Hunting TPR Domains Using Kleisli

Kui Lin[14] Anthony E. Ting[24] Jiren Wang[34] Limsoon Wong[34]

[1]NUS BioInformatics Center, National University of Singapore, Singapore 119260.
[2]Institute of Molecular & Cell Biology, 30 Medical Drive, Singapore 117609.
[3]Kent Ridge Digital Labs, 21 Heng Mui Keng Terrace, Singapore 119613.
[4]linkui@bic.nus.edu.sg, mcbaet@imcb.nus.edu.sg, {jrwang,limsoon}@krdl.org.sg.

Abstract

We have two objectives. First, we want to build a system for detecting tetratricopeptide repeats in protein sequences. Second, we want to demonstrate how the general bioinformatics database integration system called Kleisli can help build such a system easily. We achieve these two objectives by showing that short and clear programs can be written in Kleisli, using its high-level query language CPL, to build a TPR domain hunter by integrating WU-BLAST2.0, HMMER, Entrez, and PFAM.

1 Introduction

"Until recently, biological sequence databases were built by biologists. When sequence databases were first created the amount of data was small and it was important that the database entries were human readable. Database entries were constructed ... using a variety of flat file formats. The result of this effort has been to create a large number of different databases, all in different formats, typically using non-standard data query software, and only really properly accessible to bioinformatics experts" [3]. So it is a big challenge to use these pieces together to answer new questions in biology.

Kleisli [7] is an integration technology that is rather suitable in the bioinformatics arena. Many bioinformatics problems (1) require access to data sources that are highly heterogeneous, geographically distributed, highly complex, constantly evolving, and high in volume; (2) require solutions that involve multiple carefully sequenced steps; and (3) require information to be passed smoothly between the steps. Kleisli is designed to handle the first three requirements directly. In particular, Kleisli provides the high-level query language CPL [15] that can be used to express complicated transformation across multiple data sources in a clear and simple way.

In this article, we demonstrate the use of Kleisli on a particular problem in bioinformatics, the construction of a "domain hunter" for tetratricopeptide repeats [6, 10], TPR domains for short. With an exponential increase in DNA sequences being obtained, the amount of predicted protein sequences that have no specified function has also increased. In order to address this problem, one solution is to look for domains, i.e. conserved polypeptide regions within proteins. The fact that certain linear stretches of amino acids are conserved between different proteins from organisms as diverse as yeast to man suggests an important role for these domains in the function of these proteins. Here, we are searching for proteins that contain TPR domains. These domains consist of multiple repeats of 34 amino acids (hence the term, tetratricopeptide repeat), and are involved in protein-protein interactions [10]. The 34 amino acids that make up this repeat unit are present in numerous proteins with different functions. Based on biochemical and structural data [6], the TPR domain is involved in protein-protein interactions and that the minimal TPR domain consists of three repeats. The TPR domain is involved in interactions with other TPR domain containing proteins as has been suggested to be involved in multi-protein complexes with proteins containing a WD-40 motif [14]. The 34 amino acids that form the TPR motif are highly degenerate, therefore making it difficult to identify by simple protein analysis. Using Kleisli, we can integrate information from a variety of sources to identify potential proteins that contain TPR domains.

Our TPR domain hunter must be able to reliably predict positions of all TPR domains in a given protein sequence. It is also able to iterate this prediction over entire databases of protein sequences. The construction of this TPR domain hunter involves multiple data sources and procedures that are typical of many bioinformatics problems. First in order to extract previously known TPR domains, the ability to access and manipulate GenPept reports from Entrez [12] located at the National Center for Biotechnology Information (NCBI) in Washington DC is needed. Second, in order to recognize new TPR domains, multiple sequence comparison programs such as BLAST [2], WU-BLAST2.0 [1], and HMMER [13] must be accessed. Third, in order to present results properly, some nesting of data is necessary. All of these requirements serve to exercise various relevant features of Kleisli well. While we restrict ourselves to TPR domains here, our method easily generalizes to many other domains; for instance, we have also developed a PDZ domain [11] hunter in exactly the same way. If you have any suggestion for other interesting domains, please do let us know.

