# Using Kleisli to Bring Out Features in BLASTP Results

**Jing Chen** cjing@krdl.org.sg
**Daphna Strauss** daphna@krdl.org.sg
**Limsoon Wong** limsoon@krdl.org.sg

Kent Ridge Digital Labs, 21 Heng Mui Keng Terrace, Singapore 119613

**Abstract**

BLASTP gives a good overall indication of what function a protein might have. However, analysis of BLASTP reports to discover various domain features in the protein is still tedious. We address this problem by using the modern data integration system, Kleisli,[1] to bring out annotated features of BLASTP results. We further strengthen our solution by incorporating additional information from SEG, ClustalW, hmmPfam, etc. It is also noteworthy that the codes of our implementation is sufficiently short to be presented in its entirety.

## 1 Introduction

Computer-assisted analysis of protein sequences is extensively used in sequence database searches. The result of such searches is used for the rapid identification of functions of a protein by analogy to proteins of known functions. One of the most popular tools for this purpose is BLASTP [2] or its several relatives [3, 1, etc.]

The basic version of BLASTP operates as follows. We submit a protein sequence. A big database of sequences is scanned for similar sequences or, abusing the term slightly, "homologs." Once the scan is completed, BLASTP produces a summary and a detailed report of the hits. The summary is a list of homologs that are found and their similarity score. The detailed report is a list of pairwise alignments for each region of each homolog that has significant similarity to our protein. As a result, if the alignments extend strongly over the whole of our protein, we can roughly tell what kind of protein we have. However, if the alignments are in scattered regions, more tedious work is needed to examine these alignments to figure out the overall function of our protein. Moreover, if we wish to have a more detailed idea (such as identifying various domains and active sites) than the overall function of our protein, even if we have some alignments that extend over the whole of our protein, some tedious work is still needed.

What does the "tedious work" involve? At the very least, it means going to Entrez [15] to fetch the GenPept report associated with each interesting homolog, so that we can inspect the feature table in this report to see if the aligned regions fall within any interesting feature or domain annotated in the feature table. Then, we have to copy these regions to a file and perform a ClustalW [17] multiple alignment to check if the positions that aligned with our protein are interesting conserved positions.

[1] Kleisli will soon be made available by KrisTech Inc. in California under the name KRIS. KRIS stands for both "Kleisli Related Integration System" and "Kent Ridge Integration System" to distinguish it from the early prototype developed several years ago at the University of Pennsylvania. This new name KRIS is chosen also because it means the "wavy-bladed" type of daggers original to the Malay archipelago where the industrial-strength version of Kleisli was developed—Singapore.

And so on. Thus the detailed analysis of BLASTP result can be quite time consuming, especially for the inexperienced.

There have been some recent progress to make the output of BLASTP easier for analysis, especially along the visual dimension. For example, the new BLASTP server at the National Center for Biotechnology Information (NCBI) introduced a graphic representation of the hits.[2] Here is roughly what it does. A horizontal line is drawn to represent the user's protein. One horizontal line is drawn for each homolog and placed in a position that corresponds to its alignment to the user's proteins. Each line is colour-coded by its similarity to the user's protein. Also, if you click on a line, the corresponding alignment report pops up. It is certainly a more vivid summary than the pure-text version of BLASTP. An earlier effort is BEAUTY [22], which has more primitive graphics, but incorporated motif hits from Prosite [4]. A more sophisticated effort is that of [11], which added features such as a means to export the pairwise alignment reports for multiple sequence alignment.

Simple visual representation as described gives no information about domains and their positions in our protein. It merely tells us which part of our protein aligns well with which part of a homolog, as it has no idea which part of a homolog contains which domain. Prosite motifs and other domain predictions give slightly more information about the presence of domains and their positions in our protein than simple visualization. However, Prosite motifs and other domain predictions cover only about a thousand different families of domains. There are certainly many more families of domains that can be found in feature tables of proteins. In other words, annotated features in GenPept reports can potentially tell us more about the present of domains and their positions in our protein sequence.

