summaries "word small-scale",
  \y <- aa-get-reports-by-uid (x.#uid),
  \m == aa-get-muids-from-report (y)};
```

□

Of course, we can also bring back MedLine abstracts straight away.

Example 5.11 *Get MedLine abstracts of 'small-scale' Entrez amino acid entries.*

```
{ (#summary: x,
   #abstracts: z)
| \x <- aa-get-summaries "word small-scale",
  \y <- aa-get-reports-by-uid (x.#uid),
  \z == { a | \m <- aa-get-muids-from-report (y),
              \r <- ml-get-reports-by-muid (m),
              \a <- ml-get-abstracts-from-report (r) } };
```

□

Looks like I have some space left in this page, so lets do the previous example in a more interesting way.

Example 5.12 *Get MedLine abstracts of 'small-scale' Entrez amino acid entries.*

```
{ (#summary: x,
   #abstracts: { a | \m <- aa-get-muids-from-report (y),
                     \r <- ml-get-reports-by-muid (m),
                     \a <- ml-get-abstracts-from-report (r) })
| \x <- aa-get-summaries "word small-scale",
  \y <- aa-get-reports-by-uid (x.#uid) };
```

The above program and the program in the previous example are equivalent.[7] *That is, they produce exactly the same output. I guess the nested nature of the output is more easily visible from this program than the previous equivalent one.* □

[7] Kleisli/CPL axiom: The equation `{ A | \x <- B, \y == C }` = `{ A' | \x <- B }` holds, where `A'` is obtained from `A` by replacing all free occurrences of `y` by `C`.

6 Input/Output

It is often useful to store your results and make use of them later. Here are some simple ways to do that.

6.1 writefile ... to ... using ...

The command `writefile` E `to` F `using` S writes the object E to the file F using output method S. Several output methods are available, but you probably only need to know `stdout`, which outputs in the Standard Kleisli Data Exchange Format.

Example 6.1 *Save the 'small-scale' amino acid entry reports to a file* `"smallprotein"` *for later use.*

```
writefile
  { (#summary: x,
     #report: y)
  | \x <- aa-get-summaries "word small-scale",
    \y <- aa-get-reports-by-uid (x.#uid) }
to "smallprotein" using stdout;
```

□

6.2 readfile ... from ... using ...

The command `readfile` X from F using S let us read a file F using the input method S and assign the contents the name X. That is, we can refer to the contents of F using the name X. There are several input methods, but we will only introduce you to `stdin` here, it is used for reading files produced by `stdout`.

Example 6.2 *Return the amino acid sequences in those reports stored in the file* `"smallprotein"`.

```
readfile db from "smallprotein" using stdin;
{ (#summary: x.#summary,
   #sequence: s)
| \x <- db,
  \s <- aa-get-seq-from-report (x.#report) };
```

□

7 Motifs

Very frequently we want to know what kind of famous motifs can be found in our protein sequences. A popular package for doing this is Prosite Scan [1].

7.1 prosite-scanseq

The function `prosite-scanseq` requires an amino acid sequence (capital letters only) as input. It returns the set of motifs found in that sequence. These motifs are taken from a Prosite Scan collection that we maintain.

Example 7.1 *What popular motifs can be found in the sequence* `"ARGKKRRSTTP"`*?*

```
prosite-scanseq "ARGKKRRSTTP";
```

The output should look like this:[8]

```
{(#name: "CAMP_PHOSPHO_SITE",
  #prositeid: "PS00004",
  #docid: "PDOC00004",
  #description: "cAMP- and cGMP-dependent protein kinase phosphorylation site",
  #pattern: "[RK]{2}.[ST]",
  #match: [(#1: 5,
           #2: "KRRS")]),
...}
```

It says that the subsequence `"KRRS"` *found at position 5 in our sequence matches the motif* `"[RK]{2}.[ST]"` *of a cAMP-dependent protein kinase phosphorylation site. The* `#prositeid` *and* `#docid` *fields tell us where to find out more about this motif from Prosite.* □

7.2 prosite-scanseq-fussy

The function `prosite-scanseq-fussy` requires an amino acid sequence (capital letters only) as input. It returns the set of motifs found in that sequence. These motifs are taken from a Prosite Scan collection that we maintain. But this function ignores motifs that occur too frequently.

Example 7.2 *What rare motifs can be found in the sequence* `"ARGKKRRSTTP"` *that we saw before?*