The remainder of this article is organized into the following sections. Section 2 is a quick introduction to complex object data model underlying Kleisli and to the "Collection Programming Language" (CPL) of Kleisli. Section 3 describes the construction of our TPR domain hunter. We show the actual CPL programs that (1) collect previously known TPR domains, (2) integrate WU-BLAST2.0 and HMMER to predict positions of TPR domains in a given sequence, (3) iterate the prediction over entire databases. We also discuss the quality of the prediction. Section 4 provides information on the availability of our implementation.

2 A Quick Tour of Kleisli

Kleisli can be regarded as a system for data access, transformation, and integration. It sits between heterogeneous data sources and tools that operate on an integrated version of the available data. In this tour, we focus on the data model of Kleisli that is based on complex object types and on the high-level query language of Kleisli called CPL, short for Collection Programming Language.

2.1 Complex Object Types

The data model underlying Kleisli is a complex object type system that goes beyond the "sets of records" type system of relational databases. It allows arbitrarily nested records, sets, lists, bags, and variants. A bag is also called a multi-set, which is conceptually a set in which duplicates may occur. A list is conceptually a bag with order. A variant is also called a tagged union [9] and represents a type that is "either this or that". Our sets, bags, and lists are homogeneous. In order to mix objects of different types in a set, bag, or list, it is necessary to inject these objects into a variant type. We demonstrate this shortly when we build our TPR domain hunter.

As an example of a complex type, consider the following type that represents the feature table of a GenPept report from Entrez.

```
(#uid: num, #title: string, #accession: string,
 #feature: { (#name:  string, #start: num, #end: num,
```

```
              #anno: [ (#anno_name: string, #descr: string)]) })
```

It has an interesting type because it is a record of set of lists of records. Here is the detail. It is a record of four fields `#uid`, `#title`, `#accession`, and `#feature`. The first three of these store values of types `num`, `string`, and `string` respectively. The `#uid` field uniquely identifies the GenPept report. The `#feature` field is a set of records, which together form the feature table of the corresponding GenPept report. Each of these records has four fields `#name`, `#start`, `#end`, and `#anno`. The first three of these have types `string`, `num`, and `num` respectively. They represent respectively the name, start position, and end position of a paritcular feature in the feature table. The `#anno` field is a list of records. Each of these records has two fields `#anno_name` and `#descr`, both of type `string`. These records together represent all annotations on the corresponding feature.

In general, the types are given by the syntax: $t ::= \texttt{num} \mid \texttt{string} \mid \texttt{bool} \mid \{t\} \mid \{|t|\} \mid [t] \mid (l_1 : t_1, ..., l_n : t_n) \mid <l_1 : t_1, ..., l_n : t_n>$. Here `num`, `string`, and `bool` are the base types. The other types are constructors and build new types from existing types. $\{t\}$, $\{|t|\}$, and $[t]$ respectively construct set, bag, and list types from type t. $(l_1 : t_1, ..., l_n : t_n)$ construct record types from types $t_1, ..., t_n$. $<l_1 : t_1, ..., l_n : t_n>$ construct variant types from types $t_1, ..., t_n$. Values of these types can be explicitly constructed in CPL as follows, assuming the e's are values of appropriate types: $(l_1 : e_1, ..., l_n : e_n)$ for records; $<l : e>$ for variants; $\{e_1, ..., e_n\}$ for sets; $\{|e_1, ..., e_n|\}$ for bags; and $[e_1, ..., e_n]$ for lists. For example, a value conforming to our feature table type is

```
(#uid: 131470, #accession: "131470", #title: "PROTEIN-TYROSINE PHOSPHATASE ...",
 #feature: {(#name: "source", #start: 0, #end: 594,
            #anno: [(#anno_name: "organism", #descr: "Mus musculus"),
                    (#anno_name: "db_xref", #descr: "taxon:10090")]), ...})
```

This is a rich data model. The schemas and structure of all popular bioinformatics databases, flat files, and softwares are easily mapped into it. For a partial list of sources that Kleisli talks to, see `http://adenine.krdl.org.sg:8080/publications/drivers.html`.