It is not surprising that earlier approaches have ignored annotated features of homologs, in spite of their being much more informative than simple visual representation of alignments and mere incorporation of Prosite motifs and other more sophisticated forms of domain prediction. For there are several serious challenges to bringing out annotated features in BLASTP reports. First, we must have the ability to analyse BLASTP reports to figure out which feature tables are needed. Second, we must have the ability to extract these rather complicated feature tables from GenPept reports either locally or remotely from Entrez. Third, we must have the ability to integrate information extracted from feature tables and BLASTP reports and possibly other tools or sources. These obstacles call for an advanced database integration tool.

Kleisli [9] is an advanced integration technology that attempts to scale this bioinformatics "tower of babel." Many bioinformatics problems (1) require access to data sources that are high in volume, highly heterogeneous and complex, constantly evolving, and geographically dispersed; (2) require solutions that involve multiple carefully sequenced steps; (3) require information to be passed smoothly between the steps; (4) require increasing amount of computation; and (5) require increasing amount of visualization. Kleisli is designed to handle the first three requirements directly. In particular, Kleisli provides the high-level query language CPL [7] that can be used to express complicated transformation across multiple data sources in a clear and simple way. In addition, while Kleisli does not handle the last two requirements directly, it is capable of distributing computation to appropriate servers and initiating visualization programs.

In this paper, we demonstrate how to bring out annotated features from BLASTP reports and how to integrate additional sources into BLASTP reports. We show that obstacles mentioned earlier can be easily overcome by using a modern database integration system like Kleisli and a high-level query language like CPL. In fact, our implementation using Kleisli/CPL is sufficently short that we even have room to show our entire program in this paper.

We organize our presentation as follows. Section 2 gives a preliminary CPL program that brings out

[2] See `http://www.ncbi.nlm.nih.gov/cgi-bin/BLAST/nph-newblast?Jform=0`.

annotated feature from BLASTP reports. Section 3 gives a refined solution that produces less verbose output. Section 4 describes an on-line demonstration of our implementation that we have put on the World-Wide Web. Section 5 gives some closing remarks.

## 2 Preliminary Solution

We demonstrate step-by-step in this section, how simple and easy it is to add feature information to BLASTP by writing CPL programs and running them in Kleisli.[3] We assume that `SEQ` is the protein sequence from the user.

First we need some functions for calculating the minimum, maximum, and absolute difference of two numbers. Their definitions are standard. The implementation in CPL is given below. The line numbers are for ease of reference later.

```
 1. primitive min == (\x, \y) => if x < y then x else y;
 2. primitive max == (\x, \y) => if x < y then y else x;
 3. primitive diff == (\x, \y) => if x < y then y - x else x - y;
```

Next, we make a connection `nr-blast` to perform BLASTP on the non-redundant protein database "nr" at the NCBI, as shown below. Subsequently, to compare the protein sequence `SEQ` against "nr", all we need is to execute the CPL expression `process SEQ using nr-blast`.

```
 4. webblast-blastp-detail (#name: "nr-blast", #db: "nr", #level: 1);
```