```
prosite-scanseq-fussy "ARGKKRRSTTP";
```

□

[8] Kleisli/CPL syntax: Elements of a list are enclosed by `[` and `]` brackets.

Let us try it on some real sequences. Recall from the previous section that the name `db` refers to the content of the file `"smallprotein"` and this file contains reports on 'small-scale' amino acid entries.

Example 7.3 *What rare motifs can be found in the 'small-scale' amino acid sequences? Omit those sequences that have no rare motif.*

```
{ (#summary: x.#summary,
   #motifs: m)
| \x <- db,
  \s <- aa-get-seq-from-report (x.#report),
  \m == prosite-scanseq-fussy (s),
  not (m = {}) };
```

□

By the way, we very much like to augment the Prosite motifs with some extra ones. If you know of any, please let me know!

8 Sequence Matching

One of the things we often do is to look for patterns in our sequence data. Here are some other ways to do this besides Prosite Scan!

8.1 string-islike

The function `string-islike` requires a pair of strings (S, T) as input. It returns true if and only if S looks like T. The letter `%`, when found in T, means an arbitrary gap.

Example 8.1 *Does the string* `"the quick brown fox jumps over the Lazy Dog"` *contains the letters* `ICK` *followed by* `Fox` *followed by* `lazy`*?*

```
string-islike (
  "the quick brown fox jumps over the Lazy Dog",
  "%ICK%Fox%lazy%");
```

If the result was not `true`*, you probably typed something wrong! Any way, you can deduce from this that* `string-islike` *is not case sensitive.* □

Let us try it out on some real data. We will use the amino acid data `db` from file of 'small-scale' amino acid entries again.

Example 8.2 *Do these 'small-scale' proteins contain a stretch of amino acids* `rlrnen` *followed by a stretch* `keen` *at the c-terminal?*

```
{ () } = { () | \x <- db,
                \s <- aa-get-seq-from-report (x.#report),
                string-islike (s, "%rlrnen%keen") };
```

How does the above work? Well, the set on the right-hand-side of the equal sign is nonempty and contains exactly `()` *if and only if one of the sequences in* `db` *has the desired pattern.*[9] □

So `string-islike` is good when you want to test for patterns involving gaps. But what if you want to test for patterns that are more sophisticated? For example, your pattern may have gaps of size between 2 and 5 letters long. For such patterns, we need more advanced functions.

8.2 string-perlmatch

The function `string-perlmatch` requires a pair of strings (S, P) as input. It returns true if and only if the pattern P is found in S. The comparison is case sensitive. The pattern P follows the usual Perl

[9] Kleisli/CPL syntax: `()` stands for the unique record that has no field.

convention. That is, `[xyz]` means either `x`, `y`, or `z`; `a{3,5}` means `a` repeated 3, 4, or 5 times; `.` means any character; `b*` means `b` repeated arbitrarily many times; and so on.

Example 8.3 *Does* `"abc[ef]{3,5}.*g"` *match* `"pabceefekkkbfdgkk"`*?*

```
string-perlmatch ("pabceefekkkbfdgkk", "abc[ef]{3,5}.*g");
```

□

Example 8.4 *Get EMC-B virus proteins that have the phosphorylase kinase site motif* `"[KR]..S[IV]"`.

```
{ x | \x <- aa-get-summaries ("word emc-b"),
      \y <- aa-get-reports-by-uid (x.#uid),
      \s <- aa-get-seq-from-report (y),
      string-perlmatch (s, "[KR]..S[IV]") };
```

As an interesting side exercise, if I want to know if all EMC-B virus proteins have that phosphorylase kinase site motif, how would you write the query? □

8.3 string-perlextract

The function `string-perlextract` requires a pair (S, P) as input. It returns a list of strings, representing those parts of S that have pattern P. Again, the comparison is case insensitive and P follows Perl's syntax.

Example 8.5 *Get EMC-B virus proteins that have the phosphorylase kinase site motif* `"[KR]..S[IV]"`. *Show their phosphorylase kinase "sites."*

```
{ (#sumamry: x,
   #sites: a)
| \x <- aa-get-summaries ("word emc-b"),
  \y <- aa-get-reports-by-uid (x.#uid),
  \s <- aa-get-seq-from-report (y),
  \a == string-perlextract (s, "[KR]..S[IV]"),
  not (a = []) };
```

□

8.4 string-align

The function `string-align` requires a pair of strings (S, T) and returns their minimal alignment using the identity matrix. The alignment program was implemented by Zhang Louxin.

Let us use a simple example to explain this function.

Example 8.6 *Align the two strings* "abcdefg" *and* "efgj".