2.2 Collection Programming Language

The syntax of CPL is similar to that of the ODMG standard for object-oriented database languages. Rather than giving the complete syntax, which can be found in [15], we illustrate it through a series of examples. Example `eg1` extracts the titles and features of a set `DB` of feature tables. Example `eg2` extracts the titles and features of those elements of `DB` whose titles contain `tyrosine` as a substring. Note the use of `\x` to introduce the variable `x`. The effect of `\x <- DB` is to bind `x` to each element of the set `DB`. In general, an expression $\{f(x) \mid \backslash x \leftarrow S, c(x)\}$ evaluates to the set of $f(x)$ such that x is in the set S and the condition $c(x)$ is true.

```
primitive eg1 == {(#title:x.#title, #feature:x.#feature) | \x <- DB};

primitive eg2 == {(#title:x.#title, #feature:x.#feature)
                 | \x <- DB, x.#title string-islike "%tyrosine%"};
```

These queries are no more than simple project-selection queries and, except for the fact that the source data and the result are not in first normal form, could be expressed in a relational query language. However, CPL can perform more complex restructurings such as nesting and unnesting,as shown in the following examples.

```
primitive eg3 == {(#title:x.#title, #feature:f.#name,
                   #start:f.#start, #end:f.#end, #annotations:f.#anno)
```

```
                | \x <- DB, \f <- x.#feature };

primitive eg4 == {(#title:x.#title, #feature:f.#name, #start:f.#start,
                  #end:f.#end, #anno-name:a.#anno_name, #anno-descr:a.#descr)
                | \x <- DB, \f <- x.#feature, \a <--- f.#anno};

primitive eg5 == {(#entry:x, #organism:a.#descr)
                | \x <- DB, \f <- x.#feature, \a <--- f.#anno, a.#anno_name = "organism"};

primitive eg6 == [(#organism: z, #entries: {v.#entry | \v <- eg5, v.#organism = z})
                | \z <--- [u | \u <- {y.#organism | \y <- eg5}]];
```

Query `eg3` reduces the nested relation `DB`, which is a set of sets of lists, by one level, to a set of lists. It also restructures it by eliminating uids and accessions. Query `eg4` flattens `DB` completely. The `\a <--- f.#anno` has similar meaning to `\x <- DB`, but works on list instead of set. Thus it binds `a` to each item in the list `f.#anno`. Query `eg5` navigates through each annotation `a` of each feature `f` of each entry `x` in `DB` to discover the organism of that entry. It then restructures the nested relation into a database of entries with associated organisms. Query `eg6` demonstrates how to do nesting in CPL. It groups entries in `eg5` by organism names. It also sorts the output list by alphabetical order of organism names, because `[u | \u <- {y.#organism | \y <- eg5}]` converts the set `{y.#organsim | \y <- eg5}` into a duplicate-free sorted list. These examples illustrate part of the expressive power of CPL. More detailed expositions of the theory, syntax, and expressive power of CPL are given in [5, 15] and references therein.

3 A TPR Domain Hunter

In this section we show the construction of a TPR domain hunter by using Kleisli to integrate three existing softwares. The result is shown to be much more effective than any of these used alone.