Then we perform BLASTP on the protein sequence `SEQ`. Then for each homolog, we obtain its feature table and sequence from Entrez. These two pieces of information are integrated with the alignment from BLASTP to give us features in homologs that align well with `SEQ`. The CPL program to carry out this process is given below.

```
 5. primitive blast-with-feature ==
 6.  [ (#start: qs, #end: qe, #hits: [ (
 7.     #uid: x.#uid, #accn: x.#accession, #title: x.#title,
 8.     #identities: i.#Matching-Percentage, #query-start: qs, #query-end: qe,
 9.     #feat-start: f.#start, #feat-end: f.#end, #feat-anno: f.#anno)])
10.  | \x <- process SEQ using nr-blast,
11.    \t <- aa-get-feature-by-uid (x.#uid), \S <- aa-get-seq-by-uid (x.#uid),
12.    \h <--- x.#hits, h.#pscore <= 1.0E~8,
13.    \f <- t.#feature, not (f.#anno = [ ]), not (f.#name = "source"),
14.    (f.#end - f.#start) > 4,
15.    f.#start > (h.#subjectstart - 20), f.#end < (h.#subjectend + 20),
16.    \u == min (f.#end, h.#subjectend) - max (f.#start, h.#subjectstart),
17.    \v  == min (f.#end - f.#start, h.#subjectend - h.#subjectstart),
18.    (((u + 1) / (v + 1)) * 100) > 80,
```

[3] Due to space limitation, we focus on explaining the logic of our programs, rather than their syntax. The interested Reader can consult [19] for a complete definition of the syntax, as well as the definitions of all the functions available in CPL. We recommend the paper [7] as background for the comprehension syntax {$e$ | \x <- $e$, $C$} used liberally in CPL. We further recommend the papers [8] and [18] for a basic introduction to the theory underlying CPL. The paper on TPR domain hunter [13] in this volume also has more information on Kleisli.

```
19.     { () | \a <--- f.#anno, a.#descr string-islike "%conflict%"} = { },
20.     \qs == (f.#start - h.#subjectstart) + h.#querystart,
21.     \qe == (f.#end - h.#subjectend) + h.#queryend,
22.     \s1 == string-span (S.#sequence, f.#start, f.#end),
23.     \s2 == string-span (SEQ, qs, qe),
24.     \i <- minalig-doit (s1, s2)];
25. materialize "blast-with-feature";
```

As this CPL program is fairly long, we step through it in detail. First, BLASTP is performed on `SEQ` against "nr" at NCBI (line 10). For each homolog `x`, we use the CPL function `aa-get-feature-by-uid` and `aa-get-seq-by-uid` to obtain its feature table `t` and its protein sequence `S` from NCBI (line 11). Then we examine each region `h` in `x` that BLASTP has aligned with our protein `SEQ`, discarding the region if its pscore is weak (line 12); here we use $1^{-8}$ as our threshold.

Next we have to figure out which feature of `x` does this region `h` fall into. So we examine every feature `f` in the feature table `t`, discarding the feature if it has no annotation or if it is not interesting (line 13); here we take the "source" feature to be uninteresting since it merely tells us which organism the protein is from. We discard the feature if it is too short (line 14). We also discard the feature if either end of it sticks too far out of the aligned region `h` (line 15); the threshold we use is 20 amino acids.

Next we make sure that the feature overlaps at least 80% of the aligned region `h` (lines 16-18). Finally, we make sure that the feature does not contain any annotation that indicates error (line 19). The equation used for this check, `{() | \a <--- f.#anno, a.#descr string-islike "%conflict%"} = {}`, deserves a more detailed explanation. The left-hand-side of this equation evaluates to the singleton set {()} if and only if there exists some `a` in the annotations on `f` and the description of this annotation `a` contains the word "conflict." Thus the equation is true if and only if no annotation on `f` contains the word "conflict."

At this point, we have basically identified that `f` is a feature in `x` that falls in region `h` of `x` and is relevant. Now we need to map it onto our original protein sequence `SEQ` and compute its percentage sequence identity to that part of `SEQ`. To explain how we do the mapping, we need some technical detail. The description returned by BLASTP on `h` has several fields. The `#subjectstart` and `#subjectend` fields of `h` tell us the start and end positions of `h` in the sequence `S` of `x`. Similarly, the feature `f` has several fields. The `#start` and `#end` fields of `f` tell us the start and end positions of `f` in the sequence `S` of `x`. On the other hand, the `#querystart` and `#queryend` fields of `h` tell us the start and end of the region of `SEQ` that BLASTP has aligned with `h`. Thus, to derive the place in `SEQ` that `f` should align with, we have to compute the offsets of the start and end positions of `f` with respect to the start and end positions of `h` and then add these offsets to the start and end positions of the region of `SEQ` that BLASTP has aligned with `h`. These calculations are done in lines 20-21[4] and `qs` and `qe` are set to the resulting start and end position respectively. The sequence corresponding to the feature `f` and the region it maps to on `SEQ` are then extracted and aligned using the CPL function `minalig-doit` (lines 22-24) and `i` is set to the resulting alignment.