```
string-align ("abcdefg", "efgj");
```

If you run the above, you should see something like this:

```
{(#Matching-Percentage: 75.0,
  #Match-Char: 3,
  #Mismatch: 0,
  #Indel: 5,
  #Seq1: "abcdefg-",
  #Seq2: "----efgj")}
```

The `#Matching-Percentage` *field tells you the sequence identity between the two input sequences, basically this is computed as the ratio of number matched characters to the length of the shorter input sequence. The* `#Match-Char` *field tells you how many aligned characters are perfect match. The* `#Mismatch` *field tells you how many aligned characters are mismatch. The* `#Indel` *field tells you the number of insertions and deletions are there in this alignment. The remaining two fields show places where insertions and deletions are made.* □

Let us try this function out on our EMC-B viruses.

Example 8.7 *Align EMC-B virus proteins to the kinase-like sequence* "KLTSI".

```
{ (#summary: x,
   #alignment: a)
| \x <- aa-get-summaries ("word emc-b"),
  \y <- aa-get-reports-by-uid (x.#uid),
  \s <- aa-get-seq-from-report (y),
  \a == string-align (s, "KLTSI") };
```

The alignment is not very good huh? Dont't worry so much in this tutorial. We are in the process of testing out various alternative alignment algorithms and software. □

8.5 string-align-pretty

The function `string-align` requires a triple (S, T, L), where S and T are the two strings to be aligned. It returns their minimal alignment using the identity matrix and prints this alignment out assuming your screen only has $L + 6$ columns.

If you are fussy with how the alignment is displayed, try this one!

Example 8.8 *Align EMC-B virus proteins to the kinase-like sequence* "KLTSI".

```
{ (#summary: x,
   #alignment: a)
| \x <- aa-get-summaries ("word emc-b"),
  \y <- aa-get-reports-by-uid (x.#uid),
  \s <- aa-get-seq-from-report (y),
  \a == string-align-pretty (s, "KLTSI", 60) };
```

□

This concludes our tutorial. I hope you will find the time to try out queries of your own.

References

[1] R. D. Appel, A. Bairoch, and D. F. Hochstrasser. A new generation of information retrieval tools for biologists: The example of the ExPASy WWW server. *Trends in Biochem. Sci.*, 19:258–260, 1994.

[2] W.C. Barker, D.G. George, L.T. Hunt, and J.S. Garavelli. The PIR protein sequence database. *Nucleic Acids Research*, 19:2231–2236, 1991.

[3] F. C. Bernstein, T. F. Koetzle, G. J. B. Williams, Jr. E. F. Meyer, M. D. Brice, J. R. Rodgers, O. Kennard, T. Shimanouchi, and T. Tasuni. The Protein Data Bank: A computer-based archival file for macromolecular structures. *Journal of Molecular Biology*, 112:535–542, 1977.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] D. G. Higgins, R. Fuchs, P. J. Stoehr, and G. N. Cameron. The EMBL data library. *Nucleic Acids Research*, 20:2071–2074, 1992.

[10] National Center for Biotechnology Information, National Library of Medicine, Bethesda, MD. *ENTREZ: Sequences Users' Guide*, 1992. Release 1.0.

[11] Y. Tateno, Y. Ugawa, Y. Yamazaki, H. Hayashida, N. Saitou, and T. Gojobori. The DNA data bank of Japan. In W. C. Barker, editor, *CODATA Bulletin: Information Integration for Biological Macromolecules*, pages 74–75. 1991.

[12] 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`.

A tutorial-23apr97.cpl

This is the contents of the file `tutorial-23apr97.cpl`. It shows you how the various BioKleisli library functions described in the main text are implemented in Kleisli/CPL.