3.1 Collect TPR Examples

The first step in building a domain prediction tool is to collect examples. This can either be done by hand or by automatic means or by both. If there are many well annotated examples of that domain in a public database, then automatic collection of these examples is possible. Otherwise, the biologist must examine published literature and copy down examples of his domain by hand. In the case of TPR, the protein section of Entrez contains many instances of TPR domain. So automatic collection is possible. These TPR domains are about 34 amino acid long and are annotated either as matured peptides or as regions in feature tables of protein sequences. So we can simply extract them remotely from Entrez and write them into a database `known-tpr`. The script for accomplishing this is:

```
writefile
 {(#uid:u.#uid, #accession:u.#accession, #title:u.#title,
   #start:f.#start, #end:f.#end, #descr:d.#descr,
   #seq:string-span(u.#sequence, f.#start, f.#end))
 | \u <- aa-get-seqfeat-general "TPR",
   not(u.#title string-islike "%incomplete%"), not(u.#title string-islike "%putative%"),
   not(u.#title string-islike "%hypothetical%"), not(u.#title string-islike "%probable%"),
   \f <- u.#feature,
   (f.#end - f.#start) < 40, (f.#end - f.#start) > 25,
   (f.#name string-islike "%mat_peptide%") orelse (f.#name string-islike "%region%"),
   \d <--- f.#anno, d.#descr string-islike "%TPR%",
```

```
   not(d.#descr string-islike "%incomplete%"), not(d.#descr string-islike "%putative%"),
   not(d.#descr string-islike "%hypothetical%"), not(d.#descr string-islike "%probable%")}
to "known-tpr" using stdout;
```

Now let us go over the script. The first step is to ask for all GenPept reports `u` of TPRs in Entrez using `aa-get-seqfeat-general`. (See [15] for the definition of `aa-get-seqfeat-general` and other functions provided by Kleisli.) Then discard those `u` whose titles contain words that imply uncertainty. Then we consider each feature `f` in the feature table of `u`. We select those features that are about 34 amino acid long and is either a matured peptide or a region. Then we check that an annotation `d` of such a feature `f` mentions TPR and does not say it is uncertain. Then a record of the relevent information (unique identifier, title, accession, start position, end position, and actual peptide sequence) of each qualifying TPR domain is constructed and written to the database `known-tpr`.

The check on length imposed on `f` is necessary because annotations in Entrez contain some mistakes, a characteristic typical of large public biology databases based on voluntary contributions. This check relies on the established wisdom that TPR domains are about 34 amino acid long. Without it, some erroneous entries would make it into our `known-tpr` database. The annotations associated with feature tables (rather than sequence titles) are used to determine TPR domains. This avoids the common pitfall[8] of annotations that might be intended for the wrong part of the protein sequence.

3.2 Check a Sequence for TPR Domains

Once some good TPR examples have been collected, we can set up our TPR domain hunter. We combine three different procedures to help spot TPR domains in a given sequence—It is possible to use additional procedures such as pfscan [4] quite easily, but these three suffice for illustration purpose. The first procedure uses WU-BLAST2.0 [1] to check regions in the given sequence for homology against TPR domains in our `known-tpr` database. The second procedure uses HMMER [13] to check regions in the given sequence for homology against a hidden Markov model (HMM) of TPR domains in our `known-tpr` database. The third procedure uses HMMER to check regions in the given sequence for homology against the standard TPR HMM from PFAM [13]. We first show how to prepare the BLAST database and HMM for these procedures. Then we develop each of these three procedures individually. After that, we develop the program for merging them.

3.2.1 Prepare BLAST database and HMM

The BLAST database for WU-BLAST2.0 to operate on can be created using the `localblast-setdb` primitive in CPL. The HMM of our TPR domains be created using the `hmm-localtraining` primitive in CPL. The script for accomplishing the above is given below. As to the TPR HMM from PFAM, it is already available as a file, which we called `PFAM-TPR.HMM` here, so no need to create it.

```
readfile tpr from "known-tpr" using stdin;
writefile tpr to "known-tpr" using localblast-setdb;
writefile tpr to "known-tpr.HMM" using hmm-localtraining;
```