Then a record of the relevant information is constructed and included in the set `blast-with-feature` (lines 5-19). This set is "materialized" and stored for subsequent use (line 25). It is of interest to describe the members of the set `blast-with-feature` further. Each member has three fields `#start`, `#end`, and `#hits`. Each member represent a feature of a BLASTP homolog in "nr" of `SEQ`. The `#start` and `#end` fields indicate the place in `SEQ` that this feature maps to. The `#hits` field is at this point a singleton list storing information of this feature. The information is the unique identifier

[4] The formula here is good for ungapped alignment, which is easy to understand. For gapped alignment, a more complicated formula is used.

of the homolog in “nr” (the `#uid` field), the accession number of this homolog (the `#accn` field), the title of this homolog (the `#title` field), the percentage sequence identities that this feature has with respect to the region in `SEQ` that it has been mapped to (the `#identities` field), the start and end positions in `SEQ` that it maps to (the `#query-start` and `#query-end` fields), its start and end position in the homolog (the `#feat-start` and `#feat-end` fields), and the annotations on this feature (the `#feat-anno` field).

The preceding CPL program is sufficient for figuring out exactly what feature of which homologs are similar to which part of our protein `SEQ`. Let us take a look at an example output from it before we discuss some improvements we make in the next section:

```
[...
 (#start: 279, #end: 365,
  #hits: [(#uid: 2497506, #accn: "sp|Q92796|SP02_HUMAN",
           #title: "PRESYNAPTIC PROTEIN SAP102 ...",
           #identities: 35.632184, #query-start: 279, #query-end: 365,
           #feat-start: 378, #feat-end: 464,
           #feat-anno: [(#anno_name: "note", #descr: "DHR 3."),
                        (#anno_name: "region_name", #descr: "Domain")])]),
 (#start: 286, #end: 366,
  #hits: [(#uid: 400891, #accn: "sp|P31016|SP90_RAT",
           #title: "PRESYNAPTIC DENSITY PROTEIN 95 ...",
           #identities: 34.567901, #query-start: 286, #query-end: 366,
           #feat-start: 312, #feat-end: 392,
           #feat-anno: [(#anno_name: "note", #descr: "DHR 3."),
                        (#anno_name: "region_name", #descr: "Domain")])]),
...]
```

From the example output above, we see that the region 279-366 of `SEQ` has been identified as a DHR domain [14], similar to those of some presynaptic proteins. In contrast, the standard BLASTP output is merely able to tell us that `SEQ` has some homology to some presynaptic proteins. This useful information on domain is precisely the extra information that our 25-line CPL program provides.

## 3 Refined Solution

However, the output from the CPL program of the previous section is a little annoying. It tells us at least twice that the region 279-366 maps to a DHR domain. This is too verbose and is probably unnecessary. Let us now show how we can improve the output by grouping together such overlapping predictions. As a further refinement, we also sort each group by percentage sequence identity.