```
!



! (c) Copyright 1997 by Limsoon Wong, BIC, and ISS.
!
! Programs used in
!   SIMPLE MANIPULATION OF ENTREZ DATA
!   A BioKleisli/CPL Tutorial
!   23 April 1997


! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Some general utilities programs
!
! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


!
! set-general-elimination @ P @ S
! returns a maximal subset S' of S such that
! for every x, y in S', P(x) is distinct from P(y).
!
primitive set-general-elimination == \P =>
  let \incr == (\x, \A) =>
         if set-thereis @ (\y => x.P = y.P) @ A
         then A
         else x +} A
  in set-sri @ (incr, {});


!
! Make a connection to Entrez in DC first.
!
entrez-add (#name: "entrez", #level: 1);


!
! dna-get-n-summaries ("organism human", 5)
! returns any 5 human dna entry summaries in Entrez.
!
primitive dna-get-n-summaries == (\keywd, \max) =>
  { (#accession: x.#caption,
     #uid: x.#uid,
     #title: x.#title)
```

```
  | \x <- process (
            #db: "na",
            #select: keywd,
            #path: "Seq-entry",
            #args: "-D -n " ^ max.num-stringify)
          using entrez };


!
! dna-get-summaries "word small-scale"
! returns all dna entry summaries in Entrez
! that contain the word ‘small-scale’.
!
primitive dna-get-summaries == \keywd =>
  { (#accession: x.#caption,
     #uid: x.#uid,
     #title: x.#title)
  | \x <- process (
            #db: "na",
            #select: keywd,
            #path: "Seq-entry",
            #args: "-D")
          using entrez };


!
! dna-get-summaries-by-uid 49401
! returns dna entry summaries in Entrez whose uid is 49401.
!
primitive dna-get-summaries-by-uid == \uid =>
  { (#accession: x.#caption,
     #uid: x.#uid,
     #title: x.#title)
  | \x <- process (
            #db: "na",
            #select: "",
            #path: "Seq-entry",
            #args: "-D -g " ^ uid.num-stringify)
          using entrez };


!
! dna-get-reports-by-uid 49401
! returns dna entry reports in Entrez whose uid is 49401.
!
primitive dna-get-reports-by-uid == \uid =>
  let \split == fix (\f => \r =>
          case r
          of <#seq: \S> => {S}
          or <#set: (#seq_set: \S, ...)> => { x | \s <--- S, \x <- f(s) })
  in { s | \r <- process (
```

```
                 #db: "na",
                 #select: "",
                 #path: "Seq-entry",
                 #args: "-g " ^ uid.num-stringify)
               using entrez,
          \s <- split (r),
           <#gi: uid> <- s.#id };


!
! dna-get-seq-from-report (A-REPORT)
! returns the dna sequence in the Entrez
! dna entry report A-REPORT.
!
primitive dna-get-seq-from-report == \report => force (
  case report.#inst.#seq_data
  of <#iupacna: \S> => { S}
  or <#ncbi2na: \S> => { S}
  or <#ncbi4na: \S> => { S}
  or <#ncbi8na: \S> => { S}
  or <#ncbipna: \S> => { S}
  otherwise {});


!
! dna-get-annotations-from-report (A-REPORT)
! returns the annotations in the Entrez dna
! entry report A-REPORT.
!
primitive dna-get-annotations-from-report == \report =>
  set-unique (
    het-multimatch ("*%#title", report.het-mk) {+}
    het-multimatch ("*##source.%#some", report.het-mk) {+}
    het-multimatch ("*##keywords.?", report.het-mk) {+}
    het-multimatch ("*##title.?.%#name", report.het-mk) {+}
    het-multimatch ("##inst.##annot.?.##data.%#ftable.?.##qual.?.##val",
                    report.het-mk));


!
! dna-get-muids-from-report (A-REPORT)
! returns the MedLine ids in the Entrez dna
! entry report A-REPORT.
!
primitive dna-get-muids-from-report == \report =>
  set-unique (
    { z | <#pub: (#pub: \x, ...)> <- report.#descr,
          \y <- x,
          \z <- case y
                of <#muid: \m> => {m }
                or <#gen: \g> => case g.#muid
```

```
                                 of <#some: \m> => {m }
                                 otherwise {}
                otherwise {} });


! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Programs for MedLine queries
!
! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


!
! ml-get-reports-by-muid 93360826
! returns the MedLine report uniquely identified by 93360826
!
primitive ml-get-reports-by-muid == \muid =>
  process (
    #db: "ml",
    #select: "",
    #path: "Medline-entry",
    #args: "-g " ^ muid.num-stringify)
  using entrez;


!
! ml-get-abstracts-from-report (A-REPORT)
! returns the MedLine abstract in MedLine report A-REPORT
!
primitive ml-get-abstracts-from-report == \report =>
  case report.#abstract
  of <#some: \x> => { x}
  otherwise { };


! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Programs for Amino acid queries
!
! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


!
! aa-get-n-summaries ("organism 'soybean dwarf virus'", 5)
! returns any 5 soybean dwarf virus amino acid
! entry summaries in Entrez.
!
primitive aa-get-n-summaries == (\keywd, \max) =>
  { (#accession: x.#caption,
     #uid: x.#uid,
     #title: x.#title)
```

```
  | \x <- process (
           #db: "aa",
           #select: keywd,
           #path: "Seq-entry",
           #args: "-D -n " ^ max.num-stringify)
         using entrez };


!
! aa-get-summaries "organism 'soybean dwarf virus'"
! returns all soybean dwarf virus amino acid
! entry summaries in Entrez.
!
primitive aa-get-summaries == \keywd =>
  { (#accession: x.#caption,
     #uid: x.#uid,
     #title: x.#title)
  | \x <- process (
           #db: "aa",
           #select: keywd,
           #path: "Seq-entry",
           #args: "-D")
         using entrez };


!
! aa-get-summaries-by-uid 1545979
! returns aa entry summaries in Entrez whose uid is 1545979.
!
primitive aa-get-summaries-by-uid == \uid =>
  { (#accession: x.#caption,
     #uid: x.#uid,
     #title: x.#title)
  | \x <- process (
           #db: "aa",
           #select: "",
           #path: "Seq-entry",
           #args: "-D -g " ^ uid.num-stringify)
         using entrez };


!
! aa-get-reports-by-uid 486711
! returns aa entry reports in Entrez whose uid is 486711.
!
primitive aa-get-reports-by-uid == \uid =>
  let \split == fix (\f => \r =>
         case r
         of <#seq: \S> => {S}
         or <#set: (#seq_set: \S, ...)> => { x | \s <--- S, \x <- f(s) })
  in { s | \r <- process (
```

```
                  #db: "aa",
                  #select: "",
                  #path: "Seq-entry",
                  #args: "-g " ^ uid.num-stringify)
                using entrez,
          \s <- split (r),
          <#gi: uid> <- s.#id };


!
! The ncbistdaa codes goes from 0 to 25.
!
primitive ncbistdaa-code == string-implode [ x.string-chr |
  \x <--- [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
           14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]];
!
! The map to amino acid symbols as follows.
! The 3 "?" are symbols that I couldnt determined yet.
!
primitive ncbistdaa-symbol == "?ABCDEFGHIKLMNPQRSTVWXYZ??";
!
! Here is the translation function.
!
primitive ncbistdaa-map == \x =>
  string-trans (ncbistdaa-code, ncbistdaa-symbol, x);


!
! aa-get-seq-from-report (A-REPORT)
! returns the aa sequence in the Entrez
! dna entry report A-REPORT.
!
primitive aa-get-seq-from-report == \report => force (
  case report.#inst.#seq_data
  of <#iupacaa: \S> => { S}
  or <#ncbi8aa: \S> => { S}
  or <#ncbieaa: \S> => { S}
  or <#ncbipaa: \S> => { S}
  or <#ncbistdaa: \S> => { ncbistdaa-map (S)}
  otherwise {});

!
! aa-get-uid-from-report (A-REPORT)
! returns the uid of the Entrez aa entry
! report  A-REPORT
!
primitive aa-get-uid-from-report == \report =>
  { x | <#gi: \x> <- report.#id };
```

```
!
! aa-get-annotations-from-report (A-REPORT)
! returns the annotations in the Entrez aa
! entry report A-REPORT.
!
primitive aa-get-annotations-from-report == dna-get-annotations-from-report;