3.2.2 First procedure

In this procedure we want to develop a function `blastp-tpr-doit` that, given a pscore threshold `PSCORE` and an input protein sequence `SEQ`, return all positions in `SEQ` that could be a TPR domain. The way this procedure makes its prediction is by doing a BLASTP using WU-BLAST2.0 on `SEQ` against our BLAST database. Here is how. Make a connection blastp-tpr to our database known-tpr using the localblast-blastp-detail primitive of CPL. Once this is done, each request processed using blastp-tpr causes BLASTP from WU-BLAST2.0 to be run against our BLAST database known-tpr. Having done that, we can define our solution as the function blastp-tpr-doit that simply carries out the following steps. It processes SEQ using blastp-tpr to obtain homologs. Each aligned region r in each homolog h is then examined to find those whose pscore is within the threshold PSCORE. A record is then constructed to store (1) the start and end positions in SEQ that each such region r maps to; (2) the subsequence in SEQ that r maps to; and (3) the evidence for this r, consisting of the negative-log of the pscore returned by WU-BLAST2.0. Note that the evidence is constructed as a list of variants. In this case, the variant is a #our-tpr-blast variant—to distinguish it from evidence returned by the other two procedures to be described later. We make the #evidence field a list of variants to facilitate the merging of multiple evidence from our three procedures later.

```
localblast-blastp-detail (#name: "blastp-tpr", #db: "known-tpr");
primitive blastp-tpr-doit == (\PSCORE, \SEQ) => merge-blast
 [(#evidence:[<#our-tpr-blast:~(log(10,r.#pscore))>],#start:r.#querystart,#end:r.#queryend,
   #seq:list-head (string-tokenize (string-newline, r.#alignment, 0)))
 | \h <- process SEQ using blastp-tpr, \r <--- h.#hits, r.#pscore < PSCORE ];
```

The simplified description above works, but it has one shortcoming: It sometimes reports the same region of SEQ to be a putative TPR domain multiple times, once for each hit to our BLAST database. This multiple reporting is a little ugly. So we decide to keep only the prediction associated with the strongest evidence. In fact, we also decided to merge two predictions if their associated regions are overlapping significantly. This merging is performed by the function merge-blast. We omit the implementation of merge-blast here; it is very similar to the implementation of merge-all that we will describe shortly.

3.2.3 Second procedure

In this procedure we want to develop a function hmm-tpr-doit that, given a score threshold SCORE and an input protein sequence SEQ, return all positions in SEQ that could be a TPR domain. The way this procedure makes its predictions is by comparing SEQ against our HMM of TPR domains using HMMER. This is easy to accomplished, as shown in the program below. We first make a connection hmm-tpr to our TPR HMM known-tpr.HMM. This is done using the hmm-localsearch primitive of CPL. (We use standard values for the other parameters of hmm-localsearch that control the behaviour of HMMER.) Once this is done, each request processed using hmm-tpr causes HMMER to run on known-tpr.HMM. Having done this, hmm-tpr-doit can simply carry out the following steps. It processes SEQ using hmm-tpr. In this case, the hmmfs program of HMMER is run. The result is a report h from HMMER. Each aligned region r in h is examined to find those whose score is better than the threshold SCORE. A record is then constructed to store (1) the start and end positions in SEQ that each such region r maps to; (2) the subsequence in SEQ that r maps to; and (3) the evidence for this r, which is just the corresponding bits score. Note that the evidence is constructed as a list of variants. In this case, the variant is a #our-tpr-hmm variant, to distinguish it from evidence given by the other two procedures. There is no need to do any merging here, since HMMER does not return overlapping predictions.

```
hmm-localsearch (
  #name: "hmm-tpr", #model: "known-tpr.HMM", #threshold: 0, #level: 1,
  #dna: false, #window: 1000, #best: false);
primitive hmm-tpr-doit == (\SCORE, \SEQ) =>
  [(#evidence:[<#our-tpr-hmm:r.#score>], #start:r.#seqstart, #end:r.#seqend,
    #seq:list-head(list-rev(string-tokenize(string-newline,r.#alignment,0))))
  | \h <- process <#hmmfs: SEQ> using hmm-tpr, \r <- h.#hits, r.#score > SCORE];
```

3.2.4 Third procedure

In this procedure we want to develop a function `hmm-pfam-doit` that, given a score threshold `SCORE` and a protein sequence `SEQ`, return all positions in `SEQ` that could be a TPR domain. The way this procedure makes its predictions is by comparing `SEQ` against the standard TPR HMM from PFAM using HMMER. The implementation of this procedure is exactly the same as that of `hmm-tpr-doit`, with two exceptions. The first exception is that we use `PFAM-TPR.HMM`, which is the TPR HMM from PFAM. The second is that variant in the `#evidence` field is now a `#pfam-tpr-hmm` variant, to distinguish it from the other two procedures. The detail is omitted.