Before we proceed, we need some criteria to decide when to group together two domains or regions. We use a simple one here: the two regions must overlap more than 50% and the differences of their end points must be less than 30%. These conditions are captured by the CPL program `domains-should-be-merged` given below, where `d1` and `d2` are the two input regions.

```
primitive domains-should-be-merged == (\d1, \d2) =>
let (\s1, \e1, \s2, \e2) == (d1.#start, d1.#end, d2.#start, d2.#end) in
let \ave == ((e1 - s1) + (e2 - s2)) / 2
in (s1 < e2) andalso (s2 < e1)
```

```
30. andalso ((s1 diff s2) < (0.3 * ave)) andalso ((e1 diff e2) < (0.3 * ave))
31. andalso ((min (e1, e2) - max (s1, s2)) > (0.5 * ave));
```

It would be nice if we output the regions in a 5'-to-3' or left-to-right order. So we need a function `sort-domains` to sort regions according to their positions in `SEQ`. This function is generated by the higher-order function `list-gensort` provided in CPL, as shown below. Basically, given a sort predicate $P$, `list-gensort` $P$ $[o_1, \ldots, o_n]$ evaluates to $[o'_1, \ldots, o'_n]$, where $[o'_1, \ldots, o'_n]$ is a permutation of $[o_1, \ldots, o_n]$ such that $o'_1\ P\ \cdots\ P\ o'_n$; in other words, $[o'_1, \ldots, o'_n]$ is $[o_1, \ldots, o_n]$ sorted according to $P$.

```
32. primitive sort-domains == list-gensort ((\x, \y) =>
33.  (x.#start > y.#start) orelse ((x.#start = y.#start) andalso (x.#end > y.#end)));
```

Now everything is in place for us to improve our feature-enhanced BLASTP using the 14-line CPL program below. First, we sort the regions given in `blast-with-feature` in the previous section to facilitate grouping (line 35). Next, we group together those regions that overlap significantly as defined by the criteria `domains-should-be-merged` (lines 36-42). Third, within each group, we sort the regions in descending order of sequence identity (line 45), and we adopt the start and end positions of the region having the strongest sequence identity to be the start and end positions of the merged region (lines 43, 46). Finally, we sort the list according to the new start and new positions (line 47).

```
34. primitive blast-with-domains ==
35. let \sort == sort-domains (blast-with-feature) in
36. let \mix  == list-irs
37.   @ ((\x, \y) => if domains-should-be-merged (x, y.list-head)
38.         then (#start: min (x.#start, y.list-head.#start),
39.               #end: max (x.#end, y.list-head.#end),
40.               #hits: x.#hits [+] y.list-head.#hits) +] y.list-tail
41.         else x +] y, [ sort.list-head])
42.   @ (sort.list-tail)  in
43. let \nice == [ (#start: h.#query-start, #end: h.#query-end, #hits: S)
44.   | \m <--- mix,
45.     \S == list-gensort @ ((\u,\v) => u.#identities < v.#identities) @ (m.#hits),
46.     \h == S.list-head ]
47. in (_ => sort-domains (nice)) handle (_ => [ ]);
```

There is one technical item in the CPL program above that we should point out. In the process of merging the regions, we assume that the sorted list `sort` of regions is nonempty. If `sort` is empty, then line 41 and line 42, which attempt to split `sort` into its head and tail respectively, are going to fail with a run-time exception. This possibility of run-time failure is "handled" in line 47 using the CPL function `handle`. The CPL function `handle` works this way: ($f$ `handle` $g$) is evaluated by first evaluating $f()$; if it evaluates successfully to an object $o$, then $o$ is returned; if a run-time exception $e$ is raised, then $g(e)$ is evaluated and its output is returned. Thus, if an exception is raised because `sort` is empty, line 47 returns the empty list `[ ]`, which is exactly what is desired.