!
! aa-get-muids-from-report (A-REPORT)
! returns the MedLine ids in the Entrez aa
! entry report A-REPORT.
!
primitive aa-get-muids-from-report == dna-get-muids-from-report;


! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Motifs
!
! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


!
! Paths to prosite_scan program and motif collection.
!
primitive prosite-scanpath == "/usr49/BIC/prosite_scan/prosite_scan2";
primitive prosite-motifpath == "/usr49/BIC/prosite_scan/prosite.reg";


!
! prosite-scanseq A-SEQ
! returns all the motifs found in A-SEQ.
! These motifs are drawn from the entire collection
! in prosite-motifpath.
!
primitive prosite-scanseq == \seq =>
  process (prosite-scanpath, [ "-i", prosite-motifpath], seq)
  using syscall-co;


!
! prosite-scanseq-fussy A-SEQ
! Same as above, expect it ignores motifs
! that occur too often.
!
primitive prosite-scanseq-fussy == \seq =>
  process (prosite-scanpath, [ "-s", "-i", prosite-motifpath], seq)
  using syscall-co;
```

```
! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! String Matching and alignment Routines
!
! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


!
! where to find Perl and Louxin's alignment program
!
primitive perl-path == "/opt1/bin/perl";
primitive align-path == "/usr49/bicdemo/bin/louxin";


!
! string-perlmatch (S, P)
! return true if and only pattern P is in S.
! P is a pattern in Perl syntax.
!
primitive string-perlmatch == (\string, \pattern) => not ("" =
  process (
    perl-path,
    ["-ne", "print if m#" ^ pattern ^ "#"],
    string)
  using syscall-string);


!
! string-perlextract (S, P)
! extracts all substrings in S that match pattern P.
! P is a pattern in Perl syntax.
!
primitive string-perlextract == (\string, \pattern) =>
  [ y | \x <--- process (
                  perl-path,
                  ["-ne", "print if s#.*?(" ^ pattern ^ ")#-$1\n#g"],
                  string)
                using syscall-line,
        string-isprefix ("-", x, 0),
        \y == string-suffix (x, 1) ];


!
! string-align  ("abcdefg", "efgj")
! return {(#Matching-Percentage: 75.0,
!          #Match-Char: 3,
!          #Mismatch: 0,
!          #Indel: 5,
!          #Seq1: "abcdefg-",
!          #Seq2: "----efgj")}
! This uses Zhang Luoxin's alignment program.
```

```
!
primitive string-align == (\S, \T) => force {
  process
    (align-path ^ "/minalig3",
     [],
     (S.string-length + T.string-length).num-stringify ^
     string-newline ^ S ^ string-newline ^ T ^ string-newline)
  using syscall-co };


!
! This function is same as string-align,
! but its prints its output more prettily.
!
primitive string-align-pretty ==  (\S, \T, \l) =>
  let \match == fix (\F => (\c, \A, \P, \Q) =>
         if string-span (P, c, c) = ""
         then string-implode (A)
         else if string-span (P, c, c) =  string-span (Q, c, c)
         then F (c + 1, A [+] ["|"], P, Q)
         else F (c + 1, A [+] [" "], P, Q))  in
  let \pretty == fix (\F => (\c, \A, \P, \Q) =>
         (string-span (P, c, c + l - 1),
          string-span (Q, c, c + l - 1)) before ((\p, \q) =>
         if p = ""
         then A
         else F (c + l,
                A [+] [ string-newline ^ "Seq1: " ^ p ^
                        string-newline ^ "      " ^ ((0, [], p, q).match) ^
                        string-newline ^ "Seq2: " ^ q ^ string-newline ],
                P, Q))) in
  let \align == force {
         process
           (align-path ^ "/minalig3",
            [],
            (S.string-length + T.string-length).num-stringify ^
            string-newline ^ S ^ string-newline ^ T ^ string-newline)
         using syscall-co }
  in force { (
       #Matching-Percentage: x.#Matching-Percentage,
       #Match-Char: x.#Match-Char,
       #Mismatch: x.#Mismatch,
       #Indel: x.#Indel,
       #Alignment: string-implode (pretty (0, [], x.#Seq1, x.#Seq2)))
     | \x <- align };


!
```