3.2.5 Combining all three procedures

Now we need to combine all three procedures to make unified predictions. To do this, we develop a function `merge-all` that, given a list of predictions from all three procedures, merges all significantly overlapping predictions. Let us first define what we mean by overlapping significantly. Two regions `x` and `y` overlap if both of their start points come before the other's end point. They can overlap significantly in two ways: The region they are sharing is at least M amino acid long. Or, the head or tail regions that they do not share is less than N amino acid long. We set $M = 15$ and $N = 15$ here. In general, M and N can be set to some numbers around $30 - 60\%$ of the expected length of the domain to be predicted. The CPL program realising this test is `should-merge-with`, whose obvious implementation is omitted. The function `merge-all` performs the merge as follow. If two predictions overlap significantly according to `should-merge-with`, it removes the second one and adds the evidence from the second one to that of the first. Since evidence from different procedures are different variants, they will not be confused when evidence are concatenated together. `merge-all` iterates this merging process over the whole input list of predictions. The implementation of `merge-all` is omitted in the interest of space. (Note that the implementation of `merge-all` described above is designed for short domains. The implementation for long domains uses a more aggressive criteria for merging predictions.)

Finally, our TPR domain hunter can be realised by the function `tpr-hunt` below. It receives three inputs: the threshold `PSCORE` for our first procedure, the threshold `SCORE` for our second and third procedures, and the protein sequence `SEQ`. It executes the three procedures by invoking `hmm-pfam-doit`, `blastp-tpr-doit`, and `hmm-tpr-doit`. The results are concatenated (`[+]`) into a list, on which `merge-all` is run to combined all the predictions by fusing those that overlap significantly.

```
primitive tpr-hunt == (\PSCORE, \SCORE, \SEQ) => merge-all(
 hmm-pfam-doit(SCORE,SEQ) [+] blastp-tpr-doit(PSCORE,SEQ) [+] hmm-tpr-doit(SCORE,SEQ));
```

As can be seen, it takes very little effort to put together different prediction softwares using an advanced modern database integration system Kleisli/CPL. We now have a program that, given a protein sequence, returns all positions of putative TPR domains in it. We remark on the quality of its prediction later, for there is more programming that we want to do at the moment.

Let us assume that we have NR, the "nonredundant" protein database curated at the NCBI, available as a list `NR`. Then a hunt for TPR domains in sequences in NR can be realised in the short CPL program below. For each entry `s` in `NR`, `tpr-hunt` is run on it with pscore threshold 10^{-5} and bits score threshold 15. Then a record consisting of the title of that entry `s` and its corresponding putative TPR domains `P` is returned.

```
[(#title:s.#title, #tpr-domains:P) | \s <--- NR, \P <--- tpr-hunt(1.0E~5,15,s.#seq)];
```

3.3 Quality

Our automatic procedure collected 184 samples of TPR domains. We now compare them with the 42 seed TPR sequences from PFAM. The first row of the table below depicts the distribution of sequence identities when each of the 184 samples is compared with its closest homolog among the 42 seed sequences. The second row depicts the distribution of sequence identities when each of the 42 seed sequences is compared with its closest homolog among the 184 samples. It is clear from the table that the seed TPR sequences from PFAM has exceedingly poor coverage of the differences of TPR (154 out of 184 samples have less than 60% sequence homology to these seeds). At the same time, 14 of the 42 seeds have less than 60% sequence homology to the 184 samples; thus there is room for improvement in our samples' coverage, which can be accomplished by using more keywords than just `TPR` as we did in Subsection 3.1.