For completeness, the output of `blast-with-domains` is shown below. We see that the various overlapping regions for each feature are now merged into one group as desired.

```
[...
 (#start: 286, #end: 366,
```

```
#hits: [(#uid: 2497508, #accn: "sp|Q62936|SP02_RAT",
          #title: "PRESYNAPTIC PROTEIN SAP102 ...",
          #identities: 38.271605, #query-start: 286, #query-end: 366,
          #feat-start: 403, #feat-end: 483,
          #feat-anno: [(#anno_name: "note", #descr: "DHR 3."),
                       (#anno_name: "region_name", #descr: "Domain")]),
        ...
         (#uid: 2497506, #accn: "sp|Q92796|SP02_HUMAN",
          #title: "PRESYNAPTIC PROTEIN SAP102 ...",
          #identities: 35.632184, #query-start: 279, #query-end: 365,
          #feat-start: 378, #feat-end: 464,
          #feat-anno: [(#anno_name: "note", #descr: "DHR 3."),
                       (#anno_name: "region_name", #descr: "Domain")]),
        ...] ...]
```

## 4 Demonstration

It is really easy to write Kleisli/CPL programs. However, the ordinary users may prefer a more form-based interface. So it is normal to put up a web-based form interface to Kleisli/CPL programs. The procedure is generally painless and straightforward. The web interface to our "feature BLAST" program is available at `//uracil.krdl.org.sg:8080/examples/feature-blast`.

We provide a brief description here. At the above URL is the form given in Fig. 1. The user pastes his protein sequence (in plain text, without any header information) into the large input box in the middle of the form and sets various thresholds. These parameters are passed to the CPL programs described earlier for execution.

Once processing is completed, the user is provided with three views of the output. The first view is shown in Fig. 2. It offers no more information than the usual BLASTP output, but organised in a manner that we feel might be more pleasant than BLASTP.

The second view is shown in Fig. 3. It tabulates homologs reported by BLASTP, together with their specific features that align well with the user's protein sequence. It thus provides the user an overall sense of what type of domains in these homologs are similar to his sequence.

The third and final view is shown in Fig. 4. It corresponds precisely to the output of the CPL programs discussed in the preceeding section. Its rows correspond to segments of the user's protein sequence. Each row corresponds to a possible domain based on annotated features extracted from BLASTP output. It thus provides the user a more precise sense of the domains and their positions in his protein sequence.

## 5 Remarks

As can be seen from the last three sections, the basic objective of bringing out annotated features in BLASTP results can be achieved without much sweat using CPL. The explicit use of feature table information is probably a unique aspect. It helps avoid the common pitfall of using sequence title as annotations, which might be intended for a different region of the sequence [10]. Many bells and whistles can be added: the incorporation of ClustalW [17], hmmPfam [16, 6], SEG [21], etc. Due

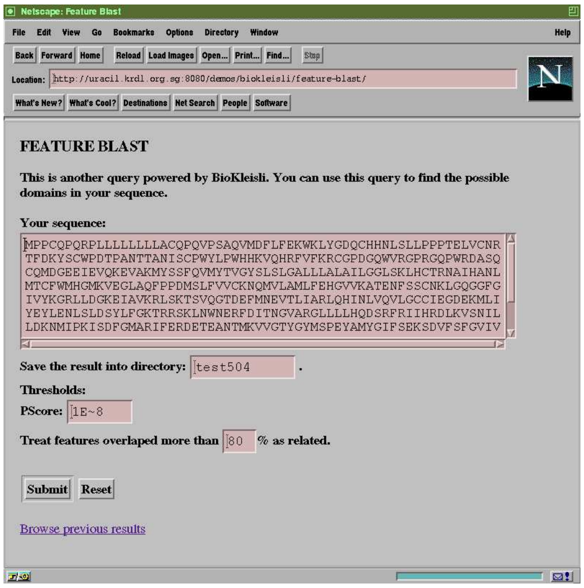


Figure 1: The Feature BLAST on-line demonstration web page.

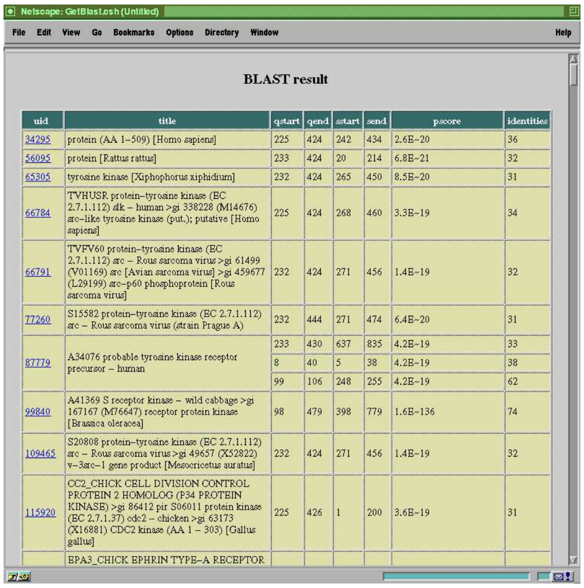


Figure 2: This view offers information similar to BLASTP.

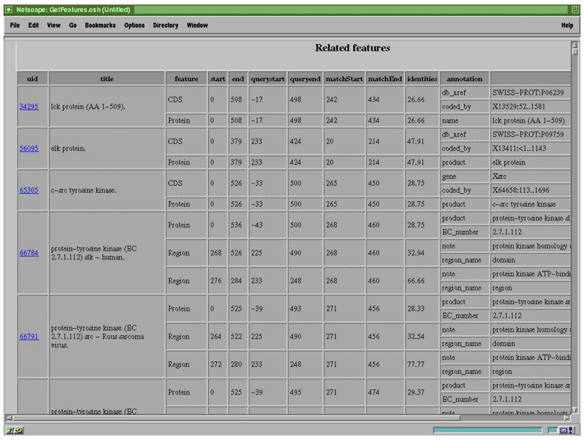


Figure 3: This view provides an overall sense of relevant features in homologs from BLASTP.

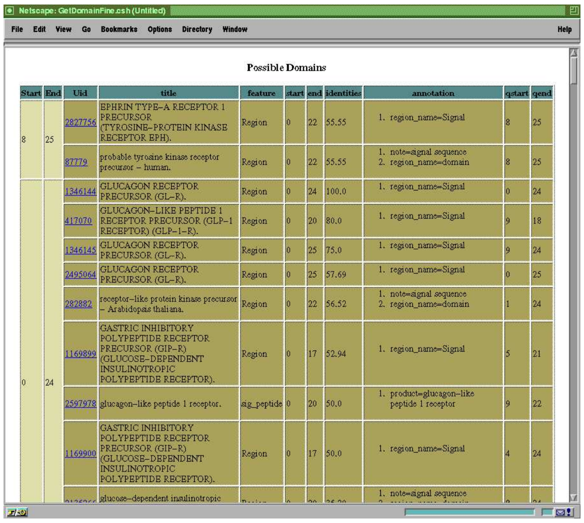


Figure 4: This view provides more precise information of domains and their positions in the input protein sequence, based on annotated features extracted from BLASTP output.

to page-lenght limit, we skip the detail, which can be found in the file `http://sdmc.krdl.org.sg/ kleisli/ psZ/ cdw-featureblast.ps`. They suggest that a reasonably good high-throughput protein sequence annotation system can be readily put together using the Kleisli system.

The accomplishments above are merely the visible part of what we have done. There is also an "invisible" part that we would like to briefly mention here. The Kleisli system does a significant amount of optimization on CPL programs behind the scene [20]. We close our discussion with a relevant one here: parallelism. Let us use lines 10-11 for illustration. These three lines say, for each BLASTP homolog `x` of `SEQ`, retrieve remotely from Entrez in Washington DC its feature table `t` and protein sequence `S`. The actual execution of these lines does not proceed in a linear fashion. The requests for each `t` and `S` of each `x` are scheduled and dispatched several at a time. This exploitation of parallelism masks the latency of remote access and improves efficiency several folds, up to the effectively sustainable level of parallelism.