%identity	0-20	20-30	30-40	40-50	50-60	60-70	70-80	80-90	90-100	total
samples vs seeds	0	4	72	66	12	0	3	1	26	184
seeds vs samples	0	0	0	7	7	4	0	0	24	42

The table below depicts the effectiveness of our three procedures in recognizing our samples and PFAM TPR seeds, using 10^{-5} as the threshold `PSCORE` and 15 as the threshold `SCORE`. As a further comparison of these procedures, we also applied them on a subset S from NR such that each member of this subset is BLASTP-related (but without any restriction on the pscore) to some member of our TPR samples or PFAM TPR seed sequences. Note that since we did not put any restriction on the pscore when we were extracting the subset S, we should find both sequences with good and poor homology to TPR sequences. Symbol A denotes those recognized by `our-tpr-blast`; Symbol B those by `our-tpr-hmm`; Symbol C those by `pfam-tpr-hmm`; thus ABC denotes those recognized by all three procedures, $AB\neg C$ those by `our-tpr-blast` and `our-tpr-hmm` but not `pfam-tpr-hmm`, etc.

	ABC	$AB\neg C$	$AC\neg B$	$BC\neg A$	$A\neg B\neg C$	$B\neg A\neg C$	$C\neg A\neg B$	total
seeds	13	0	19	2	0	0	8	42
samples	39	32	9	0	104	0	0	184
S	132	102	28	239	286	119	58	964

The table above shows that about half the predictions are shared by at least two of the procedures. But the remaining half are predicted by only one of the three procedures. Now we must ask the question: Can these predictions be trusted? To answer this question, we look at sequence identities and bit scores. The table below depicts the distribution of sequence identities of TPR domains predicted in S with respect to our TPR samples and PFAM TPR seed sequences. As can be seen, most of the predictions by `our-tpr-blast` have near 100% sequence identities to known TPR, so we consider them "real" TPR. However, as many as 209 of these "real" TPR in S and 104 in our samples are missed by both the HMM procedures at threshold `SCORE` of 15. This reflects that HMM is not as sensitive for short protein sequences and suggests that it is important to use a variety of methods (such as BLAST) to examine for protein domains.

%identity	0-20	20-30	30-40	40-50	50-60	60-70	70-80	80-90	90-100	total
ABC	0	0	0	11	18	7	6	3	87	132
$AB\neg C$	0	0	0	7	5	5	2	2	81	102
$AC\neg B$	0	0	0	4	3	1	3	0	17	28
$BC\neg A$	0	0	10	188	11	5	10	0	15	239
$A\neg B\neg C$	0	0	0	20	26	17	6	8	209	286
$B\neg A\neg C$	0	0	17	87	14	1	0	0	0	119
$C\neg A\neg B$	0	0	1	49	5	0	0	1	2	58

Those predictions made strictly by either HMM procedure show moderate sequence identities of 40-50% to known TPRs. It is not possible to ascertain their correctness without extensive literature consultation. Nevertheless, we think HMM predictions that score above 15 (our `SCORE` threshold) are probably believeable and those that score above 30 are trustworthy, on the basis of the following information (data not shown): The 184 TPR samples from Subsection 3.1 have an average score of 11.40 by `our-tpr-hmm` and an average score of 9.61 by `pfam-tpr-hmm`; the 42 PFAM TPR seeds have an average score of 13.1 by `our-tpr-hmm` and an average score of 30.94 by `pfam-tpr-hmm`; and every one of the predictions made simultaneously by all three procedures has HMM score above 15 (and over half of these are above 30). Also, we have a third HMM using only those samples that score above 15 by `our-tpr-hmm`, it cleanly splits the 184 samples into two sets, one containing 71 samples scoring above 20 and one containing 113 samples scoring 0. In addition, for those 58 TPR in S predicted solely by `pfam-tpr-hmm`, 45 of them appear with other predicted TPR domains in the characteristic "convoy" manner of TPR proteins. Similarly for those 119 TPR in S predicted solely by `our-tpr-hmm`, 90 of them appear with other predicted TPR domains. This strengthens the claim that these HMM predictions scoring above 15 are believeable.

4 Conclusions and Availability

We achieved two things in the previous sections. First, we constructed a useful system for spotting TPR domains by combining the power of WU-BLAST2.0, HMMER, PFAM, and Entrez. Notice that none of these data sources are relational (translation: none of them are convenient for integration using conventional data integration softwares.) This system was shown to be able to find more TPR domains than any of its components used alone. Second, we illustrated the ease and clarity that such a useful integrated system can be constructed in the advanced modern database integration system Kleisli using its high-level query language CPL. In addition, our results also demonstrated that using multiple prediction systems in combination is more effective than using any single prediction system, at least in the context of short domains like TPR.

Incidentally, the entire process of writing our CPL programs, downloading TPR samples from Entrez, BLASTing and running HMMs, and producing the statistics took our "expert" (Wong) just a little over an hour on a low-end Sun Entreprise. This testifies to the ease and efficiency of working with a powerful and flexible query system like Kleisli. However, it did take slightly more effort to place the system on the web.

Our TPR domain hunter is at `http://alanine.krdl.org.sg:8080/examples/tony/tpr`. We also have a PDZ domain hunter at `http://alanine.krdl.org.sg:8080/examples/tony/newpdz`. The Kleisli system will be available from KrisTech Inc. in California under the name KRIS, which stands for both "Kleisli Related Integration System" and "Kent Ridge Integration System". KRIS is *much* more robust and efficient than the old prototype built by Wong at the University of Pennsylvania.

References

[1] S. F. Altschul, W. Gish. Local alignment statistics. *Methods in Enzymology*, 266:460–480, 1996.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[3] P. G. Baker, A. Brass. Recent development in biological sequence databases. *Current Opinion in Biotechnology*, 9:54–58, 1998.

[4] P. Bucher, K. Karplus, K. Moeri, K. Hoffman. A flexible search technique based on generalized profiles. *Computers and Chemistry*, 20:3–24, 1996.

[5] P. Buneman, S. Naqvi, V. Tannen, L. Wong. Principles of programming with complex objects and collection types. *Theoretical Computer Science*, 149(1):3–48, 1995.

[6] A. K. Das, P. T. W. Cohen, D. Barford. The structure of the tetratricopeptide repeats of protein phosphatase 5: Implications for TPR-mediated protein-protein interactions. *The EMBO Journal*, 17(5):1192–1199, 1998.

[7] S. Davidson, C. Overton, V. Tannen, L. Wong. BioKleisli: A digital library for biomedical researchers. *International Journal of Digital Libraries*, 1(1):36–53, 1997.

[8] T. Doerks, A. Bairoch, P. Bork. Protein annotation: Detective work for function prediction. *TIG*, 14(6):248–250, 1998.

[9] R. Hull, C. K. Yap. The Format model: A theory of database organisation. *Journal of the ACM*, 31(3):518–537, 1984.

[10] J. R. Lamb, S. Tugendreich, P. Hieter. Tetratrico peptide repeat interactions: to TPR or not to TPR? *Trends in Biochemical Sciences*, 20(7):257–259, 1995.

[11] J. Saras, C.-H. Heldin. PDZ domains bind carboxy-terminal sequences of target proteins. *Trends in Biochemical Sciences*, 21:455–458, 1996.

[12] G. D. Schuler, J. A. Epstein, H. Ohkawa, J. A. Kans. Entrez: Molecular biology database and retrieval system. *Methods in Enzymology*, 266:141–162, 1996.

[13] E.L. Sonnhammer, S.R. Eddy, R. Durbin. Pfam: A comprehensive database of protein families based on seed alignments. *Proteins*, 28:405–420, 1997.

[14] L. van der Voorn, H. L. Ploegh. The WD-40 repeat. *FEBS Letters*, 307(2):131–134, 1992.

[15] L. Wong. *The Collection Programming Language Reference Manual.* Kent Ridge Digital Labs, 21 Heng Mui Keng Terrace, Singapore 119613, 1998. Available at `http://sdmc.krdl.org.sg/kleisli/psZ/cpl-defn.ps`.