We end this paper by making a brief comparison of Kleisli to some previous systems. It is fair to say that SRS [12] is one of the more successful tools for integrating biology databases. It is essentially a link-based interface. It is very convenient to use for simple operations. However, it offers no facility for flexible transformation of data from these sources, an important aspect in more advanced bioinformatics applications. As a consequence, one needs a significant amount of manual work if one relies mostly on SRS for data integration. TAMBIS [5] is slightly more comparable to Kleisli. It is an interface to many sources that come with a rather fixed set of queries that offer more slightly interesting integration of the sources than SRS. The kind of queries that can be expressed basically have to fall into templates anticipated by TAMBIS' designers. Thus, the expressible queries are much more limited than in Kleisli. In compensation to the limitation on its query expressive power, its queries can be formulated using a looser syntax and is supported by a knowledge-driven user interface. This could be an advantage for less skillful programmers. However, it is worth pointing out that TAMBIS actually runs on top of an old version of Kleisli, demonstrating the value of Kleisli as a substrate for constructing other integration systems.

# References


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

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

[3] Altschul, S.F., Madden, T.L., Schaffer, A.A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D.J., Gapped BLAST and PSI-BLAST: A new generation of protein database search programs, *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[4] Bairoch, A., Bucher, P., and Hofmann, K., The PROSITE database: its status in 1997, *Nucleic Acids Research*, 25(1):217–221, 1997.

[5] Baker, P.G., Brass, A., Bechhofer, S., Goble, C., Paton, N., Stevens, R., TAMBIS—transparent access to multiple bioinformatics information sources, In *Proceedings of 6th International Conference on Intelligent Systems for Molecular Biology*, 25–34, 1998.

[6] Baldi, P., Chauvin, Y., Hunkapiller, T., and McClure, M.A., Hidden Markov models of biological primary sequence information, *Proceedings of National Academy of Science*, 91(3):1059–1063, 1994.

[7] Buneman, P., Libkin, L., Suciu, D., Tannen, V., and Wong, L., Comprehension syntax, *SIGMOD Record*, 23(1):87–96, 1994.

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

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

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

[11] Durand, P., Canard, L., and Mornon, J.P., Visual BLAST and Visual FASTA: Graphic workbenches for interactive analysis of full BLAST and FASTA outputs under Microsoft Windows 95/NT, *CABIOS*, 13(4):407–413, 1997.

[12] Etzold, T. and Argos, P., SRS: Information retrieval system for molecular biology data banks, *Methods Enzymol.*, 266:114–128, 1996.

[13] Lin, K., Ting, A., Wang, J., and Wong, L., Hunting TPR Domains Using Kleisli, This volume.

[14] Ponting, C.P., DHR domains in syntrophins, neuronal NO synthases and other intracellular proteins, *Trends in Biochemical Sciences*, 20:102–103, 1995.

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

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

[17] Thompson, J.D., Higgins, D.G., and Gibson, T.J., CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties, and weight matrix choice, *Nucleic Acids Research*, 22:4673–4680, 1994.

[18] Wadler, P., Comprehending monads, *Mathematical Structures in Computer Science*, 2:461–493, 1992.

[19] Wong, L., *The CPL 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`.

[20] Wong, L., *The Kleisli/CPL Extensible Query Optimizer Programmer Guide*, Kent Ridge Digital Labs, 21 Heng Mui Keng Terrace, Singapore 119613, 1995. Available at `http://sdmc.krdl.org.sg/ kleisli/ psZ/ cplopt.ps`.

[21] Wootton, J. and Federhen, S., Statistics of local complexity in amino acid sequences and sequence databases, *Computers and Chemistry*, 17:149–163, 1993.

[22] Worley, K.C., Wiese, B.A., and Smith, R.F., BEAUTY: An enhanced BLAST-based search tool that integrates multiple biological information resources into sequence similarity search results, *Genome Research*, 5:173–184, 1995.